



# Samsung Innovation Campus

| Internet kết nối vạn vật – IoT

Together for Tomorrow!  
**Enabling People**  
Education for Future Generations

Chương 7.

# Node.js trong IoT

**Internet kết nối vạn vật – IoT**

# Mô tả chương

---



## Mục tiêu

- ✓ Tìm hiểu các khái niệm cơ bản về Node.js, cú pháp lập trình và các kỹ thuật lập trình nâng cao.
- ✓ Học cách điều khiển các cảm biến của Raspberry Pi dựa trên Node.js.



## Nội dung

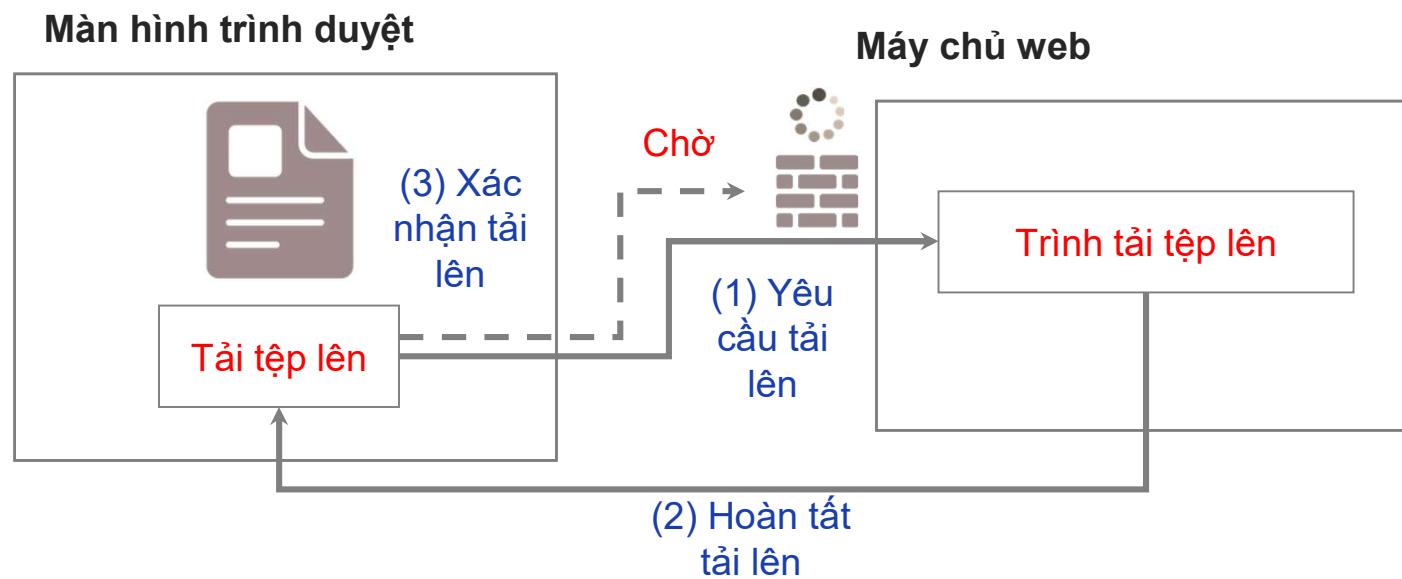
- ✓ Bài 1. Tổng quan về Node.js
- ✓ Bài 2. Lập trình Node.js cơ bản
- ✓ Bài 3. Raspberry Pi với Node-RED
- ✓ Bài 4. Xử lý lỗi và khắc phục sự cố

Bài 1.

# Tổng quan về Node.js

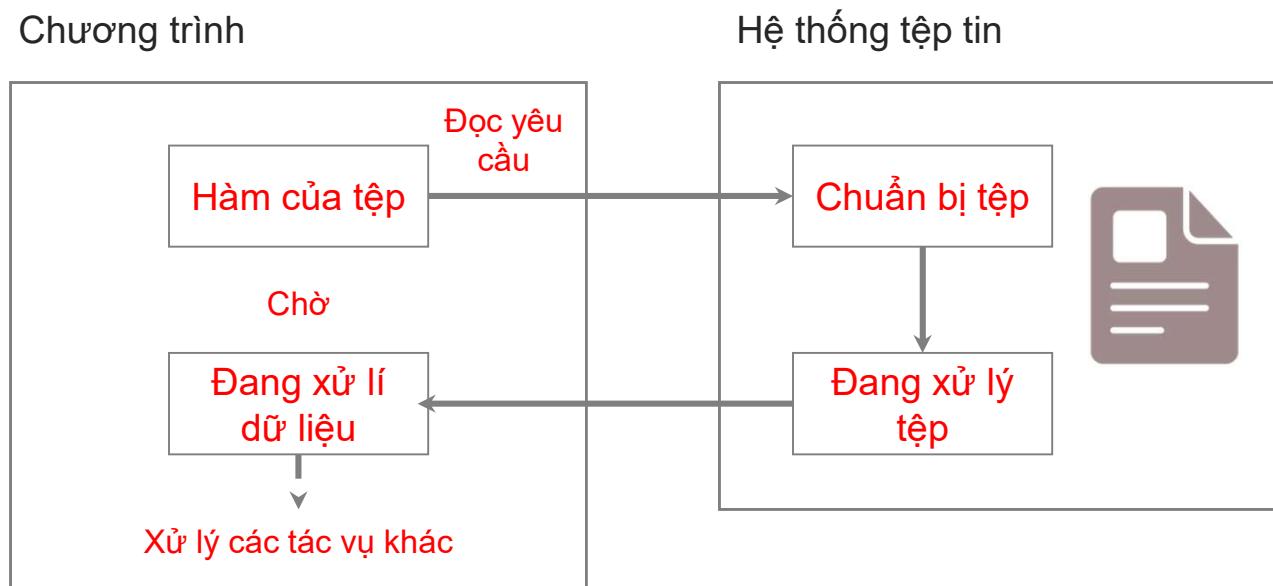
- 1.1. Tổng quan về Node.js
- 1.2. Cài đặt các công cụ phát triển
- 1.3. Tạo dự án **Node** đầu tiên
- 1.4. Gửi **Log** lên **Console**
- 1.5. Sử dụng Mô-đun trong Node
- 1.6. Sử dụng **Mô-đun** tích hợp sẵn

- Node.js còn được gọi là Node
- Hiện đã có nhiều ngôn ngữ và công cụ giúp xây dựng các hàm của máy chủ ví dụ như các dịch vụ Web. Tại sao Node được tạo ra?
  - Bắt đầu xây dựng các công cụ máy chủ mới để giải quyết các vấn đề xảy ra trong quá trình tải tệp lên.



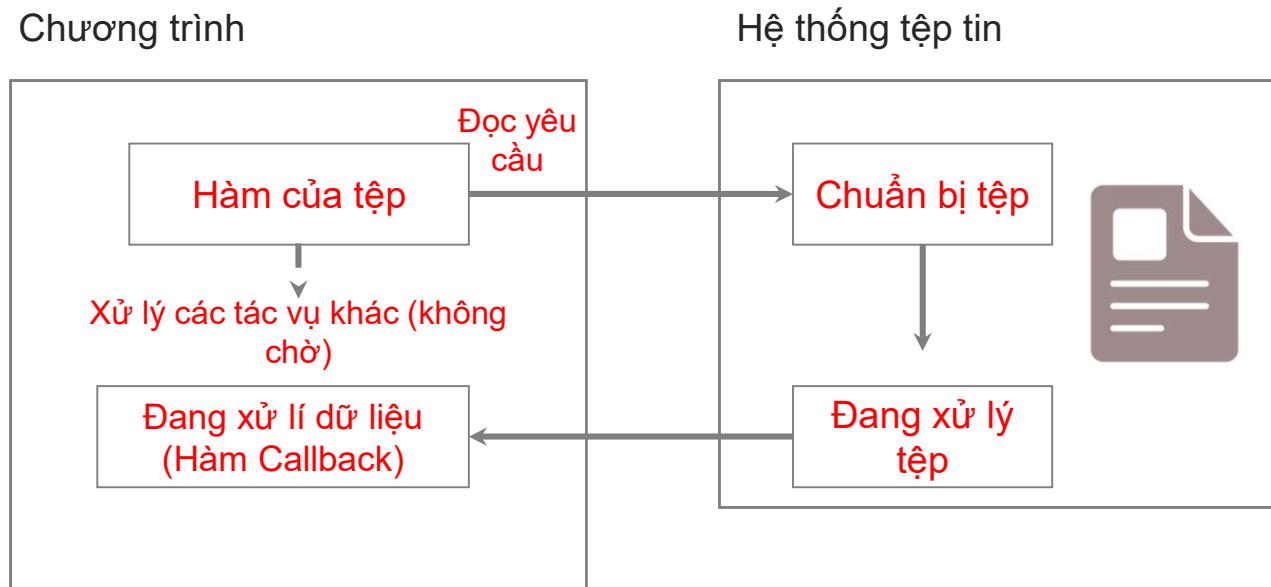
## Các phương thức xử lý I/O đồng bộ trong Node

- Các phương thức xử lý I/O đồng bộ trong Node
  - Xử lý các tác vụ khác sau khi đọc nội dung của tệp
  - Dưới đây là ví dụ về I/O đồng bộ:



## Các phương thức I/O không đồng bộ trong Node

- Các phương thức I/O đồng bộ trong Node
  - Xử lý đồng thời các yêu cầu khác mà không cần đợi quá trình yêu cầu đơn lẻ kết thúc
  - → I/O không chặn
  - Dưới đây là một ví dụ về I/O không đồng bộ có sử dụng hàm Callback



## Đồng bộ và Không đồng bộ, so sánh mẫu code

- Phương thức đồng bộ đọc nội dung của tệp và trả về dưới dạng giá trị kết quả của hàm. Chờ đến khi kết quả được trả về.
- Phương thức không đồng bộ đọc nội dung tệp và trả về dưới dạng hàm Callback. Lệnh thực thi tiếp theo sẽ được tiến hành ngay sau khi tệp bắt đầu được đọc

Phương thức chẵn

```
var contents = file.read('a.txt');
Chờ
doShow(contents);
var result = doAdd(10, 10);
```



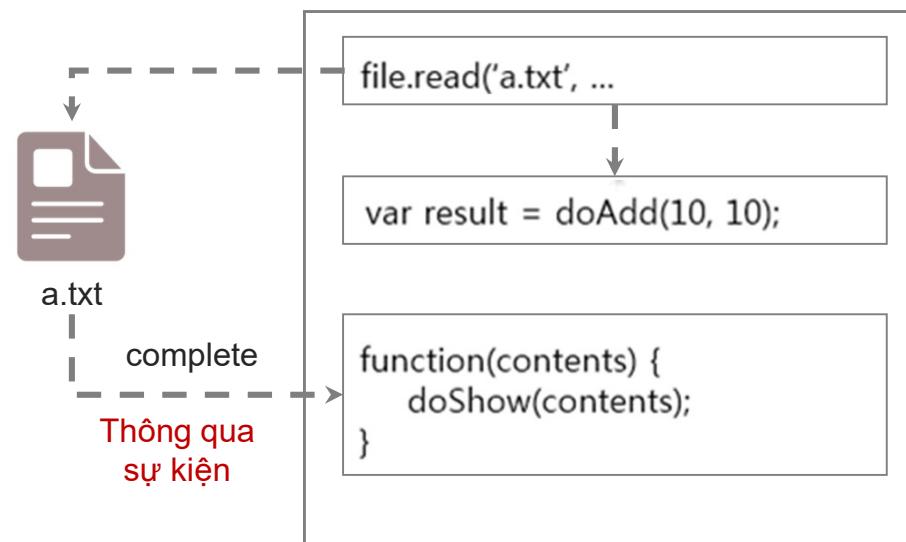
Phương thức không chẵn

```
file.read('a.txt', function(contents) {
  doShow(contents);
});
var result = doAdd(10, 10);
```

## Phương thức I/O dựa trên sự kiện

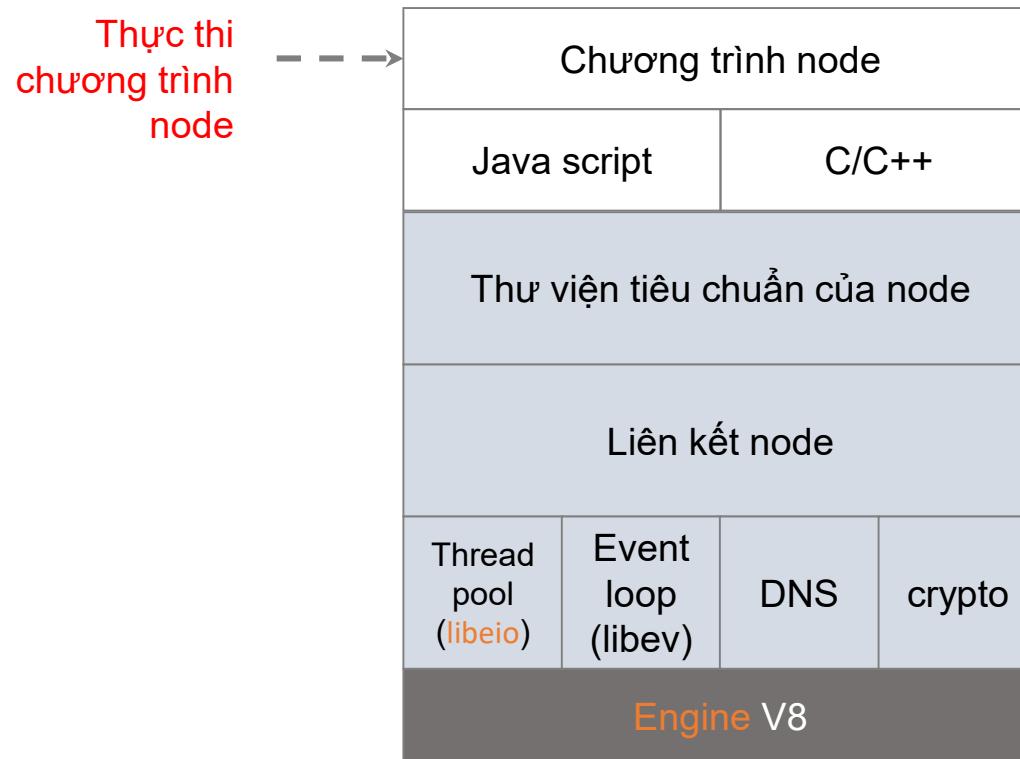
- Phương thức không đồng bộ còn được gọi là mô hình I/O dựa trên sự kiện
- Hàm Callback được thực thi khi hệ thống nhận được sự kiện

Phương thức không chặn



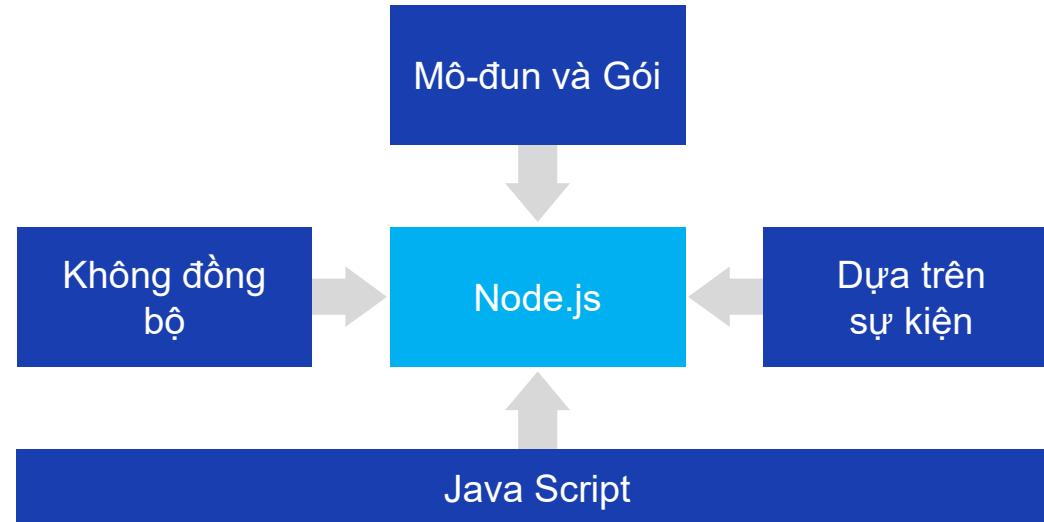
## Kiến trúc Node

- Chạy trên **Engine V8** của trình duyệt Google Chrome và sử dụng ngôn ngữ Javascript



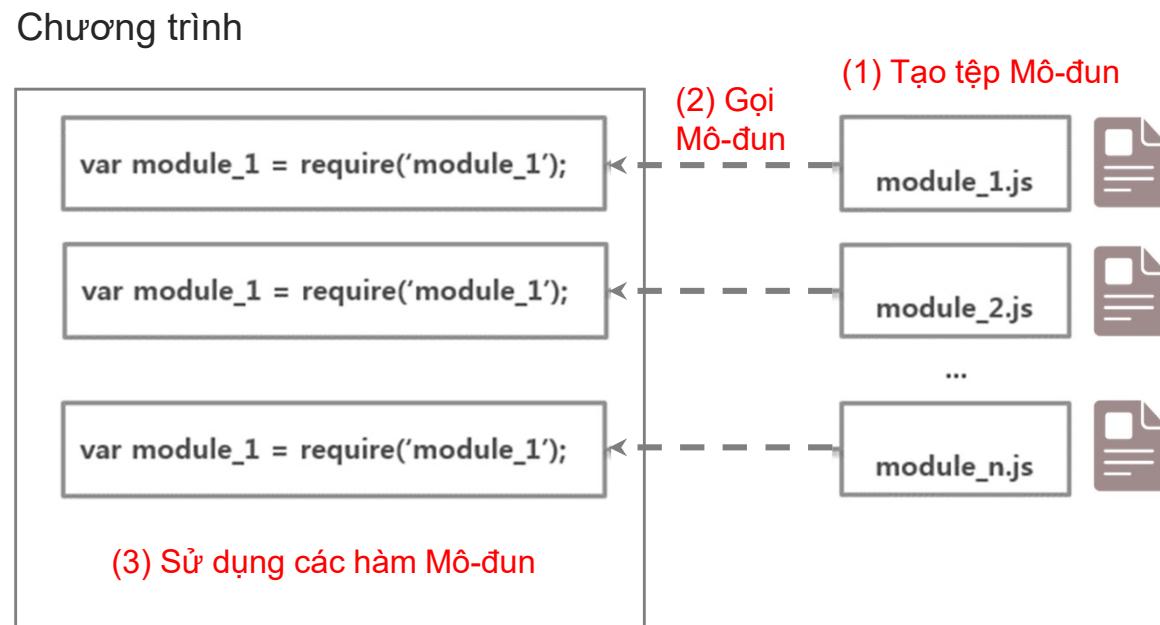
## Các tính năng chính của Node

- JavaScript là một ngôn ngữ lập trình đã được nhiều nhà phát triển web sử dụng.
- Sử dụng I/O không đồng bộ /I/O dựa trên sự kiện
- Cấu hình các chương trình máy chủ bằng các mô-đun và gói
- Các nhà phát triển đã tự phát triển rất nhiều mô-đun



## Các mô-đun giúp các Node dễ sử dụng hơn

- Tách **code** JavaScript thành một Mô-đun, một tệp riêng biệt có phần mở rộng là js.
- **Biên dịch với các tham số tiêu chuẩn của CommonJs**



Bài 1.

# Tổng quan về Node.js

- 1.1. Tổng quan về Node.js
- 1.2. Cài đặt các công cụ phát triển
- 1.3. Tạo dự án node đầu tiên
- 1.4. Gửi log lên console
- 1.5. Sử dụng Mô-đun trong Node
- 1.6. Sử dụng mô-đun tích hợp sẵn

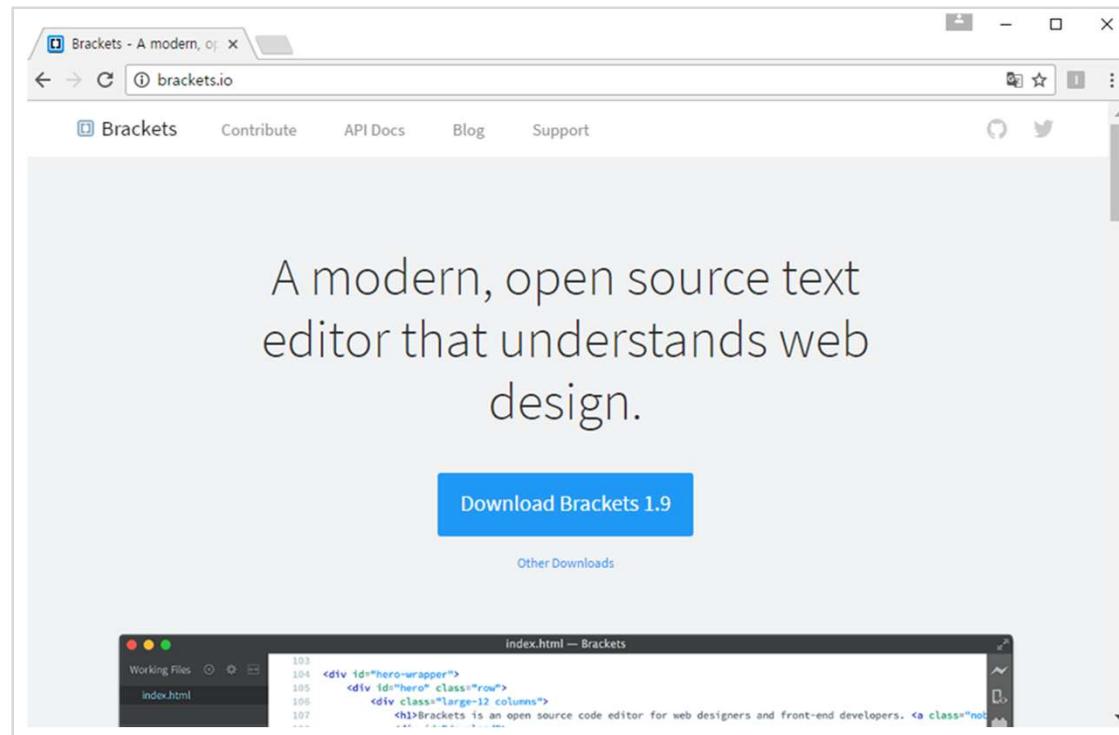
# Tiến trình cài đặt

- Có thể tạo mã nguồn của node bằng một trình soạn thảo đơn giản → Brackets, EditPlus, v.v.
- Khóa học này sử dụng trình soạn thảo văn bản nguồn Brackets



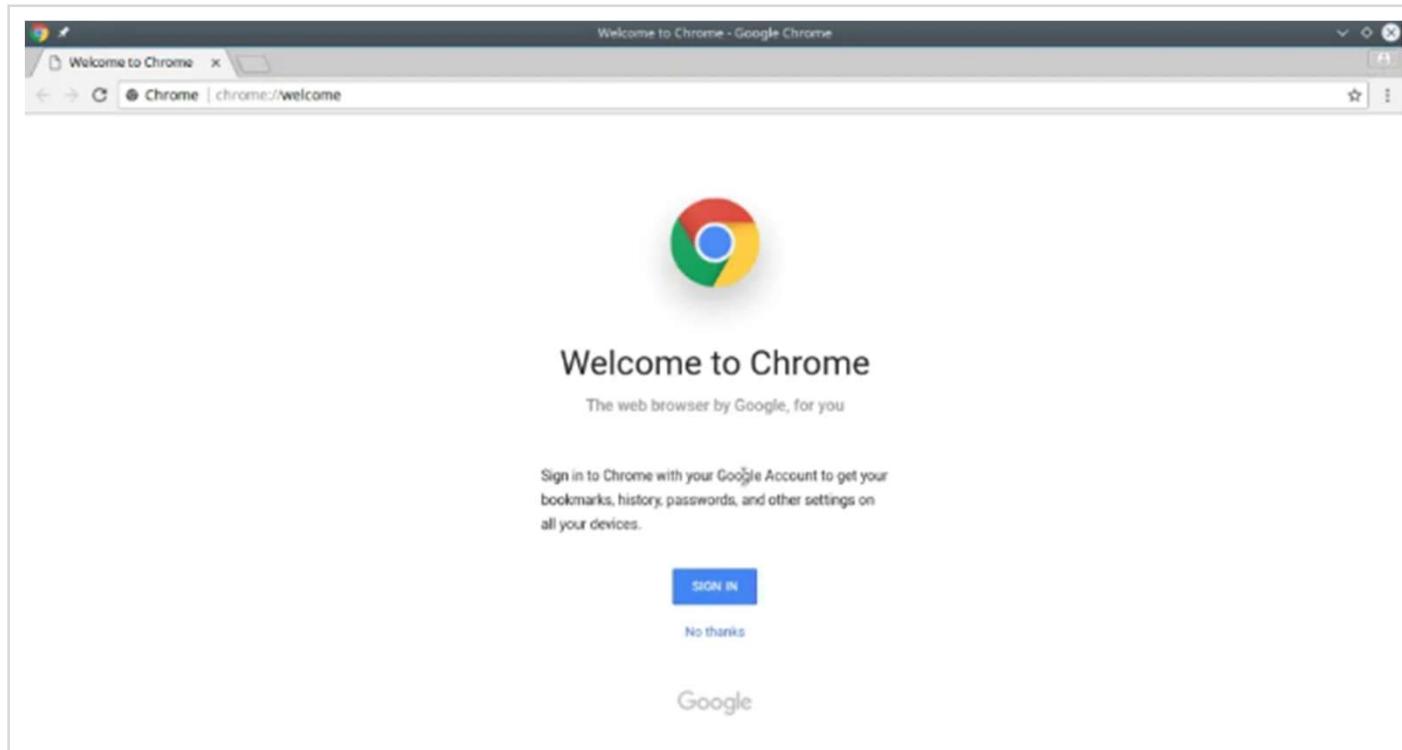
# Cài đặt trình soạn thảo văn bản nguồn Brackets

- Tải xuống từ <http://brackets.io> và cài đặt



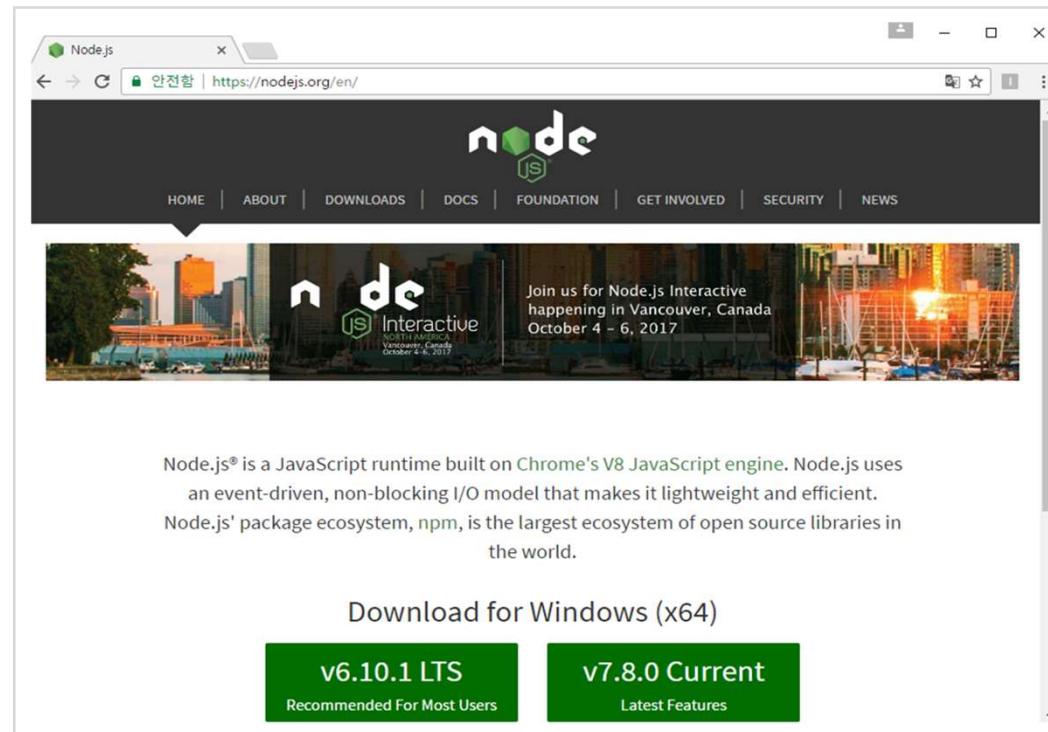
# Cài đặt trình duyệt Chrome

- Tải xuống từ <http://www.google.com> và cài đặt



## Cài đặt Node.js

- Có phiên bản LTS và phiên bản Current. Phiên bản LTS là phiên bản duy trì các hàm hiện có trong một khoảng thời gian nhất định để ổn định hệ thống và Phiên bản Current là phiên bản mới nhất.
- Khóa học này sử dụng phiên bản Current.



