

CT449: PHÁT TRIỂN ỨNG DỤNG WEB

Ứng dụng Contactbook - Backend - Phần 1

Chúng ta sẽ xây dựng ứng dụng Quản lý danh bạ theo mô hình Ứng dụng trang đơn (SPA) sử dụng công nghệ Node, Express, MongoDB phía backend (API server) và Vue phía frontend (GUI). Trong buổi thực hành 1 và 2, chúng ta sẽ xây dựng backend cho ứng dụng.

Server của ứng dụng sẽ phải hỗ trợ các yêu cầu sau:

- *POST /api/contacts*: tạo một liên hệ (contact) mới
- *GET /api/contacts*: trả về tất cả các liên hệ trong CSDL
- *DELETE /api/contacts*: xóa tất cả các liên hệ
- *GET /api/contacts/favorite*: trả về các liên hệ được yêu thích
- *GET /api/contacts/<contact-id>*: trả về thông tin một liên hệ dựa trên id
- *PUT /api/contacts/<contact-id>*: cập nhật một liên hệ dựa trên id
- *DELETE /api/contacts/<contact-id>*: xóa một liên hệ dựa trên id
- Yêu cầu đến URL không được định nghĩa: thông báo lỗi 404 "Resource not found"

Một liên hệ (contact) gồm các trường thông tin sau: name (chuỗi), email (chuỗi), address (chuỗi), phone (chuỗi), favorite (true/false). **Định dạng chung cho dữ liệu trao đổi giữa client và server là định dạng JSON.** Mã nguồn được quản lý bởi git và upload lên GitHub.

Hướng dẫn dưới đây sẽ cài đặt các yêu cầu nêu trên. Sinh viên có thể không cần cài đặt giống hướng dẫn, chỉ cần thực hiện đúng các yêu cầu đặt ra ở trên.

Yêu cầu cho báo cáo thực hành:

- Tạo file báo cáo cần nộp là tập tin PDF hoặc tập tin Word theo quy định sau:
 - Tên tập tin theo quy cách: MSSV_Hoten_BACKEND_1 (ví dụ: B1234567_Nguyen_Van_A_BACKEND_1)
 - Nội dung file báo cáo là trình bày các bước thực hiện (chụp hình code và kết quả thực hiện bằng trình duyệt và Postman).

Bước 0: Cài đặt node và git

- Tải và cài đặt nodejs: <https://nodejs.org/en/download/>. Có thể tải và cài nodejs trực tiếp hoặc cài nodejs thông qua nvm (<https://github.com/coreybutler/nvm-windows>).
- Tải và cài đặt git: <https://git-scm.com/download/win>.
- Kiểm tra rằng có thể gõ lệnh `node` và `git` trên Terminal / Command Prompt:

```
PS G:\test NODE\N.M.Trung\Backend\Backend-Phan 1> node -v
v16.17.0
PS G:\test NODE\N.M.Trung\Backend\Backend-Phan 1> git --version
git version 2.37.3.windows.1
PS G:\test NODE\N.M.Trung\Backend\Backend-Phan 1>
```

(Trong trường hợp không thể chạy lệnh từ terminal thì cần thêm đường dẫn đến thư mục chứa chương trình `node` và `git` vào biến môi trường PATH trên máy của bạn).

Bước 1: Tạo ứng dụng Node

- Tạo thư mục dự án, ví dụ "*contactbook-backend*".
- Trong thư mục này, ta khởi tạo ứng dụng Node:

```
npm init
```

npm sẽ hỏi một số thông tin để tạo tập tin package.json cho dự án:

```
package name: (contactbook-backend)
version: (1.0.0)
description:
entry point: (index.js) server.js
test command:
git repository:
keywords:
author: Trung Nguyen
license: (ISC)
About to write to C:\test-NODE\N.H.Trung\Backend\Backend-Phan-1\contactbook-backend\package.json:

{
  "name": "contactbook-backend",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Trung Nguyen",
  "license": "ISC"
}
```

