Restaurant Management System

Microservices Architecture with Docker Compose

Assignment - 1

Student Name: Syed Huzaifa Kamal

Roll Number: 21P-0500

Course: DevOps

Teacher: Usama Musharaf

Date: October 2, 2025

Table of Contents

1. Introduction

In this project, I have developed a complete Restaurant Management System using microservices architecture. The main objective was to create a scalable and containerized application that demonstrates modern cloud-native development practices.

The system allows customers to browse the menu, place orders, and reserve tables online. Additionally, it provides administrative features for managing the restaurant operations. The entire application is containerized using Docker and orchestrated with Docker Compose, making it easy to deploy and scale.

Key Features:

- User authentication and authorization
- Online menu browsing with Pakistani Rupee pricing
- Shopping cart and order placement
- Table reservation system
- Order history tracking
- Admin panel for managing operations

This project demonstrates the practical implementation of containerization, microservices communication, database management, and modern web development practices.

2. System Architecture

The Restaurant Management System follows a microservices architecture pattern where each service is independently deployable and scalable. The system consists of 10 containerized services that communicate with each other through a common Docker network.

2.1 Architecture Diagram

The following diagram illustrates the complete system architecture:

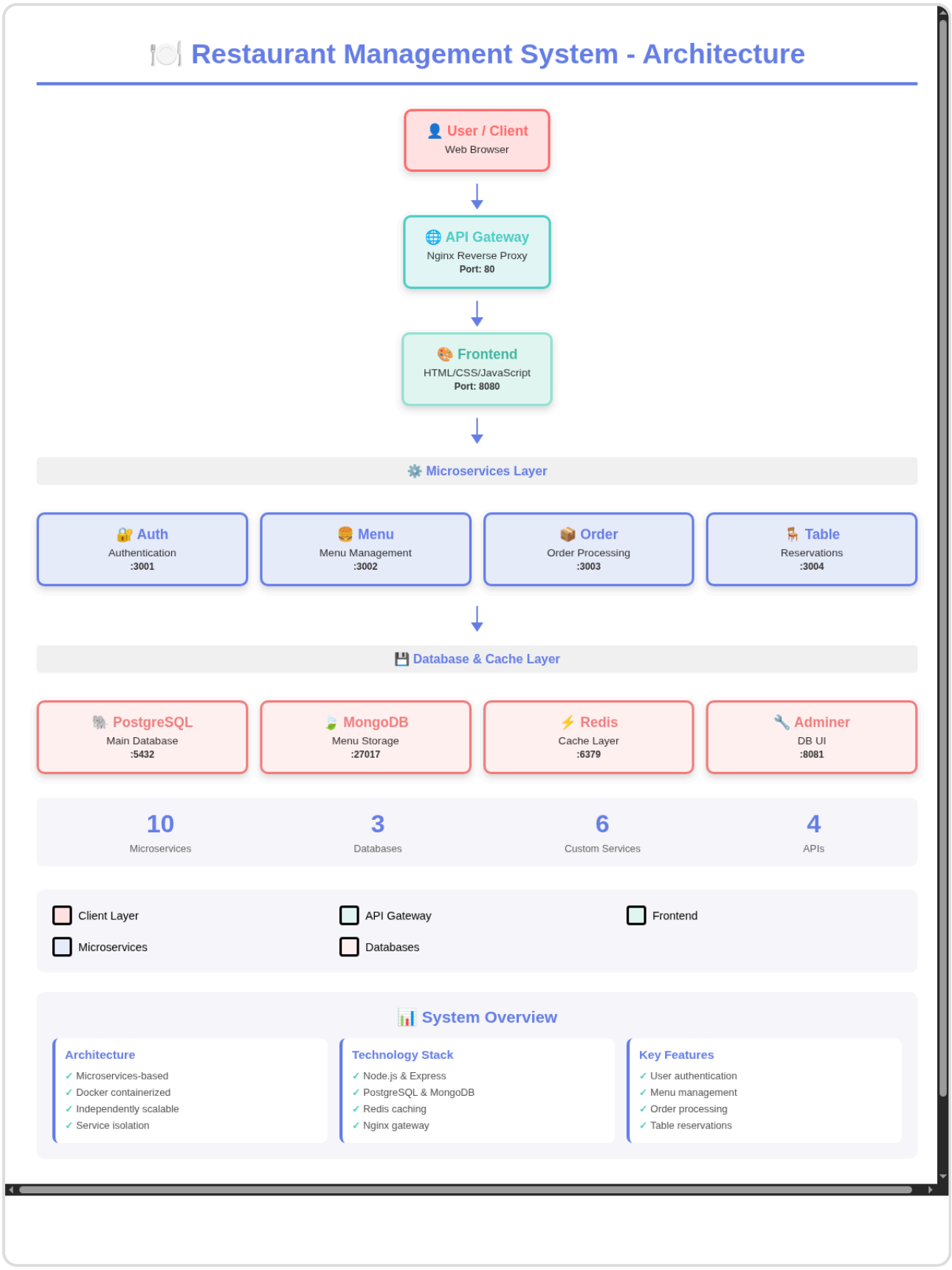

Figure 1: Restaurant Management System Architecture

2.2 Architecture Components

The architecture is divided into several layers, each serving a specific purpose. At the top, we have the client layer where users interact with the system through a web browser. Below that, the API Gateway acts as a single entry point for all requests, routing them to appropriate microservices.

The microservices layer contains four business logic services: Auth Service handles user authentication, Menu Service manages restaurant menu items, Order Service processes customer orders, and Table Service manages table reservations. Each service is designed to be independent and focused on a single responsibility.

At the bottom, we have the data layer consisting of three different database systems. PostgreSQL stores relational data like users and orders, MongoDB stores menu items with flexible schema, and Redis provides caching for improved performance.

3. Technology Stack

I have used modern and industry-standard technologies to build this system. The choice of technologies was based on scalability, performance, and ease of development.