### Thay đổi giao diện của Brackets

- Có thể thay đổi **giao diện** của brackets: Nhấp vào View → menu Themes
- Tăng kích cỡ phông chữ: <Ctrl> + +
- Giảm kích cỡ phông chữ: <Ctrl> + -

### Mở thư mục Project

- Trong File Explorer, tạo thư mục “brackets\_nodejs” tại thư mục “NodeExample” nằm trong thư mục tài khoản người dùng Windows
- Nhấp vào File → Mở menu Folder để chỉ định vị trí của thư mục
- Nhấp chuột phải vào vùng bên trái của dự án và nhấp vào menu New File để tạo tệp

# Tạo một tệp JavaScript và chạy trong dấu nhắc lệnh

- Nhập một **code** JavaScript đơn giản
- Di tới thư mục dự án và chạy tệp tại node test1.js trong dấu nhắc lệnh  
(Nếu đã tạo tên tệp là ch06\_01.js, hãy chạy tệp này dưới dạng node ch06\_01.js)

The screenshot displays two windows side-by-side. On the left is the Brackets code editor, showing a single file named 'test1.js' with the code 'console.log('Hi there!');'. On the right is a terminal window titled 'Node.js command prompt' showing the command 'C:\brackets-nodejs\NodeExample>node ch06\_01.js' followed by the output 'Hi there!'.

# Tạo tệp HTML và xem trước trong trình duyệt web

- | Tạo tệp ch06\_01.html và nhập các thẻ HTML đơn giản
- | Nhập vào biểu tượng xem trước trực tiếp để mở trang trong trình duyệt Chrome
- | Các mã thay đổi sẽ được hiển thị tự động

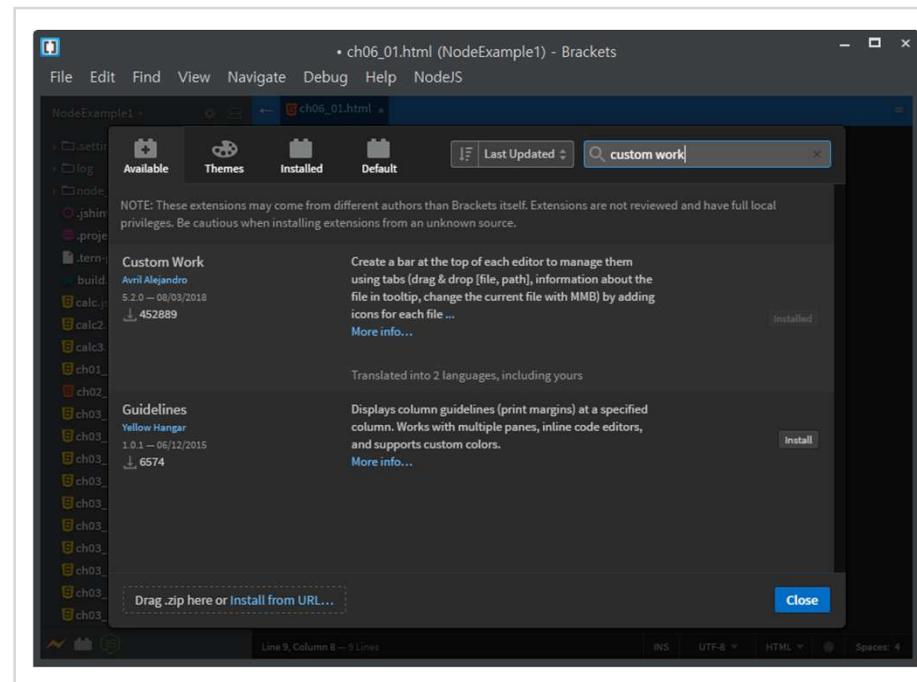
The screenshot displays two windows side-by-side. On the left is a code editor window titled 'ch06\_01.html (NodeExample1) - Brackets'. The file content is:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1>Hi there!</h1>
</body>
</html>
```

On the right is a web browser window showing the URL '127.0.0.1:2143/ch02\_01.html'. The page content is 'Hi there!'. This demonstrates how changes made in the code editor are immediately reflected in the browser's preview.

## Cài đặt phần mở rộng

- Có sẵn nhiều phần mở rộng
- Nhấp vào biểu tượng [Extension Manager] và cài đặt menu plug-in (phần bổ trợ) bên dưới:
- Brackets Icon Custom Work (Nếu bạn thấy hộp thoại Option sau khi cài đặt, hãy tắt Use Icon Option ở trên cùng)



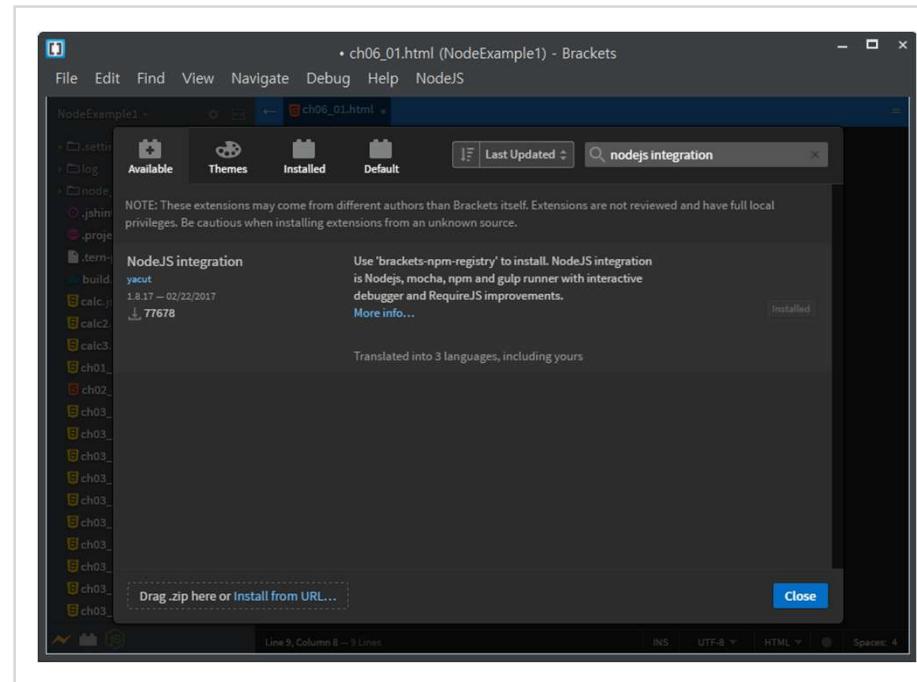
Bài 1.

# Tổng quan về Node.js

- 1.1. Tổng quan về Node.js
- 1.2. Cài đặt các công cụ phát triển
- 1.3. Tạo dự án node đầu tiên
- 1.4. Gửi log lên console
- 1.5. Sử dụng Mô-đun trong Node
- 1.6. Sử dụng mô-đun tích hợp sẵn

## Tạo Node trong Brackets

- Nhấp vào File → Mở menu Folder và chỉ định thư mục NodeExample trong thư mục brackets\_nodejs  
(Nếu đã được chỉ định sẵn, hãy giữ nguyên)
- Cài đặt NodeJs integration trong số các hàm mở rộng



### Chạy Node trực tiếp trong Brackets

- Tệp JavaScript có thể được thực thi bằng cách mở dấu nhắc lệnh và sử dụng node có thể thực thi được.
- Có thể chạy bằng phần mở rộng của NodeJs **tích hợp** trong Brackets
- Chọn tệp trong vùng bên trái của dự án và nhấn chuột phải để chọn menu Add to Node.js.
- Chọn tệp mới tạo từ thẻ dưới cùng và chạy tệp đó

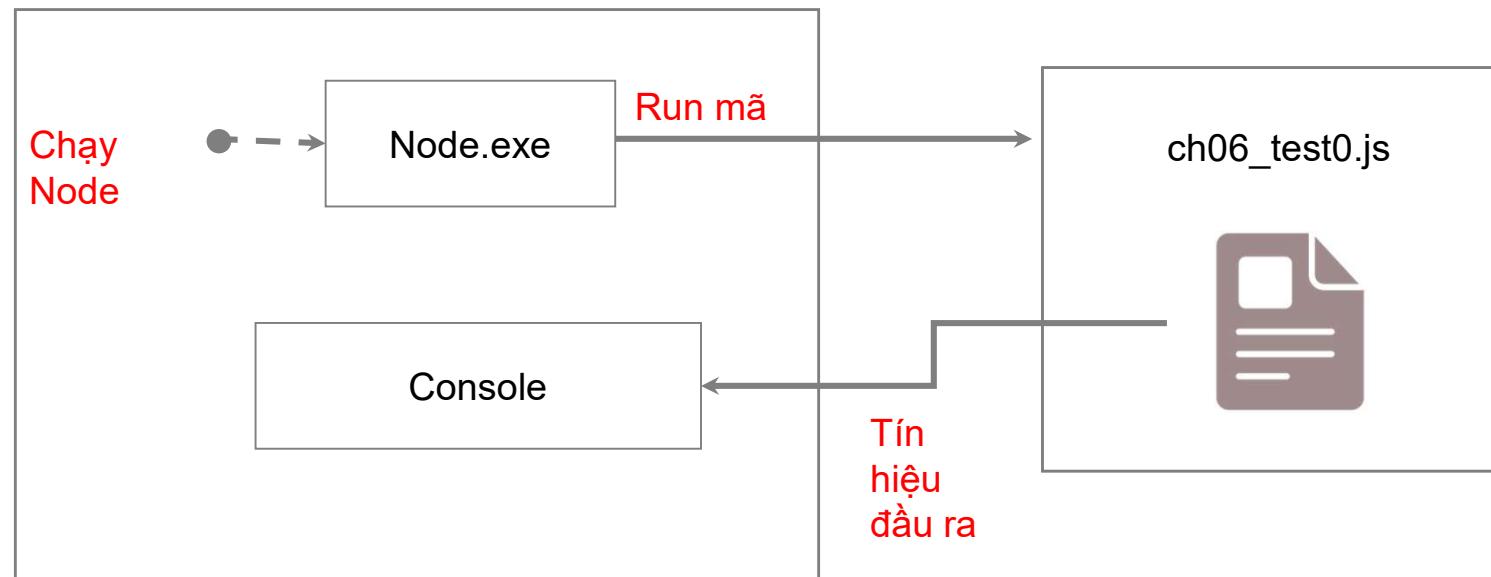
### Gỡ lỗi các node trong Brackets

- Gỡ lỗi bằng nút gỡ lỗi

## Thực thi chương trình Node

- Khi sử dụng lệnh node, nội dung của tệp JavaScript chỉ định sẽ được đọc và thực thi. Kết quả thực thi được hiển thị trong console.

Brackets



Bài 1.

# Tổng quan về Node.js

- 1.1. Tổng quan về Node.js
- 1.2. Cài đặt các công cụ phát triển
- 1.3. Tạo dự án node đầu tiên
- 1.4. Gửi log lên console
- 1.5. Sử dụng Mô-đun trong Node
- 1.6. Sử dụng mô-đun tích hợp sẵn

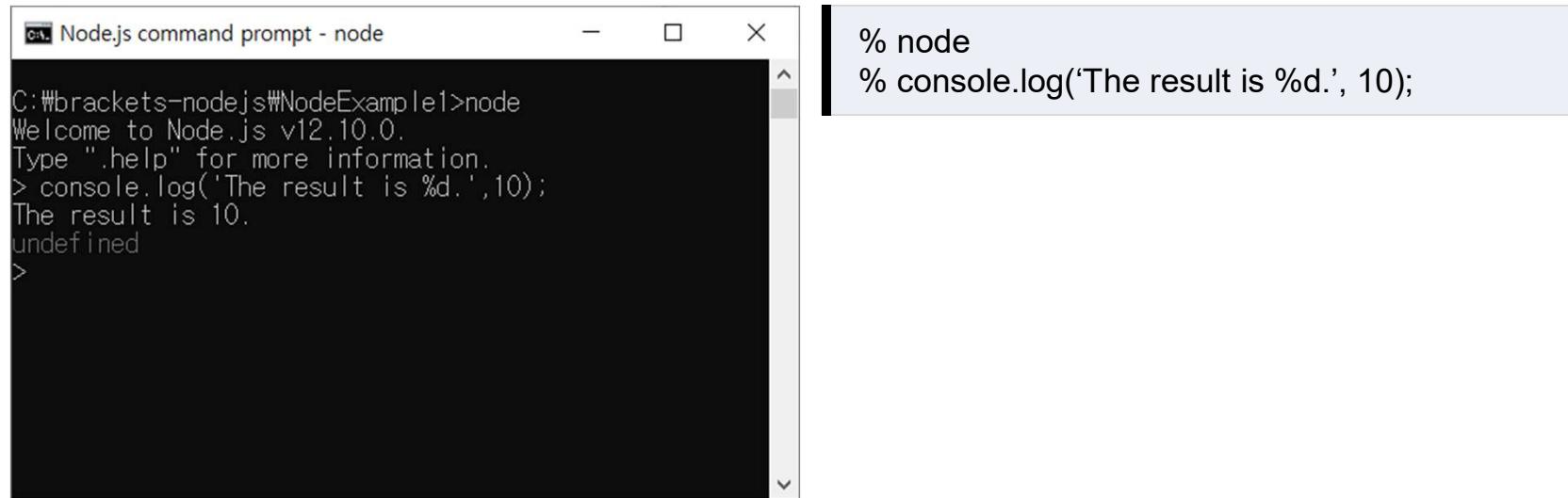
# Chạy các tệp JavaScript trong dấu nhắc lệnh

- Khi bạn chạy dấu nhắc lệnh có lệnh cmd, thư mục người dùng sẽ được đặt làm vị trí mặc định.
- Chuyển đến thư mục NodeExample và chạy javascript bằng lệnh node (không cần nhập ký hiệu %).

```
% cd brackets_nodejs\NodeExample  
% node ch06_test1.js
```

# Viết và thực thi mã ở dấu nhắc lệnh

- Có thể nhập mã trực tiếp ở dấu nhắc lệnh
- Khi bạn thực thi lệnh node, hệ thống cho phép bạn bắt đầu nhập mã.



The screenshot shows a terminal window titled "Node.js command prompt - node". The window displays the following interaction:

```
C:\brackets-nodejs\NodeExample1>node
Welcome to Node.js v12.10.0.
Type ".help" for more information.
> console.log('The result is %d.', 10);
The result is 10.
undefined
>
```

A callout box highlights the command `% node` and its output `% console.log('The result is %d.', 10);`.

## Đối tượng Console

- Console hiển thị các log dưới dạng các đối tượng toàn cục có thể được sử dụng ở mọi nơi

Tên đối tượng toàn cục	Mô tả
console	Đối tượng hiển thị kết quả trong cửa sổ console
process	Đối tượng xử lý thông tin thực thi tiến trình
exports	Đối tượng xử lý mô-đun

```
% console.log('Display numbers : %d', 10);
% console.log('Display string : %s', 'Hello!');
% console.log('Display JSON Object : %j', {name: 'June'});
```

## Các phương thức chính của các đối tượng Console

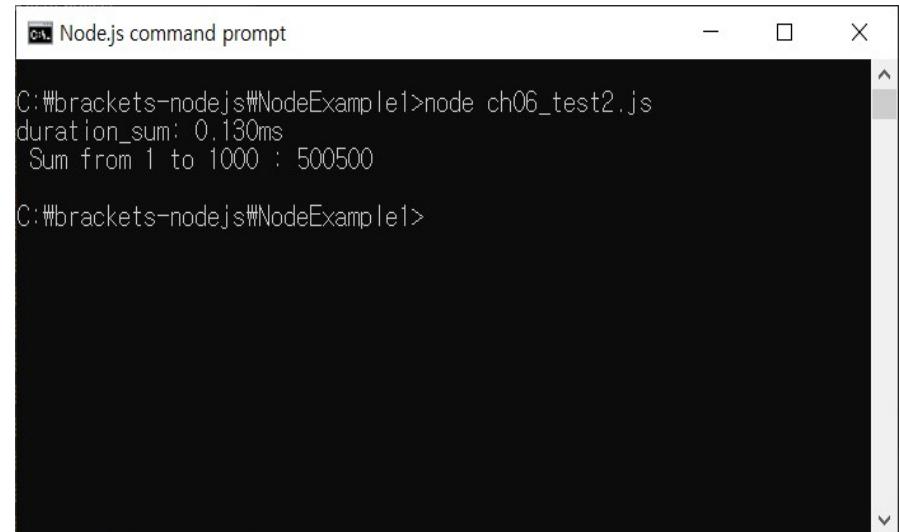
- Các phương thức như dir, time, timeEnd tồn tại trong các đối tượng Console

Tên đối tượng toàn cục	Mô tả
dir(object)	Hiển thị các thuộc tính của đối tượng JavaScript
time (id)	Ghi lại thời gian bắt đầu để đo thời gian chạy
timeEnd (id)	Ghi lại thời gian kết thúc để đo thời gian chạy

## Kiểm tra thời gian thực thi mã

- Chọn dự án [NodeExample] ở bên trái, nhấp chuột phải và chọn [Create File].
- Nhập làm tên tệp
- Nhập mã sau vào tệp mới tạo và thực thi tệp

```
1 var result = 0;
2
3 console.time('duration_sum');
4 for (var i = 1; i <= 1000; i++) {
5     result += i;
6 }
7 console.timeEnd('duration_sum');
8 console.log(' Sum from 1 to 1000 : %d', result);
```



The screenshot shows a terminal window titled "Node.js command prompt". The command entered is "node ch06\_test2.js". The output displays the time taken for the execution and the resulting sum of numbers from 1 to 1000.

```
C:\brackets-nodejs\NodeExample>node ch06_test2.js
duration_sum: 0.130ms
Sum from 1 to 1000 : 500500
C:\brackets-nodejs\NodeExample>
```

## Hiển thị tên tệp và thông tin về đối tượng

- Hiển thị tên tệp bằng cách sử dụng các biến toàn cục như \_\_filename, \_\_dirname
- Hiển thị thông tin đối tượng bằng cách sử dụng hàm console.dir
- Tạo các đối tượng JavaScript bằng cách sử dụng {}
- Định dạng là {tên thuộc tính: giá trị thuộc tính, tên thuộc tính: giá trị thuộc tính}

```
1  var result = 0;
2
3  console.time(label: 'duration_sum');
4  for (var i = 1; i <= 1000; i++) {
5      result += i;
6  }
7  console.timeEnd(label: 'duration_sum');
8  console.log(' Sum from 1 to 1000 : %d', result);
9
10 console.log(' The name of the currently executed file : %s',
11             __filename);
12 console.log(' The path of the currently executed file : %s',
13             __dirname);
14
15 var Person = {name:"June", age:20};
16 console.dir(Person);
```

```
C:\brackets-nodejs\NodeExample1>node ch06_test2.js
duration_sum: 0.115ms
Sum from 1 to 1000 : 500500
The name of the currently executed file : C:\brackets-nodejs\NodeExample1\ch06_test2.js
The path of the currently executed file : C:\brackets-nodejs\NodeExample1
{ name: 'June' , age: 20 }

C:\brackets-nodejs\NodeExample1>
```

## Đối tượng Process

- Đối tượng Process là một đối tượng xử lý thông tin về quy trình

Tên đối tượng toàn cục	Mô tả
argv	Thông tin về tham số được truyền đi khi thực thi tiến trình
env	Thông tin về biến môi trường
exit()	Phương thức kết thúc tiến trình

