

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN  
THÔNG  
KHOA CÔNG NGHỆ THÔNG TIN 1**

000



# BÁO CÁO THỰC TẬP CƠ SỞ

## VIỆT FOOD - ỨNG DỤNG WEB ĐẶT ĐỒ ĂN VIỆT NAM TÍCH HỢP AI AGENT

**Người thực hiện** Lại Xuân Hiếu  
**Giảng Viên Hướng Dẫn** TS. Dương Trần Đức

Ngày 3 tháng 6 năm 2025

# Mục lục

<b>1 Giới thiệu tổng quan</b>	<b>3</b>
1.1 Giới thiệu dự án . . . . .	3
1.2 Công nghệ sử dụng . . . . .	4
1.2.1 Backend . . . . .	4
1.2.2 Frontend . . . . .	5
<b>2 Phân tích kiến trúc và thiết kế hệ thống</b>	<b>6</b>
2.1 Kiến trúc tổng thể . . . . .	6
2.1.1 Mô hình kiến trúc . . . . .	6
2.1.2 Sơ đồ kiến trúc . . . . .	6
2.2 Thiết kế module/component chính . . . . .	8
2.2.1 Controllers . . . . .	8
2.2.2 Services . . . . .	8
2.2.3 Middlewares . . . . .	9
2.2.4 Models . . . . .	9
2.2.5 Frontend Components . . . . .	9
2.2.6 Frontend Services và Hooks . . . . .	9
2.2.7 Quản lý Routing . . . . .	10
2.2.8 State Management . . . . .	10
2.3 Thiết kế cơ sở dữ liệu . . . . .	10
2.3.1 Các collection chính . . . . .	10
2.3.2 Cơ sở dữ liệu bổ sung . . . . .	12
2.3.3 Tối ưu hóa hiệu năng . . . . .	12
2.3.4 Bảo mật dữ liệu . . . . .	12
<b>3 Phân tích chi tiết các thành phần dự án</b>	<b>13</b>
3.1 Phân tích chi tiết các thành phần của dự án . . . . .	13
3.2 Frontend - Giao diện người dùng . . . . .	13
3.2.1 Cấu trúc component hiện đại . . . . .	13
3.2.2 Styling và Thiết kế . . . . .	13
3.2.3 Quản lý dữ liệu và State Management . . . . .	14
3.2.4 Routing và Navigation . . . . .	14
3.3 Backend - Kỹ thuật tối ưu hóa . . . . .	14
3.3.1 Redis Caching . . . . .	14
3.3.2 Khái niệm và lợi ích . . . . .	15
3.3.3 Triển khai trong dự án . . . . .	15
3.3.4 Cache dữ liệu món ăn . . . . .	15
3.3.5 Prime Cache khi khởi động . . . . .	15

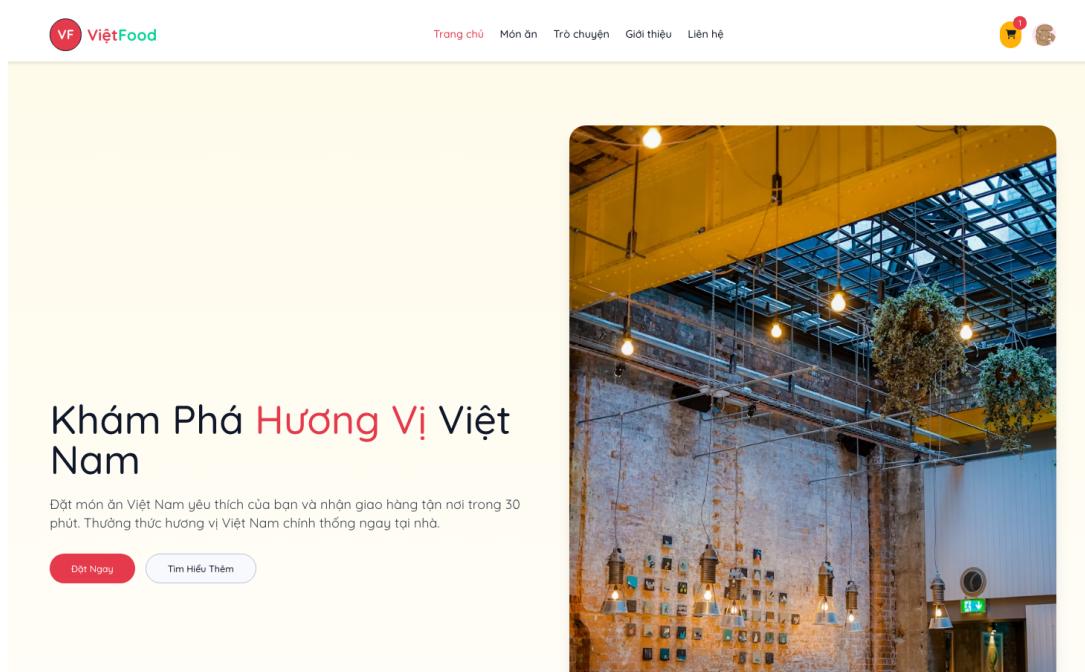
3.3.6	Invalidation cache . . . . .	15
3.4	Redis Stream . . . . .	16
3.4.1	Khái niệm và lợi ích . . . . .	16
3.4.2	Triển khai trong dự án . . . . .	16
3.5	HTTP Compression . . . . .	16
3.5.1	Khái niệm và lợi ích . . . . .	16
3.5.2	Triển khai trong dự án . . . . .	17
3.6	Tối ưu hóa hình ảnh với Sharp . . . . .	17
3.6.1	Khái niệm và lợi ích . . . . .	17
3.6.2	Triển khai trong dự án . . . . .	17
3.7	Hệ thống Agent thông minh . . . . .	18
3.7.1	Giới thiệu về Agent . . . . .	18
3.7.2	Lưu trữ message và quản lý context . . . . .	18
3.7.3	Quản lý context hội thoại . . . . .	19
3.7.4	Vấn đề đang gặp phải . . . . .	19
<b>4</b>	<b>Dánh giá hiệu quả và đề xuất cải tiến</b>	<b>20</b>
4.1	Thử nghiệm hiệu suất hệ thống . . . . .	20
4.2	Phương pháp thử nghiệm . . . . .	20
4.3	Kết quả thử nghiệm . . . . .	21
4.3.1	Phiên bản không nén dữ liệu . . . . .	21
4.3.2	Phiên bản có nén dữ liệu . . . . .	22
4.4	Phân tích kết quả . . . . .	22
4.4.1	So sánh hiệu suất . . . . .	22
4.4.2	Nhận xét . . . . .	22
4.5	Kết luận . . . . .	23
<b>5</b>	<b>Kết luận và hướng phát triển</b>	<b>24</b>
5.1	Tóm tắt kết quả phân tích . . . . .	24
5.2	Bài học kinh nghiệm . . . . .	24
5.2.1	Thiết kế hướng hiệu suất ngay từ đầu . . . . .	25
5.2.2	Caching là chìa khóa . . . . .	25
5.2.3	Phân tách xử lý không đồng bộ . . . . .	25
5.2.4	Tối ưu tài nguyên tĩnh . . . . .	25
5.2.5	Cân bằng giữa hiệu suất và phức tạp . . . . .	25
5.3	Hướng phát triển tiềm năng . . . . .	25
5.3.1	Kiến trúc Microservices . . . . .	25
5.3.2	Caching đa tầng . . . . .	26
5.3.3	Xử lý ảnh nâng cao . . . . .	26
5.3.4	Tích hợp GraphQL . . . . .	26
5.3.5	Giám sát và phân tích hiệu suất . . . . .	26
5.3.6	Serverless Computing . . . . .	27

# Chương 1

## Giới thiệu tổng quan

### 1.1 Giới thiệu dự án

Việt Food là một ứng dụng web fullstack với kiến trúc client-server nhằm cung cấp nền tảng đặt món ăn Việt Nam trực tuyến. Ứng dụng được phát triển với mục đích tạo ra trải nghiệm người dùng hiện đại, thân thiện, đồng thời đảm bảo hiệu suất cao và khả năng mở rộng. Với giao diện người dùng trực quan và hệ thống backend mạnh mẽ, Việt Food nhằm mục đích kết nối người dùng với các món ăn truyền thống và hiện đại của ẩm thực Việt Nam.



