



Sri Lanka Institute of Information Technology

Distributed System –SE3020

Cloud-Native Food Ordering & Delivery System using Microservices

Year 3, Semester 1 – 2025 Assignment 1

Group Id: SE3020_Y3S1_WE_22

Details of the Group Members:

Student ID	Student Name
IT22371768	Ranasinghe S.I
IT22303684	Abeygunasekara D.T
IT22251114	Jinad Induwithwa A.G
IT22182678	Rashani K.G.M

Table of Contents

1. Introduction.....	5
2. High-Level Architecture	6
2.1 System Overview Diagram.....	7
2.2 ER diagram	8
2.3 Use case diagram	9
2.4 MongoDB Schema	10
2.5 Sequence diagrams	11
2.5.1 Login.....	11
2.5.2 Make Order & Payment	12
2.5.3 Delivery Management	13
2.5.4 Restaurant Management	14
2.5.5 Notification Management.....	15
3. Microservices Overview	16
Auth Service	16
Order Service	16
Restaurant Service	16
Payment Service	16
Delivery Service	16
Cart Service	17
Driver Service.....	17
Notification Service.....	17
4. Service Interfaces.....	18
5. System Workflows.....	22
5.1 User Registration & Authentication	22
5.2 Placing an Order	22
5.3 Payment Processing.....	22
5.4 Delivery Assignment & Tracking	22
5.5 Notification Workflow	22
5.6 Restaurant Management Workflow.....	22
5.7 Admin Dashboard Workflow	23
5.8 Restaurant Management Workflow.....	25
Tools and Technologies.....	26
Frontend Folder Structure.....	26

Backend Folder Structure	27
6. Authentication and Security	28
7. Individual Contributions	29
8. Conclusion	30
9. References.....	31
10. Appendix.....	32
10.1 api-gateway.....	32
10.2 auth-service.....	37
10.3 cart-service.....	66
10.4 delivery-service	76
10.5 driver-service.....	86
10.6 notification-service	98
10.7 order-service	109
10.8 payment-service.....	120
10.9 restaurant-service.....	132
10.10 Dockerfile	166
10.11 docker-compose.yml	166
10.12 api-gateway-deployment.yaml	169
10.13 auth -deployment.yaml	170
10.14 cart-deployment.yaml	172
10.15 delivery-deployment.yaml.....	173
10.16 driver-deployment.yaml	174
10.17 mongodb-deployment.yaml	175
10.18 namespace-deployment.yaml	176
10.19 notification-deployment.yaml	177
10.20 order-deployment.yaml	178
10.21 payment-deployment.yaml	180
10.22 persistent-volume-claims.yaml.....	181
10.23 restaurant-deployment.yaml	182
10.24 secrets.yaml	183
10.12 Frontend.....	185

Table of Figures

Figure 1: High-Level Architecture Diagram	6
Figure 2: System Overview Diagram	7
Figure 3: ER diagram.....	8
Figure 4: Use Case Diagram.....	9
Figure 5: MongoDB Schema	10
Figure 6: Sequence of Login	11
Figure 7: Sequence Diagram of Make Order & Payment	12
Figure 8: Sequence diagram of Delivery Management.....	13
Figure 9: Sequence diagram of Restaurant Management.....	14
Figure 10: Sequence diagram of Notification Management	15
Figure 11: Frontend Folder Structure	26
Figure 12: Backend Folder Structure.....	27

1. Introduction

In today's fast-paced world, convenient and efficient food ordering is essential for both customers and restaurants. This report describes the creation of a microservices-based cloud-native food ordering and delivery system. While allowing restaurant owners and delivery personnel to effectively manage their respective responsibilities, the system also facilitates an online platform like PickMe Food or UberEats, which allows users to browse restaurants, place orders, and track deliveries. Restaurant management, order management, delivery assignment, secure payment processing with notification systems are among the essential services. Session management and different backend code have been created for the backend services in order to ensure security. The MERN stack has been utilized in terms of tools and technologies. Tailwind CSS was used for styling, and React JS was used for front-end development. NodeJS and Express have been used for the backend, and MongoDB has been used for data storage. Docker and Kubernetes have been used to deploy the application. The payment gateway is called Payhere. Email and SMS notifications are sent using Nodemailer. All things considered, it offers features for the three primary roles of delivery personnel, restaurant admin, and customers. Authentication, order management, delivery management, restaurant, cart, driver, payment, and notification services are main services.

Objectives:

- Simplify the process of ordering food online.
- Enhance customer satisfaction through real-time tracking and notifications.
- Optimize restaurant and delivery operations.
- Ensure secure payments and robust authentication.

2. High-Level Architecture

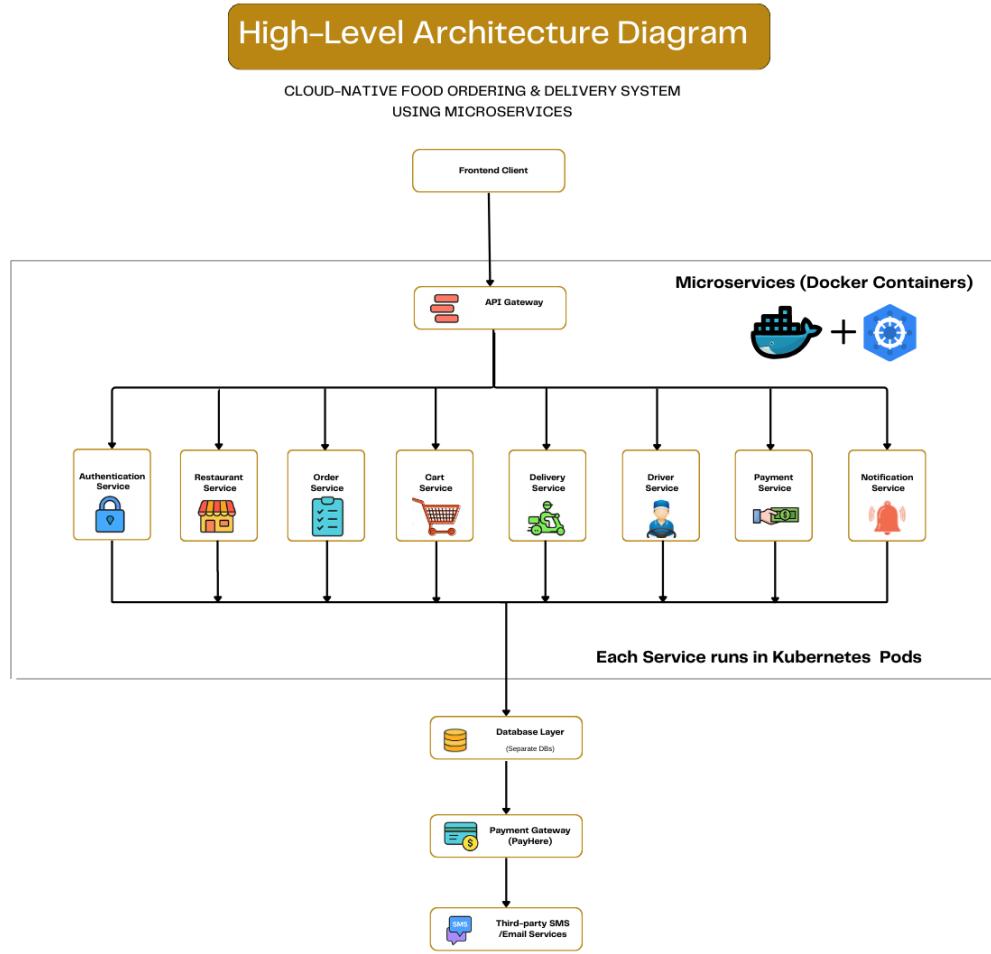


Figure 1: High-Level Architecture Diagram

The system is composed of the following microservices, each independently deployable and scalable:

- API Gateway: Entry point for all client requests, handling routing, authentication, and load balancing.
- User Service: Manages user registration, authentication, and roles (Customer, Restaurant Admin, Delivery Personnel).
- Restaurant Service: Allows restaurant admins to manage menus and orders.
- Order Service: Handles order placement, modification, and tracking.
- Delivery Service: Assigns delivery personnel and tracks deliveries in real-time.
- Payment Service: Integrates with PayHere for secure payment processing.
- Notification Service: Sends SMS and email notifications to customers and delivery personnel.

All services communicate via RESTful APIs and are containerized using Docker and orchestrated with Kubernetes.