```
1  console.log(' Parameter number of argv attribute : '
2      + process.argv.length);
3  console.dir(process.argv);
4
5  process.argv.forEach(function(item, index) {
6      console.log(index + ' : ', item);
7  });

```

Bài 1.

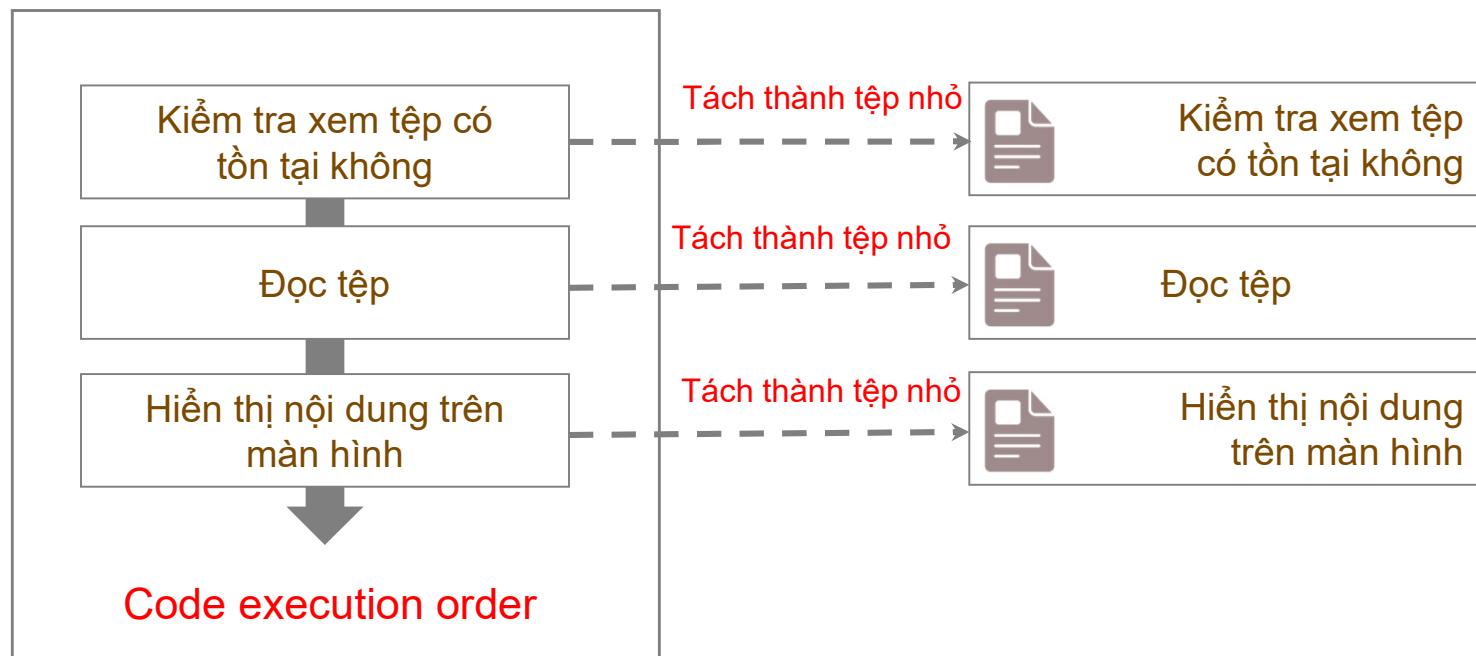
# Tổng quan về Node.js

- 1.1. Tổng quan về Node.js
- 1.2. Cài đặt các công cụ phát triển
- 1.3. Tạo dự án node đầu tiên
- 1.4. Gửi log lên console
- 1.5. Sử dụng Mô-đun trong Node
- 1.6. Sử dụng mô-đun tích hợp sẵn

## Chia hàm thành các tệp riêng biệt

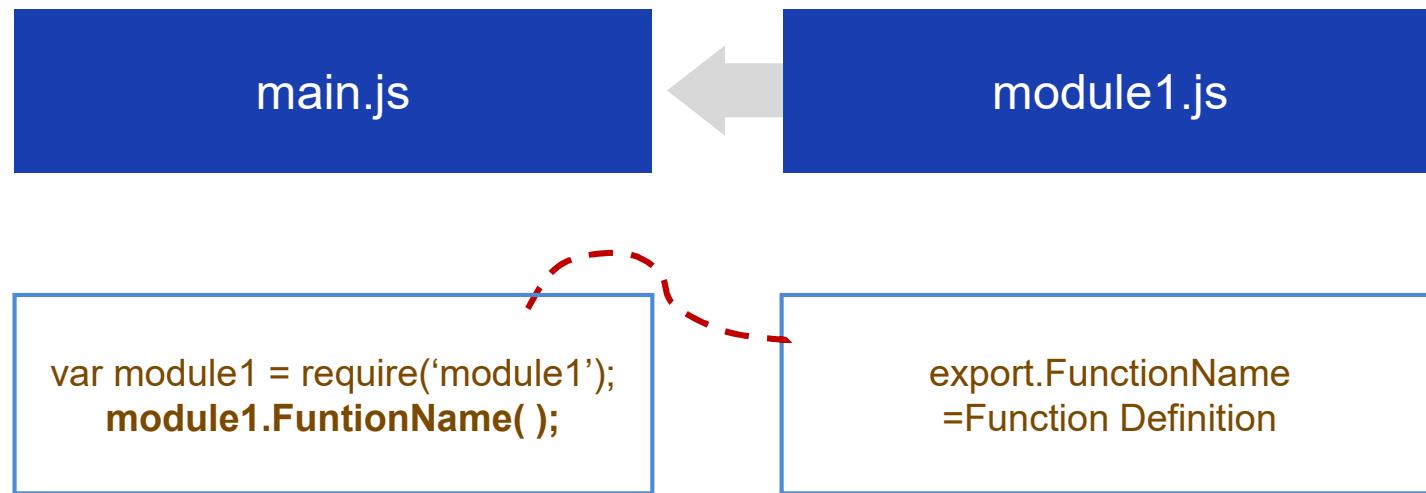
- Mã code là cần thiết khi đọc các tệp có thể được chia thành các tệp riêng biệt tùy thuộc vào hàm của chúng.

Tệp chính



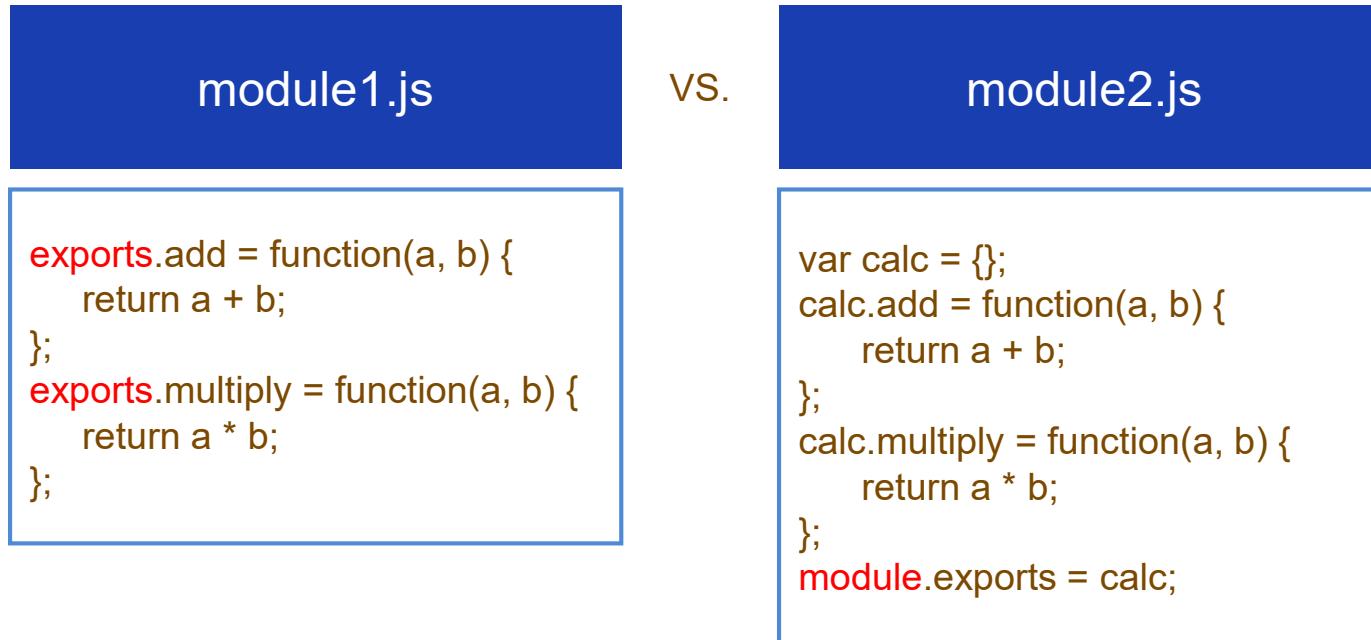
## Mô-đun

- Mô-đun: Một hàm độc lập được tách thành một tệp
- Khi bạn đã tạo một mô-đun, bạn có thể tải mô-đun đó từ một tệp khác để sử dụng.
- Biên dịch với các tham số tiêu chuẩn của CommonJS và sử dụng đối tượng toàn cục exports



## Sử dụng exports và module.exports

- Bạn có thể sử dụng exports trong các tệp mô-đun hoặc module.exports
- Để trực tiếp cấp phát một đối tượng, hãy sử dụng module.exports.



## Tách một hàm **cộng** thành các mô-đun

- Mã cho hàm plus trước khi tách thành các mô-đun

```
1 var calc = {};
2 calc.add = function(a, b) {
3     return a + b;
4 }
5 console.log(' Before splitting into modules - ' +
6     'the result of calling calc.add function : %d',
7     calc.add( a: 10, b: 10));
```

The screenshot shows a terminal window titled "Node.js command prompt". The command entered is "node ch06\_test4.js". The output displayed is:

```
C:\brackets-nodejs\NodeExample1>node ch06_test4.js
Before split into modules - the result of calling calc.add function : 20
C:\brackets-nodejs\NodeExample1>
```

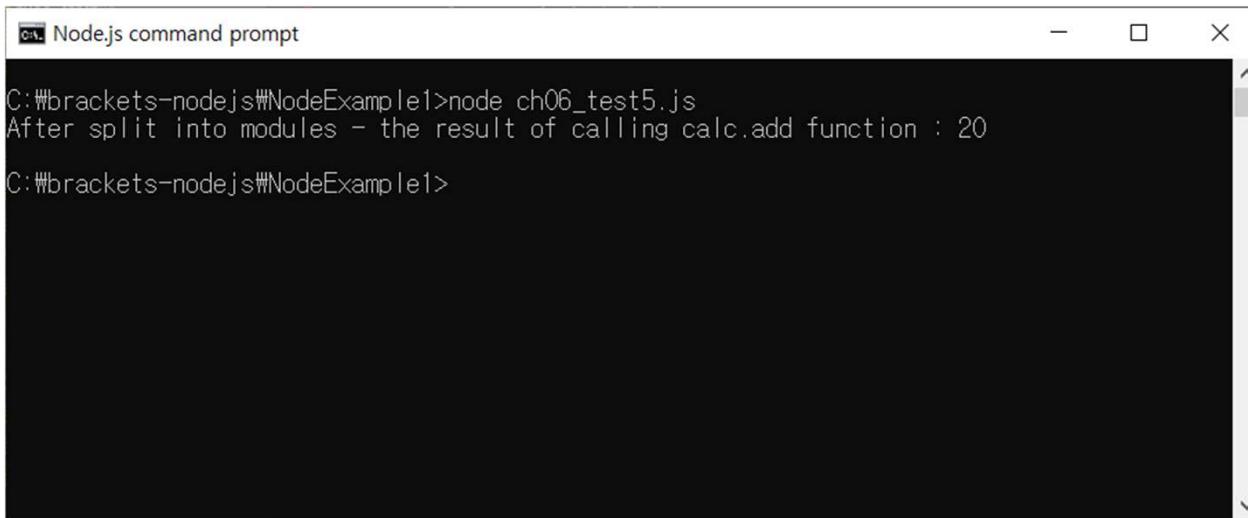
## Tệp mô-đun

- Tạo tệp mô-đun → calc.js
- Thêm một hàm dưới dạng thuộc tính add của đối tượng exports

```
1  exports.add = function(a, b) {  
2      return a + b;  
3  }
```

- Gọi hàm add sau khi nạp tệp mô-đun
- Nạp tệp mô-đun bằng hàm require
- Đối tượng result đã nhập có thể được coi như đối tượng exports
- Việc chỉ định một thư mục thay vì một tệp sẽ gọi tệp index.js trong thư mục đó

```
1 var calc = require('./calc');
2 console.log(' After splitting into modules - ' +
3   'the result of calling calc.add function : %d',
4   calc.add(10, 10));
```



The screenshot shows a terminal window titled "Node.js command prompt". The command entered is "node ch06\_test5.js". The output displayed is "After split into modules - the result of calling calc.add function : 20".

```
C:\brackets>node ch06_test5.js
After split into modules - the result of calling calc.add function : 20
```

## Sử dụng đối tượng module.exports trong một tệp mô-đun

- Tạo tệp mô-đun, calc2.js
- Tạo một đối tượng calc và gán đối tượng đó cho module.exports

```
1  var calc = {};
2  calc.add = function(a, b) {
3      return a + b;
4  }
5  module.exports = calc;
```

## Tệp chính

- Gọi hàm add sau khi nạp tệp mô-đun
- Nạp tệp mô-đun bằng hàm require
- Đối tượng result đã nhập có thể được coi như một đối tượng module.exports

```
1 var calc2 = require('./calc2');
2 console.log(' After splitting into modules - ' +
3     'the result of calling calc2.add function : %d',
4     calc2.add(10, 10));
```

The screenshot shows a terminal window titled "Node.js command prompt". The command entered is "node ch06\_test5.js". The output shows two lines of text: "After split into modules - the result of calling calc.add function : 20" and "After split into modules - the result of calling calc2.add function : 20". This demonstrates that both module exports are available in the global scope.

## Sử dụng mô-đun bên ngoài

- Không sử dụng đường dẫn tương đối khi sử dụng mô-đun bên ngoài.
- mô-đun nconf cho phép truy cập vào các biến môi trường trên hệ thống

```
1 var nconf = require('nconf');
2 nconf.env();
3 console.log('value of OS environment variables : %s',
4     nconf.get('OS'));
```

## Sử dụng mô-đun bên ngoài

- Để sử dụng mô-đun bên ngoài, bạn cần cài đặt gói bằng npm.
- Sau khi cài đặt gói, chạy tệp JavaScript đã tạo trước đó rồi hiển thị các biến môi trường trên OS.

```
% npm install nconf
```

The screenshot shows a terminal window titled "Node.js command prompt". The command "% npm install nconf" was entered at the prompt. The output shows the installation of the nconf package from version 0.8.5 to 0.1.0. A warning message indicates that the package NodeExample1 has no repository field. The command was run in a directory named "NodeExample1" located in "brackets-nodejs". The audit step found 0 vulnerabilities. The terminal prompt ends with "C:\brackets-nodejs\NodeExample1>".

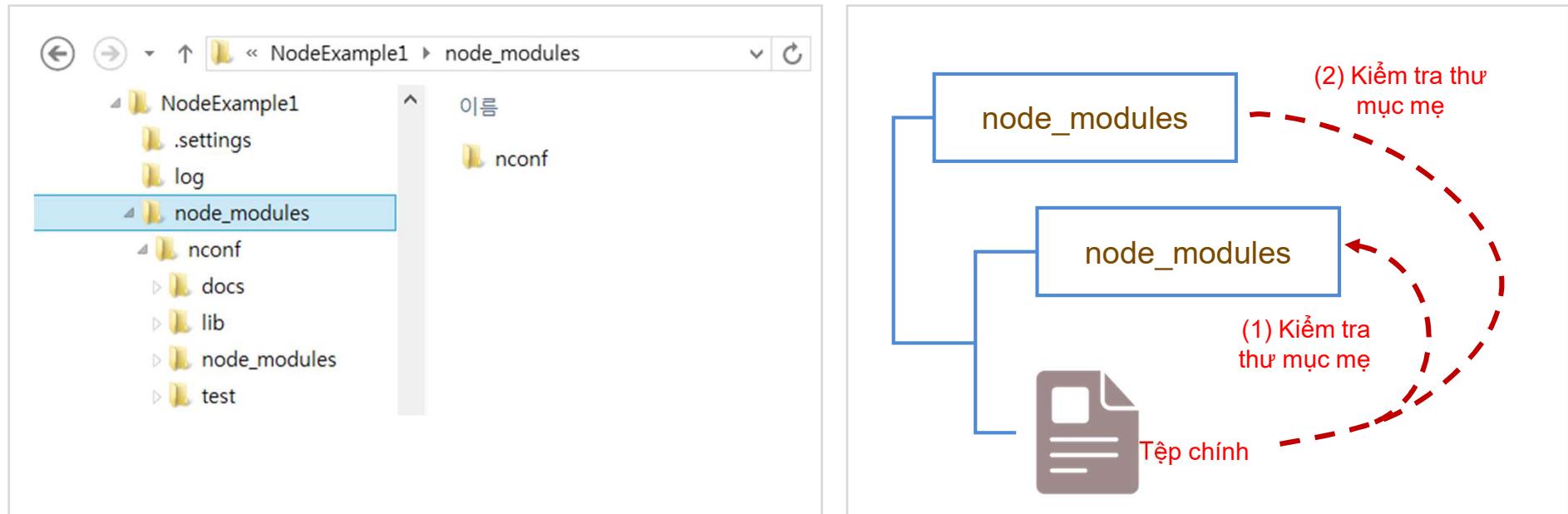
```
C:\brackets-nodejs\NodeExample1>npm install nconf
npm WARN NodeExample1@0.1.0 No repository field.

+ nconf@0.8.5
updated 1 package and audited 47 packages in 1.643s
found 0 vulnerabilities

C:\brackets-nodejs\NodeExample1>
```

## Sử dụng mô-đun bên ngoài đã cài đặt

- Các gói được cài đặt bên trong thư mục node\_modules.
- Có thể được sử dụng theo từng dự án nhưng cũng có thể sử dụng trong thư mục mẹ của dự án.



## tệp package.json

- Bạn có thể kiểm tra thông tin gói đã cài đặt bằng npm.

```
{  
  "name": "NodeExample",  
  "version": "0.1.0",  
  "description": "NodeExample1",  
  "main": "hello-world-server.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified! Configure in package.json\\\" && exit 1"  
  },  
  "repository": "",  
  "keywords": [  
    "node.js",  
    "eclipse",  
    "nodeclipse"  
  ],  
  "author": "",  
  "license": "MIT",  
  "readmeFilename": "README.md"  
}
```

## Thêm thông tin vào Package.json và xóa các gói

- Xóa bằng cách sử dụng 'npm uninstall nconf'.
- Thêm thông tin gói vào tệp 'package.json' với tùy chọn 'npm install nconf --save'.
- Nếu bạn đã sao chép thư mục dự án, hãy sử dụng lệnh npm install để cài đặt tất cả các gói bằng thông tin gói có trong 'package.json'.

```
{  
  ....  
  ....  
  "dependencies": {  
    "nconf": "^0.3.4"  
  }  
}
```

Bài 1.

# Tổng quan về Node.js

- 1.1. Tổng quan về Node.js
- 1.2. Cài đặt các công cụ phát triển
- 1.3. Tạo dự án node đầu tiên
- 1.4. Gửi log lên console
- 1.5. Sử dụng Mô-đun trong Node
- 1.6. Sử dụng mô-đun tích hợp sẵn

## Mô-đun tích hợp

- Mô-đun được cung cấp khi cài đặt node.
- Có sẵn trên trang web <http://nodejs.org/api>.
- Mô-đun OS thông báo cho bạn thông tin về hệ thống.

Tên đối tượng toàn cục	Giải thích
hostname()	Cho bạn biết tên máy chủ của hệ điều hành.
totalmem()	Cho bạn biết tổng dung lượng bộ nhớ của hệ thống.
freemem()	Cho bạn biết dung lượng còn trống trên hệ thống.
cpus()	Cho bạn biết thông tin về CPU
networkInterfaces()	Trả về một đối tượng array chứa thông tin về giao diện mạng.

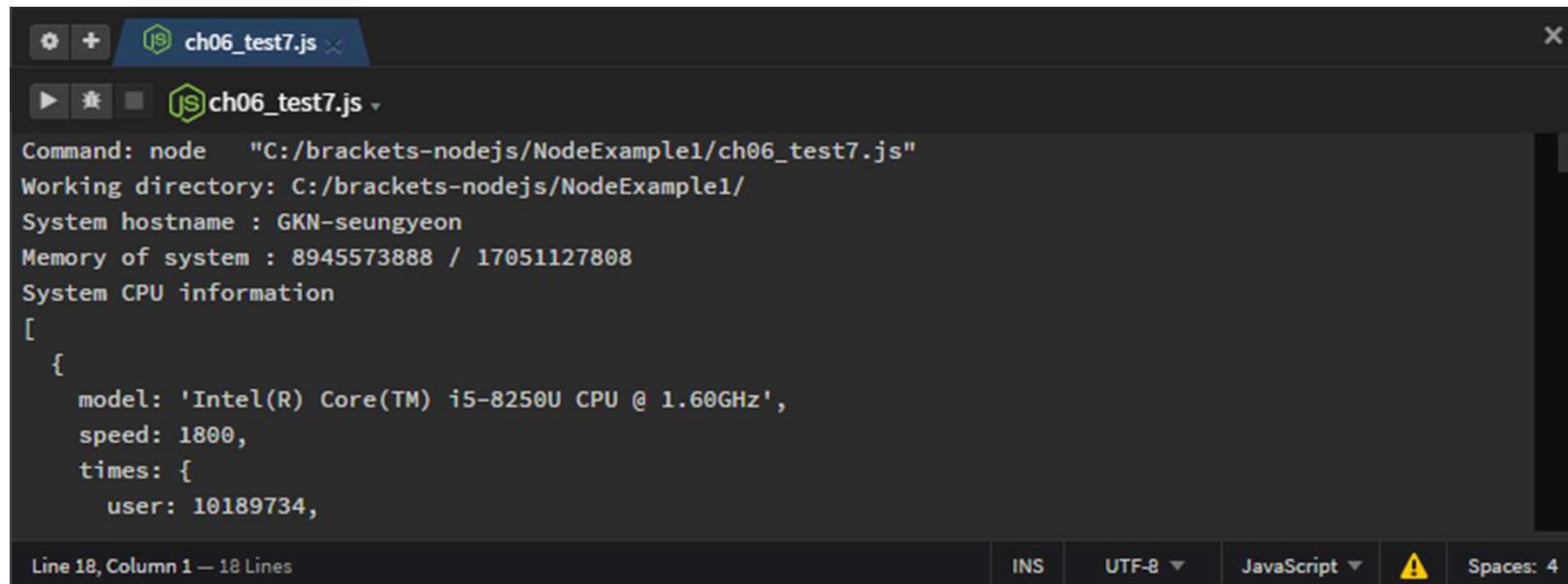
## Mô-đun OS

- Gọi mô-đun OS bằng hàm 'require'.

```
1  var os = require('os');
2  console.log('System hostname:% s', os.hostname ());
3  console.log ('Memory of system:% d /% d',
4    os.freemem (), os.totalmem ());
5  console.log('System CPU information \ n');
6  console.dir(os.cpus());
7  console.log ('System network interface information \ n');
8  console.dir(os.networkInterfaces());
```

## Mô-đun OS

- Nạp mô-đun bằng hàm require



```
ch06_test7.js
ch06_test7.js

Command: node "C:/brackets-nodejs/NodeExample1/ch06_test7.js"
Working directory: C:/brackets-nodejs/NodeExample1/
System hostname : GKN-seungyeon
Memory of system : 8945573888 / 17051127808
System CPU information
[
  {
    model: 'Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz',
    speed: 1800,
    times: {
      user: 10189734,
```

Line 18, Column 1 — 18 Lines      INS      UTF-8 ▾      JavaScript ▾      A      Spaces: 4

## Mô-đun đường dẫn

- Cung cấp các tính năng như phân biệt tên tệp với đường dẫn tệp

Tên đối tượng toàn cục	Giải thích
join()	Kết hợp các tên với nhau thành một đường dẫn tệp duy nhất. Đối tượng này tự động điều chỉnh dấu phân cách khi hoàn thành đường dẫn tệp
dirname()	Trả về tên thư mục từ đường dẫn tệp.
basename()	Trả về tên đường dẫn tệp mà không có phần mở rộng tệp.
extname()	Trả về phần mở rộng của tệp trong đường dẫn tệp.

## Mô-đun đường dẫn

- Xác định thư mục, tên tệp và phần mở rộng trong đường dẫn tệp.

```
1  var path = require('path');
2  // Combining Directory Names
3  var directories = ["users", "mike", "docs"];
4  var docsDirectory = directories.join(path.sep);
5  console.log(' Document directory : %s', docsDirectory);
6  // Combining Directory Names and File Names
7  var curPath = path.join( separator: '/Users/mike', 'notepad.exe');
8  console.log(' File path : %s', curPath);
9
10 // Identifies directories, file names, and extensions in path
11 var filename = "C:\\\\Users\\\\mike\\\\notepad.exe";
12 var dirname = path.dirname(filename);
13 var basename = path.basename(filename);
14 var extname = path.extname(filename);
15 console.log('directory : %s, file name : %s, extension : %s',
16             dirname, basename, extname);
```

## Mô-đun đường dẫn

- Xác định thư mục, tên tệp và phần mở rộng trong đường dẫn tệp.



```
ch06_test8.js
ch06_test8.js

Command: node "C:/brackets-nodejs/NodeExample1/ch06_test8.js"
Working directory: C:/brackets-nodejs/NodeExample1/
Document directory : users\mike\docs
File path : \Users\mike\notepad.exe
directory : C:\Users\mike, file name : notepad.exe, extension : .exe
Program exited with code 0

Line 28, Column 1 — 28 Lines           INS    UTF-8 ▾   JavaScript ▾   Spaces: 4
```

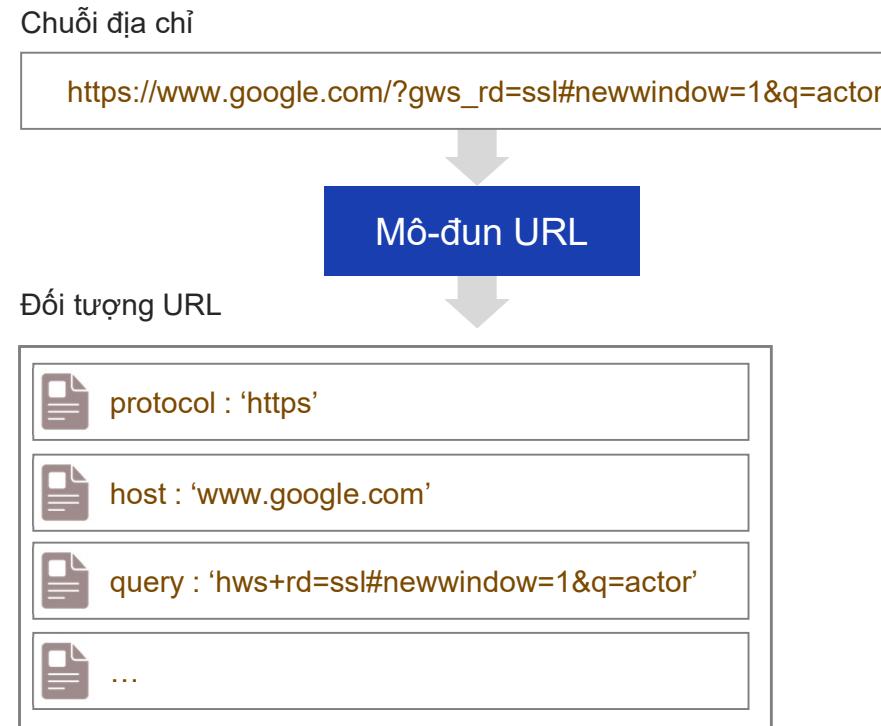
Bài 2.

# Lập trình Node.js cơ bản

- 2.1. Chuỗi địa chỉ và tham số truy vấn
- 2.2. Tìm hiểu về Sự kiện
- 2.3. Sử dụng Tệp
- 2.4. Duy trì tệp Log

# Mô-đun URL được sử dụng để tách các chuỗi địa chỉ

- Chuyển đổi chuỗi plain string thành đối tượng URL hoặc chuyển đổi đối tượng URL thành chuỗi plain string



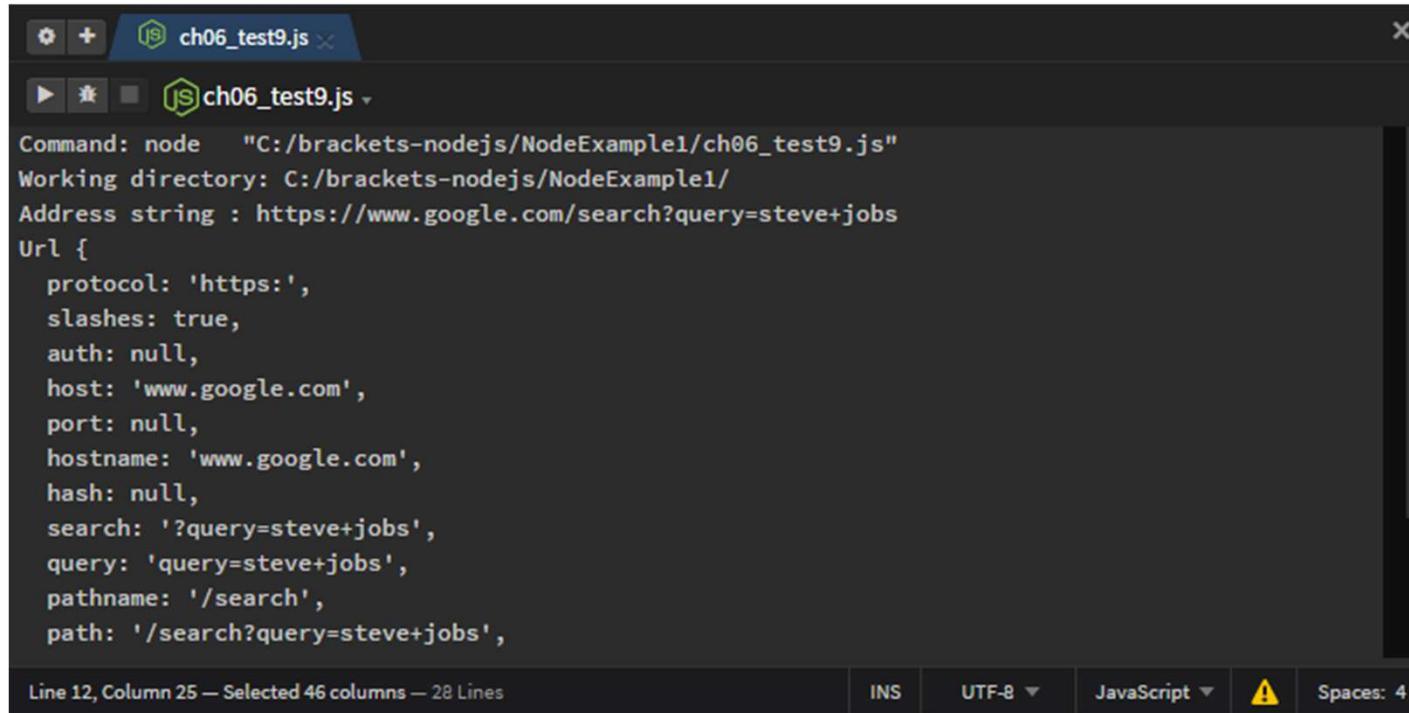
## Các phương thức chính của mô-đun URL

Phương thức	Giải thích
parse(urlStr, [parseQueryString = false, slashesDenoteHost = false])	Phân tích cú pháp chuỗi địa chỉ để tạo đối tượng URL
format(urlObj)	Chuyển đổi đối tượng URL thành chuỗi địa chỉ

