

React Advance

Bài 2: Giới thiệu ExpressJS



Nội dung

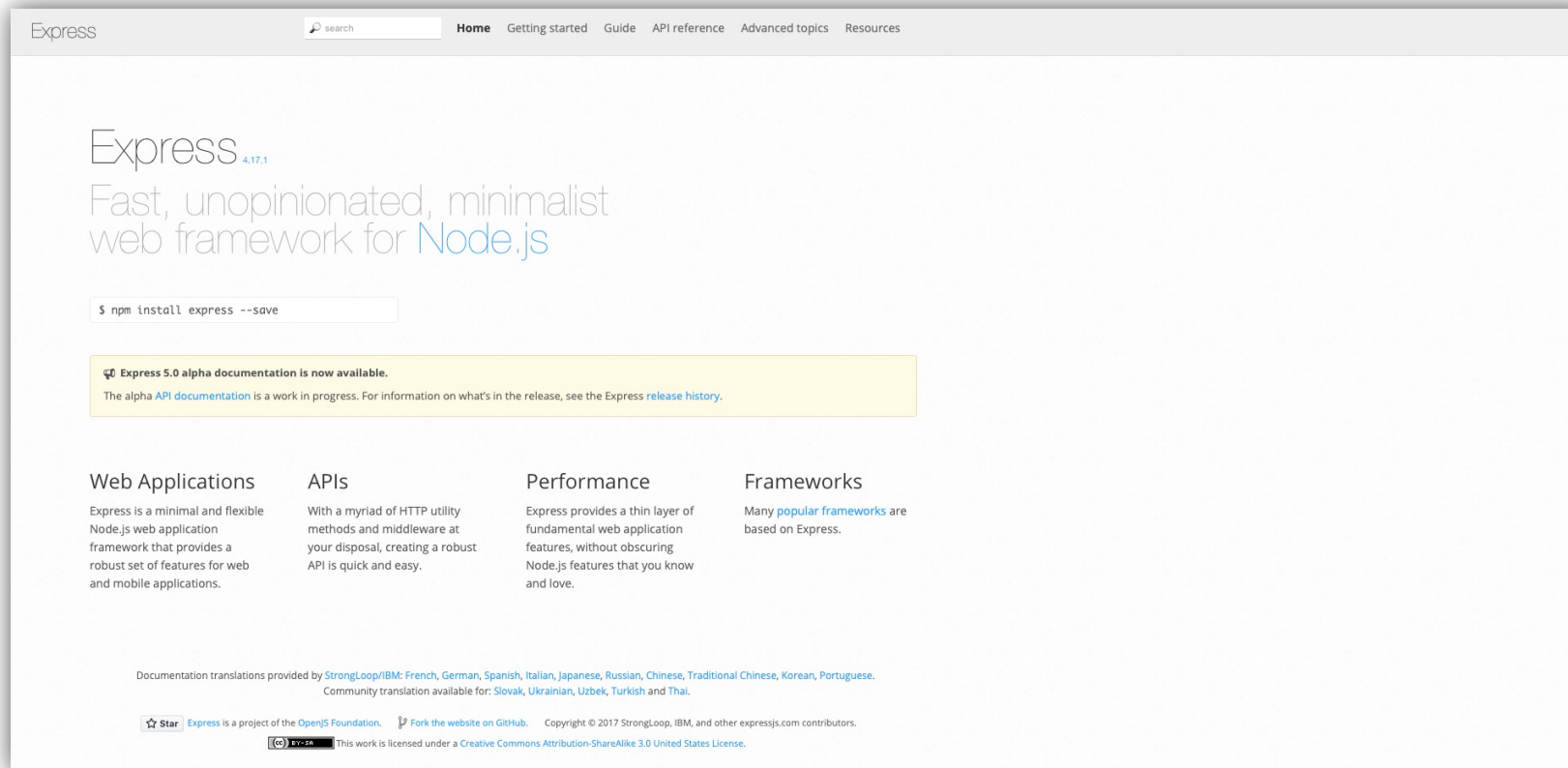
1. Giới thiệu ExpressJS
2. Cài đặt môi trường lập trình
3. Xây dựng ứng dụng Hello World
4. Routing cơ bản
5. Static files



