Hướng dẫn Chi Tiết Tích Hợp Ollama, FastAPI và Next.js

Author: 8 Sync Dev

Mục lục

Tổng quan

Dự án này xây dựng một hệ thống chatbot thông minh tích hợp Ollama LLM, FastAPI backend và Next.js frontend. Hệ thống cho phép:

- **Xử lý truy vấn ngôn ngữ tự nhiên thành SQL**
- **Tùy chỉnh các tham số của mô hình LLM**
- **Giao diện người dùng thân thiện với Next.js**
- **Tích hợp Docker để dễ dàng triển khai**

Kiến trúc hệ thống

1. Backend Components

Ollama Service

- Chạy mô hình `qwen2.5-coder:0.5b`
- Xử lý ngôn ngữ tự nhiên
- Được containerized với Docker

FastAPI Service

- REST API endpoints
- Tích hợp Langchain
- Xử lý database operations
- Được containerized với Docker

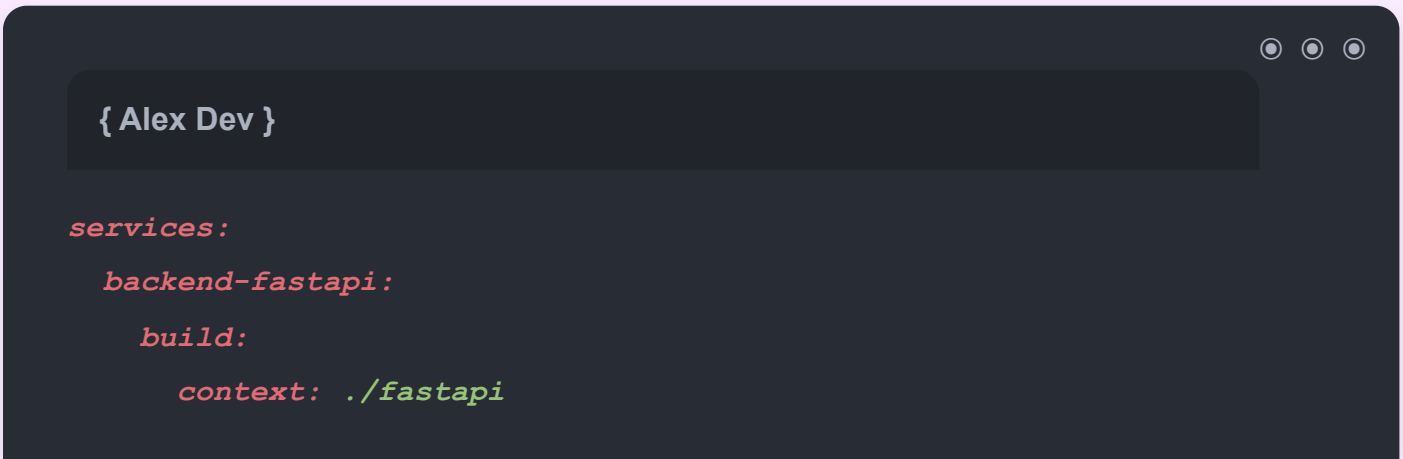
2. Frontend Component (Next.js)

- Server actions cho API calls
- UI components với Tailwind CSS
- Markdown rendering
- File upload handling

Cài đặt và Triển khai

1. Docker Configuration

Hệ thống sử dụng Docker Compose để quản lý các services:



```
      dockerfile: Dockerfile
    image: ollama-fastapi-backend
    ports:
      - 8000:8000
    networks:
      - app-net
    container_name: ollama-fastapi-backend
    volumes:
      - ./fastapi:/app
  ollama-server:
    build:
      context: ./ollama
      dockerfile: Dockerfile
    image: ollama-backend
    ports:
      - 11434:11434
    networks:
      - app-net
    entrypoint: ["/pull-qwen.sh"]
    container_name: ollama-backend
    volumes:
      - ollama-vol:/ollama

networks:
  app-net:
    driver: bridge

volumes:
  ollama-vol:
    driver: local
```