Hình 1.1: Giao diện trang chủ ứng dụng Việt Food

Hình 1.1 thể hiện giao diện chính của ứng dụng, nơi người dùng có thể xem các món ăn nổi bật, tìm kiếm món ăn và dễ dàng đặt hàng.

The left side of the image shows a search interface for dishes. It includes a sidebar with filters for category (Bò lợn), price range (0 - 1000k), and other criteria like 'Danh mục' (Beef). Below this is a grid of dish cards, each with a thumbnail, name, rating, and price. For example, 'Cháo tôm' costs 40.000đ and 'Bánh bột lọc' costs 60.000đ. Each card has a 'Thêm vào giỏ' (Add to cart) button.

The right side shows a detailed view of a dish, specifically 'Bánh bột lọc', which is described as 'Bánh bột lọc trong veo, dai mềm và nhân bột thơm đậm vị thịt bò'. It includes a large thumbnail, a detailed description, and a 'Thêm vào giỏ' button.

Hình 1.2: Trang đặt món ăn

This screenshot shows a user profile page. At the top, there's a placeholder for a profile picture with the name 'Lại Xuân Hiếu'. Below it, there are sections for 'Thông tin tài khoản' (Account info) and 'Địa chỉ giao hàng' (Delivery address). The account info includes 'Họ và tên' (Name: Lại Xuân Hiếu), 'Số điện thoại' (Phone: 0912345678), and 'Ghi chú' (Notes: Khoác). The delivery address lists two locations: 'Lại Xuân Hiếu' and 'Lại Xuân Hiếu' with the same details. At the bottom, there are links for 'Chỉnh sửa mật khẩu' (Change password), 'Điều khoản sử dụng' (Terms of service), and 'Sitemap'.

Hình 1.4: Trang thông tin cá nhân

This screenshot shows a dashboard titled 'Quản lý Vietnamese Food'. It features several summary boxes: 'Tổng quan' (Overview) showing 0 đơn hàng (orders), 41 người dùng (users), and 100 sản phẩm (products); 'Đơn hàng gần đây' (Recent orders) listing one order for 195.000đ; and 'Top sản phẩm bán chạy' (Top-selling products) showing a list of items like 'Trà Đá', 'Cà Phê Sữa Đá', and 'Nước Mía'. On the left, there's a sidebar with navigation links for 'Thống kê', 'Quản lý sản phẩm', 'Đơn hàng', 'Người dùng', 'Sản phẩm', and 'Thiết lập'.

Hình 1.5: Trang quản trị viên

## 1.2 Công nghệ sử dụng

Dự án được xây dựng bằng các công nghệ hiện đại:

### 1.2.1 Backend

- Node.js và Express:** Framework cho phát triển ứng dụng web server-side.
- TypeScript:** Ngôn ngữ lập trình đảm bảo tính mạnh mẽ và an toàn cho mã nguồn.
- MongoDB:** Cơ sở dữ liệu NoSQL phục vụ lưu trữ dữ liệu phi quan hệ.
- Redis:** Hệ thống cache bộ nhớ để tối ưu hóa hiệu suất truy vấn.

- **Redis Stream:** Xử lý các tin nhắn và sự kiện real-time.
- **Compression:** Middleware giảm kích thước phản hồi HTTP.
- **Sharp:** Thư viện xử lý và tối ưu hóa hình ảnh.
- **Elasticsearch:** Công cụ tìm kiếm và đánh chỉ mục dữ liệu nhanh chóng.
- **Gemini AI:** Tích hợp trí tuệ nhân tạo cho các tính năng thông minh.

### 1.2.2 Frontend

- **React:** Thư viện JavaScript để xây dựng giao diện người dùng tương tác.
- **TypeScript:** Dảm bảo type safety và khả năng bảo trì cho mã nguồn frontend.
- **Tailwind CSS:** Framework CSS tiện ích giúp phát triển UI nhanh chóng và đồng nhất.
- **React Hook Form:** Quản lý biểu mẫu với hiệu suất cao.
- **Zod:** Thư viện xác thực dữ liệu với TypeScript.

# Chương 2

## Phân tích kiến trúc và thiết kế hệ thống

### 2.1 Kiến trúc tổng thể

Việt Food áp dụng kiến trúc client-server hiện đại, với sự phân chia rõ ràng giữa frontend (React) và backend (Node.js/Express). Kiến trúc này tạo nên một hệ thống linh hoạt và mạnh mẽ, mang lại nhiều lợi ích:

- Phát triển độc lập:
- Khả năng mở rộng
- Tái sử dụng API
- Bảo mật tốt hơn
- Hiệu suất tối ưu

