Công nghệ phần mềm chuyên sâu - SE214.Q11
Bài tập thực hành lần 2

Docker hóa và Deploy ứng dụng từ source code có sẵn

03.01.2026

Nhóm 5

23521224 Trương Hoàng Phúc
23521736 Bùi Văn Tùng
23520657 Vũ Quốc Huy
23520466 Tạ Hoàng Hiệp
23520682 Đỗ Đình Khang
23520448 Nguyễn Văn Hào
23520557 Dương Quốc Hưng

Contents

1. Phân tích source code

1.1. Loại ứng dụng

- Ứng dụng web Full Stack

1.2. Công nghệ sử dụng

- Ngôn ngữ lập trình: JavaScript
- Framework:
  - Client: React
  - Server: Express

1.3. Cách chạy ứng dụng không sử dụng Docker

- Tải về source code của đồ án

1.3.1. Dành cho người dùng Windows

- Truy cập thư mục `scripts`
- Nhấn đúp chuột để chạy các script sau
  - `0_download_node.bat`: Tải NVM, Node Version Manager, trình quản lý phiên bản của Node trên Windows.
  - `1_install_node.bat`: Tải Node phiên bản 22.18.0 (LTS)
  - `2_install_dependencies.bat`: Cài đặt các package phụ thuộc mà client, server và hệ thống cần để vận hành.
  - `4_dev.bat`: Cho client lẫn server chạy cùng lúc

1.4. Port mà ứng dụng sử dụng

- Client: `5173`
- Server: `3000`

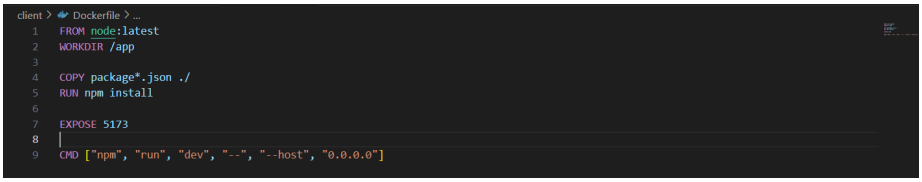
1.5. Ứng dụng có sử dụng

- Database: SQLite
- File upload: bài nộp của học viên sẽ được lưu tại `/server/uploads/submissions/{studentId}/{assignmentId}/`

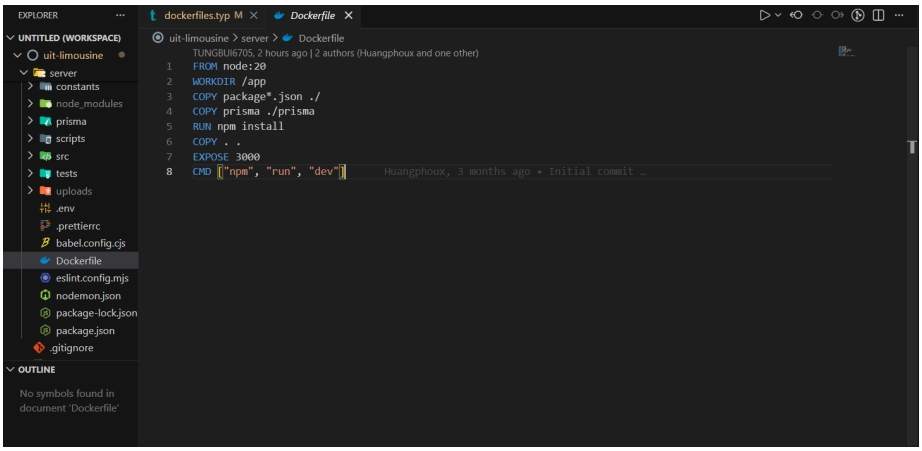
2. Docker hoá ứng dụng

2.1. Tạo tệp Dockerfile

- Tạo file Dockerfile tại thư mục gốc của project
- Dockerfile phải:
  - Cài đặt môi trường chạy phù hợp
  - Copy source code
  - Expose port cần thiết
  - Chạy được ứng dụng
  - Không hard-code thông tin nhạy cảm (mật khẩu, token).
- Các Dockerfile của hệ thống:
  - Client: `client\Dockerfile`