Backend Technologies:

- **Node.js:** JavaScript runtime for building server-side applications
- **Express.js:** Web framework for creating RESTful APIs
- **PostgreSQL:** Relational database for structured data
- **MongoDB:** NoSQL database for flexible document storage
- **Redis:** In-memory cache for performance optimization

Frontend Technologies:

- **HTML5/CSS3:** Structure and styling of web pages
- **JavaScript:** Client-side interactivity
- **Nginx:** Web server for serving static files

DevOps & Infrastructure:

- **Docker:** Containerization platform
- **Docker Compose:** Multi-container orchestration
- **Nginx (Gateway):** Reverse proxy and load balancer

3.1 Technology Justification

Technology	Reason for Selection
Node.js	Fast, lightweight, and perfect for microservices
PostgreSQL	Reliable for transactional data like orders and users
MongoDB	Flexible schema ideal for menu items with varying attributes
Redis	Extremely fast caching reduces database load
Docker	Ensures consistency across development and production

4. Microservices Overview

The application is built with 10 microservices, each running in its own Docker container. This section provides detailed information about each service.

4.1 API Gateway (Port 80)

The API Gateway serves as the single entry point for all client requests. It uses Nginx to route traffic to the appropriate backend services. This pattern provides several benefits including load balancing, SSL termination, and simplified client-side code.

4.2 Frontend Service (Port 8080)

The frontend is a single-page application built with HTML, CSS, and JavaScript. It provides an intuitive user interface for browsing the menu, placing orders, and managing reservations. Nginx serves the static files efficiently.

4.3 Auth Service (Port 3001)

This service handles all authentication and authorization logic. It provides user registration, login functionality, and JWT token generation. Passwords are securely hashed using bcrypt before storing in the database.

4.4 Menu Service (Port 3002)

The Menu Service manages all menu-related operations. It uses MongoDB to store menu items with their prices in Pakistani Rupees. The service provides APIs for viewing, adding, updating, and deleting menu items.

4.5 Order Service (Port 3003)

This service processes customer orders and maintains order history. It integrates with Redis for caching frequently accessed order data, which significantly improves response times. All orders are stored in PostgreSQL for reliability.

4.6 Table Service (Port 3004)

The Table Service manages restaurant table reservations. Customers can view available tables and make reservations for specific dates and times. The service tracks table capacity and availability status.