```
1 var url = require('url');
2 var curURL = url.parse(
3   'https://www.google.com/search?query=steve+jobs');
4 var curStr = url.format(curURL);
5
6 console.log('address string : %s', curStr);
7 console.dir(curURL);
```

## Phân tích URL bằng phương thức parse (phân tích cú pháp)

- Màn hình hiển thị kết quả đã phân tích



```
ch06_test9.js
ch06_test9.js

Command: node "C:/brackets-nodejs/NodeExample1/ch06_test9.js"
Working directory: C:/brackets-nodejs/NodeExample1/
Address string : https://www.google.com/search?query=steve+jobs
Url {
  protocol: 'https:',
  slashes: true,
  auth: null,
  host: 'www.google.com',
  port: null,
  hostname: 'www.google.com',
  hash: null,
  search: '?query=steve+jobs',
  query: 'query=steve+jobs',
  pathname: '/search',
  path: '/search?query=steve+jobs',
```

Line 12, Column 25 — Selected 46 columns — 28 Lines    INS    UTF-8 ▾    JavaScript ▾    ⚠    Spaces: 4

## Mô-đun querystring dùng để Kiểm tra tham số truy vấn

- Được sử dụng để phân tách tham số truy vấn được phân đoạn bằng ký hiệu &
- Sử dụng Phương thức parse và stringify sau khi nạp mô-đun với phương thức require

```
1  var url = require('url');
2  var curURL = url.parse(
3      s: 'https://www.google.com/search?query=steve+jobs');
4  var curStr = url.format(curURL);
5
6  console.log('address string : %s', curStr);
7  console.dir(curURL);
8
9  var querystring = require('querystring');
10 var param = querystring.parse(curURL.query);
11 console.log('query value in the query parameters : %s',
12             param.query);
13 console.log('original query parameter : %s',
14             querystring.stringify(param));
```

## Các phương thức trong mô-đun querystring

- Các phương thức trong mô-đun querystring

Phương thức	Giải thích
parse()	Phân tích chuỗi tham số truy vấn để tạo đối tượng tham số truy vấn
stringfy()	Chuyển đổi đối tượng tham số truy vấn thành chuỗi

```
ch06_test9.js
{
  sames: true,
  auth: null,
  host: 'www.google.com',
  port: null,
  hostname: 'www.google.com',
  hash: null,
  search: '?query=steve+jobs',
  query: 'query=steve+jobs',
  pathname: '/search',
  path: '/search?query=steve+jobs',
  href: 'https://www.google.com/search?query=steve+jobs'
}
Query value in the query parameters : steve jobs
Original query parameter : query=steve%20jobs
Program exited with code 0

Line 12, Column 25 — Selected 46 columns — 28 Lines
```

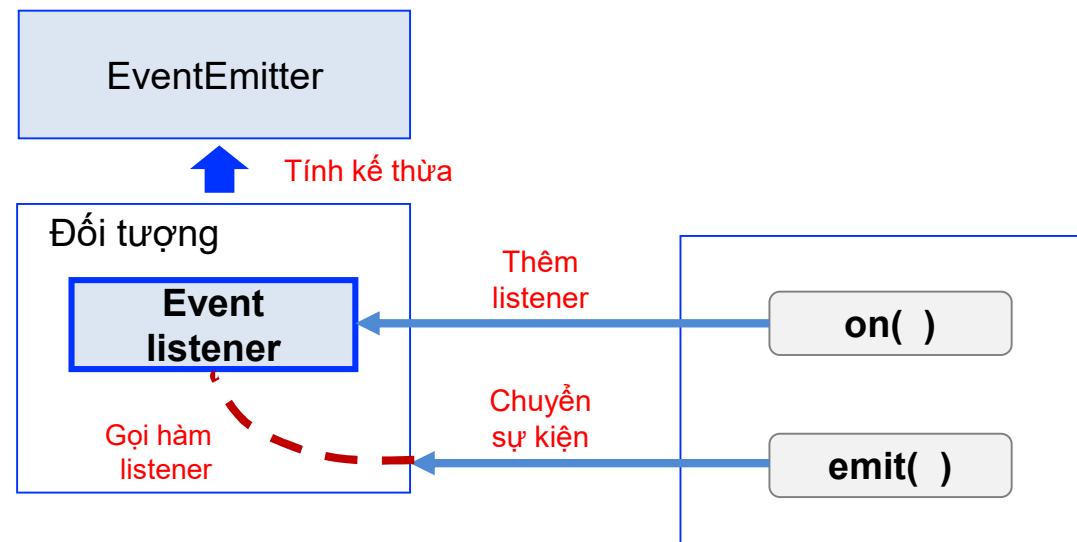
Bài 2.

# Lập trình Node.js cơ bản

- 2.1. Chuỗi địa chỉ và tham số truy vấn
- 2.2. Tìm hiểu về Sự kiện
- 2.3. Sử dụng Tệp
- 2.4. Duy trì tệp Log

## Sự kiện là gì?

- Chuyển dữ liệu từ bên này sang bên khác để xử lý không đồng bộ
- Sử dụng hàm EventEmitter
- Khi gửi sự kiện từ bên này sang bên khác, sử dụng phương thức emit(). Phương thức on() **thêm** event listener cho đối tượng mà sự kiện được chuyển đến.



## Gửi và nhận sự kiện

- Đăng ký listener với on() và **chuyển** sự kiện bằng emit()

Phương thức	Giải thích
on(event, listener)	Thêm một listener cho một event chỉ định
once(event, listener)	Thêm một listener cho một event chỉ định, nhưng tự động xóa listener đó sau một lần chạy
removeListener(event, listener)	Xóa một listener cho một event chỉ định
emit(event, param)	Chuyển sự kiện

```
1 process.on('tick', function(count) {  
2     console.log('tick event occurs: %s', count);  
3 });  
4 setTimeout(handler: function() {  
5     console.log('attempt to pass tick event in 2 seconds.');  
6     process.emit('tick', '2');  
7 }, timeout: 2000);
```

## Cấu hình đối tượng Calculator thành Mô-đun

- Các phương thức emit và on đều khả dụng nếu đối tượng Calculator kế thừa EventEmitter

```
1  var util = require('util');
2  var EventEmitter = require('events').EventEmitter;
3  var Calc = function() {
4      var self = this;
5
6      this.on('stop', function() {
7          console.log('Passed stop event to Calc.');
8      });
9  };
10 util.inherits(Calc, EventEmitter);
11 Calc.prototype.add = function(a, b) {
12     return a + b;
13 }
14 module.exports = Calc;
15 module.exports.title = 'calculator';
```

## Sử dụng Đối tượng Calculator trong Tệp tin chính

- **Chuyển** sự kiện đến emit

```
1 var Calc = require('./calc3');
2 var calc = new Calc();
3 calc.emit('stop');
4 console.log('stop event sent to ' + Calc.title);
```

The screenshot shows the Brackets IDE interface. The top bar has tabs for 'ch06\_test10.js' and 'ch06\_test10.js'. The main area displays the code and its execution output. The output window shows:

```
Command: node "C:/brackets-nodejs/NodeExample1/ch06_test10.js"
Working directory: C:/brackets-nodejs/NodeExample1/
Passed stop event to Calc.
Dispatched stop event to calculator
Program exited with code 0
```

At the bottom, status bars indicate 'Line 10, Column 23 — 15 Lines', 'INS', 'UTF-8', 'JavaScript', and 'Spaces: 4'.

Bài 2.

## Lập trình Node.js cơ bản

- 2.1. Chuỗi địa chỉ và tham số truy vấn
- 2.2. Tìm hiểu về Sự kiện
- 2.3. Sử dụng Tệp
- 2.4. Duy trì tệp Log

## Hệ thống tệp của Node

- Cung cấp I/O đồng bộ và không đồng bộ
- Lưu ý rằng I/O đồng bộ sẽ chờ cho đến khi hoàn thành thao tác tệp
- Trong phương thức I/O đồng bộ, từ khóa Sync được thêm vào ở cuối dòng lệnh

```
1  var fs = require('fs');
2
3  var data = fs.readFileSync('./package.json', 'utf8');
4  console.log(data);
```

## Đọc tệp không đồng bộ

- Chuyển hàm Callback làm tham số, sử dụng phương thức readFile

```
1  var fs = require('fs');
2
3  fs.readFile('./package.json', 'utf8', function(err, data) {
4      console.log(data);
5  });
6
7  console.log(' Request to read package.json file ' +
8      'in the project folder.');
```

The screenshot shows the Brackets IDE interface with a code editor and a terminal window.

**Code Editor:**

```
ch06_test11.js
Command: node "C:/brackets-nodejs/NodeExample1/ch06_test11.js"
Working directory: C:/brackets-nodejs/NodeExample1/
Request to read package.json file in project folder.
{
  "name": "NodeExample1",
  "version": "0.1.0",
  "description": "NodeExample1",
  "main": "ch02_test0.js",
  "errprint": true
```

**Terminal Output:**

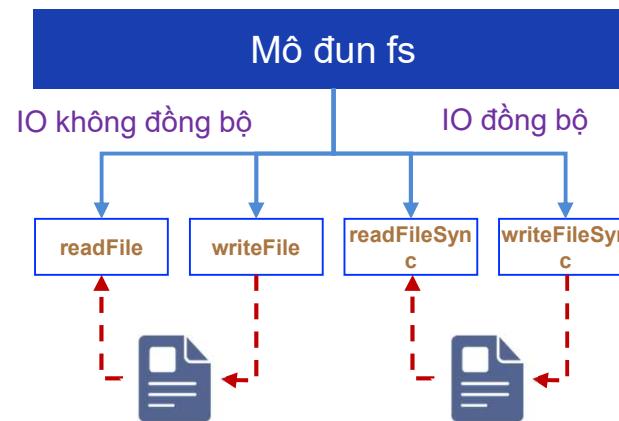
```
Line 17, Column 1 — 17 Lines | INS | UTF-8 | JavaScript | A | Spaces: 4
```

## Phương thức chính của mô-đun fs

- Đọc bằng readFile và ghi bằng writeFile

Phương thức	Giải thích
readFile(filename, [encoding], [callback])	Đọc tệp bằng I/O không đồng bộ
readFileSync(filename, [encoding])	Đọc tệp bằng I/O đồng bộ
writeFile(filename, data, encoding='utf8', [callback])	Ghi tệp vào I/O không đồng bộ
writeFileSync(filename, data, encoding='utf8')	Ghi tệp vào I/O đồng bộ
stat(path, callback)	Kiểm tra thông tin trong tệp

Ghi dữ liệu không đồng bộ có nghĩa là luồng thực thi sẽ tiếp tục ngay lập tức mà không cần chờ quá trình ghi hoàn tất. Điều này giúp tránh tắc nghẽn và tăng hiệu suất, đặc biệt quan trọng trong các ứng dụng yêu cầu xử lý nhiều tác vụ đồng thời.



Ghi dữ liệu đồng bộ có nghĩa là luồng thực thi sẽ dừng lại cho đến khi quá trình ghi hoàn tất. → Điều này làm cho mã dễ đọc hơn nhưng có thể gây ra tình trạng tắc nghẽn (blocking) trong các ứng dụng có tải cao hoặc các tác vụ I/O tốn thời gian.

## Ghi tệp không đồng bộ

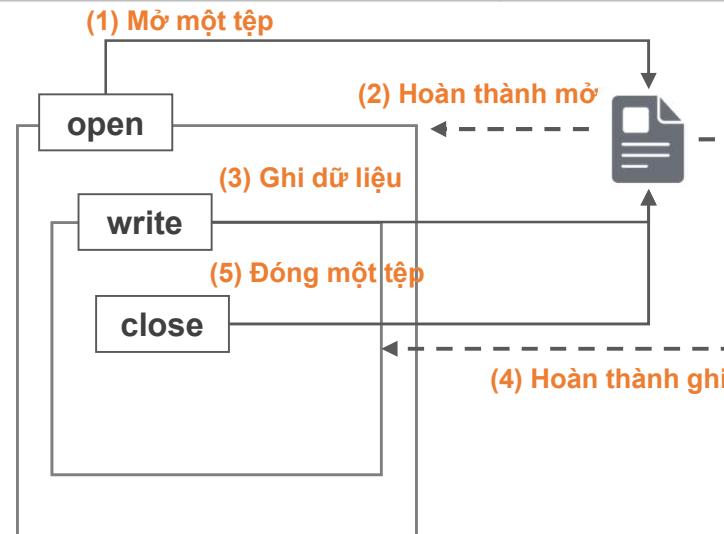
- Truyền hàm callback cho tham số bằng phương thức readFile

```
1  var fs = require('fs');
2
3  fs.writeFile('./output.txt', 'Hello World!', function(err) {
4      if(err) {
5          console.log('Error : ' + err);
6      }
7      console.log('Complete writing data to output.txt file');
8  });
```

## Đọc hoặc ghi trong khi mở và đóng tệp

- Sử dụng các phương thức open, read, write, close, v.v.

Phương thức	Giải thích
open(path, flags, [mode], [callback])	Mở tệp tin
read(fd, buffer, offset, length, position, [callback])	Đọc nội dung tệp của phần được chỉ định
write(fd, buffer, offset, length, position, [callback])	Ghi dữ liệu vào phần được chỉ định của tệp
close(fd, [callback])	Đóng tệp



## Mở tệp trực tiếp và ghi dữ liệu

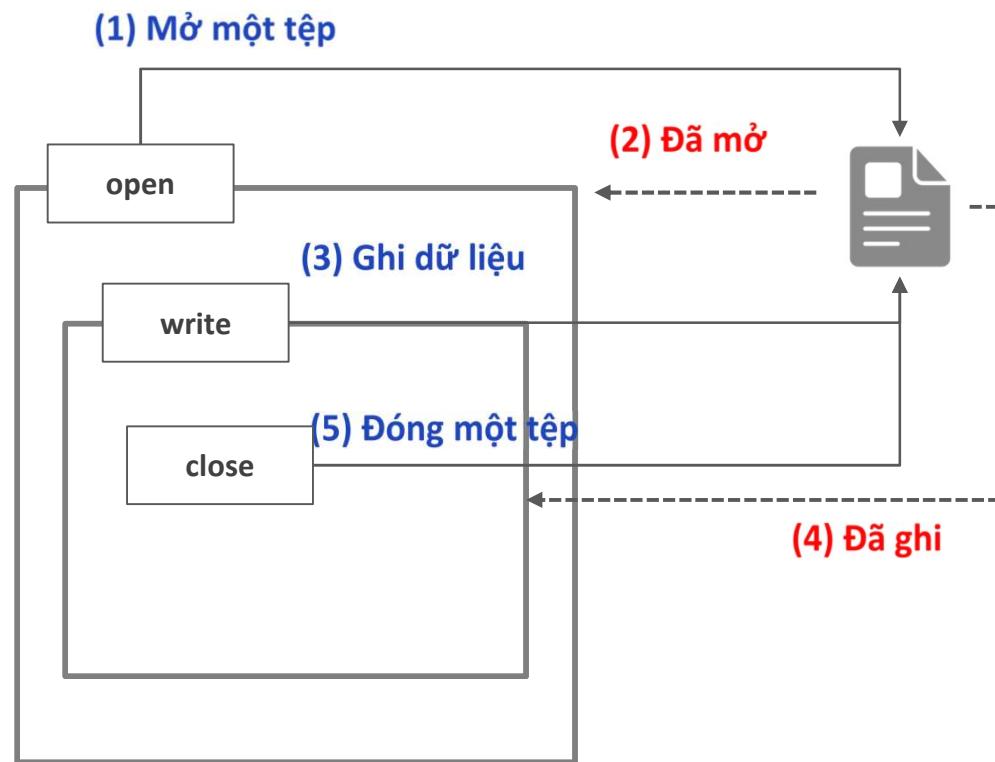
- Mở và ghi

```
1  var fs = require('fs');
2
3  fs.open('./output.txt', 'w', function(err, fd) {
4      if(err) throw err;
5      var buf = new Buffer('Hello!\n');
6      fs.write(fd, buf, 0, buf.length, null,
7          function(err, written, buffer) {
8              if(err) throw err;
9              console.log(err, written, buffer);
10
11             fs.close(fd, function() {
12                 console.log('Open file, write data and close file.');
13             });
14         });
15     });

A1
```

## Luồng xử lý tệp

- Tiến trình mở bằng phương thức open và ghi bằng phương thức write



## Tự mở và đọc tệp

- Mở, đọc và sử dụng bộ đệm

```
1  var fs = require('fs');
2
3  fs.open('./output.txt', 'r', function(err, fd) {
4      if(err) throw err;
5      var buf = new Buffer(10);
6      console.log(' Buffer type : %s', Buffer.isBuffer(buf));
7      fs.read(fd, buf, 0, buf.length, null,
8          function(err, bytesRead, buffer) {
9              if(err) throw err;
10             var inStr = buffer.toString('utf8', 0, bytesRead);
11             console.log(' Data read from file : %s', inStr);
12             console.log(err, bytesRead, buffer);
13             fs.close(fd, function() {
14                 console.log('Open the output.txt file, ' +
15                     'read it, complete.');
16             });
17         });
18     });

});
```

## Kiểm tra thông tin

- Kiểm tra thông tin tệp bằng stat
- Cung cấp các hàm như isFile(), isDirectory() và isFIFO()

```
1 var fs = require('fs');
2
3 fs.stat('./stat.js', function(err, stats){
4     if (err) throw err;
5
6     console.log(stats);
7     console.log('isFile() : %s', stats.isFile());
8 });
9
```

- Màn hình kết quả.

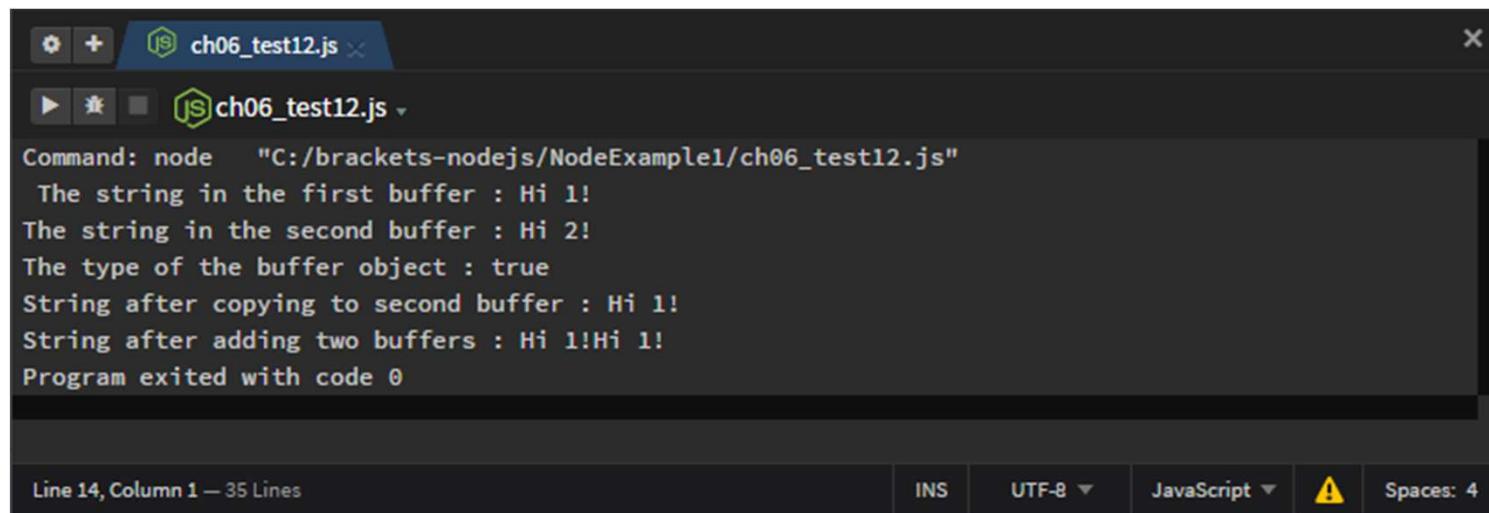
```
{ dev: -55011900,  
mode: 33206,  
nlink: 1,  
uid: 0,  
gid: 0,  
rdev: 0,  
blksize: undefined,  
ino: 56013520365545000,  
size: 168,  
blocks: undefined,  
atime: Sun May 29 2019 20:50:49 GMT+0900 (Korea Standard Time),  
mtime: Sun May 29 2019 20:50:49 GMT+0900 (Korea Standard Time),  
ctime: Sun May 29 2019 20:50:49 GMT+0900 (Korea Standard Time),  
birthtime: Sun May 29 2019 20:49:59 GMT+0900 (Korea Standard Time) }  
isFile() : true
```

## Sử dụng bộ đệm

- Tạo bộ đệm mới bằng phương thức 'new' và sử dụng các phương thức như 'Buffer.isBuffer ()' và 'Buffer.concat ()'.

```
1  var output = ' Hi 1!';
2  var buffer1 = new Buffer(10);
3  var len = buffer1.write(output, 'utf8');
4  console.log('The string in the first buffer : %s',
5    buffer1.toString());
6
7  var buffer2 = new Buffer(' Hi 2!', 'utf8');
8  console.log('The string in the second buffer : %s',
9    buffer2.toString());
10
11 console.log('The type of the buffer object : %s',
12   Buffer.isBuffer(buffer1));
13
14 var byteLen = Buffer.byteLength(output);
15 var str1 = buffer1.toString('utf8', 0, byteLen);
16 var str2 = buffer2.toString('utf8');
17
18 buffer1.copy(buffer2, 0, 0, len);
19 console.log('The string after copying to second buffer : %s',
20   buffer2.toString('utf8'));
```

- Tạo bộ đệm mới bằng phương thức 'new' và sử dụng các phương thức như 'Buffer.isBuffer ()' và 'Buffer.concat ()'.



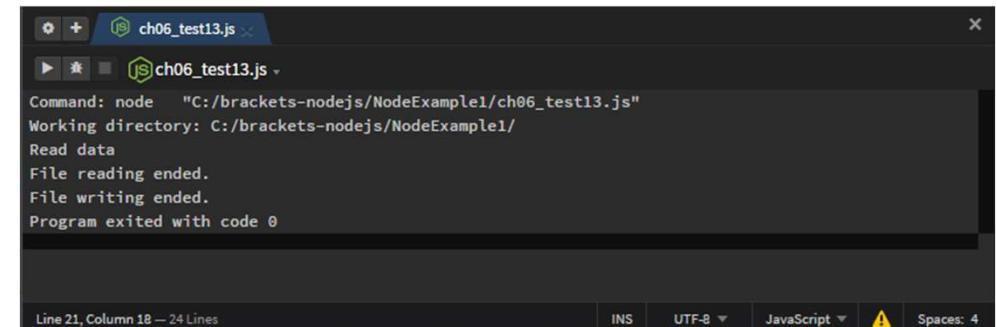
```
Command: node "C:/brackets-nodejs/NodeExample1/ch06_test12.js"
The string in the first buffer : Hi 1!
The string in the second buffer : Hi 2!
The type of the buffer object : true
String after copying to second buffer : Hi 1!
String after adding two buffers : Hi 1!Hi 1!
Program exited with code 0
```

Line 14, Column 1 — 35 Lines      INS      UTF-8 ▾      JavaScript ▾      ⚠      Spaces: 4

## Đọc các tệp và ghi vào luồng

- Mở để đọc bằng 'createReadStream', Mở để ghi bằng 'createWriteStream'

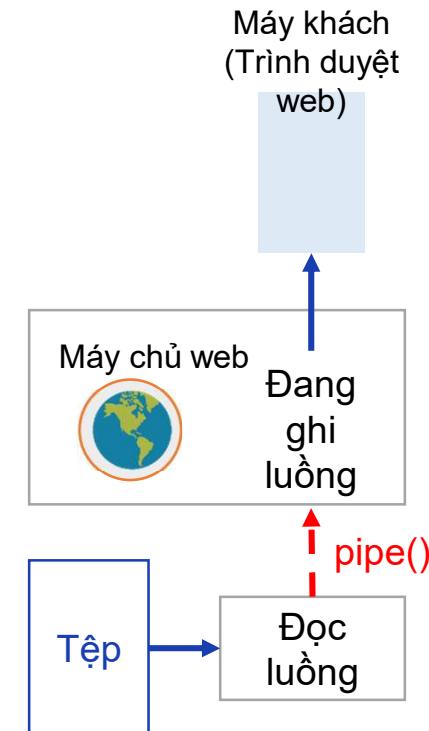
```
1  var fs = require('fs');
2
3  var infile = fs.createReadStream(
4      './output.txt', {flags: 'r'});
5  var outfile = fs.createWriteStream(
6      './output2.txt', {flags: 'w'});
7
8  infile.on('data', function(data) {
9      console.log(' Read data ', data);
10     outfile.write(data);
11 });
12
13 infile.on('end', function() {
14     console.log('File read end.');
15     outfile.end(function() {
16         console.log(' File read end.');
17     })
18});
```



## Đọc và phản hồi với nội dung tệp được yêu cầu từ mô-đun “http”

- Đọc tệp theo luồng và kết nối dữ liệu với một 'pipe' (đường ống).