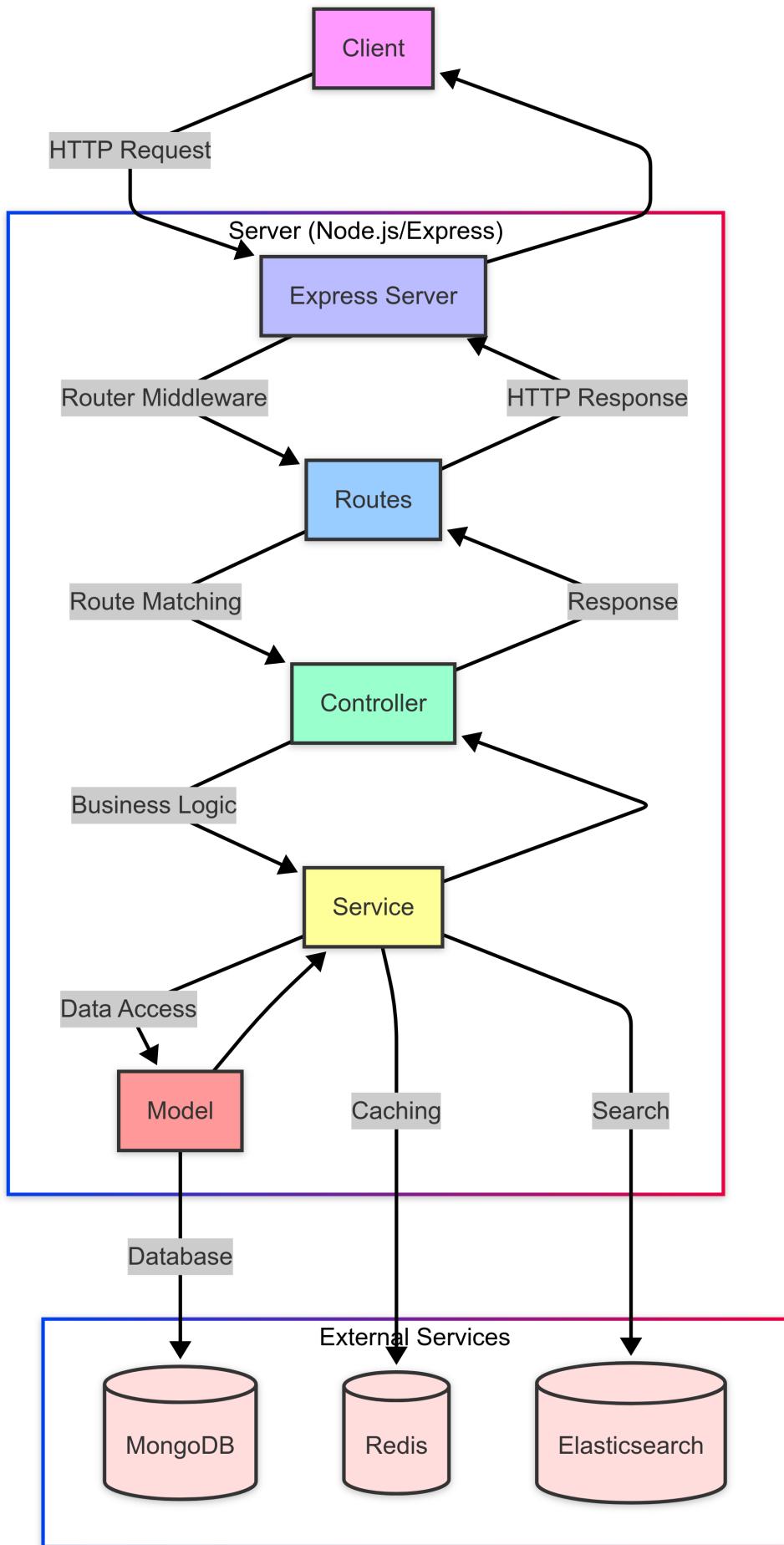
#### 2.1.1 Mô hình kiến trúc

Hệ thống áp dụng kiến trúc MVC (Model-View-Controller) kết hợp với các dịch vụ (Services):

- **Model:** Đại diện cho cấu trúc dữ liệu và xử lý logic liên quan đến dữ liệu.
- **View:** Được thực hiện ở phía client, hiển thị thông tin cho người dùng.
- **Controller:** Xử lý các yêu cầu từ client, tương tác với các service và trả về kết quả.
- **Service:** Chứa logic nghiệp vụ, tương tác với database và các thành phần bên ngoài.

#### 2.1.2 Sơ đồ kiến trúc

Kiến trúc hệ thống được thiết kế với nhiều lớp:



Hình 2.1: Sơ đồ kiến trúc tổng thể của hệ thống Việt Food

## 2.2 Thiết kế module/component chính

Cấu trúc dự án được tổ chức thành các module chức năng riêng biệt, gồm cả phần frontend và backend:

### 2.2.1 Controllers

Xử lý các request từ client và điều phối các thao tác:

- **DishController:** Quản lý thông tin món ăn
- **CategoryController:** Quản lý danh mục
- **AuthController:** Xử lý xác thực người dùng
- **OrderController:** Xử lý đơn hàng
- **PaymentController:** Xử lý thanh toán
- **UserController:** Xử lý thông tin người dùng
- **UserProfileController:** Xử lý thông tin cá nhân
- **AgentController:** Xử lý tin nhắn chat
- **DashboardController:** Xử lý thống kê

### 2.2.2 Services

Chứa logic nghiệp vụ chính:

- **DishService:** Xử lý logic liên quan đến món ăn, bao gồm caching với Redis
- **CategoryService:** Quản lý danh mục món ăn
- **AuthService:** Xử lý đăng nhập, đăng ký và xác thực
- **SearchService:** Tương tác với Elasticsearch để tìm kiếm
- **PaymentService:** Xử lý thanh toán
- **OrderService:** Xử lý đơn hàng
- **UserService:** Xử lý thông tin người dùng
- **AgentService:** Xử lý tin nhắn chat
- **DashboardService:** Xử lý thống kê
- **MessageWorkerService:** Xử lý tin nhắn với Redis Stream
- **MessageService:** Xử lý tin nhắn

### 2.2.3 Middlewares

Xử lý các tác vụ trung gian:

- **Authentication:** Kiểm tra và xác thực người dùng
- **ImageProcessor:** Xử lý và tối ưu hóa hình ảnh với Sharp

### 2.2.4 Models

Mô tả cấu trúc dữ liệu chính trong backend:

- **DishModel:** Mô hình dữ liệu cho món ăn
- **CategoryModel:** Mô hình dữ liệu cho danh mục
- **UserModel:** Mô hình dữ liệu người dùng
- **MessageModel:** Mô hình dữ liệu tin nhắn
- **OrderModel:** Mô hình dữ liệu đơn hàng
- **PaymentModel:** Mô hình dữ liệu thanh toán

### 2.2.5 Frontend Components

Frontend được xây dựng theo kiến trúc component-based với React, tổ chức thành các thành phần:

- **UI Components:** Các component giao diện cơ bản, được xây dựng trên Radix UI và tùy chỉnh với Tailwind CSS
- **Layout Components:** Quản lý bố cục chung của ứng dụng
- **Page Components:** Tương ứng với các trang trong ứng dụng

### 2.2.6 Frontend Services và Hooks

Phần frontend sử dụng các service và custom hooks để quản lý logic:

- **API Services:** Xử lý các tương tác với backend thông qua axios
- **Context API:** Sử dụng React Context để quản lý trạng thái toàn cục như xác thực người dùng (CartContext, AuthContext)
- **Custom Hooks:** Các hooks chuyên biệt như useAuth, useCart, useProfile để đơn giản hóa quản lý trạng thái và data fetching
- **Local Storage:** Lưu trữ tokens xác thực và thông tin session người dùng

## 2.2.7 Quản lý Routing

Việt Food sử dụng thư viện wouter để quản lý routing trong ứng dụng, với các route chính:

- **Public Routes:** Trang chủ, danh sách món ăn, chi tiết món ăn, tìm kiếm
- **Protected Routes:** Giỏ hàng, quản lý đơn hàng, tài khoản cá nhân
- **Admin Routes:** Quản lý món ăn, danh mục, người dùng và thống kê

## 2.2.8 State Management

Quản lý trạng thái trong ứng dụng được thực hiện thông qua:

- **React Context:** Quản lý trạng thái toàn cục như thông tin người dùng, giỏ hàng
- **Local Component State:** Quản lý trạng thái độc lập của từng component