2.1 System Overview Diagram

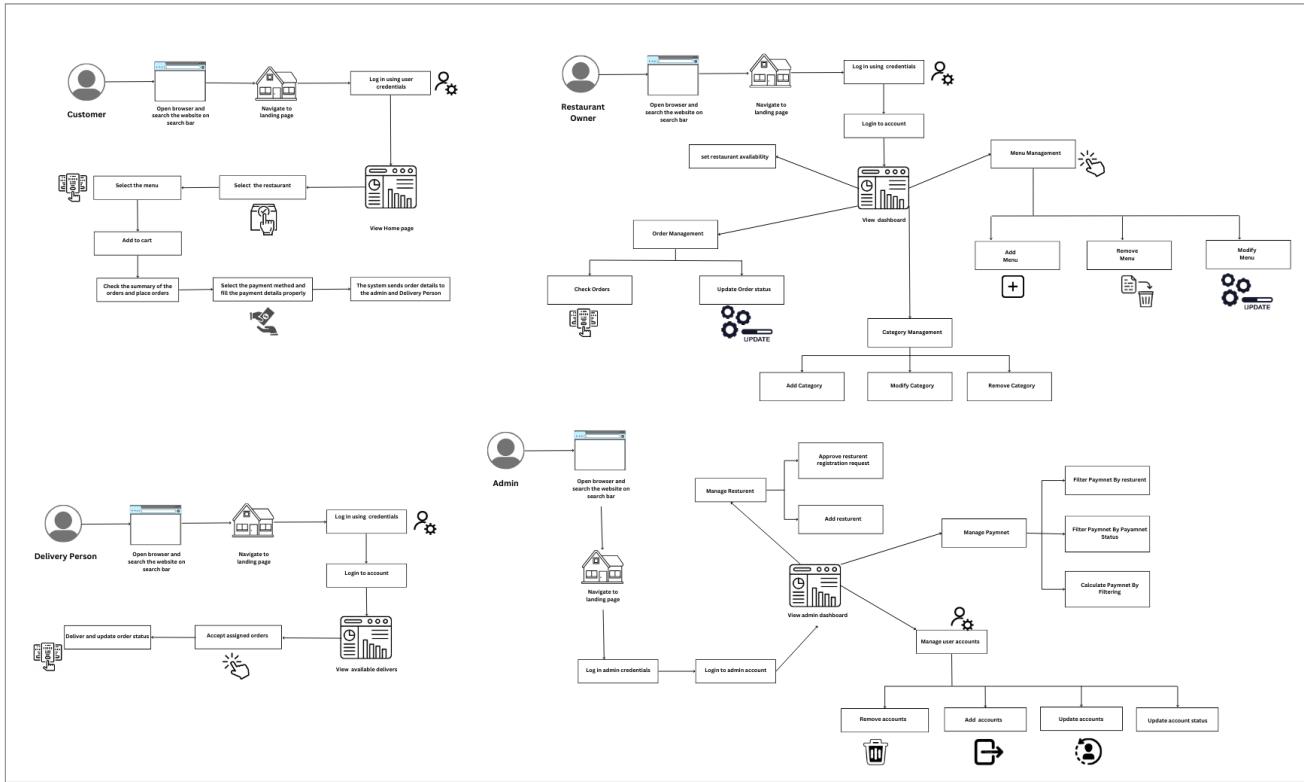


Figure 2: System Overview Diagram

2.2 ER diagram

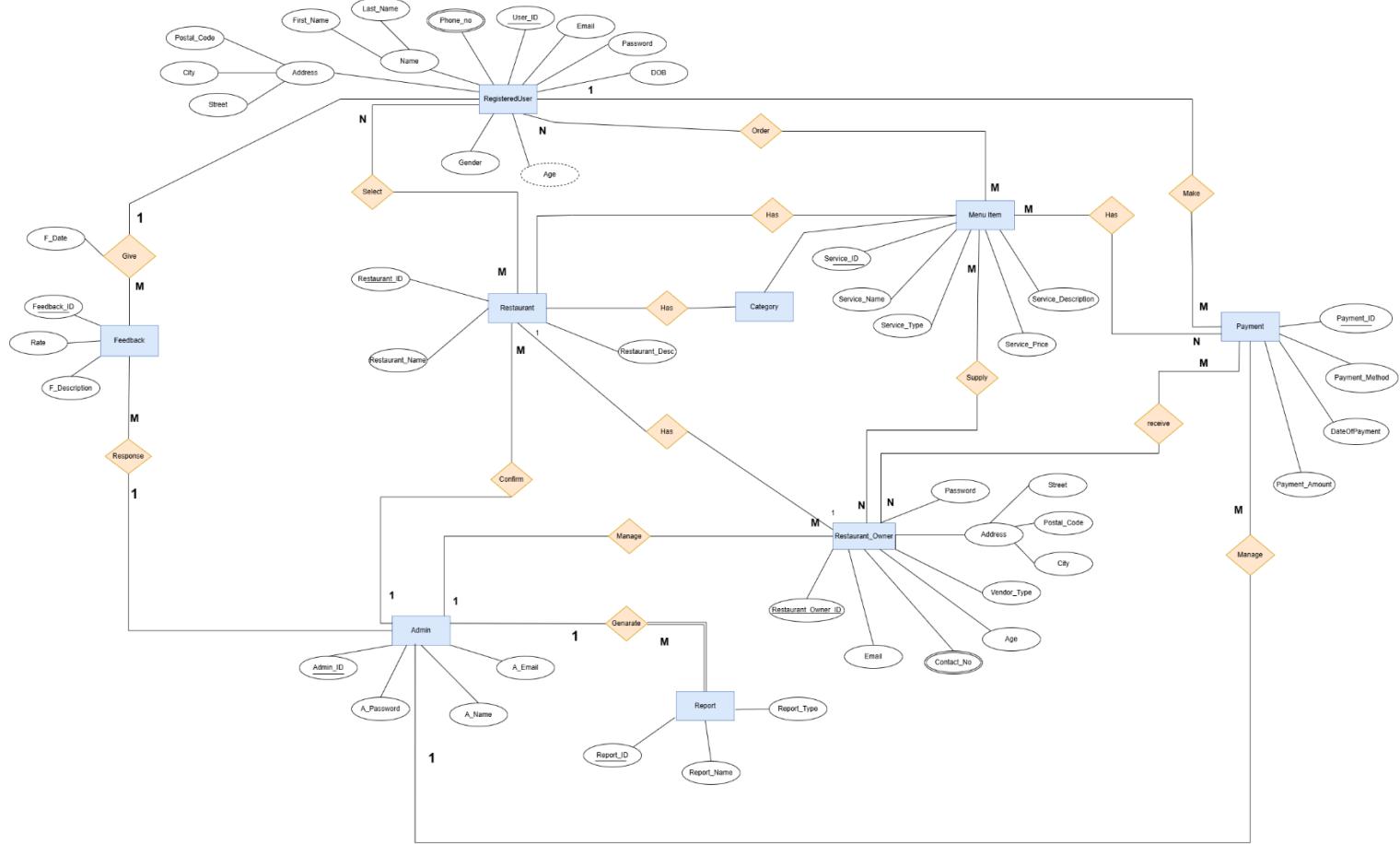


Figure 3: ER diagram

2.3 Use case diagram

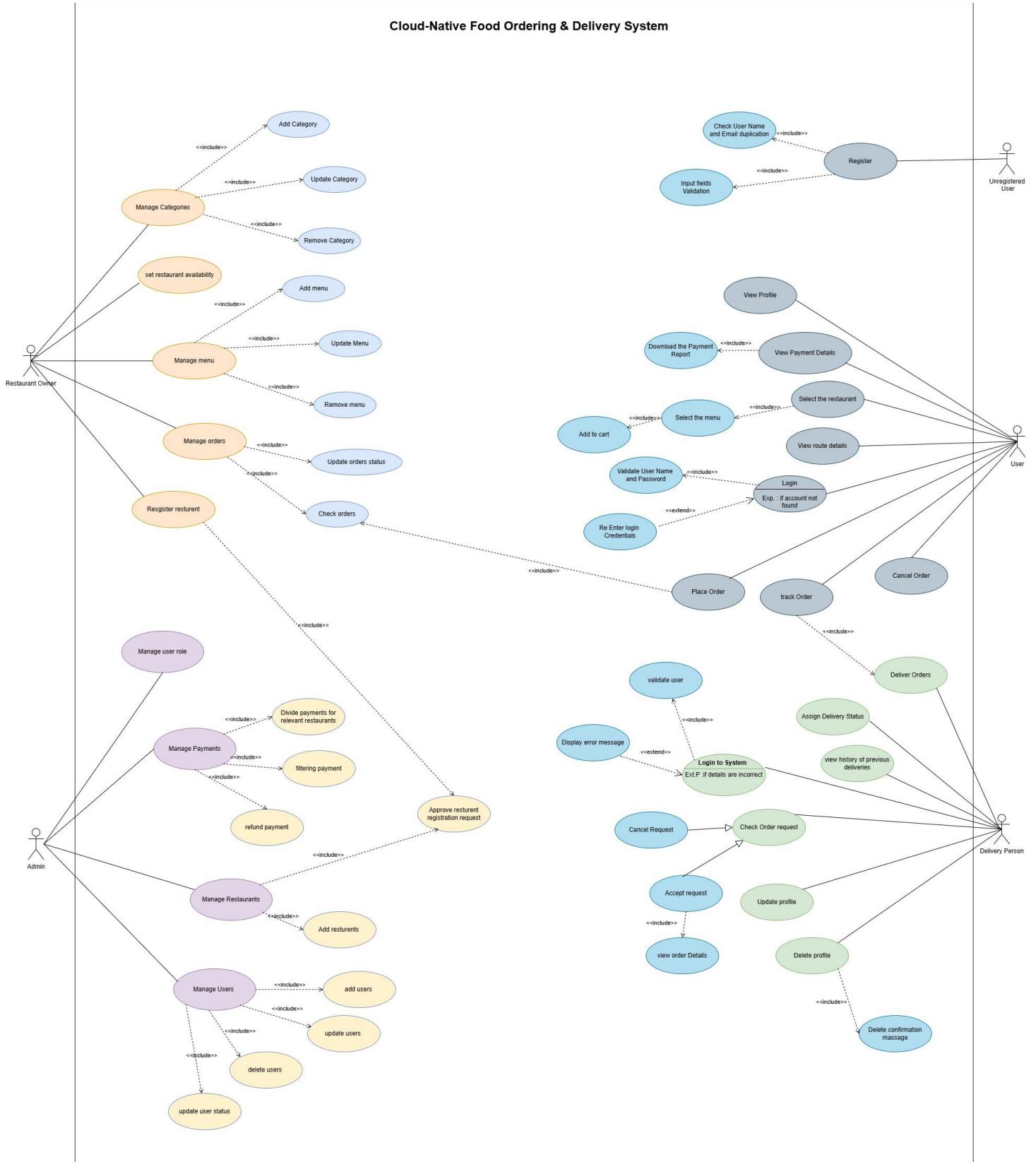


Figure 4: Use Case Diagram

2.4 MongoDB Schema

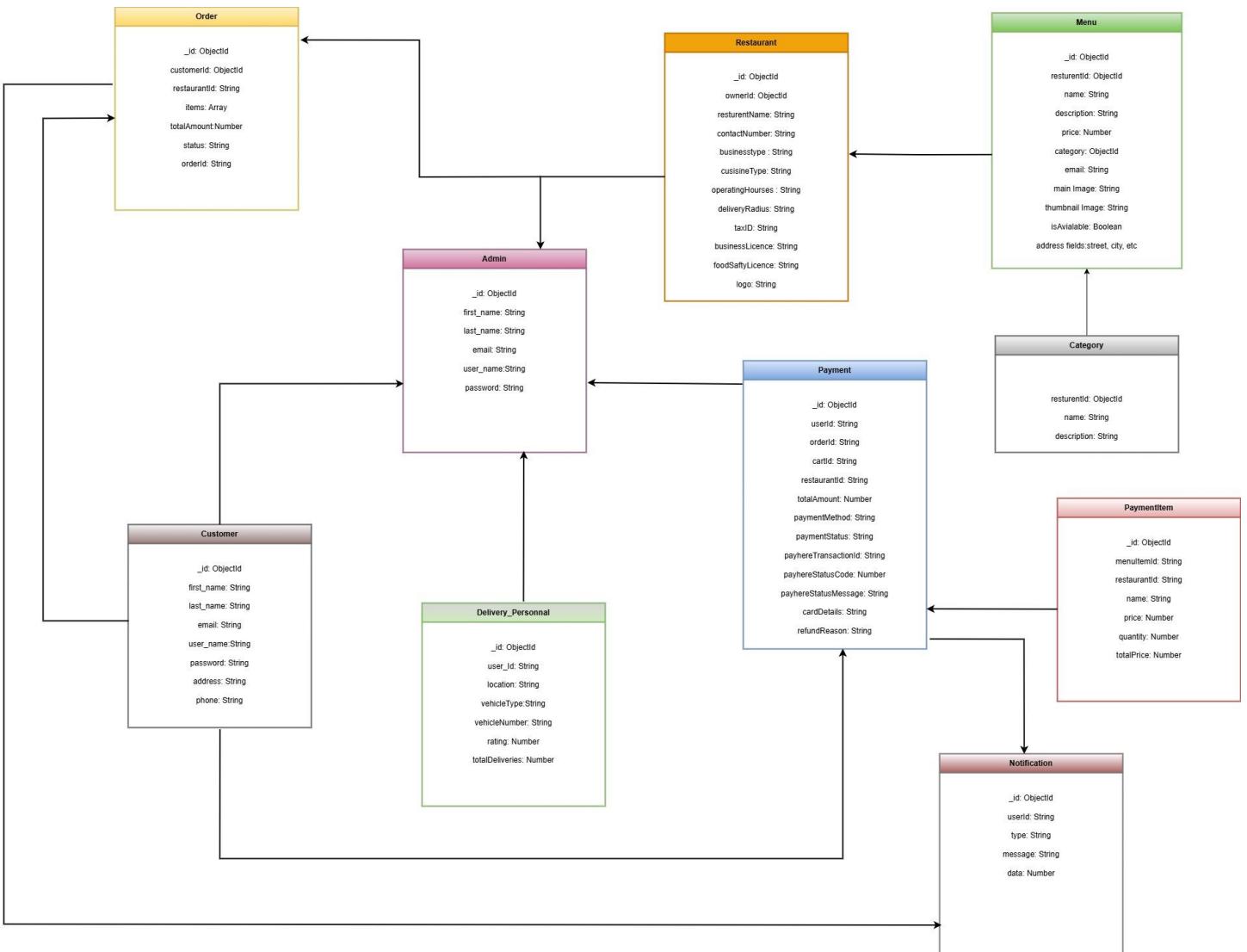


Figure 5: MongoDB Schema

2.5 Sequence diagrams

2.5.1 Login

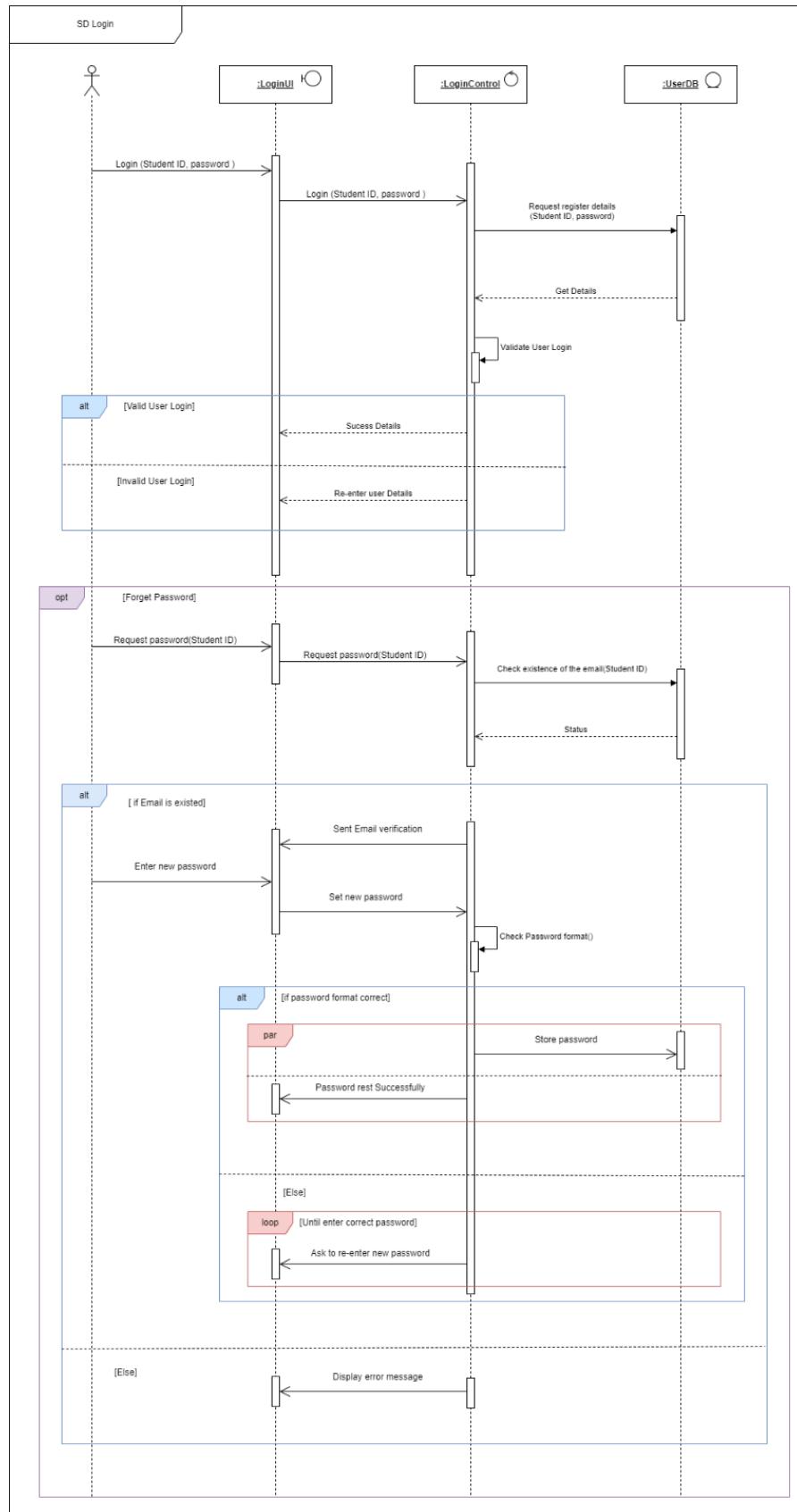


Figure 6: Sequence of Login

2.5.2 Make Order & Payment

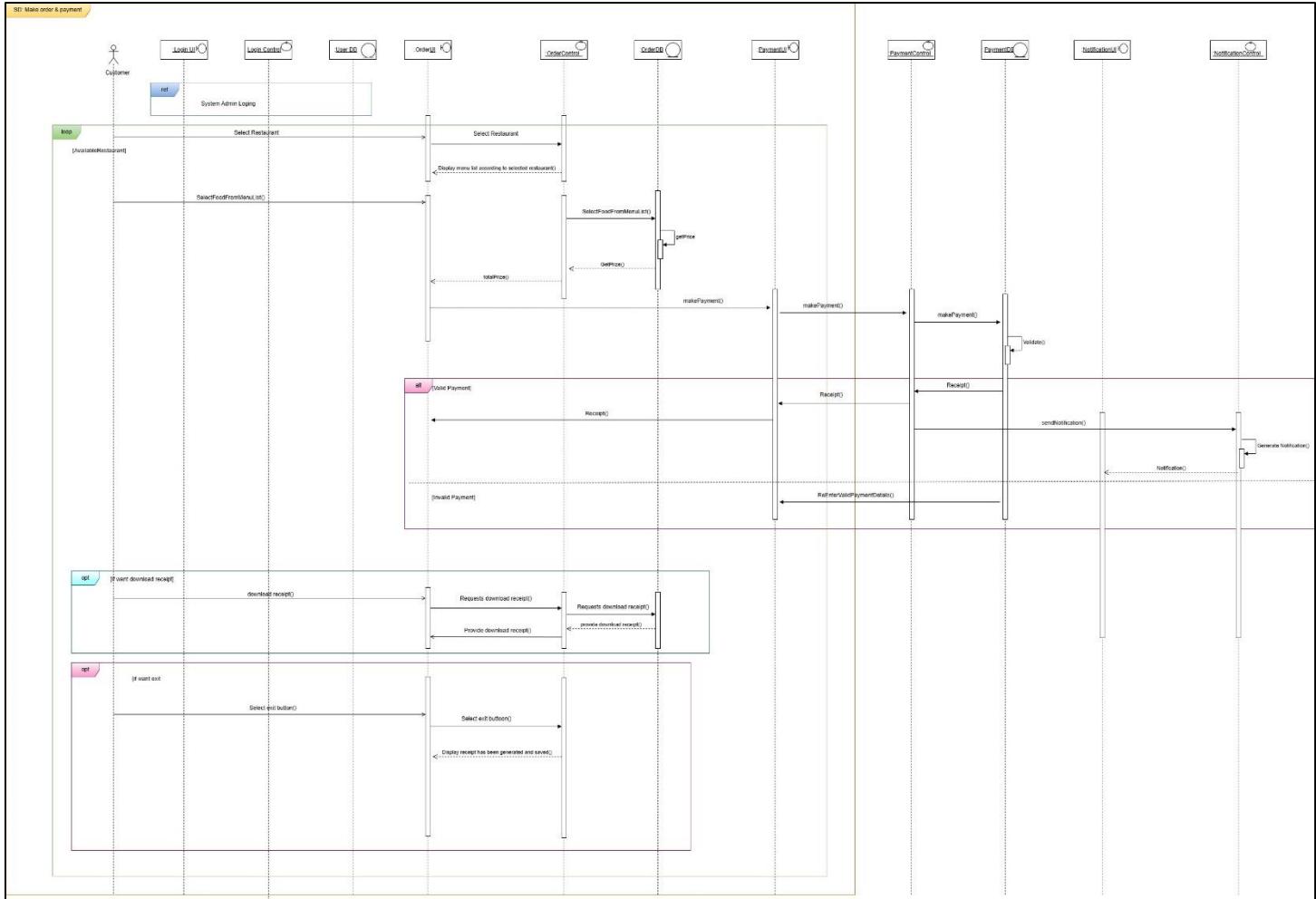


Figure 7: Sequence Diagram of Make Order & Payment

2.5.3 Delivery Management

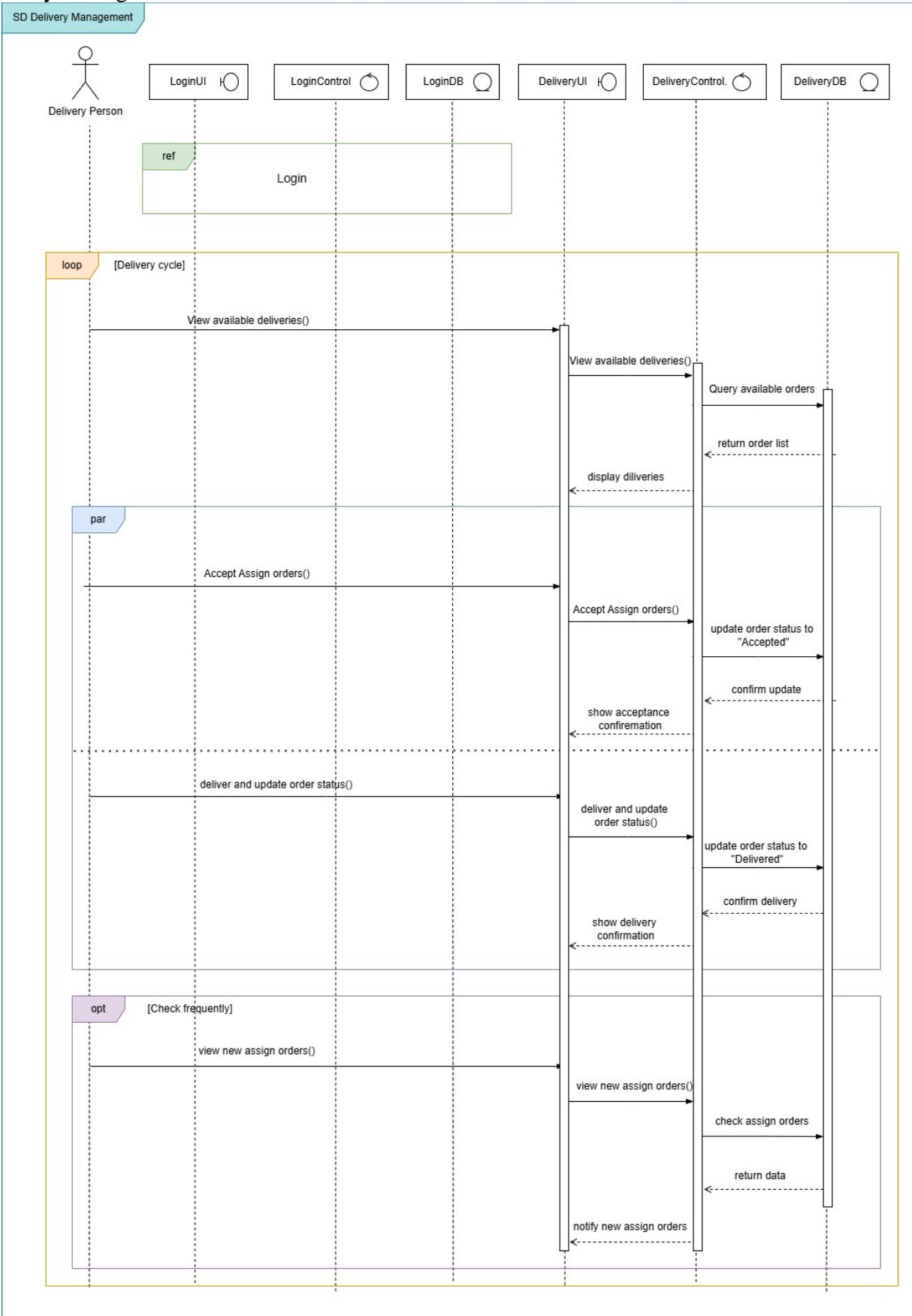


Figure 8: Sequence diagram of Delivery Management

2.5.4 Restaurant Management

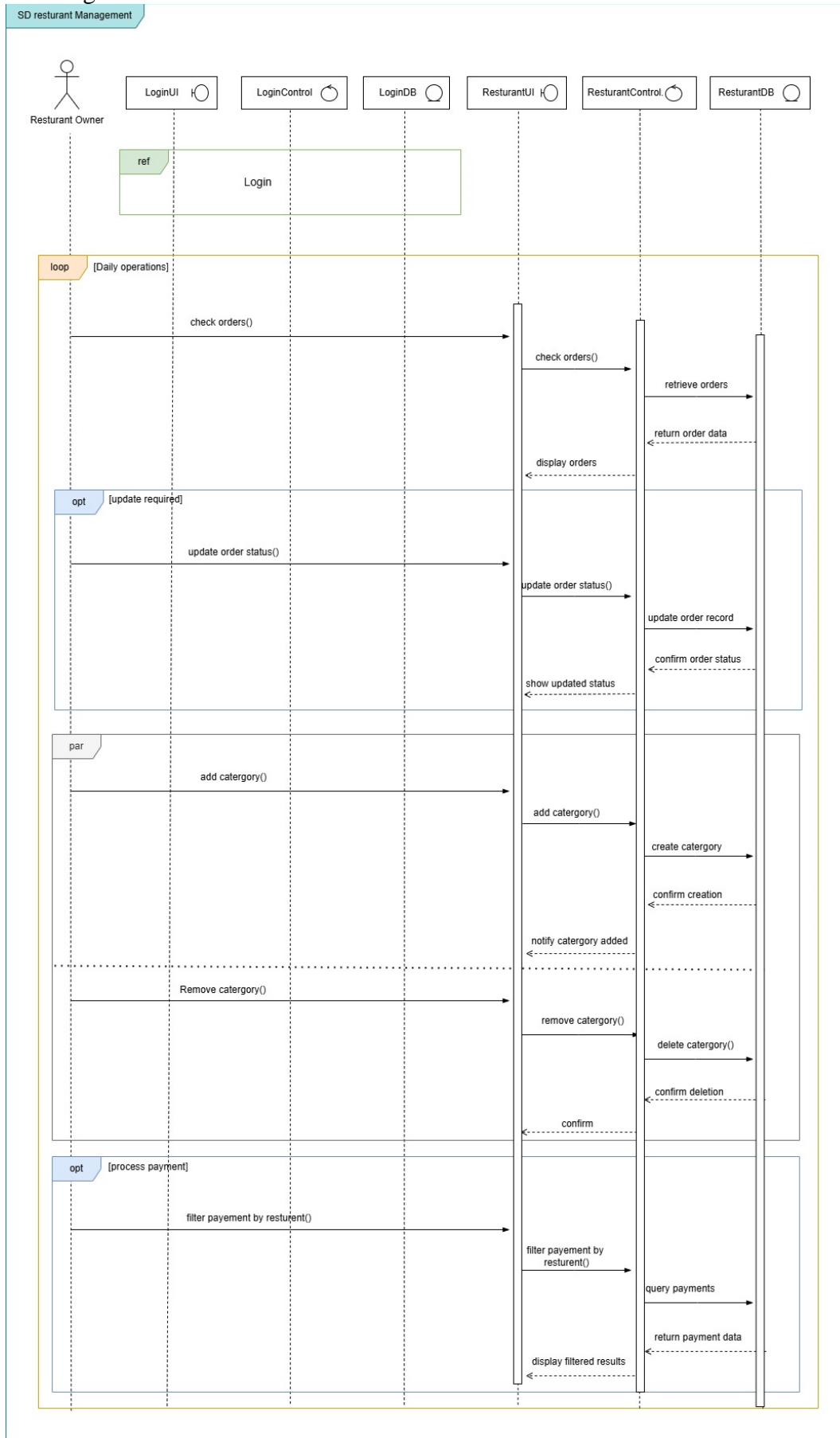


Figure 9: Sequence diagram of Restaurant Management

2.5.5 Notification Management

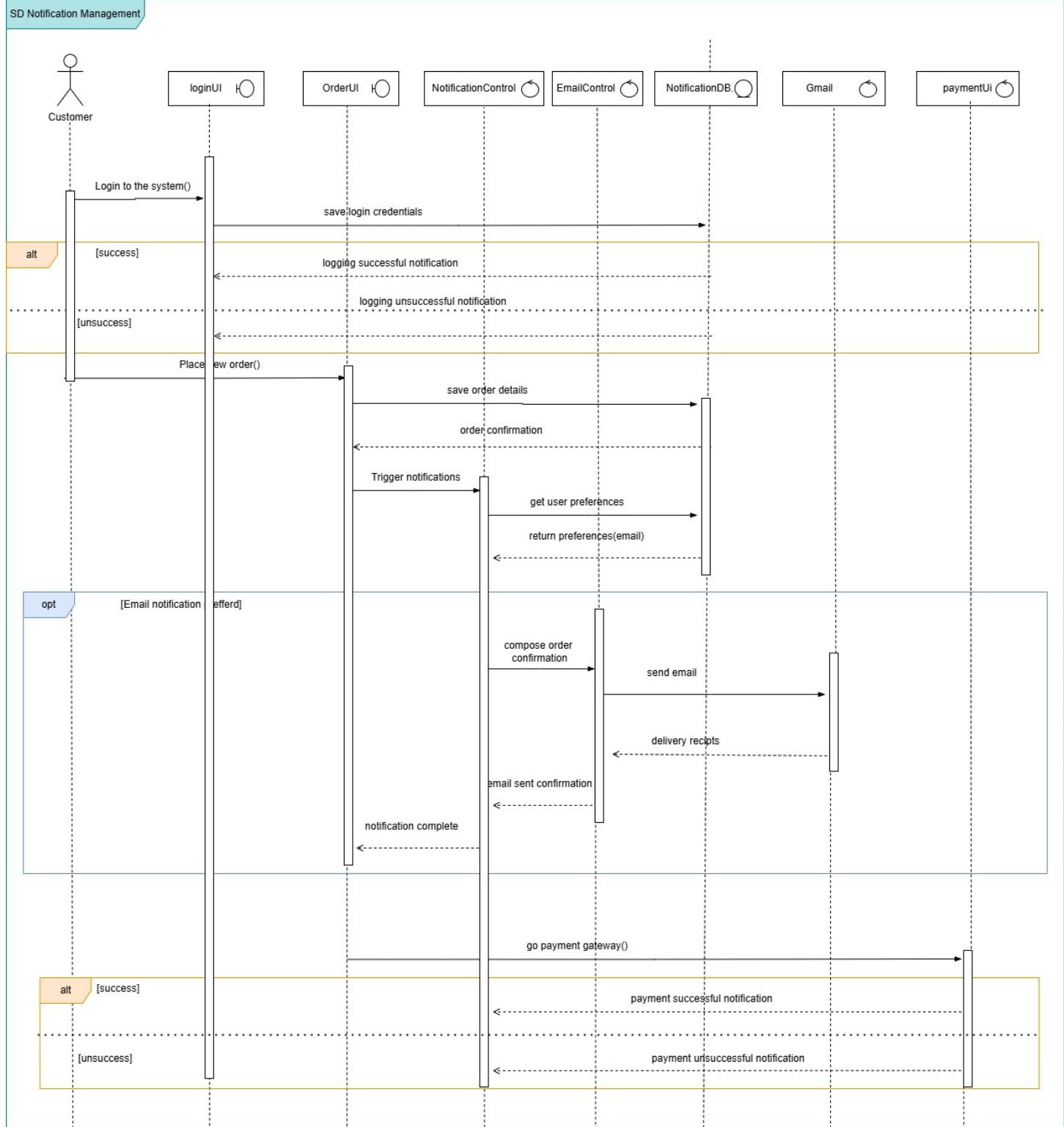


Figure 10: Sequence diagram of Notification Management

3. Microservices Overview

Auth Service

The auth-service manages user registration, authentication, roles, and profiles. Customers, restaurant admins, and delivery personnel are among the various user types it supports. JWT-based authentication ensures secure access. The service is essential for implementing role-based access control throughout the platform and provides user management APIs.

Order Service

The Order Service manages the entire order lifecycle. It handles order creation, updates, status tracking, and history. When a customer places an order, this service validates the request, stores the order data, and coordinates with other services (such as Payment and Delivery) to ensure smooth fulfillment. It also provides APIs for retrieving order history and status, supporting both customers and restaurant staff. The microservices approach ensures that order processing is accurate, efficient, and easily extendable for analytics or new business requirements.

Restaurant Service

Restaurant profiles, menus, categories, orders and associated information are managed by the restaurant service. This service is used by restaurant administrators or owners to manage availability, add new items, set prices, and update their menus. Additionally, it makes APIs available for listing eateries and their menus to customers. By decoupling restaurant management from other concerns, this service allows for independent scaling and rapid updates, ensuring restaurants can efficiently manage their offerings.

Payment Service

All payments are processed securely by the Payment Service. It manages payment initiation, confirmation, and status updates by integrating with third-party payment gateways (PayHere). By separating sensitive financial operations from other business logic, this service improves security and complies with regulations. It communicates with the Order Service to confirm payment completion and trigger order fulfillment workflows.

Delivery Service

The Delivery Service manages the assignment and tracking of deliveries. Once an order is ready, this service assigns available delivery personnel, tracks delivery status in real-time, and updates both customers and restaurants. It ensures efficient routing, handles delivery status changes, and supports notifications for timely updates. The service can be extended to integrate with mapping or real-time tracking APIs for enhanced delivery experiences.

Cart Service

The Cart Service enables customers to add, update, or remove items from their shopping cart before placing an order. It maintains the current state of the cart for each user, supports quantity adjustments, and calculates totals. This service ensures that cart operations are fast and isolated, and can be independently scaled to handle peak browsing and ordering times.

Driver Service

The Driver Service manages delivery personnel profiles, availability, and assignments. It allows drivers to update their status (online/offline), view assigned deliveries, and update delivery progress. By separating driver management from delivery logistics, this service supports efficient workforce management and can be integrated with HR or payroll systems as needed.

Notification Service

The Notification Service is responsible for sending real-time notifications to users, restaurant admins, and delivery personnel via email, SMS, or messaging apps. It is designed for high scalability and reliability, supporting both simple and bulk notifications. The service can integrate with providers like SendGrid, Twilio, or WhatsApp, and ensures that all stakeholders are promptly informed about order updates, delivery status, and promotions.

4. Service Interfaces

Each microservice exposes RESTful APIs to ensure decoupled interactions. Below are the primary interfaces:

❖ auth-Service:

- POST /api/auth/register - validate Registration
- POST /api/auth/login - validate Login
- POST /api/auth/verify-email - validate Email Verification
- POST /api/auth/forgot-password - validateForgotPassword
- POST /api/auth/reset-password - validateResetPassword
- GET /api/auth/profile - getProfile
- POST /api/auth/change-password - validateChangePassword
- POST /api/auth/logout - logout
- POST /api/auth/ verify - verifyAuth
- GET /api/auth/users - getAllUsers
- GET /api/auth/users/:id - getUserId
- PATCH /api/auth /users/:id - updateUser
- DELETE /api/auth /users/:id - deleteUser
- PATCH /api/auth /users/:id/status - updateUserStatus
- PATCH /api/auth /users/:id/role - updateUserRole

❖ Restaurant Service:

- GET /api/restaurants/:restaurantId/categories – Get all categories for a specific restaurant (requires authentication).
- POST /api/restaurants/:restaurantId/categories – Add a new category for a restaurant (validate and authenticate).
- PATCH /api/restaurants/:restaurantId/categories/:categoryId – Update an existing category (validate and authenticate).
- DELETE /api/restaurants/:restaurantId/categories/:categoryId – Delete a category from a restaurant (requires authentication).
- GET /api/restaurants/:restaurantId/menu-items – Fetch all menu items for a specific restaurant (public).
- GET /api/restaurants/:restaurantId/menu-items/:menuItem - Fetch details of a single menu item by ID (public).

- POST /api/restaurants/:restaurantId/menu-items – Add a new menu item (requires authentication, file upload, and validation).
- PATCH /api/restaurants/:restaurantId/menu-items/:menuItemID – Update an existing menu item (requires authentication, file upload, and validation).
- DELETE /api/restaurants/:restaurantId/menu-items/:menuItemID – Delete a menu item (requires authentication).
- GET /api/restaurants/all – Fetch all restaurants (public).
- GET /api/restaurants/:id – Fetch details of a restaurant by ID (public).
- POST /api/restaurants/register – Register a new restaurant (requires file upload and validation).
- GET /api/restaurants/ – Get the restaurant linked to the authenticated user (requires authentication).
- PATCH /api/restaurants/availability – Update restaurant availability status (requires authentication and validation).
- PATCH /api/restaurants/:id – Update restaurant status (requires authentication and admin privileges).
- PUT /api/restaurants/:id – Update restaurant information (requires authentication and file upload).
- DELETE /api/restaurants/:id – Delete a restaurant (requires authentication and admin privileges).

❖ Order Service:

- POST /api/orders/ – Create a new order (validate order details).
- GET /api/orders/:orderId – Get order details by order ID.
- GET /api/orders/user/:userId – Get all orders of a specific user.
- GET /api/orders/restaurant/:restaurantId – Get all orders of a specific restaurant.
- PATCH /api/orders/:orderId/status – Update the status of an order (validate order status).
- PATCH /api/orders/:orderId/payment – Update the payment status of an order (validate payment status).
- POST /api/orders/:orderId/cancel – Cancel an existing order.
- DELETE /api/orders/:orderId – Delete an order by order ID.

❖ Delivery Service:

- POST delivery/assign - Assign delivery driver
- PUT delivery/:deliveryId/status - Update delivery status
- GET api/delivery/:deliveryId/status - get delivery status
- GET /api/delivery/:deliveryId/location - get driver location
- GET delivery/order/:orderId - get delivery by order id

❖ Notification Service:

- POST /api/email/reject-restaurant – Send restaurant rejection email.
- POST /api/email/approve-restaurant – Send restaurant approval email.
- POST /api/email/block-restaurant – Send restaurant blocked email.
- POST /api/email/verify – Send user verification email.
- POST /api/email/reset-password – Send password reset email.
- POST /api/email/order-confirmation – Send order confirmation email.
- POST /api/email/order-status – Send order status update email.
- POST /api/email/payment-confirmation – Send payment confirmation email.
- POST /api/ notification/ – Create a new notification.
- GET /api/ notification/:userId – Get all notifications for a specific user.
- PATCH /api/ notification/:notificationId/read – Mark a specific notification as read.
- DELETE /api/ notification/:notificationId – Delete a specific notification.

❖ Payment Service:

- POST /api/payments/process – Initiate and process a new payment.
- POST /api/payments/notify – Handle payment notifications (e.g., from payment gateway).
- GET /api/payments/return – Handle user return after payment completion.
- GET /api/payments/cancel – Handle payment cancellation by user.
- GET /api/payments/all – Retrieve all payment records.
- POST /api/payments/refund – Process a refund for a completed payment.

❖ Cart Service:

- GET /api/carts/:userId – Get the cart items for a specific user.
- POST /api/carts/:userId/items – Add a menu item to the user's cart (with validation).
- PATCH /api/carts/:userId/items/:menuItemID – Update quantity or details of an item in the cart (with validation).
- DELETE /api/carts/:userId/items/:menuItemID – Remove a specific menu item from the cart.
- DELETE /api/carts/:userId – Clear all items from the user's cart.

❖ Driver Service:

- POST /api/drivers/register – Register a new driver.
- GET /api/drivers/me – Get the current authenticated driver's details.
- PUT /api/drivers/:driverId/location – Update the driver's current location.
- PUT /api/drivers/:driverId/availability – Update the driver's availability status.
- GET /api/drivers/available – Get a list of available drivers (optionally filter by location).
- GET /api/drivers/:driverId – Get details of a specific driver by ID.
- POST /api/drivers/:driverId/assign – Assign a delivery task to a driver.
- POST /api/drivers/:driverId/complete – Mark a delivery as completed by the driver.

5. System Workflows

The system utilizes several workflows to accomplish the end-to-end ordering and delivery process.

5.1 User Registration & Authentication

- User registers via API Gateway.
- User Service creates user and assigns role.
- User logs in and receives JWT token for authentication.

5.2 Placing an Order

- Customer browse restaurants and menus via the Restaurants Service.
- Customer adds items to the cart and places an order via Order Service.
- Order Service creates an order and awaits payment.

5.3 Payment Processing

- Customer initiates payment via Payment Service (/payment/start).
- Payment Service integrates with PayHere for secure transactions.
- Upon successful payment, Payment Service updates order status and triggers Notification Service.

5.4 Delivery Assignment & Tracking

- Delivery Service automatically assigns available delivery personnel.
- Customer receives real-time SMS/email notifications.
- Delivery personnel updates delivery status, which is tracked by the customer.

5.5 Notification Workflow

- Notification Service sends confirmation to customer and delivery personnel upon order placement, assignment, and delivery completion.

5.6 Restaurant Management Workflow

- Delivery personnel log in
- View available deliveries
- Accept assigned orders
- Deliver and update order status

5.7 Admin Dashboard Workflow

The Admin Dashboard provides centralized control over the entire food ordering and delivery platform. It enables the admin to manage restaurants, users, orders, payments, deliveries, and system settings.

Admin Login

- Admin authenticates via the API Gateway.
- JWT token is issued for secure access.

Dashboard Overview

- View real-time analytics: total orders, sales, active users, delivery status, etc.
- Access summary reports and system notifications.

Restaurant Management

- View, approve, or reject new restaurant registrations.
- Update restaurant details.
- Enable/disable restaurant profiles.

User Management

- View and manage customer, restaurant admin, and delivery personnel accounts.
- Reset passwords, update roles, or deactivate users as needed.

Order & Payment Management

- Monitor all orders in real-time.
- Assign or reassign orders to restaurants or delivery personnel.
- Track payment status, generate invoices, and handle refunds or disputes.

Delivery Management

- Track all ongoing deliveries.
- Monitor delivery personnel status and performance.
- Resolve delivery issues.

Reporting & Analytics

- Generate and export reports: sales, payments, order trends, customer insights.
- Use AI-driven analytics to identify business opportunities or issues.

System Settings

- Configure service locations, system currency, SMS/email settings, and third-party integrations (e.g., payment gateways, CRM).
- Manage content (website/app titles, descriptions, keywords).

5.8 Restaurant Management Workflow

The Restaurant Dashboard is designed for restaurant owners to efficiently handle their own operations, menus, orders.

Restaurant Admin Login

- Restaurant admin authenticates via the API Gateway.
- JWT token is issued for secure access.

Dashboard Overview

- View real-time metrics: active orders, sales, menu performance, customer reviews.
- Monitor notifications and alerts.

Menu Management

- Add, update, or remove menu items (including prices, descriptions, images).
- Manage item availability (in stock/out of stock).

Order Management

- Receive and view new orders in real-time.
- Accept, prepare, or reject orders.
- Update order status (preparing, ready for pickup/delivery).
- Communicate with customers or delivery personnel if needed.

Delivery Coordination

- Track delivery assignments and status.
- Coordinate with delivery personnel for timely handoff.

Tools and Technologies

- ❖ Code Editor - VS Code
- ❖ React Version - 18.2.0
- ❖ Node Version - 18.12.1
- ❖ Docker Version – 3.8
- ❖ Package Manager - Yarn

Frontend Folder Structure

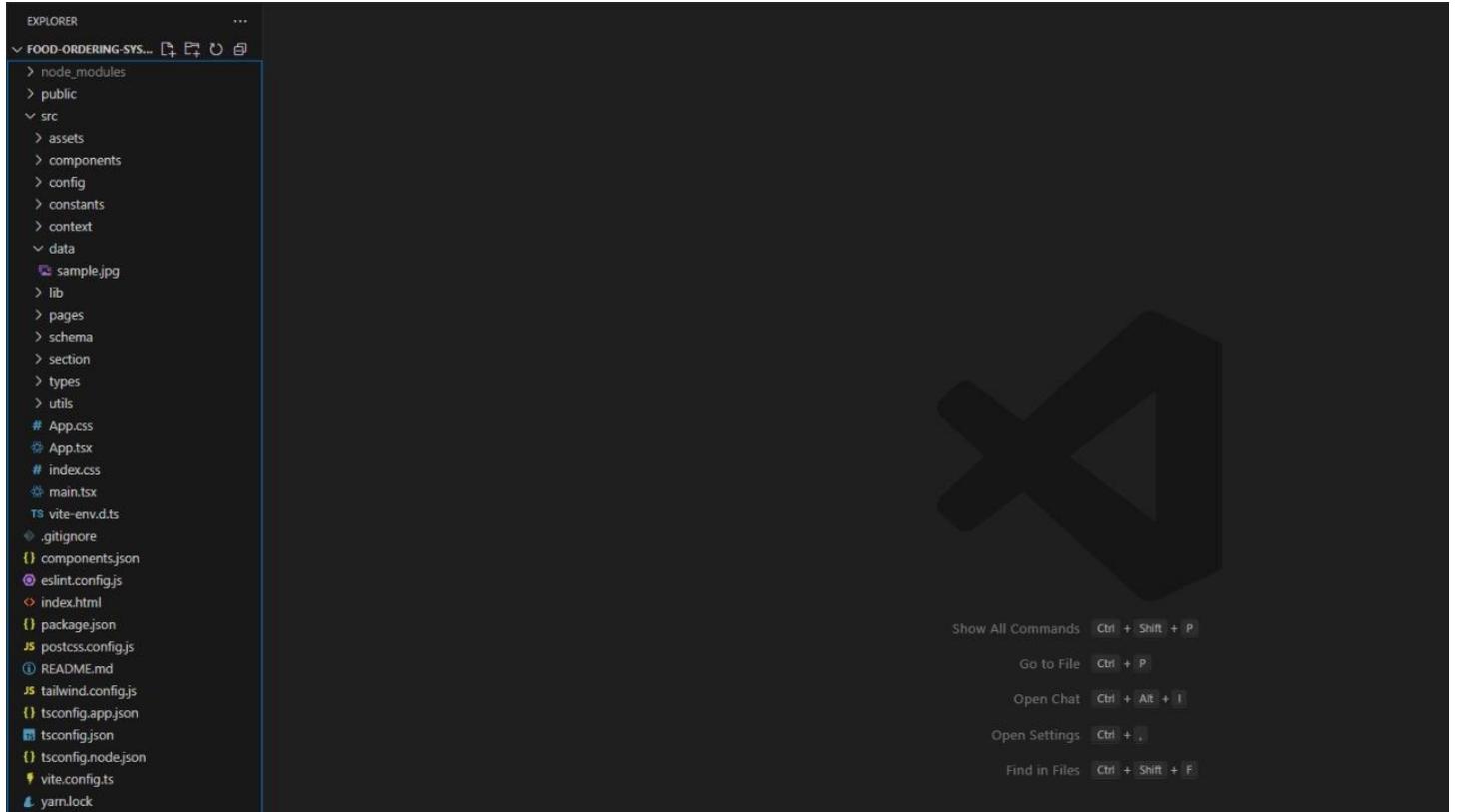


Figure 11: Frontend Folder Structure

Backend Folder Structure

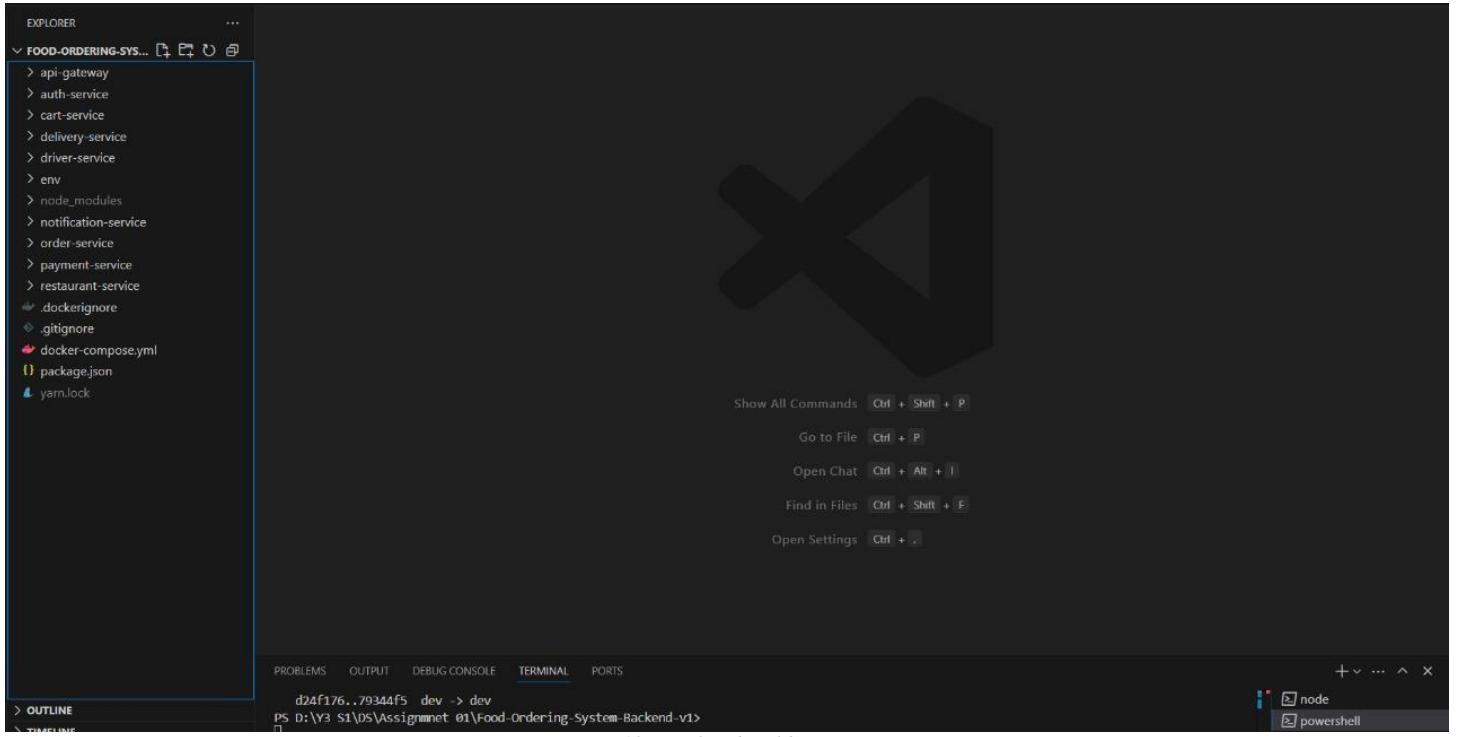


Figure 12: Backend Folder Structure

- ✓ GitHub repository link for Backend :- <https://github.com/IT22303684/Food-Ordering-System-Backend-v1.git>
- ✓ GitHub repository link for Frontend :- <https://github.com/IT22303684/Food-Ordering-System.git>
- ✓ YouTube video link :- <https://youtu.be/dAWgV8OYrQ>

6. Authentication and Security

Security is enforced using JWT-based authentication. Each user role (Customer, Restaurant Admin, Delivery Personnel) is assigned a unique token upon login. Access to service APIs is controlled via role-based access control (RBAC).

All sensitive data transfers are secured over HTTPS.

Passwords are hashed before storing using bcrypt or similar cryptographic functions.

- **JWT Authentication:** All users authenticate via the API Gateway and receive a JWT token, which is required for accessing protected endpoints.
- **Role-Based Access Control:** Endpoints are protected based on user roles (Customer, Restaurant Admin, Delivery Personnel).
- **Secure Payment Integration:** Payment Service uses PayHere's secure APIs; sensitive data is never exposed to the frontend.
- **Data Encryption:** Sensitive data is encrypted in transit (HTTPS) and at rest (database encryption).

7. Individual Contributions

Each member has contributed equally in development and documentation.

Student ID	Student Name	Contribution
IT22371768	Ranasinghe S.I	Frontend- Menu Page, Cart Page, Payment Backend- Payment service, Notification Service Report preparing, high-level architecture diagram, ER diagram, use case diagram, System Overview Diagram
IT22303684	Abeygunasekara D.T	Frontend- Login, Signup, Home, Driver Dashboard Backend- Auth Service, Notification Service, api gateway service, delivery service, driver service sequence diagram, MongoDB schema
IT22251114	Jinad Induwithwa A.G	Frontend- resturent registration page, Resturent Page, resturent terms & condition page Backend- Restaurant Service sequence diagram, use case diagram
IT22182678	Rashani K.G.M	Frontend- Order Page Backend- Order Service sequence diagram

8. Conclusion

The Food Ordering & Delivery System's development as a microservices-based application has illustrated the useful benefits of distributed systems in creating scalable, robust, and maintainable solutions for actual business requirements. We have made sure that every business domain can develop, grow, and be maintained independently by breaking the platform down into separate services, such as order service, restaurant service, payment service, delivery service, cart service, driver service, notification service, authentication service, and API gateway.

This architecture improves system resilience and fault tolerance in addition to streamlining development and facilitating the quick rollout of new features. The platform is made to be both contemporary and expandable through the use of RESTful APIs for inter-service communication, Docker for containerization, and safe integration with third-party services (like PayHere for payments and SMS/email gateways for notifications).

By implementing role-based access control, encrypted communication channels, and JWT-based authentication, security has been given top priority. A smooth user experience is offered by real-time notifications and tracking, and future expansion like adding new payment gateways or entering new markets is supported by the modular design.

In conclusion, this project serves as a scalable basis for future innovation in addition to meeting the needs of a modern food ordering and delivery platform.