4.7 Database Services

Three database services support the application: PostgreSQL for relational data, MongoDB for flexible document storage, and Redis for caching. Each database is configured with persistent volumes to ensure data is not lost when containers restart.

4.8 Adminer (Port 8081)

Adminer provides a web-based database management interface. It allows administrators to view and manage data in both PostgreSQL and MongoDB databases through a simple browser interface.

5. Implementation Details

5.1 Docker Configuration

Each microservice has its own Dockerfile that defines how the container should be built. I used Node.js Alpine images to keep the container sizes small. The docker-compose.yml file orchestrates all services, defining their dependencies, networks, and volume mounts.

5.2 Database Schema

The PostgreSQL database contains four main tables: users for authentication, orders for customer orders, tables for restaurant table information, and reservations for booking records. MongoDB stores menu items as flexible documents, allowing for easy addition of new attributes.

5.3 Security Implementation

Security was a priority in this project. User passwords are hashed using bcrypt with a salt factor of 10. JWT tokens are used for session management, ensuring secure communication between the client and server. All services communicate within an isolated Docker network.

5.4 Performance Optimization

To improve performance, I implemented Redis caching for order data. When users view their orders, the system first checks Redis cache. If data is not found, it queries PostgreSQL and then caches the result for 30 seconds. This reduces database load significantly.

5.5 Deployment Process

The entire application can be deployed with a single command: `docker-compose up -d`. This builds all containers, creates networks, initializes databases, and starts all services. The modular architecture makes it easy to scale individual services based on demand.

6. Screenshots and Demonstration

This section presents screenshots of the running application, demonstrating all key features and functionalities.

6.1 Container Status

The first step was to verify that all Docker containers are running correctly. Using the docker-compose ps command, I confirmed that all 10 services were up and healthy.

```
└─$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED             STATUS          PORTS                                                 NAMES
884d5184f15b   restaurant-app-frontend        "/docker-entrypoint.…"   3 minutes ago       Up 3 minutes    0.0.0.0:8080->80/tcp, :::8080->80/tcp                 restaurant-fr
ontend
7876c3c09c7b   restaurant-app-menu-service    "docker-entrypoint.s…"   53 minutes ago      Up 48 minutes   0.0.0.0:3002->3002/tcp, :::3002->3002/tcp             restaurant-me
nu-service
5abe22ffaf08   restaurant-app-api-gateway     "/docker-entrypoint.…"   About an hour ago   Up 8 minutes    0.0.0.0:80->80/tcp, :::80->80/tcp                     restaurant-ap
i-gateway
714fbc69e00f   adminer:latest                 "entrypoint.sh docke…"   About an hour ago   Up 48 minutes   0.0.0.0:8081->8080/tcp, :::8081->8080/tcp             restaurant-ad
miner
eff057804c71   restaurant-app-auth-service    "docker-entrypoint.s…"   About an hour ago   Up 48 minutes   0.0.0.0:3001->3001/tcp, :::3001->3001/tcp             restaurant-au
th-service
158a9d5812e4   restaurant-app-table-service   "docker-entrypoint.s…"   About an hour ago   Up 48 minutes   0.0.0.0:3004->3004/tcp, :::3004->3004/tcp             restaurant-ta
ble-service
d6eb74b260a9   restaurant-app-order-service   "docker-entrypoint.s…"   About an hour ago   Up 48 minutes   0.0.0.0:3003->3003/tcp, :::3003->3003/tcp             restaurant-or
der-service
fcce63661d40   mongo:7                        "docker-entrypoint.s…"   About an hour ago   Up 48 minutes   0.0.0.0:27017->27017/tcp, :::27017->27017/tcp         restaurant-mo
ngo
2d567ba2cc94   postgres:15-alpine             "docker-entrypoint.s…"   About an hour ago   Up 48 minutes   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp             restaurant-po
stgres
9c4c384101df   redis:7-alpine                 "docker-entrypoint.s…"   About an hour ago   Up 48 minutes   0.0.0.0:6379->6379/tcp, :::6379->6379/tcp             restaurant-re
dis
```

Figure 2: All 10 Docker containers running successfully

6.2 Application Homepage

When users first visit the application, they see the homepage displaying the restaurant menu. The interface shows all available dishes with prices in Pakistani Rupees. Users can browse through different categories including Main Course, Starters, Desserts, and Beverages.