## 2.3 Thiết kế cơ sở dữ liệu

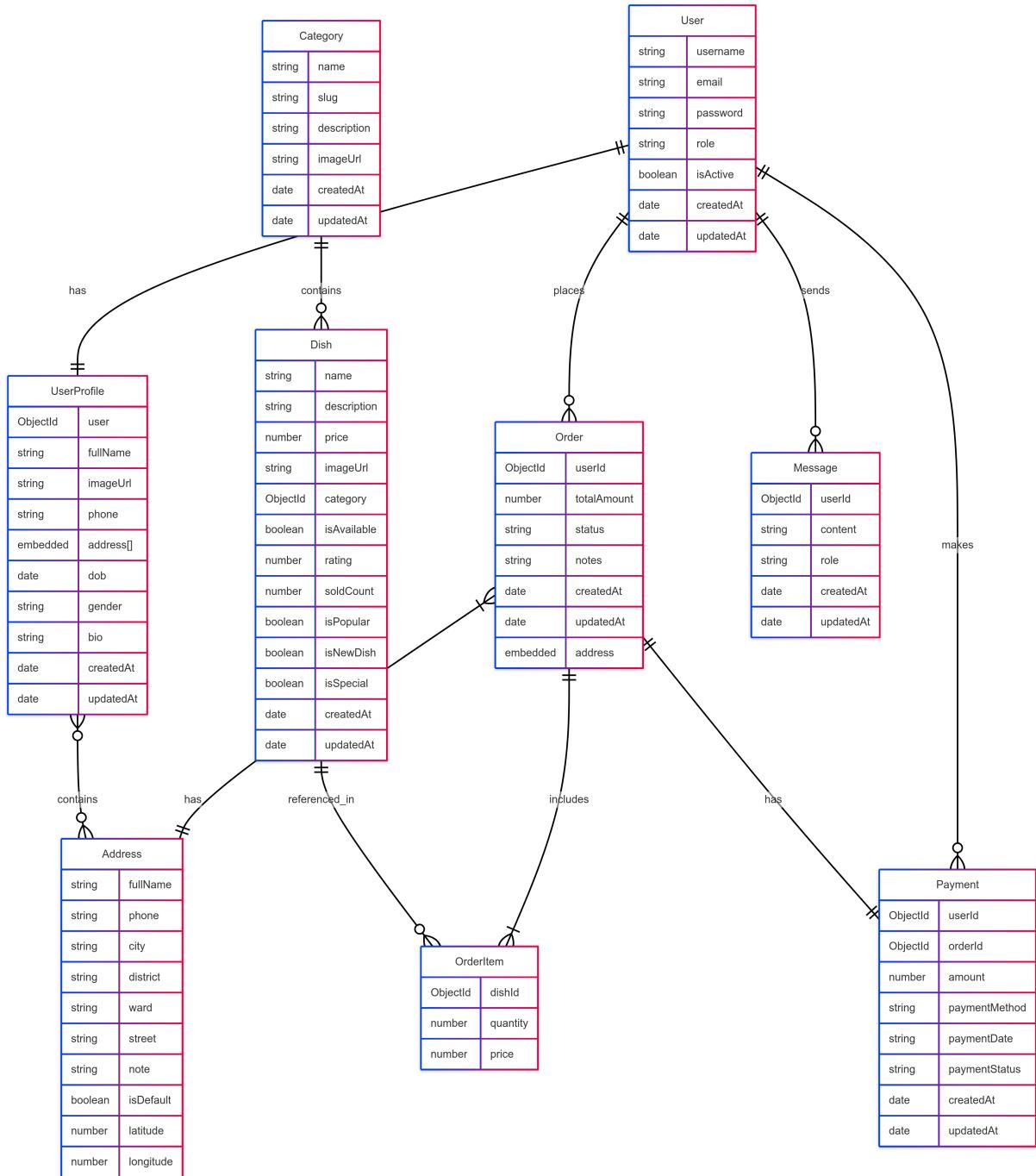
Hệ thống sử dụng MongoDB làm cơ sở dữ liệu chính với thiết kế schema linh hoạt, phù hợp với đặc thù của ứng dụng đặt đồ ăn. Các collection chính được thiết kế như sau:

### 2.3.1 Các collection chính

- **Users:** Quản lý thông tin người dùng
  - `username`, `email`: Thông tin đăng nhập (duy nhất)
  - `password`: Mật khẩu đã được mã hóa
  - `role`: Phân quyền người dùng (USER/ADMIN)
  - `isActive`: Trạng thái tài khoản
- **Dishes:** Quản lý thông tin món ăn
  - `name`, `description`: Thông tin cơ bản
  - `price`, `imageUrl`: Giá và hình ảnh
  - `category`: Tham chiếu đến danh mục
  - `rating`, `soldCount`: Đánh giá và số lượng bán
  - `isAvailable`, `isPopular`, `isNewDish`, `isSpecial`: Các cờ trạng thái
- **Orders:** Quản lý đơn hàng
  - `userId`: Người đặt hàng
  - `orderItems`: Danh sách món đã đặt
  - `totalAmount`: Tổng tiền
  - `status`: Trạng thái đơn hàng
  - `address`: Thông tin giao hàng

- **Messages:** Lưu trữ lịch sử chat

- **userId:** Người gửi
- **content:** Nội dung tin nhắn
- **role:** Loại người gửi (user/assistant)



Hình 2.2: Sơ đồ cơ sở dữ liệu của hệ thống Việt Food

### **2.3.2 Cơ sở dữ liệu bổ sung**

Bên cạnh MongoDB, hệ thống còn tích hợp:

- **Redis:**

- Caching dữ liệu thường truy cập (thông tin món ăn, danh mục)
- Quản lý phiên đăng nhập
- Xử lý tin nhắn real-time thông qua Redis Stream

- **Elasticsearch:**

- Dánh chỉ mục dữ liệu món ăn
- Hỗ trợ tìm kiếm full-text với khả năng ranking
- Tìm kiếm gợi ý (suggestions) khi người dùng nhập

### **2.3.3 Tối ưu hóa hiệu năng**

- **Indexing:** Các trường thường xuyên truy vấn được đánh index
- **Caching:** Sử dụng Redis để giảm tải cho database chính
- **Phân trang:** Áp dụng cho danh sách món ăn và lịch sử đơn hàng
- **Aggregation:** Tổng hợp dữ liệu thống kê hiệu quả

### **2.3.4 Bảo mật dữ liệu**

- Mã hóa mật khẩu bằng bcrypt
- Xác thực JWT cho API
- Phân quyền chi tiết theo vai trò
- Kiểm tra quyền truy cập ở tầng middleware

# Chương 3

## Phân tích chi tiết các thành phần dự án

### 3.1 Phân tích chi tiết các thành phần của dự án

Phần này phân tích chi tiết các thành phần frontend và backend của dự án Việt Food. Em sẽ đi sâu vào các tính năng nổi bật, kỹ thuật triển khai, và những giải pháp được áp dụng để đảm bảo hiệu suất và trải nghiệm người dùng tốt.

### 3.2 Frontend - Giao diện người dùng

#### 3.2.1 Cấu trúc component hiện đại

Frontend của Việt Food được xây dựng bằng React và TypeScript, áp dụng một cấu trúc có tính mô-đun cao và dễ bảo trì. Hệ thống component được tổ chức theo các lớp:

1. **Chia nhỏ component:** Các thành phần UI được tách thành các component nhỏ để tăng khả năng tái sử dụng.
2. **Cô lập logic:** Logic được tách biệt với giao diện thông qua các custom hooks.