```
1  var fs = require('fs');
2  var http = require('http');
3
4  var server = http.createServer(function(req, res) {
5      // Read the file and associate it with the
6      // response stream as 'pipe ()'.
7      var instream = fs.createReadStream('./output.txt');
8      instream.pipe(res);
9  });
10
11 server.listen(7001, '127.0.0.1');
```



## Tạo và xóa thư mục mới

- Tạo thư mục bằng phương thức 'mkdir' và xóa thư mục bằng phương thức "rmdir".

```
1 var fs = require('fs');
2 fs.mkdir('./docs', '0666', function(err) {
3     if(err) throw err;
4     console.log(' I have created a new docs folder.');
5
6     fs.rmdir('./docs', function(err) {
7         if(err) throw err;
8         console.log('docs folder deleted.');
9     });
10});
```

```
ch06_test14.js
ch06_test14.js

Command: node "C:/brackets-nodejs/NodeExample1/ch06_test14.js"
Working directory: C:/brackets-nodejs/NodeExample1/
New docs folder created.
Docs folder deleted.
Program exited with code 0

Line 17, Column 8 — 20 Lines
```

Bài 2.

## Lập trình Node.js cơ bản

- 2.1. Chuỗi địa chỉ và tham số truy vấn
- 2.2. Tìm hiểu về Sự kiện
- 2.3. Sử dụng Tệp
- 2.4. Duy trì tệp Log

## Ghi log bằng mô-đun “winston”

- Một vài mô-đun có thể giúp bạn ghi log.
- Bạn nên ghi log theo ngày.
- Có nhiều cài đặt khác có thể thực thi được trong mô-đun 'winston'.

```
1  var winston = require('winston');      // Log processing module
2  var winstonDaily = require('winston-daily-rotate-file');
3  // Daily log processing module
4  var moment = require('moment');        // Time processing module
5
6  function timeStampFormat() {
7      return moment().format('YYYY-MM-DD HH:mm:ss.SSS ZZ');
8      // '2019-05-01 20:14:28.500 +0900'
9  };
```

## Cấu hình mô-đun 'winston'

- Bạn có thể thực hiện các cài đặt khác nhau, bao gồm cả vị trí các tệp log không thay đổi.

```

11  var logger = new (winston.Logger)({
12    transports: [
13      new (winstonDaily)({
14        name: 'info-file',
15        filename: './log/server',
16        datePattern: '_yyyy-MM-dd.log',
17        colorize: false,
18        maxsize: 50000000,
19        maxFiles: 1000,
20        level: 'info',
21        showLevel: true,
22        json: false,
23        timestamp: timeStampFormat
24      }),
25      new (winston.transports.Console)({
26        name: 'debug-console',
27        colorize: true,
28        level: 'debug',
29        showLevel: true,
30        json: false,
31        timestamp: timeStampFormat
32    })
33  ],
34 });

```

```

34   exceptionHandlers: [
35     new (winstonDaily)({
36       name: 'exception-file',
37       filename: './log/exception',
38       datePattern: '_yyyy-MM-dd.log',
39       colorize: false,
40       maxsize: 50000000,
41       maxFiles: 1000,
42       level: 'error',
43       showLevel: true,
44       json: false,
45       timestamp: timeStampFormat
46     }),
47     new (winston.transports.Console)({
48       name: 'exception-console',
49       colorize: true,
50       level: 'debug',
51       showLevel: true,
52       json: false,
53       timestamp: timeStampFormat
54     })
55   ]
56 });

```

## Sử dụng mô-đun winston

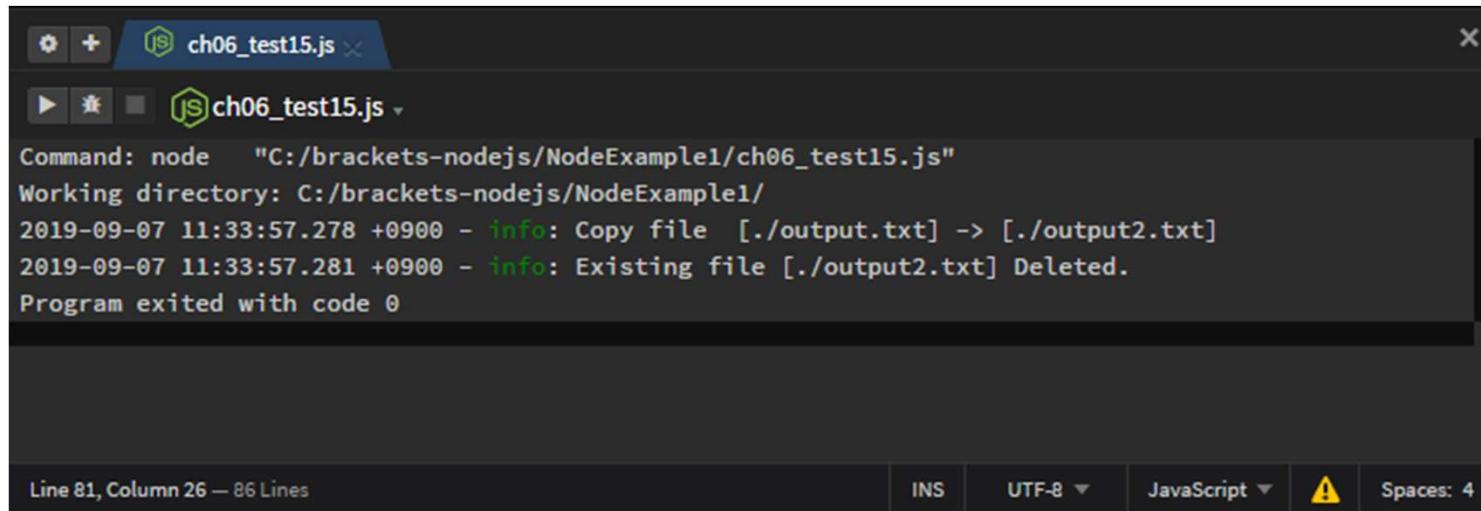
- Trong một đoạn mã sao chép tệp, ghi log bằng phương thức 'logger.info ()'.

```
58  var fs = require('fs');
59
60  var inname = './output.txt';
61  var outname = './output2.txt';
62
63  if(fs.existsSync(outname, function(exists) {
64      if(exists) {
65          fs.unlink(outname, function(err) {
66              if(err) throw err;
67              logger.info(' Existing file [' + outname + '] Deleted.');
68          });
69      }
70
71      var infile = fs.createReadStream(inname, {flags: 'r'});
72      var outfile = fs.createWriteStream(outname, {flags: 'w'});
73
74      infile.pipe(outfile);
75      logger.info(' Copy file [' + inname + '] -> [' + outname + ']');
76  }));

```

- Cài đặt mô-đun bên ngoài

```
% npm install winston --save  
% npm install winston-daily-rotate-file --save  
% npm install moment --save
```



The screenshot shows the Brackets IDE interface with a code editor window titled "ch06\_test15.js". The code in the editor is as follows:

```
Command: node "C:/brackets-nodejs/NodeExample1/ch06_test15.js"  
Working directory: C:/brackets-nodejs/NodeExample1/  
2019-09-07 11:33:57.278 +0900 - info: Copy file [./output.txt] -> [./output2.txt]  
2019-09-07 11:33:57.281 +0900 - info: Existing file [./output2.txt] Deleted.  
Program exited with code 0
```

Below the code editor, the status bar displays "Line 81, Column 26 — 86 Lines". At the bottom of the interface, there are tabs for "INS", "UTF-8", "JavaScript", and "Spaces: 4".

Bài 3.

# Raspberry Pi với Node-RED

- 3.1. Cài đặt ban đầu
- 3.2. Điều khiển đèn LED bằng Node-RED
- 3.3. Phát hiện truy cập bằng Node-RED

## Cài đặt node.js trên Raspberry Pi

### Cài đặt NodeJS mới nhất

- Cài đặt 'node.js' trên 'Raspberry Pi'

```
$ wget -O - https://raw.githubusercontent.com/meech-ward/NodeJs-Raspberry-Pi/master/Install-Node.sh | sudo bash;
```

- Kiểm tra phiên bản Node.js trên Raspberry Pi.

```
$ node -v
```

```
pi@raspberrypi:~ $ node -v
v16.10.0
```

- Bạn có thể sử dụng 1 bài thử nghiệm với câu lệnh đơn giản nếu cài đặt thành công.

```
pi@raspberrypi:~ $ node
Welcome to Node.js v16.10.0.
Type ".help" for more information.
> var greeting = "Hello"
undefined
> console.log(greeting)
Hello
undefined
>
pi@raspberrypi:~ $
```

## Cài đặt Node-RED

- Trước khi bắt đầu Cài đặt, chúng ta nên cập nhật các thư viện hiện có.
  - Luôn cập nhật thư viện bằng cách nhập liên tiếp các lệnh sau:

```
$ sudo apt update  
$ sudo apt upgrade
```

- Tải xuống Node-RED. (mất tương đối thời gian.)
  - Nhập y cho hai câu hỏi ở giữa.

```
$ bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installer/master/deb/update-nodejs-and-nodered)
```

- Quy trình cài đặt Node-RED

```
Running Node-RED install for user pi at /home/pi on raspbian

This can take 20-30 minutes on the slower Pi versions - please wait.

Stop Node-RED               ✓
Remove old version of Node-RED   ✓
Node option not specified      : --node12 or --node14
Leave existing Node.js          : v16.10.0   Npm 7.24.0
Clean npm cache                -
Install Node-RED core          ✓  2.0.6
Move global nodes to local
Npm rebuild existing nodes
Install extra Pi nodes
Add shortcut commands
Update systemd script

Any errors will be logged to    /var/log/nodered-install.log
```

## Khởi động Node-RED

- Khởi động Node-RED
  - Thực thi với Node JS thông qua lệnh thực thi sau:

```
$ node-red-start
```

```
pi@raspberrypi:~ $ node-red-start

Start Node-RED

Once Node-RED has started, point a browser at http://192.168.0.8:1880
On Pi Node-RED works better with the Firefox or Chrome browser

Use  node-red-stop          to stop Node-RED
Use  node-red-start         to start Node-RED again
Use  node-red-log           to view the recent log output
Use  sudo systemctl enable nodered.service  to autostart Node-RED at every boot
Use  sudo systemctl disable nodered.service  to disable autostart on boot