9. References

- ❖ mehdihadeli. (2022). Food Delivery Microservices. GitHub Repository.
<https://github.com/mehdihadeli/food-delivery-microservices>
- ❖ Pratap Sharma. (2023). Architecture and Design Principle for Online Food Delivery System.
<https://pratapsharma.io/architecture-of-food-delivery-app/>
- ❖ IJRPR. (2022). A Smart Menu Based Online Food Ordering System with using Microservices Architecture.
<https://ijrpr.com/uploads/V5ISSUE4/IJRPR25865.pdf>
- ❖ LinkedIn Pulse. (2023). Architecture and Design Principle for Online Food Delivery System by Pratap Sharma.
<https://www.linkedin.com/pulse/architecture-design-principle-online-food-delivery-system-sharma>
- ❖ wkrzywiec. (2022). Food Delivery App - DEV Community.
<https://dev.to/wkrzywiec/food-delivery-app-1g2b>
- ❖ GaloisGun. Food-Delivery-Application. GitHub Repository.
<https://github.com/GaloisGun/Food-Delivery-Application>
- ❖ Yu Yang. (2022). Design and Implementation of Online Food Ordering System Based on Springcloud. Clausius Scientific Press.
<https://www.clausiuspress.com/article/5191.html>
- ❖ IJRTE. (2019). Online Food Ordering System.
<https://www.ijrte.org/wp-content/uploads/papers/v8i2S3/B11560782S319.pdf>

10. Appendix

10.1 api-gateway

api-gateway/src/index.js

```
import express from "express";
import { createProxyMiddleware } from "http-proxy-middleware";
import cors from "cors";
import morgan from "morgan";
import dotenv from "dotenv";

dotenv.config();

const app = express();

// Middleware
app.use(
  cors({
    origin: "*", // Allow all origins in development
    methods: ["GET", "POST", "PUT", "DELETE", "PATCH"],
    allowedHeaders: ["Content-Type", "Authorization"],
    credentials: true,
  })
);
app.use(express.json({ limit: "50mb" }));
app.use(express.urlencoded({ extended: true, limit: "50mb" }));
app.use(morgan("dev"));

// Request logging middleware
app.use((req, res, next) => {
  console.log(`Incoming request: ${req.method} ${req.url}`, {
    body: req.body,
    headers: req.headers,
  });
  next();
});

// Proxy configuration
const proxyOptions = {
  changeOrigin: true,
  secure: false,
  timeout: 30000, // 30 seconds timeout
  proxyTimeout: 30000,
  onError: (err, req, res) => {
    console.error("Proxy Error:", err);
    res.status(500).json({
      message: "Service is currently unavailable",
      error: err.message,
    });
  },
  onProxyReq: (proxyReq, req, res) => {
```

```

console.log("Proxying request:", req.method, req.url);
if (
  req.body &&
  req.headers["content-type"] &&
  req.headers["content-type"].includes("application/json")
) {
  const bodyData = JSON.stringify(req.body);
  proxyReq.setHeader("Content-Length", Buffer.byteLength(bodyData));
  proxyReq.write(bodyData);
}
},
onProxyRes: (proxyRes, req, res) => {
  console.log("Received response:", proxyRes.statusCode);
},
};

// Auth Service Proxy
app.use(
  "/api/auth",
  createProxyMiddleware({
    target: process.env.AUTH_SERVICE_URL,
    ...proxyOptions,
    pathRewrite: {
      "^/api/auth": "/api/auth",
    },
  })
);

// Notification Service Proxy
app.use(
  "/api/notifications",
  createProxyMiddleware({
    target: process.env.NOTIFICATION_SERVICE_URL,
    ...proxyOptions,
    pathRewrite: {
      "^/api/notifications": "/api/notifications",
    },
  })
);

// Email Service Proxy
app.use(
  "/api/email",
  createProxyMiddleware({
    target: process.env.NOTIFICATION_SERVICE_URL,
    ...proxyOptions,
    pathRewrite: {
      "^/api/email": "/api/email",
    },
  })
);

// Restaurant Service Proxy

```

```
app.use(
  "/api/restaurants",
  createProxyMiddleware({
    target: process.env.RESTAURANT_SERVICE_URL,
    ...proxyOptions,
    pathRewrite: {
      "^/api/restaurants": "/api/restaurants",
    },
    onError: (err, req, res) => {
      console.error("Restaurant Service Proxy Error:", err.message, {
        target: process.env.RESTAURANT_SERVICE_URL,
        url: req.url,
      });
      res.status(500).json({
        message: "Service is currently unavailable",
        error: err.message,
      });
    },
  })
);

// Cart Service Proxy
app.use(
  "/api/carts",
  createProxyMiddleware({
    target: process.env.CART_SERVICE_URL,
    ...proxyOptions,
    pathRewrite: {
      "^/api/carts": "/api/carts",
    },
    onError: (err, req, res) => {
      console.error("Cart Service Proxy Error:", err.message, {
        target: process.env.CART_SERVICE_URL,
        url: req.url,
      });
      res.status(500).json({
        message: "Service is currently unavailable",
        error: err.message,
      });
    },
  })
);

// Order Service Proxy
app.use(
  "/api/orders",
  createProxyMiddleware({
    target: process.env.ORDER_SERVICE_URL,
    ...proxyOptions,
    pathRewrite: {
      "^/api/orders": "/api/orders",
    },
    onError: (err, req, res) => {
```

```
console.error("Order Service Proxy Error:", err.message, {
  target: process.env.ORDER_SERVICE_URL,
  url: req.url,
});
res.status(500).json({
  message: "Service is currently unavailable",
  error: err.message,
});
},
})
);
// Delivery Service Proxy
app.use(
  "/api/delivery",
  createProxyMiddleware({
    target: process.env.DELIVERY_SERVICE_URL || "http://localhost:3007",
    ...proxyOptions,
    pathRewrite: {
      "^/api/delivery": "/api/delivery",
    },
    onError: (err, req, res) => {
      console.error("Delivery Service Proxy Error:", err.message, {
        target: process.env.DELIVERY_SERVICE_URL,
        url: req.url,
      });
      res.status(500).json({
        message: "Service is currently unavailable",
        error: err.message,
      });
    },
  })
);
// Payment Service Proxy
app.use(
  "/api/payments",
  createProxyMiddleware({
    target: process.env.PAYMENT_SERVICE_URL,
    ...proxyOptions,
    pathRewrite: {
      "^/api/payments": "/api/payments",
    },
    onError: (err, req, res) => {
      console.error("Payment Service Proxy Error:", err.message, {
        target: process.env.PAYMENT_SERVICE_URL,
        url: req.url,
      });
      res.status(500).json({
        message: "Payment is currently unavailable",
        error: err.message,
      });
    },
  })
);
```

```

);

// Driver Service Proxy
app.use(
  "/api/drivers",
  createProxyMiddleware({
    target: process.env.DRIVER_SERVICE_URL || "http://localhost:3008",
    ...proxyOptions,
    pathRewrite: {
      "^/api/drivers": "/api/drivers",
    },
    onError: (err, req, res) => {
      console.error("Driver Service Proxy Error:", err.message, {
        target: process.env.DRIVER_SERVICE_URL,
        url: req.url,
      });
      res.status(500).json({
        message: "Service is currently unavailable",
        error: err.message,
      });
    },
  })
);

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: "Something went wrong!" });
});

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
  console.log(`API Gateway running on port ${PORT}`);
  console.log(`Auth Service URL: ${process.env.AUTH_SERVICE_URL}`);
  console.log(`Notification Service URL: ${process.env.NOTIFICATION_SERVICE_URL}`);
  );
  console.log(`Restaurant Service URL: ${process.env.RESTAURANT_SERVICE_URL}`);
  console.log(`Cart Service URL: ${process.env.CART_SERVICE_URL}`);
  console.log(`Order Service URL: ${process.env.ORDER_SERVICE_URL}`);
  console.log(`Paymnet Service URL: ${process.env.PAYMENT_SERVICE_URL}`);
  console.log(`Delivery Service URL: ${
    process.env.DELIVERY_SERVICE_URL || "http://localhost:3007"
  }`);
);
console.log(`Driver Service URL: ${
  process.env.DRIVER_SERVICE_URL || "http://localhost:3008"
}`);
});
});
```

10.2 auth-service

auth-service/src/controllers/auth.controller.js

```
// auth-service/src/controllers/auth.controller.js
import { AuthService } from "../services/auth.service.js";
import jwt from "jsonwebtoken";
import { User } from "../models/user.model.js";
import logger from "../utils/logger.js";
import axios from "axios";

export class AuthController {
  constructor() {
    this.authService = new AuthService();

    this.register = this.register.bind(this);
    this.login = this.login.bind(this);
    this.getProfile = this.getProfile.bind(this);
    this.verifyEmail = this.verifyEmail.bind(this);
    this.forgotPassword = this.forgotPassword.bind(this);
    this.resetPassword = this.resetPassword.bind(this);
    this.updateProfile = this.updateProfile.bind(this);
    this.changePassword = this.changePassword.bind(this);
    this.logout = this.logout.bind(this);
    this.verifyAuth = this.verifyAuth.bind(this);
    this.getAllUsers = this.getAllUsers.bind(this);
    this.getUserById = this.getUserById.bind(this);
    this.updateUser = this.updateUser.bind(this);
    this.deleteUser = this.deleteUser.bind(this);
    this.updateUserStatus = this.updateUserStatus.bind(this);
    this.updateUserRole = this.updateUserRole.bind(this);
  }

  async register(req, res) {
    try {
      const {
        email,
        password,
        role,
        firstName,
        lastName,
        address,
        vehicleType,
        vehicleNumber,
      } = req.body;
      console.log("Registration request received for:", email);

      const { user } = await this.authService.register({
        email,
        password,
        firstName,
        lastName,
        address,
      });
    }
  }
}
```

```

role: role || "CUSTOMER",
});

// If registering as a driver, create driver profile
if (role === "DELIVERY") {
  try {
    // Call driver service to create driver profile
    await axios.post(
      `${process.env.DRIVER_SERVICE_URL}/api/drivers/register`,
      {
        userId: user._id,
        vehicleType,
        vehicleNumber,
        location: [0, 0], // Default location
      }
    );
  } catch (error) {
    // If driver profile creation fails, delete the user
    await User.findByIdAndDelete(user._id);
    throw new Error("Failed to create driver profile");
  }
}

const token = jwt.sign(
  { userId: user._id, email: user.email, role: user.role },
  process.env.JWT_SECRET,
  { expiresIn: "1d" }
);

console.log("Registration successful for:", email);
res.status(201).json({
  message:
    "User registered successfully. Please check your email for verification code.",
  token,
  user: {
    id: user._id,
    email: user.email,
    role: user.role,
    firstName: user.firstName,
    lastName: user.lastName,
    address: user.address,
    isVerified: user.isVerified,
  },
});
} catch (error) {
  console.error("Registration error:", error.message);
  console.error("Error stack trace:", error.stack); // Log the full stack trace
  if (error.name === "ValidationError") {
    console.error(
      "Validation errors:",
      Object.values(error.errors).map((err) => err.message)
    );
    return res.status(400).json({

```

```
        message: "Invalid data",
        errors: Object.values(error.errors).map((err) => err.message),
    });
}

if (error.isOperational) {
    console.error("Operational error:", error.message);
    return res.status(error.statusCode).json({
        message: error.message,
    });
}
console.error("Server error:", error.message);
res.status(500).json({ message: "Server error" });
}

}

async login(req, res) {
try {
    const { email, password } = req.body;
    console.log("Login attempt for:", email);

    // Find user by email
    const user = await User.findOne({ email });
    console.log("User found:", user ? "Yes" : "No");

    if (!user) {
        console.log("Login failed: User not found");
        return res.status(401).json({ message: "Invalid credentials" });
    }

    // Check if user is verified
    if (!user.isVerified) {
        console.log("Login failed: User not verified");
        return res
            .status(401)
            .json({ message: "Please verify your email first" });
    }

    // Compare password
    const isMatch = await user.comparePassword(password);
    console.log("Password match:", isMatch);

    if (!isMatch) {
        console.log("Login failed: Invalid password");
        return res.status(401).json({ message: "Invalid credentials" });
    }

    // Generate JWT token
    const token = jwt.sign(
        { userId: user._id, email: user.email, role: user.role },
        process.env.JWT_SECRET,
        { expiresIn: "1d" }
    );
}
```

```

// Set cookie with token
res.cookie("token", token, {
  httpOnly: true,
  secure: process.env.NODE_ENV === "production",
  sameSite: "strict",
  maxAge: 24 * 60 * 60 * 1000, // 1 day in milliseconds
});

console.log("Login successful");
res.status(200).json({
  message: "Login successful",
  token,
  user: {
    id: user._id,
    email: user.email,
    firstName: user.firstName,
    lastName: user.lastName,
    role: user.role,
  },
});
} catch (error) {
  console.error("Login error:", error.message);
  res.status(500).json({ message: error.message });
}
}

async getProfile(req, res) {
  try {
    const user = await User.findById(req.user.id).select("-password");
    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }
    res.json(user);
  } catch (error) {
    logger.error("Get profile error:", error);
    res.status(500).json({ message: "Error fetching profile" });
  }
}

async verifyEmail(req, res) {
  try {
    const { pin } = req.body;
    await this.authService.verifyEmail(pin);

    res.json({
      status: "success",
      message: "Email verified successfully",
    });
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
}

```

```
async forgotPassword(req, res) {
  try {
    const resetToken = await this.authService.forgotPassword(req.body.email);

    res.json({
      status: "success",
      message: "Password reset instructions sent to email",
    });
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
}

async resetPassword(req, res) {
  try {
    await this.authService.resetPassword(req.params.token, req.body.password);

    res.json({
      status: "success",
      message: "Password reset successful",
    });
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
}

async updateProfile(req, res) {
  try {
    const updatedUser = await this.authService.updateProfile(
      req.user.id,
      req.body
    );
    res.json({
      status: "success",
      data: {
        user: updatedUser,
      },
    });
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
}

async changePassword(req, res) {
  try {
    await this.authService.changePassword(
      req.user.id,
      req.body.currentPassword,
      req.body.newPassword
    );

    res.json({
      status: "success",
    });
  }
}
```

```
    message: "Password changed successfully",
  });
} catch (error) {
  res.status(400).json({ message: error.message });
}
}

logout(req, res) {
  // Clear the token cookie
  res.clearCookie("token", {
    httpOnly: true,
    secure: process.env.NODE_ENV === "production",
    sameSite: "strict",
  });

  res.status(200).json({
    status: "success",
    message: "Logged out successfully",
  });
}

async verifyAuth(req, res) {
  try {
    // The authMiddleware already verified the token and attached the user
    const user = await User.findById(req.user.id).select("-password");

    if (!user) {
      return res.status(404).json({
        success: false,
        message: "User not found",
      });
    }

    // Return user information
    res.status(200).json({
      success: true,
      user: {
        id: user._id,
        email: user.email,
        firstName: user.firstName,
        lastName: user.lastName,
        role: user.role,
        isVerified: user.isVerified,
        isActive: user.isActive,
      },
    });
  } catch (error) {
    logger.error("Auth verification error:", error);
    res.status(500).json({
      success: false,
      message: "Authentication verification failed",
    });
  }
}
```

```

}

//-----admin controllers-----

//sent welcome email
async sendWelcomeEmail(req, res) {
  try {
    const { email, password } = req.body;
    if (!email || !password) {
      return res.status(400).json({ message: "Email and password are required" });
    }
  }

  const resetToken = await authService.sendWelcomeEmail(email, password);

  res.json({
    status: "success",
    message: "Welcome email sent successfully",
    data: { resetToken },
  });
} catch (error) {
  logger.error("Send welcome email error:", { error: error.message, stack: error.stack });
  res.status(error.statusCode || 500).json({ message: error.message });
}
}

//get all users
async getAllUsers(req, res) {
  try {
    const users = await this.authService.getAllUsers();
    res.status(200).json({
      status: "success",
      data: { users },
    });
  } catch (error) {
    logger.error("Get all users error:", error);
    res.status(error.statusCode || 500).json({ message: error.message });
  }
}

//get user by id
async getUserById(req, res) {
  try {
    const user = await this.authService.getUserById(req.params.id);
    res.status(200).json({
      status: "success",
      data: { user },
    });
  } catch (error) {
    logger.error("Get user by ID error:", error);
    res.status(error.statusCode || 500).json({ message: error.message });
  }
}

//update user
async updateUser(req, res) {
  try {

```

```
const updatedUser = await this.authService.updateUser(
  req.params.id,
  req.body
);
res.status(200).json({
  status: "success",
  data: { user: updatedUser },
});
} catch (error) {
  logger.error("Update user error:", error);
  res.status(error.statusCode || 500).json({ message: error.message });
}
}

// delete user
async deleteUser(req, res) {
  try {
    await this.authService.deleteUser(req.params.id);
    res.status(200).json({
      status: "success",
      message: "User deleted successfully",
    });
  } catch (error) {
    logger.error("Delete user error:", error);
    res.status(error.statusCode || 500).json({ message: error.message });
  }
}

//update user status
async updateUserStatus(req, res) {
  try {
    const updatedUser = await this.authService.updateUserStatus(
      req.params.id,
      req.body
    );
    res.status(200).json({
      status: "success",
      data: { user: updatedUser },
    });
  } catch (error) {
    logger.error("Update user status error:", error);
    res.status(error.statusCode || 500).json({ message: error.message });
  }
}

//update user role
async updateUserRole(req, res) {
  try {
    const updatedUser = await this.authService.updateUserRole(
      req.params.id,
      req.body.role
    );
    res.status(200).json({
      status: "success",
      data: { user: updatedUser },
    });
  }
```

```

    } catch (error) {
      logger.error("Update user role error:", error);
      res.status(error.statusCode || 500).json({ message: error.message });
    }
  }
}

```

auth-service/src/middleware/auth.middleware.js

```

import jwt from "jsonwebtoken";
import { User } from "../models/user.model.js";
import logger from "../utils/logger.js";

export const auth = async (req, res, next) => {
  try {
    // Get token from header or cookie
    const token =
      req.header("Authorization")?.replace("Bearer ", "") || req.cookies.token;

    if (!token) {
      return res
        .status(401)
        .json({ message: "No token, authorization denied" });
    }

    // Verify token
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Find user
    const user = await User.findById(decoded.userId);
    if (!user) {
      return res.status(401).json({ message: "User not found" });
    }

    // Attach user to request
    req.user = user;
    next();
  } catch (error) {
    logger.error("Authentication error:", error);
    res.status(401).json({ message: "Token is not valid" });
  }
};

export const authorize = (...roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({
        message: "You do not have permission to perform this action",
      });
    }
    next();
  };
};

```

auth-service/src/models/user.model.js

```
// auth-service/src/models/user.model.js
import mongoose from "mongoose";
import bcrypt from "bcryptjs";

const userSchema = new mongoose.Schema(
{
  email: {
    type: String,
    required: [true, "Email is required"],
    unique: true,
    lowercase: true,
    trim: true,
  },
  password: {
    type: String,
    required: [true, "Password is required"],
    minlength: 6,
    select: true, // Changed to true to include password in queries
  },
  firstName: {
    type: String,
    required: [true, "First name is required"],
    trim: true,
  },
  lastName: {
    type: String,
    required: [true, "Last name is required"],
    trim: true,
  },
  address: {
    street: {
      type: String,
      required: [true, "Street address is required"],
      trim: true,
    },
    city: {
      type: String,
      required: [true, "City is required"],
      trim: true,
    },
    state: {
      type: String,
      required: [true, "State is required"],
      trim: true,
    },
    zipCode: {
      type: String,
      required: [true, "Zip code is required"],
      trim: true,
    },
    country: {

```

```
        type: String,
        required: [true, "Country is required"],
        trim: true,
    },
},
phone: {
    type: String,
    trim: true,
    match: [/^\+?[1-9]\d{1,14}$/, "Please provide a valid phone number"],
},
role: {
    type: String,
    enum: ["CUSTOMER", "RESTAURANT", "DELIVERY", "ADMIN"],
    default: "CUSTOMER",
},
isVerified: {
    type: Boolean,
    default: false,
},
verificationPin: {
    type: String,
    select: false,
},
verificationPinExpires: {
    type: Date,
    select: false,
},
verificationToken: String,
resetPasswordToken: {
    type: String,
    select: false,
},
resetPasswordExpires: {
    type: Date,
    select: false,
},
isActive: {
    type: Boolean,
    default: true,
},
createdAt: {
    type: Date,
    default: Date.now,
},
{
    timestamps: true,
}
);

// Hash password before saving
userSchema.pre("save", async function (next) {
    if (!this.isModified("password")) return next();
})
```

```

try {
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
} catch (error) {
  next(error);
}
});

// Compare password method
userSchema.methods.comparePassword = async function (candidatePassword) {
  return bcrypt.compare(candidatePassword, this.password);
};

export const User = mongoose.model("User", userSchema);

```

auth-service/src/routes/auth.routes.js

```

// auth-service/src/routes/auth.routes.js
import express from "express";
import { AuthController } from "../controllers/auth.controller.js";
import { auth, authorize } from "../middleware/auth.middleware.js";
import {
  validateRegistration,
  validateLogin,
  validateEmailVerification,
  validateForgotPassword,
  validateResetPassword,
  validateProfileUpdate,
  validateChangePassword,
  validateUpdateUser,
  validateUpdateUserStatus,
  validateUpdateUserRole,
} from "../validation/auth.validation.js";

const router = express.Router();
const authController = new AuthController();

// Public routes
router.post("/register", validateRegistration, authController.register);
router.post("/login", validateLogin, authController.login);
router.post(
  "/verify-email",
  validateEmailVerification,
  authController.verifyEmail
);
router.post(
  "/forgot-password",
  validateForgotPassword,
  authController.forgotPassword
);

```

```
router.post(
  "/reset-password/:token",
  validateResetPassword,
  authController.resetPassword
);

// Protected routes
router.get("/profile", auth, authController.getProfile);
router.patch(
  "/profile",
  auth,
  validateProfileUpdate,
  authController.updateProfile
);
router.post(
  "/change-password",
  auth,
  validateChangePassword,
  authController.changePassword
);
router.post("/logout", auth, authController.logout);

// Authentication verification endpoint for other services
router.post("/verify", auth, authController.verifyAuth);

// Admin routes
//router.get("/users", auth, authorize("ADMIN"), );
router.get("/users", auth, authorize("ADMIN"), authController.getAllUsers);
router.get("/users/:id", auth, authorize("ADMIN"), authController.getUserById);
router.patch(
  "/users/:id",
  auth,
  authorize("ADMIN"),
  validateUpdateUser,
  authController.updateUser
);
router.delete("/users/:id", auth, authorize("ADMIN"), authController.deleteUser);
router.patch(
  "/users/:id/status",
  auth,
  authorize("ADMIN"),
  validateUpdateUserStatus,
  authController.updateUserStatus
);
router.patch(
  "/users/:id/role",
  auth,
  authorize("ADMIN"),
  validateUpdateUserRole,
  authController.updateUserRole
);

export default router;
```

auth-service/src/services/auth.service.js

```
// auth-service/src/services/auth.service.js
import { User } from "../models/user.model.js";
import logger from "../utils/logger.js";
import jwt from "jsonwebtoken";
import crypto from "crypto";
import bcrypt from "bcryptjs";
import axios from "axios";
import { emailClient } from "./email.client.js";

const JWT_SECRET = process.env.JWT_SECRET || "your-secret-key";
const JWT_EXPIRES_IN = process.env.JWT_EXPIRES_IN || "24h";
const EMAIL_SERVICE_URL = process.env.EMAIL_SERVICE_URL;

class AppError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
    this.status = `${statusCode}`.startsWith("4") ? "fail" : "error";
    this.isOperational = true;
    Error.captureStackTrace(this, this.constructor);
  }
}

export class AuthService {
  generateToken(user) {
    try {
      logger.info("Generating JWT token for user:", { userId: user._id });
      const token = jwt.sign(
        {
          userId: user._id,
          role: user.role,
          email: user.email,
        },
        JWT_SECRET,
        { expiresIn: JWT_EXPIRES_IN }
      );
      logger.info("JWT token generated successfully");
      return token;
    } catch (error) {
      logger.error("Error generating JWT token:", {
        userId: user._id,
        error: error.message,
      });
      throw error;
    }
  }

  async register(userData) {
    try {
      console.log("Starting registration for:", userData.email);
```

```

const existingUser = await User.findOne({ email: userData.email });
if (existingUser) {
  console.log(
    "Registration failed - Email already exists:",
    userData.email
  );
  throw new AppError("Email already registered", 400);
}

const verificationPin = Math.floor(
  100000 + Math.random() * 900000
).toString();
const verificationPinExpires = new Date(Date.now() + 10 * 60 * 1000);
console.log("Generated verification PIN:", verificationPin);

const user = await User.create({
  ...userData,
  verificationPin,
  verificationPinExpires,
  isVerified: false,
});
console.log("User created successfully:", user._id);

try {
  await emailClient.sendVerificationEmail(user.email, verificationPin);
  console.log("Verification email sent to:", user.email);
} catch (emailError) {
  console.error("Failed to send verification email:", emailError.message);
}

return { user };
} catch (error) {
  console.error("Registration error:", error.message);
  throw error;
}
}

async verifyEmail(pin) {
try {
  logger.info("Starting email verification process", { pin });

  const user = await User.findOne({
    verificationPin: pin,
    verificationPinExpires: { $gt: Date.now() },
  });

  if (!user) {
    logger.warn("Email verification failed: Invalid or expired PIN", {
      pin,
    });
    throw new AppError("Invalid or expired verification PIN", 400);
  }
}

```

```

user.isVerified = true;
user.verificationPin = undefined;
user.verificationPinExpires = undefined;
await user.save();

logger.info("Email verified successfully", { userId: user._id });

return user;
} catch (error) {
  logger.error("Email verification failed:", {
    pin,
    error: error.message,
  });
  throw error;
}
}

async login(email, password) {
  try {
    logger.info("Starting login process", { email });

    const user = await User.findOne({ email }).select("+password");
    if (!user) {
      logger.warn("Login failed: User not found", { email });
      throw new AppError("Invalid email or password", 401);
    }

    const isPasswordValid = await user.comparePassword(password);
    if (!isPasswordValid) {
      logger.warn("Login failed: Invalid password", { email });
      throw new AppError("Invalid email or password", 401);
    }

    if (!user.isVerified) {
      logger.warn("Login failed: Email not verified", { email });
      throw new AppError(
        "Please verify your email first. Check your inbox for the verification PIN.",
        403
      );
    }
  }

  logger.info("Login successful", { userId: user._id });
  return user;
} catch (error) {
  logger.error("Login failed:", {
    email,
    error: error.message,
    stack: error.stack,
  });
  throw error;
}
}

```

```

async forgotPassword(email) {
  const user = await User.findOne({ email });
  if (!user) {
    throw new AppError("User not found", 404);
  }

  const resetToken = crypto.randomBytes(32).toString("hex");
  user.resetPasswordToken = resetToken;
  user.resetPasswordExpires = Date.now() + 3600000; // 1 hour
  await user.save();

  try {
    // Send password reset email through notification service
    await axios.post(`${EMAIL_SERVICE_URL}/reset-password`, {
      email: user.email,
      resetToken,
    });
    logger.info("Password reset email sent successfully");
  } catch (emailError) {
    logger.error("Failed to send password reset email:", {
      userId: user._id,
      error: emailError.message,
    });
    // Don't throw the error, just log it
  }

  return resetToken;
}

async resetPassword(token, newPassword) {
  const user = await User.findOne({
    resetPasswordToken: token,
    resetPasswordExpires: { $gt: Date.now() },
  });

  if (!user) {
    throw new AppError("Invalid or expired reset token", 400);
  }

  user.password = await bcrypt.hash(newPassword, 10);
  user.resetPasswordToken = undefined;
  user.resetPasswordExpires = undefined;
  await user.save();

  logger.info("Password reset successfully", { userId: user._id });

  return user;
}

async updateProfile(userId, updateData) {
  const user = await User.findById(userId);
  if (!user) {
    throw new AppError("User not found", 404);
  }
}

```

```

}

const allowedUpdates = ["firstName", "lastName", "phone"];
Object.keys(updateData).forEach((key) => {
  if (allowedUpdates.includes(key)) {
    user[key] = updateData[key];
  }
});

await user.save();
return user;
}

async changePassword(userId, currentPassword, newPassword) {
  const user = await User.findById(userId).select("+password");
  if (!user) {
    throw new AppError("User not found", 404);
  }

  const isPasswordValid = await bcrypt.compare(
    currentPassword,
    user.password
  );
  if (!isPasswordValid) {
    throw new AppError("Current password is incorrect", 400);
  }

  user.password = await bcrypt.hash(newPassword, 10);
  await user.save();
}

//----- admin services -----

async sendWelcomeEmail(email, password) {
  // Find user by email
  const user = await User.findOne({ email });
  if (!user) {
    throw new AppError("User not found", 404);
  }

  // Generate reset token
  const resetToken = crypto.randomBytes(32).toString("hex");
  user.resetPasswordToken = resetToken;
  user.resetPasswordExpires = Date.now() + 3600000; // 1 hour
  await user.save();

  // Prepare reset URL
  const resetUrl = `http://localhost:3010/reset-password/${resetToken}`;

  try {
    // Send welcome email with password and reset link
    await emailService.sendWelcomeEmail(user.email, user.firstName, password, resetUrl);
    logger.info("Welcome email sent successfully", { userId: user._id, email });
  }
}

```

```

} catch (emailError) {
  logger.error("Failed to send welcome email:", {
    userId: user._id,
    error: emailError.message,
  });
  // Don't throw the error, just log it to ensure registration isn't blocked
}

return resetToken;
}

// get all users
async getAllUsers() {
  try {
    logger.info("Fetching all users");
    const users = await User.find()
      .select("-password -verificationPin -resetPasswordToken -verificationPinExpires -resetPasswordExpires")
      .lean();
    logger.info("Successfully fetched all users", { count: users.length });
    return users;
  } catch (error) {
    logger.error("Error fetching all users:", error);
    throw new AppError("Failed to fetch users", 500);
  }
}
// get user by id
async getUserById(userId) {
  try {
    logger.info("Fetching user by ID:", { userId });
    const user = await User.findById(userId)
      .select("-password -verificationPin -resetPasswordToken -verificationPinExpires -resetPasswordExpires")
      .lean();
    if (!user) {
      logger.warn("User not found:", { userId });
      throw new AppError("User not found", 404);
    }
    logger.info("Successfully fetched user:", { userId });
    return user;
  } catch (error) {
    logger.error("Error fetching user by ID:", error);
    if (error instanceof AppError) throw error;
    throw new AppError("Failed to fetch user", 500);
  }
}
// update user
async updateUser(userId, updateData) {
  try {
    logger.info("Updating user:", { userId });
    const user = await User.findByIdAndUpdate(userId);
    if (!user) {
      logger.warn("User not found:", { userId });
      throw new AppError("User not found", 404);
    }
  }
}

```

```

// Check for email uniqueness if email is being updated
if (updateData.email && updateData.email !== user.email) {
  const existingUser = await User.findOne({ email: updateData.email });
  if (existingUser) {
    logger.warn("Email already exists:", { email: updateData.email });
    throw new AppError("Email already registered", 400);
  }
}

// Update allowed fields
const allowedUpdates = [
  "email",
  "firstName",
  "lastName",
  "phone",
  "role",
  "address.street",
  "address.city",
  "address.state",
  "address.zipCode",
  "address.country",
];
Object.keys(updateData).forEach((key) => {
  if (allowedUpdates.includes(key)) {
    if (key.startsWith("address.")) {
      const addressField = key.split(".")[1];
      user.address[addressField] = updateData[key];
    } else {
      user[key] = updateData[key];
    }
  }
});
};

// Handle password update if provided
if (updateData.password) {
  user.password = updateData.password; // Will be hashed by pre-save hook
}

await user.save();
logger.info("User updated successfully:", { userId });

// Return user without sensitive fields
return await User.findById(userId)
  .select("-password -verificationPin -resetPasswordToken -verificationPinExpires -resetPasswordExpires")
  .lean();
} catch (error) {
  logger.error("Error updating user:", error);
  if (error instanceof AppError) throw error;
  throw new AppError("Failed to update user", 500);
}
}
// delete user

```

```

async deleteUser(userId) {
  try {
    logger.info("Deleting user:", { userId });
    const user = await User.findById(userId);
    if (!user) {
      logger.warn("User not found:", { userId });
      throw new AppError("User not found", 404);
    }

    await user.deleteOne();
    logger.info("User deleted successfully:", { userId });
  } catch (error) {
    logger.error("Error deleting user:", error);
    if (error instanceof AppError) throw error;
    throw new AppError("Failed to delete user", 500);
  }
}

// update user status
async updateUserStatus(userId, statusData) {
  try {
    logger.info("Updating user status:", { userId });
    const user = await User.findById(userId);
    if (!user) {
      logger.warn("User not found:", { userId });
      throw new AppError("User not found", 404);
    }

    // Update status fields
    if (typeof statusData.isActive !== "undefined") {
      user.isActive = statusData.isActive;
    }
    if (typeof statusData.isVerified !== "undefined") {
      user.isVerified = statusData.isVerified;
    }

    await user.save();
    logger.info("User status updated successfully:", { userId });

    // Return user without sensitive fields
    return await User.findById(userId)
      .select("-password -verificationPin -resetPasswordToken -verificationPinExpires -resetPasswordExpires")
      .lean();
  } catch (error) {
    logger.error("Error updating user status:", error);
    if (error instanceof AppError) throw error;
    throw new AppError("Failed to update user status", 500);
  }
}

// update user role
async updateUserRole(userId, role) {
  try {
    logger.info("Updating user role:", { userId, role });
    const user = await User.findById(userId);
  }
}

```

```

if (!user) {
  logger.warn("User not found:", { userId });
  throw new AppError("User not found", 404);
}

user.role = role;
await user.save();
logger.info("User role updated successfully:", { userId, role });

return await User.findById(userId)
  .select("-password -verificationPin -resetPasswordToken -verificationPinExpires -resetPasswordExpires")
  .lean();
} catch (error) {
  logger.error("Error updating user role:", error);
  if (error instanceof AppError) throw error;
  throw new AppError("Failed to update user role", 500);
}
}
}
}

```

auth-service/src/services/email.client.js

```

import axios from "axios";
import logger from "../utils/logger.js";

const EMAIL_SERVICE_URL =
  process.env.EMAIL_SERVICE_URL || "http://localhost:3003/api/email";

class EmailClient {
  async sendVerificationEmail(email, pin) {
    try {
      await axios.post(`${EMAIL_SERVICE_URL}/verify`, {
        email,
        pin,
      });
      logger.info("Verification email sent successfully", { email });
      return true;
    } catch (error) {
      logger.error("Failed to send verification email", {
        email,
        error: error.message,
      });
      throw error;
    }
  }

  async sendPasswordResetEmail(email, token) {
    try {
      await axios.post(`${EMAIL_SERVICE_URL}/reset-password`, {
        email,
        token,
      });
    }
  }
}

```

```
    logger.info("Password reset email sent successfully", { email });
    return true;
} catch (error) {
  logger.error("Failed to send password reset email", {
    email,
    error: error.message,
  });
  throw error;
}
}

export const emailClient = new EmailClient();
```

auth-service/src/utils/logger.js

```
import winston from "winston";

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || "info",
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.Console({
      format: winston.format.combine(
        winston.format.colorize(),
        winston.format.simple()
      ),
    }),
  ],
});
};

export default logger;
```

auth-service/src/validation/auth.validation.js

```
import { body, validationResult } from "express-validator";

// Common validation middleware
export const validateRequest = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }
  next();
};

// User validation rules
export const userValidationRules = [
  body("email").isEmail().withMessage("Please provide a valid email"),
```

```
body("password")
    .isLength({ min: 6 })
    .withMessage("Password must be at least 6 characters long"),
];
// Login validation rules
export const loginValidationRules = [
    body("email").isEmail().withMessage("Please provide a valid email"),
    body("password").notEmpty().withMessage("Password is required"),
];
// Password reset validation rules
export const passwordResetValidationRules = [
    body("email").isEmail().withMessage("Please provide a valid email"),
];
// New password validation rules
export const newPasswordValidationRules = [
    body("password")
        .isLength({ min: 6 })
        .withMessage("Password must be at least 6 characters long"),
    body("token").notEmpty().withMessage("Token is required"),
];
// Registration validation
export const validateRegistration = [
    body("email")
        .trim()
        .isEmail()
        .withMessage("Please provide a valid email address")
        .normalizeEmail(),
    body("password")
        .isLength({ min: 6 })
        .withMessage("Password must be at least 6 characters long")
        .matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/)
        .withMessage(
            "Password must contain at least one uppercase letter, one lowercase letter, and one number"
        ),
    body("firstName")
        .trim()
        .notEmpty()
        .withMessage("First name is required")
        .isLength({ min: 2 })
        .withMessage("First name must be at least 2 characters long"),
    body("lastName")
        .trim()
        .notEmpty()
        .withMessage("Last name is required")
        .isLength({ min: 2 })
        .withMessage("Last name must be at least 2 characters long"),
];
```

```
body("phone")
    .optional()
    .trim()
    .matches(/^\+?[1-9]\d{1,14}$/)
    .withMessage("Please provide a valid phone number"),

body("role")
    .optional()
    .isIn(["CUSTOMER", "RESTAURANT", "DELIVERY", "ADMIN"])
    .withMessage("Invalid role specified"),

validateRequest,
];

// Login validation
export const validateLogin = [
    body("email")
        .trim()
        .isEmail()
        .withMessage("Please provide a valid email address")
        .normalizeEmail(),

    body("password").notEmpty().withMessage("Password is required"),

    validateRequest,
];

// Email verification validation
export const validateEmailVerification = [
    body("pin")
        .trim()
        .isLength({ min: 6, max: 6 })
        .withMessage("PIN must be 6 digits")
        .matches(/^\d+$/)
        .withMessage("PIN must contain only numbers"),

    validateRequest,
];

// Password reset request validation
export const validateForgotPassword = [
    body("email")
        .trim()
        .isEmail()
        .withMessage("Please provide a valid email address")
        .normalizeEmail(),

    validateRequest,
];

// Password reset validation
export const validateResetPassword = [
```

```
body("password")
    .isLength({ min: 6 })
    .withMessage("Password must be at least 6 characters long")
    .matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/)
    .withMessage(
        "Password must contain at least one uppercase letter, one lowercase letter, and one number"
    ),

body("confirmPassword").custom((value, { req }) => {
    if (value !== req.body.password) {
        throw new Error("Passwords do not match");
    }
    return true;
}),

validateRequest,
];

// Update profile validation
export const validateProfileUpdate = [
    body("firstName")
        .optional()
        .trim()
        .isLength({ min: 2 })
        .withMessage("First name must be at least 2 characters long"),

    body("lastName")
        .optional()
        .trim()
        .isLength({ min: 2 })
        .withMessage("Last name must be at least 2 characters long"),

    body("phone")
        .optional()
        .trim()
        .matches(/^\+?[1-9]\d{1,14}$/)
        .withMessage("Please provide a valid phone number"),

    validateRequest,
];

// Change password validation
export const validateChangePassword = [
    body("currentPassword")
        .notEmpty()
        .withMessage("Current password is required"),

    body("newPassword")
        .isLength({ min: 6 })
        .withMessage("New password must be at least 6 characters long")
        .matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/)
        .withMessage(
            "New password must contain at least one uppercase letter, one lowercase letter, and one number"
        )
];
```

```
    ),  
  
    body("confirmNewPassword").custom((value, { req }) => {  
      if (value !== req.body.newPassword) {  
        throw new Error("New passwords do not match");  
      }  
      return true;  
    }),  
  
    validateRequest,  
  ];  
  
// Update user validation (for admin)  
export const validateUpdateUser = [  
  body("email")  
    .optional()  
    .trim()  
    .isEmail()  
    .withMessage("Please provide a valid email address")  
    .normalizeEmail(),  
  body("firstName")  
    .optional()  
    .trim()  
    .isLength({ min: 2 })  
    .withMessage("First name must be at least 2 characters long"),  
  body("lastName")  
    .optional()  
    .trim()  
    .isLength({ min: 2 })  
    .withMessage("Last name must be at least 2 characters long"),  
  body("phone")  
    .optional()  
    .trim()  
    .matches(/^\+?[1-9]\d{1,14}$/)  
    .withMessage("Please provide a valid phone number"),  
  body("role")  
    .optional()  
    . isIn(["CUSTOMER", "RESTAURANT", "DELIVERY", "ADMIN"])  
    .withMessage("Invalid role specified"),  
  body("address.street")  
    .optional()  
    .trim()  
    .notEmpty()  
    .withMessage("Street address is required"),  
  body("address.city")  
    .optional()  
    .trim()  
    .notEmpty()  
    .withMessage("City is required"),  
  body("address.state")  
    .optional()  
    .trim()  
    .notEmpty()
```

```

.withMessage("State is required"),
body("address.zipCode")
.optional()
.trim()
.isNotEmpty()
.withMessage("Zip code is required"),
body("address.country")
.optional()
.trim()
.isNotEmpty()
.withMessage("Country is required"),
body("password")
.optional()
.isLength({ min: 6 })
.withMessage("Password must be at least 6 characters long")
.matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/)
.withMessage(
  "Password must contain at least one uppercase letter, one lowercase letter, and one number"
),
body("confirmPassword")
.optional()
.custom((value, { req }) => {
  if (req.body.password && value !== req.body.password) {
    throw new Error("Passwords do not match");
  }
  return true;
}),
validateRequest,
];