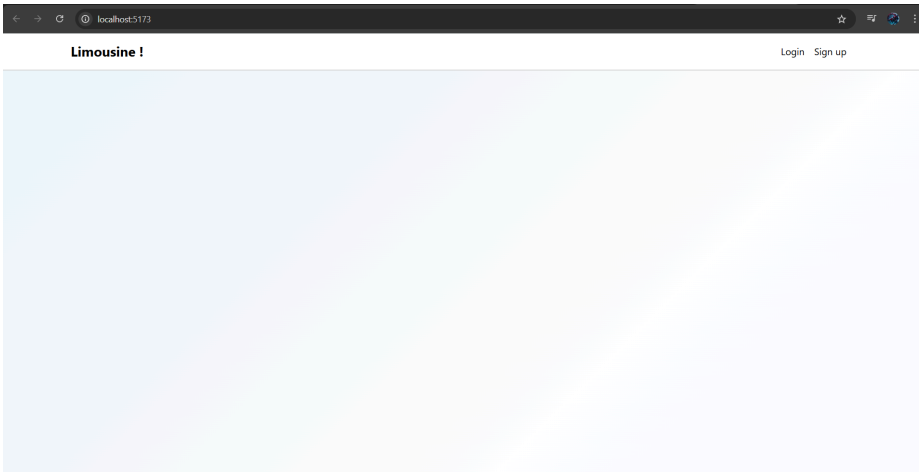


- Server: `server\Dockerfile`



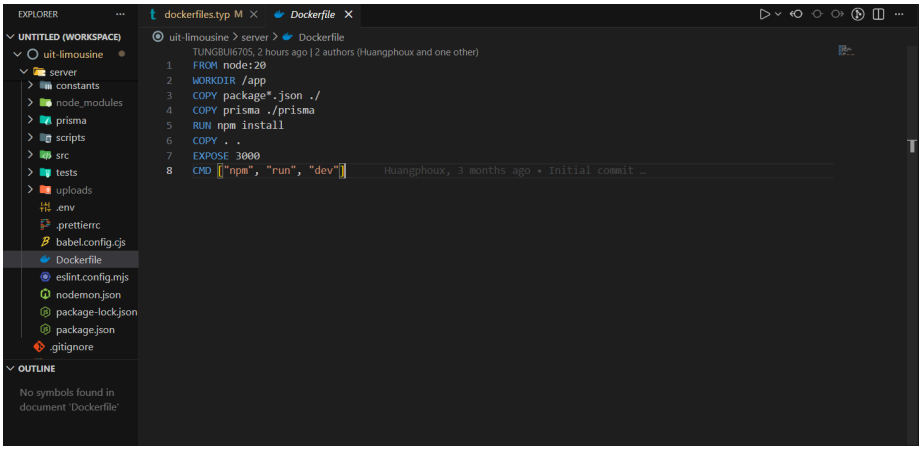
- Thư mục root không cần Dockerfile do đã có tệp `docker-compose.yml`.

2.1.1. Giải thích Dockerfile của Client



- FROM node:latest - Sử dụng image Node.js (nhãn latest) làm môi trường cơ bản để chạy và cài đặt các gói npm.
- WORKDIR /app - Thiết lập thư mục làm việc bên trong container là /app. Mọi lệnh tiếp theo sẽ thực thi từ thư mục này.
- COPY package*.json ./ - Sao chép package.json và package-lock.json vào container. Mục đích là tách bước cài đặt phụ thuộc ra thành một layer riêng để tận dụng cache khi không thay đổi các file này.
- RUN npm install - Chạy lệnh cài đặt tất cả dependencies được khai báo trong package.json. Sau bước này, node_modules sẽ được cài trong image.
- COPY . . - Copy toàn bộ mã nguồn từ thư mục gốc của dự án vào container.
- EXPOSE 5173 - Khai báo rằng ứng dụng sẽ lắng nghe cổng 5173 (metadata cho người đọc và công cụ; vẫn cần map port khi chạy container).
- CMD [“npm”, “run”, “dev”, “–”, “–host”, “0.0.0.0”] - Lệnh mặc định khi container khởi động: chạy dev server (ví dụ Vite) và truyền –host 0.0.0.0 để server lắng nghe mọi interface, cho phép truy cập từ bên ngoài container.