To find more nodes and example flows - go to http://flows.nodered.org

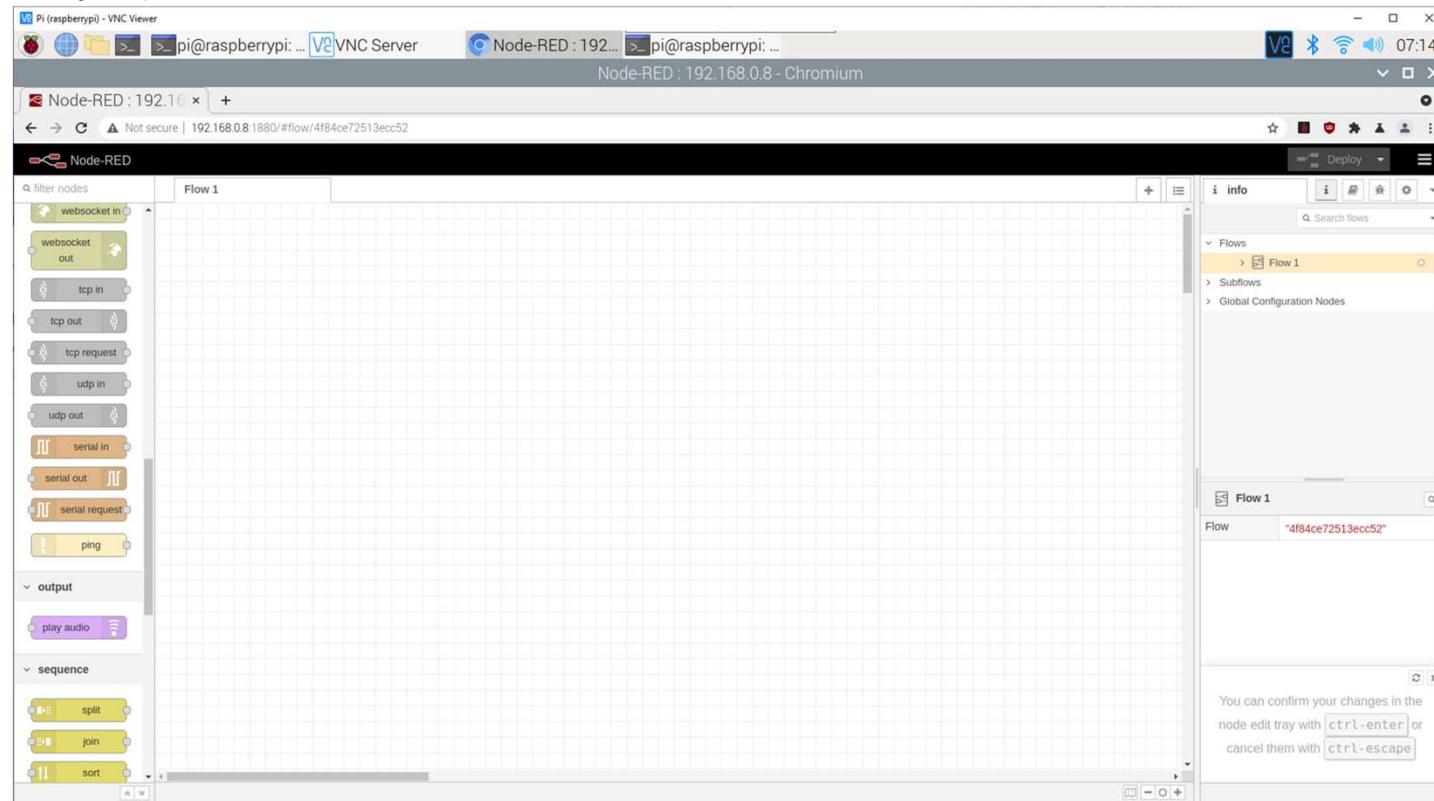
Starting as a systemd service.
(node:7238) [DEP0128] DeprecationWarning: Invalid 'main' field in '/usr/lib/node_modules/node-red/node_modules/@n
ode-red/editor-client/package.json' of './lib/index.js'. Please either fix that or report it to the module author
(Use `node --trace-deprecation ...` to show where the warning was created)
7 Oct 07:07:09 - [info]
Welcome to Node-RED
=====
7 Oct 07:07:09 - [info] Node-RED version: v2.0.6
7 Oct 07:07:09 - [info] Node.js version: v16.10.0
7 Oct 07:07:09 - [info] Linux 5.10.52-v7+ arm LE
7 Oct 07:07:14 - [info] Loading palette nodes
```

### 3.1. Cài đặt ban đầu

Bài 03

- Khởi động Node-RED

- Truy cập trang web `http://{Raspberry Pi's IP Address}:1880` trên trình duyệt của bạn.
- Trình duyệt NC và trình duyệt Windows đều có sẵn, nhưng ở đây, hãy sử dụng trình duyệt trong chính Raspberry Pi để truy cập.



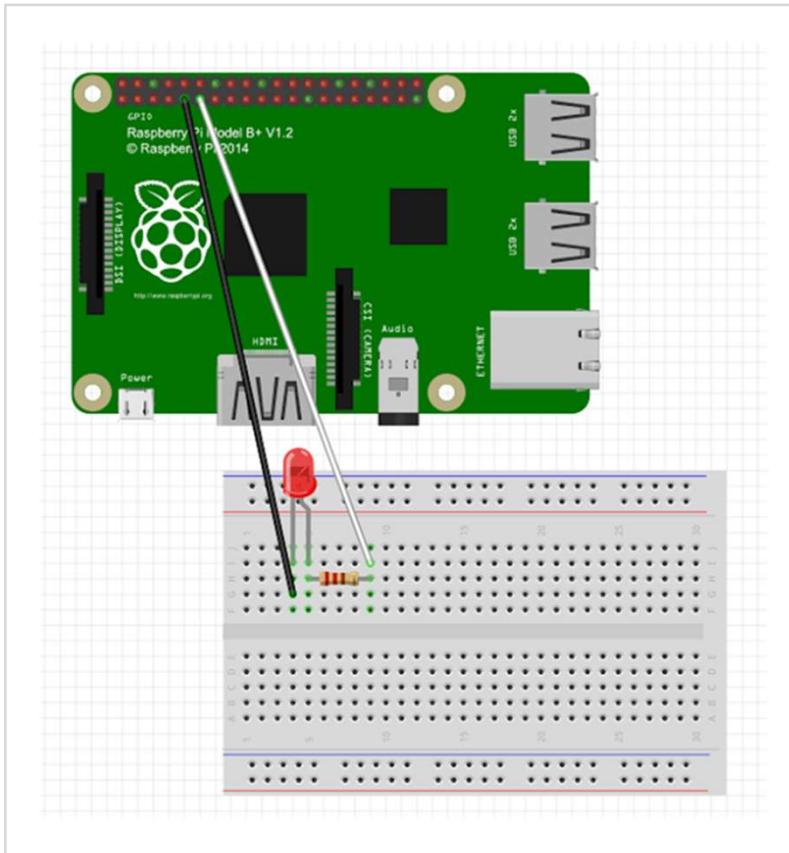
Bài 3.

# Raspberry Pi với Node-RED

- 3.1. Cài đặt ban đầu
- 3.2. Điều khiển đèn LED bằng Node-RED
- 3.3. Phát hiện truy cập bằng Node-RED

## Điều khiển đèn LED bằng Node-RED

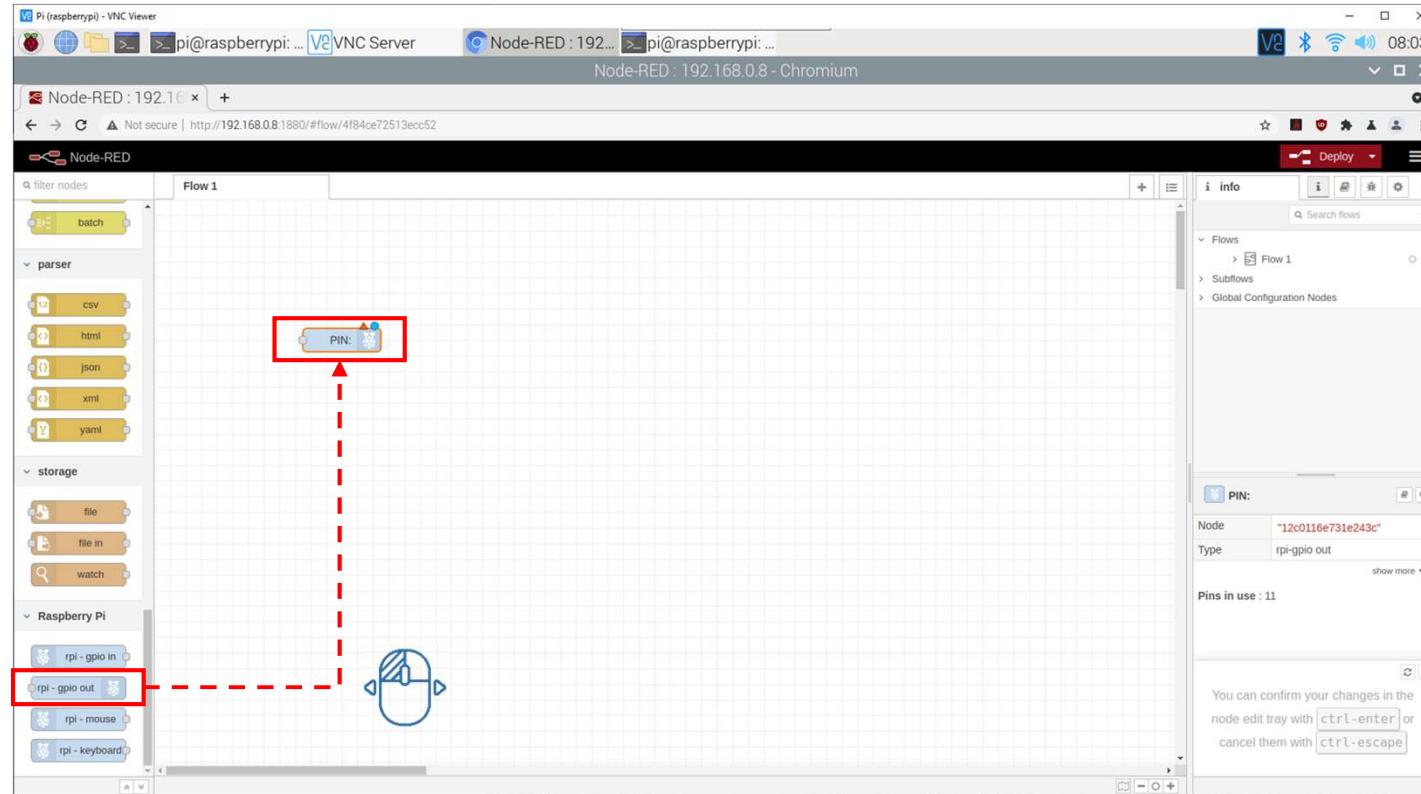
- Ví dụ điều khiển đèn LED đơn giản cho việc sử dụng Node-RED



- ▶ Một ví dụ về điều khiển đèn LED bằng cách gửi tín hiệu đến GPIO bằng Node-RED.
- ▶ Sơ đồ mạch
  - Raspberry Pi GPIO 17 – Resistor – LED Anode(+)
  - LED Cathode(-) - GND

## Các chân của GPIO

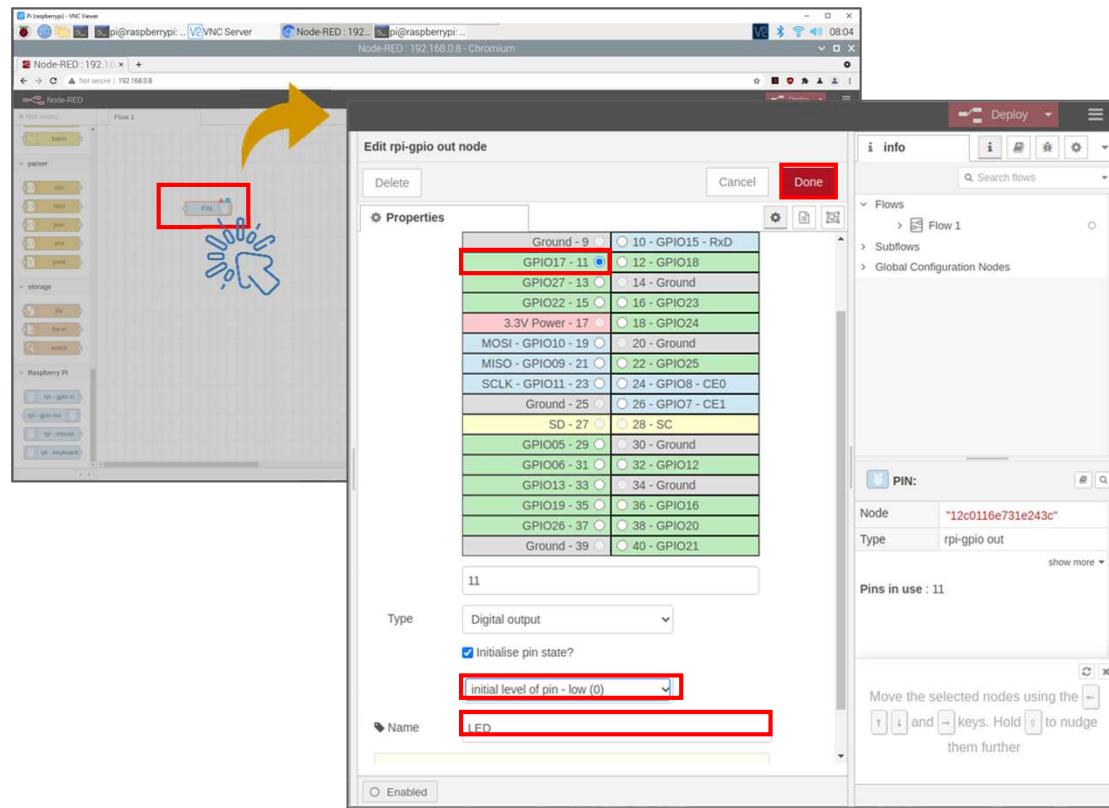
- Thêm chân vào Node-RED
  - Tìm rpi-gpio-out trong danh mục Raspberry Pi và kéo nó vào lưới để thêm chân.



### 3.2. Điều khiển đèn LED bằng Node-RED

Bài 03

- Chỉnh sửa thuộc tính chân
  - ▶ Nhấp đúp vào chân đã tạo để chỉnh sửa Thuộc tính.

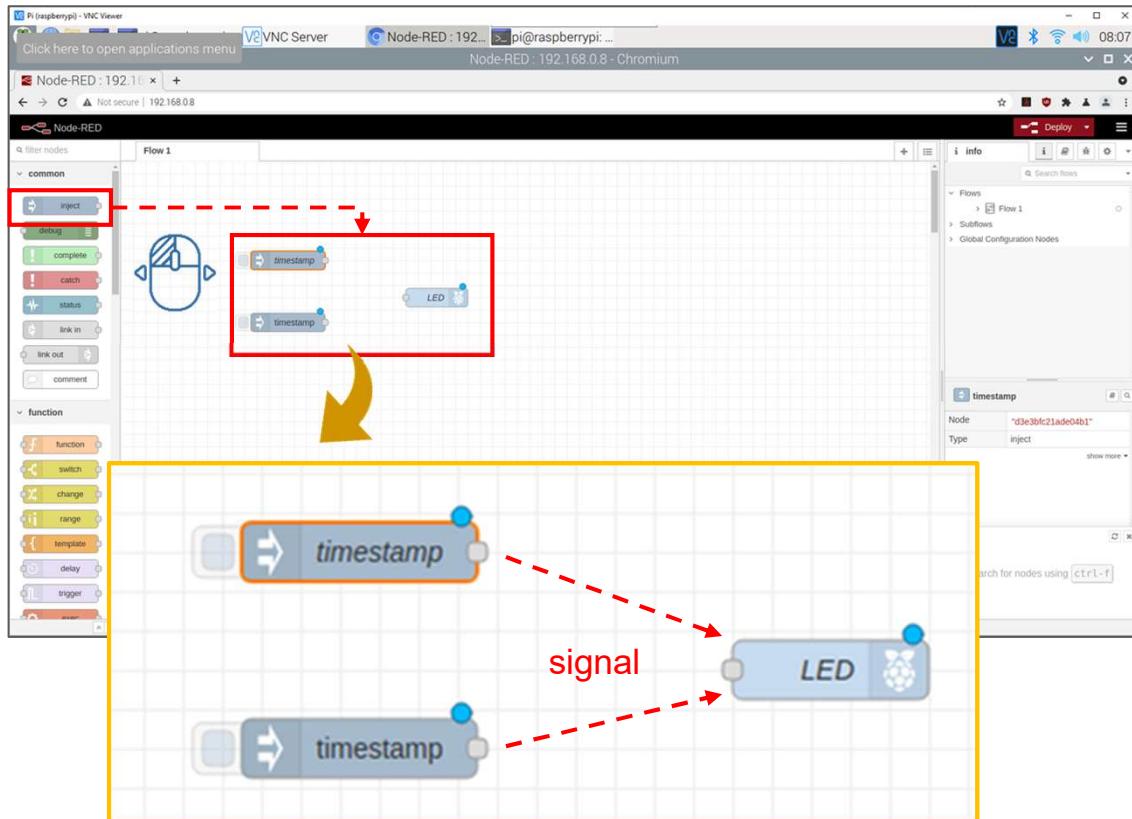


- ▶ Chọn GPIO 17, đặt Trạng thái khởi tạo chân thành trạng thái thấp và thay đổi Tên thành LED.
- ▶ Sau khi chỉnh sửa, nhấp vào nút Done màu đỏ.

### 3.2. Điều khiển đèn LED bằng Node-RED

Bài 03

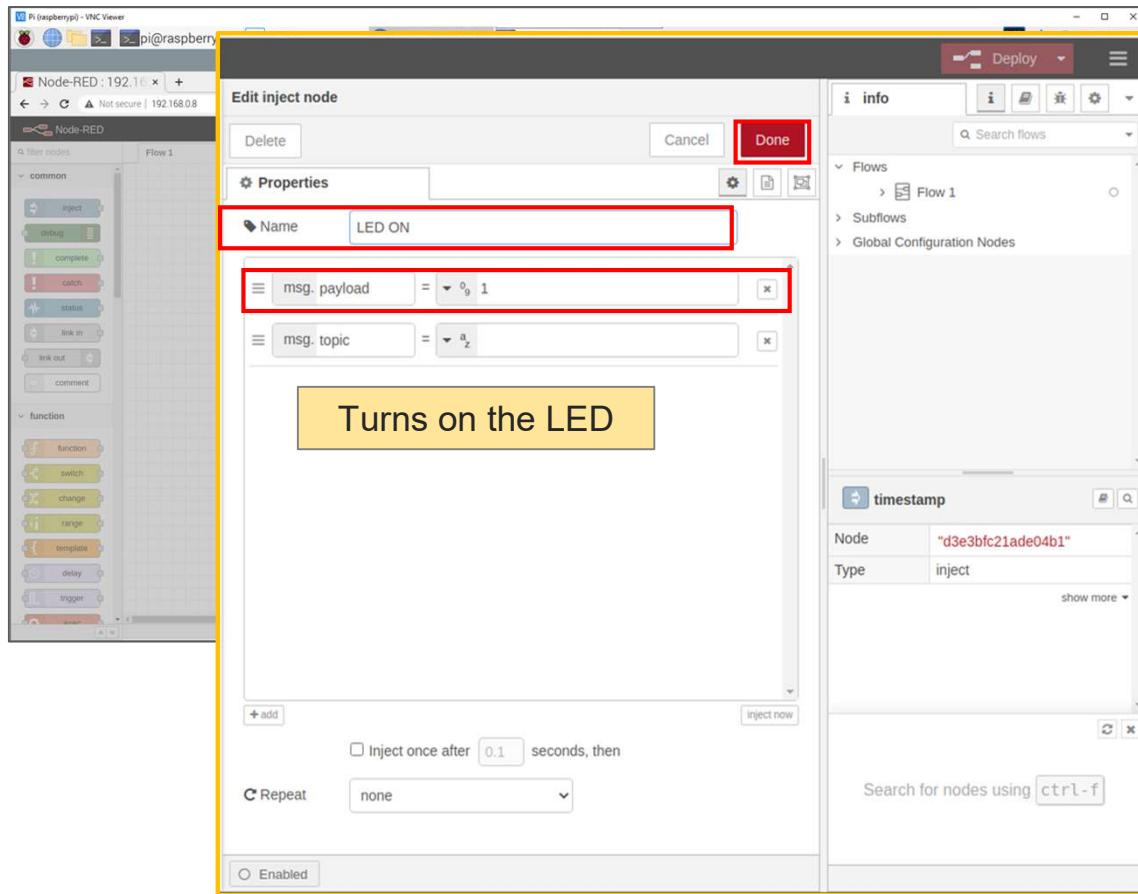
- Thêm chỉ lệnh LED ON, LED OFF



- ▶ Kéo chỉ lệnh trong danh mục chung và tạo ra hai chỉ lệnh.
- ▶ Hai chỉ lệnh được tạo ra sẽ gửi tín hiệu đến đèn LED.

## Chỉnh sửa thuộc tính của chỉ lệnh

- Chỉnh sửa thuộc tính của chỉ lệnh

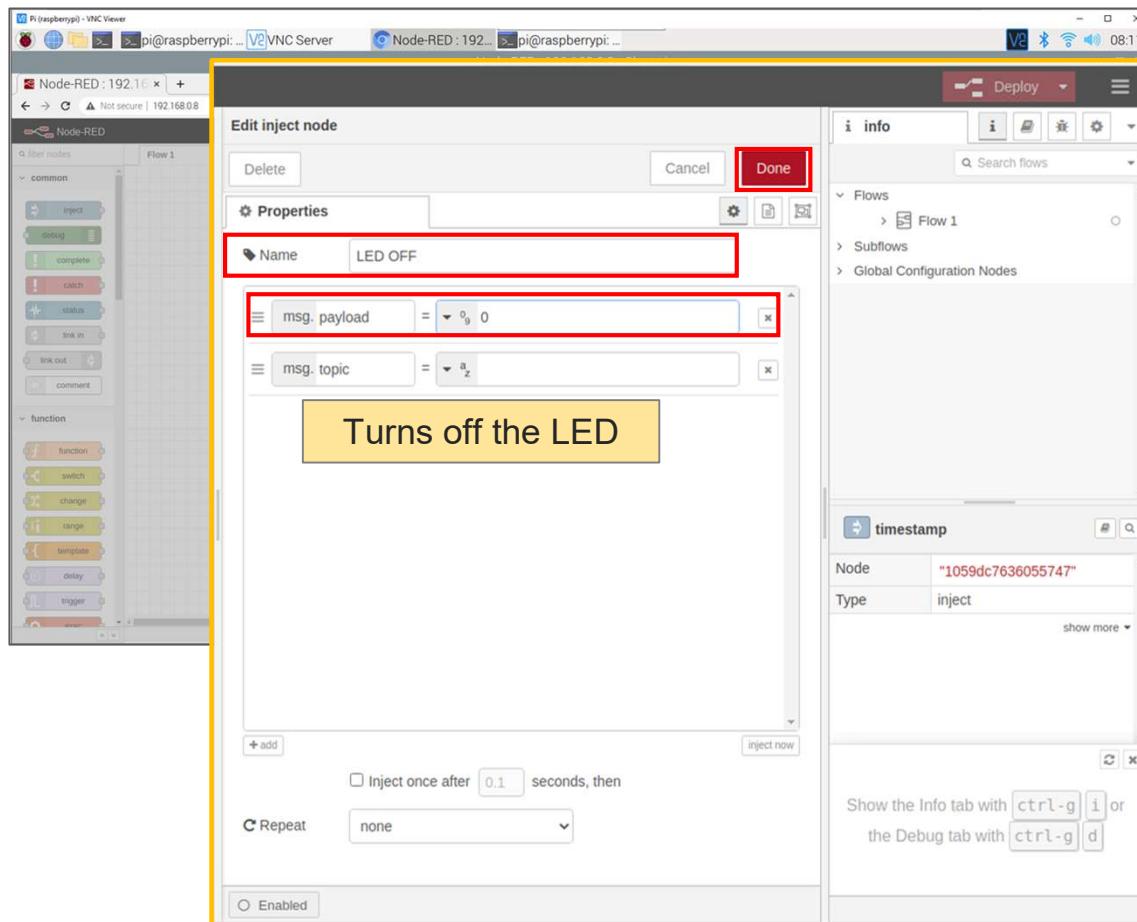


- Sửa đổi các thuộc tính của chỉ lệnh đã tạo.
- Trong trường có một chỉ lệnh, tên được đổi thành LED ON và msg.payload được đổi thành số 1. Chỉ lệnh LED ON này sẽ bật đèn LED.
- Sau khi chỉnh sửa, nhấp vào nút “Done” màu đỏ.

### 3.2. Điều khiển đèn LED bằng Node-RED

Bài 03

- Chỉnh sửa thuộc tính chỉ lệnh



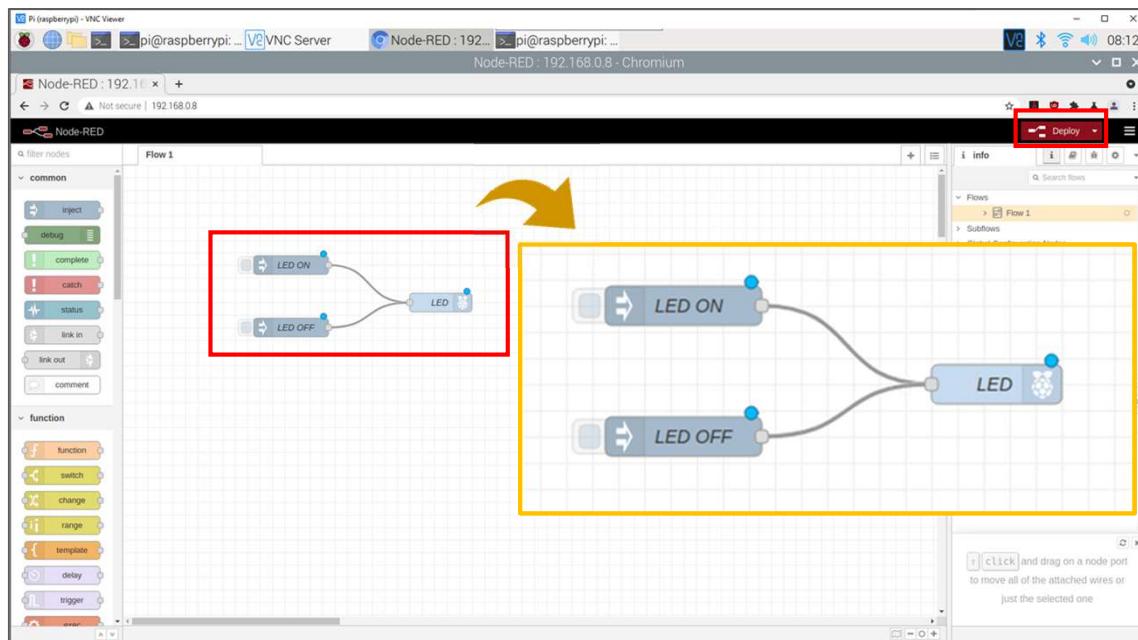
- Trong trường hợp chỉ lệnh khác, tên được thay đổi thành LED ON và msg.payload được thay đổi thành số 0. Chỉ lệnh LED OFF này sẽ tắt đèn LED.
- Sau khi chỉnh sửa, nhấp vào nút “Done” màu đỏ.

### 3.2. Điều khiển đèn LED bằng Node-RED

Bài 03

## Kết nối và triển khai

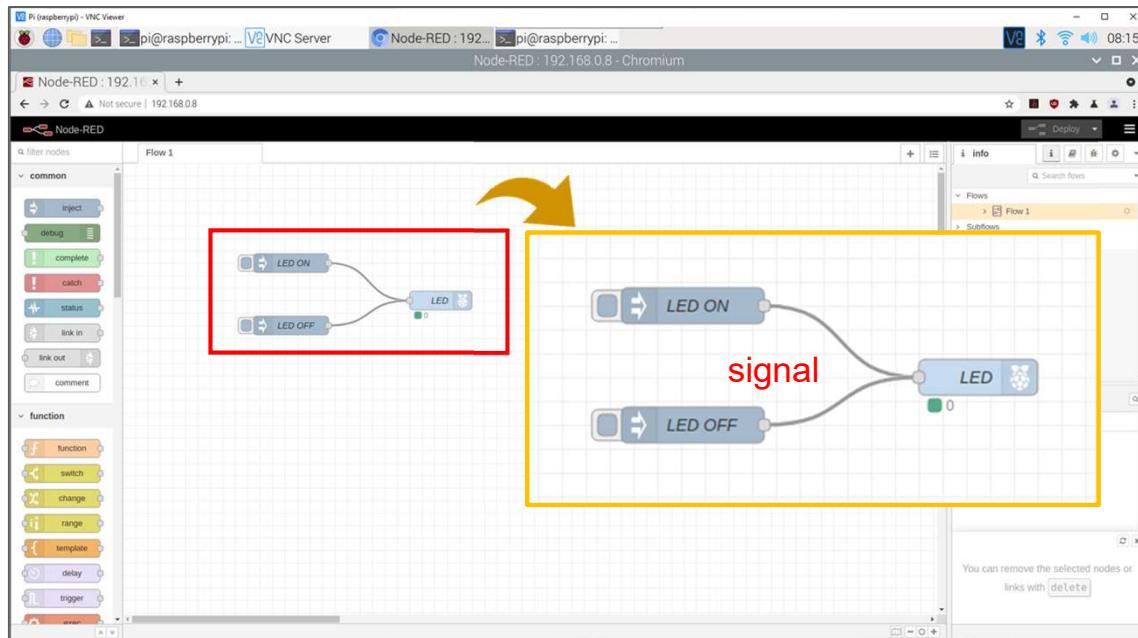
- Chỉnh sửa thuộc tính chân



- Kết nối để chỉ lệnh được tạo có thể gửi tín hiệu tới **GPIO**.
- Sử dụng con trỏ **chuột** để kết nối hình vuông màu xám ở bên phải của nút “LED ON” và “LED OFF” với hình vuông màu xám ở bên trái của LED.
- Vì tất cả các cài đặt đã hoàn tất, hãy nhấp vào nút “Deploy” để chạy Flow.

## Kiểm tra

- Ví dụ kiểm tra



- Sau khi nhấn nút “Deploy”, nếu bạn nhấn vào ô vuông ở bên trái nút “LED ON” và “LED OFF”, chỉ lệnh sẽ hoạt động và gửi tín hiệu đến đèn LED.
- Nhập và kiểm tra xem đèn LED có được điều khiển hay không.

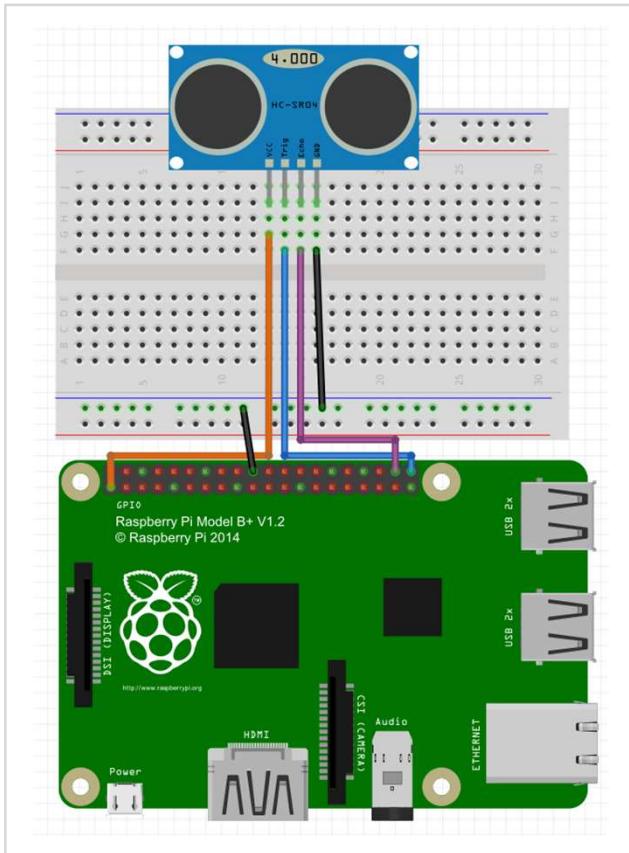
Bài 3.

# Raspberry Pi với Node-RED

- 3.1. Cài đặt ban đầu
- 3.2. Điều khiển đèn LED bằng Node-RED
- 3.3. Phát hiện truy cập bằng Node-RED

## Phát hiện truy cập bằng Node-RED

- Phát hiện truy cập bằng Node-RED

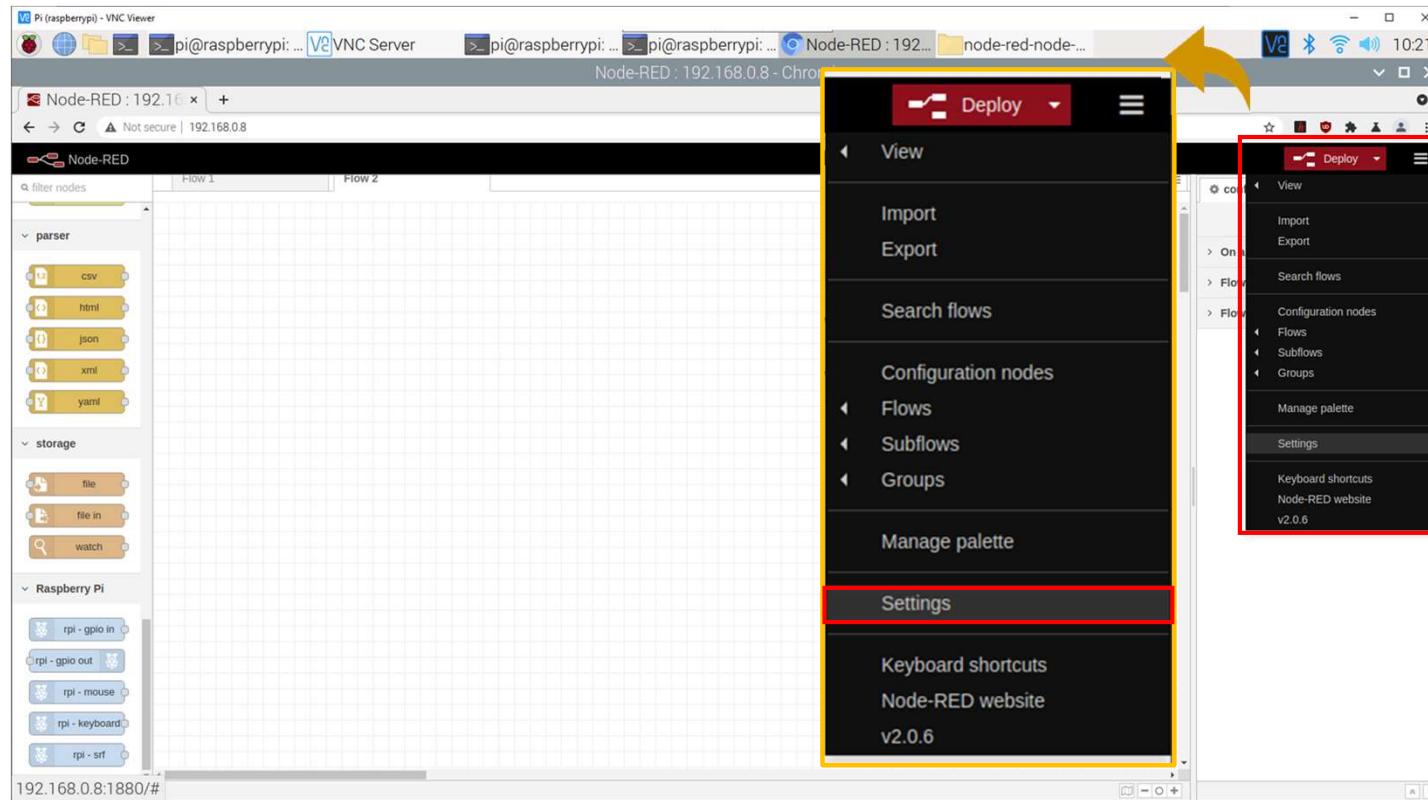


- ▶ Một ví dụ về trực quan hóa khoảng cách đo được bằng cách đo khoảng cách bằng Cảm biến siêu âm với Node-RED.
- ▶ Sơ đồ mạch
  - HC-SR04 VCC – Raspberry Pi 3.3v
  - HC-SR04 GND – GNDs
  - HC-SR04 Trig – Raspberry Pi GPIO 21
  - HC-SR04 Echo – Raspberry Pi GPIO 20
  - Raspberry Pi GND – GNDs

## Cài đặt

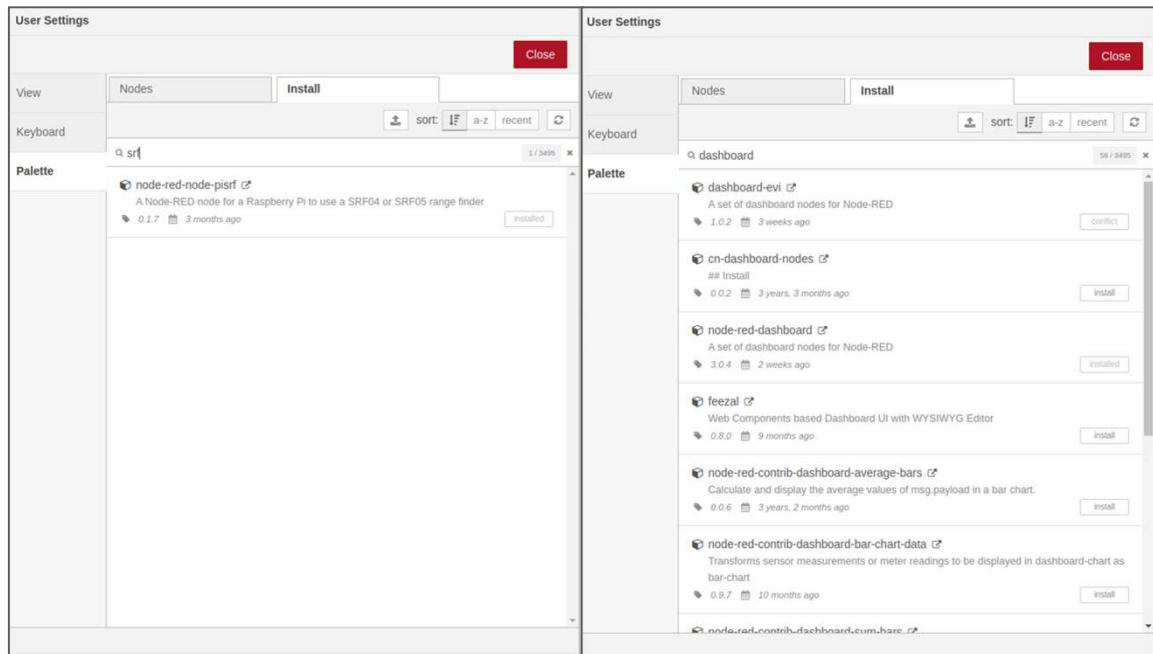
- Cài đặt Nodes

‣ Để cài đặt Node Cảm biến Siêu âm và Node Social, hãy nhấn Menu ở bên phải màn hình và chọn vào Cài đặt.



## Cài đặt Nodes

- Cài đặt node-red-node-pisrf



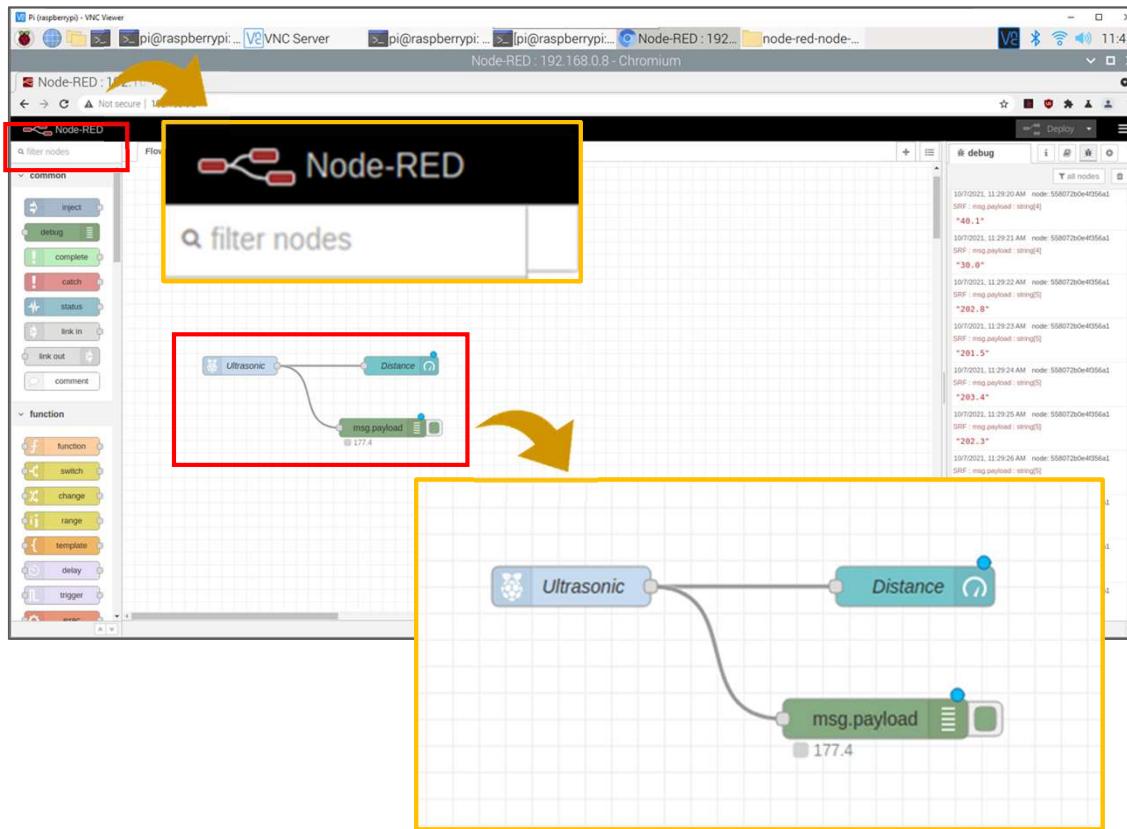
- ▶ Tìm srf trong tab Palette và cài đặt iperf.
- ▶ Lắp đặt cảm biến siêu âm để sử dụng trong Node-RED.
- ▶ Tìm kiếm bảng điều khiển và cài đặt node-red-dashboard.
- ▶ Cài đặt để sử dụng nút bảng điều khiển của Node-RED giúp đơn giản việc trực quan hóa.

### 3.3. Phát hiện truy cập bằng Node-RED

Bài 03

## Thêm Nodes

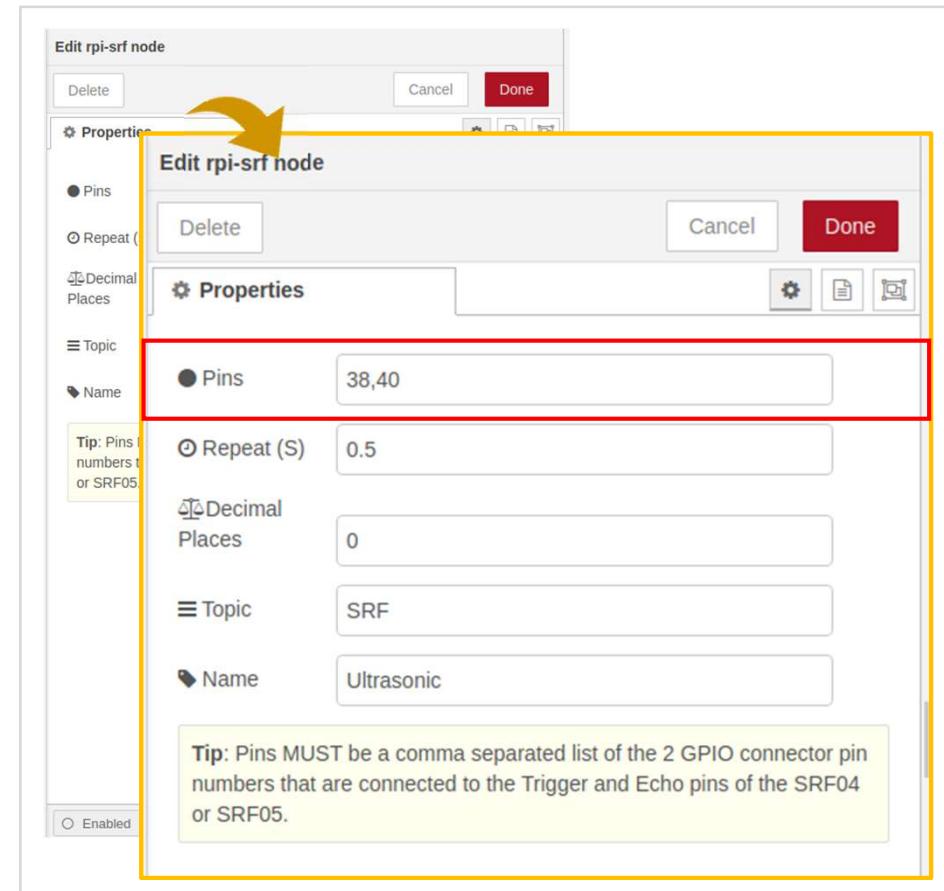
- Ví dụ bài kiểm tra



- ▶ Nhập srf, đo và gỡ lỗi trong trường nhập tìm kiếm ở trên cùng bên trái nơi ghi các nút bộ lọc, tạo từng nút một và kết nối chúng theo cách đang được thể hiện trong hình bên trái.
- ▶ Do các màu được tách rời nhau nên chỉ tiến hành sắp xếp và kết nối, còn việc cài đặt chi tiết được tiếp tục.

## Chỉnh sửa Nodes

Chỉnh sửa thuộc tính của các node.



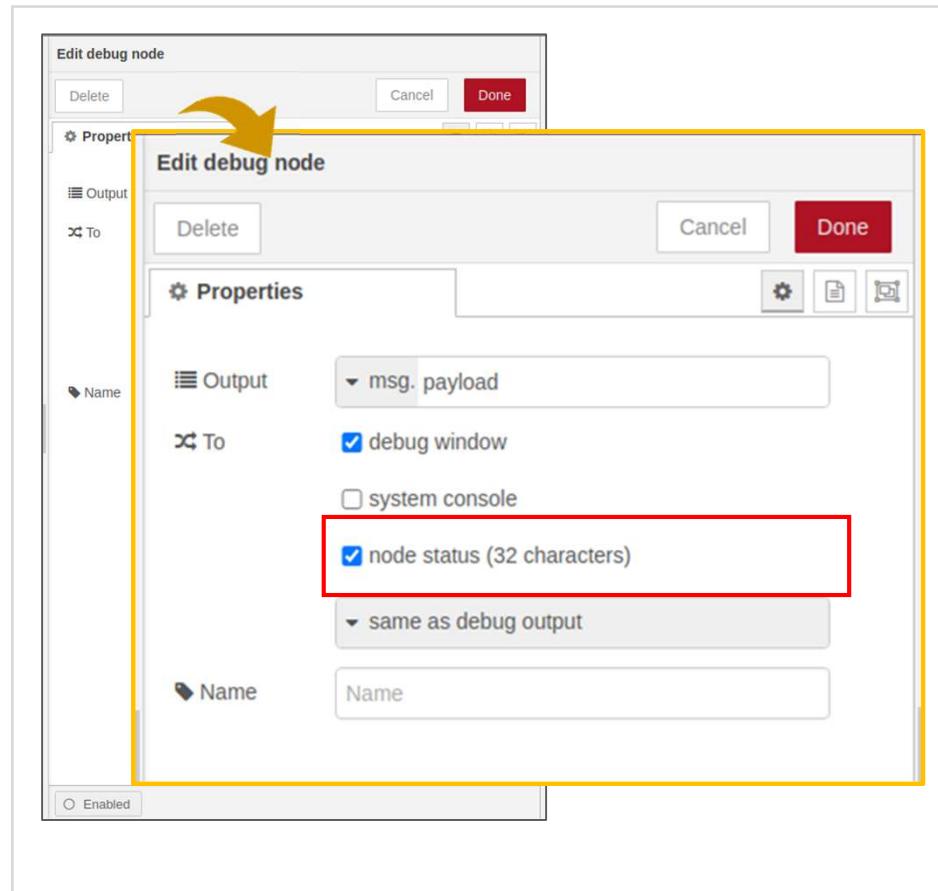
### rpi-srf node

- Một điều cần lưu ý về các chân là bạn phải nhập Trig và Echo theo thứ tự và viết số của chính chân đó, không phải số chân của gpio.
- Trig và Ccho lần lượt được kết nối với GPIO 21 và GPIO 20 và số lượng các chân này là 38 và 40.
- Lặp lại: lặp lại chu kỳ
- Tên: Siêu âm

### 3.3. Phát hiện truy cập bằng Node-RED

Bài 03

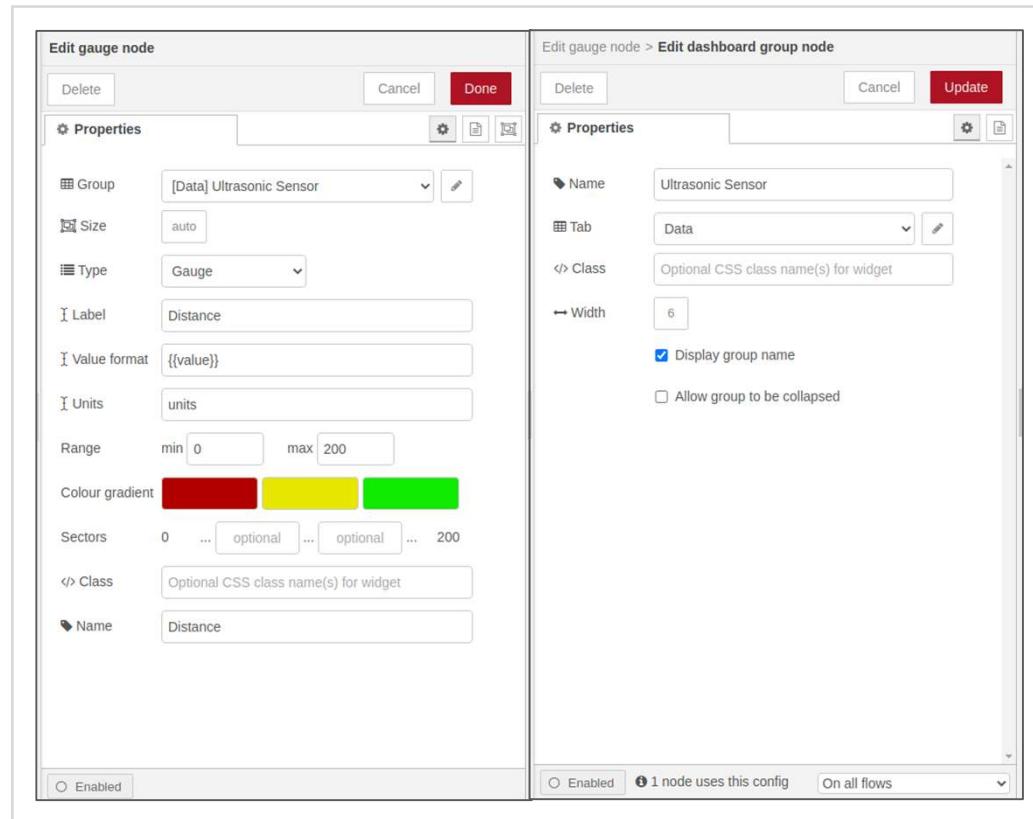
- Chỉnh sửa thuộc tính của node.



- Gỡ lỗi node
  - Kiểm tra trạng thái của node.

### 3.3. Phát hiện truy cập bằng Node-RED

- Chỉnh sửa thuộc tính của node.

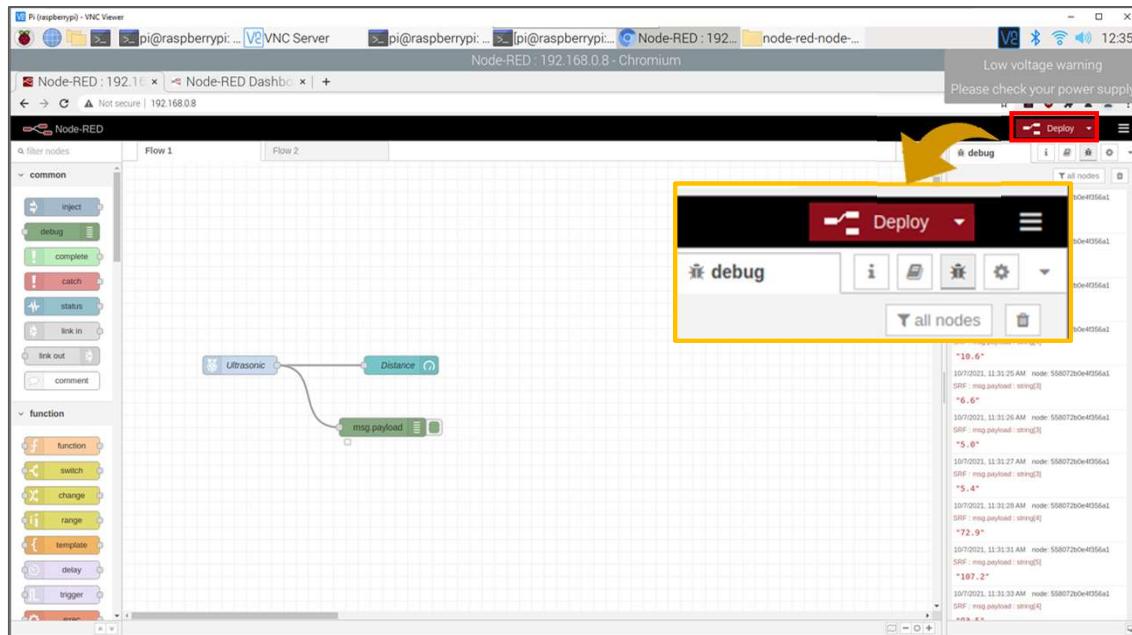


#### Đo node

- Nhóm: Chính sửa thuộc tính của các node thông qua tab mở rộng và chỉnh sửa node của nhóm bảng điều khiển sao cho giống trạng thái như minh họa bên phải.
- Nhãn : Khoảng cách
- Phạm vi: 0 – 200
- Thay đổi màu sắc tự do
- Tên: Khoảng cách

## Triển khai Nodes

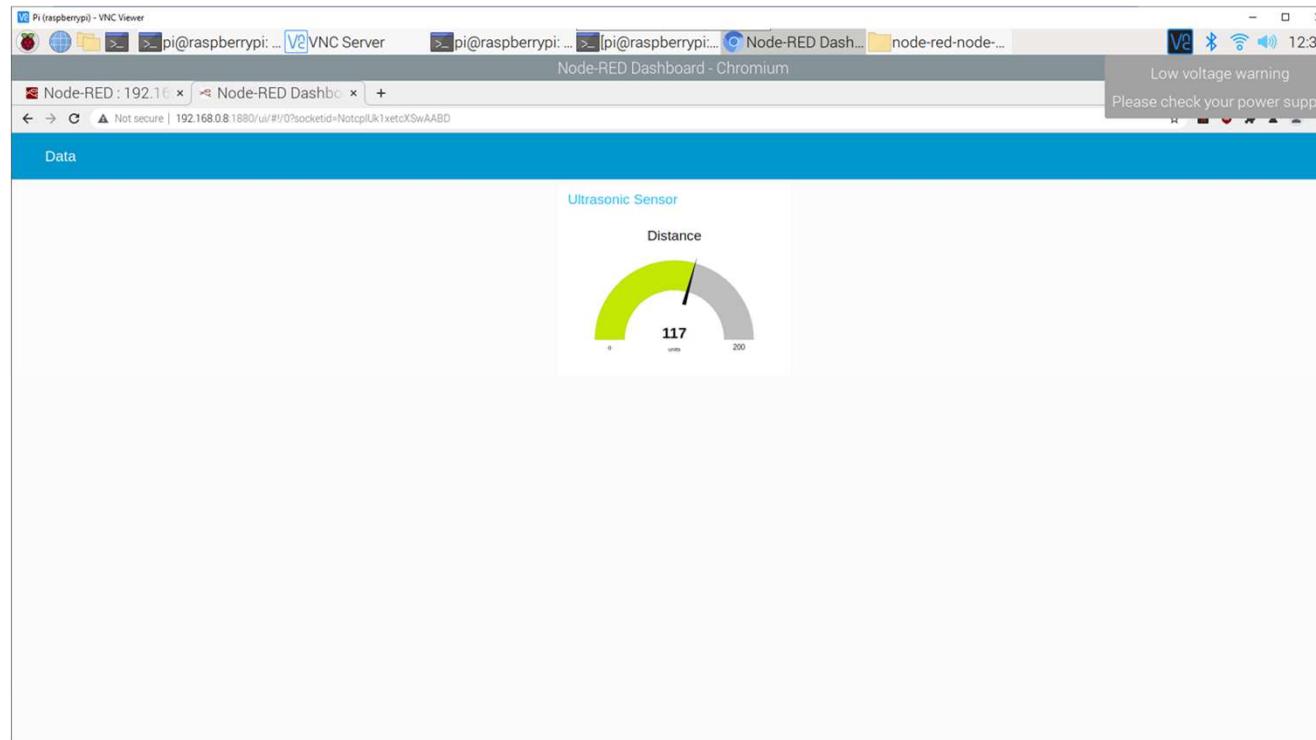
- Triển khai và kiểm tra bảng điều khiển



- ▶ Nhấp vào nút “Deploy” để khởi động quá trình.
- ▶ Nếu bạn nhấp vào Thông báo gỡ lỗi, bạn có thể thấy rằng các giá trị quan sát là các giá trị đầu ra.

## Kiểm tra bảng điều khiển

- Triển khai và kiểm tra bảng điều khiển
  - Kết nối với http://Địa chỉ IP của Raspberry Pi:1880/ui, để bạn có thể thấy thước đo thay đổi theo các phép đo của cảm biến.



Bài 4.

# Xử lý lỗi và khắc phục sự cố

- 4.1. Các lỗi phổ biến nhất về cài đặt
- 4.2. Các lỗi phổ biến nhất về thông báo lỗi và lập trình
- 4.3. Sử dụng Async-Await và promises để async xử lý lỗi không đồng bộ
- 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp
- 4.5. Phân biệt lỗi vận hành và lập trình viên
- 4.6. Xử lý lỗi tập trung
- 4.7. Sử dụng trình ghi nhật cập nhật để tăng khả năng hiển thị lỗi
- 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý
- 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

### Xử lý các lỗi trong việc cài đặt

- Tìm phiên bản của gói npm đã được cài đặt
  - Cách tìm phiên bản của gói node.js/npm đã được cài đặt
  - Gói **cục bộ**

Danh sách \$ npm

- Gói cài đặt toàn cục

Danh sách \$ npm

- Tìm phiên bản của một gói cụ thể bằng cách chuyển tên của nó làm đối số

Danh sách \$ npm grunt

## 4.1. Các lỗi phổ biến nhất về cài đặt

Bài 04

- Cách cài đặt phiên bản chính xác trước đó của gói NPM
  - Nếu bạn phải cài đặt phiên bản cũ hơn của gói, hãy làm rõ nó.

Cài đặt \$ npm <package>@<version>

x

Cài đặt \$ npm express@3.0.0

- Bạn cũng có thể thêm cờ `--save` vào lệnh đó để thêm nó vào phần phụ thuộc pack.json của bạn hoặc cờ `--save-exact` nếu bạn muốn phiên bản chính xác được chỉ định trong phần phụ thuộc pack.json của mình.
- Nếu bạn không chắc có những phiên bản nào của gói, bạn có thể sử dụng:

\$ npm view <package> versions

- Thông báo "Message failed to fetch from registry" trong khi cố gắng cài đặt mô-đun bất kỳ
  - Nếu bạn gặp vấn đề với việc **cài đặt npm {node module bất kỳ}**
  - Cập nhật node (và npm) giải quyết vấn đề.
  - Đầu tiên, hãy gỡ cài đặt phiên bản lỗi thời (tùy chọn, điều này đã khắc phục sự cố gặp phải với các mô-đun chung không được dẫn vào).

