



Cấu trúc thư mục Backend (Node.js/Express)

Trong backend Node.js thường tuân thủ phong cách MVC/MVC-chiều phục vụ API. Ví dụ có thư mục **controllers/** chứa các module xử lý request (mỗi model/tính năng tương ứng một controller, ví dụ `user.controller.js`, `product.controller.js`)^{1 2}. Thư mục **routes/** định nghĩa các endpoint và chuyển hướng đến controller tương ứng (ví dụ `user.routes.js` khai báo `router.get('/users', userController.list)`); các file route này được nhập vào `app.js` và gắn vào `app.use()`, ví dụ `app.use('/', websiteRoutes); app.use('/api', apiRoutes)`³. Ngoài ra thường có **models/** (định nghĩa schema hoặc ORM models tương ứng mỗi bảng MySQL) và **services/** (chứa logic nghiệp vụ tách biệt, được controller gọi tới) để tách rõ trách nhiệm. Thư mục **middleware/** dùng cho các hàm middleware chung (xác thực, xử lý lỗi toàn cục, CORS, v.v.). Các thư mục phụ như **config/** chứa file cấu hình (ví dụ `db.config.js`, `auth.config.js`), **utils/** hoặc **helpers/** chứa hàm tiện ích, và **database/** (nếu dùng ORM như Sequelize thì có file khởi tạo kết nối ở đây) cũng được dùng. Ví dụ minh họa:

```
backend/
├── config/          # file cấu hình (ví dụ db.conf.js, app.keys.js, ... )
├── database/        # khởi tạo kết nối (ví dụ Sequelize hoặc DB connection)
├── controllers/    # controller xử lý request (App.controller.js,
User.controller.js, ...) 1 2
├── routes/          # định nghĩa route (App.routes.js, Auth.routes.js, ...)
4
└── models/          # định nghĩa schema/bảng DB (User.model.js,
Product.model.js, ...)
├── services/        # (tùy chọn) logic nghiệp vụ (UserService.js, ...)
├── middleware/      # middleware chung (auth, error handling, logging, ...)
├── utils/           # hàm tiện ích (ví dụ Logger, helper)
├── server.js        # file khởi chạy Express app
├── .env              # biến môi trường (DB_HOST, JWT_SECRET, ...)
└── package.json      # khai báo dependencies và npm scripts
```

Trong đó, **controllers** chịu trách nhiệm xử lý yêu cầu và trả dữ liệu (JSON hoặc render view)²; **routes** chỉ dùng để định tuyến và gọi controller thích hợp. Ví dụ đoạn mã cấu hình route trong `app.js`:

```
const websiteRoutes = require("./routes/Website.routes");
const apiRoutes = require("./routes/Api.routes");
// ...
app.use("/", websiteRoutes);
app.use("/api", apiRoutes);
```

(câu lệnh tương tự minh họa trong nguồn³).

Cấu trúc thư mục Frontend (Vue.js + Tailwind)

Frontend Vue thường có cấu trúc dưới `src/` như sau để tách biệt trách nhiệm:

- **assets/**: lưu các tệp tĩnh (hình ảnh, font, stylesheet chung) ⁵. Thư mục này chứa các file không gắn với một component nào cụ thể, ví dụ ảnh minh họa hoặc styles toàn cục (Tailwind CSS cũng có file `tailwind.config.js` tại gốc).
- **components/**: các component tái sử dụng (nút bấm, form, card, widget, v.v.) ⁶. Component ở đây độc lập, phục vụ nhiều trang.
- **layouts/**: chứa các layout dùng chung như header, footer, sidebar, dùng cho các trang nhất định ⁷. Ví dụ có `AdminLayout.vue` cho trang quản trị có sidebar, và `MainLayout.vue` cho website người dùng.
- **views/** (hoặc **pages/**): chứa các trang (route components) chính; mỗi file Vue ở đây tương ứng một route.
- **router/** hoặc **router.js**: cấu hình Vue Router (các route mapping đến component trong views) ⁸.
- **store/**: nếu dùng Vuex hoặc Pinia, để quản lý state toàn cục ⁹. Ví dụ có các module như `auth.js`, `user.js` trong store.
- **services/** (tùy chọn): chứa logic gọi API (Axios instance, các hàm gọi backend).
- **utils** hoặc **helpers**: các hàm tiện ích dùng chung (định dạng ngày, xử lý dữ liệu, v.v.).
- **main.js**, **App.vue**: entrypoint của ứng dụng Vue (bootstrap router, store, render App).