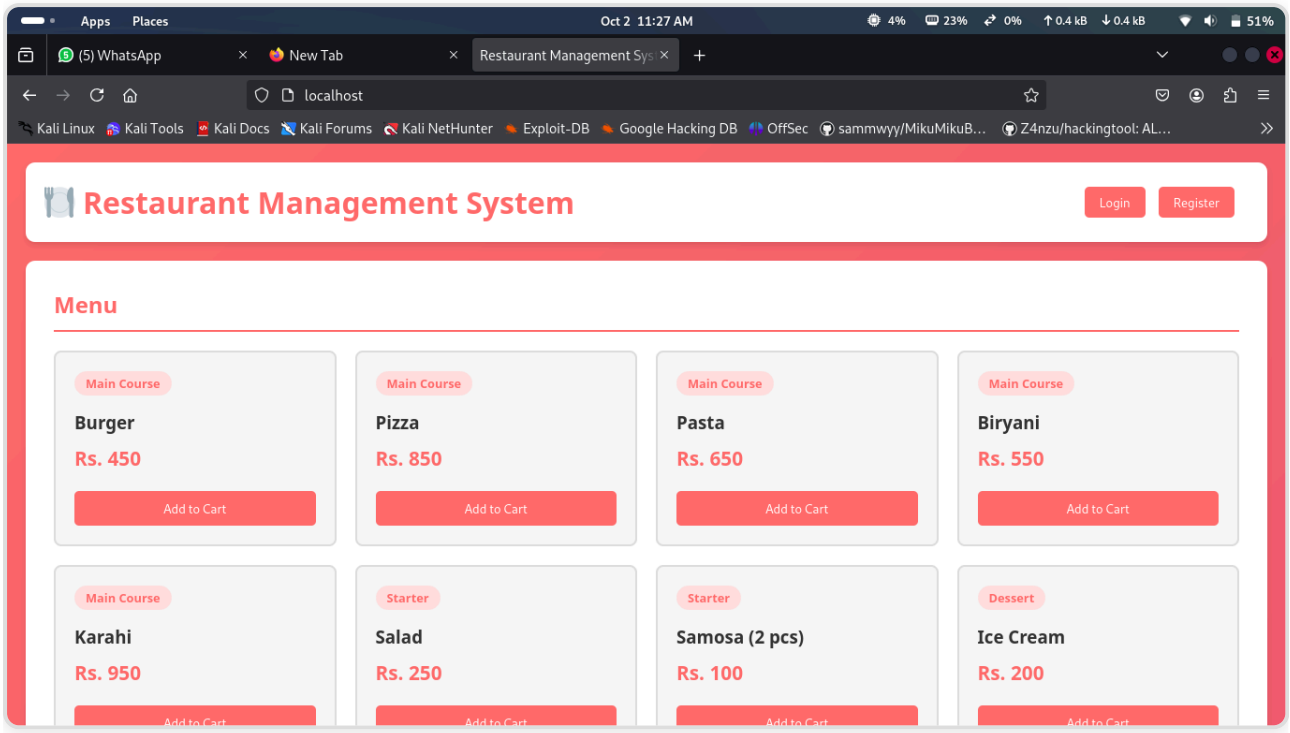

Figure 3: Restaurant Management System Homepage

6.3 User Registration

New users can register by clicking the Register button. The registration form asks for a username, password, and role (either Customer or Admin). The Auth Service validates the input and creates a new user account with a securely hashed password.