```
$ sudo apt-get purge nodejs npm
```

- Sau đó bật repo của nodesource và cài đặt.

```
$ curl -sL https://deb.nodesource.com/setup | sudo bash -  
$ sudo apt-get install -y nodejs
```

- Khi cố gắng tìm kiếm hoặc cài đặt thứ gì đó bằng npm, sẽ xảy ra lỗi sau nếu bạn không sử dụng lệnh ‘sudo’ ngay cả khi bạn đã là quản trị viên.

- Đây có vẻ như là sự cố về quyền trong thư mục chính của bạn. Để đòi lại quyền sở hữu thư mục .npm, hãy thực thi.

```
$ sudo chown -R $(whoami) ~/.npm
```

- Ngoài ra, nếu bắt buộc, bạn sẽ cần quyền ghi trong thư mục [node\\_modules](#).

```
$ sudo chown -R $USER /usr/local/lib/node_modules
```

- npm WARN chưa đáp ứng được sự phụ thuộc (unmet dependency)
  - Sau đây là những giải pháp có thể
    - Cần cài đặt các mô-đun cấp cao nhất theo cách thủ công, chứa các unmet dependency: cài đặt npm [findup-sync@0.1.2](#)
    - Cấu trúc lại package.json. Đặt tất cả các module cấp cao (đóng vai trò phụ thuộc cho các module khác) ở dưới cùng.
    - Chạy lại lệnh [cài đặt npm](#).
  - Sự cố có thể do npm không thể tải xuống tất cả các gói do hết thời gian chờ hoặc do nguyên nhân khác
  - Bạn có thể cài đặt các gói bị lỗi theo cách thủ công bằng cách sử dụng npm install [findup-sync@0.1.2](#).
  - Nếu bạn gặp thông báo lỗi như thế này khi cố gắng cài đặt npm từ gói gốc.

```
npm WARN unmet dependency /Users/seanmackesey/google_drive/code/explore/generator/node_modules/findup-sync/node_modules/glob requires graceful-fs@'~1.2.0' but will load
```

- Khả năng do độ phân giải phụ thuộc bị hỏng nhẹ, xem  
<https://github.com/npm/npm/issues/1341#issuecomment-20634338>

Bài 4.

# Xử lý lỗi và khắc phục sự cố

- 4.1. Các lỗi phổ biến nhất về cài đặt
- 4.2. Các lỗi phổ biến nhất về thông báo lỗi và lập trình
- 4.3. Sử dụng Async-Await và promises để async xử lý lỗi không đồng bộ
- 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp
- 4.5. Phân biệt lỗi vận hành và lập trình viên
- 4.6. Xử lý lỗi tập trung
- 4.7. Sử dụng trình ghi nhật ký cập nhật để tăng khả năng hiển thị lỗi
- 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý
- 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

## Xử lý lỗi về cài đặt

- Tìm phiên bản của gói npm đã cài đặt.
  - Sử dụng Node.js **console.time()** và **console.timeEnd()**

x

```
var i;
console.time(label: "dbsave");

for(i = 1; i < LIMIT; i++){
    db.users.save({id : i, name : "MongoUser [" + i + "]"}, end);
}

end = function(err, saved) {
    console.log((err || !saved)?"Error":"Saved");
    if(--i === 1){console.timeEnd(label: "dbsave");}
};
```

- In dấu vết ngăn xếp trong Node.js

- . Bất cứ đối tượng `Error` cũng chứa phần tử `ngăn xếp` có tác dụng dừng con trỏ tại vị trí mà nó được gọi tới.

```
var stack = new Error().stack  
console.log( stack )
```

- hoặc đơn giản hơn như dưới đây.

```
console.trace( data: "Here I am!" )
```

- Lỗi: EACCES: quyền bị từ chối, quyền truy cập '/usr/local/lib/node\_modules'(1)

- Thay đổi quyền của bạn
  - Trước tiên, hãy kiểm tra xem ai sở hữu thư mục.

```
$ ls -la /usr/local/lib/node_modules
```

- nó đang từ chối quyền truy cập vì thư mục node\_module được sở hữu bởi root.

```
drwxr-xr-x 3 root  wheel 102 Jun 24 23:24 node_modules
```

- Vì vậy, điều này cần được thay đổi bằng cách thay đổi quyền root thành người dùng của bạn, nhưng trước tiên, hãy chạy lệnh bên dưới để kiểm tra tình trạng người dùng hiện tại [Làm cách nào để biết tên của người dùng đang hoạt động thông qua dòng lệnh trong OS X?](#)

- Lỗi: EACCES: quyền bị từ chối, quyền truy cập '/usr/local/lib/node\_modules'(2)

- **id -un** HOẶC **whoami**

```
$ id -un
```

```
$ whoami
```

- Thay đó, thay đổi người sở hữu

```
$ sudo chown -R [owner]:[owner] /usr/local/lib/node_modules
```

Hoặc

```
$ sudo chown -R ownerName: /usr/local/lib/node_modules
```

- Mã lỗi ENOSPC

- “ENOSPC” có nghĩa là không có dung lượng trống trên ổ đĩa.
  - Chạy lệnh dưới đây để tránh ENOSPC.

```
$ echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf && sudo sysctl -p
```

- Đối với Arch Linux, hãy thêm lệnh bên dưới vào /etc/sysctl.d/99-sysctl.conf.

```
fs.inotify.max_user_watches=524288
```

- Sau đó, thực hiện:

```
$ sysctl --system
```

- Lỗi: Không thể đặt tiêu đề sau khi được gửi tới máy khách hàng
  - Khắc phục sự cố này hoạt động bNode.js 4.10 và Express 2.4.3.
  - Đối tượng **res** trong Express là một tập con [của httpServerResponse](#). của Node.js (đọc nguồn http.js). Bạn được phép gọi **res.setHeader(tên, giá trị)** bao nhiêu lần tùy thích cho đến khi bạn gọi **res.writeHead(statusCode)**. Sau **writeHead**, các tiêu đề được đưa vào và bạn chỉ có thể gọi **res.write(data)** và cuối cùng là **res.end(data)**.
  - Lỗi "Lỗi: Không thể đặt tiêu đề sau khi được gửi." có nghĩa là bạn đã ở trạng thái Đang thực hiện hoặc Đã hoàn thành, nhưng một số chức năng đã cố đặt tiêu đề hoặc Mã trạng thái. Khi bạn thấy lỗi này, hãy cố gắng tìm kiếm bất kỳ thứ gì cố gắng gửi tiêu đề sau khi một phần nội dung đã được viết.

**BT** Tìm kiếm các Callbacks vô tình được gọi hai lần hoặc bất kỳ lỗi nào xảy ra sau khi nội dung được gửi.

- Cách khắc phục lỗi npm "npm ERR! code ELIFECYCLE"
  - Nếu bạn gặp mã lỗi ELFECYCLE khi chạy npm start:

Bước 1

```
$ npm cache clean --force
```

- Bước 2: Xóa thư mục node\_modules bằng thư mục [\\$ rm -rf node\\_modules \(rmdir /S /Q node\\_modules trong windows\)](#) hoặc xóa thủ công bằng cách vào thư mục. Nhấp chuột phải > xóa/di chuyển vào thùng rác. Nếu không cập nhật các gói của mình, bạn có thể xóa tệp pack-lock.json.