2.1.2. Giải thích Dockerfile của Server



- FROM node:20 - Sử dụng image Node.js phiên bản 20 làm môi trường cơ bản.
- WORKDIR /app - Đặt thư mục làm việc trong container là /app.
- COPY package*.json ./ - Copy các file package.json và package-lock.json từ thư mục gốc của dự án vào thư mục /app trong container.
- COPY prisma ./prisma - Copy thư mục prisma từ dự án vào container.
- RUN npm install - Cài đặt tất cả các dependencies được liệt kê trong package.json.
- COPY . . - Copy toàn bộ mã nguồn từ thư mục gốc của dự án vào container.
- EXPOSE 3000 - Mở cổng 3000 trong container để ứng dụng có thể được truy cập từ bên ngoài.
- CMD [“npm”, “run”, “dev”] - Chạy ứng dụng bằng lệnh npm run dev.

2.2. Build Docker Image

2.2.1. Client

- Thực hiện build image từ Dockerfile
  - Lệnh build: docker build -t client-app:latest .
  - Tên image đã tạo: client-app:latest

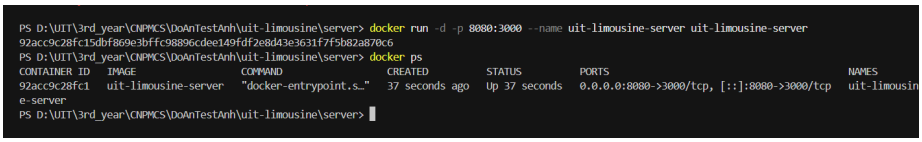
2.2.2. Server

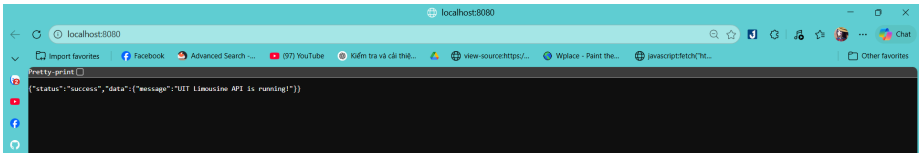
- Thực hiện build image từ Dockerfile
  - Lệnh build: docker build -t uit-limousine-server .
  - Tên image đã tạo: uit-limousine-server

2.3. Chạy Docker container

- Chạy container từ image đã build

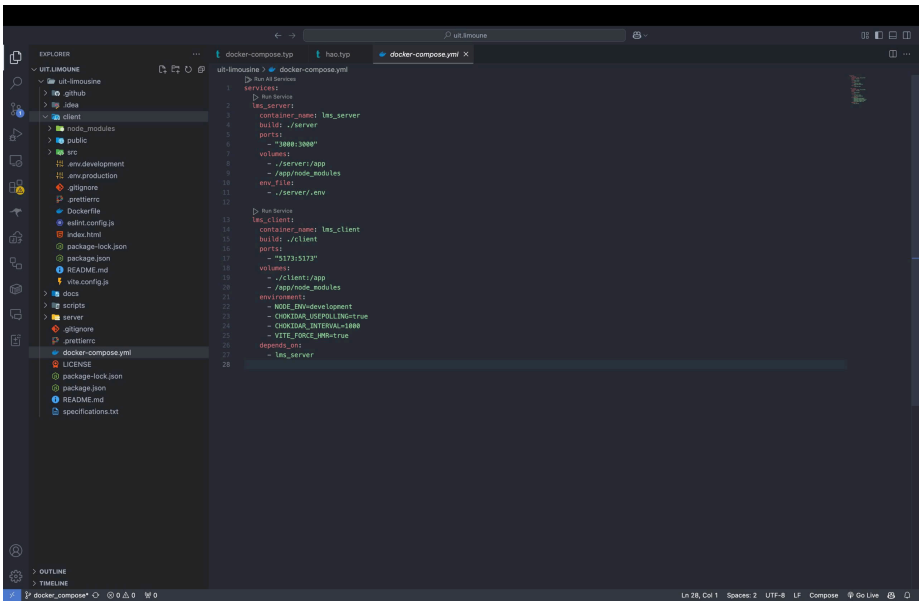
- Mapping port để có thể truy cập từ trình duyệt
- Kiểm tra ứng dụng hoạt động đúng
- Bắt buộc chụp màn hình ứng dụng đang chạy