Tại gốc frontend còn có các file cấu hình: `package.json` (scripts `serve`, `build`), `tailwind.config.js` và `postcss.config.js` (cài Tailwind), `.env` (các biến môi trường front-end như `API_URL`), và thư mục `public/` chứa `index.html`, favicon, v.v.

Ví dụ minh họa cấu trúc (đơn lẻ) cho một ứng dụng Vue:

```
frontend/ (hoặc src/)  
|   ├── assets/  
|   ├── components/  
|   ├── layouts/  
|   ├── views/      # hoặc pages  
|   ├── router/  
|   |   └── index.js  
|   ├── store/  
|   |   └── index.js  
|   ├── App.vue  
|   └── main.js
```

Danh sách thư mục Vue này tuân thủ gợi ý chuẩn của Vue: assets lưu tài nguyên tĩnh, components chứa component độc lập ⁵ ⁶, layouts chứa bố cục dùng chung ¹⁰, router và store lần lượt cho cấu hình router và state ¹¹.

Frontend: Tách riêng client và admin

Trong dự án có cả website người dùng và dashboard admin, thường tách riêng mã nguồn cho mỗi bên. Ví dụ dùng Vue CLI multi-page hoặc monorepo chứa hai app con. Một cách tổ chức:

```

src/
├─ Admin/      # mã nguồn cho trang admin
│   ├─ components/
│   ├─ layouts/
│   ├─ views/    # ví dụ: Dashboard.vue, Users.vue, ...
│   └─ router.js # cấu hình Vue Router cho admin
└─ main.js     # entry cho admin

├─ Client/     # mã nguồn cho trang người dùng
│   ├─ components/
│   ├─ layouts/
│   ├─ views/    # ví dụ: Home.vue, About.vue, ...
│   └─ router.js # cấu hình Vue Router cho client
└─ main.js     # entry cho client

└─ assets/     # tài nguyên chung (nếu có)
└─ components/ # component dùng chung
└─ layouts/    # layout chung
└─ router.js   # (tùy chọn) router tổng
└─ store/
└─ main.js     # (nếu làm single SPA); nhưng với multi-page thường không
cần

```

Cấu hình multi-page trong `vue.config.js` có thể như ví dụ bên dưới, để build hai trang riêng **client** và **admin**:

```

module.exports = {
  pages: {
    client: {
      entry: 'src/Client/main.js',
      template: 'public/client.html',
      filename: 'client.html'
    },
    admin: {
      entry: 'src/Admin/main.js',
      template: 'public/admin.html',
      filename: 'admin.html'
    }
  }
}

```

Khi build, sẽ sinh ra hai tệp HTML riêng (`client.html` và `admin.html`). Cấu hình tương tự có thể tham khảo [12](#) [13](#). Tóm lại mỗi phần (client/admin) có route và layout riêng; ví dụ URL `/admin/dashboard` sẽ đến component trong thư mục Admin (có thể sử dụng middleware/router guard để xác thực).

Kết nối tới MySQL và Cấu hình

Thông thường backend sử dụng thư viện `mysql2` hoặc ORM như Sequelize để kết nối MySQL. Các thông tin kết nối (`host, user, password, database`) được đưa vào biến môi trường trong file `.env` hoặc tập tin cấu hình (`config/db.config.js`). Ví dụ, tạo một file `config.js` ở gốc dự án chứa:

```

module.exports = {
  database: {
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASS,
    database: process.env.DB_NAME
  }
};

```

Sau đó trong `server.js` hoặc `app.js` import config này và thiết lập kết nối:

```

const config = require("./config");
const mysql = require('mysql2');
const conn = mysql.createConnection({
  host: config.database.host,
  user: config.database.user,
  password: config.database.password,
  database: config.database.database
});
conn.connect(err => {
  if (err) throw err;
  console.log("Connected to MySQL!");
});

```

(Ý tưởng này dựa theo hướng dẫn: đặt file config ở gốc project rồi `require` vào mã chính [14](#) [15](#).)
Hoặc nếu dùng Sequelize: khởi tạo ở `database/index.js` như `sequelize = new Sequelize(db_name, user, pass, {host, dialect: 'mysql'})`.

Cấu hình và Script build

Một số tệp cấu hình quan trọng:

- **.env** (trong backend, frontend nếu cần) chứa biến môi trường (cổng, DB, keys). *Không đưa .env lên Git!*
- **package.json** ở mỗi phần (backend/frontend) định nghĩa scripts (ví dụ `"start": "node server.js"`, `"serve": "vue-cli-service serve"`, `"build": "vue-cli-service build"`), cùng dependencies (Express, Vue, Tailwind, ...).
- **tailwind.config.js** và **postcss.config.js** để cấu hình Tailwind (bao gồm đường dẫn đến các file Vue để purge CSS).
- **vue.config.js** hoặc **vite.config.js** cho Vue nếu cần tùy chỉnh (ví dụ setup proxy API, pages multi-entry như trên).