- Cài đặt các package cần thiết:

```
npm install express cors
```

Bước 2: Quản lý mã nguồn dự án với git và GitHub

- Tạo tập tin .gitignore: `npx gitignore node`. Tập tin *.gitignore* liệt kê các thư mục, tập tin sẽ không được quản lý bởi git (ví dụ như thư mục *node_modules*).
- Trong thư mục gốc dự án, chạy lệnh `git init`. Gõ tiếp `git status`, chúng ta sẽ thấy có 3 tập tin chưa được quản lý bởi git là *.gitignore*, *package-lock.json* và *package.json*.

```
PS D:\CTU\Courses\CT449.WebApps\labs\lab1\contactbook-backend> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        package-lock.json
        package.json

nothing added to commit but untracked files present (use "git add" to track)
```

- Cho git quản lý 3 tập tin *.gitignore*, *package-lock.json* và *package.json* bằng lệnh sau đây:

```
git add .gitignore package-lock.json package.json
```

Gõ `git status` để kiểm tra lại:

```
PS D:\CTU\Courses\CT449.WebApps\labs\lab1\contactbook-backend> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   package-lock.json
        new file:   package.json
```

- Thực thi `git commit -m "Cai dat du an"` để lưu các thay đổi. Tùy chọn `-m` trong câu lệnh cho phép tạo ghi chú cho commit này.

Chú ý: Nếu nhận được thông báo như sau:

```
*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.
```

thì git yêu cầu thông tin về tên và địa chỉ email. Thực thi hai câu lệnh git config sau đây để cung cấp git tên và địa chỉ email của bạn:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

- Đăng ký tài khoản và tạo dự án trên GitHub. Sau khi tạo, chúng ta upload dự án cục bộ lên GitHub như sau:

```
git remote add origin <github-url>
git push origin master
```

Trong đó, bạn sẽ thay thế `<github-url>` bằng URL của dự án đã tạo trên GitHub.

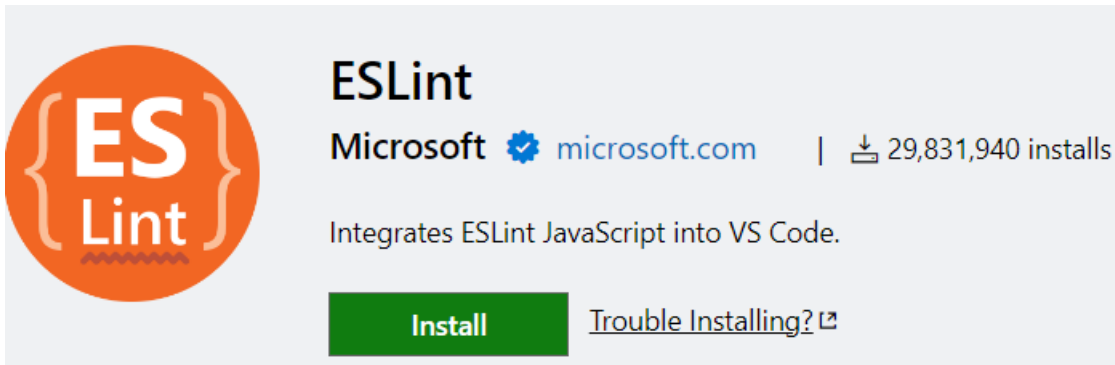
Thực hiện chứng thực tài khoản và kiểm tra lại các tập tin đã được tải lên GitHub:



Bước 3: Cấu hình Visual Studio Code, ESLint và Prettier

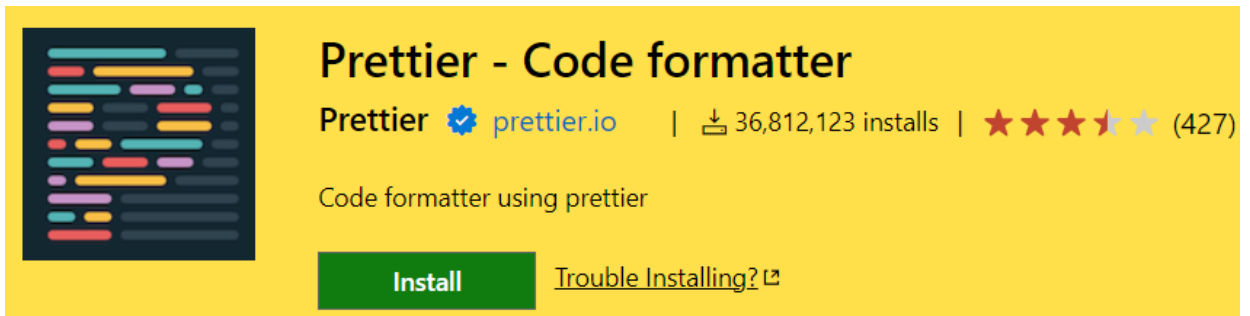
- ESLint là một công cụ phân tích tĩnh mã JavaScript, giúp tìm và sửa các vấn đề trong mã nguồn. Cài

đặt **ESLint extension** cho VSCode.



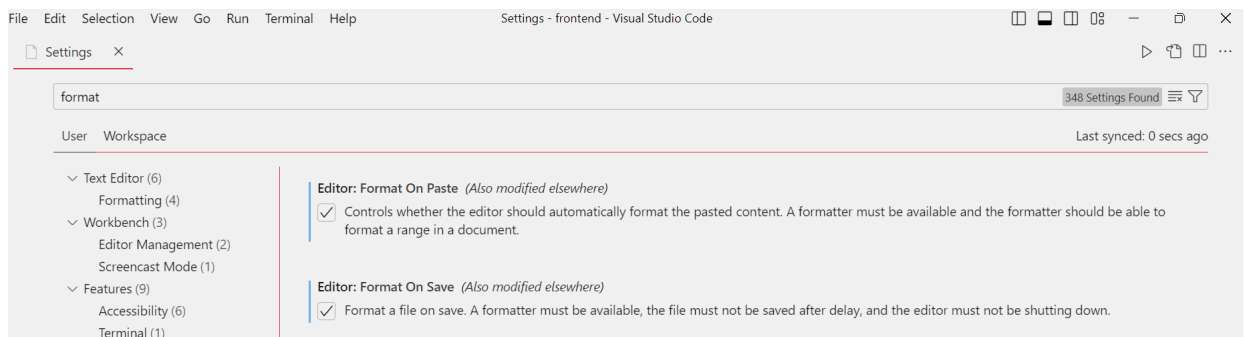
The image shows the ESLint extension card in the VS Code marketplace. On the left is the ESLint logo, which consists of an orange circle with the text 'ES' in white and 'Lint' in orange below it. To the right of the logo, the text 'ESLint' is in large black font, followed by 'Microsoft' and a blue checkmark icon next to 'microsoft.com'. Below this, it says 'Integrates ESLint JavaScript into VS Code.' At the bottom, there is a green 'Install' button and a link 'Trouble Installing?' with an external link icon.

- Prettier là công cụ hỗ trợ định dạng tự động các tập tin mã nguồn. Cài đặt **Prettier - Code formatter** cho VSCode.



The image shows the Prettier - Code formatter extension card in the VS Code marketplace. On the left is the Prettier logo, which is a dark blue square with colorful horizontal lines. To the right of the logo, the text 'Prettier - Code formatter' is in large black font, followed by 'Prettier' and a blue checkmark icon next to 'prettier.io'. Below this, it says 'Code formatter using prettier'. At the bottom, there is a green 'Install' button and a link 'Trouble Installing?' with an external link icon.

- Cấu hình VSCode cho phép định dạng mã lệnh khi save hoặc paste: Vào File > Preferences > Settings, gõ "format" vào ô tìm kiếm và đánh dấu vào hai tùy chọn "Editor: Format on Paste" và "Editor: Format on Save":



- Để eslint và prettier hoạt động trên VSCode thì cần phải lần lượt cài gói eslint và prettier toàn cục hoặc cục bộ trong từng dự án. Trong thư mục dự án, cài đặt các gói thư viện cần thiết như sau:

```
npm i -D eslint prettier eslint-config-prettier
```

(Các luật của eslint có thể xung đột với cách prettier định dạng tập tin mã nguồn. Để tránh điều này, chúng ta có thể sử dụng gói `eslint-config-prettier` nhằm mục đích tắt đi các luật eslint liên quan đến định dạng mã nguồn).