```

```

// Update user status validation (for admin)
export const validateUpdateUserStatus = [
  body("isActive")
    .optional()
    .isBoolean()
    .withMessage("isActive must be a boolean"),
  body("isVerified")
    .optional()
    .isBoolean()
    .withMessage("isVerified must be a boolean"),
  validateRequest,
];

```

```

// Update user role validation (for admin)
export const validateUpdateUserRole = [
  body("role")
    .notEmpty()
    .withMessage("Role is required")
    .isIn(["CUSTOMER", "RESTAURANT", "DELIVERY", "ADMIN"])
    .withMessage("Invalid role specified"),
  validateRequest,
];

```

auth-service/src/index.js

```
// auth-service/src/index.js
import "dotenv/config";
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import morgan from "morgan";
import cookieParser from "cookie-parser";
import logger from "./utils/logger.js";
import authRoutes from "./routes/auth.routes.js";

const app = express();

// Middleware
app.use(
  cors({
    origin: "*",
    methods: ["GET", "POST", "PUT", "DELETE", "PATCH"],
    allowedHeaders: ["Content-Type", "Authorization"],
    credentials: true,
  })
);
app.use(express.json({ limit: "50mb" }));
app.use(express.urlencoded({ extended: true, limit: "50mb" }));
app.use(cookieParser());
app.use(morgan("dev"));

// Request logging middleware
// app.use((req, res, next) => {
//   logger.info(`Incoming request: ${req.method} ${req.url}`, {
//     body: req.body,
//     headers: req.headers,
//   });
//   next();
// });

// Routes
app.use("/api/auth", authRoutes);

// Error handling middleware
app.use((err, req, res, next) => {
  logger.error(err.stack);
  res.status(500).json({ message: "Something went wrong!" });
});

// Database connection
mongoose
  .connect(process.env.MONGODB_URI)
  .then(() => {
    logger.info("Connected to MongoDB", {
      timestamp: new Date().toISOString(),
    });
  });

```

```

app.listen(process.env.PORT, () => {
  logger.info(`Auth service running on port ${process.env.PORT}`, {
    timestamp: new Date().toISOString(),
  });
});
})
.catch((err) => {
  logger.error("MongoDB connection error:", err);
  process.exit(1);
});

```

10.3 cart-service

cart-service/src/controllers/cart.controller.js

```

import { cartService } from "../services/cart.service";
import logger from "../utils/logger.js";

class CartController {
  async getCart(req, res) {
    try {
      const cart = await cartService.getCart(req.params.userId);
      res.json(cart);
    } catch (error) {
      logger.error("Error in getCart controller:", error);
      res.status(400).json({ message: error.message });
    }
  }

  async addToCart(req, res) {
    try {
      const { userId } = req.params;
      logger.info("Received addToCart request:", {
        userId,
        body: req.body,
      });

      const itemData = {
        menuItemId: req.body.menuItemId,
        restaurantId: req.body.restaurantId,
        name: req.body.name,
        price: req.body.price,
        quantity: req.body.quantity,
        mainImage: req.body.mainImage,
        thumbnailImage: req.body.thumbnailImage,
      };
      logger.info("Processed itemData:", itemData);

      const cart = await cartService.addCart(userId, itemData);
      logger.info("Item added to cart successfully", {
        userId,
      });
    }
  }
}

```

```
    menuItemId: itemData.menuItemId,
  });
  res.status(201).json(cart);
} catch (error) {
  logger.error("Error in addToCart controller:", error);
  res.status(400).json({ message: error.message });
}
}

async updateCartItem(req, res) {
try {
  const { userId, menuItemId } = req.params;
  const { quantity } = req.body;

  const cart = await cartService.updateCartItem(
    userId,
    menuItemId,
    quantity
  );
  logger.info("Cart item updated successfully", {
    userId,
    menuItemId,
    quantity,
  });
  res.json(cart);
} catch (error) {
  logger.error("Error in updateCartItem controller:", error);
  res.status(400).json({ message: error.message });
}
}

async removeFromCart(req, res) {
try {
  const { userId, menuItemId } = req.params;
  const cart = await cartService.removeFromCart(userId, menuItemId);
  logger.info("Item removed from cart successfully", {
    userId,
    menuItemId,
  });
  res.json(cart);
} catch (error) {
  logger.error("Error in removeFromCart controller:", error);
  res.status(400).json({ message: error.message });
}
}

async clearCart(req, res) {
try {
  const { userId } = req.params;
  const cart = await cartService.clearCart(userId);
  logger.info("Cart cleared successfully", { userId });
  res.json(cart);
} catch (error) {
```

```

        logger.error("Error in clearCart controller:", error);
        res.status(400).json({ message: error.message });
    }
}
}

export const cartController = new CartController();

```

cart-service/src/middleware/validate.middleware.js

```

import { body, validationResult } from "express-validator";
import logger from "../utils/logger.js";

export const validateAddToCart = [
    body("menuItemId").notEmpty().withMessage("Menu item ID is required"),
    body("restaurantId").notEmpty().withMessage("Restaurant ID is required"),
    body("name").notEmpty().withMessage("Item name is required"),
    body("price")
        .isFloat({ min: 0 })
        .withMessage("Price must be a positive number"),
    body("quantity").isInt({ min: 1 }).withMessage("Quantity must be at least 1"),
    body("mainImage").notEmpty().withMessage("Main image URL is required"),
    body("thumbnailImage")
        .notEmpty()
        .withMessage("Thumbnail image URL is required"),
];

```

```

export const validateUpdateCartItem = [
    body("quantity").isInt({ min: 1 }).withMessage("Quantity must be at least 1"),
];

```

```

export const validateRequest = (req, res, next) => {
    logger.info("Validating request body:", req.body);
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        logger.error("Validation error:", {
            errors: errors.array(),
            body: req.body,
        });
        return res.status(400).json({ errors: errors.array() });
    }
    next();
};

```

cart-service/src/models/cart.model.js

```

import mongoose from "mongoose";

const cartItemSchema = new mongoose.Schema(
{
    menuItemId: {
        type: String,
        required: true,

```

```
},
restaurantId: {
  type: String,
  required: true,
},
name: {
  type: String,
  required: true,
},
price: {
  type: Number,
  required: true,
},
quantity: {
  type: Number,
  required: true,
  min: 1,
},
totalPrice: {
  type: Number,
  required: true,
},
mainImage: {
  type: String,
  required: true,
  default: "",
},
thumbnailImage: {
  type: String,
  required: true,
  default: "",
},
{
  _id: true
});
```

```
const cartSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
    unique: true,
  },
  restaurantId: {
    type: String,
    required: function () {
      return this.items && this.items.length > 0;
    },
  },
  items: [cartItemSchema],
  totalAmount: {
    type: Number,
    required: true,
    default: 0,
```

```

},
createdAt: {
  type: Date,
  default: Date.now,
},
updatedAt: {
  type: Date,
  default: Date.now,
},
});

// Update the updatedAt timestamp before saving
cartSchema.pre("save", function (next) {
  this.updatedAt = Date.now();
  next();
});

// Calculate total amount before saving
cartSchema.pre("save", function (next) {
  this.totalAmount = this.items.reduce(
    (total, item) => total + item.totalPrice,
    0
  );
  next();
});

// Validate image fields before saving
cartSchema.pre("save", function (next) {
  if (this.items && this.items.length > 0) {
    this.items.forEach((item) => {
      if (!item.mainImage) {
        item.mainImage = "";
      }
      if (!item.thumbnailImage) {
        item.thumbnailImage = "";
      }
    });
  }
  next();
});

export const Cart = mongoose.model("Cart", cartSchema);

```

cart-service/src/routes/cart.routes.js

```

import express from "express";
import { cartController } from "../controllers/cart.controller.js";
import {
  validateAddToCart,
  validateUpdateCartItem,
  validateRequest,
} from "../middleware/validate.middleware.js";

```

```

const router = express.Router();

// Routes
router.get("/:userId", cartController.getCart);
router.post(
  "/:userId/items",
  validateAddToCart,
  validateRequest,
  cartController.addToCart
);
router.patch(
  "/:userId/items/:menuItemID",
  validateUpdateCartItem,
  validateRequest,
  cartController.updateCartItem
);
router.delete("/:userId/items/:menuItemID", cartController.removeFromCart);
router.delete("/:userId", cartController.clearCart);

export default router;

```

cart-service/src/services/cart.service.js

```

import { Cart } from "../models/cart.model.js";
import logger from "../utils/logger.js";

class CartService {
  async getCart(userId) {
    try {
      let cart = await Cart.findOne({ userId });
      if (!cart) {
        cart = new Cart({ userId, items: [], restaurantId: null });
        await cart.save();
      }
      return cart;
    } catch (error) {
      logger.error("Error getting cart:", error);
      throw new Error(`Error getting cart: ${error.message}`);
    }
  }

  async addToCart(userId, itemData) {
    try {
      let cart = await Cart.findOne({ userId });

      // If cart doesn't exist, create a new one
      if (!cart) {
        cart = new Cart({
          userId,
          restaurantId: itemData.restaurantId,
          items: [],
        });
        await cart.save();
      }

      const existingItemIndex = cart.items.findIndex(
        (item) => item.menuItemID === itemData.menuItemID
      );
      if (existingItemIndex === -1) {
        cart.items.push(itemData);
      } else {
        cart.items[existingItemIndex].quantity += itemData.quantity;
      }

      await cart.save();
    } catch (error) {
      logger.error("Error adding item to cart:", error);
      throw new Error(`Error adding item to cart: ${error.message}`);
    }
  }
}

```

```

    totalAmount: 0,
  });
}

// Check if item is from the same restaurant
if (
  cart.items.length > 0 &&
  cart.restaurantId !== itemData.restaurantId
) {
  throw new Error(
    "Cannot add items from different restaurants to the same cart"
  );
}

// Set restaurant ID if cart is empty
if (cart.items.length === 0) {
  cart.restaurantId = itemData.restaurantId;
}

// Calculate total price for the item
const totalPrice = itemData.price * itemData.quantity;

// Check if item already exists in cart
const existingItemIndex = cart.items.findIndex(
  (item) => item.menuItemId === itemData.menuItemId
);

if (existingItemIndex > -1) {
  // Update existing item
  const existingItem = cart.items[existingItemIndex];
  existingItem.quantity += itemData.quantity;
  existingItem.totalPrice += totalPrice;
  // Preserve image fields
  existingItem.mainImage = itemData.mainImage || existingItem.mainImage;
  existingItem.thumbnailImage =
    itemData.thumbnailImage || existingItem.thumbnailImage;
} else {
  // Create new cart item with all required fields
  const newItem = {
    menuItemId: itemData.menuItemId,
    restaurantId: itemData.restaurantId,
    name: itemData.name,
    price: itemData.price,
    quantity: itemData.quantity,
    totalPrice: totalPrice,
    mainImage: itemData.mainImage || "",
    thumbnailImage: itemData.thumbnailImage || "",
  };
}

// Add new item to cart
cart.items.push(newItem);
}

```

```

// Calculate total amount
cart.totalAmount = cart.items.reduce(
  (total, item) => total + item.totalPrice,
  0
);

// Save the cart
const savedCart = await cart.save();
return savedCart;
} catch (error) {
  logger.error("Error adding to cart:", error);
  throw new Error(`Error adding to cart: ${error.message}`);
}

async updateCartItem(userId, menuItemId, quantity) {
try {
  const cart = await Cart.findOne({ userId });
  if (!cart) {
    throw new Error("Cart not found");
  }

  const itemIndex = cart.items.findIndex(
    (item) => item.menuItemId === menuItemId
  );
  if (itemIndex === -1) {
    throw new Error("Item not found in cart");
  }

  // Update quantity and total price while preserving other fields
  const item = cart.items[itemIndex];
  item.quantity = quantity;
  item.totalPrice = item.price * quantity;

  await cart.save();
  return cart;
} catch (error) {
  logger.error("Error updating cart item:", error);
  throw new Error(`Error updating cart item: ${error.message}`);
}
}

async removeFromCart(userId, menuItemId) {
try {
  const cart = await Cart.findOne({ userId });
  if (!cart) {
    throw new Error("Cart not found");
  }

  cart.items = cart.items.filter((item) => item.menuItemId !== menuItemId);

  // Reset restaurant ID if cart becomes empty
  if (cart.items.length === 0) {

```

```

        cart.restaurantId = null;
    }

    await cart.save();
    return cart;
} catch (error) {
    logger.error("Error removing from cart:", error);
    throw new Error(`Error removing from cart: ${error.message}`);
}

async clearCart(userId) {
    try {
        const cart = await Cart.findOne({ userId });
        if (!cart) {
            throw new Error("Cart not found");
        }

        cart.items = [];
        cart.restaurantId = null;
        cart.totalAmount = 0;

        await cart.save();
        return cart;
    } catch (error) {
        logger.error("Error clearing cart:", error);
        throw new Error(`Error clearing cart: ${error.message}`);
    }
}

export const cartService = new CartService();

```

cart-service/src/utils/logger.js

```

import winston from "winston";

const logger = winston.createLogger({
    level: "info",
    format: winston.format.combine(
        winston.format.timestamp(),
        winston.format.json()
    ),
    transports: [
        new winston.transports.File({ filename: "error.log", level: "error" }),
        new winston.transports.File({ filename: "combined.log" }),
    ],
});

if (process.env.NODE_ENV !== "production") {
    logger.add(
        new winston.transports.Console({

```

```
        format: winston.format.simple(),
    })
);
}

export default logger;
```

cart-service/src/index.js

```
import "dotenv/config";
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import morgan from "morgan";
import cookieParser from "cookie-parser";
import logger from "./utils/logger.js";
import cartRoutes from "./routes/cart.routes.js";

const app = express();

// Middleware
app.use(
  cors({
    origin: "*",
    methods: ["GET", "POST", "PUT", "DELETE", "PATCH"],
    allowedHeaders: ["Content-Type", "Authorization"],
    credentials: true,
  })
);
app.use(express.json({ limit: "50mb" }));
app.use(express.urlencoded({ extended: true, limit: "50mb" }));
app.use(cookieParser());
app.use(morgan("dev"));

// Routes
app.use("/api/carts", cartRoutes);

// Error handling middleware
app.use((err, req, res, next) => {
  logger.error(err.stack);
  res.status(500).json({ message: "Something went wrong!" });
});

// Database connection
mongoose
  .connect(process.env.MONGODB_URI || "mongodb://localhost:27017/cart-service")
  .then(() => {
    logger.info("Connected to MongoDB", {
      timestamp: new Date().toISOString(),
    });
    app.listen(process.env.PORT || 3004, () => {
      logger.info(`Cart service running on port ${process.env.PORT || 3004}`, {
        timestamp: new Date().toISOString(),
      });
    });
  })
);
```

```

        timestamp: new Date().toISOString(),
    });
});
})
.catch((err) => {
    logger.error("MongoDB connection error:", err);
    process.exit(1);
});

```

10.4 delivery-service

delivery-service/src/controllers/delivery.controller.js

```

import { Delivery } from "../models/delivery.model.js";
import { logger } from "../utils/logger.js";
import axios from "axios";
import {
    getAvailableDrivers,
    updateDriverAvailability,
    completeDelivery,
} from "../services/driver.service.js";

// Service URLs
const DRIVER_SERVICE_URL =
    process.env.DRIVER_SERVICE_URL || "http://localhost:3010";

export const assignDeliveryDriver = async (req, res, next) => {
    try {
        const { orderId, customerLocation } = req.body;

        // Check if order already has a delivery
        const existingDelivery = await Delivery.findOne({ orderId });
        if (existingDelivery) {
            return res.status(400).json({
                status: "error",
                message: "Order already has a delivery assigned",
            });
        }

        // Get available drivers from driver service
        const availableDrivers = await getAvailableDrivers(
            customerLocation[1], // latitude
            customerLocation[0], // longitude
            5000 // max distance in meters
        );
    }
}

```

```

if (!availableDrivers || availableDrivers.length === 0) {
  return res.status(404).json({
    status: "error",
    message: "No available drivers found in the area",
  });
}

// Assign the first available driver
const assignedDriver = availableDrivers[0];

// Create delivery record
const delivery = new Delivery({
  orderId,
  driverId: assignedDriver._id,
  status: "ASSIGNED",
  customerLocation: {
    type: "Point",
    coordinates: customerLocation,
  },
  driverLocation: assignedDriver.location,
});

await delivery.save();

// Update driver availability and assign delivery in driver service
try {
  await updateDriverAvailability(assignedDriver._id, false, delivery._id);
} catch (error) {
  logger.error("Error updating driver status:", error);
  // If driver update fails, delete the delivery to maintain consistency
  await Delivery.findByIdAndDelete(delivery._id);
  throw error;
}

// Schedule status update to PICKED_UP after 5 seconds
setTimeout(async () => {
  try {
    delivery.status = "PICKED_UP";
    await delivery.save();

    // Emit delivery picked up event if Socket.IO is available
    const io = req.app.get("io");
    if (io) {
      io.emit("delivery-picked-up", {
        deliveryId: delivery._id,
        driverId: assignedDriver._id,
        orderId: orderId,
      });
    }
  }

  logger.info(`Delivery ${delivery._id} status updated to PICKED_UP`);
} catch (error) {
  logger.error("Error updating delivery status to PICKED_UP:", error);
}

```

```

    },
    }, 5000);

// Emit delivery assigned event if Socket.IO is available
const io = req.app.get("io");
if (io) {
  io.emit("delivery-assigned", {
    deliveryId: delivery._id,
    driverId: assignedDriver._id,
    orderId: orderId,
  });
}

res.status(201).json({
  status: "success",
  data: delivery,
});
} catch (error) {
  logger.error("Error assigning delivery driver:", error);
  next(error);
}
};

export const updateDeliveryStatus = async (req, res, next) => {
  try {
    const { deliveryId } = req.params;
    const { status, location } = req.body;

    const delivery = await Delivery.findById(deliveryId);
    if (!delivery) {
      return res.status(404).json({
        status: "error",
        message: "Delivery not found",
      });
    }

    // Update status
    delivery.status = status;

    // Update location if provided
    if (location) {
      delivery.driverLocation = {
        type: "Point",
        coordinates: location,
      };
    }

    // Update delivery completion time if delivered
    if (status === "DELIVERED") {
      delivery.actualDeliveryTime = new Date();
    }

    await delivery.save();
  }
}

```

```
res.json({
  status: "success",
  data: delivery,
});
} catch (error) {
  logger.error("Error updating delivery status:", error);
  next(error);
}
};

export const getDeliveryStatus = async (req, res, next) => {
  try {
    const { deliveryId } = req.params;
    const delivery = await Delivery.findById(deliveryId);

    if (!delivery) {
      return res.status(404).json({
        status: "error",
        message: "Delivery not found",
      });
    }
    res.json({
      status: "success",
      data: delivery,
    });
  } catch (error) {
    next(error);
  }
};

export const getDriverLocation = async (req, res, next) => {
  try {
    const { deliveryId } = req.params;
    const delivery = await Delivery.findById(deliveryId);

    if (!delivery) {
      return res.status(404).json({
        status: "error",
        message: "Delivery not found",
      });
    }
    res.json({
      status: "success",
      data: {
        location: delivery.driverLocation,
        lastUpdated: delivery.updatedAt,
      },
    });
  } catch (error) {
    next(error);
  }
};
```

delivery-service/src/middleware/error.middleware.js

```
import { logger } from "../utils/logger.js";

export const errorHandler = (err, req, res, next) => {
  logger.error(err.stack);

  const statusCode = err.statusCode || 500;
  const message = err.message || "Internal Server Error";

  res.status(statusCode).json({
    status: "error",
    statusCode,
    message,
  });
};
```

delivery-service/src/models/delivery.model.js

```
import mongoose from "mongoose";

const deliverySchema = new mongoose.Schema({
  orderId: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: "Order",
  },
  driverId: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: "Driver",
  },
  status: {
    type: String,
    enum: ["ASSIGNED", "PICKED_UP", "IN_TRANSIT", "DELIVERED", "CANCELLED"],
    default: "ASSIGNED",
  },
  customerLocation: {
    type: {
      type: String,
      enum: ["Point"],
      required: true,
    },
    coordinates: {
      type: [Number],
      required: true,
    },
  },
  driverLocation: {
    type: {
      type: String,
      enum: ["Point"],
      required: true,
    },
  },
});
```

```

},
coordinates: {
  type: [Number],
  required: true,
},
},
estimatedDeliveryTime: Date,
actualDeliveryTime: Date,
createdAt: {
  type: Date,
  default: Date.now,
},
updatedAt: {
  type: Date,
  default: Date.now,
},
});
// Create index for geospatial queries
deliverySchema.index({ customerLocation: "2dsphere" });
deliverySchema.index({ driverLocation: "2dsphere" });

// Update the updatedAt field before saving
deliverySchema.pre("save", function (next) {
  this.updatedAt = new Date();
  next();
});

export const Delivery = mongoose.model("Delivery", deliverySchema);

```

delivery-service/src/routes/delivery.routes.js

```

import express from "express";
import {
  assignDeliveryDriver,
  updateDeliveryStatus,
  getDeliveryStatus,
  getDriverLocation,
} from "../controllers/delivery.controller.js";

const router = express.Router();

// Assign a driver to a delivery
router.post("/assign", assignDeliveryDriver);

// Update delivery status
router.put("/:deliveryId/status", updateDeliveryStatus);

// Get delivery status
router.get("/:deliveryId/status", getDeliveryStatus);

// Get driver's current location for a delivery

```

```
router.get("/:deliveryId/location", getDriverLocation);

export default router;
```

delivery-service/src/services/delivery.service.js

```
import axios from "axios";
import { logger } from "../utils/logger.js";

const DRIVER_SERVICE_URL =
  process.env.DRIVER_SERVICE_URL || "http://localhost:3008";

export const getAvailableDrivers = async (
  latitude,
  longitude,
  maxDistance = 5000
) => {
  try {
    const response = await axios.get(
      `${DRIVER_SERVICE_URL}/api/drivers/available`,
      {
        params: {
          latitude,
          longitude,
          maxDistance,
        },
      }
    );
    return response.data.data;
  } catch (error) {
    logger.error("Error fetching available drivers:", error.message);
    throw error;
  }
};

export const updateDriverAvailability = async (
  driverId,
  isAvailable,
  deliveryId
) => {
  console.log("deliveryId", deliveryId);
  try {
    const response = await axios.put(
      `${DRIVER_SERVICE_URL}/api/drivers/${driverId}/availability`,
      {
        isAvailable,
        deliveryId,
      }
    );
    return response.data.data;
  } catch (error) {
    logger.error("Error updating driver availability:", error.message);
  }
};
```

```

        throw error;
    }
};

export const completeDelivery = async (driverId) => {
    try {
        const response = await axios.put(
            `${DRIVER_SERVICE_URL}/api/drivers/${driverId}/complete`
        );
        return response.data.data;
    } catch (error) {
        logger.error("Error completing delivery:", error.message);
        throw error;
    }
};

```

delivery-service/src/socket/delivery.socket.js

```

import { Delivery } from "../models/delivery.model.js";
import { logger } from "../utils/logger.js";

export const setupSocketHandlers = (io) => {
    io.on("connection", (socket) => {
        logger.info("New client connected");

        // Join delivery room for real-time updates
        socket.on("join-delivery", async (deliveryId) => {
            try {
                const delivery = await Delivery.findById(deliveryId);
                if (!delivery) {
                    socket.emit("error", { message: "Delivery not found" });
                    return;
                }

                socket.join(`delivery:${deliveryId}`);
                logger.info(`Client joined delivery room: ${deliveryId}`);
            } catch (error) {
                logger.error("Error joining delivery room:", error);
                socket.emit("error", { message: "Internal server error" });
            }
        });

        // Handle location updates from drivers
        socket.on("update-location", async (data) => {
            try {
                const { deliveryId, location } = data;
                const delivery = await Delivery.findById(deliveryId);

                if (!delivery) {
                    socket.emit("error", { message: "Delivery not found" });
                    return;
                }
            }
        });
    });
};

```

```
delivery.driverLocation = location;
await delivery.save();

// Broadcast location update to all clients in the delivery room
io.to(`delivery:${deliveryId}`).emit("location-updated", {
  deliveryId,
  location,
  timestamp: new Date(),
});
} catch (error) {
  logger.error("Error updating location:", error);
  socket.emit("error", { message: "Internal server error" });
}
});

// Handle delivery status updates
socket.on("update-status", async (data) => {
  try {
    const { deliveryId, status } = data;
    const delivery = await Delivery.findById(deliveryId);

    if (!delivery) {
      socket.emit("error", { message: "Delivery not found" });
      return;
    }

    delivery.status = status;
    await delivery.save();

    // Broadcast status update to all clients in the delivery room
    io.to(`delivery:${deliveryId}`).emit("status-updated", {
      deliveryId,
      status,
      timestamp: new Date(),
    });
  } catch (error) {
    logger.error("Error updating status:", error);
    socket.emit("error", { message: "Internal server error" });
  }
});

socket.on("disconnect", () => {
  logger.info("Client disconnected");
});
```

delivery-service/src/utils/logger.js

```
import winston from "winston";

const logger = winston.createLogger({
  level: "info",
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.Console({
      format: winston.format.combine(
        winston.format.colorize(),
        winston.format.simple()
      ),
    }),
    new winston.transports.File({ filename: "error.log", level: "error" }),
    new winston.transports.File({ filename: "combined.log" }),
  ],
});

export { logger };
```

delivery-service/src/index.js

```
import express from "express";
import cors from "cors";
import mongoose from "mongoose";
import dotenv from "dotenv";
import morgan from "morgan";
import deliveryRoutes from "./routes/delivery.routes";
import { errorHandler } from "./middleware/error.middleware.js";
import { logger } from "./utils/logger.js";

dotenv.config();

const app = express();

// Middleware
app.use(cors());
app.use(express.json());
app.use(morgan("dev"));

// Routes
app.use("/api/delivery", deliveryRoutes);

// Error handling
app.use(errorHandler);

// MongoDB connection
mongoose
```

```

.connect(
  process.env.MONGODB_URI || "mongodb://localhost:27017/delivery-service"
)
.then(() => logger.info("Connected to MongoDB"))
.catch((err) => {
  logger.error("MongoDB connection error:", err);
  process.exit(1);
});

const PORT = process.env.PORT || 3007;
app.listen(PORT, () => {
  logger.info(`Delivery service running on port ${PORT}`);
});

```

10.5 driver-service

driver-service/src/controllers/driver.controller.js

```

import { Driver } from "../models/driver.model.js";
import { logger } from "../utils/logger.js";
import { updateDeliveryStatus } from "../services/delivery.service.js";

export const registerDriver = async (req, res, next) => {
  try {
    const { userId, vehicleType, vehicleNumber, location } = req.body;

    // Validate required fields
    if (!userId || !vehicleType || !vehicleNumber || !location) {
      return res.status(400).json({
        status: "error",
        message: "Missing required fields",
      });
    }

    // Check if driver profile already exists for this userId
    const existingDriver = await Driver.findOne({ userId });
    if (existingDriver) {
      return res.status(400).json({
        status: "error",
        message: "Driver profile already exists for this user",
      });
    }

    // Create new driver
    const driver = new Driver({
      userId,
      vehicleType,
      vehicleNumber,
      location: {
        type: "Point",
        coordinates: location,
      }
    });
    await driver.save();
    logger.info(`New driver registered: ${driver._id}`);
    res.status(201).json(driver);
  } catch (err) {
    logger.error(`Error registering driver: ${err}`);
    res.status(500).json({ status: "error", message: "Internal server error" });
  }
};

```

```
        },
        isAvailable: true,
    });

await driver.save();

res.status(201).json({
    status: "success",
    data: driver,
});

} catch (error) {
    logger.error("Error registering driver:", error);
    next(error);
}

};

export const updateDriverLocation = async (req, res, next) => {
    try {
        const { driverId } = req.params;
        const { location } = req.body;

        const driver = await Driver.findById(driverId);
        if (!driver) {
            return res.status(404).json({
                status: "error",
                message: "Driver not found",
            });
        }

        driver.location = {
            type: "Point",
            coordinates: location,
        };

        await driver.save();

        res.json({
            status: "success",
            data: driver,
        });

    } catch (error) {
        next(error);
    }
};

export const updateDriverAvailability = async (req, res, next) => {
    try {
        const { driverId } = req.params;
        const { isAvailable, deliveryId } = req.body;

        const driver = await Driver.findById(driverId);
        if (!driver) {
            return res.status(404).json({
```

```

    status: "error",
    message: "Driver not found",
  });
}

// If driver has an active delivery, prevent changing availability
if (driver.currentDelivery) {
  return res.status(400).json({
    status: "error",
    message:
      "Cannot change availability while having an active delivery. Please complete or cancel the delivery first.",
  });
}

// If setting to unavailable and providing a deliveryId
if (!isAvailable && deliveryId) {
  driver.currentDelivery = deliveryId;
  driver.isAvailable = false; // Force isAvailable to false when assigned a delivery
}

// If setting to available, clear currentDelivery
if (isAvailable) {
  driver.currentDelivery = null;
  driver.isAvailable = true;
}

await driver.save();

res.json({
  status: "success",
  message: isAvailable
    ? "You are now online and available for deliveries"
    : "You are now offline",
  data: driver,
});
} catch (error) {
  logger.error("Error updating driver availability:", error);
  next(error);
}
};

// Add middleware to check driver availability
export const checkDriverAvailability = async (req, res, next) => {
  try {
    const { driverId } = req.params;
    const driver = await Driver.findById(driverId);

    if (!driver) {
      return res.status(404).json({
        status: "error",
        message: "Driver not found",
      });
    }
  }
};

```

```

// If driver has a current delivery, force isAvailable to false
if (driver.currentDelivery && driver.isAvailable) {
  driver.isAvailable = false;
  await driver.save();
}

next();
} catch (error) {
  logger.error("Error checking driver availability:", error);
  next(error);
}
};

export const getAvailableDrivers = async (req, res, next) => {
  try {
    const { latitude, longitude, maxDistance = 5000 } = req.query;

    let query = { isAvailable: true };

    if (latitude && longitude) {
      query.location = {
        $near: {
          $geometry: {
            type: "Point",
            coordinates: [parseFloat(longitude), parseFloat(latitude)],
          },
          $maxDistance: parseInt(maxDistance),
        },
      };
    }
  }

  const drivers = await Driver.find(query);

  res.json({
    status: "success",
    data: drivers,
  });
} catch (error) {
  next(error);
}
};

export const getDriverDetails = async (req, res, next) => {
  try {
    const { driverId } = req.params;
    const driver = await Driver.findById(driverId);

    if (!driver) {
      return res.status(404).json({
        status: "error",
        message: "Driver not found",
      });
    }
  }
}

```

```

}

res.json({
  status: "success",
  data: driver,
});
} catch (error) {
  next(error);
}
};

export const assignDelivery = async (req, res, next) => {
  try {
    const { driverId } = req.params;
    const { deliveryId } = req.body;

    const driver = await Driver.findById(driverId);
    if (!driver) {
      return res.status(404).json({
        status: "error",
        message: "Driver not found",
      });
    }

    // Check if driver is already assigned to a delivery
    if (driver.currentDelivery) {
      return res.status(400).json({
        status: "error",
        message: "Driver is already assigned to a delivery",
      });
    }

    // Check if driver is available
    if (!driver.isAvailable) {
      return res.status(400).json({
        status: "error",
        message: "Driver is not available",
      });
    }

    // Update delivery status to PICKED_UP
    await updateDeliveryStatus(
      deliveryId,
      "PICKED_UP",
      driver.location.coordinates
    );

    // Update driver status
    driver.currentDelivery = deliveryId;
    driver.isAvailable = false;
    await driver.save();

    res.json({