Ví dụ minh họa cây thư mục

Dưới đây là ví dụ sơ đồ thư mục tổng hợp cho dự án fullstack (backend + frontend tách client/admin) với Express+MySQL và Vue+Tailwind:

```

project-root/
  └── backend/
      ├── config/
      │   ├── db.config.js          # cấu hình DB
      │   └── auth.config.js        # cấu hình auth, JWT
      ├── database/
      │   └── index.js             # khởi tạo kết nối MySQL (Sequelize hoặc
mysql2)
      ├── controllers/
      │   ├── user.controller.js   # ví dụ controller xử lý Users
      │   └── auth.controller.js
      ├── routes/
      │   ├── user.routes.js       # định nghĩa đường dẫn /users
      │   └── auth.routes.js
      ├── models/
      │   └── user.model.js         # định nghĩa bảng users
      ├── services/
      │   └── user.service.js       # logic nghiệp vụ (nếu có)
      ├── middleware/
      │   └── auth.middleware.js
      ├── utils/
      │   └── logger.js
      ├── server.js                # entrypoint Express
      ├── .env                      # chứa DB_HOST, DB_USER, JWT_SECRET, ...
      └── package.json

  └── frontend/
      ├── src/
          ├── Admin/
          │   ├── components/
          │   ├── layouts/
          │   ├── views/                 # ví dụ: Dashboard.vue, Users.vue
          │   └── router.js              # routes cho /admin
          ├── Client/
          │   ├── components/
          │   ├── layouts/
          │   ├── views/                 # ví dụ: Home.vue, About.vue
          │   └── router.js              # routes cho website chính
          ├── assets/                  # ảnh, fonts, css chung
          ├── components/              # component dùng chung
          ├── layouts/                 # layout chung (NavBar, Footer...)
          ├── views/                   # (nếu không tách Client/Admin)
          ├── router.js                # (nếu dùng chung) hoặc index.js
          ├── store/                   # Vuex/Pinia store
          ├── App.vue
          └── main.js
      ├── public/
          ├── index.html
          └── admin.html
      └── tailwind.config.js

```

```
└── package.json  
└── vite.config.js (hoặc vue.config.js)
```

Mỗi thư mục trên có vai trò cụ thể: `controllers` chịu trách nhiệm xử lý request và trả dữ liệu [1](#) [2](#); `routes` chỉ định các đường dẫn API [4](#); `models` (hoặc `schema` nếu dùng thư viện như Joi) định nghĩa cấu trúc dữ liệu; `services` (nếu có) tách logic nghiệp vụ; `components/views/layouts/assets` ở frontend như đã mô tả ở trên [5](#) [6](#); `store` chứa quản lý trạng thái toàn cục [9](#). Với thiết lập này, lập trình viên dễ dàng mở rộng (thêm model, route mới hoặc giao diện mới) và duy trì dự án quy mô vừa-lớn.

Nguồn tham khảo: Cấu trúc thư mục điển hình cho Node/Express [1](#) [4](#) và cho Vue.js [5](#) [6](#) [10](#) [9](#); phương pháp thiết lập multi-page (client/admin) cho Vue CLI [12](#) [13](#); hướng dẫn kết nối MySQL trong Node.js [14](#) [15](#).

[1](#) Express Tutorial Part 4: Routes and controllers - Learn web development | MDN

https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/routes

[2](#) [3](#) [4](#) Folder Structure for NodeJS & ExpressJS project | by Vaibhav Mehta | Medium

<https://mr-alien.medium.com/folder-structure-for-nodejs-expressjs-project-56be9ec35548>

[5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [11](#) How Should Your Vue Application Folder Structure Look? | by Ashutosh Batra | Medium

<https://medium.com/@betecieai/how-should-your-vue-application-folder-structure-look-3f027cfde3cb>

[12](#) [13](#) javascript - Configure VueJS pages (multiple pages in vue.config.js) MPA - Stack Overflow

<https://stackoverflow.com/questions/59059563/configure-vuejs-pages-multiple-pages-in-vue-config-js-mpa>

[14](#) [15](#) javascript - Where should be located a database config file in Node.js and Express? - Stack Overflow

<https://stackoverflow.com/questions/17053777/where-should-be-located-a-database-config-file-in-node-js-and-express>