#### 3.2.2 Styling và Thiết kế

Việt Food sử dụng Tailwind CSS kết hợp với các component từ Radix UI để tạo ra một hệ thống giao diện nhất quán và hiện đại:

- **Utility-first CSS:** Sử dụng các lớp của Tailwind giúp phát triển nhanh và nhất quán.
- **Theme Configuration:** Hệ thống màu sắc, typography, và spacing được cấu hình trong tailwind.config.ts.
- **Component Libraries:** Sử dụng Radix UI làm nền tảng, tùy chỉnh với Tailwind CSS.
- **Responsive Design:** Ứng dụng được thiết kế đáp ứng trên các kích thước màn hình khác nhau.

### 3.2.3 Quản lý dữ liệu và State Management

Việt Food sử dụng một hỗn hợp các công cụ để quản lý dữ liệu và trạng thái ứng dụng:

- **React Context API:** Quản lý trạng thái toàn cục của ứng dụng như xác thực người dùng (auth), giỏ hàng (cart).
- **Custom Hooks:** Tạo ra các hooks chuyên biệt như useAuth, userProfile để truy cập và cập nhật trạng thái.
- **Zod:** Validation dữ liệu chặt chẽ với TypeScript.
- **Local Storage:** Lưu trữ access token và refresh token trên client.

Hệ thống quản lý giỏ hàng sử dụng Context API với các tính năng chính:

- Quản lý danh sách sản phẩm trong giỏ hàng
- Theo dõi trạng thái mở/đóng giỏ hàng
- Các thao tác thêm, xóa, cập nhật số lượng sản phẩm
- Tính toán tự động tổng tiền và số lượng sản phẩm

Custom hook `useCart` được sử dụng để truy cập ngẫu nhiên giỏ hàng từ bất kỳ component nào trong ứng dụng một cách dễ dàng.

### 3.2.4 Routing và Navigation

Việt Food sử dụng thư viện wouter cho routing nhẹ nhàng và hiệu quả:

- **Lazy Loading:** Các trang được tải lazy để giảm kích thước bundle ban đầu.
- **Route Protection:** Các route được bảo vệ dựa trên vai trò người dùng.
- **URL Params và Query Strings:** Xử lý các tham số đường dẫn và query string cho tìm kiếm và lọc.

Hệ thống routing được cấu hình với các tính năng chính:

- Định nghĩa các route chính: Trang chủ, Danh sách món ăn, Chi tiết món, Giỏ hàng
- Sử dụng lazy loading để tải các component khi cần thiết
- Bảo vệ các route yêu cầu xác thực
- Xử lý các tham số động trong URL

## 3.3 Backend - Kỹ thuật tối ưu hóa

### 3.3.1 Redis Caching

Hệ thống sử dụng Redis làm hệ thống cache để tối ưu hóa thời gian truy vấn dữ liệu. Redis được cấu hình và khởi tạo trong file `config/redis.ts`. Cache dữ liệu món ăn là loại dữ liệu được truy cập thường xuyên nhưng ít thay đổi, nên việc cache là rất hiệu quả. Hệ thống sử dụng kỹ thuật "prime cache" khi khởi động server - tức là nạp trước dữ liệu phổ biến vào cache. Khi dữ liệu thay đổi, cache cần được cập nhật hoặc xóa để tránh dữ liệu không đồng bộ.

### 3.3.2 Khái niệm và lợi ích

Redis là hệ thống lưu trữ dữ liệu key-value trong bộ nhớ, có khả năng lưu trữ nhiều kiểu dữ liệu khác nhau. Trong Việt Food, Redis được sử dụng chủ yếu làm hệ thống cache để tối ưu hóa thời gian truy vấn dữ liệu:

- **Tốc độ truy xuất nhanh:** Dữ liệu được lưu trong bộ nhớ, giảm độ trễ so với truy vấn database
- **Giảm tải cho database chính:** Các truy vấn phổ biến được cache, giảm số lượng truy cập vào MongoDB
- **Tăng khả năng phục vụ đồng thời:** Hệ thống có thể xử lý nhiều request hơn trong cùng một thời điểm

### 3.3.3 Triển khai trong dự án

Trong Việt Food, Redis được cấu hình và khởi tạo trong file `config/redis.ts`:

Cấu hình Redis được thiết lập thông qua thư viện `ioredis`, kết nối đến máy chủ Redis thông qua các biến môi trường. Hệ thống ghi log khi kết nối thành công hoặc gặp lỗi.

### 3.3.4 Cache dữ liệu món ăn

Dữ liệu món ăn là loại dữ liệu được truy cập thường xuyên nhưng ít thay đổi, nên việc cache là rất hiệu quả. Trong `DishService`, các phương thức tìm kiếm món ăn đều áp dụng caching:

Phương thức 'findById' triển khai cơ chế cache-aside pattern. Khi nhận yêu cầu, hệ thống kiểm tra dữ liệu trong cache trước, nếu không có mới truy vấn database. Dữ liệu được lưu trong cache 24 giờ.

### 3.3.5 Prime Cache khi khởi động

Hệ thống áp dụng kỹ thuật "prime cache" khi khởi động server, nạp trước tất cả món ăn đang có sẵn vào Redis. Quá trình này giúp giảm thời gian phản hồi cho các truy vấn đầu tiên bằng cách đảm bảo dữ liệu thường dùng luôn sẵn sàng trong bộ nhớ đệm. Khi server khởi động, hệ thống thực hiện các bước sau:

1. Kết nối cơ sở dữ liệu chính
2. Thiết lập kết nối với Elasticsearch
3. Gọi phương thức 'primeAllDishesCache()' để nạp trước dữ liệu món ăn vào Redis

### 3.3.6 Invalidation cache

Khi dữ liệu thay đổi, cache cần được cập nhật hoặc xóa để tránh dữ liệu không đồng bộ. Việt Food xử lý vấn đề này bằng cách xóa các key cache liên quan khi có thay đổi:

Khi dữ liệu món ăn thay đổi, hệ thống tự động xóa các key cache liên quan để đảm bảo tính nhất quán dữ liệu. Quá trình này sử dụng pipeline để tối ưu hiệu năng khi xóa nhiều key cùng lúc, đảm bảo dữ liệu hiển thị cho người dùng luôn được cập nhật mới nhất.

## 3.4 Redis Stream

### 3.4.1 Khái niệm và lợi ích

Redis Stream là một cấu trúc dữ liệu mới trong Redis, cho phép lưu trữ và quản lý các dòng sự kiện theo thời gian thực, tương tự như một message broker:

- **Xử lý tin nhắn không đồng bộ:** Các thành phần khác nhau có thể giao tiếp mà không cần chờ đợi nhau
- **Mô hình publish-subscribe:** Phân tách giữa producer và consumer
- **Khả năng mở rộng:** Nhiều consumer có thể xử lý cùng một stream
- **Độ tin cậy:** Tin nhắn được lưu trữ lâu dài và có thể được xử lý lại nếu cần

### 3.4.2 Triển khai trong dự án

Trong Việt Food, Redis Stream được sử dụng chủ yếu để xử lý tin nhắn. Đầu tiên, stream key được định nghĩa trong file cấu hình Redis:

Hệ thống định nghĩa một hằng số MESSAGE\_STREAM\_KEY với giá trị "message\_stream" để xác định Redis Stream dùng cho việc xử lý tin nhắn.