```

```

    status: "success",
    data: driver,
  });
} catch (error) {
  logger.error("Error assigning delivery:", error);
  next(error);
}
};

export const completeDelivery = async (req, res, next) => {
  try {
    const { driverId } = req.params;

    const driver = await Driver.findById(driverId);
    if (!driver) {
      return res.status(404).json({
        status: "error",
        message: "Driver not found",
      });
    }

    // Check if driver has an active delivery
    if (!driver.currentDelivery) {
      return res.status(400).json({
        status: "error",
        message: "Driver has no active delivery",
      });
    }

    // Update delivery status in delivery service
    await updateDeliveryStatus(
      driver.currentDelivery,
      "DELIVERED",
      driver.location.coordinates
    );

    // Update driver status
    driver.isAvailable = true;
    driver.currentDelivery = null;
    driver.totalDeliveries += 1;
    await driver.save();

    res.json({
      status: "success",
      data: driver,
    });
  } catch (error) {
    logger.error("Error completing delivery:", error);
    next(error);
  }
};

export const getCurrentDriver = async (req, res, next) => {

```

```

try {
  // Get userId from query parameters
  const { userId } = req.query;

  if (!userId) {
    return res.status(400).json({
      status: "error",
      message: "User ID is required",
    });
  }

  // Find driver by userId
  const driver = await Driver.findOne({ userId });

  if (!driver) {
    return res.status(404).json({
      status: "error",
      message: "Driver profile not found",
    });
  }

  res.json({
    status: "success",
    data: {
      _id: driver._id,
      userId: driver.userId,
      vehicleType: driver.vehicleType,
      vehicleNumber: driver.vehicleNumber,
      location: driver.location,
      isAvailable: driver.isAvailable,
      currentDelivery: driver.currentDelivery,
      totalDeliveries: driver.totalDeliveries,
      createdAt: driver.createdAt,
      updatedAt: driver.updatedAt,
    },
  });
} catch (error) {
  logger.error("Error getting current driver:", error);
  next(error);
}
};

```

driver-service/src/middleware/error.middleware.js

```

import { logger } from "../utils/logger.js";

export const errorHandler = (err, req, res, next) => {
  logger.error(err.stack);

  const statusCode = err.statusCode || 500;
  const message = err.message || "Internal Server Error";

```

```
res.status(statusCode).json({  
  status: "error",  
  statusCode,  
  message,  
});  
};
```

driver-service/src/models/driver.model.js

```
import mongoose from "mongoose";  
  
const driverSchema = new mongoose.Schema({  
  userId: {  
    type: mongoose.Schema.Types.ObjectId,  
    required: true,  
    ref: "User",  
  },  
  isAvailable: {  
    type: Boolean,  
    default: true,  
  },  
  currentDelivery: {  
    type: mongoose.Schema.Types.ObjectId,  
    ref: "Delivery",  
    default: null,  
  },  
  location: {  
    type: {  
      type: String,  
      enum: ["Point"],  
      required: true,  
    },  
    coordinates: {  
      type: [Number],  
      required: true,  
    },  
  },  
  vehicleType: {  
    type: String,  
    enum: ["BIKE", "SCOOTER", "CAR"],  
    required: true,  
  },  
  vehicleNumber: {  
    type: String,  
    required: true,  
  },  
  rating: {  
    type: Number,  
    min: 0,  
    max: 5,  
    default: 0,  
  },
```

```

    },
  totalDeliveries: {
    type: Number,
    default: 0,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
  updatedAt: {
    type: Date,
    default: Date.now,
  },
};

// Create index for geospatial queries
driverSchema.index({ location: "2dsphere" });

// Update the updatedAt field before saving
driverSchema.pre("save", function (next) {
  this.updatedAt = new Date();
  next();
});

export const Driver = mongoose.model("Driver", driverSchema);

```

driver-service/src/routes/driver.routes.js

```

import express from "express";
import {
  registerDriver,
  updateDriverLocation,
  updateDriverAvailability,
  getAvailableDrivers,
  getDriverDetails,
  assignDelivery,
  completeDelivery,
  getCurrentDriver,
} from "../controllers/driver.controller.js";

const router = express.Router();

// Register a new driver
router.post("/register", registerDriver);

// Get current driver details
router.get("/me", getCurrentDriver);

// Update driver's current location
router.put("/:driverId/location", updateDriverLocation);

// Update driver's availability status

```

```

router.put("/:driverId/availability", updateDriverAvailability);

// Get available drivers (with optional location filter)
router.get("/available", getAvailableDrivers);

// Get driver details
router.get("/:driverId", getDriverDetails);

// Assign delivery to driver
router.post("/:driverId/assign", assignDelivery);

// Complete delivery
router.post("/:driverId/complete", completeDelivery);

export default router;

```

driver-service/src/services/delivery.service.js

```

import axios from "axios";
import { logger } from "../utils/logger.js";

const DELIVERY_SERVICE_URL =
  process.env.DELIVERY_SERVICE_URL || "http://localhost:3010";
export const updateDeliveryStatus = async (deliveryId, status, location) => {
  try {
    const response = await axios.put(
      `${DELIVERY_SERVICE_URL}/api/delivery/${deliveryId}/status`,
      {
        status,
        location,
      }
    );
    logger.info(`Delivery status updated: ${deliveryId} - ${status}`);
    return response.data;
  } catch (error) {
    logger.error("Error updating delivery status:", error.message);
    if (error.response) {
      // The request was made and the server responded with a status code
      // that falls out of the range of 2xx
      logger.error("Response data:", error.response.data);
      logger.error("Response status:", error.response.status);
      logger.error("Response headers:", error.response.headers);
    } else if (error.request) {
      // The request was made but no response was received
      logger.error("No response received:", error.request);
    } else {
      // Something happened in setting up the request that triggered an Error
      logger.error("Error setting up request:", error.message);
    }
    throw error;
  }
};

```

driver-service/src/utils/logger.js

```
import winston from "winston";

const logger = winston.createLogger({
  level: "info",
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.Console({
      format: winston.format.combine(
        winston.format.colorize(),
        winston.format.simple()
      ),
    }),
    new winston.transports.File({ filename: "error.log", level: "error" }),
    new winston.transports.File({ filename: "combined.log" }),
  ],
});

export { logger };
```

driver-service/src/index.js

```
import express from "express";
import cors from "cors";
import mongoose from "mongoose";
import dotenv from "dotenv";
import morgan from "morgan";
import driverRoutes from "./routes/driver.routes.js";
import { errorHandler } from "./middleware/error.middleware.js";
import { logger } from "./utils/logger.js";

dotenv.config();

const app = express();

// Middleware
app.use(cors());
app.use(express.json());
app.use(morgan("dev"));

// Routes
app.use("/api/drivers", driverRoutes);

// Error handling
app.use(errorHandler);
```

```
// MongoDB connection
mongoose
  .connect(
    process.env.MONGODB_URI || "mongodb://localhost:27017/driver-service"
  )
  .then(() => logger.info("Connected to MongoDB"))
  .catch((err) => {
    logger.error("MongoDB connection error:", err);
    process.exit(1);
  });

const PORT = process.env.PORT || 3008;
app.listen(PORT, () => {
  logger.info(`Driver service running on port ${PORT}`);
});
```

10.6 notification-service

notification-service/src/controllers/email.controller.js

```
import emailService from "../services/email.service.js";
import logger from "../utils/logger.js";

export class EmailController {
  // Send rejection email for a restaurant
  async sendRejectionEmail(req, res) {
    try {
      const { email } = req.body;
      if (!email) {
        return res.status(400).json({ message: 'Email is required' });
      }
      logger.info('Sending rejection email', { email });
      await emailService.sendRejectionEmail(email);
      res.status(200).json({ message: 'Rejection email sent successfully' });
    } catch (error) {
      logger.error('Rejection email error', { error: error.message, stack: error.stack });
      res.status(500).json({ message: 'Error sending rejection email' });
    }
  }

  // Send approval email for a restaurant
  async sendApprovedEmail(req, res) {
    try {
      const { email } = req.body;
      if (!email) {
        return res.status(400).json({ message: 'Email is required' });
      }
      logger.info('Sending approval email', { email });
      await emailService.sendApprovedEmail(email);
      res.status(200).json({ message: 'Approval email sent successfully' });
    } catch (error) {
      logger.error('Approval email error', { error: error.message, stack: error.stack });
      res.status(500).json({ message: 'Error sending approval email' });
    }
  }

  // Send blocked email for a restaurant
  async sendBlockedEmail(req, res) {
    try {
      const { email } = req.body;
      if (!email) {
        return res.status(400).json({ message: 'Email is required' });
      }
      logger.info('Sending approval email', { email });
      await emailService.sendBlockedEmail(email);
      res.status(200).json({ message: 'Blocked email sent successfully' });
    } catch (error) {
      logger.error('Blocked email error', { error: error.message, stack: error.stack });
      res.status(500).json({ message: 'Error sending approval email' });
    }
  }
}
```

```
        }
    }

async sendVerificationEmail(req, res) {
    try {
        const { email, pin } = req.body;
        console.log("Sending verification email to:", email);

        await emailService.sendVerificationEmail(email, pin);
        res.status(200).json({ message: "Verification email sent successfully" });
    } catch (error) {
        console.error("Verification email error:", error.message);
        res.status(500).json({ message: "Error sending verification email" });
    }
}

async sendPasswordResetEmail(req, res) {
    try {
        const { email, token } = req.body;
        console.log("Sending password reset email to:", email);

        await emailService.sendPasswordResetEmail(email, token);
        res
            .status(200)
            .json({ message: "Password reset email sent successfully" });
    } catch (error) {
        console.error("Password reset email error:", error.message);
        res.status(500).json({ message: "Error sending password reset email" });
    }
}

async sendOrderConfirmationEmail(req, res) {
    try {
        const { email, orderDetails } = req.body;
        console.log("Sending order confirmation email to:", email);

        await emailService.sendOrderConfirmation(email, orderDetails);
        res
            .status(200)
            .json({ message: "Order confirmation email sent successfully" });
    } catch (error) {
        console.error("Order confirmation email error:", error.message);
        res
            .status(500)
            .json({ message: "Error sending order confirmation email" });
    }
}

async sendOrderStatusUpdateEmail(req, res) {
    try {
        const { email, deliveryDetails } = req.body;
        console.log("Sending order status update email to:", email);
```

```

await emailService.sendDeliveryUpdate(email, deliveryDetails);
res
  .status(200)
  .json({ message: "Order status update email sent successfully" });
} catch (error) {
  console.error("Order status update email error:", error.message);
  res
    .status(500)
    .json({ message: "Error sending order status update email" });
}
}

// send payment confirmation email
async sendPaymentConfirmationEmail(req, res) {
  try {
    const { email, paymentDetails } = req.body;
    console.log("Sending payment confirmation email to:", email);
    await emailService.sendPaymentConfirmationEmail(email, paymentDetails);
    res.status(200).json({ message: "Payment confirmation email sent successfully" });
  } catch (error) {
    console.error("Payment confirmation email error:", error.message);
    res.status(500).json({ message: "Error sending payment confirmation email" });
  }
}
}

```

notification-service/src/controllers/notification.controller.js

```

import Notification from "../models/notification.model.js";
import logger from "../utils/logger.js";

export class NotificationController {
  async create(req, res) {
    try {
      const { userId, type, message, data } = req.body;

      const notification = new Notification({
        userId,
        type,
        message,
        data,
        isRead: false,
      });

      await notification.save();
      res.status(201).json(notification);
    } catch (error) {
      logger.error("Create notification error:", error);
      res.status(500).json({ message: "Error creating notification" });
    }
  }
}

```

```
async getAll(req, res) {
  try {
    const { userId } = req.params;
    const notifications = await Notification.find({ userId }).sort({
      createdAt: -1,
    });

    res.json(notifications);
  } catch (error) {
    logger.error("Get notifications error:", error);
    res.status(500).json({ message: "Error fetching notifications" });
  }
}

async markAsRead(req, res) {
  try {
    const { notificationId } = req.params;
    const notification = await Notification.findByIdAndUpdate(
      notificationId,
      { isRead: true },
      { new: true }
    );

    if (!notification) {
      return res.status(404).json({ message: "Notification not found" });
    }

    res.json(notification);
  } catch (error) {
    logger.error("Mark notification as read error:", error);
    res.status(500).json({ message: "Error updating notification" });
  }
}

async delete(req, res) {
  try {
    const { notificationId } = req.params;
    const notification = await Notification.findByIdAndDelete(notificationId);

    if (!notification) {
      return res.status(404).json({ message: "Notification not found" });
    }

    res.json({ message: "Notification deleted successfully" });
  } catch (error) {
    logger.error("Delete notification error:", error);
    res.status(500).json({ message: "Error deleting notification" });
  }
}
```

notification-service/src/models/notification.model.js

```
import mongoose from "mongoose";

const notificationSchema = new mongoose.Schema(
{
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  type: {
    type: String,
    required: true,
    enum: ["order", "system", "promotion"],
  },
  message: {
    type: String,
    required: true,
  },
  data: {
    type: mongoose.Schema.Types.Mixed,
    default: {},
  },
  isRead: {
    type: Boolean,
    default: false,
  },
},
{
  timestamps: true,
}
);

const Notification = mongoose.model("Notification", notificationSchema);

export default Notification;
```

notification-service/src/routes/email.routes.js

```
import express from "express";
import { EmailController } from "../controllers/email.controller.js";

const router = express.Router();
const emailController = new EmailController();

// Send rejection email for a restaurant
router.post('/reject-restaurant', emailController.sendRejectionEmail);

// Send approval email for a restaurant
router.post('/approve-restaurant', emailController.sendApprovedEmail);
```

```

// Send blocked email for a restaurant
router.post('/block-restaurant', emailController.sendBlockedEmail);

// Send verification email
router.post("/verify", emailController.sendVerificationEmail);

// Send password reset email
router.post("/reset-password", emailController.sendPasswordResetEmail);

// Send order confirmation email
router.post("/order-confirmation", emailController.sendOrderConfirmationEmail);

// Send order status update email
router.post("/order-status", emailController.sendOrderStatusUpdateEmail);

// Send payment conformation email
router.post("/payment-confirmation", emailController.sendPaymentConfirmationEmail);

export default router;

```

notification-service/src/routes/notification.routes.js

```

import express from "express";
import { NotificationController } from "../controllers/notification.controller.js";

const router = express.Router();
const notificationController = new NotificationController();

// Create a new notification
router.post("/", notificationController.create.bind(notificationController));

// Get all notifications for a user
router.get(
  "/:userId",
  notificationController.getAll.bind(notificationController)
);

// Mark a notification as read
router.patch(
  "/:notificationId/read",
  notificationController.markAsRead.bind(notificationController)
);

// Delete a notification
router.delete(
  "/:notificationId",
  notificationController.delete.bind(notificationController)
);

export default router;

```

notification-service/src/services/email.service.js

```
import nodemailer from "nodemailer";
import logger from "../utils/logger.js";

class EmailService {
  constructor() {
    this.transporter = nodemailer.createTransport({
      service: "gmail",
      auth: {
        user: process.env.EMAIL_USER,
        pass: process.env.EMAIL_PASS,
      },
    });
  }

  async sendEmail(to, subject, text, html = null) {
    try {
      console.log("Attempting to send email to:", to);
      const mailOptions = {
        from: process.env.EMAIL_FROM,
        to,
        subject,
        text,
        html,
      };

      const info = await this.transporter.sendMail(mailOptions);
      console.log("Email sent successfully:", info.messageId);
      return info;
    } catch (error) {
      console.error("Email sending error:", error.message);
      throw error;
    }
  }

  // Send rejection email for a restaurant
  async sendRejectionEmail(to) {
    const subject = 'Restaurant Application Rejected';
    const text = 'We regret to inform you that your restaurant application has been rejected.';
    const html = `
      <h1>Restaurant Application Rejected</h1>
      <p>We regret to inform you that your restaurant application has been rejected.</p>
      <p>Please contact support for more details.</p>
    `;
    return this.sendEmail(to, subject, text, html);
  }

  // Send approval email for a restaurant
  async sendApprovedEmail(to) {
    const subject = 'Restaurant Application Approved';
    const text = 'Congratulations! Your restaurant application has been approved.';
    const html = `
```

```
<h1>Restaurant Application Approved</h1>
<p>Congratulations! Your restaurant application has been approved.</p>
<p>You can now start managing your restaurant on our platform.</p>
`;
return this.sendEmail(to, subject, text, html);
}

// Send approval email for a restaurant
async sendBlockedEmail(to) {
  const subject = 'Restaurant Blocked';
  const text = 'your resturent is blocked.';
  const html = `
    <h1>Restaurant Application block</h1>
    <p>Your restaurant application has been approved.</p>
    <p>You can now start managing your restaurant on our platform.</p>
`;
return this.sendEmail(to, subject, text, html);
}

async sendVerificationEmail(to, pin) {
  const subject = "Email Verification";
  const text = `Your verification code is: ${pin}`;
  const html = `
    <h1>Email Verification</h1>
    <p>Your verification code is: <strong>${pin}</strong></p>
    <p>This code will expire in 10 minutes.</p>
`;
return this.sendEmail(to, subject, text, html);
}

async sendPasswordResetEmail(to, token) {
  const subject = "Password Reset";
  const text = `Your password reset token is: ${token}`;
  const html = `
    <h1>Password Reset</h1>
    <p>Your password reset token is: <strong>${token}</strong></p>
    <p>This token will expire in 1 hour.</p>
`;
return this.sendEmail(to, subject, text, html);
}

async sendOrderConfirmation(to, orderDetails) {
  const subject = "Order Confirmation";
  const text = `Thank you for your order! Order ID: ${orderDetails.orderId}`;
  const html = `
    <h1>Order Confirmation</h1>
    <p>Thank you for your order!</p>
    <p>Order ID: ${orderDetails.orderId}</p>
    <p>Total Amount: $$ ${orderDetails.totalAmount}</p>
    <p>Estimated Delivery Time: ${orderDetails.estimatedDeliveryTime}</p>
`;
```

```

`;

return this.sendEmail(to, subject, text, html);
}

async sendDeliveryUpdate(to, deliveryDetails) {
  const subject = "Delivery Update";
  const text = `Your order is on the way! Order ID: ${deliveryDetails.orderId}`;
  const html = `
    <h1>Delivery Update</h1>
    <p>Your order is on the way!</p>
    <p>Order ID: ${deliveryDetails.orderId}</p>
    <p>Status: ${deliveryDetails.status}</p>
    <p>Estimated Arrival: ${deliveryDetails.estimatedArrival}</p>
  `;

  return this.sendEmail(to, subject, text, html);
}

// send payment confirmation email
async sendPaymentConfirmationEmail(to, paymentDetails) {
  const subject = "Payment Confirmation";
  const text = `Your payment for order ${paymentDetails.orderId} has been successfully processed!`;
  const itemsList = paymentDetails.items
    .map(item => `<li>${item.name} - $$ ${item.price} x ${item.quantity} = $$ ${item.totalPrice}</li>`)
    .join("");
  const html = `
    <h1>Payment Confirmation</h1>
    <p>Your payment for order <strong>${paymentDetails.orderId}</strong> has been successfully
    processed!</p>
    <p><strong>Details:</strong></p>
    <ul>
      <li>Total Amount: $$ ${paymentDetails.totalAmount}</li>
      <li>Payment Method: ${paymentDetails.paymentMethod}</li>
      <li>Transaction ID: ${paymentDetails.transactionId}</li>
    </ul>
    <p><strong>Items:</strong></p>
    <ul>${itemsList}</ul>
    <p>Thank you for your purchase!</p>
  `;

  return this.sendEmail(to, subject, text, html);
}

export default new EmailService();

```

notification-service/src/services/sms.service.js

```

import twilio from "twilio";
import logger from "../utils/logger.js";

class SMSService {

```

```

constructor() {
  this.client = twilio(
    process.env.TWILIO_ACCOUNT_SID,
    process.env.TWILIO_AUTH_TOKEN
  );
}

async sendSMS(to, message) {
  try {
    const response = await this.client.messages.create({
      body: message,
      to,
      from: process.env.TWILIO_PHONE_NUMBER,
    });

    logger.info(`SMS sent: ${response.sid}`);
    return response;
  } catch (error) {
    logger.error("SMS sending error:", error);
    throw error;
  }
}

async sendOrderConfirmation(to, orderDetails) {
  const message = `Thank you for your order! Order ID: ${orderDetails.orderId}. Total: ${orderDetails.totalAmount}. Estimated delivery: ${orderDetails.estimatedDeliveryTime}`;
  return this.sendSMS(to, message);
}

async sendDeliveryUpdate(to, deliveryDetails) {
  const message = `Your order is on the way! Order ID: ${deliveryDetails.orderId}. Status: ${deliveryDetails.status}. Estimated arrival: ${deliveryDetails.estimatedArrival}`;
  return this.sendSMS(to, message);
}

export default new SMSService();

```

notification-service/src/utils/logger.js

```

import winston from "winston";

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || "info",
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.Console({
      format: winston.format.combine(
        winston.format.colorize(),

```

```

    winston.format.simple()
),
}),
new winston.transports.File({ filename: "error.log", level: "error" }),
new winston.transports.File({ filename: "combined.log" }),
],
});
};

export default logger;

```

notification-service/src/index.js

```

import "dotenv/config";
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import morgan from "morgan";
import logger from "./utils/logger.js";
import emailRoutes from "./routes/email.routes.js";
import notificationRoutes from "./routes/notification.routes.js";

const app = express();

// Middleware
app.use(cors());
app.use(express.json());
app.use(morgan("dev"));

// Routes
app.use("/api/email", emailRoutes);
app.use("/api/notifications", notificationRoutes);

// Error handling middleware
app.use((err, req, res, next) => {
  logger.error(err.stack);
  res.status(500).json({ message: "Something went wrong!" });
});

// Database connection
mongoose
  .connect(process.env.MONGODB_URI)
  .then(() => {
    logger.info("Connected to MongoDB");
    app.listen(process.env.PORT, () => {
      logger.info(`Notification service running on port ${process.env.PORT}`);
    });
  })
  .catch((err) => {
    logger.error("MongoDB connection error:", err);
    process.exit(1);
  });

```

10.7 order-service

order-service/src/controllers/order.controller.js

```
import { orderService } from "../services/order.service.js";
import logger from "../utils/logger.js";

class OrderController {
  async createOrder(req, res) {
    try {
      const order = await orderService.createOrder(req.body);
      logger.info("Order created successfully", { orderId: order._id });
      res.status(201).json(order);
    } catch (error) {
      logger.error("Error in createOrder controller:", error);
      res.status(400).json({ message: error.message });
    }
  }

  async getOrderById(req, res) {
    try {
      const order = await orderService.getOrderById(req.params.orderId);
      res.json(order);
    } catch (error) {
      logger.error("Error in getOrderById controller:", error);
      res.status(404).json({ message: error.message });
    }
  }

  async getUserOrders(req, res) {
    try {
      const orders = await orderService.getUserOrders(req.params.userId);
      res.json(orders);
    } catch (error) {
      logger.error("Error in getUserOrders controller:", error);
      res.status(400).json({ message: error.message });
    }
  }

  async getRestaurantOrders(req, res) {
    try {
      const orders = await orderService.getRestaurantOrders(
        req.params.restaurantId
      );
      res.json(orders);
    } catch (error) {
      logger.error("Error in getRestaurantOrders controller:", error);
      res.status(400).json({ message: error.message });
    }
  }

  async updateOrderStatus(req, res) {
    try {
```

```

const { orderId } = req.params;
const { status } = req.body;
const order = await orderService.updateOrderStatus(orderId, status);
logger.info("Order status updated", { orderId, status });
res.json(order);
} catch (error) {
  logger.error("Error in updateOrderStatus controller:", error);
  res.status(400).json({ message: error.message });
}
}

// Updates the status of a specific order by its order ID
async updatePaymentStatus(req, res) {
  try {
    const { orderId } = req.params;
    const { paymentStatus, paymentId } = req.body;
    const order = await orderService.updatePaymentStatus(
      orderId,
      paymentStatus,
      paymentId
    );
    logger.info("Payment status updated", { orderId, paymentStatus });
    res.json(order);
  } catch (error) {
    logger.error("Error in updatePaymentStatus controller:", error);
    res.status(400).json({ message: error.message });
  }
}

async cancelOrder(req, res) {
  try {
    const order = await orderService.cancelOrder(req.params.orderId);
    logger.info("Order cancelled", { orderId: order._id });
    res.json(order);
  } catch (error) {
    logger.error("Error in cancelOrder controller:", error);
    res.status(400).json({ message: error.message });
  }
}

async deleteOrder(req, res) {
  try {
    const { orderId } = req.params;
    await orderService.deleteOrder(orderId);
    logger.info("Order deleted successfully", { orderId });
    res.status(204).send();
  } catch (error) {
    logger.error("Error in deleteOrder controller:", error);
    res.status(400).json({ message: error.message });
  }
}

// Fetches all orders associated with a specific restaurant ID
async getRestaurantOrders(req, res) {

```

```

try {
  const orders = await orderService.getRestaurantOrders(
    req.params.restaurantId
  );
  res.json(orders);
} catch (error) {
  logger.error("Error in getRestaurantOrders controller:", error);
  res.status(400).json({ message: error.message });
}

// Updates the status of a specific order by its order ID
// async updateOrderStatus(req, res) {
//   try {
//     const { orderId } = req.params;
//     const { status } = req.body;
//     const order = await orderService.updateOrderStatus(orderId, status);
//     logger.info("Order status updated", { orderId, status });
//     res.json(order);
//   } catch (error) {
//     logger.error("Error in updateOrderStatus controller:", error);
//     res.status(400).json({ message: error.message });
//   }
// }

}

export const orderController = new OrderController();

```

order-service/src/middleware/validate.middleware.js

```

import { body, validationResult } from "express-validator";
import logger from "../utils/logger.js";

export const validateOrder = [
  body("userId").notEmpty().withMessage("User ID is required"),
  body("restaurantId").notEmpty().withMessage("Restaurant ID is required"),
  body("items").isArray().withMessage("Items must be an array"),
  body("items.*.menuItemId").notEmpty().withMessage("Menu item ID is required"),
  body("items.*.name").notEmpty().withMessage("Item name is required"),
  body("items.*.quantity")
    .isInt({ min: 1 })
    .withMessage("Quantity must be at least 1"),
  body("items.*.price")
    .isFloat({ min: 0 })
    .withMessage("Price must be a positive number"),
  body("deliveryAddress")
    .isObject()
    .withMessage("Delivery address is required"),
  body("deliveryAddress.street").notEmpty().withMessage("Street is required"),
  body("deliveryAddress.city").notEmpty().withMessage("City is required"),
  body("deliveryAddress.state").notEmpty().withMessage("State is required"),
]

```

```

body("deliveryAddress.zipCode")
    .notEmpty()
    .withMessage("Zip code is required"),
body("deliveryAddress.country").notEmpty().withMessage("Country is required"),
body("paymentMethod")
    .isIn(["CREDIT_CARD", "DEBIT_CARD", "CASH", "WALLET"])
    .withMessage("Invalid payment method"),
];
;

export const validateOrderStatus = [
  body("status")
    .isIn([
      "PENDING",
      "CONFIRMED",
      "PREPARING",
      "READY",
      "OUT_FOR_DELIVERY",
      "DELIVERED",
      "CANCELLED",
    ])
    .withMessage("Invalid order status"),
];
;

export const validatePaymentStatus = [
  body("paymentStatus")
    .isIn(["PENDING", "PAID", "FAILED", "REFUNDED"])
    .withMessage("Invalid payment status"),
];
;

export const validateRequest = (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    logger.error("Validation error:", errors.array());
    return res.status(400).json({ errors: errors.array() });
  }
  next();
};

```

order-service/src/models/order.model.js

```

import mongoose from "mongoose";

const orderSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
  },
  restaurantId: {
    type: String,
    required: true,
  },
  items: [

```

```
{  
  menuItemId: {  
    type: String,  
    required: true,  
  },  
  name: {  
    type: String,  
    required: true,  
  },  
  quantity: {  
    type: Number,  
    required: true,  
    min: 1,  
  },  
  price: {  
    type: Number,  
    required: true,  
  },  
  totalPrice: {  
    type: Number,  
    required: false,  
  },  
},  
],  
totalAmount: {  
  type: Number,  
  required: true,  
  default: 0,  
},  
status: {  
  type: String,  
  enum: [  
    "PENDING",  
    "CONFIRMED",  
    "PREPARING",  
    "READY",  
    "OUT_FOR_DELIVERY",  
    "DELIVERED",  
    "CANCELLED",  
  ],  
  default: "PENDING",  
},  
deliveryAddress: {  
  street: String,  
  city: String,  
  state: String,  
  zipCode: String,  
  country: String,  
},  
paymentStatus: {  
  type: String,  
  enum: ["PENDING", "PAID", "FAILED", "REFUNDED"],  
  default: "PENDING",  
},
```

```

},
paymentMethod: {
  type: String,
  enum: ["CREDIT_CARD", "DEBIT_CARD", "CASH", "WALLET"],
  required: true,
},
createdAt: {
  type: Date,
  default: Date.now,
},
updatedAt: {
  type: Date,
  default: Date.now,
},
});

// Calculate total price for each item and total amount before saving
orderSchema.pre("save", function (next) {
  // Calculate total price for each item
  this.items.forEach((item) => {
    item.totalPrice = item.price * item.quantity;
  });

  // Calculate total amount
  this.totalAmount = this.items.reduce(
    (total, item) => total + item.totalPrice,
    0
  );

  // Update timestamp
  this.updatedAt = Date.now();
  next();
});

export const Order = mongoose.model("Order", orderSchema);

```

order-service/src/routes/order.routes.js

```

import express from "express";
import { orderController } from "../controllers/order.controller.js";
import {
  validateOrder,
  validateOrderStatus,
  validatePaymentStatus,
  validateRequest,
} from "../middleware/validate.middleware.js";

const router = express.Router();

// Routes
router.post("/", validateOrder, validateRequest, orderController.createOrder);
router.get("/:orderId", orderController.getOrderById);

```

```

router.get("/user/:userId", orderController.getUserOrders);
router.get("/restaurant/:restaurantId", orderController.getRestaurantOrders);
router.get(
  "/restaurant/:restaurantId",
  validateRequest,
  orderController.getRestaurantOrders
)
router.patch(
  "/:orderId/status",
  validateOrderStatus,
  validateRequest,
  orderController.updateOrderStatus
);
router.patch(
  "/:orderId/payment",
  validatePaymentStatus,
  validateRequest,
  orderController.updatePaymentStatus
);
router.post("/:orderId/cancel", orderController.cancelOrder);
router.delete("/:orderId", orderController.deleteOrder);

```

----- resturent routes -----

```
export default router;
```

order-service/src/services/order.service.js

```

import { Order } from "../models/order.model.js";
import logger from "../utils/logger.js";

class OrderService {
  async createOrder(orderData) {
    try {
      const order = new Order(orderData);
      return await order.save();
    } catch (error) {
      logger.error("Error creating order:", error);
      throw new Error(`Error creating order: ${error.message}`);
    }
  }

  async getOrderById(orderId) {
    try {
      const order = await Order.findById(orderId);
      if (!order) {
        throw new Error("Order not found");
      }
      return order;
    } catch (error) {
      logger.error("Error fetching order:", error);
    }
  }
}

```

```
        throw new Error(`Error fetching order: ${error.message}`);
    }
}

async getUserOrders(userId) {
    try {
        return await Order.find({ userId }).sort({ createdAt: -1 });
    } catch (error) {
        logger.error("Error fetching user orders:", error);
        throw new Error(`Error fetching user orders: ${error.message}`);
    }
}

async getRestaurantOrders(restaurantId) {
    try {
        return await Order.find({ restaurantId }).sort({ createdAt: -1 });
    } catch (error) {
        logger.error("Error fetching restaurant orders:", error);
        throw new Error(`Error fetching restaurant orders: ${error.message}`);
    }
}

async updateOrderStatus(orderId, status) {
    try {
        const order = await Order.findByIdAndUpdate(
            orderId,
            { status },
            { new: true }
        );
        if (!order) {
            throw new Error("Order not found");
        }
        return order;
    } catch (error) {
        logger.error("Error updating order status:", error);
        throw new Error(`Error updating order status: ${error.message}`);
    }
}

async updatePaymentStatus(orderId, paymentStatus, paymentId) {
    try {
        const order = await Order.findByIdAndUpdate(
            orderId,
            { paymentStatus, paymentId },
            { new: true }
        );
        if (!order) {
            throw new Error("Order not found");
        }
        return order;
    } catch (error) {
        logger.error("Error updating payment status:", error);
        throw new Error(`Error updating payment status: ${error.message}`);
    }
}
```

```

        }
    }

async cancelOrder(orderId) {
    try {
        const order = await Order.findByIdAndUpdate(
            orderId,
            { status: "cancelled" },
            { new: true }
        );
        if (!order) {
            throw new Error("Order not found");
        }
        return order;
    } catch (error) {
        logger.error("Error cancelling order:", error);
        throw new Error(`Error cancelling order: ${error.message}`);
    }
}

```

```

async deleteOrder(orderId) {
    try {
        const order = await Order.findByIdAndDelete(orderId);
        if (!order) {
            throw new Error("Order not found");
        }
        logger.info("Order deleted from database", { orderId });
    } catch (error) {
        logger.error("Error deleting order:", error);
        throw new Error(`Error deleting order: ${error.message}`);
    }
}

```

```

// Retrieves all orders for a given restaurant ID, sorted by creation date (newest first)
async getRestaurantOrders(restaurantId) {
    try {
        return await Order.find({ restaurantId }).sort({ createdAt: -1 });
    } catch (error) {
        logger.error("Error fetching restaurant orders:", error);
        throw new Error(`Error fetching restaurant orders: ${error.message}`);
    }
}

```

```

// Updates the status of an order by its ID and returns the updated order
// async updateOrderStatus(orderId, status) {
//     try {
//         const order = await Order.findByIdAndUpdate(
//             orderId,
//             { status },
//             { new: true }
//         );
//         if (!order) {
//             throw new Error("Order not found");
//         }
//     }
// }