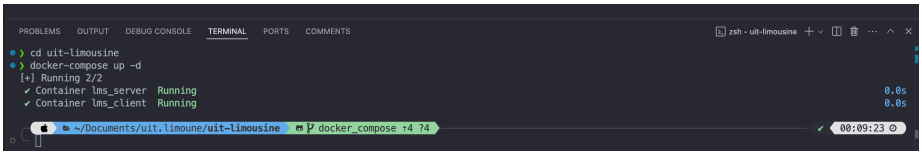


3. Deploy bằng Docker Compose

- Tạo tệp `docker-compose.yml`

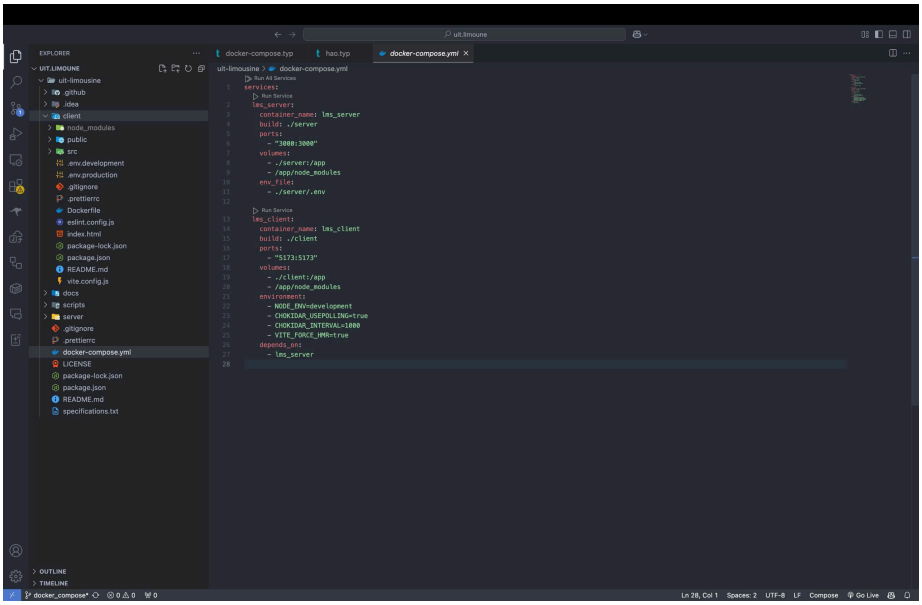


- Sử dụng Docker Compose để deploy:
  - Ứng dụng
  - (Nếu có) database hoặc service phụ trợ



- Hệ thống phải chạy được chỉ với một lệnh duy nhất: `docker-compose up -d`

3.0.1. Giải thích docker-compose.yml



- services: - Khai báo danh sách các service (container) sẽ được Docker Compose

quản lý và triển khai trong hệ thống.

- lms_server: - Định nghĩa service backend của hệ thống.
- container_name: lms_server - Đặt tên container backend là lms_server để dễ quản lý và nhận diện.
- build: ./server - Chỉ định Docker build image từ thư mục ./server. Docker sẽ sử dụng Dockerfile trong thư mục này để tạo image backend.
- ports: - Khai báo các cổng được ánh xạ giữa máy host và container.
- “3000:3000” - Ánh xạ cổng 3000 của máy host vào cổng 3000 của container,

cho phép truy cập backend thông qua http://localhost:3000.

