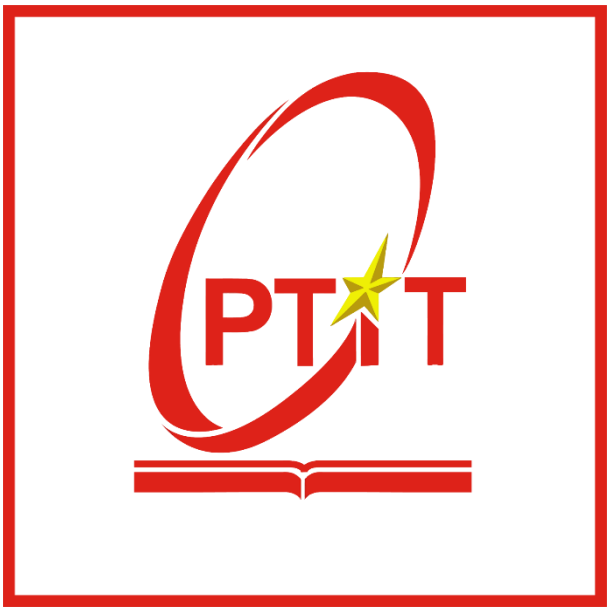
BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO THỰC TẬP CƠ SỞ TUẦN 4

Xây dựng Restful API với Nodejs và Expressjs

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

Sinh viên thực hiện:

Lê Quang Thanh – B22DCVT509

1. **API trong Nodejs là gì?**

- API (Application Programming Interface) là một tập hợp các quy tắc, định dạng và giao thức giúp các thành phần trong ứng dụng giao tiếp trao đổi dữ liệu với nhau. Trong các dự án Fullstack JavaScript, API đóng vai trò là cầu nối quan trọng, đảm bảo sự phối hợp liền mạch giữa frontend và backend:

- Frontend (React.js): Gửi các yêu cầu (request) đến backend để lấy dữ liệu, như danh sách sản phẩm, thông tin người dùng, hoặc trạng thái đơn hàng. Sau đó, nó hiển thị dữ liệu này cho người dùng qua giao diện.
- Backend (Node.js + Express.js): Tiếp nhận yêu cầu từ frontend, xử lý logic nghiệp vụ, truy vấn cơ sở dữ liệu và gửi phản hồi (response) chứa dữ liệu cần thiết trở lại cho frontend.

2. **Cách thiết kế 1 API hiệu quả**

- Để xây dựng một API hoạt động mượt mà và thân thiện với người dùng, phải tuân thủ các yêu cầu sau:

- Tuân Thủ RESTful Principles:
    - RESTful API tuân theo các nguyên tắc tiêu chuẩn, sử dụng các phương thức HTTP như:
        - GET: Lấy thông tin dữ liệu.
        - POST: Thêm mới dữ liệu.
        - PUT: Cập nhật dữ liệu hiện có.
        - DELETE: Xóa dữ liệu.
    - Cách đặt tài nguyên trong URL phải logic và dễ hiểu (ví dụ: /users, /orders), giúp API rõ ràng hơn trong mắt người sử dụng.
- Sử Dụng JSON làm định dạng dữ liệu: Việc chọn dữ liệu định dạng JSON sẽ đảm bảo hoạt động và hiệu quả trong quá trình trao đổi thông tin giữa giao diện người dùng và phụ trợ. JSON được xem dưới dạng chuẩn định dạng trong API phát triển nhờ các điểm nổi bật ưu tiên:
- Đọc và viết dễ dàng : Đơn giản, thân thiện với cả người và máy tính.
- Tương tự rộng rãi : JSON được hỗ trợ trên hầu hết các biến phổ cài đặt ngôn ngữ, từ JavaScript, Python đến Java.

- Phân Trang Dữ Liệu (Pagination):

Khi trả về lượng dữ liệu lớn, việc gửi toàn bộ dữ liệu trong một yêu cầu sẽ khiến máy chủ tải quá tải và giảm tốc độ phản hồi. Thay vào đó, nên áp dụng:

- Kỹ thuật phân trang : Sử dụng các tham số như page và limit để kiểm tra việc kiểm soát dữ liệu trả về. Ví dụ: /products?page=1&limit=20: Lấy 20 sản phẩm đầu tiên trên trang 1.
- Bảo Mật API:

Đảm bảo an toàn cho API là yếu tố không thể bỏ qua. Một số biện pháp bảo vệ hiệu quả bao gồm:

- Sử dụng JWT (JSON Web Token) : Tạo token để xác thực người dùng. Mã thông báo này cần được đính kèm trong mỗi yêu cầu để xác định tính hợp lệ.
- Mã hóa HTTPS : Đảm bảo an toàn bộ truyền dữ liệu qua API đều được mã hóa, tránh các nguy cơ như sinh sản hoặc tấn công trung gian (MITM).

3. Xây dựng API với Nodejs và Expressjs

3.1. Cài Đặt Backend

- Tạo một thư mục mới và khởi động dự án Node.js

```
mkdir api-backend && cd api-backend
npm init -y
```

- Cài đặt các thư viện cần thiết :

- Express.js: Framework giúp xây dựng API nhanh chóng.
- Body-parser: Hỗ trợ xử lý JSON dữ liệu trong nội dung yêu cầu.
- Nodemon: Giúp tự động khởi động lại máy chủ khi có thay đổi về nguồn mã.