MessageWorkerService xử lý tin nhắn từ stream:

Lớp MessageWorkerService đảm nhận việc xử lý tin nhắn từ Redis Stream với hai phương thức chính:

1. `createConsumerGroup()`: Thiết lập nhóm người tiêu dùng (consumer group) cho Redis Stream nếu chưa tồn tại
2. `processStreamMessages()`: Vòng lặp vô hạn đọc và xử lý tin nhắn từ stream, sử dụng cơ chế blocking read để tiết kiệm tài nguyên

Quá trình xử lý bao gồm việc chuyển đổi dữ liệu từ định dạng stream sang đối tượng tin nhắn và lưu vào cơ sở dữ liệu thông qua 'MessageService'.

Stream consumer được khởi động riêng biệt từ file `consumer.ts`:

Quá trình khởi động consumer bao gồm các bước: 1. Nhập module 'MessageWorkerService' từ đường dẫn tương ứng 2. Tạo một hàm tự thực thi không đồng bộ (async IIFE) 3. Khởi tạo nhóm người tiêu dùng 4. Bắt đầu xử lý tin nhắn từ stream

Cách tiếp cận này đảm bảo consumer luôn sẵn sàng xử lý tin nhắn mới ngay khi khởi động.

## 3.5 HTTP Compression

### 3.5.1 Khái niệm và lợi ích

HTTP Compression là kỹ thuật nén dữ liệu trước khi gửi từ server đến client, giúp giảm kích thước dữ liệu truyền tải:

- **Giảm băng thông:** Tiết kiệm băng thông và chi phí mạng
- **Tăng tốc độ tải trang:** Người dùng nhận được phản hồi nhanh hơn
- **Cải thiện trải nghiệm:** Đặc biệt hiệu quả với người dùng có kết nối mạng chậm

### 3.5.2 Triển khai trong dự án

Việt Food sử dụng middleware `compression` của Express để nén dữ liệu:

Hệ thống sử dụng middleware `'compression'` của Express với cấu hình tối ưu: `ngưỡng nén`

Cấu hình này được thiết lập trong `server.ts`, áp dụng cho tất cả các request. Điểm đáng chú ý:

- **threshold 1024 bytes:** Chỉ nén các phản hồi có kích thước lớn hơn 1KB, tránh lãng phí tài nguyên CPU cho các phản hồi nhỏ
- **level 6:** Mức nén cân bằng giữa tốc độ và hiệu quả nén

## 3.6 Tối ưu hóa hình ảnh với Sharp

### 3.6.1 Khái niệm và lợi ích

Sharp là thư viện xử lý hình ảnh cho Node.js, được xây dựng trên libvips - thư viện xử lý hình ảnh hiệu suất cao:

- **Tốc độ xử lý cao:** Nhanh hơn nhiều so với các thư viện xử lý ảnh thông thường
- **Tiết kiệm bộ nhớ:** Sử dụng bộ nhớ hiệu quả khi xử lý ảnh lớn
- **Tối ưu hóa định dạng:** Chuyển đổi sang các định dạng hiệu quả như WebP
- **Thay đổi kích thước:** Giảm kích thước ảnh nhưng vẫn giữ chất lượng tốt

### 3.6.2 Triển khai trong dự án

Việt Food sử dụng Sharp thông qua middleware `image-processor.middleware.ts` để xử lý ảnh khi người dùng tải lên.

Hệ thống xử lý ảnh tải lên bằng thư viện Sharp, thực hiện các bước tối ưu: chuyển đổi sang định dạng WebP, giảm kích thước tối đa 1000px, đặt chất lượng 80

Middleware này thực hiện các tối ưu:

- **Resize:** Giảm kích thước ảnh xuống tối đa 1000px để tiết kiệm không gian lưu trữ
- **Chuyển đổi sang WebP:** Định dạng hiện đại có kích thước nhỏ hơn nhưng chất lượng tương đương hoặc tốt hơn JPEG/PNG
- **Điều chỉnh chất lượng:** Cân bằng giữa kích thước file và chất lượng hình ảnh (quality: 80)
- **Mức độ nén tối ưu:** Cấu hình `effort: 6` đảm bảo cân bằng giữa tốc độ xử lý và hiệu quả nén

Middleware này được áp dụng cho các route xử lý upload hình ảnh, đảm bảo tất cả hình ảnh đều được tối ưu trước khi lưu trữ.

## 3.7 Hệ thống Agent thông minh

### 3.7.1 Giới thiệu về Agent

Việt Food triển khai một hệ thống Agent thông minh hỗ trợ khách hàng trong quá trình đặt món và tương tác với ứng dụng:

- **Trợ lý ảo:** Agent hoạt động như một trợ lý ảo thông minh, tương tác với người dùng qua giao diện chat
- **Xử lý ngôn ngữ tự nhiên:** Sử dụng mô hình NLP để hiểu và phản hồi câu hỏi của người dùng
- **Tư vấn món ăn:** Gợi ý món ăn dựa trên sở thích và lịch sử đặt hàng của người dùng
- **Xử lý quy trình:** Hướng dẫn người dùng hoàn thành quy trình đặt hàng

### 3.7.2 Lưu trữ message và quản lý context

Một trong những thách thức lớn nhất của hệ thống Agent là việc lưu trữ tin nhắn và duy trì context hội thoại. Việt Food sử dụng MongoDB để lưu trữ tin nhắn và Redis để quản lý context session:

Mô hình dữ liệu tin nhắn được thiết kế với các trường chính sau:

- ‘userId’: Định danh duy nhất của người dùng, dùng để liên kết tin nhắn với tài khoản người dùng cụ thể
- ‘content’: Nội dung văn bản của tin nhắn, được mã hóa UTF-8
- ‘role’: Phân biệt giữa tin nhắn từ người dùng (‘user’) và hệ thống (‘system’ hoặc ‘assistant’)
- ‘timestamps’: Tự động thêm thời gian tạo (‘createdAt’) và cập nhật (‘updatedAt’)

Dịch vụ ‘MessageService’ cung cấp bộ phương thức toàn diện để quản lý vòng đời tin nhắn:

- Tạo và lưu trữ tin nhắn mới trong cơ sở dữ liệu MongoDB
- Truy vấn lịch sử tin nhắn theo người dùng, với phân trang và sắp xếp thời gian giảm dần
- Tích hợp Redis Stream cho xử lý bất đồng bộ, đảm bảo khả năng mở rộng
- Quản lý trạng thái tin nhắn thời gian thực thông qua WebSocket

### 3.7.3 Quản lý context hội thoại

Duy trì context hội thoại là yếu tố quan trọng để Agent có thể hiểu được tiến trình đối thoại đang diễn ra. Một phương pháp tối ưu để quản lý context hội thoại được triển khai bằng cách sử dụng Google GenAI kết hợp với cơ chế in-memory cache. Hệ thống quản lý các phiên hội thoại thông qua các thành phần chính sau:

- **AgentSessionEntry:** Lưu trữ thông tin phiên hội thoại, bao gồm đối tượng chat và thời gian truy cập cuối cùng
- **Session Management:** Tự động dọn dẹp các phiên không hoạt động sau 1 giờ
- **In-memory Storage:** Sử dụng Map để lưu trữ các phiên với thời gian thực
- **Automatic Cleanup:** Hệ thống tự động dọn dẹp các phiên không hoạt động mỗi 10 phút