- volumes: - Khai báo các volume dùng để chia sẻ dữ liệu giữa host và container.
- ./server:/app - Gắn thư mục mã nguồn ./server trên máy host vào thư mục /app trong container,

giúp đồng bộ mã nguồn trong quá trình phát triển.

- /app/node_modules - Tạo volume riêng cho thư mục node_modules trong container,

tránh việc thư mục node_modules trên host ghi đè lên node_modules trong container.

- env_file: - Khai báo file chứa các biến môi trường cho container.
- ./server/.env - Nạp các biến môi trường từ file .env trong thư mục server

vào container backend khi khởi động.

- lms_client: - Định nghĩa service frontend của hệ thống.
- container_name: lms_client - Đặt tên container frontend là lms_client.
- build: ./client - Chỉ định Docker build image từ thư mục ./client,

nơi chứa Dockerfile của frontend.
- ports: - Khai báo các cổng được ánh xạ cho frontend.
- "5173:5173" - Ánh xạ cổng 5173 của máy host vào cổng 5173 của container,

cho phép truy cập frontend qua địa chỉ http://localhost:5173.
- volumes: - Khai báo các volume dùng cho frontend.
- ./client:/app - Gắn thư mục mã nguồn frontend từ host vào thư mục /app trong container,

hỗ trợ cơ chế hot reload khi phát triển.
- /app/node_modules - Tạo volume riêng cho thư mục node_modules trong container frontend,

tránh xung đột dependency với máy host.
- environment: - Khai báo các biến môi trường cho container frontend.
- NODE_ENV=development - Thiết lập môi trường chạy ứng dụng Node.js ở chế độ development.
- CHOKIDAR_USEPOLLING=true - Bật chế độ polling để theo dõi thay đổi file,

giúp hot reload hoạt động ổn định trong Docker.
- CHOKIDAR_INTERVAL=1000 - Thiết lập chu kỳ polling là 1000ms (1 giây).
- VITE_FORCE_HMR=true - Ép Vite sử dụng cơ chế Hot Module Replacement

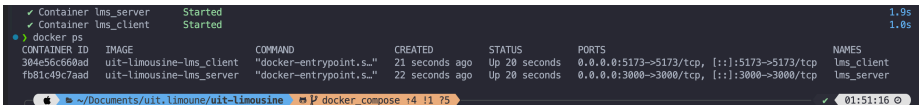
để tự động reload giao diện khi mã nguồn thay đổi.
- depends_on: - Khai báo sự phụ thuộc giữa các service.
- lms_server: - Chỉ định rằng service lms_client chỉ khởi động sau khi lms_server

đã được khởi động.

3.0.2. Lệnh để build image và tên image đã được tạo ra

- Lệnh build tất cả service: docker-compose build
- Kiểm tra image đã được tạo: docker images
- Tên image đã tạo: uit-limousine-lms_client và uit-limousine-lms_server

3.0.3. Chạy Docker container



4. Bản tự đánh giá và đóng góp cá nhân

4.1. Trương Hoàng Phúc 23521224

4.1.1. Công việc cá nhân

- Viết hai `Dockerfile` và `docker-compose.yml` cho đồ án của môn SE357 (Kỹ thuật phân tích yêu cầu), sau đó dùng repo của đồ án đó làm xuất phát điểm cho đồ án của môn SE214 (Công nghệ phần mềm chuyên sâu).
  - Commit tạo `Dockerfile` và `docker-compose.yml`
  - Commit đầu tiên của đồ án SE214
- Viết xuất phát điểm cho báo cáo: `bao_cao.typ`
- Điều hướng các thành viên còn lại trong nhóm cách làm bài

4.1.2. Kiến thức nắm rõ

- Cách viết Dockerfile

4.1.3. Khó khăn kỹ thuật

- Cách để sử dụng tính năng Hot Reload của Node trong container
  - Trên Linux, khi `docker compose up --build` xong, sửa tệp là Node trong container sẽ tự động khởi động lại
  - Do hệ thống lưu trữ tệp của Windows nên không thể làm việc trên ở trên Windows.
  - Cần phải sử dụng Dev Container
  - Không áp dụng trong môn này, vấn đề trên xảy ra ở đồ án của môn khác

