APRIL 29, 2025

CLOUD-NATIVE FOOD ORDERING & DELIVERY SYSTEM

ASSIGNMENT – DISTRIBUTED SYSTEMS

IT22630834 | IT22552860

Y3S1_WD_09

Contents

1. Introduction

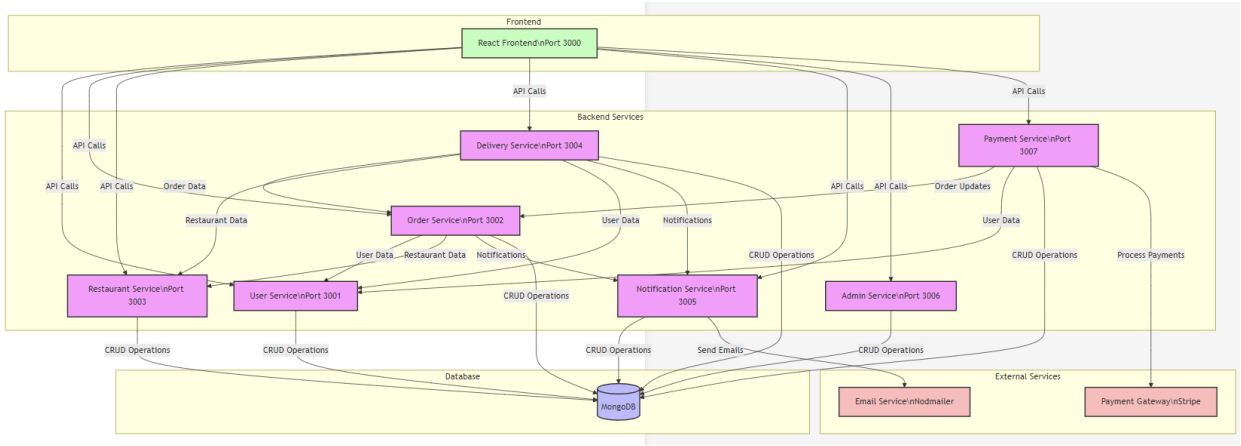
In the age of digital transformation, the demand for fast, efficient, and reliable food delivery services has surged dramatically. Platforms like UberEats, PickMe Food, and DoorDash have revolutionized the food industry by offering customers the convenience of ordering meals from a wide variety of restaurants with just a few taps on their smartphones. The Foodey platform aims to deliver a similar experience tailored to the Sri Lankan market while leveraging modern cloud-native technologies to ensure performance, scalability, and maintainability.

Foodey is a cloud-native food ordering and delivery system designed using microservices architecture. The application facilitates seamless interaction between three key user roles: customers, restaurant administrators, and delivery personnel. Customers can explore menus from multiple restaurants, add items to their cart, make payments, and track orders in real-time. Restaurant administrators are empowered with tools to manage menus, receive and update orders, and control restaurant availability. Delivery personnel, in turn, receive order assignments automatically and can update delivery statuses as they fulfill them. This multi-role ecosystem ensures smooth coordination and minimizes manual intervention.

The system is built upon a RESTful API structure using microservices, ensuring that each service (e.g., order management, restaurant management, delivery, payment integration) operates independently and is easily scalable. By containerizing services using Docker and orchestrating them with Kubernetes, Foodey achieves high availability and ease of deployment. This modular approach not only supports future enhancements but also reduces interdependency, which is critical in high-load production environments.

Foodey also integrates modern third-party services such as PayHere and Dialog Genie for payments, and Twilio for SMS/email notifications, ensuring secure and timely communication with users. By adopting robust authentication mechanisms, each user is uniquely identified and provided with role-specific access to system features. Ultimately, Foodey is more than just a food ordering application—it is a demonstration of how microservices, containerization, and cloud-native development principles can be brought together to create a practical, reliable, and user-friendly solution in the real-world domain of food technology.