1. Giới thiệu ExpressJS



Giới thiệu ExpressJS



<http://expressjs.com>



Giới thiệu ExpressJS

- ExpressJS là một web framework cho NodeJS với các ưu điểm:
 - Fast - nhanh
 - Unopinionated – ổn định
 - Minimalist – gọn, nhẹ
- ExpressJS hỗ trợ
 - Web application – gọn nhẹ và linh hoạt (flexible)
 - APIs – hỗ trợ đầy đủ các giao thức HTTP, dễ sử dụng dụng và cài đặt
 - Performance – cung cấp đủ các chức năng để xây dựng một ứng dụng Web cơ bản
 - Framework – đây là một trong những chương trình khung ứng dụng cơ sở của NodeJS



2. Cài đặt môi trường lập trình



Cài đặt

- Sử dụng lệnh sau để cài đặt ExpressJS vào node-module của ứng dụng web với NodeJS

```
> npm install express --save
```

- Để thực hiện hoàn chỉnh, cần thực hiện các bước sau
 - Khởi tạo ứng dụng NodeJS với lệnh \$ **npm init**
 - Điền đầy đủ thông tin cấu hình ứng dụng NodeJS
 - File package.json sẽ lưu đầy đủ các thông tin của project vừa được khởi tạo
 - Cài đặt ExpressJS cho ứng dụng thông qua lệnh

```
> npm install express --save
```



Cài đặt

```
> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (lab02)
version: (1.0.0)
description: ExpressJS
entry point: (index.js)
test command:
git repository:
keywords: nodejs, expressjs
author: teacher
license: (ISC)
About to write to /Users/ndhuy/Docs/Class/VietAusCenter/Syllabus/ReactJS Advance/7. Document Mentor/Lab02/package.json:
{
  "name": "lab02",
  "version": "1.0.0",
  "description": "ExpressJS",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "nodejs",
    "expressjs"
  ],
  "author": "teacher",
  "license": "ISC"
}

Is this OK? (yes) yes
```



```
> npm install express --save

added 50 packages, and audited 51 packages in 2s

found 0 vulnerabilities
```



```
package.json > ...
1  {
2    "name": "lab02",
3    "version": "1.0.0",
4    "description": "ExpressJS",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [
10     "nodejs",
11     "expressjs"
12   ],
13   "author": "teacher",
14   "license": "ISC",
15   "dependencies": {
16     "express": "^4.17.1"
17   }
18 }
```




Cài đặt

- Lệnh `$ npm install express --save`
 - `--save` - có nghĩa là sẽ lưu lại thông tin của framework express trong dependencies của package.json
 - `--no-save` – sẽ chỉ cài đặt framework express trong node-modules chứ không lưu lại thông tin của express



3. Xây dựng ứng dụng Hello World



Xây dựng ứng dụng “Hello World”

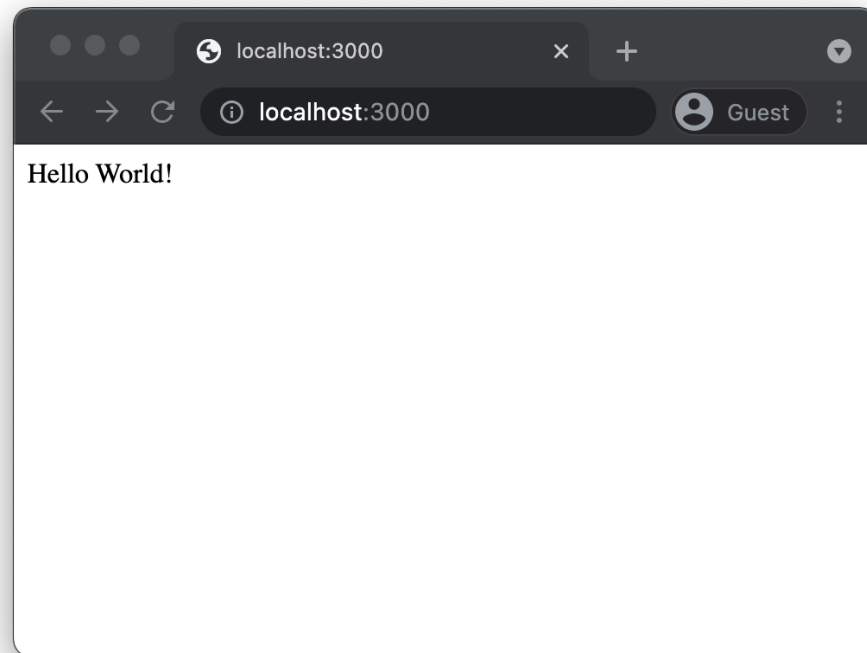
```
JS index.js > ...
1  const express = require('express')
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => {
6    res.send('Hello World!')
7  })
8
9  app.listen(port, () => {
10    console.log(`Example app listening at http://localhost:${port}`)
11  })
```



```
> node index.js
Example app listening at http://localhost:3000
█
```