- Lệnh cài đặt:

```
npm install express body-parser
npm install --save-dev nodemon
```

- Tạo máy chủ khởi động tệp :Tạo tệp server.js và thêm mẫu mã

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = 3000;

app.use(bodyParser.json());

app.get('/', (req, res) => {
    res.send('API is running!');
});

app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
});
```

- Chạy máy chủ : Thêm tập lệnh khởi động trong package.json

```
"scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
}
```

- Khởi động máy chủ bằng lệnh: npm run dev

3.2. Tạo API Đơn Giản – CRUD Danh Sách Công Việc

- Xây dựng 1 API CRUD (Tạo, Đọc, Cập nhật, Xóa) để quản lý danh sách công việc:

- Tạo điểm cuối để lấy danh sách công việc :

```
let tasks = [];

app.get('/tasks', (req, res) => {
    res.json(tasks);
});
```

- Thêm công việc mới :

```
app.post('/tasks', (req, res) => {
    const task = req.body;
    tasks.push(task);
    res.status(201).json(task);
});
```

- Cập nhật công việc :

```
app.put('/tasks/:id', (req, res) => {
    const { id } = req.params;
    const updatedTask = req.body;
    tasks = tasks.map(task => (task.id === parseInt(id) ? updatedTask : task));
    res.json(updatedTask);
});
```

- Xóa công việc :

```
app.delete('/tasks/:id', (req, res) => {
    const { id } = req.params;
    tasks = tasks.filter(task => task.id !== parseInt(id));
    res.status(204).send();
});
```

4. Kết Nối API Với Frontend React.js

- Sau khi backend hoạt động thành công, bước tiếp theo là tích hợp API hợp lý với giao diện người dùng để tạo ra một ứng dụng hoàn chỉnh. Đây là lúc frontend (React.js) trở thành giao diện trực tiếp cho người dùng, giúp dễ dàng tương tác với dữ liệu từ backend. Quá trình này không chỉ đơn giản là gửi và nhận dữ liệu mà còn là cơ hội để tối ưu hóa trải nghiệm của người dùng. Từ việc hiển thị thông tin, bổ

sung mới, cập nhật hay xóa dữ liệu, kết nối API đóng vai trò như “cánh tay kết nối dài” giữa giao diện và xử lý logic bên dưới.

```jsx
import React, { useState, useEffect } from 'react';

function TodoList() {
  const [todos, setTodos] = useState([]);

  useEffect(() => {
    fetch('http://localhost:3000/api/todos')
      .then((response) => response.json())
      .then((data) => setTodos(data));
  }, []);

  return (
    <ul>
      {todos.map((todo) => (
        <li key={todo._id}>
          {todo.task} - {todo.completed ? 'Done' : 'Pending'}
        </li>
      ))}
    </ul>
  );
}

export default TodoList;
```

5. Bảo Mật API Với JWT

- Để bảo vệ API khỏi những quyền truy cập trái phép và bảo đảm hợp lệ người dùng chỉ có thể sử dụng, cơ chế bảo mật được kiểm tra là điều không thể thiếu. Một trong những phương pháp phổ biến và hiệu quả nhất là sử dụng JWT (JSON Web Token) .

- JWT là một mã hóa chuỗi chứa người dùng xác thực thông tin. Mã thông báo này sẽ được tạo khi người dùng đăng nhập thành công và được sử dụng trong mọi yêu cầu gửi đến máy chủ.

- Cách kích hoạt cơ sở của JWT:

- Đăng nhập và tạo mã thông báo : Khi người dùng cung cấp thông tin đăng nhập chính xác, máy chủ sẽ tạo một JWT chứa xác thực thông tin và gửi về ứng dụng khách.
- Gửi yêu cầu mã thông báo đính kèm : Với mỗi yêu cầu sau đó, khách hàng phải gửi JWT đính kèm này trong tiêu đề (thường là Authorization: Bearer <token>).
- Xác thực mã thông báo : Mã thông báo kiểm tra phụ trợ, xác thực tính hợp lệ và quyền của người dùng trước khi xử lý yêu cầu.

- Cách sử dụng JWT không chỉ tăng cường bảo mật mà còn giúp API hoạt động linh hoạt hơn khi phát triển các tính năng như phân quyền hoặc hạn chế truy cập theo vai trò người dùng.

- Thêm middleware bảo vệ API:

```
const jwt = require('jsonwebtoken');

// Middleware xác thực
function authenticateToken(req, res, next) {
  const token = req.header('Authorization');
  if (!token) return res.status(401).send('Access denied');

  try {
    const verified = jwt.verify(token, 'yourSecretKey');
    req.user = verified;
    next();
  } catch (err) {
    res.status(400).send('Invalid token');
  }
}
```

- Bảo vệ endpoint:

```
app.get('/api/secure-todos', authenticateToken, async (req, res) => {
  const todos = await Todo.find();
  res.json(todos);
});
```