2. High -Level Architecture Diagram



3. Service Interfaces

The Foodey platform is developed using a microservices architecture, where each service is independently developed, deployed, and maintained. The key services expose their functionality through well-defined RESTful APIs, ensuring modularity and ease of integration. Below is a summary of the main service interfaces and their respective endpoints

a. Authentication Service

Manages user accounts and role-based authentication

Base URL : /api/users

HTTP Method	Endpoint	Description
POST	/register	Register a new user (Customer/Delivery/Admin)
POST	/login	Authenticate and issue JWT token
GET	/me	Get details of currently logged-in user

b. Restaurant Management Service

This service enables restaurant owners to managers their profiles, menu items, and restaurant availability.

Base URL : /api/restaurants

HTTP Method	Endpoint	Description
POST	/register	Register a new restaurant
GET	/	Get list of all registered restaurants
GET	/{id}	Get details of a specific retaurant
PUT	/{id}	Update restaurant details
DELETE	/{id}	Delete a restaurant
POST	/{id}/menu	Add a menu item to a restaurant
PUT	/menu/{itemId}	Update a specific menu item
DELETE	/menu/{itemId}	Delete a menu item
PUT	/{id}/availability	Set restaurant availability (open/closed)
GET	/{id}/orders	View orders received by the restaurant

c. Order Management Service

This service allows customers to place and manage their food orders

Base URL : /api/orders

HTTP Method	Endpoint	Description
POST	/	Place a new order
GET	/{orderId}	Get details of a specific order
PUT	/{orderId}	Modify an order before confirmation
DELETE	/{orderId}	Cancel an unconfirmed order
GET	/customer/{userId}	Get all orders made by a specific customer
PUT	/{orderId}/status	Update order status (confirmed, preparing, etc.)

d. Delivery Management Service

Handles the assignment of drivers and tracking of deliveries

Base URL : /api/deliveries

HTTP Method	Endpoint	Description
GET	/available-drivers	List available drivers for assignment
POST	/assign/{orderId}	Assign a delivery driver based on location & status
PUT	/{deliveryId}/status	Update delivery status (picked, out-for-delivery)
GET	/track/{orderId}	Track delivery status in real-time

e. Notification Service

Responsible for sending SMS and email notifications upon order placement and status updates

Base URL : /api/notifications

HTTP Method	Endpoint	Description
POST	/send-email	Send an email notification
POST	/send-sms	Send an SMS notification