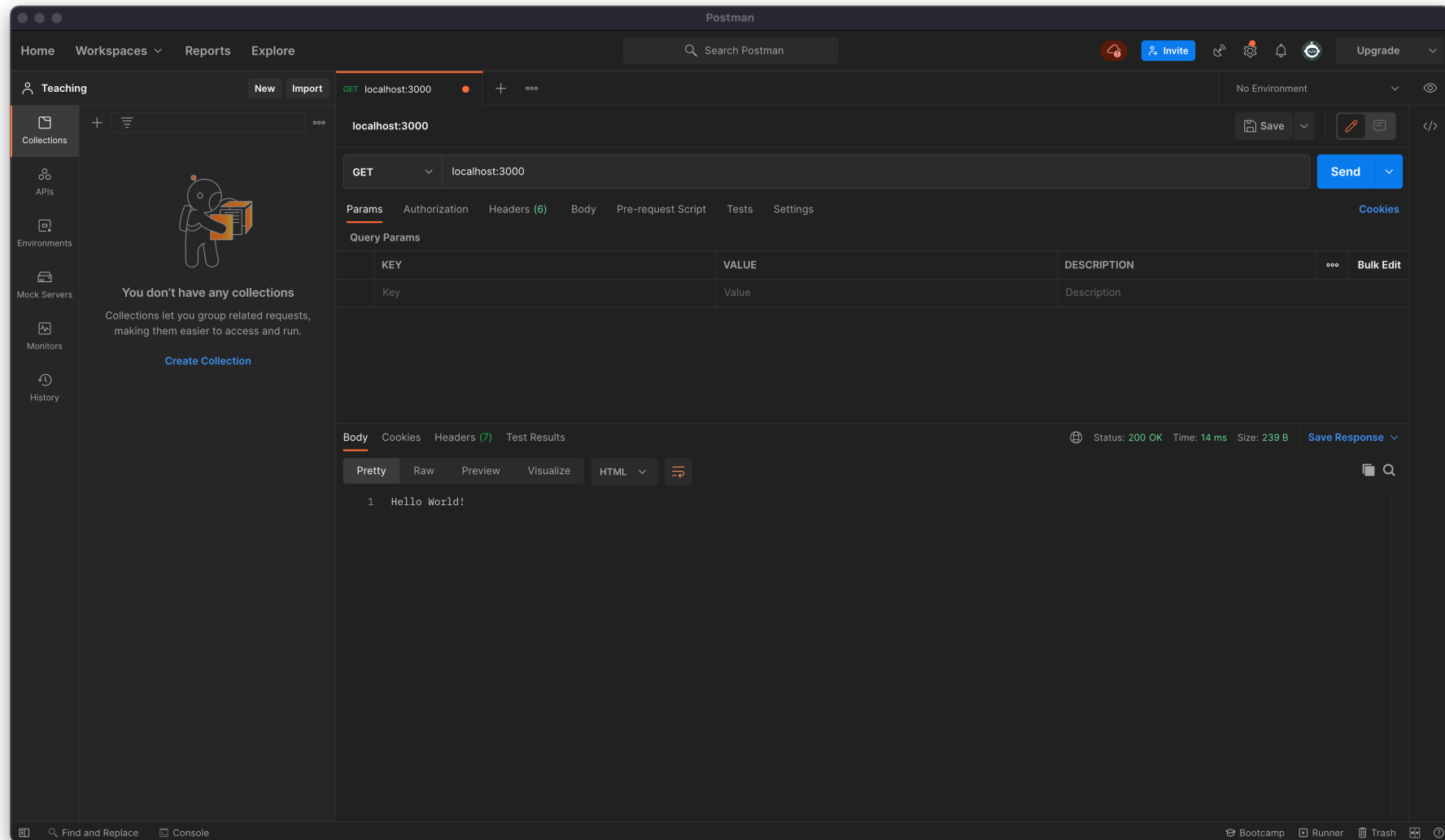


Xây dựng ứng dụng “Hello World” – duyệt trên trình duyệt Chrome





Xây dựng ứng dụng “Hello World” – sử dụng Postman app





4. Routing cơ bản



Routing cơ bản

- Routing (bộ định tuyến) sẽ xác định các phản hồi (response) từ các yêu cầu (request) của client với các endpoint (điểm đến/điểm truy cập) cụ thể được gọi là URI
- Các phương thức (method) của HTTP sẽ có 4 loại thông dụng:
 - GET – dùng cho việc lấy thông tin từ server về (READ/SELECT)
 - POST – dùng cho việc tạo mới thông tin (CREATE/INSERT)
 - PUT/PATCH – dùng cho việc cập nhập thông tin (UPDATE)
 - DELETE – dùng cho việc xóa thông tin (DELETE)



Routing cơ bản

- Mỗi route có thể có một hoặc nhiều hàm hanler (hàm xử lý) khác nhau
- Route sẽ được định nghĩa theo cấu trúc như sau

```
app.METHOD(PATH, HANDLER)
```

- Với
 - app – là một instance (thể hiện) của express
 - METHOD – là
 - Một HTTP request method (GET, POST, PUT, DELETE)
 - Thường được viết dưới dạng lowercase
 - PATH - là đường dẫn đến endpoint trên server
 - HANDLER – là hàm xử lý khi route được khớp với request từ client với lên



Routing cơ bản

- Phản hồi với thông điệp “Hello world” từ homepage (trang chủ)
 - Giao thức - GET
 - Endpoint – ‘/’

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



Routing cơ bản

- Phản hồi với request POST từ root route '/'
 - Giao thức - POST
 - Endpoint – '/'

```
app.post('/', (req, res) => {  
  res.end('Got a POST request')  
})
```



Routing cơ bản

- Phản hồi với request PUT tới '/user' route
 - Giao thức - PUT
 - Endpoint – '/user'

```
app.put('/user', (req, res) => {  
  res.end('Got a PUT request at /user')  
})
```



Routing cơ bản

- Phản hồi với request DELETE tới '/user' route
 - Giao thức - DELETE
 - Endpoint – '/user'

```
app.delete('/user', (req, res) => {  
  res.end('Got a DELETE request at /user')  
})
```



Routing cơ bản

```
index.js > ...