```

```
// }
// return order;
// } catch (error) {
//   logger.error("Error updating order status:", error);
//   throw new Error(`Error updating order status: ${error.message}`);
// }
// }
}

export const orderService = new OrderService();
```

order-service/src/utils/logger.js

```
import winston from "winston";

const logger = winston.createLogger({
  level: "info",
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.File({ filename: "error.log", level: "error" }),
    new winston.transports.File({ filename: "combined.log" })
  ],
});

if (process.env.NODE_ENV !== "production") {
  logger.add(
    new winston.transports.Console({
      format: winston.format.simple(),
    })
  );
}

export default logger;
```

order-service/src/index.js

```
import "dotenv/config";
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import morgan from "morgan";
import cookieParser from "cookie-parser";
import logger from "./utils/logger.js";
import orderRoutes from "./routes/order.routes.js";

const app = express();

// Middleware
```

```
app.use(
  cors({
    origin: "*",
    methods: ["GET", "POST", "PUT", "DELETE", "PATCH"],
    allowedHeaders: ["Content-Type", "Authorization"],
    credentials: true,
  })
);

app.use(express.json({ limit: "50mb" }));
app.use(express.urlencoded({ extended: true, limit: "50mb" }));
app.use(cookieParser());
app.use(morgan("dev"));

// Routes
app.use("/api/orders", orderRoutes);

// Error handling middleware
app.use((err, req, res, next) => {
  logger.error(err.stack);
  res.status(500).json({ message: "Something went wrong!" });
});

// Database connection
mongoose
  .connect(process.env.MONGODB_URI || "mongodb://localhost:27017/order-service")
  .then(() => {
    logger.info("Connected to MongoDB", {
      timestamp: new Date().toISOString(),
    });
    app.listen(process.env.PORT || 3003, () => {
      logger.info(`Order service running on port ${process.env.PORT || 3003}`, {
        timestamp: new Date().toISOString(),
      });
    });
  })
  .catch((err) => {
    logger.error("MongoDB connection error:", err);
    process.exit(1);
});
```

10.8 payment-service

payment-service/src/controllers/payment.controller.js

```
import { processPayment, handlePayhereNotification } from "../services/payment.service.js";
import logger from "../utils/logger.js";

export const createPayment = async (req, res) => {
  try {
    const {
      userId,
      cartId,
      orderId,
      restaurantId,
      items,
      totalAmount,
      paymentMethod,
      cardDetails,
    } = req.body;

    // Extract JWT token from Authorization header
    const token = req.headers.authorization?.split(" ")[1];
    if (!token) {
      throw new Error("Authorization token missing");
    }

    // Use ngrok URL for PayHere
    const baseUrl = process.env.API_GATEWAY_URL || "http://localhost:3010";
    const returnUrl = `${baseUrl}/api/payments/return`;
    const cancelUrl = `${baseUrl}/api/payments/cancel`;
    const notifyUrl = `${baseUrl}/api/payments/notify`;

    const result = await processPayment({
      userId,
      cartId,
      orderId,
      restaurantId,
      items,
      totalAmount,
      paymentMethod,
      cardDetails,
      returnUrl,
      cancelUrl,
      notifyUrl,
      token, // Pass JWT token
    });

    res.status(201).json({
      success: true,
      data: result,
      message: "Payment initiated successfully",
    });
  } catch (error) {
```

```

logger.error("Create payment error:", error);
res.status(400).json({
  success: false,
  message: error.message,
});
}

export const handleNotification = async (req, res) => {
  try {
    const notification = req.body;
    const result = await handlePayhereNotification(notification);
    res.status(200).json({
      success: true,
      data: result,
      message: "Notification processed successfully",
    });
  } catch (error) {
    logger.error("Notification handler error:", error);
    res.status(400).json({
      success: false,
      message: error.message,
    });
  }
};

export const handleReturn = async (req, res) => {
  res.redirect("http://localhost:5173/orders");
};

export const handleCancel = async (req, res) => {
  res.redirect("http://localhost:5173/cart");
};

```

payment-service/src/models/payment.model.js

```

import mongoose from "mongoose";

const paymentItemSchema = new mongoose.Schema({
  menuItemId: {
    type: String,
    required: true,
  },
  restaurantId: {
    type: String,
    required: true,
  },
  name: {
    type: String,
    required: true,
  },
  price: {

```

```
type: Number,
  required: true,
},
quantity: {
  type: Number,
  required: true,
  min: 1,
},
totalPrice: {
  type: Number,
  required: true,
},
});
};

const paymentSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
  },
  orderId: {
    type: String,
    required: true,
    unique: true,
  },
  cartId: {
    type: String,
    required: true,
  },
  restaurantId: {
    type: String,
    required: true,
  },
  items: [paymentItemSchema],
  totalAmount: {
    type: Number,
    required: true,
  },
  paymentMethod: {
    type: String,
    enum: ["CREDIT_CARD", "DEBIT_CARD"],
    required: true,
  },
  paymentStatus: {
    type: String,
    enum: ["PENDING", "COMPLETED", "FAILED", "REFUNDED"],
    default: "PENDING",
  },
  payhereTransactionId: {
    type: String,
    required: false,
  },
  payhereStatusCode: {
    type: Number,
```

```

    required: false,
  },
  payhereStatusMessage: {
    type: String,
    required: false,
  },
  cardDetails: {
    maskedCardNumber: {
      type: String,
      required: false,
    },
    cardHolderName: {
      type: String,
      required: false,
    },
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
  updatedAt: {
    type: Date,
    default: Date.now,
  },
);
paymentSchema.pre("save", function (next) {
  this.updatedAt = Date.now();
  next();
});

export const Payment = mongoose.model("Payment", paymentSchema);

```

payment-service/src/routes/payment.routes.js

```

import express from "express";
import {
  createPayment,
  handleNotification,
  handleReturn,
  handleCancel,
} from "../controllers/payment.controller.js";

const router = express.Router();

router.post("/process", createPayment);
router.post("/notify", handleNotification);
router.get("/return", handleReturn);
router.get("/cancel", handleCancel);

export default router;

```

payment-service/src/services/email.client.js

```
import axios from 'axios';
import axiosRetry from 'axios-retry';
import logger from './utils/logger.js';

// Base URL for the email service via API Gateway
const EMAIL_SERVICE_URL = process.env.EMAIL_SERVICE_URL || 'http://localhost:3010/api/email';

// Configure axios with retries for network errors
const axiosInstance = axios.create();
axiosRetry(axiosInstance, {
  retries: 3,
  retryDelay: (retryCount) => retryCount * 1000, // 1s, 2s, 3s
  retryCondition: (error) => error.code === 'ENOTFOUND' || error.code === 'ECONNREFUSED' ||
  axios.isRetryableError(error),
});

class EmailClient {
  // Validate email address format
  #validateEmail(email) {
    if (!email || typeof email !== 'string' || !email.includes('@')) {
      throw new Error('Invalid email address');
    }
  }
}

async sendPaymentConfirmationEmail(email, paymentDetails) {
  try {
    this.#validateEmail(email);
    await axiosInstance.post(` ${EMAIL_SERVICE_URL}/payment-confirmation`, {
      email,
      paymentDetails,
    });
    logger.info('Payment confirmation email sent successfully', { email });
    return true;
  } catch (error) {
    logger.error('Failed to send payment confirmation email', {
      email,
      error: error.message,
      stack: error.stack,
    });
    throw error;
  }
}

export const emailClient = new EmailClient();
```

payment-service/src/services/payment.service.js

```
import axios from "axios";
import crypto from "crypto";
import jwt from "jsonwebtoken";
import { Payment } from "../models/payment.model.js";
import logger from "../utils/logger.js";
import { emailClient } from "./email.client.js";

export const processPayment = async ({
  userId,
  cartId,
  orderId,
  restaurantId,
  items,
  totalAmount,
  paymentMethod,
  cardDetails,
  returnUrl,
  cancelUrl,
  notifyUrl,
  token,
}) => {
  try {
    // Validate inputs
    if (!userId || !cartId || !orderId || !restaurantId || !items || !totalAmount || !paymentMethod || !token) {
      throw new Error("Missing required fields");
    }
    if (!cardDetails || !cardDetails.cardNumber || !cardDetails.cardHolderName) {
      throw new Error("Invalid card details");
    }
    if (isNaN(totalAmount) || totalAmount <= 0) {
      throw new Error("Invalid total amount");
    }

    // Decode JWT to get user email
    let userData = {
      firstName: cardDetails.cardHolderName.split(" ")[0],
      lastName: cardDetails.cardHolderName.split(" ").slice(1).join(" ") || "N/A",
      email: "customer@example.com",
      phone: "0771234567",
      address: { street: "N/A", city: "Colombo", country: "Sri Lanka" },
    };

    try {
      const decoded = jwt.decode(token);
      if (decoded && decoded.email) {
        userData.email = decoded.email;
        logger.info(`Extracted email from JWT: ${userData.email}`);
      } else {
        logger.warn("No email found in JWT, using default");
      }
    } catch (error) {
```

```

logger.warn("Failed to decode JWT:", error.message);
}

// Attempt to fetch additional user data (optional)
try {
  const response = await axios.get(
    `${process.env.API_GATEWAY_URL}/api/users/${userId}`,
    { headers: { Authorization: `Bearer ${token}` } }
  );
  logger.info("User service response:", response.data);
  const user = response.data.user || response.data.data || response.data;
  if (user) {
    userData = {
      firstName: user.firstName || userData.firstName,
      lastName: user.lastName || userData.lastName,
      email: userData.email, // Prioritize JWT email
      phone: user.phone || userData.phone,
      address: user.address || userData.address,
    };
  }
} catch (error) {
  logger.warn("Failed to fetch user data:", error.message);
}

// Prepare PayHere payment request
const merchantId = process.env.PAYHERE_MERCHANT_ID;
const merchantSecret = process.env.PAYHERE_MERCHANT_SECRET;
if (!merchantId || !merchantSecret) {
  throw new Error("PayHere merchant configuration missing");
}

const orderIdUnique = `${orderId}-${Date.now()}`;
const currency = "LKR";
const formattedAmount = Number(totalAmount).toFixed(2);

// Generate PayHere hash
const hashedSecret = crypto
  .createHash("md5")
  .update(merchantSecret)
  .digest("hex")
  .toUpperCase();
const hashString = `${merchantId}${orderIdUnique}${formattedAmount}${currency}${hashedSecret}`;
logger.info("PayHere hash string:", hashString);
const hash = crypto
  .createHash("md5")
  .update(hashString)
  .digest("hex")
  .toUpperCase();

const payherePayload = {
  merchant_id: merchantId,
  return_url: returnUrl,
  cancel_url: cancelUrl,
}

```

```

notify_url: notifyUrl,
order_id: orderIdUnique,
items: items.map((item) => item.name).join(", ") || "Order Items",
currency,
amount: formattedAmount,
first_name: userData.firstName,
last_name: userData.lastName,
email: userData.email,
phone: userData.phone,
address: userData.address.street,
city: userData.address.city,
country: userData.address.country,
custom_1: userId,
custom_2: cartId,
};

logger.info("PayHere payload:", payherePayload);

// Save initial payment record
const payment = new Payment({
  userId,
  orderId,
  cartId,
  restaurantId,
  items,
  totalAmount,
  paymentMethod,
  paymentStatus: "PENDING",
  cardDetails: {
    maskedCardNumber: cardDetails.cardNumber.slice(-4).padStart(16, "*****"),
    cardHolderName: cardDetails.cardHolderName,
  },
});

await payment.save();

return {
  paymentId: payment._id,
  payherePayload,
  hash,
  paymentStatus: "PENDING",
};

} catch (error) {
  logger.error("Payment processing error:", error);
  throw new Error(`Payment initiation failed: ${error.message}`);
}
};

export const handlePayhereNotification = async (notification) => {
  try {
    const {
      merchant_id,
      order_id,
    }
  }

```

```

payment_id,
status_code,
status_message,
md5sig,
} = notification;

// Verify PayHere signature
const merchantSecret = process.env.PAYHERE_MERCHANT_SECRET;
const hashedSecret = crypto
  .createHash("md5")
  .update(merchantSecret)
  .digest("hex")
  .toUpperCase();
const localMd5sig = crypto
  .createHash("md5")
  .update(
    `${merchant_id}${order_id}${payment_id}${status_code}${hashedSecret}`
  )
  .digest("hex")
  .toUpperCase();

if (localMd5sig !== md5sig) {
  throw new Error("Invalid PayHere signature");
}

// Find payment by orderId
const originalOrderId = order_id.split("-")[0];
const payment = await Payment.findOne({ orderId: originalOrderId });

if (!payment) {
  throw new Error("Payment record not found");
}

// Update payment status
let paymentStatus = "PENDING";
if (status_code === "2") {
  paymentStatus = "COMPLETED";
} else if (status_code === "-1" || status_code === "-2") {
  paymentStatus = "FAILED";
} else if (status_code === "-3") {
  paymentStatus = "REFUNDED";
} else if (status_code === "0") {
  paymentStatus = "PENDING";
}

payment.paymentStatus = paymentStatus;
payment.payhereTransactionId = payment_id;
payment.payhereStatusCode = status_code;
payment.payhereStatusMessage = status_message;
await payment.save();

logger.info(`Payment status updated: ${paymentStatus} for order ${originalOrderId}`);

```

```

// Update order status
let orderStatus = "PENDING";
let orderPaymentStatus = "PENDING";
if (paymentStatus === "COMPLETED") {
  orderStatus = "CONFIRMED";
  orderPaymentStatus = "PAID";
} else if (paymentStatus === "FAILED") {
  orderStatus = "CANCELLED";
  orderPaymentStatus = "FAILED";
} else if (paymentStatus === "REFUNDED") {
  orderStatus = "CANCELLED";
  orderPaymentStatus = "REFUNDED";
}

try {
  await axios.patch(
    `${process.env.API_GATEWAY_URL}/api/orders/${payment.orderId}`,
    {
      paymentStatus: orderPaymentStatus,
      status: orderStatus,
    }
  );
  logger.info(`Order status updated: ${orderStatus} for order ${payment.orderId}`);
} catch (error) {
  logger.error("Failed to update order:", error.message);
  throw new Error("Order update failed");
}

// Clear cart and send email if payment is COMPLETED
if (paymentStatus === "COMPLETED") {
  try {
    await axios.delete(
      `${process.env.API_GATEWAY_URL}/api/carts/${payment.cartId}`
    );
    logger.info(`Cart cleared for cartId ${payment.cartId}`);
  } catch (error) {
    logger.error("Failed to clear cart:", error.message);
  }
}

// Send payment confirmation email
try {
  // Fetch user email from payment record or JWT (stored in payherePayload.email)
  let userEmail = payment.payherePayload?.email || "customer@example.com";
  try {
    const userResponse = await axios.get(
      `${process.env.API_GATEWAY_URL}/api/users/${payment.userId}`
    );
    const user = userResponse.data.user || userResponse.data.data || userResponse.data;
    if (user && user.email) {
      userEmail = user.email;
    }
  } catch (error) {
    logger.warn("Failed to fetch user email, using stored email:", error.message);
  }
}

```

```

    }

    await emailClient.sendPaymentConfirmationEmail(userEmail, {
      orderId: payment.orderId,
      totalAmount: payment.totalAmount,
      paymentMethod: payment.paymentMethod,
      transactionId: payment.payhereTransactionId,
      items: payment.items,
    });
    logger.info(`Payment confirmation email sent to ${userEmail}`);
  } catch (error) {
    logger.error("Failed to send payment confirmation email:", error.message);
  }
}

return { success: true, paymentId: payment._id, paymentStatus };
} catch (error) {
  logger.error("PayHere notification error:", error);
  throw new Error(`Notification processing failed: ${error.message}`);
}
};


```

payment-service/src/utils/logger.js

```

import winston from "winston";

const logger = winston.createLogger({
  level: "info",
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.File({ filename: "error.log", level: "error" }),
    new winston.transports.File({ filename: "combined.log" }),
  ],
});

if (process.env.NODE_ENV !== "production") {
  logger.add(
    new winston.transports.Console({
      format: winston.format.simple(),
    })
  );
}

export default logger;

```

payment-service/src/index.js

```
import "dotenv/config";
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import morgan from "morgan";
import cookieParser from "cookie-parser";
import logger from "./utils/logger.js";
import paymentRoutes from "./routes/payment.routes.js";
const app = express();

// Middleware
app.use(
  cors({
    origin: "*",
    methods: ["GET", "POST", "PUT", "DELETE", "PATCH"],
    allowedHeaders: ["Content-Type", "Authorization"],
    credentials: true,
  })
);
app.use(express.json({ limit: "50mb" }));
app.use(express.urlencoded({ extended: true, limit: "50mb" }));
app.use(cookieParser());
app.use(morgan("dev"));

// Routes
app.use("/api/payments", paymentRoutes);

// Error handling middleware
app.use((err, req, res, next) => {
  logger.error(err.stack);
  res.status(500).json({ message: "Something went wrong!" });
});

// Database connection
mongoose
  .connect(process.env.MONGODB_URI || "mongodb://localhost:27017/payment-service")
  .then(() => {
    logger.info("Connected to MongoDB", {
      timestamp: new Date().toISOString(),
    });
    app.listen(process.env.PORT || 3003, () => {
      logger.info(`Payment service running on port ${process.env.PORT || 3003}`, {
        timestamp: new Date().toISOString(),
      });
    });
  })
  .catch((err) => {
    logger.error("MongoDB connection error:", err);
    process.exit(1);
  });
}
```

10.9 restaurant-service

restaurant-service/src/controllers/category.controller.js

```
import { CategoryService } from './services/category.service.js';
import logger from './utils/logger.js';

export class CategoryController {
  constructor() {
    this.categoryService = new CategoryService();
    this.getCategories = this.getCategories.bind(this);
    this.addCategory = this.addCategory.bind(this);
    this.updateCategory = this.updateCategory.bind(this);
    this.deleteCategory = this.deleteCategory.bind(this);
  }

  async getCategories(req, res) {
    try {
      const { restaurantId } = req.params;
      logger.info('Fetching categories for restaurant:', { restaurantId });

      const categories = await this.categoryService.getCategories(restaurantId, req.user.id);

      logger.info('Categories fetched successfully:', { restaurantId });

      res.status(200).json(categories);
    } catch (error) {
      logger.error('Get categories error:', { message: error.message, stack: error.stack });
      if (error.isOperational) {
        return res.status(error.statusCode).json({ message: error.message });
      }
      res.status(500).json({ message: 'Error fetching categories' });
    }
  }

  async addCategory(req, res) {
    try {
      const { restaurantId } = req.params;
      logger.info('Adding category for restaurant:', { restaurantId });

      const category = await this.categoryService.addCategory(
        restaurantId,
        req.body,
        req.user.id
      );

      logger.info('Category added successfully:', { categoryId: category._id });

      res.status(201).json({
        message: 'Category added successfully',
        category,
      });
    } catch (error) {
  
```

```

logger.error('Add category error:', { message: error.message, stack: error.stack });
if (error.isOperational) {
  return res.status(error.statusCode).json({ message: error.message });
}
res.status(500).json({ message: 'Error adding category' });
}

async updateCategory(req, res) {
  try {
    const { restaurantId, categoryId } = req.params;
    logger.info('Updating category:', { restaurantId, categoryId });

    const updatedCategory = await this.categoryService.updateCategory(
      restaurantId,
      categoryId,
      req.body,
      req.user.id
    );

    logger.info('Category updated successfully:', { categoryId });

    res.status(200).json({
      message: 'Category updated successfully',
      category: updatedCategory,
    });
  } catch (error) {
    logger.error('Update category error:', { message: error.message, stack: error.stack });
    if (error.isOperational) {
      return res.status(error.statusCode).json({ message: error.message });
    }
    res.status(500).json({ message: 'Error updating category' });
  }
}

async deleteCategory(req, res) {
  try {
    const { restaurantId, categoryId } = req.params;
    logger.info('Deleting category:', { restaurantId, categoryId });
    await this.categoryService.deleteCategory(restaurantId, categoryId, req.user.id);
    logger.info('Category deleted successfully:', { categoryId });

    res.status(200).json({ message: 'Category deleted successfully' });
  } catch (error) {
    logger.error('Delete category error:', { message: error.message, stack: error.stack });
    if (error.isOperational) {
      return res.status(error.statusCode).json({ message: error.message });
    }
    res.status(500).json({ message: 'Error deleting category' });
  }
}

```

restaurant-service/src/controllers/menu.controller.js

```
import { MenuService } from './services/menu.service.js';
import logger from './utils/logger.js';

export class MenuController {
  constructor() {
    this.menuService = new MenuService();
    this.addMenuItem = this.addMenuItem.bind(this);
    this.getMenuItems = this.getMenuItems.bind(this);
    this.getMenuItemById = this.getMenuItemById.bind(this);
    this.updateMenuItem = this.updateMenuItem.bind(this);
    this.deleteMenuItem = this.deleteMenuItem.bind(this);
  }

  // add menu items
  async addMenuItem(req, res) {
    try {
      const { restaurantId } = req.params;
      logger.info('Adding menu item for restaurant:', { restaurantId });

      const menuItem = await this.menuService.addMenuItem(
        restaurantId,
        req.body,
        req.files,
        req.user.id
      );

      logger.info('Menu item added successfully:', { menuItemId: menuItem._id });

      res.status(201).json({
        message: 'Menu item added successfully',
        menuItem,
      });
    } catch (error) {
      logger.error('Add menu item error:', { message: error.message, stack: error.stack });
      if (error.isOperational) {
        return res.status(error.statusCode).json({ message: error.message });
      }
      res.status(500).json({ message: 'Error adding menu item' });
    }
  }

  // get menu items
  async getMenuItems(req, res) {
    try {
      const { restaurantId } = req.params;
      logger.info('Fetching menu items for restaurant:', { restaurantId });

      const menuItems = await this.menuService.getMenuItems(restaurantId, req.user?.id);

      logger.info('Menu items fetched successfully:', { restaurantId });
    }
  }
}
```

```

res.status(200).json(menuItems);
} catch (error) {
logger.error('Get menu items error:', { message: error.message, stack: error.stack });
if (error.isOperational) {
  return res.status(error.statusCode).json({ message: error.message });
}
res.status(500).json({ message: 'Error fetching menu items' });
}

// fetch a specific menu item by ID
async getMenuItemById(req, res) {
try {
  const { restaurantId, menuItemId } = req.params;
  logger.info('Fetching menu item:', { restaurantId, menuItemId });

  const menuItem = await this.menuService.getMenuItemById(restaurantId, menuItemId);

  logger.info('Menu item fetched successfully:', { restaurantId, menuItemId });

  res.status(200).json(menuItem);
} catch (error) {
  logger.error('Get menu item error:', { message: error.message, stack: error.stack });
  if (error.isOperational) {
    return res.status(error.statusCode).json({ message: error.message });
  }
  res.status(500).json({ message: 'Error fetching menu item' });
}

}

// update menu items
async updateMenuItem(req, res) {
try {
  const { menuItemId } = req.params;
  logger.info('Updating menu item:', { menuItemId });

  const updatedMenuItem = await this.menuService.updateMenuItem(
    menuItemId,
    req.body,
    req.files,
    req.user.id
  );

  logger.info('Menu item updated successfully:', { menuItemId });

  res.status(200).json({
    message: 'Menu item updated successfully',
    menuItem: updatedMenuItem,
  });
} catch (error) {
  logger.error('Update menu item error:', { message: error.message, stack: error.stack });
  if (error.isOperational) {
    return res.status(error.statusCode).json({ message: error.message });
  }
}
}

```

```

        }
        res.status(500).json({ message: 'Error updating menu item' });
    }
}

// delete menu item
async deleteMenuItem(req, res) {
    try {
        const { menuItemId } = req.params;
        logger.info('Deleting menu item:', { menuItemId });

        await this.menuService.deleteMenuItem(menuItemId, req.user.id);

        logger.info('Menu item deleted successfully:', { menuItemId });

        res.status(200).json({ message: 'Menu item deleted successfully' });
    } catch (error) {
        logger.error('Delete menu item error:', { message: error.message, stack: error.stack });
        if (error.isOperational) {
            return res.status(error.statusCode).json({ message: error.message });
        }
        res.status(500).json({ message: 'Error deleting menu item' });
    }
}
}

```

restaurant-service/src/controllers/restaurant.controller.js

```

import { RestaurantService } from './services/restaurant.service.js';
import logger from './utils/logger.js';

export class RestaurantController {
    constructor() {
        this.restaurantService = new RestaurantService();
        this.registerRestaurant = this.registerRestaurant.bind(this);
        this.getRestaurantById = this.getRestaurantById.bind(this);
        this.updateRestaurantStatus = this.updateRestaurantStatus.bind(this);
        this.getAllRestaurants = this.getAllRestaurants.bind(this);
        this.updateRestaurant = this.updateRestaurant.bind(this);
        this.deleteRestaurant = this.deleteRestaurant.bind(this);
        this.getRestaurantByUserId = this.getRestaurantByUserId.bind(this);
        this.updateRestaurantAvailability = this.updateRestaurantAvailability.bind(this);
    }
    // register resturent
    async registerRestaurant(req, res) {
        try {
            const {
                restaurantName, contactPerson, phoneNumber, businessType, cuisineType, operatingHours,
                deliveryRadius, taxId, streetAddress, city, state, zipCode, country, email, password, agreeTerms,
                availability
            } = req.body;

            logger.info('Restaurant registration request received for:', { email });

```

```

const { restaurant } = await this.restaurantService.registerRestaurant({
  restaurantName, contactPerson, phoneNumber, businessType, cuisineType, operatingHours,
  deliveryRadius, taxId, streetAddress, city, state, zipCode, country, email, password, agreeTerms,
  availability
}, req.files);

logger.info('Restaurant registration successful for:', { email });

res.status(201).json({
  message: 'Restaurant registration submitted. Awaiting admin approval.',
  restaurant: {
    id: restaurant._id,
    userId: restaurant.userId,
    email: restaurant.email,
    restaurantName: restaurant.restaurantName,
    status: restaurant.status,
    availability: restaurant.availability
  }
});
} catch (error) {
  logger.error('Restaurant registration error:', {
    message: error.message,
    stack: error.stack,
    requestBody: req.body.email
  });
}

if (error.name === 'ValidationError') {
  logger.error('Validation errors:', Object.values(error.errors).map(err => err.message));
  return res.status(400).json({
    message: 'Invalid data',
    errors: Object.values(error.errors).map(err => err.message)
  });
}

if (error.isOperational) {
  logger.error('Operational error:', error.message);
  return res.status(error.statusCode).json({
    message: error.message
  });
}

logger.error('Server error:', {
  message: error.message,
  stack: error.stack
});
res.status(500).json({ message: 'Server error: ' + error.message });
}

// get resturent by ID
async getRestaurantById(req, res) {
  try {
    logger.info('Fetching restaurant by ID:', { id: req.params.id });
  }
}

```

```

const restaurant = await this.restaurantService.getRestaurantById(req.params.id);

logger.info('Restaurant fetched successfully:', { id: req.params.id });

res.status(200).json(restaurant);
} catch (error) {
  logger.error('Get restaurant error:', error.message);
  logger.error('Error stack trace:', error.stack);

  if (error.isOperational) {
    return res.status(error.statusCode).json({ message: error.message });
  }

  res.status(500).json({ message: 'Error fetching restaurant' });
}

// update resturent status
async updateRestaurantStatus(req, res) {
  try {
    const { status } = req.body;
    logger.info('Updating restaurant status:', { id: req.params.id, status });

    await this.restaurantService.updateRestaurantStatus(req.params.id, status);

    logger.info('Restaurant status updated successfully:', { id: req.params.id, status });

    res.status(200).json({ message: 'Restaurant status updated' });
  } catch (error) {
    logger.error('Update restaurant status error:', error.message);
    logger.error('Error stack trace:', error.stack);

    if (error.isOperational) {
      return res.status(error.statusCode).json({ message: error.message });
    }

    res.status(500).json({ message: 'Error updating restaurant status' });
  }
}

// get all resturent
async getAllRestaurants(req, res) {
  try {
    logger.info('Fetching all restaurants');

    // Extract page and limit from query parameters (default to 1 and 10)
    const page = req.query.page || 1;
    const limit = req.query.limit || 10;

    const result = await this.restaurantService.getAllRestaurants(page, limit);

    logger.info('Restaurants fetched successfully', {
      count: result.restaurants.length,
      totalCount: result.totalCount,
    });
  }
}

```

```

page: result.currentPage,
totalPages: result.totalPages,
});

res.status(200).json({
  success: true,
  data: result.restaurants,
  pagination: {
    totalCount: result.totalCount,
    currentPage: result.currentPage,
    totalPages: result.totalPages,
    pageSize: result.pageSize,
  },
});
} catch (error) {
  logger.error('Get all restaurants error:', error.message);
  logger.error('Error stack trace:', error.stack);

  if (error.isOperational) {
    return res.status(error.statusCode).json({ message: error.message });
  }

  res.status(500).json({ message: 'Error fetching restaurants' });
}
}

// update resturent
async updateRestaurant(req, res) {
  try {
    const { id } = req.params;
    logger.info('Updating restaurant:', { id });

    const updatedRestaurant = await this.restaurantService.updateRestaurant(id, req.body, req.files);

    logger.info('Restaurant updated successfully:', { id });

    res.status(200).json({
      message: 'Restaurant updated successfully',
      restaurant: updatedRestaurant
    });
  } catch (error) {
    logger.error('Update restaurant error:', error.message);
    logger.error('Error stack trace:', error.stack);

    if (error.name === 'ValidationError') {
      logger.error('Validation errors:', Object.values(error.errors).map(err => err.message));
      return res.status(400).json({
        message: 'Invalid data',
        errors: Object.values(error.errors).map(err => err.message)
      });
    }
  }

  if (error.isOperational) {
    logger.error('Operational error:', error.message);
  }
}

```

```

    return res.status(error.statusCode).json({ message: error.message });
}

res.status(500).json({ message: 'Error updating restaurant' });
}
}

// delete resturent
async deleteRestaurant(req, res) {
try {
  const { id } = req.params;
  logger.info('Deleting restaurant:', { id });

  await this.restaurantService.deleteRestaurant(id);

  logger.info('Restaurant deleted successfully:', { id });

  res.status(200).json({ message: 'Restaurant deleted successfully' });
} catch (error) {
  logger.error('Delete restaurant error:', error.message);
  logger.error('Error stack trace:', error.stack);

  if (error.isOperational) {
    return res.status(error.statusCode).json({ message: error.message });
  }

  res.status(500).json({ message: 'Error deleting restaurant' });
}
}

// get resturent by userId
async getRestaurantByUserId(req, res) {
try {
  const userId = req.query.userId || req.user.id; // Use req.user.id from JWT if query parameter is not provided
  if (!userId) {
    const error = new Error('User ID not provided');
    error.statusCode = 400;
    throw error;
  }

  logger.info('Fetching restaurant by userId:', { userId });

  const restaurant = await this.restaurantService.getRestaurantByUserId(userId);

  logger.info('Restaurant fetched successfully by userId:', { userId });

  res.status(200).json(restaurant);
} catch (error) {
  logger.error('Get restaurant by userId error:', error.message);
  logger.error('Error stack trace:', error.stack);

  if (error.isOperational) {
    return res.status(error.statusCode).json({ message: error.message });
  }
}
}
```

```

    res.status(500).json({ message: 'Error fetching restaurant' });
}
}
// Update restaurant availability
async updateRestaurantAvailability(req, res) {
try {
  const { availability } = req.body;
  const userId = req.user.id; // From authMiddleware

  logger.info('Updating restaurant availability:', { userId, availability });

  const { restaurant } = await this.restaurantService.updateRestaurantAvailability(userId, availability);

  logger.info('Restaurant availability updated successfully:', { userId, availability });

  res.status(200).json({
    message: 'Restaurant availability updated',
    restaurant: {
      id: restaurant._id,
      restaurantName: restaurant.restaurantName,
      availability: restaurant.availability
    }
  });
} catch (error) {
  logger.error('Update restaurant availability error:', error.message);
  logger.error('Error stack trace:', error.stack);

  if (error.isOperational) {
    return res.status(error.statusCode).json({ message: error.message });
  }

  res.status(500).json({ message: 'Error updating restaurant availability' });
}
}
}

```

restaurant-service/src/middleware/auth.js

```

import jwt from 'jsonwebtoken';
import logger from '../utils/logger.js';
import Restaurant from '../models/restaurant.model.js'; // Use import instead of require

export const authMiddleware = async (req, res, next) => {
try {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) {
    const error = new Error('No token provided');
    error.statusCode = 401;
    throw error;
  }

  const decoded = jwt.verify(token, process.env.JWT_SECRET);

```

```

// Handle both `id` and `userId` in the token payload
const userId = decoded.id || decoded.userId;
if (!userId) {
  const error = new Error('Invalid token: user ID not found');
  error.statusCode = 401;
  throw error;
}
req.user = { id: userId, role: decoded.role };

if (req.params.id || req.query.userId) {
  const restaurantId = req.params.id;
  const userIdToCheck = req.query.userId || req.user.id;

  if (req.user.role === 'RESTAURANT' && req.query.userId && req.query.userId !== req.user.id) {
    const error = new Error('Unauthorized: Cannot access other restaurants');
    error.statusCode = 403;
    throw error;
  }

  if (restaurantId) {
    const restaurant = await Restaurant.findById(restaurantId);
    if (!restaurant) {
      const error = new Error('Restaurant not found');
      error.statusCode = 404;
      throw error;
    }
    if (req.user.role === 'RESTAURANT' && restaurant.userId.toString() !== req.user.id) {
      const error = new Error('Unauthorized: Not your restaurant');
      error.statusCode = 403;
      throw error;
    }
  }
}

next();
} catch (error) {
  logger.error('Auth middleware error:', {
    message: error.message,
    stack: error.stack,
  });
  const statusCode = error.statusCode || 401;
  res.status(statusCode).json({ message: error.message });
}
};


```

restaurant-service/src/middleware/upload.js

```

import multer from 'multer';
import path from 'path';

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  }
});


```

```

},
filename: (req, file, cb) => {
  const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
  cb(null, file.fieldname + '-' + uniqueSuffix + path.extname(file.originalname));
}
});

const fileFilter = (req, file, cb) => {
  const filetypes = /jpeg|jpg|png|pdf/;
  const extname = filetypes.test(path.extname(file.originalname).toLowerCase());
  const mimetype = filetypes.test(file.mimetype);

  if (extname && mimetype) {
    return cb(null, true);
  } else {
    cb(new Error('Only images and PDFs are allowed'));
  }
};

export default multer({
  storage,
  fileFilter,
  limits: { fileSize: 5 * 1024 * 1024 } // 5MB limit
}).fields([
  { name: 'businessLicense', maxCount: 1 },
  { name: 'foodSafetyCert', maxCount: 1 },
  { name: 'exteriorPhoto', maxCount: 1 },
  { name: 'logo', maxCount: 1 },
  { name: 'mainImage', maxCount: 1 },
  { name: 'thumbnailImage', maxCount: 1 }
]);

```

restaurant-service/src/models/category.model.js

```

import mongoose from 'mongoose';

const categorySchema = new mongoose.Schema({
  restaurantId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Restaurant',
    required: true,
  },
  name: {
    type: String,
    required: true,
    unique: true,
    trim: true,
  },
  description: {
    type: String,
    trim: true,
  },
});