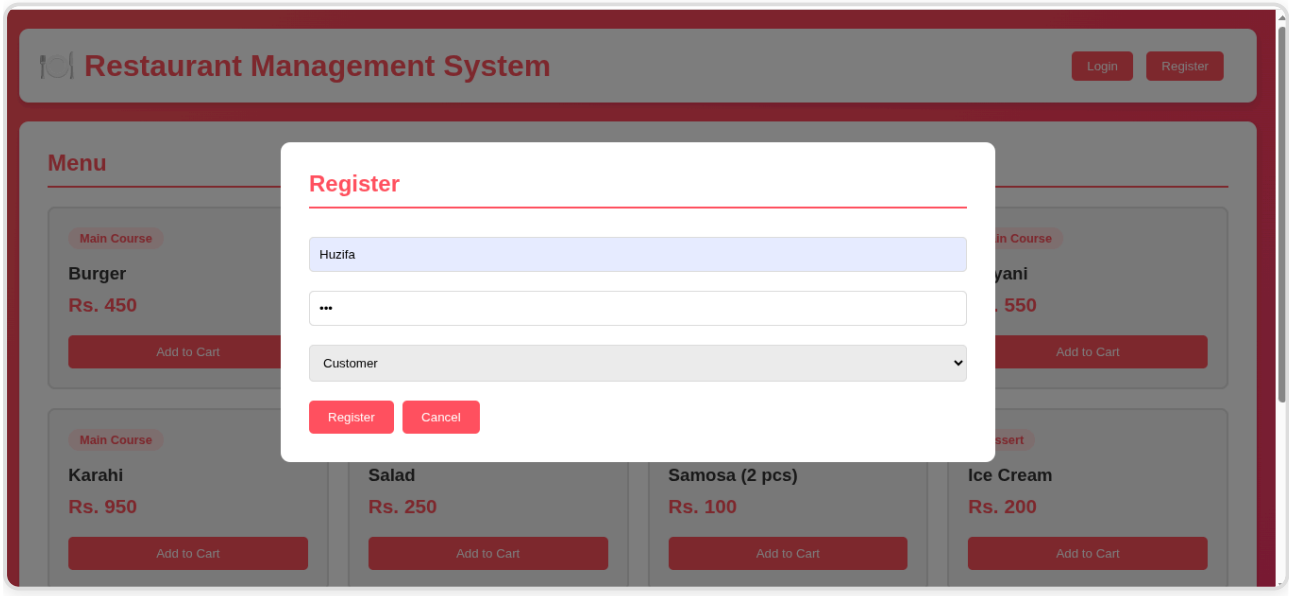

Figure 4: User registration interface

6.4 User Login

After registration, users can log in using their credentials. The system validates the username and password, then generates a JWT token for the session. Once logged in, users see personalized features including their order history and reservation options.