1  const express = require('express')
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => {
6    res.send('Hello World!')
7  })
8
9  app.post('/', (req, res) => {
10   res.end('Got a POST request')
11 })
12
13 app.put('/user', (req, res) => {
14   res.end('Got a PUT request at /user')
15 })
16
17 app.delete('/user', (req, res) => {
18   res.end('Got a DELETE request at /user')
19 })
20
21 app.listen(port, () => {
22   console.log(`Example app listening at http://localhost:${port}`)
23 })
```



Routing cơ bản – request GET

The screenshot shows the Postman interface for a GET request to `localhost:3000`. The request is saved and ready to be sent. The response is displayed in the 'Body' tab, showing 'Hello World!' in a code editor. The status is 200 OK, with a response time of 21 ms and a size of 239 B.

Request Details:

- Method: GET
- URL: localhost:3000
- Environment: No Environment

Response Details:

- Status: 200 OK
- Time: 21 ms
- Size: 239 B
- Body: Hello World!

KEY	VALUE	DESCRIPTION
Key	Value	Description



Routing cơ bản – request POST

POST localhost:3000

No Environment

localhost:3000

Save

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	KEY	VALUE	DESCRIPTION		Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 6 ms

Size: 164 B

Save Response

Pretty

Raw

Preview

Visualize

Text

1 Got a POST request



Routing cơ bản – request PUT

The screenshot shows a REST client interface with a PUT request configured to `localhost:3000/user`. The interface includes tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, and Settings. The Params tab is active, showing a table for Query Params.

KEY	VALUE	DESCRIPTION
Key	Value	Description

Below the Params tab, the Body tab is selected, showing a single log entry: "1 Got a PUT request at /user". The status bar at the bottom indicates a successful response: Status: 200 OK, Time: 5 ms, Size: 172 B.