File compose.yml định nghĩa:

- **backend-fastapi**: FastAPI service chạy trên port 8000
- **ollama-server**: Ollama LLM service chạy trên port 11434
- Shared network `app-net` cho communication
- Volume `ollama-vol` để persist model data

2. Ollama Service Setup

{ Alex Dev }

```
FROM ollama/ollama

COPY ./pull-qwen.sh /pull-qwen.sh
```

Script để pull và khởi chạy model:

{ Alex Dev }

```
./bin/ollama serve &

pid=$!

sleep 5


echo "Pulling qwen2.5-coder model"
ollama pull qwen2.5-coder:0.5b


wait $pid
```

Quá trình:

1. **Khởi động Ollama server**
2. **Pull model qwen2.5-coder:0.5b**
3. **Đợi server hoàn tất**

3. FastAPI Service Setup

{ Alex Dev }

```
FROM python:3.11-slim

WORKDIR /app

COPY ./requirements.txt /app/requirements.txt
RUN pip3 install -r /app/requirements.txt

CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]
```

Chi tiết các thành phần

1. Frontend Implementation

Server Actions cho API calls:

{ Alex Dev }

```
"use server";

import { URL_BACKEND } from "./const";

interface ChatBotResponse {
  answer: string;
  data_db: any;
  evaluation: string;
}

interface ChatBotRequest {
```

```
  context?: string;
  file_path?: string;
  prompt: string;
  temperature?: number;
  top_k?: number;
  top_p?: number;
  num_ctx?: number;
  num_predict?: number;
  repeat_last_n?: number;
  repeat_penalty?: number;
}

export async function askChatBot(data: ChatBotRequest):
Promise<ChatBotResponse> {
  try {
    const response = await fetch(`${URL_BACKEND}/ask`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        accept: "application/json",
      },
      body: JSON.stringify({
        context: data.context || "",
        file_path: data.file_path || "",
        prompt: data.prompt,
        temperature: data.temperature || 0.1,
        top_k: data.top_k || 10,
        top_p: data.top_p || 0.95,
        num_ctx: data.num_ctx || 2048,
        num_predict: data.num_predict || 200,
        repeat_last_n: data.repeat_last_n || 64,
        repeat_penalty: data.repeat_penalty || 1.15
      }),
      cache: "no-store"
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const result = await response.json();
```