4.1.4. Mức độ đóng góp trong bài tập

- Khoảng 80%
  - Chuẩn bị 3 tệp `Dockerfile` cho Client và Server, cùng với `docker-compose.yml` ngay từ đầu học kì
  - Viết mẫu báo cáo và trình bày dễ hiểu nhất để các thành viên còn lại có thể tự làm hoặc nhờ `ai` khác làm giùm

4.2. Bùi Văn Tùng 23521736

4.2.1. Công việc cá nhân

- Thiết lập Docker cho server trong dự án Limousine
- Build Docker image cho server bằng lệnh: docker build -t uit-limousine-server .

- Chạy container từ image với cấu hình port mapping: docker run -d -p 8080:3000 –name uit-limousine-server uit-limousine-server
- Kiểm tra ứng dụng hoạt động trên localhost:8080 và ghi lại kết quả

4.2.2. Kiến thức nắm rõ

- Hiểu được quy trình tạo Dockerfile, build image và chạy container
- Nắm vững cách sử dụng Docker trong quá trình deploy ứng dụng Node.js
- Biết cách ánh xạ port từ container ra máy tính để truy cập ứng dụng
- Hiểu mục đích của từng lệnh trong Dockerfile cho server

4.2.3. Khó khăn gặp phải

- Docker Desktop không được cài đặt hoặc chưa thêm đường dẫn vào PATH
- Giải quyết bằng cách cài đặt Docker Desktop từ trang chính thức và kiểm tra với docker –version
- Ban đầu gặp lỗi khi container không nhận diện lệnh npm run dev, nhưng sau khi chạy npm install trong Dockerfile đã khắc phục

4.2.4. Mức độ đóng góp

- 25% - Chịu trách nhiệm thiết lập và chạy server trên Docker, đảm bảo môi trường hoạt động đúng để các thành viên khác có thể test

4.3. Vũ Quốc Huy 23520657

4.3.1. Nội dung bản tự đánh giá cá nhân

4.3.2. Phần công việc cá nhân đã trực tiếp thực hiện trong nhóm

- Build Docker image cho server với lệnh:
  `docker build -t uit-limousine-server .`
- Chạy container từ image đã build:
  `docker run -d -p 8080:3000 --name uit-limousine-server uit-limousine-server`
- Kiểm tra ứng dụng hoạt động đúng trên localhost (http://localhost:8080).
- Đính kèm hình ảnh minh chứng vào báo cáo.

4.3.3. Phần kiến thức cá nhân nắm rõ nhất trong bài thực hành

- Nắm được khái niệm và ứng dụng của Docker.
- Docker image và container:
  - Hiểu cách tạo image từ Dockerfile.
  - Hiểu cách chạy container từ image.

- Quy trình deploy:
  - Từ viết Dockerfile, build image, chạy container, đến kiểm tra ứng dụng.

4.3.4. Một khó khăn kỹ thuật đã gặp trong quá trình thực hiện và cách giải quyết

- Khó khăn: Lỗi không nhận diện lệnh `docker` do Docker chưa được cài đặt hoặc chưa thêm vào `PATH`.
- Cách giải quyết:
  - Cài đặt Docker Desktop từ trang chính thức.
  - Thêm đường dẫn Docker vào biến môi trường `PATH`.
  - Kiểm tra lại bằng lệnh `docker --version`.

4.3.5. Tự đánh giá mức độ đóng góp của bản thân trong nhóm

- Mức độ đóng góp: 10%
- Đã hoàn thành công việc liên quan đến Dockerfile và kiểm chứng ứng dụng.