f. Payment Service

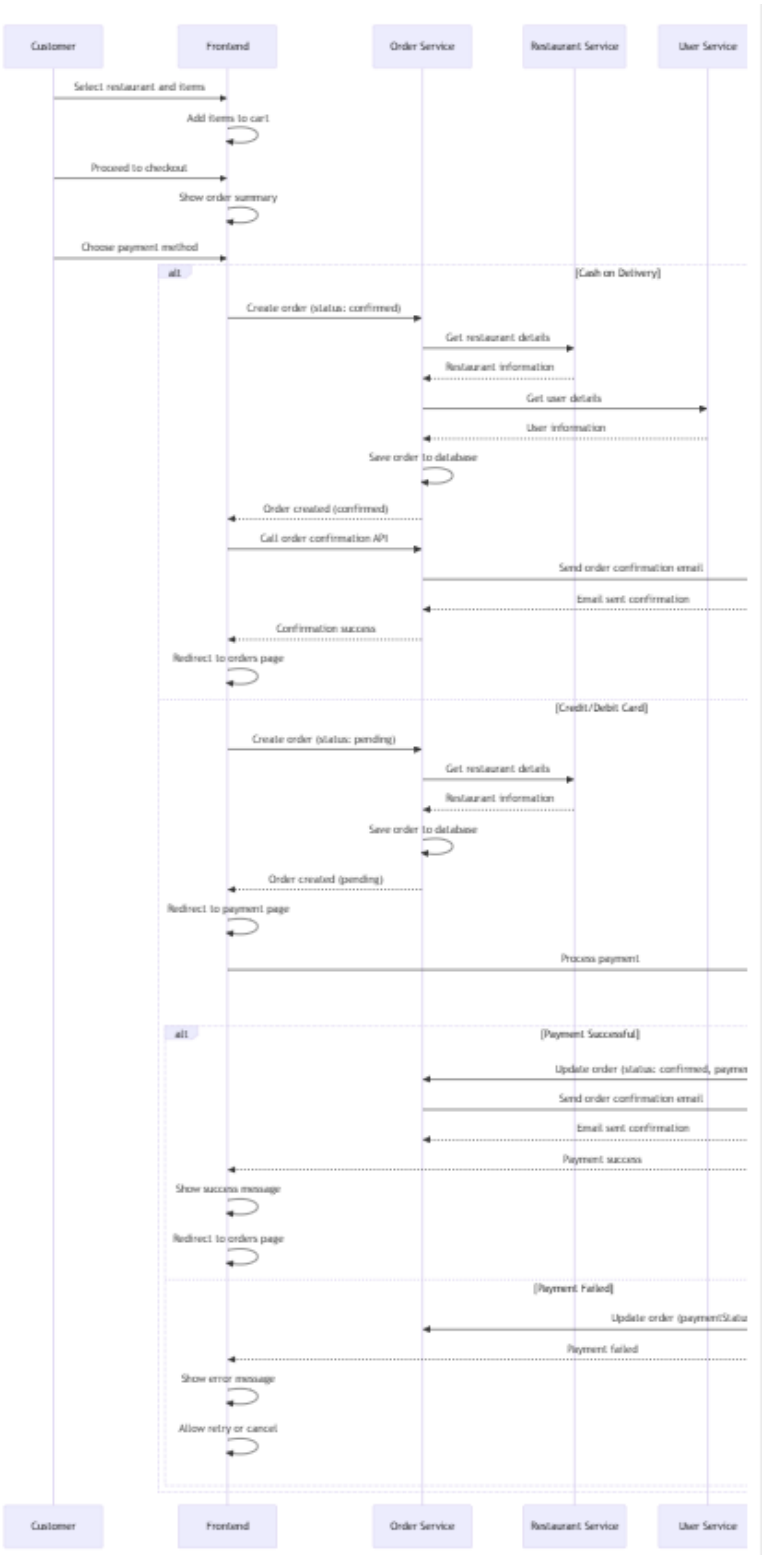
Handles online payment transactions using third-party integratinos like Stripe