```

```
}, { timestamps: true });

// Index to enforce unique category names per restaurant
categorySchema.index({ restaurantId: 1, name: 1 }, { unique: true });

export default mongoose.model('Category', categorySchema);
```

restaurant-service/src/models/menu.model.js

```
import mongoose from 'mongoose';

const menuItemSchema = new mongoose.Schema({
  restaurantId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Restaurant',
    required: true
  },
  name: {
    type: String,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  price: {
    type: Number,
    required: true
  },
  category: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Category',
    required: true,
  },
  mainImage: {
    type: String
  }, // Cloudinary URL for main image
  thumbnailImage: {
    type: String
  }, // Cloudinary URL for thumbnail image
  isAvailable: {
    type: Boolean,
    default: true
  }
}, { timestamps: true });

export default mongoose.model('MenuItem', menuItemSchema);
```

restaurant-service/src/models/restaurant.model.js

```
import mongoose from 'mongoose';

const addressSchema = new mongoose.Schema({
  streetAddress: { type: String, required: true },
  city: { type: String, required: true },
  state: { type: String, required: true },
  zipCode: { type: String, required: true },
  country: { type: String, required: true }
});

const restaurantSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    unique: true
  },
  restaurantName: { type: String, required: true },
  contactPerson: { type: String, required: true },
  phoneNumber: { type: String, required: true },
  businessType: { type: String, required: true },
  cuisineType: { type: String, required: true },
  operatingHours: { type: String, required: true },
  deliveryRadius: { type: String, required: true },
  taxId: { type: String, required: true },
  address: { type: addressSchema, required: true },
  email: { type: String, required: true, unique: true },
  status: {
    type: String,
    enum: ['pending', 'approved', 'rejected', 'blocked'],
    default: 'pending'
  },
  agreeTerms: { type: Boolean, required: true },
  businessLicense: { type: String },
  foodSafetyCert: { type: String },
  exteriorPhoto: { type: String },
  logo: { type: String },
  availability: { type: Boolean, default: true }
}, { timestamps: true });

export default mongoose.model('Restaurant', restaurantSchema);
```

restaurant-service/src/routes/category.route.js

```
import express from 'express';
import { CategoryController } from '../controllers/category.controller.js';
import { authMiddleware } from '../middleware/auth.js';
import { validateCategory } from '../validation/category.validation.js';

const router = express.Router();
const categoryController = new CategoryController();
```

```
router.get('/:restaurantId/categories', authMiddleware, categoryController.getCategories);
router.post('/:restaurantId/categories', authMiddleware, validateCategory, categoryController.addCategory);
router.patch('/:restaurantId/categories/:categoryId', authMiddleware, validateCategory,
categoryController.updateCategory);
router.delete('/:restaurantId/categories/:categoryId', authMiddleware, categoryController.deleteCategory);

export default router;
```

restaurnt-service/src/routes/menu.routes.js

```
import express from 'express';
import { MenuController } from '../controllers/menu.controller.js';
import { authMiddleware } from '../middleware/auth.js';
import upload from '../middleware/upload.js';
import { validateMenuItem } from '../validation/menu.validation.js';

const router = express.Router();
const menuController = new MenuController();

// Public route
router.get('/:restaurantId/menu-items', menuController.getMenuItems);
router.get('/:restaurantId/menu-items/:menuItemID', menuController.getMenuItemById);

// Private routes
router.post('/:restaurantId/menu-items', authMiddleware, upload, validateMenuItem,
menuController.addMenuItem);
router.patch('/:restaurantId/menu-items/:menuItemID', authMiddleware, upload, validateMenuItem,
menuController.updateMenuItem);
router.delete('/:restaurantId/menu-items/:menuItemID', authMiddleware, menuController.deleteMenuItem);

export default router;
```

restaurnt-service/src/routes/restaurant.routes.js

```
import express, { Router } from 'express';
import { RestaurantController } from '../controllers/restaurant.controller.js';
import { authMiddleware } from '../middleware/auth.js';
import upload from '../middleware/upload.js';
import { validateRestaurantRegistration, validateUpdateAvailability } from
'../validation/restaurant.validation.js';

const router = express.Router();
const restaurantController = new RestaurantController();

// Middleware to check if user is an admin
const adminMiddleware = (req, res, next) => {
  if (req.user.role !== 'ADMIN') {
    const error = new Error('Unauthorized: Admins only');
    error.statusCode = 403;
    throw error;
  }
  next();
}

router.get('/restaurants', adminMiddleware, restaurantController.getRestaurants);
router.get('/restaurants/:id', adminMiddleware, restaurantController.getRestaurant);
router.post('/restaurants', adminMiddleware, validateRestaurantRegistration,
restaurantController.createRestaurant);
router.patch('/restaurants/:id', adminMiddleware, validateUpdateAvailability,
restaurantController.updateRestaurant);
router.delete('/restaurants/:id', adminMiddleware, restaurantController.deleteRestaurant);
```

```

        }
        next();
    };

// Public route - MUST be before dynamic routes like /:id
router.get('/all', restaurantController.getAllRestaurants);
router.get('/:id', restaurantController.getRestaurantById);

// Protected routes
router.post('/register', upload, validateRestaurantRegistration, restaurantController.registerRestaurant);
router.get('/', authMiddleware, restaurantController.getRestaurantByUserId);
router.patch('/availability', authMiddleware, validateUpdateAvailability,
    restaurantController.updateRestaurantAvailability );
router.patch('/:id', authMiddleware, adminMiddleware, restaurantController.updateRestaurantStatus);
router.put('/:id', authMiddleware, upload, restaurantController.updateRestaurant);
router.delete('/:id', authMiddleware, adminMiddleware, restaurantController.deleteRestaurant);

export default router;

```

restaurnt-service/src/services/category.service.js

```

import Category from './models/category.model.js';
import Restaurant from './models/restaurant.model.js';
import MenuItem from './models/menu.model.js';
import logger from './utils/logger.js';

export class CategoryService {
    async getCategories(restaurantId, userId) {
        const restaurant = await Restaurant.findById(restaurantId);
        if (!restaurant) {
            const error = new Error('Restaurant not found');
            error.statusCode = 404;
            error.isOperational = true;
            throw error;
        }

        if (restaurant.userId.toString() !== userId) {
            const error = new Error('Access denied: You do not own this restaurant');
            error.statusCode = 403;
            error.isOperational = true;
            throw error;
        }

        const categories = await Category.find({ restaurantId });
        return categories;
    }

    async addCategory(restaurantId, data, userId) {
        const restaurant = await Restaurant.findById(restaurantId);
        if (!restaurant) {
            const error = new Error('Restaurant not found');
            error.statusCode = 404;

```

```
error.isOperational = true;
throw error;
}

if (restaurant.userId.toString() !== userId) {
  const error = new Error('Access denied: You do not own this restaurant');
  error.statusCode = 403;
  error.isOperational = true;
  throw error;
}

const category = new Category({
  restaurantId,
  name: data.name,
  description: data.description, // Added
});

await category.save();
return category;
}

async updateCategory(restaurantId, categoryId, data, userId) {
  const restaurant = await Restaurant.findById(restaurantId);
  if (!restaurant) {
    const error = new Error('Restaurant not found');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }

  if (restaurant.userId.toString() !== userId) {
    const error = new Error('Access denied: You do not own this restaurant');
    error.statusCode = 403;
    error.isOperational = true;
    throw error;
  }

  const category = await Category.findOne({ _id: categoryId, restaurantId });
  if (!category) {
    const error = new Error('Category not found');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }

  if (data.name) category.name = data.name;
  if (data.description !== undefined) category.description = data.description; // Added

  await category.save();
  return category;
}

async deleteCategory(restaurantId, categoryId, userId) {
```

```

const restaurant = await Restaurant.findById(restaurantId);
if (!restaurant) {
  const error = new Error('Restaurant not found');
  error.statusCode = 404;
  error.isOperational = true;
  throw error;
}

if (restaurant.userId.toString() !== userId) {
  const error = new Error('Access denied: You do not own this restaurant');
  error.statusCode = 403;
  error.isOperational = true;
  throw error;
}

const category = await Category.findOne({ _id: categoryId, restaurantId });
if (!category) {
  const error = new Error('Category not found');
  error.statusCode = 404;
  error.isOperational = true;
  throw error;
}

const menuItems = await MenuItem.find({ category: categoryId });
if (menuItems.length > 0) {
  const error = new Error('Cannot delete category: It is used by menu items');
  error.statusCode = 400;
  error.isOperational = true;
  throw error;
}

await category.deleteOne();
return { message: 'Category deleted successfully' };
}
}

```

restaurnt-service/src/services/email.client.js

```

import axios from 'axios';
import axiosRetry from 'axios-retry';
import logger from './utils/logger.js';

// Base URL for the email service via API Gateway
const EMAIL_SERVICE_URL = process.env.EMAIL_SERVICE_URL || 'http://localhost:3010/api/email';

// Configure axios with retries for network errors
const axiosInstance = axios.create();
axiosRetry(axiosInstance, {
  retries: 3,
  retryDelay: (retryCount) => retryCount * 1000, // 1s, 2s, 3s
  retryCondition: (error) => error.code === 'ENOTFOUND' || error.code === 'ECONNREFUSED' ||
  axios.isRetryableError(error),
}

```

```
});

class EmailClient {
  // Validate email address format
  #validateEmail(email) {
    if (!email || typeof email !== 'string' || !email.includes('@')) {
      throw new Error('Invalid email address');
    }
  }

  // Send rejection email for a restaurant
  async sendRejectionEmail(email) {
    try {
      this.#validateEmail(email);
      await axiosInstance.post(` ${EMAIL_SERVICE_URL}/reject-restaurant`, { email });
      logger.info('Reject restaurant email sent successfully', { email });
      return true;
    } catch (error) {
      logger.error('Failed to send reject restaurant email', {
        email,
        error: error.message,
        stack: error.stack,
      });
      throw error;
    }
  }

  // Send approval email for a restaurant
  async sendApprovedEmail(email) {
    try {
      this.#validateEmail(email);
      await axiosInstance.post(` ${EMAIL_SERVICE_URL}/approve-restaurant`, { email });
      logger.info('Approve restaurant email sent successfully', { email });
      return true;
    } catch (error) {
      logger.error('Failed to send approve restaurant email', {
        email,
        error: error.message,
        stack: error.stack,
      });
      throw error;
    }
  }

  // Send blocked email for a restaurant
  async sendBlockedEmail(email) {
    try {
      this.#validateEmail(email);
      await axiosInstance.post(` ${EMAIL_SERVICE_URL}/block-restaurant`, { email });
      logger.info('Blocked restaurant email sent successfully', { email });
      return true;
    } catch (error) {
      logger.error('Failed to send blocked restaurant email', {

```

```

    email,
    error: error.message,
    stack: error.stack,
  });
  throw error;
}
}

export const emailClient = new EmailClient();

```

restaurnt-service/src/services/menu.service.js

```

import MenuItem from './models/menu.model.js';
import Restaurant from './models/restaurant.model.js';
import { uploadToCloudinary } from './utils/cloudinary.js';
import fs from 'fs/promises';
import logger from './utils/logger.js';

export class MenuService {

  // add menu items
  async addMenuItem(restaurantId, data, files, userId) {
    const restaurant = await Restaurant.findById(restaurantId);
    if (!restaurant) {
      const error = new Error('Restaurant not found');
      error.statusCode = 404;
      error.isOperational = true;
      throw error;
    }

    if (restaurant.userId.toString() !== userId) {
      const error = new Error('Access denied: You do not own this restaurant');
      error.statusCode = 403;
      error.isOperational = true;
      throw error;
    }

    let mainImageUrl, thumbnailImageUrl;
    try {
      if (files?.mainImage) {
        mainImageUrl = await uploadToCloudinary(files.mainImage[0].path);
        await fs.unlink(files.mainImage[0].path);
      }
      if (files?.thumbnailImage) {
        thumbnailImageUrl = await uploadToCloudinary(files.thumbnailImage[0].path);
        await fs.unlink(files.thumbnailImage[0].path);
      }
    } catch (error) {
      const uploadError = new Error(`Cloudinary upload failed: ${error.message}`);
      uploadError.statusCode = 500;
      uploadError.isOperational = true;
    }
  }
}

```

```

    throw uploadError;
}

const menuItem = new MenuItem({
  restaurantId,
  name: data.name,
  description: data.description,
  price: data.price,
  category: data.category,
  mainImage: mainImageUrl,
  thumbnailImage: thumbnailImageUrl,
  isAvailable: data.isAvailable ?? true,
});

await menuItem.save();
return menuItem;
}

// get mmenu items by resturent
async getMenuItems(restaurantId, userId = null) {
  const restaurant = await Restaurant.findById(restaurantId);
  if (!restaurant) {
    const error = new Error('Restaurant not found');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }

  // Skip ownership check if userId is not provided (public route)
  if (userId && restaurant.userId.toString() !== userId) {
    const error = new Error('Access denied: You do not own this restaurant');
    error.statusCode = 403;
    error.isOperational = true;
    throw error;
  }

  const menuItems = await MenuItem.find({ restaurantId });
  return menuItems;
}

// fetch a specific menu item by ID
async getMenuItemById(restaurantId, menuItemId) {
  const restaurant = await Restaurant.findById(restaurantId);
  if (!restaurant) {
    const error = new Error('Restaurant not found');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }

  const menuItem = await MenuItem.findOne({ _id: menuItemId, restaurantId });
  if (!menuItem) {
    const error = new Error('Menu item not found');

```

```
error.statusCode = 404;
error.isOperational = true;
throw error;
}

return menuItem;
}

// update menu item
async updateMenuItem(menuItemId, data, files, userId) {
const menuItem = await MenuItem.findById(menuItemId);
if (!menuItem) {
  const error = new Error('Menu item not found');
  error.statusCode = 404;
  error.isOperational = true;
  throw error;
}

const restaurant = await Restaurant.findById(menuItem.restaurantId);
if (restaurant.userId.toString() !== userId) {
  const error = new Error('Access denied: You do not own this restaurant');
  error.statusCode = 403;
  error.isOperational = true;
  throw error;
}

if (data.name) menuItem.name = data.name;
if (data.description) menuItem.description = data.description;
if (data.price) menuItem.price = data.price;
if (data.category) menuItem.category = data.category;
if (data.isAvailable !== undefined) menuItem.isAvailable = data.isAvailable;

let mainImageUrl, thumbnailImageUrl;
try {
  if (files?.mainImage) {
    mainImageUrl = await uploadToCloudinary(files.mainImage[0].path);
    await fs.unlink(files.mainImage[0].path);
    menuItem.mainImage = mainImageUrl;
  }
  if (files?.thumbnailImage) {
    thumbnailImageUrl = await uploadToCloudinary(files.thumbnailImage[0].path);
    await fs.unlink(files.thumbnailImage[0].path);
    menuItem.thumbnailImage = thumbnailImageUrl;
  }
} catch (error) {
  const uploadError = new Error(`Cloudinary upload failed: ${error.message}`);
  uploadError.statusCode = 500;
  uploadError.isOperational = true;
  throw uploadError;
}

await menuItem.save();
return menuItem;
```

```

}

// delete menu item
async deleteMenuItem(menuItemId, userId) {
  const menuItem = await MenuItem.findById(menuItemId);
  if (!menuItem) {
    const error = new Error('Menu item not found');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }

  const restaurant = await Restaurant.findById(menuItem.restaurantId);
  if (restaurant.userId.toString() !== userId) {
    const error = new Error('Access denied: You do not own this restaurant');
    error.statusCode = 403;
    error.isOperational = true;
    throw error;
  }

  await menuItem.deleteOne();
  return { message: 'Menu item deleted successfully' };
}
}

```

restaurnt-service/src/services/restaurant.service.js

```

import Restaurant from './models/restaurant.model.js';
import { uploadToCloudinary } from './utils/cloudinary.js';
import fs from 'fs/promises';
import axios from 'axios';
import logger from './utils/logger.js';
import { emailClient } from "./email.client.js";

const EMAIL_SERVICE_URL = process.env.EMAIL_SERVICE_URL;

export class RestaurantService {

  // register resturent
  async registerRestaurant(data, files) {
    const {
      restaurantName, contactPerson, phoneNumber, businessType, cuisineType, operatingHours,
      deliveryRadius, taxId, streetAddress, city, state, zipCode, country, email, password, agreeTerms, availability
    } = data;

    // Check if restaurant email already exists
    logger.info('Checking for existing restaurant with email:', { email });
    const existingRestaurant = await Restaurant.findOne({ email });
    if (existingRestaurant) {
      const error = new Error('Email already registered');
      error.statusCode = 400;
      error.isOperational = true;
      throw error;
    }
  }
}

```

```

}

// Register user in auth-service
logger.info('Registering user in auth-service for email:', { email });
let userId;
try {
  const authResponse = await axios.post(`process.env.AUTH_SERVICE_URL}/api/auth/register`, {
    email,
    password,
    role: 'RESTAURANT',
    firstName: contactPerson,
    lastName: restaurantName,
    address: {
      street: streetAddress,
      city,
      state,
      zipCode,
      country
    }
  });
  if (authResponse.status !== 201) {
    throw new Error('Failed to create user in auth-service');
  }
  userId = authResponse.data.user.id;
  logger.info('User created in auth-service', { userId, email });

  logger.warn(
    'User created but not verified. To allow login, manually set isVerified: true in the auth-service database.',
    { userId, email }
  );
} catch (error) {
  const errorMessage = error.code === 'ECONNREFUSED'
    ? `Could not connect to auth-service at ${process.env.AUTH_SERVICE_URL}.`
    : error.response?.data?.message || error.message || 'Unknown error';

  logger.error('Error creating user in auth-service: ' + errorMessage, {
    status: error.response?.status,
    data: error.response?.data,
    stack: error.stack
  });

  const authError = new Error(`Failed to create user in auth-service: ${errorMessage}`);
  authError.statusCode = error.response?.status || 503;
  authError.isOperational = true;
  throw authError;
}

// Step 3: Upload files to Cloudinary
logger.info('Uploading files to Cloudinary', { files: Object.keys(files) });
let businessLicenseUrl, foodSafetyCertUrl, exteriorPhotoUrl, logoUrl;
try {

```

```

if (files?.businessLicense) {
  logger.debug('Uploading businessLicense to Cloudinary');
  businessLicenseUrl = await uploadToCloudinary(files.businessLicense[0].path);
  logger.debug('Deleting local businessLicense file');
  await fs.unlink(files.businessLicense[0].path);
}
if (files?.foodSafetyCert) {
  logger.debug('Uploading foodSafetyCert to Cloudinary');
  foodSafetyCertUrl = await uploadToCloudinary(files.foodSafetyCert[0].path);
  logger.debug('Deleting local foodSafetyCert file');
  await fs.unlink(files.foodSafetyCert[0].path);
}
if (files?.exteriorPhoto) {
  logger.debug('Uploading exteriorPhoto to Cloudinary');
  exteriorPhotoUrl = await uploadToCloudinary(files.exteriorPhoto[0].path);
  logger.debug('Deleting local exteriorPhoto file');
  await fs.unlink(files.exteriorPhoto[0].path);
}
if (files?.logo) {
  logger.debug('Uploading logo to Cloudinary');
  logoUrl = await uploadToCloudinary(files.logo[0].path);
  logger.debug('Deleting local logo file');
  await fs.unlink(files.logo[0].path);
}
} catch (error) {
  logger.error('Cloudinary upload or file cleanup failed:', {
    message: error.message,
    stack: error.stack,
    files: Object.keys(files || {})
  });
  const uploadError = new Error(`Cloudinary upload failed: ${error.message}`);
  uploadError.statusCode = 500;
  uploadError.isOperational = true;
  throw uploadError;
}

```

```

// Step 4: Create new restaurant
logger.info('Creating new restaurant in database');
const restaurant = new Restaurant({
  _id: userId,
  userId,
  restaurantName,
  contactPerson,
  phoneNumber,
  businessType,
  cuisineType,
  operatingHours,
  deliveryRadius,
  taxId,
  address: { streetAddress, city, state, zipCode, country },
  email,
  agreeTerms,
  businessLicense: businessLicenseUrl,

```

```

foodSafetyCert: foodSafetyCertUrl,
exteriorPhoto: exteriorPhotoUrl,
logo: logoUrl,
availability: availability !== undefined ? availability : true
});

try {
  await restaurant.save();
  logger.info('Restaurant saved successfully', { restaurantId: restaurant._id });
} catch (error) {
  logger.error('Failed to save restaurant to database:', {
    message: error.message,
    stack: error.stack,
    restaurantData: {
      userId,
      restaurantName,
      email
    }
  });
  const dbError = new Error(`Failed to save restaurant: ${error.message}`);
  dbError.statusCode = 500;
  dbError.isOperational = true;
  throw dbError;
}

// Step 5: Notify admin via notification-service
logger.info('Sending notification to admin');
try {
  await axios.post(`${process.env.NOTIFICATION_SERVICE_URL}/api/notifications/send`, {
    to: 'admin@example.com',
    type: 'email',
    subject: 'New Restaurant Registration',
    message: `A new restaurant (${restaurantName}) has registered and is awaiting your approval.`
  });
  logger.info('Notification sent successfully');
} catch (err) {
  logger.error('Notification error:', {
    message: err.message,
    stack: err.stack
  });
}

return { restaurant };
}

// get resturent by ID
async getRestaurantById(id) {
  const restaurant = await Restaurant.findById(id);
  if (!restaurant) {
    const error = new Error('Restaurant not found');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }
}

```

```

    return restaurant;
}
// Update restaurant status and send corresponding email
async updateRestaurantStatus(id, status) {
    // Find the restaurant by ID
    const restaurant = await Restaurant.findById(id);
    if (!restaurant) {
        const error = new Error('Restaurant not found');
        error.statusCode = 404;
        error.isOperational = true;
        throw error;
    }

    // Update the status
    restaurant.status = status;
    const updatedRestaurant = await restaurant.save();
    logger.info(`Restaurant status updated to ${status} for ID: ${id}`);
}

// Send email based on the new status
try {
    if (status === 'rejected') {
        await emailClient.sendRejectionEmail(restaurant.email);
        logger.info(`Rejection email sent to: ${restaurant.email}`);
    } else if (status === 'approved') {
        await emailClient.sendApprovedEmail(restaurant.email);
        logger.info(`Approval email sent to: ${restaurant.email}`);
    } else if (status === 'blocked') {
        await emailClient.sendBlockedEmail(restaurant.email);
        logger.info(`Approval email sent to: ${restaurant.email}`);
    } else {
        logger.warn(`No email configured for status: ${status}`);
    }
} catch (emailError) {
    // Log email failure but don't fail the status update
    logger.error(`Failed to send email for status ${status}: ${emailError.message}`, {
        email: restaurant.email,
        stack: emailError.stack,
    });
}

return updatedRestaurant;
}
// get all resturent
async getAllRestaurants(page = 1, limit = 10) {
    // Convert page and limit to integers and ensure they are positive
    page = Math.max(1, parseInt(page, 10));
    limit = Math.max(1, parseInt(limit, 10));

    // Calculate the number of documents to skip
    const skip = (page - 1) * limit;

    // Fetch paginated restaurants and total count
    const restaurantsPromise = Restaurant.find()

```

```

.skip(skip)
.limit(limit)
.exec();

const totalCountPromise = Restaurant.countDocuments().exec();

// Execute both queries in parallel
const [restaurants, totalCount] = await Promise.all([restaurantsPromise, totalCountPromise]);

return {
  restaurants,
  totalCount,
  currentPage: page,
  totalPages: Math.ceil(totalCount / limit),
  pageSize: limit,
};

}

// update resturent
async updateRestaurant(id, data, files) {
  const restaurant = await Restaurant.findById(id);
  if (!restaurant) {
    const error = new Error('Restaurant not found');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }

  const {
    restaurantName, contactPerson, phoneNumber, businessType, cuisineType, operatingHours,
    deliveryRadius, taxId, streetAddress, city, state, zipCode, country, email
  } = data;

  if (restaurantName) restaurant.restaurantName = restaurantName;
  if (contactPerson) restaurant.contactPerson = contactPerson;
  if (phoneNumber) restaurant.phoneNumber = phoneNumber;
  if (businessType) restaurant.businessType = businessType;
  if (cuisineType) restaurant.cuisineType = cuisineType;
  if (operatingHours) restaurant.operatingHours = operatingHours;
  if (deliveryRadius) restaurant.deliveryRadius = deliveryRadius;
  if (taxId) restaurant.taxId = taxId;
  if (streetAddress) restaurant.address.streetAddress = streetAddress;
  if (city) restaurant.address.city = city;
  if (state) restaurant.address.state = state;
  if (zipCode) restaurant.address.zipCode = zipCode;
  if (country) restaurant.address.country = country;
  if (email && email !== restaurant.email) {
    const existingRestaurant = await Restaurant.findOne({ email });
    if (existingRestaurant) {
      const error = new Error('Email already registered');
      error.statusCode = 400;
      error.isOperational = true;
      throw error;
    }
  }
}

```

```

    restaurant.email = email;
}

let businessLicenseUrl, foodSafetyCertUrl, exteriorPhotoUrl, logoUrl;
try {
  if (files?.businessLicense) {
    businessLicenseUrl = await uploadToCloudinary(files.businessLicense[0].path);
    await fs.unlink(files.businessLicense[0].path);
  }
  if (files?.foodSafetyCert) {
    foodSafetyCertUrl = await uploadToCloudinary(files.foodSafetyCert[0].path);
    await fs.unlink(files.foodSafetyCert[0].path);
  }
  if (files?.exteriorPhoto) {
    exteriorPhotoUrl = await uploadToCloudinary(files.exteriorPhoto[0].path);
    await fs.unlink(files.exteriorPhoto[0].path);
  }
  if (files?.logo) {
    logoUrl = await uploadToCloudinary(files.logo[0].path);
    await fs.unlink(files.logo[0].path);
  }
} catch (error) {
  const uploadError = new Error(`Cloudinary upload failed: ${error.message}`);
  uploadError.statusCode = 500;
  uploadError.isOperational = true;
  throw uploadError;
}

await restaurant.save();
return restaurant;
}
// delete resturent
async deleteRestaurant(id) {
  const restaurant = await Restaurant.findById(id);
  if (!restaurant) {
    const error = new Error('Restaurant not found');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }
  await restaurant.deleteOne();
  return { message: 'Restaurant deleted successfully' };
}
// get restaurant by userId
async getRestaurantByUserId(userId) {
  logger.info('Fetching restaurant by userId:', { userId });
  const restaurant = await Restaurant.findOne({ userId });
  if (!restaurant) {
    const error = new Error('Restaurant not found for this user');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }
}

```

```

    return restaurant;
}

// Update restaurant availability
async updateRestaurantAvailability(id, availability) {
  logger.info('Updating restaurant availability resturent:', { id, availability });
  logger.info('meka thama id eka', { id });

  const restaurant = await Restaurant.findById(id);
  if (!restaurant) {
    const error = new Error('Restaurant not found');
    error.statusCode = 404;
    error.isOperational = true;
    throw error;
  }

  restaurant.availability = availability;
  try {
    await restaurant.save();
    logger.info('Restaurant availability updated successfully', { restaurantId: restaurant._id });
  } catch (error) {
    logger.error('Failed to update restaurant availability:', {
      message: error.message,
      stack: error.stack
    });
    const dbError = new Error('Error updating restaurant availability');
    dbError.statusCode = 500;
    dbError.isOperational = true;
    throw dbError;
  }

  return { restaurant };
}
}

```

restaurnt-service/src/utils/cloudinary.js

```

import { v2 as cloudinary } from 'cloudinary';
import dotenv from 'dotenv';

dotenv.config();

// Configure Cloudinary
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET
});

// Function to upload file to Cloudinary
const uploadToCloudinary = async (filePath, folder = 'restaurants') => {
  try {

```

```
const result = await cloudinary.uploader.upload(filePath, {
  folder,
  resource_type: 'auto'
});
return result.secure_url;
} catch (error) {
  throw new Error(`Cloudinary upload failed: ${error.message}`);
}
};

export { uploadToCloudinary };
```

restaurnt-service/src/utils/logger.js

```
import winston from 'winston';

// Configure Winston logger
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' })
  ]
});

export default logger;
```

restaurnt-service/src/validation/category.validation.js

```
import { body, validationResult } from 'express-validator';

export const validateCategory = [
  // Name validation: required, string, trimmed, 3–50 characters
  body('name')
    .isString()
    .trim()
    .notEmpty()
    .withMessage('Category name is required')
    .isLength({ min: 3, max: 50 })
    .withMessage('Category name must be between 3 and 50 characters'),

  // Description validation: optional, string, trimmed, max 200 characters
  body('description')
    .optional({ checkFalsy: true })
    .isString()
    .trim()
```

```

.isLength({ max: 200 })
.withMessage('Category description must not exceed 200 characters'),

// Middleware to check validation results
(req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ message: errors.array()[0].msg });
  }
  next();
},
];

```

restaurnt-service/src/validation/menu.validation.js

```

import { body, validationResult } from 'express-validator';

export const validateMenuItem = [
  body('name').notEmpty().withMessage('Name is required'),
  body('description').notEmpty().withMessage('Description is required'),
  body('price').isFloat({ min: 0 }).withMessage('Price must be a positive number'),
  body('category').notEmpty().withMessage('Category is required'),
  body('isAvailable').optional().isBoolean().withMessage('isAvailable must be a boolean'),
  (req, res, next) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ message: 'Invalid data', errors: errors.array() });
    }
    next();
  }
];

```

restaurnt-service/src/validation/restaurant.Validation.js

```

import { body } from 'express-validator';

export const validateRestaurantRegistration = [
  body('restaurantName')
    .trim()
    .notEmpty()
    .withMessage('Restaurant name is required'),
  body('contactPerson')
    .trim()
    .notEmpty()
    .withMessage('Contact person is required'),
  body('phoneNumber')
    .trim()
    .matches(/^\+?[1-9]\d{1,14}$/)
    .withMessage('Please provide a valid phone number'),
  body('businessType')
    .trim()

```

```
.notEmpty()
.withMessage('Business type is required'),
body('cuisineType')
.trim()
.isNotEmpty()
.withMessage('Cuisine type is required'),
body('operatingHours')
.trim()
.isNotEmpty()
.withMessage('Operating hours are required'),
body('deliveryRadius')
.trim()
.isNotEmpty()
.withMessage('Delivery radius is required'),
body('taxId')
.trim()
.isNotEmpty()
.withMessage('Tax ID is required'),
body('streetAddress')
.trim()
.isNotEmpty()
.withMessage('Street address is required'),
body('city')
.trim()
.isNotEmpty()
.withMessage('City is required'),
body('state')
.trim()
.isNotEmpty()
.withMessage('State is required'),
body('zipCode')
.trim()
.isNotEmpty()
.withMessage('Zip code is required'),
body('country')
.trim()
.isNotEmpty()
.withMessage('Country is required'),
body('confirmPassword')
.custom((value, { req }) => value === req.body.password)
.withMessage('Passwords do not match'),
body('agreeTerms')
.equals('true')
.withMessage('You must agree to the terms and conditions'),
body('availability')
.optional()
.isBoolean()
.withMessage('Availability must be a boolean')
];
// Update restaurant availability
export const validateUpdateAvailability = [
  body('availability')
```

```
.isBoolean()
.withMessage('Availability must be a boolean')
];
```

restaurnt-service/src/index.js

```
import "dotenv/config";
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import morgan from "morgan";
import logger from "./utils/logger.js";
import restaurantRoutes from "./routes/restaurant.routes.js";
import menuRoutes from "./routes/menu.routes.js";
import categoryRoutes from "./routes/category.route.js";

const app = express();

// Middleware
app.use(
  cors({
    origin: "*",
    methods: ["GET", "POST", "PUT", "DELETE", "PATCH"],
    allowedHeaders: ["Content-Type", "Authorization"],
    credentials: true,
  })
);
app.use(express.json({ limit: "50mb" }));
app.use(express.urlencoded({ extended: true, limit: "50mb" }));
app.use(morgan("dev"));

// Routes
app.use("/api/restaurants", restaurantRoutes);
app.use("/api/restaurants", menuRoutes);
app.use("/api/restaurants", categoryRoutes);

// Error handling middleware
app.use((err, req, res, next) => {
  logger.error(err.stack);
  res.status(500).json({ message: "Something went wrong!" });
});

// Database connection
mongoose
  .connect(process.env.MONGODB_URI)
  .then(() => {
    logger.info("Connected to MongoDB", {
      timestamp: new Date().toISOString(),
    });
    app.listen(process.env.PORT, () => {
      logger.info(`Resturent service running on port ${process.env.PORT}`, {
        timestamp: new Date().toISOString(),
      });
    });
  })
);
```

```

    });
  });
}
).catch((err) => {
  logger.error("MongoDB connection error:", err);
  process.exit(1);
});

```

10.10 Dockerfile

```

FROM node:18-alpine

# Set working directory
WORKDIR /app

# Install dependencies first to leverage Docker cache
COPY package.json yarn.lock ./

# Set NODE_TLS_REJECT_UNAUTHORIZED to 0 for development
ENV NODE_TLS_REJECT_UNAUTHORIZED=0

# Install dependencies
RUN yarn install --frozen-lockfile

# Copy the rest of the application
COPY . .

# Expose the port
EXPOSE 3002

# Start the application
CMD ["yarn", "dev"]

```

10.11 docker-compose.yml

```

# version: "3.8"

services:
  #API Gateway
  api-gateway:
    build: ./api-gateway
    ports:
      - "3010:3010"
    volumes:
      - ./api-gateway:/app
      - /app/node_modules
    env_file:
      - ./env/api-gateway.env
    depends_on:
      - auth-service