Sau đó tạo một tập tin tên `.eslintrc.js` tại thư mục gốc của dự án:

```
module.exports = {
  env: {
    node: true,
    commonjs: true,
    es2021: true,
  },
  extends: ['eslint:recommended', 'prettier'],
};
```

(Tập tin `.eslintrc.js` cũng có thể được tạo ra bằng cách thực thi câu lệnh `npx eslint --init` và trả lời một số câu hỏi lựa chọn).

- Lưu thay đổi lên git:

```
git add -u
git add .eslintrc.js
git commit -m "Cau hinh ESLint cho du an"
```

Bước 4: Cài đặt Express

Trong thư mục gốc của dự án, tạo tập tin `app.js` và `server.js` lần lượt với nội dung sau:

- `app.js`:

```
const express = require("express");
const cors = require("cors");

const app = express();

app.use(cors());
app.use(express.json());

app.get("/", (req, res) => {
  res.json({ message: "Welcome to contact book application." });
});

module.exports = app;
```

- `server.js`:

```
const app = require("./app");
const config = require("./app/config");

// start server
const PORT = config.app.port;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Lần lượt tạo thư mục `app`, `app/config` và tập tin `app/config/index.js`.

```
const config = {
  app: {
    port: process.env.PORT || 3000,
  }
};

module.exports = config;
```

Mở cửa sổ terminal tại thư mục gốc của dự án và thực hiện lệnh chạy server: `node server.js`. Truy cập `http://localhost:3000/` để kiểm tra kết quả.

Để dễ dàng hơn cho việc phát triển code, chúng ta có thể cài đặt thêm gói hỗ trợ `nodemon`. Gói `nodemon` này sẽ giám sát các thay đổi trên mã nguồn và tự động khởi động server:

```
npm install nodemon --save-dev
```

Sau khi cài đặt, mở tập tin `package.json` và thay đổi cấu hình mục "scripts" như sau:

```
...
"script": {
  "start": "nodemon server.js"
},
...
```

Với cấu hình như trên, chúng ta có thể chạy server bằng lệnh `npm start` thay vì `node server.js` và mỗi lần các tập tin mã nguồn bị thay đổi, nodemon sẽ tự động khởi động lại server giúp chúng ta.

Lưu các thay đổi trên vào git:

```
git add -u # Xác nhận thay đổi trên các tập tin đã được theo dõi bởi git
git add app/ app.js server.js # Thêm các thư mục và tập tin mới để git theo dõi
git commit -m "Cài đặt express chạy thông báo xin chào"
```

Lưu ý: trước khi thực hiện commit thì nên kiểm tra xem có còn bỏ sót tập tin nào không bằng lệnh `git status`.

Bước 5: Định nghĩa Controller và các route

- Tạo thư mục `controllers` trong thư mục `app`. Sau đó tạo tập tin `contact.controller.js` trong thư mục `controllers` với nội dung sau:

app > controllers > JS contact.controller.js > ...

```
1  exports.create = (req, res) => {
2    |    res.send({ message: "create handler" });
3  };
4
5  exports.findAll = (req, res) => {
6    |    res.send({ message: "findAll handler" });
7  };
8
9  exports.findOne = (req, res) => {
10   |    res.send({ message: "findOne handler" });
11 };
12
13 exports.update = (req, res) => {
14   |    res.send({ message: "update handler" });
15 };
16
17 exports.delete = (req, res) => {
18   |    res.send({ message: "delete handler" });
19 };
20
21 exports.deleteAll = (req, res) => {
22   |    res.send({ message: "deleteAll handler" });
23 };
24
25 exports.findAllFavorite = (req, res) => {
26   |    res.send({ message: "findAllFavorite handler" });
27 };
```

- Tạo thư mục *routes* trong thư mục *app*. Sau đó tạo tập tin *contact.route.js* trong thư mục *routes* với nội dung sau:

```

1  const express = require("express");
2  const contacts = require("../controllers/contact.controller");
3
4  const router = express.Router();
5
6  router.route("/")
7    .get(contacts.findAll)
8    .post(contacts.create)
9    .delete(contacts.deleteAll);
10
11  router.route("/favorite")
12    .get(contacts.findAllFavorite);
13
14  router.route("/:id")
15    .get(contacts.findOne)
16    .put(contacts.update)
17    .delete(contacts.delete);
18
19  module.exports = router;

```

Trong đoạn code trên, chúng ta định nghĩa các route quản lý liên hệ được hỗ trợ bởi ứng dụng web. Mỗi route là sự kết hợp của đường dẫn, phương thức HTTP (GET, POST, PUT, DELETE) và đoạn code xử lý.

Tiếp theo, chúng ta sẽ đăng ký các route vừa tạo với express app. Hiệu chỉnh tập tin *app.js* như sau:

```

...
const contactsRouter = require("./app/routes/contact.route");
...
...
app.use("/api/contacts", contactsRouter);

module.exports = app;

```

Các route quản lý liên hệ sẽ được dùng khi đường dẫn bắt đầu là */api/contacts*. Ví dụ để yêu cầu server gửi về các contact được yêu thích (route định nghĩa tại dòng lệnh số 11, tập tin *contact.route.js*), client cần gửi yêu cầu GET đến URL: <http://localhost:3000/api/contacts/favorite>.

Sử dụng ứng dụng Postman hoặc curl để kiểm tra các route vừa tạo.

Sau khi đảm bảo các route hoạt động tốt, lưu các thay đổi lên git:

```

git add -u
git add app/controllers app/routes
git commit -m "Định nghĩa các route cho tài nguyên Contact"

```

Bước 6: Cài đặt xử lý lỗi

Tạo tập tin *app/api-error.js*:


```
class ApiError extends Error {
  constructor(statusCode, message) {
    super();
    this.statusCode = statusCode;
    this.message = message;
  }
}

module.exports = ApiError;
```

Mở tập tin *app.js* và thêm vào các middleware xử lý lỗi như sau:

```
...
const ApiError = require("../app/api-error");

const app = express();
...
app.use("/api/contacts", contactsRouter);

// handle 404 response
app.use((req, res, next) => {
  // Code ở đây sẽ chạy khi không có route được định nghĩa nào
  // khớp với yêu cầu. Gọi next() để chuyển sang middleware xử lý lỗi
  return next(new ApiError(404, "Resource not found"));
});

// define error-handling middleware last, after other app.use() and routes calls
app.use((err, req, res, next) => {
  // Middleware xử lý lỗi tập trung.
  // Trong các đoạn code xử lý ở các route, gọi next(error) sẽ chuyển về middleware xử
  lý lỗi này
  return res.status(error.statusCode || 500).json({
    message: error.message || "Internal Server Error",
  });
});
...

```

Sử dụng một HTTP client (Chrome, Postman, curl, ...) và gửi yêu cầu đến một URL không được định nghĩa trong ứng dụng, kiểm tra nhận được lỗi 404 và thông báo "Resource not found".

Lưu các thay đổi vào git:

```
git add -u
git add app/api-error.js
git commit -m "Cai dat xu ly loi"
```

Đẩy các thay đổi lên GitHub bằng lệnh: `git push origin master`.

Cấu trúc thư mục dự án đến thời điểm này sẽ như sau:

- ▼ app
 - ▼ config
 - JS index.js
 - ▼ controllers
 - JS contact.controller.js
 - ▼ routes
 - JS contact.route.js
 - JS api-error.js
 - > node_modules
 - 🔗 .eslintrc.js
 - 🔗 .gitignore
 - JS app.js
 - { } package-lock.json
 - { } package.json
 - JS server.js