Base URL : /api/payments

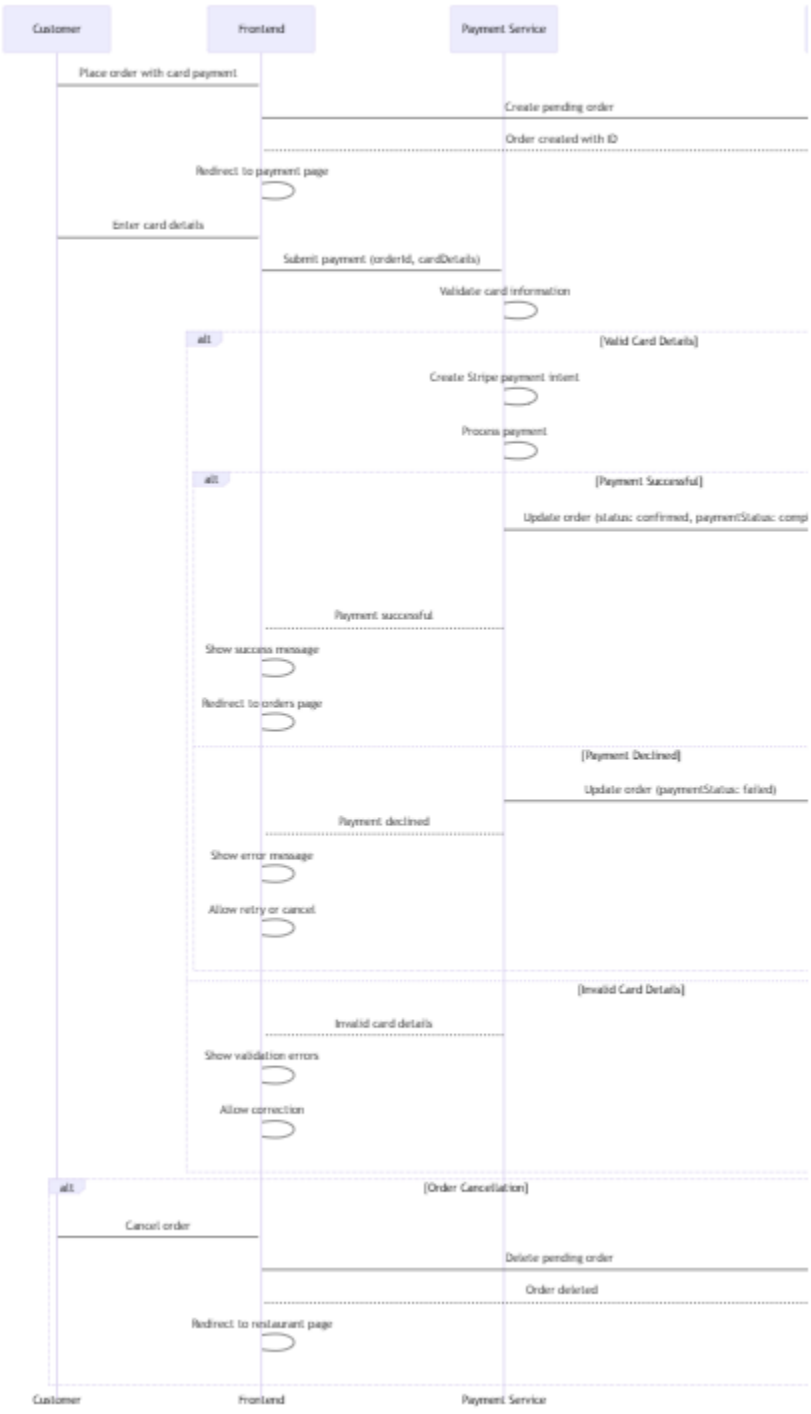
HTTP Method	Endpoint	Description
POST	/initiate	Initiate a payment for an order
POST	/callback	Handle payment confirmation callback
GET	/status/{oderId}	Get payment status of a specific order