```
$ rm -rf node_modules package-lock.json
```

- Bước 3

```
$ npm install
```

- Bước 4: Thử lại

```
$ npm start
```

- SyntaxError: Sử dụng const trong chế độ nghiêm ngặt

- Chúng tôi đang làm việc trong node.js và chúng tôi đang sử dụng nó giới hạn ở "chế độ nghiêm ngặt" trong một trong các tệp js. Có lỗi xảy ra khi cố gắng thực thi.
- Answer Case 1 : Updating nodejs solved the issue:

```
npm cache clean - f  
sudo npm install - g n  
sudo n stable      #or, $nvm install stable  
node--version  
node app.js
```

Cập nhật NodeJS giải quyết vấn đề này. Tuy nhiên, sau khi chạy **\$sudo npm install -g n** , bạn có thể gặp lỗi sau:

**npm**: tái định vị **error: npm**: symbol SSL\_set\_cert\_cb, phiên bản libssl.so .10 không được xác định trong tập tin libssl.so .10 với thời gian tham chiếu

Để khắc phục lỗi này, hãy thử nâng cấp openssl bằng lệnh bên dưới:

```
sudo yum update openssl
```

- Đáp án trường hợp 2:

```
'use strict'  
const MAX_IMAGE_SIZE = 1024*1024; // 1 MB
```

- Chạy Node.js như một tiến trình nền không bao giờ chết
  - Nếu bạn đã cố chạy nó dưới dạng một quy trình nền như:

```
$ node server.js &
```

- Khi thiết bị đầu cuối không hoạt động và quá trình chết.
- Bạn muốn tìm một cách có thể duy trì tiến trình ngay cả khi thiết bị đầu cuối bị ngắt kết nối.

```
$ nohup node server.js > /dev/null 2>&1 &
```

- nohup: Không chấm dứt quá trình này ngay cả khi stty bị cắt.
- > /dev/null: stdout đi tới /dev/null (đó là một thiết bị giả không ghi lại bất kỳ kết quả nào).
- 2>&1: stderr also goes to the stdout (which is already redirected to /dev/null). Bạn có thể thay thế &1 bằng đường dẫn tệp để ghi nhật ký lỗi).
- &: chạy lệnh này như một tác vụ nền.

Bài 4.

# Xử lý lỗi và khắc phục sự cố

- 4.1. Các lỗi phổ biến nhất về cài đặt
- 4.2. Các lỗi phổ biến nhất về thông báo lỗi và lập trình
- 4.3. Sử dụng Async-Await và Promises để xử lý lỗi không đồng bộ
- 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp
- 4.5. Phân biệt lỗi vận hành và lập trình viên
- 4.6. Xử lý lỗi tập trung
- 4.7. Sử dụng trình ghi nhật ký cập nhật để tăng khả năng hiển thị lỗi
- 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý
- 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

### Xử lý lỗi không đồng bộ

- Các Callbacks không mở rộng quy mô tốt vì hầu hết các lập trình viên không quen thuộc với chúng.
- Họ nhìn thấy toàn bộ lỗi, xử lý việc lồng mã phức tạp và gây phiền toái cho việc suy luận về dòng mã.
- Các thư viện Promise như BlueBird, async và Q sử dụng RETURN và THROW để nén các kiểu mã tiêu chuẩn kiểm soát luồng chương trình.
- Đặc biệt, nó hỗ trợ kiểu xử lý lỗi bắt thử được yêu thích, kiểu này ngăn đường dẫn mã mặc định trong việc xử lý lỗi cho tất cả các chức năng.

- **Code ví dụ - sử dụng Promise để bắt lỗi**

```
return functionA()
  .then(functionB)
  .then(functionC)
  .then(functionD)
  .catch((err) => logger.error(err))
  .then(alwaysExecuteThisFunction)
```

- Code ví dụ - sử dụng async/await để bắt lỗi

```
async function executeAsyncTask () {  
    try {  
        const valueA = await functionA();  
        const valueB = await functionB(valueA);  
        const valueC = await functionC(valueB);  
        return await functionD(valueC);  
    }  
    catch (err) {  
        logger.error(err);  
    } finally {  
        await alwaysExecuteThisFunction();  
    }  
}
```

## 4.3. Sử dụng Async-Await và promises để async xử lý lỗi không đồng bộ

Bài 04

- Code ví dụ mã chống mău - xử lý lỗi kiểu Callbacks

```
1  getData(someParameter, function(err, result) {  
2      if(err !== null) {  
3          // do something like  
4          // calling the given callback function  
5          // and pass the error  
6          getMoreData(a, function(err, result) {  
7              if(err !== null) {  
8                  // do something like  
9                  // calling the given callback function  
10                 // and pass the error  
11                 getMoreData(b, function(c) {  
12                     getMoreData(d, function(e) {  
13                         if(err !== null ) {  
14                             // you get the idea?  
15                         }  
16                         })  
17                         });  
18                         }  
19                         });  
20                         }  
21                     });
```

Bài 4.

# Xử lý lỗi và khắc phục sự cố

- 4.1. Các lỗi phổ biến nhất về cài đặt
- 4.2. Các lỗi phổ biến nhất về thông báo lỗi và lập trình
- 4.3. Sử dụng Async-Await và promises để async xử lý lỗi không đồng bộ
- 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp
- 4.5. Phân biệt lỗi vận hành và lập trình viên
- 4.6. Xử lý lỗi tập trung
- 4.7. Sử dụng trình ghi nhật ký cập nhật để tăng khả năng hiển thị lỗi
- 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý
- 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

## Đối tượng Lỗi tích hợp

- Bản chất dễ tiếp cận của JavaScript và các tùy chọn luồng mã khác nhau (chẳng hạn như EventEmitterCallbacks, Promises v.v.) thay đổi cách các nhà phát triển đưa ra lỗi. Một số sử dụng chuỗi, những người khác xác định các loại tùy chỉnh của riêng họ.
- Đối tượng Lỗi tích hợp sẵn của Node.js cho phép bạn duy trì tính đồng nhất trong mã của mình và cũng bảo toàn thông tin quan trọng như StackTrace cùng với các thư viện của bên thứ ba.
- Khi đưa ra một ngoại lệ, bạn nên đưa vào ngoại lệ các thuộc tính ngữ cảnh bổ sung, chẳng hạn như tên lỗi và mã lỗi HTTP liên quan.
- Để đạt được sự thống nhất và thông lệ này, hãy xem xét việc mở rộng đối tượng Lỗi bằng các thuộc tính bổ sung, nhưng hãy cẩn thận không lạm dụng nó.
- Nói chung, bạn chỉ nên mở rộng đối tượng Lỗi tích hợp sẵn một lần với AppError cho tất cả các lỗi cấp ứng dụng và chuyển dữ liệu cần thiết làm đối số để phân biệt giữa các loại lỗi khác nhau.
- Không cần phải mở rộng đối tượng Lỗi nhiều lần (một lần cho từng trường hợp lỗi như DbError, HttpError).

## 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp

Bài 04

- **Code ví dụ - làm đúng**

```
// throwing an Error from typical function,  
// whether sync or async  
if(!productToAdd)  
    throw new Error('How can I add new product ' +  
        'when no value provided?');  
  
// 'throwing' an Error from EventEmitter  
const myEmitter = new MyEmitter();  
myEmitter.emit('error', new Error('whoops!'));  
  
// 'throwing' an Error from a Promise  
const addProduct = async (productToAdd) => {  
    try {  
        const existingProduct =  
            await DAL.getProduct(productToAdd.id);  
        if (existingProduct !== null) {  
            throw new Error('Product already exists!');  
        }  
    } catch (err) {  
        // ...  
    }  
}
```

- **Code ví dụ - Anti Pattern**

```
// throwing a string lacks any stack trace information  
// and other important data properties  
if(!productToAdd)  
    throw ('How can I add new product when no value provided?');
```

- **Code ví dụ - làm tốt hơn**

```
// centralized error object that derives from Node's Error
function AppError(name, httpCode, description, isOperational) {
  Error.call(this);
  Error.captureStackTrace(this);
  this.name = name;
  //...other properties assigned here
}

AppError.prototype = Object.create(Error.prototype);
AppError.prototype.constructor = AppError;

module.exports.AppError = AppError;

// client throwing an exception
if(user == null)
  throw new AppError(
    commonErrors.resourceNotFound,
    commonHTTPErrors.notFound,
    description: 'further explanation',
    isOperational: true)
```

Bài 4.

# Xử lý lỗi và khắc phục sự cố

- 4.1. Các lỗi phổ biến nhất về cài đặt
- 4.2. Các lỗi phổ biến nhất về thông báo lỗi và lập trình
- 4.3. Sử dụng Async-Await và promises để async xử lý lỗi không đồng bộ
- 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp
- 4.5. Phân biệt lỗi vận hành và **lỗi của người lập trình**
- 4.6. Xử lý lỗi tập trung
- 4.7. Sử dụng trình ghi nhật cập nhật để tăng khả năng hiển thị lỗi
- 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý
- 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

## Lỗi Vận hành & Lỗi của người Lập trình

- Phân biệt hai loại lỗi này sẽ giúp bạn giảm thiểu thời gian ngừng hoạt động của ứng dụng và tránh các lỗi phức tạp: Lỗi vận hành đề cập đến các tình huống mà bạn hiểu điều gì đã xảy ra và tác động của nó. Ví dụ: một truy vấn tới một số dịch vụ HTTP không thành công do sự cố kết nối.
- Mặt khác, lỗi lập trình đề cập đến các trường hợp bạn không biết tại sao và đôi khi chúng đến từ đâu. Đó có thể là một số mã đang cố đọc nhóm kết nối DB với các giá trị không xác định hoặc rò rỉ bộ nhớ.
- Các lỗi vận hành tương đối dễ xử lý. Thường là lỗi đăng nhập.
- Khi lỗi lập trình viên xảy ra, mọi thứ trở nên phức tạp. Ứng dụng của bạn có thể ở trạng thái không nhất quán và không có gì tốt hơn là khởi động lại.

- Code ví dụ - đánh dấu lỗi thuộc về vận hành (đáng tin cậy)

```
// marking an error object as operational
const myError = new Error('How can I add new product ' +
    'when no value provided?');
myError.isOperational = true;

// or if you're using some centralized error factory
// (see other examples at the bullet
// "Use only the built-in Error object")
class AppError {
    constructor (commonType, description, isOperational) {
        Error.call(this);
        Error.captureStackTrace(this);
        this.commonType = commonType;
        this.description = description;
        this.isOperational = isOperational;
    }
}

throw new AppError(
    errorManagement.commonErrors.InvalidInput,
    description: 'Describe here what happened',
    isOperational: true);
```

Bài 4.

# Xử lý lỗi và khắc phục sự cố

- 4.1. Các lỗi phổ biến nhất về cài đặt
- 4.2. Các lỗi phổ biến nhất về thông báo lỗi và lập trình
- 4.3. Sử dụng Async-Await và promises để async xử lý lỗi không đồng bộ
- 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp
- 4.5. Phân biệt lỗi vận hành và lập trình viên
- 4.6. Xử lý lỗi tập trung
- 4.7. Sử dụng trình ghi nhật ký cập nhật để tăng khả năng hiển thị lỗi
- 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý
- 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

## Không nằm trong phần mềm trung gian

- Nếu không có một đối tượng chuyên dụng để xử lý lỗi, khả năng xử lý lỗi không nhất quán sẽ tăng lên.
- Lỗi xảy ra trong yêu cầu web có thể được xử lý khác với lỗi xảy ra trong giai đoạn khởi động và lỗi xảy ra trong các tác vụ đã được lên kế hoạch.
- Điều này có thể dẫn đến một số loại lỗi được quản lý sai.
- Đối tượng trình xử lý lỗi duy nhất này làm cho lỗi hiển thị bằng cách ghi vào trình nhật ký trực quan, kích hoạt số liệu bằng cách sử dụng một số sản phẩm giám sát (như Prometheus, CloudWatch, DataDog và Sentry) và quyết định xem quy trình có bị lỗi hay không.
- Hầu hết các khung web đều cung cấp cơ chế phần mềm trung gian bắt lỗi. Một lỗi phổ biến là mã xử lý lỗi xuất hiện trong phần mềm trung gian.
- Điều này đảm bảo rằng không thể sử dụng lại cùng một trình xử lý cho các lỗi gặp phải trong các tình huống khác nhau, chẳng hạn như các tác vụ đã lên lịch, người đăng ký hàng đợi tin nhắn và các ngoại lệ chưa được phát hiện.
- Phần mềm trung gian lỗi chỉ nên bắt lỗi và chuyển nó cho trình xử lý.
- Quy trình xử lý lỗi điển hình như sau: Một số mô-đun đưa ra lỗi → bộ định tuyến API bắt lỗi → truyền lỗi đến phần mềm trung gian chịu trách nhiệm bắt lỗi (ví dụ: các cơ chế khác để bắt lỗi cấp độ yêu cầu) → trình xử lý lỗi tập trung được sử dụng.

## Xử lý lỗi tập trung

- **Code ví dụ -** một luồng lỗi điển hình

```
// DAL layer, we don't handle errors here
DB.addDocument(newCustomer, (error, result) => {
  if (error)
    throw new Error('Great error explanation comes here',
      other useful parameters);
});

// API route code, we catch both sync and async errors and
// forward to the middleware
try {
  customerService.addNew(req.body).then((result) => {
    res.status(200).json(result);
  }).catch((error) => {
    next(error);
  });
}
catch (error) {
  next(error);
}

// Error handling middleware, we delegate the handling
// to the centralized error handler
app.use(async (err, req, res, next) => {
  await errorHandler.handleError(err, res);
  //The error handler will send a response
});

process.on("uncaughtException", error => {
  errorHandler.handleError(error);
});

process.on("unhandledRejection", (reason) => {
  errorHandler.handleError(reason);
});
```

- **Code ví dụ** - xử lý lỗi trong một đối tượng chuyên dụng

```
module.exports.handler = new errorHandler();

function errorHandler() {
  this.handleError = async (error, responseStream) => {
    await logger.logError(error);
    await fireMonitoringMetric(error);
    await crashIfUntrustedErrorOrSendResponse(
      error, responseStream);
  };
}
```

- **Code ví dụ - Anti Pattern:** Xử lý lỗi trong phần mềm trung gian

```
// middleware handling the error directly,  
// who will handle Cron jobs and testing errors?  
app.use((err, req, res, next) => {  
    logger.logError(err);  
    if (err.severity == errors.high) {  
        mailer.sendMail(configuration.adminMail,  
            'Critical error occurred', err);  
    }  
    if (!err.isOperational) {  
        next(err);  
    }  
});
```

Bài 4.

# Xử lý lỗi và khắc phục sự cố

- 4.1. Các lỗi phổ biến nhất về cài đặt
- 4.2. Các lỗi phổ biến nhất về thông báo lỗi và lập trình
- 4.3. Sử dụng Async-Await và promises để async xử lý lỗi không đồng bộ
- 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp
- 4.5. Phân biệt lỗi vận hành và lập trình viên
- 4.6. Xử lý lỗi tập trung
- 4.7. Sử dụng trình ghi nhật ký cập nhật để tăng khả năng hiển thị lỗi
- 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý
- 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

# Sử dụng Logger

- Chúng tôi yêu thích console.log nhưng một trình ghi nhật ký bền bỉ và có uy tín như Pino (một tùy chọn mới hơn tập trung vào hiệu suất) là bắt buộc đối với các dự án nghiêm túc.
- Các công cụ ghi nhật ký hiệu suất cao giúp xác định lỗi và các sự cố có thể xảy ra.
- Các đề xuất ghi nhật ký bao gồm:
  - Đăng nhập thường xuyên bằng các cấp độ khác nhau (gỡ lỗi, thông tin, lỗi).
  - Khi ghi nhật ký, hãy cung cấp thông tin theo ngữ cảnh dưới dạng đối tượng JSON.
  - Giám sát và lọc nhật ký bằng API truy vấn nhật ký (tích hợp sẵn cho nhiều trình ghi nhật ký) hoặc phần mềm xem nhật ký.
  - Hiển thị và sắp xếp các báo cáo nhật ký bằng các công cụ thông minh hoạt động như Splunk.

## 4.7. Sử dụng trình ghi nhật ký trưởng thành để tăng khả năng hiển thị lỗi

Bài 04

- Code ví dụ

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

// your centralized logger object
const logger = pino();

// custom code somewhere using the logger
logger.info({ anything: 'This is metadata' },
  'Test Log Message with some parameter %s', 'some parameter');
```

Bài 4.

# Xử lý lỗi và khắc phục sự cố

- 4.1. Các lỗi phổ biến nhất về cài đặt
- 4.2. Các lỗi phổ biến nhất về thông báo lỗi và lập trình
- 4.3. Sử dụng Async-Await và promises để async xử lý lỗi không đồng bộ
- 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp
- 4.5. Phân biệt lỗi vận hành và lập trình viên
- 4.6. Xử lý lỗi tập trung
- 4.7. Sử dụng trình ghi nhật ký cập nhật để tăng khả năng hiển thị lỗi
- 4.8. Nắm bắt các chỉ lệnh Promises chưa được xử lý
- 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

## Tùy chọn Promise chưa được xử lý

- Nói chung, hầu hết mã ứng dụng Node.js/Express hiện đại đều chạy bên trong các lời hứa, cho dù bên trong trình xử lý .then, hàm callbacks hay catch blocks.
- Đáng ngạc nhiên là các lỗi được đưa ra ở những nơi này không được xử lý bởi trình xử lý sự kiện Ngoại lệ và biến mất, trừ khi nhà phát triển nhớ thêm mệnh đề `.catch`.
- Các phiên bản gần đây của Node đã thêm thông báo cảnh báo khi hiển thị tùy chọn chưa được xử lý. Mặc dù điều này có thể giúp bạn nhận thấy khi có sự cố xảy ra, nhưng rõ ràng đó không phải là phương pháp xử lý lỗi phù hợp.
- Một giải pháp đơn giản là chuyển hướng đến trình xử lý lỗi tập trung mà không quên thêm mệnh đề `.catch` trong mỗi lệnh gọi chuỗi promise.
- Việc xây dựng một chiến lược xử lý lỗi dựa trên các nguyên tắc của nhà phát triển là không chắc chắn.
- Cần nhắc sử dụng một phương án dự phòng và đăng ký `process.on('unhandledRejection', callback)`. Điều này sẽ xử lý tất cả các lỗi đã được chỉ định nếu không được xử lý cục bộ.

## 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý

Bài 04

- Code ví dụ: Các lỗi này sẽ không bị phát hiện bởi bất kỳ trình xử lý lỗi nào (ngoại trừ unhandledRejection)

```
DAL.getUserById(1).then((johnSnow) => {
  // this error will just vanish
  if(johnSnow.isAlive === false)
    throw new Error('ahhhh');
});
```

## 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý

Bài 04

- **Code ví dụ:** Nắm bắt các Promises chưa được giải quyết và bị từ chối

```
process.on('unhandledRejection', (reason, p) => {
  // I just caught an unhandled promise rejection,
  // since we already have fallback handler for
  // unhandled errors (see below),
  // let throw and let him handle that
  throw reason;
});

process.on('uncaughtException', (error) => {
  // I just received an error that was never handled,
  // time to handle it and then decide
  // whether a restart is needed
  errorManagement.handler.handleError(error);
  if (!errorManagement.handler.isTrustedError(error))
    process.exit(1);
});
```

Bài 4.

# Xử lý lỗi và khắc phục sự cố

- 4.1. Các lỗi phổ biến nhất về cài đặt
- 4.2. Các lỗi phổ biến nhất về thông báo lỗi và lập trình
- 4.3. Sử dụng Async-Await và promises để async xử lý lỗi không đồng bộ
- 4.4. Chỉ sử dụng đối tượng Lỗi tích hợp
- 4.5. Phân biệt lỗi vận hành và lập trình viên
- 4.6. Xử lý lỗi tập trung
- 4.7. Sử dụng trình ghi nhật ký cập nhật để tăng khả năng hiển thị lỗi
- 4.8. Nắm bắt các chỉ lệnh promises chưa được xử lý
- 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

### Để tránh các lỗi ẩn

- Tất cả chúng ta đều biết tầm quan trọng của việc kiểm tra các đối số và nhanh chóng thất bại để tránh các lỗi ẩn (xem ví dụ về mã anti-pattern bên dưới).
- Nếu không, hãy đọc về Lập trình rõ ràng và Lập trình phòng thủ.
- Trên thực tế, chúng ta có xu hướng tránh điều này vì những phiền toái khi mã hóa (ví dụ: nghĩ đến việc xác thực các đối tượng JSON phân cấp với các trường như email và ngày tháng) – Các thư viện như Joi và Trình xác thực biến nhiệm vụ tẻ nhạt này trở nên dễ dàng.
- Lập trình phòng thủ là một cách tiếp cận để cải thiện phần mềm và mã nguồn của nó về chất lượng chung, giảm số lỗi và sự cố phần mềm.
- Làm cho mã nguồn dễ hiểu – Mã nguồn phải dễ đọc và dễ hiểu để được kiểm tra trình phiên dịch.
- Nó đảm bảo rằng phần mềm hoạt động theo cách có thể dự đoán được bắt chấp hành động của người dùng hoặc thông tin đưa vào ngoài dự kiến.

## 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

Bài 04

- **Code ví dụ:** xác thực đầu vào JSON phức tạp bằng cách sử dụng 'Joi'

```
var memberSchema = Joi.object().keys({
  password: Joi.string().regex(/^[a-zA-Z0-9]{3,30}$/),
  birthyear: Joi.number().integer().min(1900).max(2013),
  email: Joi.string().email()
});

function addNewMember(newMember) {
  // assertions come first
  Joi.assert(newMember, memberSchema);
  //throws if validation fails

  // other logic here
}
```

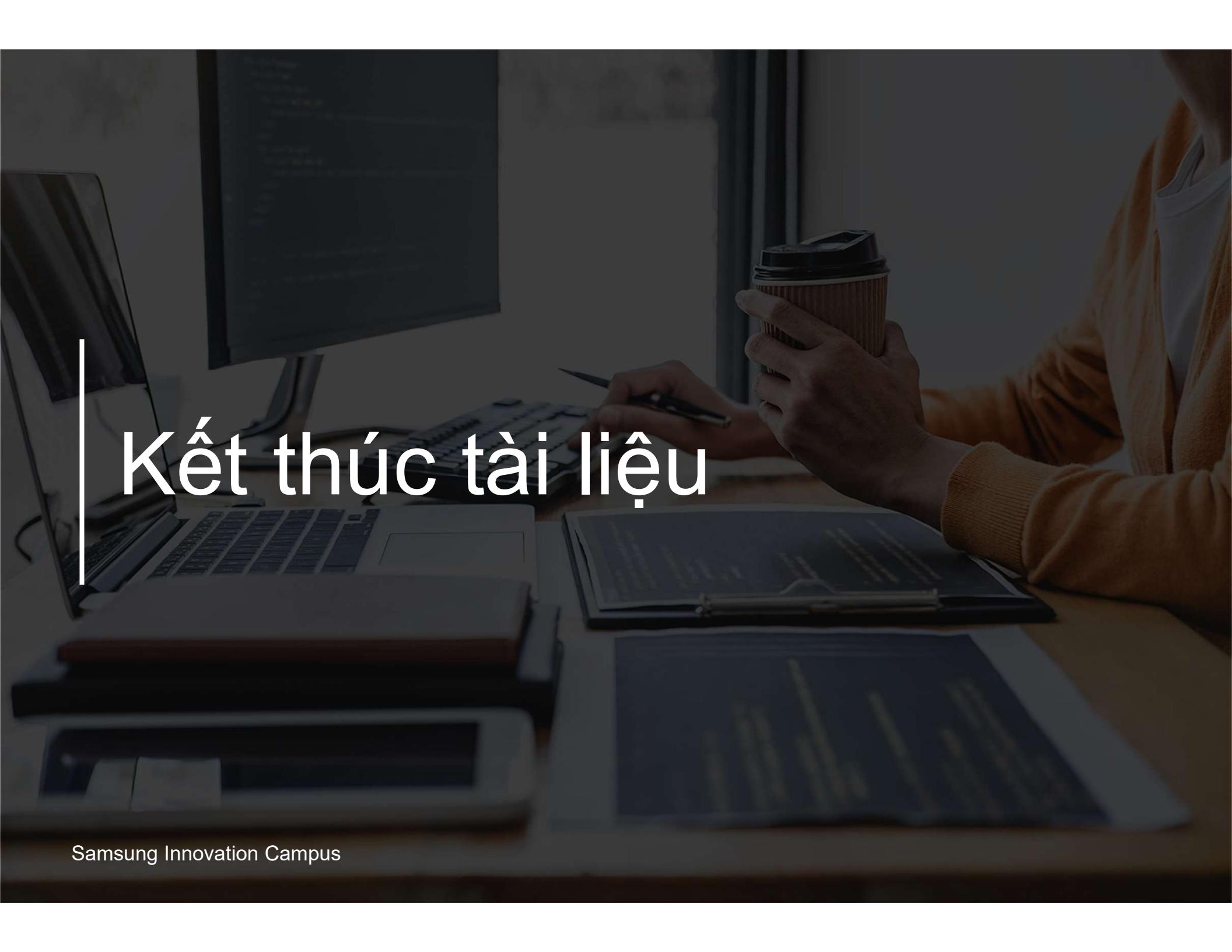
## 4.9. Thất bại nhanh chóng, xác thực các đối số bằng thư viện chuyên dụng

Bài 04

- Anti-pattern: không có xác nhận dẫn đến các lỗi khó chịu

```
// if the discount is positive
// let's then redirect the user to print his discount coupons
function redirectToPrintDiscount(httpResponse, member, discount) {
  if (discount != 0) {
    httpResponse.redirect(`/discountPrintView/${member.id}`);
  }
}

redirectToPrintDiscount(httpResponse, someMember);
// forgot to pass the parameter discount,
// why the heck was the user redirected to the discount screen?
```

A photograph of a person's hands and arms. They are wearing a yellow long-sleeved shirt. In their left hand, they hold a white paper coffee cup with a black lid. In their right hand, they hold a black pen. They are resting their right hand on a white keyboard. To the left of the keyboard is a silver laptop. Behind the laptop is a black computer monitor. On the desk in front of the laptop are several thick, dark-colored books.

# Kết thúc tài liệu



Together for Tomorrow!  
**Enabling People**

Education for Future Generations

©2022 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of book.

This book is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.

To use this book other than the curriculum of Samsung Innovation Campus or to use the entire or part of this book, you must receive written consent from copyright holder.