Routing cơ bản – request DELETE

The screenshot shows a REST client interface with a dark theme. At the top, the URL bar displays 'localhost:3000/user' with a 'DELETE' method selected. Below the URL bar, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is active, showing a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. The table contains one row with 'Key' and 'Value'. Below the table, there is a 'Send' button. At the bottom, the 'Body' tab is active, showing a 'Pretty' view of the response. The response is a single line: '1 Got a DELETE request at /user'. The status bar at the bottom right shows 'Status: 200 OK', 'Time: 5 ms', and 'Size: 175 B'.

DEL localhost:3000/user

localhost:3000/user

DELETE localhost:3000/user

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

1 Got a DELETE request at /user

Status: 200 OK Time: 5 ms Size: 175 B



5. Static files



Static file

- Để bảo mật việc truy cập các tài nguyên trên web-server nên
 - NodeJS & ExpressJS chỉ cho phép truy cập các tài nguyên thông qua các endpoint đã được khai báo
 - Các URI khác đều không có quyền truy cập
- Tuy nhiên, khi xây dựng ứng dụng Web, thì các tài nguyên về hình ảnh, về CSS cần phải được truy cập tự do
- Để xử lý vấn đề này, ExpressJS định nghĩa các tập lệnh để khai báo static files như hình ảnh, CSS, các file JavaScript được kích hoạt ở client ...
- Sử dụng câu lệnh **express.static** để khai báo các static file này



Static file

- Cấu trúc lệnh khai báo static file

```
express.static(root, [options])
```

- **root** – là đường dẫn đến folder chứa các static files cần khai báo
- **options** – là phần định nghĩa các tham số hỗ trợ như
 - **dotfile** – các file bắt đầu bằng dấu "." vì thường các file này là các file cấu hình
 - **etag** – bật hoặc tắt chế độ caching thông qua etag header
 - **extensions** – cho phép set các đuôi file được khai báo static như ['html', 'htm']
 - **fallthrough** – chỉ dẫn cho client biết có lỗi xảy ra ở request nên yêu cầu xử lý không được thực hiện
 - **immutable** – bật hoặc tắt chế độ cache-control trong response header
 - **index** – trả về file index trong folder được chỉ định (được khai báo)
 - **lastModified** – set Last-Modified header cho lần cuối thay đổi file ở OS
 - **maxAge** – set thuộc tính max-age trong cache-control được tính bằng đơn vị milli-seconds hoặc string
 - **redirect** – chuyển hướng đến dấu "/" khi tên path là một folder
 - **setHeaders** – là function cho việc cấu hình setting HTTP header khi serve với các file



Static file

- Ví dụ sẽ khai báo static file cho nguyên folder public (chứa các resource tĩnh của web)

```
app.use(express.static('public'))
```



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



Static file

```
app.use(express.static('public'))  
app.use(express.static('files'))  
app.use('/static', express.static('public'))
```



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



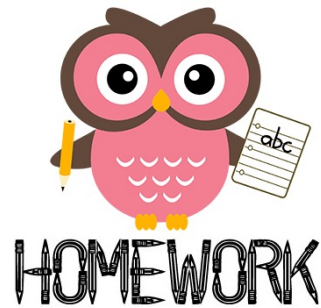
Static file

- Hoặc có thể khai báo đường dẫn cố định từ code

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



Bài tập về nhà



- Xây dựng một ứng dụng ExpressJS thực hiện các yêu cầu sau:
 - Xây dựng ứng dụng Web-Server thực hiện lấy thông tin của một list các item được đọc lên từ file data.json có cấu trúc như hình bên
 - Thực hiện các truy cập sau với các phương thức sau:
 - GET – lấy hết danh sách các item (2 điểm)
 - POST – tự tạo mới một item (2 điểm)
 - PUT – cập nhật tên của item đầu tiên thành ABC (2 điểm)
 - DELETE – xóa item có id = 2 (2 điểm)
 - Dựng được web-server (2 điểm)

```
[  
  {  
    "id": 1,  
    "name": "iPhone 12",  
    "price": 1000  
  },  
  {  
    "id": 2,  
    "name": "iPad pro M1",  
    "price": 1200  
  },  
  {  
    "id": 4,  
    "name": "macBook pro M1",  
    "price": 2000  
  },  
  {  
    "id": 3,  
    "name": "ABC",  
    "price": 1200  
  },  
  {  
    "id": 5,  
    "name": "FEG",  
    "price": 2222  
  }  
]
```