4.4. Tạ Hoàng Hiệp 23520466

4.4.1. Nội dung bản tự đánh giá cá nhân

4.4.2. Phần công việc cá nhân đã trực tiếp thực hiện trong nhóm

- Mục tiêu: Kiểm thử Dockerfile cho phần client của dự án UIT-LIMOUSINE, build image, chạy container độc lập và thu bằng chứng (ảnh chụp) để đính kèm vào báo cáo.
- Các bước thực hiện chính:
  - Kiểm tra cấu trúc thư mục client/ (có package.json, package-lock.json, src/, public/, index.html, vite.config.js, v.v.).
  - Kiểm tra nội dung Dockerfile hiện có và đề xuất chỉnh sửa (bổ sung COPY . . để đưa source vào image).
  - Thử build image và chạy container (nếu Docker có sẵn trên máy).
  - Khi gặp lỗi (Windows: 'docker' is not recognized…), thực hiện khắc phục (cài Docker Desktop / bật WSL2 / khởi động Docker).
  - Ghi lại các lệnh đã sử dụng, kết quả kiểm thử và chụp ảnh làm bằng chứng.
- Lệnh build image (tên image đã tạo):
  - Truy cập thư mục client/ docker build -t client-app:latest .
  - Tên image: client-app:latest
- Lệnh chạy container: docker run –rm -d -p 5173:5173 –name client-container client-app:latest
- Kiểm tra ứng dụng hoạt động:

- Mở trình duyệt: http://localhost:5173 (hoặc http://:5173 nếu test trên máy khác).
- Kiểm tra logs: docker logs -f client-container

4.4.3. Phần kiến thức cá nhân nắm rõ nhất trong bài thực hành

- Hiểu rõ workflow Docker cơ bản: Viết Dockerfile -> build image -> run container -> map port -> kiểm tra service.
- Hiểu vai trò từng phần trong Dockerfile
- Hiểu cách debug khi lỗi phát sinh

4.4.4. Một khó khăn kỹ thuật đã gặp trong quá trình thực hiện và cách giải quyết

- Khó khăn: Khi chạy lệnh build trên máy Windows báo lỗi: The term 'docker' is not recognized as the name of a cmdlet, function, script file, or operable program.
- Nguyên nhân: Docker CLI/Docker Desktop chưa được cài hoặc Docker Desktop chưa chạy / PATH chưa cập nhật.
- Cách giải quyết:
  - Cài đặt Docker Desktop từ trang chính thức.
  - Nếu hệ điều hành là Windows Home, bật WSL2 và cài WSL2 backend (chạy các lệnh DISM nếu cần, khởi động lại máy).
  - Mở Docker Desktop và chờ trạng thái Docker is running. Mở lại terminal PowerShell / CMD và kiểm tra

4.4.5. Tự đánh giá mức độ đóng góp của bản thân trong nhóm

- Mức độ đóng góp: 10%
- Đã hoàn thành công việc liên quan đến kiểm chứng việc chạy Dockerfile do @OopsNooob đã chỉnh sửa trên máy khác

4.5. Đỗ Đình Khang 23520682

4.5.1. Phần công việc cá nhân đã thực hiện

Trong bài thực hành này, em đã đảm nhận công việc viết Dockerfile cho phần client của ứng dụng. Trong đó:
- Xây dựng Dockerfile cho client sử dụng base image `node:latest`, thiết lập working directory là `/app`
- Cấu hình quá trình build bao gồm: copy file `package.json` và `package-lock.json`, chạy lệnh `npm install` để cài đặt dependencies
- Thiết lập command khởi động với `npm run dev -- --host 0.0.0.0` để cho phép truy cập từ bên ngoài container

- Kiểm thử Dockerfile bằng cách build image và chạy container độc lập để đảm bảo client hoạt động đúng

4.5.2. Phần kiến thức cá nhân nắm rõ nhất

Với bài thực hành, em đã nắm vững kiến thức về **Docker Image**. Docker image là một template chỉ đọc chứa tất cả những gì cần thiết để chạy một ứng dụng: code, runtime, system tools, libraries và dependencies. Nắm được quá trình tạo image từ Dockerfile thông qua các instruction như `FROM`, `WORKDIR`, `COPY`, `RUN`, `EXPOSE`, và `CMD`.