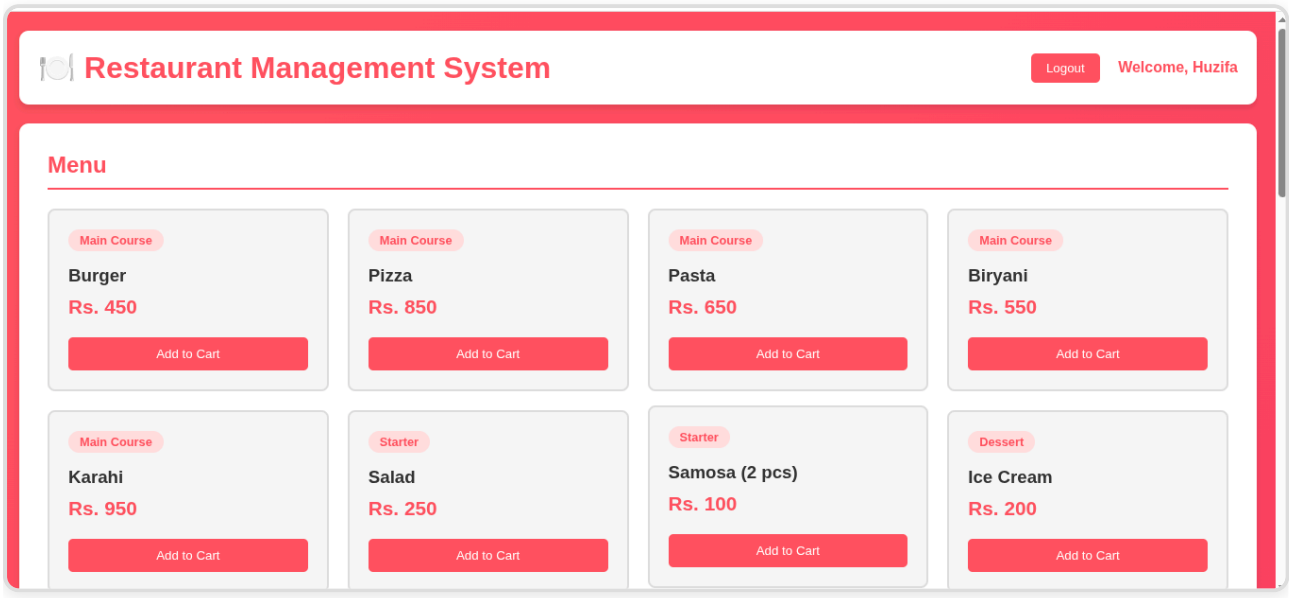

Figure 5: Successful login with menu displayed

6.5 Application Features

The application provides several key features including shopping cart functionality, order placement, and table reservations. Users can add multiple items to their cart, view the total price in Pakistani Rupees, and place orders with a single click. The system stores all order information in the database for future reference.

The table reservation system allows customers to book tables in advance. They can select from available tables, choose their preferred date and time, and specify the number of guests. The system automatically tracks table availability and prevents double bookings.

6.6 Database Management

Adminer provides a web-based interface for database administration. Through this tool, I can view all database tables, inspect stored data, run SQL queries, and perform maintenance tasks. The interface supports both PostgreSQL and MongoDB databases used in the application.

6.7 System Monitoring

Each microservice in the system exposes health check endpoints that return JSON responses indicating the service status. These endpoints are essential for monitoring system health and troubleshooting issues. Additionally, Docker provides comprehensive logging capabilities that help track service initialization, database connections, and API requests.

7. Conclusion

7.1 Project Summary

In this project, I successfully developed and deployed a complete Restaurant Management System using microservices architecture. The application demonstrates modern cloud-native development practices and showcases the benefits of containerization.

The system consists of 10 independently deployable services, each with a specific responsibility. By using Docker and Docker Compose, I achieved consistency across development and production environments. The application can be easily scaled by running multiple instances of individual services.

7.2 Key Achievements

- Successfully implemented 10 microservices with proper separation of concerns
- Integrated three different database technologies for optimal data storage
- Implemented secure authentication using JWT and bcrypt
- Created a responsive and user-friendly web interface
- Achieved performance optimization through Redis caching
- Containerized the entire application for easy deployment

7.3 Challenges Faced

During development, I encountered several challenges. Database initialization timing was an issue where services tried to connect before databases were ready. I solved this by implementing retry logic and proper dependency management in Docker Compose.

Another challenge was ensuring proper communication between services within the Docker network. By using Docker's built-in DNS resolution and service names, I established reliable inter-service communication.

7.4 Lessons Learned

This project taught me valuable lessons about microservices architecture and containerization. I learned the importance of proper service boundaries, the benefits of using different databases for different use cases, and how Docker simplifies deployment and scaling.

I also gained practical experience with API design, database schema design, and security best practices. The project reinforced the importance of documentation and clear architecture diagrams.

7.5 Future Enhancements

While the current implementation is fully functional, there are several areas for future improvement. These include adding payment gateway integration, implementing real-time notifications using WebSockets, creating a dedicated admin dashboard, and adding comprehensive unit and integration tests.

Additionally, the system could benefit from implementing API rate limiting, adding monitoring and logging solutions like Prometheus and Grafana, and deploying to a cloud platform like AWS or Google Cloud.

7.6 Final Thoughts

This Restaurant Management System demonstrates the power and flexibility of microservices architecture. Through this project, I have gained hands-on experience with modern development practices and tools that are widely used in the industry.

The containerized approach makes the application portable, scalable, and easy to maintain. I am confident that the skills and knowledge gained from this project will be valuable in my future career as a software developer.

8. References

1. Docker Documentation - https://docs.docker.com/
2. Node.js Documentation - https://nodejs.org/docs/
3. Express.js Guide - https://expressjs.com/
4. PostgreSQL Documentation - https://www.postgresql.org/docs/
5. MongoDB Manual - https://docs.mongodb.com/
6. Redis Documentation - https://redis.io/documentation
7. Nginx Documentation - https://nginx.org/en/docs/

End of Report

Restaurant Management System - Microservices Architecture

October 2025