Cách tiếp cận này đảm bảo hiệu suất cao trong khi vẫn duy trì được trạng thái hội thoại cần thiết cho trải nghiệm người dùng liền mạch. Các function tool quan trọng được triển khai cho hệ thống bao gồm:

- **searchDishes:** Tìm kiếm món ăn theo các tiêu chí
- **addToCart:** Thêm món ăn vào giỏ hàng
- **removeFromCart:** Xóa món ăn khỏi giỏ hàng
- **getDishesInUserCart:** Lấy danh sách món ăn trong giỏ hàng
- **getAllDishes:** Lấy toàn bộ danh sách món ăn

### 3.7.4 Vấn đề đang gặp phải

Hệ thống Agent đang gặp phải một số vấn đề như:

- **Quản lý context hội thoại:** Chưa áp dụng cơ chế lọc context để lấy thông tin cần thiết của người dùng trong cuộc hội thoại
- **Agent:** Chưa tối ưu khả năng làm công việc phức tạp, chỉ hoạt động tốt với các yêu cầu cơ bản
- **Function Calling:** Chưa triển khai đầy đủ các function tool để Agent có thể thực hiện các tác vụ phức tạp
- **Context hội thoại:** Nguy cơ mất dữ liệu message do redis stream không được lưu trữ lâu dài

# Chương 4

## Đánh giá hiệu quả và đề xuất cải tiến

### 4.1 Thủ nghiệm hiệu suất hệ thống

Chương này trình bày kết quả thử nghiệm hiệu suất của hệ thống Việt Food, tập trung vào đánh giá khả năng chịu tải và hiệu quả của các cơ chế tối ưu hóa. Các thử nghiệm được thực hiện trong môi trường phát triển với cấu hình phần cứng: Mac M4, 16GB RAM, 10-core CPU (chạy trên 1 core).

### 4.2 Phương pháp thử nghiệm

Để đánh giá hiệu suất của hệ thống, Em đã thực hiện các bài kiểm tra tải (load test) sử dụng công cụ Apache Bench (ab) với các tham số sau:

```
ab -n 10000 -c 200 http://127.0.0.1:3000/api/dishes?page=1&limit=9&searchTerm=com
```

Trong đó:

- **ab:** Công cụ Apache Bench
- **-n 10000:** Tổng số request gửi đi là 10,000
- **-c 200:** Số lượng request đồng thời (concurrent) là 200
- **API:** API lấy danh sách món ăn với phân trang và tìm kiếm 9 món ăn và từ khóa com

Em tiến hành thử nghiệm với tổng số lượng món ăn là 10000 trên hai phiên bản của hệ thống:

1. Phiên bản không nén dữ liệu (No compression)
2. Phiên bản có bật nén HTTP (With compression)

## 4.3 Kết quả thử nghiệm

### 4.3.1 Phiên bản không nén dữ liệu

```
Document Path:          /api/dishes?page=1&limit=9&searchTerm=com
Document Length:        3330 bytes

Concurrency Level:      200
Time taken for tests:  4.226 seconds
Complete requests:     10000
Failed requests:        0
Total transferred:     35950000 bytes
HTML transferred:       33300000 bytes
Requests per second:   2366.34 [#/sec] (mean)
Time per request:      84.519 [ms] (mean)
Time per request:      0.423 [ms] (mean, across all concurrent requests)
Transfer rate:          8307.59 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.4     0     4
Processing:    19   83  21.7    81    215
Waiting:       15   83  21.6    80    215
Total:         19   84  21.7    82    217
```

Hình 4.1: Kết quả kiểm thử tải - Không nén dữ liệu

Bảng 4.1: Chỉ số hiệu suất - Không nén dữ liệu

Chỉ số	Giá trị
Document Length	3330 [bytes]
Requests per second	2366,34 [req/s]
Time per request (mean)	84.519 [ms]
Time per request (across concurrent)	0.423 [ms]
Transfer rate	8307.59[KBytes/sec]

### 4.3.2 Phiên bản có nén dữ liệu

Document Path:	/api/dishes?page=1&limit=9&searchTerm=com
Document Length:	883 bytes
Concurrency Level:	200
Time taken for tests:	4.398 seconds
Complete requests:	10000
Failed requests:	0
Total transferred:	11500000 bytes
HTML transferred:	8830000 bytes
Requests per second:	2273.62 [#/sec] (mean)
Time per request:	87.965 [ms] (mean)
Time per request:	0.440 [ms] (mean, across all concurrent requests)
Transfer rate:	2553.38 [Kbytes/sec] received
Connection Times (ms)	
	min mean[+/-sd] median max
Connect:	0 0 2.1 0 75
Processing:	12 87 24.7 85 208
Waiting:	8 85 24.7 83 206
Total:	12 87 24.8 85 208

Hình 4.2: Kết quả kiểm thử tải - Có nén dữ liệu

Bảng 4.2: Chỉ số hiệu suất - Có nén dữ liệu

Chỉ số	Giá trị
Document Length	883 [bytes]
Requests per second	2273.62 [req/s]
Time per request (mean)	87.965 [ms]
Time per request (across concurrent)	0.44 [ms]
Transfer rate	2553.38 [KBytes/sec]

## 4.4 Phân tích kết quả

### 4.4.1 So sánh hiệu suất

Bảng 4.3: So sánh hiệu suất giữa hai phiên bản

Chỉ số	Không nén	Có nén	Thay đổi
Requests/second	2366.34	2273.62	-3.9%
Time/req (mean)	84.519 ms	87.965 ms	-3.9%
Document Length	3330 [bytes]	883 [bytes]	-73.5%

### 4.4.2 Nhận xét

- Hiệu suất xử lý:** Phiên bản có bật nén cho thấy hiệu suất tốt hơn nhẹ với số lượng request xử lý mỗi giây giảm 3.2% (từ 2,366.34 xuống còn 2,273.62 requests/giây).

2. **Thời gian phản hồi:** Thời gian xử lý trung bình tăng 3.1%, từ 84.519ms lên còn 87.965ms.

3. **Độ ổn định:** Cả hai phiên bản đều xử lý thành công 100% các request, cho thấy độ ổn định cao của hệ thống ngay cả dưới tải lớn.

## 4.5 Kết luận

Các kết quả thử nghiệm cho thấy:

1. Việc áp dụng HTTP compression mang lại hiệu quả đáng kể trong việc tiết kiệm băng thông mạng (giảm hơn 71.2%) trong khi chỉ ảnh hưởng không đáng kể đến hiệu suất xử lý của server.

2. Hệ thống có khả năng mở rộng tốt, có thể xử lý hơn 2000 requests mỗi giây chỉ sử dụng 1 core CPU

3. Thời gian phản hồi ổn định và nằm trong ngưỡng chấp nhận được cho ứng dụng web, với 90% request hoàn thành dưới 200ms.