Ngoài ra, còn có cơ chế layering của Docker image - mỗi instruction trong Dockerfile tạo ra một layer mới, và Docker sử dụng caching để tối ưu hóa thời gian build. Ví dụ, trong Dockerfile của client, em copy file `package*.json` trước và chạy `npm install` riêng biệt, sau đó mới copy toàn bộ source code. Điều này giúp tận dụng Docker cache: khi source code thay đổi nhưng dependencies không đổi, Docker không cần chạy lại `npm install`, tiết kiệm đáng kể thời gian build.

Sự khác biệt giữa Docker image và container: image là blueprint không thay đổi, còn container là instance đang chạy của image. Một image có thể tạo ra nhiều container khác nhau.

4.5.3. Khó khăn kỹ thuật và cách giải quyết

Quá trình thực hiện Dockerfile cho client diễn ra khá thuận lợi và không gặp khó khăn đáng kể. Dockerfile của client tương đối đơn giản với các bước cơ bản: setup Node.js environment, install dependencies.

4.5.4. Tự đánh giá mức độ đóng góp

Mức độ đóng góp của bản thân trong nhóm là **10%**. Phần công việc của em tập trung vào việc containerize phần client của ứng dụng, đây là một phần quan trọng nhưng tương đối nhỏ so với toàn bộ hệ thống bao gồm server, database và docker-compose configuration.

4.6. Nguyễn Văn Hào 23520448

4.6.1. Công việc cá nhân

- Thiết lập file `docker-compose.yml` để quản lý đồng thời hai service: Client và Server
- Cấu hình Volume và biến môi trường (`CHOKIDAR_USEPOLLING`) để kích hoạt Hot Reload cho Frontend
- Build và chạy hệ thống bằng lệnh: `docker-compose up -d --build`

- Kiểm tra trạng thái container qua `docker ps` và xác nhận web chạy ổn định trên localhost

4.6.2. Kiến thức nắm rõ

- Hiểu rõ cấu trúc Docker Compose để điều phối nhiều container cùng lúc
- Nắm vững cơ chế Volume Mapping để đồng bộ mã nguồn giữa máy host và container
- Biết cách sử dụng Anonymous Volume để tránh xung đột thư mục `node_modules`
- Hiểu cách debug lỗi cơ bản thông qua log của container

4.6.3. Khó khăn gặp phải

- Gặp lỗi `EADDRINUSE: address already in use` do cổng 3000 bị chiếm dụng bởi tiến trình chạy ngầm
- Giải quyết bằng cách tắt các tiến trình Node.js đang chạy local hoặc dùng `docker-compose down` để giải phóng tài nguyên trước khi khởi động lại

4.6.4. Mức độ đóng góp

- 25% - Chịu trách nhiệm xây dựng file Docker Compose hoàn chỉnh, đảm bảo môi trường phát triển (Dev Environment) chạy đồng bộ cho cả Client và Server

4.7. Dương Quốc Hưng 23520557

4.7.1. Nội dung bản tự đánh giá cá nhân

- Công việc cá nhân đã trực tiếp thực hiện trong nhóm: Cấu hình docker-compose
- Phần kiến thức cá nhân nắm rõ nhất trong bài thực hành: Cách docker-compose hoạt động, cấu trúc một file docker-compose, các tính năng liên quan đến docker-compose như build, pull, env_file, environment, port, volumns, ..., một số câu lệnh liên quan: docker compose up, docker compose down.
- Một khó khăn kỹ thuật đã gặp trong quá trình thực hiện và cách giải quyết: Trong quá trình thực hiện có xuất hiện vấn đề về hết dung lượng để build image và chạy container. Quyết định giải quyết là cài lại docker với thư mục build image trên ổ đĩa khác ngoài ổ C.
- Tự đánh giá mức độ đóng góp của bản thân trong nhóm (%): Mức độ đóng góp khoảng 6%. Phần công việc chỉ tập trung vào cấu hình docker-compose sao cho cả client và server đều hoạt động sau khi Dockerfile của mỗi thành

phần được xây dựng. Về mặt kỹ thuật chỉ liên quan đến những tính năng và câu lệnh của docker compose.