4. Workflows

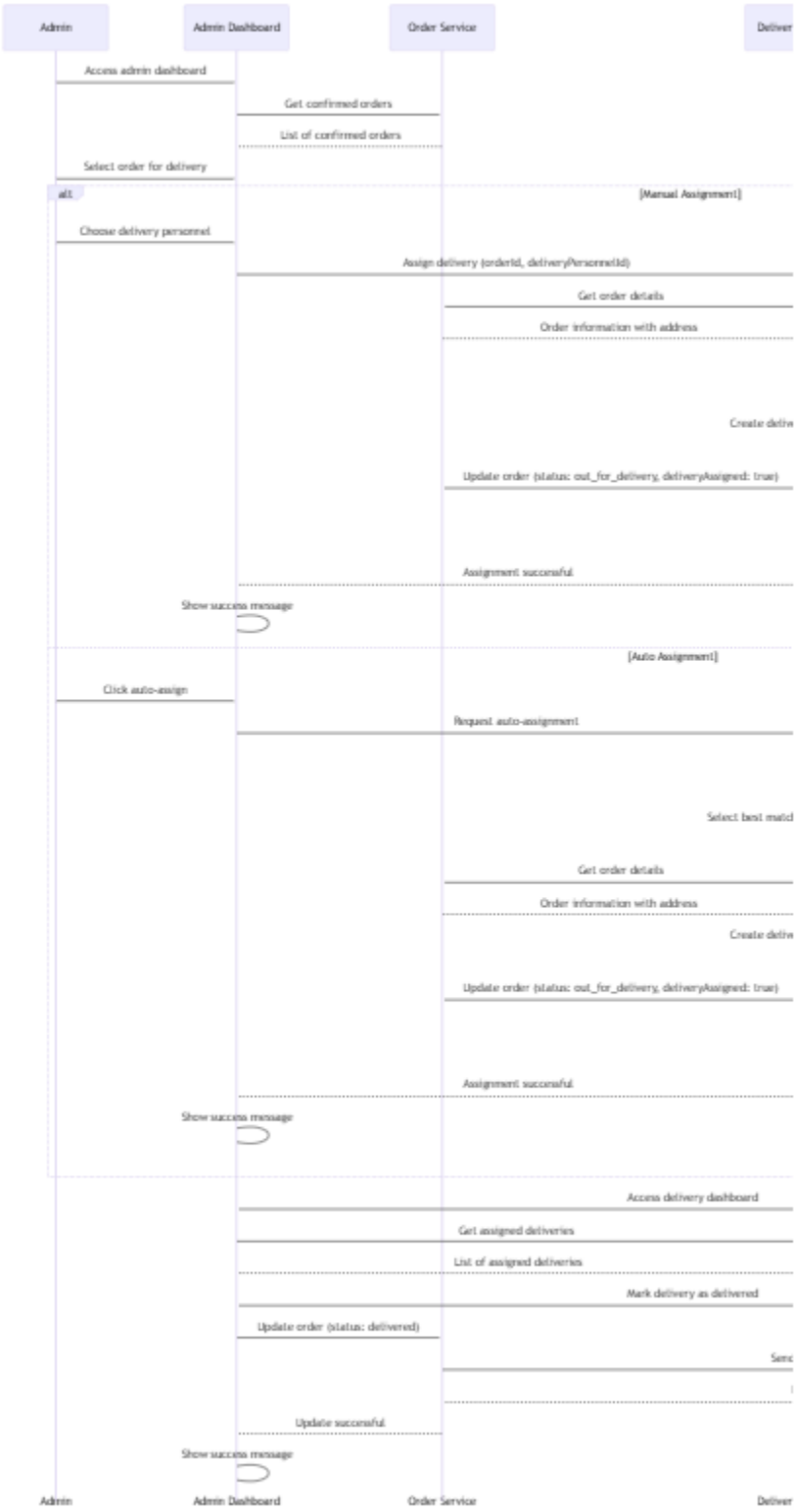
a. Order Placement Workflow



b. Restaurant Management Workflow



c. Delivery Assignment Workflow



5. Authentication & Security Mechanisms

4.1. Authentication Approach

- The system implements JWT (JSON Web Token)-based authentication for secure and stateless access control across all microservices.

- Upon successful login, a JWT token is issued containing the user's ID, role (`CUSTOMER`, `RESTAURANT_ADMIN`, `DELIVERY_PERSONNEL`, or `SYSTEM_ADMIN`), and expiry time.

- This token is attached to the `Authorization` header as `Bearer <token>` in all subsequent requests to protected endpoints.

4.2. Role-Based Access Control (RBAC)

- Endpoints are secured based on user roles:

  - Customer: Can browse menus, place orders, and view delivery tracking.

  - Restaurant Admin: Can manage restaurant info, menus, and incoming orders.

  - Delivery Personnel: Can accept, view, and mark deliveries.

  - Admin: Can approve restaurant registrations, manage user accounts, and oversee payments.

- Spring Security (or chosen framework) handles access control via annotations like `@PreAuthorize("hasRole('ROLE_CUSTOMER')")`.

4.3. Password Security

- User passwords are hashed using bcrypt before storage in the database.

- During login, passwords are verified using a secure hash comparison to prevent timing attacks.

4.4. Secure Communication

- All internal and external API calls are secured using HTTPS.

- Sensitive information (passwords, tokens) is never logged or exposed in responses.

4.5. Token Expiry and Refresh

- Tokens have a defined short expiry time (e.g., 15–30 minutes) to reduce attack risk.

- A refresh token mechanism is used to allow session continuation without repeated logins.

4.6. Service-to-Service Communication

- Services validate tokens using a shared public key or a central Auth Service for token verification (depending on implementation).

- Internal calls may also use mutual TLS or API keys between trusted services (optional advanced security).

4.7. Security Best Practices Followed

- CORS policies are set up to restrict cross-origin access.

- Input validation and sanitization are performed to prevent XSS and injection attacks.

- Rate limiting and error handling are applied to avoid brute-force attacks.

6. Individual Contributions

a. G W L Malkith

b. R A K N Ranathunga

7. Code Snippets

8. Appendix