4. Để tối ưu hơn nữa, có thể áp dụng: - Tối ưu hóa các truy vấn cơ sở dữ liệu - Áp dụng thêm caching ở nhiều tầng - Triển khai cân bằng tải khi mở rộng hệ thống - Chạy nhiều instance của ứng dụng để xử lý song song

# Chương 5

## Kết luận và hướng phát triển

### 5.1 Tóm tắt kết quả phân tích

Qua quá trình phân tích dự án Việt Food, chúng tôi đã khảo sát chi tiết các kỹ thuật tối ưu backend được áp dụng trong dự án. Những kỹ thuật này đóng vai trò quan trọng trong việc nâng cao hiệu suất, đảm bảo khả năng phục vụ và tạo trải nghiệm người dùng tốt hơn. Tóm tắt các phát hiện chính:

- **Redis Caching:** Dự án đã triển khai một hệ thống cache toàn diện cho dữ liệu món ăn và danh mục, với chiến lược priming thông minh khi khởi động server. Cách tiếp cận này giúp giảm đáng kể thời gian phản hồi và tải trên database, đặc biệt hữu ích với dữ liệu món ăn - loại dữ liệu được truy cập thường xuyên nhưng ít thay đổi.
- **Redis Stream:** Việc sử dụng Redis Stream để xử lý tin nhắn theo mô hình producer-consumer đã tách biệt được các tác vụ xử lý nặng khỏi luồng chính của ứng dụng. Cách thiết kế này giúp hệ thống có khả năng mở rộng và phục hồi tốt hơn.
- **HTTP Compression:** Compression middleware được áp dụng toàn cục với cấu hình cân bằng giữa hiệu suất nén và tài nguyên CPU. Kỹ thuật này giúp giảm đáng kể kích thước dữ liệu truyền tải, cải thiện thời gian tải trang.
- **Tối ưu hình ảnh với Sharp:** Dự án sử dụng Sharp để xử lý và tối ưu hóa hình ảnh tải lên, chuyển đổi sang định dạng WebP hiệu quả và resize phù hợp. Cách tiếp cận này giúp tiết kiệm không gian lưu trữ và băng thông.

Nhìn chung, Việt Food đã áp dụng các kỹ thuật tối ưu backend một cách hợp lý và hiệu quả. Các kỹ thuật này không chỉ hoạt động độc lập mà còn hỗ trợ cho nhau, tạo nên một hệ thống có hiệu suất cao và khả năng phục vụ tốt.

### 5.2 Bài học kinh nghiệm

Quá trình phân tích dự án Việt Food đã mang lại nhiều bài học quý báu về tối ưu hóa backend trong phát triển ứng dụng web:

### **5.2.1 Thiết kế hiệu suất ngay từ đầu**

Việt Food đã áp dụng các kỹ thuật tối ưu ngay từ giai đoạn thiết kế ban đầu, chứ không phải như một giải pháp bổ sung sau này. Điều này cho thấy tầm quan trọng của việc xem xét các vấn đề hiệu suất ngay từ khi bắt đầu dự án.

### **5.2.2 Caching là chìa khóa**

Redis caching được triển khai một cách toàn diện trong dự án, minh họa rằng chiến lược caching tốt có thể mang lại lợi ích hiệu suất đáng kể. Đặc biệt, việc xác định đúng những dữ liệu nào nên được cache (như dữ liệu món ăn) và cách quản lý vòng đời cache là rất quan trọng.

### **5.2.3 Phân tách xử lý không đồng bộ**

Sử dụng Redis Stream để tách biệt xử lý tin nhắn là một ví dụ tốt về cách phân tách các tác vụ nặng và không đồng bộ khỏi luồng chính của ứng dụng. Cách tiếp cận này giúp ứng dụng vẫn có thể phản hồi nhanh chóng ngay cả khi xử lý các tác vụ phức tạp.

### **5.2.4 Tối ưu tài nguyên tĩnh**

Việc sử dụng Sharp để tối ưu hình ảnh và compression để giảm kích thước dữ liệu truyền tải cho thấy tầm quan trọng của việc tối ưu tài nguyên tĩnh. Những kỹ thuật này có thể mang lại cải thiện hiệu suất đáng kể với chi phí triển khai tương đối thấp.

### **5.2.5 Cân bằng giữa hiệu suất và phức tạp**

Dự án Việt Food đã chọn cách triển khai hợp lý, cân bằng giữa hiệu suất và độ phức tạp của mã nguồn. Một số kỹ thuật tối ưu có thể được cải tiến thêm, nhưng cách tiếp cận hiện tại đã mang lại lợi ích đáng kể mà không làm tăng quá mức độ phức tạp của hệ thống.

## **5.3 Hướng phát triển tiềm năng**

Dựa trên phân tích của chúng tôi, có một số hướng phát triển tiềm năng để nâng cao hơn nữa hiệu suất và khả năng mở rộng của Việt Food:

### **5.3.1 Kiến trúc Microservices**

Với quy mô ngày càng tăng, Việt Food có thể xem xét chuyển từ kiến trúc monolithic hiện tại sang kiến trúc microservices. Điều này sẽ cho phép:

- Phát triển và triển khai độc lập các thành phần
- Mở rộng có chọn lọc các dịch vụ cần thiết
- Cố lập lỗi tốt hơn

Ví dụ, có thể tách thành các service riêng biệt cho quản lý món ăn, xử lý đơn hàng, hệ thống tin nhắn, v.v.

### 5.3.2 Caching đa tầng

Phát triển hệ thống cache thành nhiều tầng:

- **Client-side caching:** Áp dụng HTTP caching với ETag và Cache-Control headers
- **CDN caching:** Triển khai CDN cho tài nguyên tĩnh
- **API Gateway caching:** Cache ở tầng API Gateway
- **Application caching:** Tiếp tục sử dụng Redis, nhưng với chiến lược cache invalidation tinh tế hơn
- **Database caching:** Query caching ở tầng database

### 5.3.3 Xử lý ảnh nâng cao

Cải tiến hệ thống xử lý ảnh:

- **Responsive images:** Tạo nhiều phiên bản của mỗi hình ảnh cho các thiết bị khác nhau
- **Tích hợp CDN chuyên biệt:** Sử dụng CDN có khả năng xử lý hình ảnh theo yêu cầu
- **AVIF format:** Áp dụng định dạng AVIF mới, hiệu quả hơn cả WebP
- **Lazy loading và progressive loading:** Tối ưu tải hình ảnh

### 5.3.4 Tích hợp GraphQL

GraphQL có thể là một bổ sung hữu ích cho API RESTful hiện tại:

- Cho phép client chỉ định chính xác dữ liệu cần thiết
- Giảm over-fetching và under-fetching
- Hỗ trợ tốt cho các ứng dụng mobile với kết nối mạng không ổn định

### 5.3.5 Giám sát và phân tích hiệu suất

Cải thiện hệ thống giám sát:

- **APM (Application Performance Monitoring):** Triển khai công cụ giám sát hiệu suất ứng dụng
- **Metrics collection:** Thu thập metrics về cache hit/miss ratio, thời gian phản hồi, v.v.
- **Distributed tracing:** Theo dõi các request qua nhiều service
- **Tự động điều chỉnh:** Điều chỉnh cấu hình cache, số lượng worker dựa trên metrics

### 5.3.6 Serverless Computing

Xem xét chuyển một số thành phần sang mô hình serverless:

- **Image processing:** Chuyển xử lý hình ảnh sang serverless functions
- **Periodic tasks:** Các tác vụ định kỳ như cập nhật cache
- **Event-driven processing:** Xử lý sự kiện như đơn hàng mới, đánh giá mới

Việt Food đã có nền tảng tốt với các kỹ thuật tối ưu hiện tại. Việc áp dụng các hướng phát triển này sẽ giúp dự án tiếp tục mở rộng quy mô và duy trì hiệu suất cao trong tương lai.