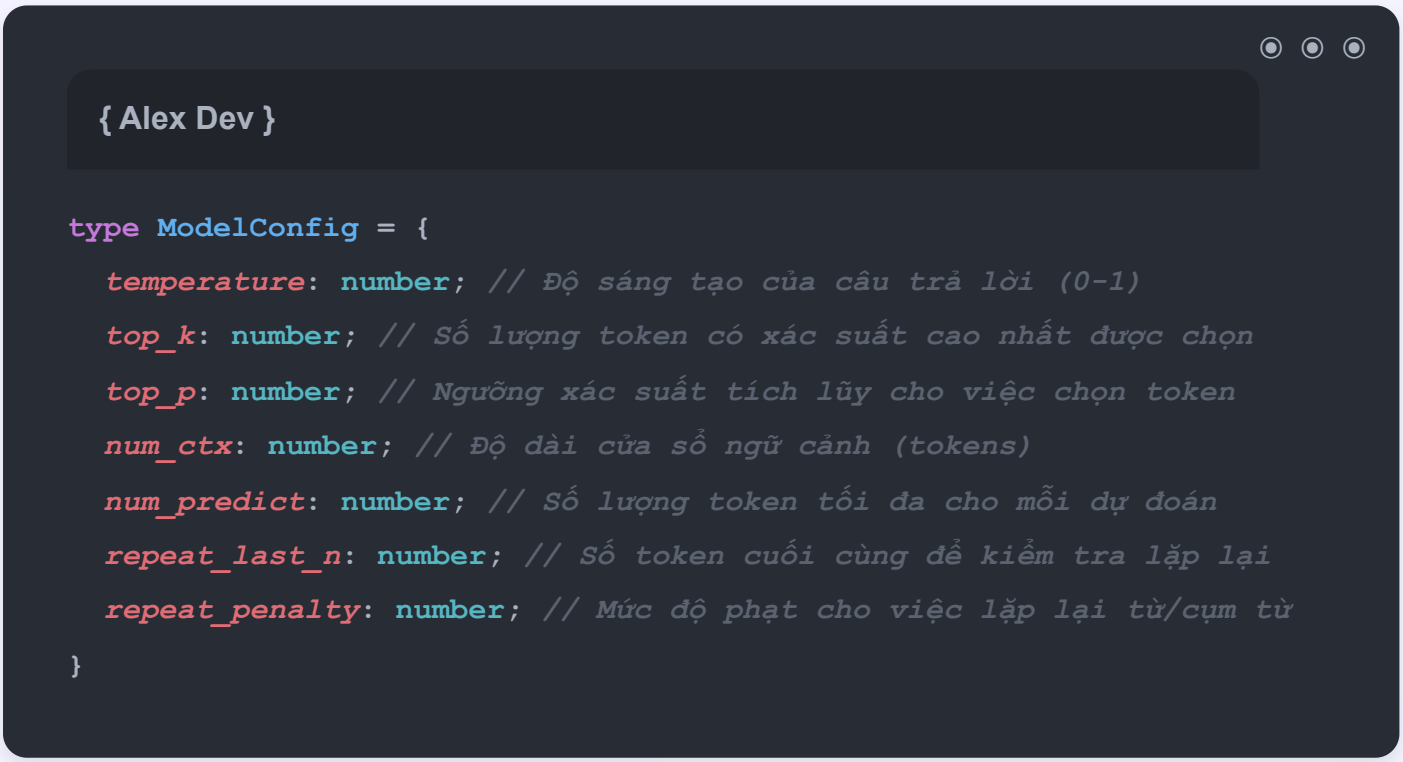
```
    return result as ChatBotResponse;
  } catch (error) {
    console.error("Error calling chat bot:", error);
    throw error;
  }
}
```

Xử lý:

- Gửi request đến FastAPI backend
- Truyền các tham số cấu hình LLM
- Error handling và response parsing

2. Model Configuration

Các tham số quan trọng của LLM:



Giải thích các tham số:

- **temperature**: Độ sáng tạo (0-1)
- **top_k**: Số token xác suất cao nhất

- **top_p**: Ngưỡng xác suất tích lũy
- **num_ctx**: Độ dài cửa sổ ngữ cảnh
- **num_predict**: Số token tối đa cho mỗi dự đoán
- **repeat_last_n**: Kiểm tra lặp lại
- **repeat_penalty**: Mức phạt cho lặp lại

Tối ưu và Mở rộng

1. Performance Optimization

- Sử dụng caching cho LLM responses
- Lazy loading cho UI components
- Optimized Docker images

2. Scaling Considerations

- Load balancing cho multiple Ollama instances
- Database sharding nếu cần
- Horizontal scaling cho FastAPI service

3. Security Best Practices

- Input validation
- Rate limiting
- SQL injection prevention
- Secure file uploads

Tài liệu tham khảo

Official Documentation

- Ollama Documentation

- FastAPI Documentation
- Next.js Documentation
- Docker Documentation

Additional Resources

- Langchain Python Documentation
- Qwen Model Documentation
- Docker Compose Best Practices
- FastAPI Best Practices

Community Resources

- Ollama GitHub Repository
- FastAPI GitHub Repository
- Next.js GitHub Repository

Related Articles

- Understanding LLM Parameters
- Docker Networking Guide
- FastAPI with Docker Guide