```

```
- notification-service  
networks:  
  - food-ordering-network
```

```
# Auth Service  
auth-service:  
  build: ./auth-service  
  ports:  
    - "3002:3002"  
  volumes:  
    - ./auth-service:/app  
    - /app/node_modules  
  env_file:  
    - ./env/auth-service.env  
networks:  
  - food-ordering-network
```

```
# Customer Service  
# customer-service:  
#  build: ./customer-service  
#  ports:  
#    - "3002:3002"  
#  volumes:  
#    - ./customer-service:/app  
#    - /app/node_modules  
#  env_file:  
#    - ./env/customer-service.env  
#  depends_on:  
#    - auth-service
```

```
# Restaurant Service  
# restaurant-service:  
#  build: ./restaurant-service  
#  ports:  
#    - "3003:3003"  
#  volumes:  
#    - ./restaurant-service:/app  
#    - /app/node_modules  
#  env_file:  
#    - ./env/restaurant-service.env  
#  depends_on:  
#    - auth-service
```

```
# # Order Service  
# order-service:  
#  build: ./order-service  
#  ports:  
#    - "3004:3004"  
#  volumes:  
#    - ./order-service:/app  
#    - /app/node_modules  
#  env_file:  
#    - ./env/order-service.env
```

```
# depends_on:
#   - auth-service
#   - customer-service
#   - restaurant-service

# # Delivery Service
# delivery-service:
#   build: ./delivery-service
#   ports:
#     - "3005:3005"
#   volumes:
#     - ./delivery-service:/app
#     - /app/node_modules
#   env_file:
#     - ./env/delivery-service.env
# depends_on:
#   - auth-service
#   - order-service

# # Payment Service
payment-service:
  build: ./payment-service
  ports:
    - "3006:3006"
  volumes:
    - ./payment-service:/app
    - /app/node_modules
  env_file:
    - ./env/payment-service.env
  depends_on:
    - auth-service
    - order-service

# Notification Service
notification-service:
  build: ./notification-service
  ports:
    - "3003:3003"
  volumes:
    - ./notification-service:/app
    - /app/node_modules
  env_file:
    - ./env/notification-service.env
  depends_on:
    - auth-service
  networks:
    - food-ordering-network

networks:
  food-ordering-network:
    driver: bridge
    name: food-ordering-network
```

10.12 api-gateway-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
  namespace: food-ordering
spec:
  replicas: 2
  selector:
    matchLabels:
      app: api-gateway
  template:
    metadata:
      labels:
        app: api-gateway
    spec:
      containers:
        - name: api-gateway
          image: api-gateway:latest
          ports:
            - containerPort: 3010
      env:
        - name: AUTH_SERVICE_URL
          value: "http://auth-service:3000"
        - name: ORDER_SERVICE_URL
          value: "http://order-service:3000"
        - name: DELIVERY_SERVICE_URL
          value: "http://delivery-service:3000"
        - name: DRIVER_SERVICE_URL
          value: "http://driver-service:3000"
        - name: RESTAURANT_SERVICE_URL
          value: "http://restaurant-service:3000"
        - name: PAYMENT_SERVICE_URL
          value: "http://payment-service:3000"
        - name: CART_SERVICE_URL
          value: "http://cart-service:3000"
        - name: NOTIFICATION_SERVICE_URL
          value: "http://notification-service:3000"
      resources:
        requests:
          memory: "256Mi"
          cpu: "100m"
        limits:
          memory: "512Mi"
          cpu: "200m"
      livenessProbe:
        httpGet:
          path: /health
          port: 3010
        initialDelaySeconds: 30
        periodSeconds: 10
      readinessProbe:
```

```

httpGet:
  path: /health
  port: 3010
  initialDelaySeconds: 5
  periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: api-gateway
  namespace: food-ordering
spec:
  selector:
    app: api-gateway
  ports:
    - port: 3010
      targetPort: 3010
  type: LoadBalancer

```

10.13 auth -deployment.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-service
  namespace: food-ordering
spec:
  replicas: 2
  selector:
    matchLabels:
      app: auth-service
  template:
    metadata:
      labels:
        app: auth-service
  spec:
    containers:
      - name: auth-service
        image: auth-service:latest
        ports:
          - containerPort: 3000
    env:
      - name: MONGODB_URI
        valueFrom:
          secretKeyRef:
            name: mongodb-secret
            key: uri
      - name: JWT_SECRET
        valueFrom:
          secretKeyRef:
            name: jwt-secret
            key: secret

```

```
- name: JWT_EXPIRES_IN
  value: "1h"
- name: PASSWORD_RESET_EXPIRES_IN
  value: "1h"
resources:
requests:
  memory: "256Mi"
  cpu: "100m"
limits:
  memory: "512Mi"
  cpu: "200m"
livenessProbe:
  httpGet:
    path: /health
    port: 3000
  initialDelaySeconds: 30
  periodSeconds: 10
readinessProbe:
  httpGet:
    path: /health
    port: 3000
  initialDelaySeconds: 5
  periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: auth-service
  namespace: food-ordering
spec:
  selector:
    app: auth-service
  ports:
    - port: 3000
      targetPort: 3000
  type: ClusterIP
```

10.14 cart-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cart-service
  namespace: food-ordering
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cart-service
  template:
    metadata:
      labels:
        app: cart-service
    spec:
      containers:
        - name: cart-service
          image: cart-service:latest
          ports:
            - containerPort: 3000
      env:
        - name: MONGODB_URI
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: uri
      resources:
        requests:
          memory: "256Mi"
          cpu: "100m"
        limits:
          memory: "512Mi"
          cpu: "200m"
      livenessProbe:
        httpGet:
          path: /health
          port: 3000
        initialDelaySeconds: 30
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /health
          port: 3000
        initialDelaySeconds: 5
        periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: cart-service
  namespace: food-ordering
```

```
spec:  
  selector:  
    app: cart-service  
  ports:  
    - port: 3000  
      targetPort: 3000  
  type: ClusterIP
```

10.15 delivery-deployment.yaml

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: delivery-service  
  namespace: food-ordering  
spec:  
  replicas: 2  
  selector:  
    matchLabels:  
      app: delivery-service  
  template:  
    metadata:  
      labels:  
        app: delivery-service  
  spec:  
    containers:  
      - name: delivery-service  
        image: delivery-service:latest  
        ports:  
          - containerPort: 3000  
    env:  
      - name: MONGODB_URI  
        valueFrom:  
          secretKeyRef:  
            name: mongodb-secret  
            key: uri  
      - name: DRIVER_SERVICE_URL  
        value: "http://driver-service:3000"  
      - name: NOTIFICATION_SERVICE_URL  
        value: "http://notification-service:3000"  
    resources:  
      requests:  
        memory: "256Mi"  
        cpu: "100m"  
      limits:  
        memory: "512Mi"  
        cpu: "200m"  
    livenessProbe:  
      httpGet:  
        path: /health  
        port: 3000
```

```

initialDelaySeconds: 30
periodSeconds: 10
readinessProbe:
  httpGet:
    path: /health
    port: 3000
  initialDelaySeconds: 5
  periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: delivery-service
  namespace: food-ordering
spec:
  selector:
    app: delivery-service
  ports:
    - port: 3000
      targetPort: 3000
  type: ClusterIP

```

10.16 driver-deployment.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: driver-service
  namespace: food-ordering
spec:
  replicas: 2
  selector:
    matchLabels:
      app: driver-service
  template:
    metadata:
      labels:
        app: driver-service
    spec:
      containers:
        - name: driver-service
          image: driver-service:latest
          ports:
            - containerPort: 3000
          env:
            - name: MONGODB_URI
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret
                  key: uri
            - name: DELIVERY_SERVICE_URL

```

```

    value: "http://delivery-service:3000"
  - name: NOTIFICATION_SERVICE_URL
    value: "http://notification-service:3000"
resources:
  requests:
    memory: "256Mi"
    cpu: "100m"
  limits:
    memory: "512Mi"
    cpu: "200m"
livenessProbe:
  httpGet:
    path: /health
    port: 3000
  initialDelaySeconds: 30
  periodSeconds: 10
readinessProbe:
  httpGet:
    path: /health
    port: 3000
  initialDelaySeconds: 5
  periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: driver-service
  namespace: food-ordering
spec:
  selector:
    app: driver-service
  ports:
    - port: 3000
      targetPort: 3000
  type: ClusterIP

```

10.17 mongodb-deployment.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb
  namespace: food-ordering
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:

```

```
app: mongodb
spec:
  containers:
    - name: mongodb
      image: mongo:latest
      ports:
        - containerPort: 27017
      resources:
        requests:
          memory: "256Mi"
          cpu: "100m"
        limits:
          memory: "512Mi"
          cpu: "200m"
      volumeMounts:
        - name: mongodb-data
          mountPath: /data/db
    volumes:
      - name: mongodb-data
        persistentVolumeClaim:
          claimName: mongodb-pvc
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: mongodb
  namespace: food-ordering
spec:
  selector:
    app: mongodb
  ports:
    - port: 27017
      targetPort: 27017
  type: ClusterIP
```

10.18 namespace-deployment.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: food-ordering
  labels:
    name: food-ordering
```

10.19 notification-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: notification-service
  namespace: food-ordering
spec:
  replicas: 2
  selector:
    matchLabels:
      app: notification-service
  template:
    metadata:
      labels:
        app: notification-service
    spec:
      containers:
        - name: notification-service
          image: notification-service:latest
          ports:
            - containerPort: 3000
      env:
        - name: MONGODB_URI
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: uri
        - name: SENDGRID_API_KEY
          valueFrom:
            secretKeyRef:
              name: sendgrid-secret
              key: api_key
        - name: TWILIO_ACCOUNT_SID
          valueFrom:
            secretKeyRef:
              name: twilio-secret
              key: account_sid
        - name: TWILIO_AUTH_TOKEN
          valueFrom:
            secretKeyRef:
              name: twilio-secret
              key: auth_token
        - name: WHATSAPP_BUSINESS_ID
          valueFrom:
            secretKeyRef:
              name: whatsapp-secret
              key: business_id
      resources:
        requests:
          memory: "256Mi"
          cpu: "100m"
      limits:
```

```
    memory: "512Mi"
    cpu: "200m"
  livenessProbe:
    httpGet:
      path: /health
      port: 3000
    initialDelaySeconds: 30
    periodSeconds: 10
  readinessProbe:
    httpGet:
      path: /health
      port: 3000
    initialDelaySeconds: 5
    periodSeconds: 5
---
```

```
apiVersion: v1
kind: Service
metadata:
  name: notification-service
  namespace: food-ordering
spec:
  selector:
    app: notification-service
  ports:
    - port: 3000
      targetPort: 3000
  type: ClusterIP
```

10.20 order-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-service
  namespace: food-ordering
spec:
  replicas: 2
  selector:
    matchLabels:
      app: order-service
  template:
    metadata:
      labels:
        app: order-service
    spec:
      containers:
        - name: order-service
          image: order-service:latest
          ports:
            - containerPort: 3000
      env:
```

```
- name: MONGODB_URI
  valueFrom:
    secretKeyRef:
      name: mongodb-secret
      key: uri
- name: DELIVERY_SERVICE_URL
  value: "http://delivery-service:3000"
- name: NOTIFICATION_SERVICE_URL
  value: "http://notification-service:3000"
resources:
requests:
  memory: "256Mi"
  cpu: "100m"
limits:
  memory: "512Mi"
  cpu: "200m"
livenessProbe:
  httpGet:
    path: /health
    port: 3000
  initialDelaySeconds: 30
  periodSeconds: 10
readinessProbe:
  httpGet:
    path: /health
    port: 3000
  initialDelaySeconds: 5
  periodSeconds: 5
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: order-service
  namespace: food-ordering
spec:
  selector:
    app: order-service
  ports:
    - port: 3000
      targetPort: 3000
  type: ClusterIP
```

10.21 payment-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: payment-service
  namespace: food-ordering
spec:
  replicas: 2
  selector:
    matchLabels:
      app: payment-service
  template:
    metadata:
      labels:
        app: payment-service
    spec:
      containers:
        - name: payment-service
          image: payment-service:latest
          ports:
            - containerPort: 3000
      env:
        - name: MONGODB_URI
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: uri
        - name: RABBITMQ_URL
          valueFrom:
            secretKeyRef:
              name: rabbitmq-secret
              key: url
        - name: PAYHERE_MERCHANT_ID
          valueFrom:
            secretKeyRef:
              name: payhere-secret
              key: merchant_id
        - name: PAYHERE_SECRET
          valueFrom:
            secretKeyRef:
              name: payhere-secret
              key: secret
        - name: PAYHERE_MODE
          value: "sandbox"
      resources:
        requests:
          memory: "256Mi"
          cpu: "100m"
        limits:
          memory: "512Mi"
          cpu: "200m"
      livenessProbe:
```

```

httpGet:
  path: /health
  port: 3000
  initialDelaySeconds: 30
  periodSeconds: 10
readinessProbe:
  httpGet:
    path: /health
    port: 3000
    initialDelaySeconds: 5
    periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: payment-service
  namespace: food-ordering
spec:
  selector:
    app: payment-service
  ports:
    - port: 3000
      targetPort: 3000
  type: ClusterIP

```

10.22 persistent-volume-claims.yaml

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
  namespace: food-ordering
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: restaurant-uploads-pvc
  namespace: food-ordering
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi

```

10.23 restaurant-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: restaurant-service
  namespace: food-ordering
spec:
  replicas: 2
  selector:
    matchLabels:
      app: restaurant-service
  template:
    metadata:
      labels:
        app: restaurant-service
    spec:
      containers:
        - name: restaurant-service
          image: restaurant-service:latest
          ports:
            - containerPort: 3000
      env:
        - name: MONGODB_URI
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: uri
        - name: UPLOAD_PATH
          value: "/uploads"
        - name: MAX_FILE_SIZE
          value: "5MB"
      resources:
        requests:
          memory: "256Mi"
          cpu: "100m"
        limits:
          memory: "512Mi"
          cpu: "200m"
      volumeMounts:
        - name: uploads
          mountPath: /uploads
      livenessProbe:
        httpGet:
          path: /health
          port: 3000
        initialDelaySeconds: 30
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /health
          port: 3000
        initialDelaySeconds: 5
```

```

    periodSeconds: 5
  volumes:
    - name: uploads
      persistentVolumeClaim:
        claimName: restaurant-uploads-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: restaurant-service
  namespace: food-ordering
spec:
  selector:
    app: restaurant-service
  ports:
    - port: 3000
      targetPort: 3000
  type: ClusterIP

```

10.24 secrets.yaml

```

apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
  namespace: food-ordering
type: Opaque
data:
  uri: bW9uZ29kYjovL21vbmdvZGI6MjcwMTcvZm9vZF9vcmRlcmluZw==
---
apiVersion: v1
kind: Secret
metadata:
  name: rabbitmq-secret
  namespace: food-ordering
type: Opaque
data:
  url: YW1xcDovL3JhYmJpdG1xOjU2NzI=
  username: YWRtaW4=
  password: YWRtaW4=
---
apiVersion: v1
kind: Secret
metadata:
  name: jwt-secret
  namespace: food-ordering
type: Opaque
data:
  secret: c3VwZXJfc2VjcmV0X2tleQ==
---
apiVersion: v1

```

```
kind: Secret
metadata:
  name: payhere-secret
  namespace: food-ordering
type: Opaque
data:
  merchant_id: eW91cl9tZXJjaGFudF9pZA==
  secret: eW91cl9wYXloZXJlX3NlY3JldA==

---
apiVersion: v1
kind: Secret
metadata:
  name: sendgrid-secret
  namespace: food-ordering
type: Opaque
data:
  api_key: eW91cl9zZW5kZ3JpZF9hcGlfa2V5

---
apiVersion: v1
kind: Secret
metadata:
  name: twilio-secret
  namespace: food-ordering
type: Opaque
data:
  account_sid: eW91cl90d2lsaW9fYWNjb3VudF9zaWQ=
  auth_token: eW91cl90d2lsaW9fYXV0aF90b2tlbg==

---
apiVersion: v1
kind: Secret
metadata:
  name: whatsapp-secret
  namespace: food-ordering
type: Opaque
data:
  business_id: eW91cl93aGF0c2FwcF9idXNpbmVzc19pZA==
```

10.12 Frontend

src/components/admin/AdminAllUsersTable.tsx

```
import React, { useState } from "react";

export interface User {
  _id: string;
  email: string;
  firstName: string;
  lastName: string;
  role: "CUSTOMER" | "RESTAURANT" | "DELIVERY" | "ADMIN";
  isActive: boolean;
  isVerified: boolean;
  address: {
    street: string;
    city: string;
    state: string;
    zipCode: string;
    country: string;
  };
  createdAt: string;
  updatedAt: string;
}

interface AdminAllUserTableProps {
  headers: string[];
  data: User[];
}

const AdminAllUserTable: React.FC<AdminAllUserTableProps> = ({ headers, data }) => {
  const [users] = useState<User[]>(data);
  const [searchTerm, setSearchTerm] = useState<string>("");
  const [roleFilter, setRoleFilter] = useState<string>("all");

  // Map headers to data fields
  const getFieldValue = (user: User, header: string) => {
    switch (header.toLowerCase()) {
      case "email":
        return user.email;
      case "first name":
        return user.firstName;
      case "last name":
        return user.lastName;
      case "role":
        return user.role;
      case "active status":
        return (
          <span className={getStatusStyles(user.isActive)}>
            {user.isActive ? "Active" : "Inactive"}
          </span>
        );
      case "verification status":
```

```

return (
  <span className={getStatusStyles(user.isVerified)}>
    {user.isVerified ? "Verified" : "Unverified"}
  </span>
);
case "street":
  return user.address.street;
case "city":
  return user.address.city;
case "state":
  return user.address.state;
case "zip code":
  return user.address.zipCode;
case "country":
  return user.address.country;
case "created at":
  return new Date(user.createdAt).toLocaleDateString();
case "updated at":
  return new Date(user.updatedAt).toLocaleDateString();
default:
  return "";
}
};

// Filter users by search term and role
const filteredUsers = users.filter((user) => {
  const matchesSearch = [
    user.email,
    user.firstName,
    user.lastName,
    user.role,
    user.address.street,
    user.address.city,
    user.address.state,
    user.address.country,
  ].some((value) => value.toLowerCase().includes(searchTerm.toLowerCase()));

  const matchesRole = roleFilter === "all" || user.role.toLowerCase() === roleFilter.toLowerCase();

  return matchesSearch && matchesRole;
});

// Sort users by email
const sortedUsers = [...filteredUsers].sort((a, b) => {
  const emailA = a.email.toLowerCase();
  const emailB = b.email.toLowerCase();
  return emailA.localeCompare(emailB);
});

// Get status color styles
const getStatusStyles = (status: boolean) => {
  const baseStyles = "px-2 py-1 text-xs font-semibold rounded-lg";
  return status
};

```

```

? `${baseStyles} bg-green-100 text-green-600 dark:bg-green-700 dark:text-green-200`  

: `${baseStyles} bg-red-100 text-red-600 dark:bg-red-700 dark:text-red-200`;  

};  
  

return (  

  <>  

  /* Search and Filter Controls */  

<div className="p-5 flex flex-col sm:flex-row gap-4 justify-between items-center">  

  <input  

    type="text"  

    placeholder="Search users..."  

    value={searchTerm}  

    onChange={(e) => setSearchTerm(e.target.value)}  

    className="w-full sm:w-64 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white  

    dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"  

  />  

  <select  

    value={roleFilter}  

    onChange={(e) => setRoleFilter(e.target.value)}  

    className="w-full sm:w-40 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white  

    dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"  

>  

  <option value="all">All Roles</option>  

  <option value="customer">Customer</option>  

  <option value="restaurant">Restaurant</option>  

  <option value="delivery">Delivery</option>  

  <option value="admin">Admin</option>  

</select>  

</div>  
  

/* Desktop Table View with Horizontal Scroll */  

<div className="hidden lg:block p-5 w-full text-sm rounded-md overflow-x-auto bg-white dark:bg-gray-  

700">  

  <table className="min-w-[1200px] w-full">  

    <thead className="bg-gray-200 border-gray-200 dark:bg-gray-700 dark:border-gray-500">  

      <tr>  

        {headers.map((header, index) => (  

          <th  

            key={index}  

            className="text-start py-4 px-6 font-semibold text-gray-600 dark:text-gray-200"  

          >  

            {header}  

          </th>  

        ))}  

      </tr>  

    </thead>  

    <tbody>  

      {sortedUsers.map((user, index) => (  

        <tr  

          key={user._id}  

          className={`border-b border-gray-200 dark:border-gray-500 ${  

            index % 2 === 0 ? "bg-gray-50 dark:bg-gray-600" : "bg-gray-100 dark:bg-gray-700"  

          } hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors`}  


```

```

    >
      {headers.map((header, idx) => (
        <td key={idx} className="py-4 px-6 text-gray-800 dark:text-white">
          {getFieldValue(user, header)}
        </td>
      )))
    </tr>
  )));
</tbody>
</table>
</div>
/* Mobile Grid View */
<div className="lg:hidden p-5 space-y-4 overflow-y-scroll h-full">
  {sortedUsers.map((user) => (
    <div
      key={user._id}
      className="border rounded-md p-4 shadow-md hover:shadow-lg transition-shadow border-gray-200 dark:border-gray-500 bg-gray-100 dark:bg-gray-600"
    >
      <div className="space-y-2">
        {headers.map((header, idx) => (
          <div key={idx} className="flex justify-between items-center py-1">
            <span className="font-medium text-gray-800 dark:text-white">{header}</span>
            <span className="text-sm text-gray-600 dark:text-gray-300">
              {getFieldValue(user, header)}
            </span>
          </div>
        )));
      </div>
    </div>
  )));
</div>
/* No Results Message */
{sortedUsers.length === 0 && (
  <div className="p-5 text-center text-gray-600 dark:text-gray-300">
    No users found.
  </div>
)
</>
);
};
};

export default AdminAllUserTable;

```

src/components/admin/AdminResturentRequestTable.tsx

```

import React, { useState } from "react";
import { updateRestaurantStatus } from "../../utils/api";
import ConfirmationModal from "../UI/ConfirmationModal";

export interface Restaurant {
  _id: string;
  restaurantName: string;
  contactPerson: string;

```

```

phoneNumber: string;
businessType: string;
cuisineType: string;
operatingHours: string;
deliveryRadius: string;
taxId: string;
address: {
  streetAddress: string;
  city: string;
  state: string;
  zipCode: string;
  country: string;
};
email: string;
businessLicense?: string | null;
foodSafetyCert?: string | null;
exteriorPhoto?: string | null;
logo?: string | null;
status?: string;
}
}

interface AdminResturentTableProps {
  headers: string[];
  data: Restaurant[];
}

```

```

const AdminResturentRequestTable: React.FC<AdminResturentTableProps> = ({ headers, data }) => {
  const [restaurants, setRestaurants] = useState<Restaurant[]>(
    data.map((restaurant) => ({
      ...restaurant,
      status: restaurant.status ? restaurant.status.trim().toLowerCase() : "pending",
    }))
  );
  const [searchTerm, setSearchTerm] = useState<string>("");
  const [sortOrder, setSortOrder] = useState<"asc" | "desc">("asc");
  const [error, setError] = useState<string | null>(null);
  const [modal, setModal] = useState<{
    isOpen: boolean;
    action: "approve" | "reject" | null;
    restaurantId: string | null;
    restaurantName: string | null;
  }>({
    isOpen: false,
    action: null,
    restaurantId: null,
    restaurantName: null,
  });
  // Handle approve action
  const handleApprove = async (restaurantId: string, restaurantName: string) => {
    setModal({
      isOpen: true,
      action: "approve",
    });
  };
}

```

```

restaurantId,
restaurantName,
});
};

// Handle reject action
const handleReject = async (restaurantId: string, restaurantName: string) => {
setModal({
isOpen: true,
action: "reject",
restaurantId,
restaurantName,
});
};

// Confirm modal action
const handleConfirm = async () => {
if (!modal.restaurantId || !modal.restaurantName || !modal.action) return;

try {
const status = modal.action === "approve" ? "approved" : "rejected";
await updateRestaurantStatus(modal.restaurantId, status);
setRestaurants((prev) =>
prev.map((restaurant) =>
restaurant._id === modal.restaurantId
? { ...restaurant, status: status.charAt(0).toUpperCase() + status.slice(1) }
: restaurant
)
);
setError(null);
} catch (err: any) {
setError(`Failed to ${modal.action} ${modal.restaurantName}: ${err.message}`);
} finally {
setModal({ isOpen: false, action: null, restaurantId: null, restaurantName: null });
}
};

// Close modal
const handleCloseModal = () => {
setModal({ isOpen: false, action: null, restaurantId: null, restaurantName: null });
};

// Filter restaurants to only show Pending status
const filteredRestaurants = restaurants.filter(
(restaurant) => restaurant.status?.toLowerCase() === "pending"
);

// Apply search filter
const searchedRestaurants = filteredRestaurants.filter((restaurant) =>
[
restaurant.restaurantName,
restaurant.contactPerson,
restaurant.email,
]
);

```

```

    restaurant.cuisineType,
  ].some((value) => value.toLowerCase().includes(searchTerm.toLowerCase())))
);

// Sort restaurants by restaurantName
const sortedRestaurants = [...searchedRestaurants].sort((a, b) => {
  const nameA = a.restaurantName.toLowerCase();
  const nameB = b.restaurantName.toLowerCase();
  return sortOrder === "asc" ? nameA.localeCompare(nameB) : nameB.localeCompare(nameA);
});

// Toggle sort order
const toggleSortOrder = () => {
  setSortOrder((prev) => (prev === "asc" ? "desc" : "asc"));
};

// Get status color styles
const getStatusStyles = (status: string) => {
  const baseStyles = "px-2 py-1 text-xs font-semibold rounded-lg";
  switch (status.toLowerCase()) {
    case "approved":
      return `${baseStyles} bg-green-100 text-green-600 dark:bg-green-700 dark:text-green-200`;
    case "rejected":
      return `${baseStyles} bg-red-100 text-red-600 dark:bg-red-700 dark:text-red-200`;
    case "pending":
    default:
      return `${baseStyles} bg-yellow-100 text-yellow-600 dark:bg-yellow-700 dark:text-yellow-200`;
  }
};

return (
  <>
  {/* Search and Sort Controls */}
  <div className="p-5 flex flex-col sm:flex-row gap-4 justify-between items-center">
    <input
      type="text"
      placeholder="Search restaurants..."
      value={searchTerm}
      onChange={(e) => setSearchTerm(e.target.value)}
      className="w-full sm:w-64 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
    />
    <button
      onClick={toggleSortOrder}
      className="w-full sm:w-auto px-4 py-2 bg-gray-200 border-gray-200 dark:bg-gray-800 dark:border-gray-500 rounded-lg transition-colors"
    >
      Sort by Name {sortOrder === "asc" ? "↑" : "↓"}
    </button>
  </div>

  {/* Error Message */}
  {error && <p className="p-5 text-center text-red-600">{error}</p>}

```

```

/* Confirmation Modal */
<ConfirmationModal
  isOpen={modal.isOpen}
  onClose={handleCloseModal}
  onConfirm={handleConfirm}
  title={modal.action === "approve" ? "Approve Restaurant" : "Reject Restaurant"}
  message={`Are you sure you want to ${modal.action} ${modal.restaurantName}?`}
  confirmText={modal.action === "approve" ? "Approve" : "Reject"}
  cancelText="Cancel"
/>

/* Desktop Table View with Horizontal Scroll */
<div className="hidden lg:block p-5 w-full text-sm overflow-x-auto bg-white dark:bg-gray-700">
  <table className="min-w-[1200px] w-full">
    <thead className="bg-gray-200 border-gray-200 dark:bg-gray-700 dark:border-gray-500">
      <tr>
        {headers.map((header, index) => (
          <th
            key={index}
            className="text-start py-4 px-6 font-semibold text-gray-600 dark:text-gray-200"
          >
            {header}
          </th>
        ))}
      </tr>
    </thead>
    <tbody>
      {sortedRestaurants.map((restaurant, index) => (
        <tr
          key={restaurant._id}
          className={`${"border-b border-gray-200 dark:border-gray-500 ${
            index % 2 === 0
              ? "bg-gray-50 dark:bg-gray-600"
              : "bg-gray-100 dark:bg-gray-700"
            } hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors`}`}
        >
          <td className="py-4 px-6 text-gray-800 dark:text-white">
            {restaurant.restaurantName}
          </td>
          <td className="py-4 px-6 text-gray-800 dark:text-white">
            {restaurant.contactPerson}
          </td>
          <td className="py-4 px-6 text-gray-800 dark:text-white">
            {restaurant.phoneNumber}
          </td>
          <td className="py-4 px-6 text-gray-800 dark:text-white">
            {restaurant.businessType}
          </td>
          <td className="py-4 px-6 text-gray-800 dark:text-white">
            {restaurant.cuisineType}
          </td>
          <td className="py-4 px-6 text-gray-800 dark:text-white">
            {restaurant.address}
          </td>
        </tr>
      ))}
    </tbody>
  </table>
</div>

```

```
{restaurant.operatingHours}  
</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">  
  {restaurant.deliveryRadius}  
</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">{restaurant.taxId}</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">  
  {restaurant.address.streetAddress}  
</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">{restaurant.address.city}</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">{restaurant.address.state}</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">  
  {restaurant.address.zipCode}  
</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">  
  {restaurant.address.country}  
</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">{restaurant.email}</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">  
  {restaurant.businessLicense ? (  
    <a  
      href={restaurant.businessLicense}  
      target="_blank"  
      rel="noopener noreferrer"  
      className="text-blue-500 underline"  
    >  
      License  
    </a>  
  ) : (  
    "N/A"  
  )}  
</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">  
  {restaurant.foodSafetyCert ? (  
    <a  
      href={restaurant.foodSafetyCert}  
      target="_blank"  
      rel="noopener noreferrer"  
      className="text-blue-500 underline"  
    >  
      Cert  
    </a>  
  ) : (  
    "N/A"  
  )}  
</td>  
<td className="py-4 px-6 text-gray-800 dark:text-white">  
  {restaurant.exteriorPhoto ? (  
    <a  
      href={restaurant.exteriorPhoto}  
      target="_blank"  
      rel="noopener noreferrer"  
      className="text-blue-500 underline"  
    >
```

```

>
  Photo
</a>
):(
  "N/A"
)}
</td>
<td className="py-4 px-6 text-gray-800 dark:text-white">
{restaurant.logo ? (
  <a
    href={restaurant.logo}
    target="_blank"
    rel="noopener noreferrer"
    className="text-blue-500 underline"
  >
    Logo
  </a>
):(
  "N/A"
)}
</td>
<td className="py-4 px-6 text-gray-800 dark:text-white">
<span className={getStatusStyles(restaurant.status || "Pending")}>
  {restaurant.status || "Pending"}
</span>
</td>
<td className="py-4 px-6 flex space-x-2">
<button
  onClick={() => handleApprove(restaurant._id, restaurant.restaurantName)}
  disabled={restaurant.status?.toLowerCase() !== "pending"}
  className={`px-3 py-1 rounded-lg text-white transition-colors ${{
    restaurant.status?.toLowerCase() !== "pending"
      ? "bg-gray-400 cursor-not-allowed"
      : "bg-green-500 hover:bg-green-600"
  }}`}
>
  Approve
</button>
<button
  onClick={() => handleReject(restaurant._id, restaurant.restaurantName)}
  disabled={restaurant.status?.toLowerCase() !== "pending"}
  className={`px-3 py-1 rounded-lg text-white transition-colors ${{
    restaurant.status?.toLowerCase() !== "pending"
      ? "bg-gray-400 cursor-not-allowed"
      : "bg-red-500 hover:bg-red-600"
  }}`}
>
  Reject
</button>
</td>
</tr>
))}
</tbody>

```

```

</table>
</div>

{/* Mobile Grid View */}
<div className="lg:hidden p-5 space-y-4 overflow-y-scroll h-full">
  {sortedRestaurants.map((restaurant) => (
    <div
      key={restaurant._id}
      className="border rounded-md p-4 bg-white shadow-md hover:shadow-lg transition-shadow border-gray-200 dark:bg-gray-600 dark:border-gray-500"
    >
      <div className="space-y-2">
        <div className="flex justify-between items-center py-1">
          <span className="font-medium text-gray-800 dark:text-white">Restaurant Name:</span>
          <span className="text-sm text-gray-600 dark:text-gray-300">
            {restaurant.restaurantName}
          </span>
        </div>
        <div className="flex justify-between items-center py-1">
          <span className="font-medium text-gray-800 dark:text-white">Contact Person:</span>
          <span className="text-sm text-gray-600 dark:text-gray-300">
            {restaurant.contactPerson}
          </span>
        </div>
        <div className="flex justify-between items-center py-1">
          <span className="font-medium text-gray-800 dark:text-white">Phone Number:</span>
          <span className="text-sm text-gray-600 dark:text-gray-300">
            {restaurant.phoneNumber}
          </span>
        </div>
        <div className="flex justify-between items-center py-1">
          <span className="font-medium text-gray-800 dark:text-white">Business Type:</span>
          <span className="text-sm text-gray-600 dark:text-gray-300">
            {restaurant.businessType}
          </span>
        </div>
        <div className="flex justify-between items-center py-1">
          <span className="font-medium text-gray-800 dark:text-white">Cuisine Type:</span>
          <span className="text-sm text-gray-600 dark:text-gray-300">
            {restaurant.cuisineType}
          </span>
        </div>
        <div className="flex justify-between items-center py-1">
          <span className="font-medium text-gray-800 dark:text-white">Operating Hours:</span>
          <span className="text-sm text-gray-600 dark:text-gray-300">
            {restaurant.operatingHours}
          </span>
        </div>
        <div className="flex justify-between items-center py-1">
          <span className="font-medium text-gray-800 dark:text-white">Delivery Radius:</span>
          <span className="text-sm text-gray-600 dark:text-gray-300">
            {restaurant.deliveryRadius}
          </span>
        </div>
      </div>
    </div>
  ))
</div>

```

```
</div>
<div className="flex justify-between items-center py-1">
  <span className="font-medium text-gray-800 dark:text-white">Tax ID:</span>
  <span className="text-sm text-gray-600 dark:text-gray-300">{ restaurant.taxId }</span>
</div>
<div className="flex justify-between items-center py-1">
  <span className="font-medium text-gray-800 dark:text-white">Street Address:</span>
  <span className="text-sm text-gray-600 dark:text-gray-300">
    { restaurant.address.streetAddress }
  </span>
</div>
<div className="flex justify-between items-center py-1">
  <span className="font-medium text-gray-800 dark:text-white">City:</span>
  <span className="text-sm text-gray-600 dark:text-gray-300">{ restaurant.address.city }</span>
</div>
<div className="flex justify-between items-center py-1">
  <span className="font-medium text-gray-800 dark:text-white">State:</span>
  <span className="text-sm text-gray-600 dark:text-gray-300">{ restaurant.address.state }</span>
</div>
<div className="flex justify-between items-center py-1">
  <span className="font-medium text-gray-800 dark:text-white">Zip Code:</span>
  <span className="text-sm text-gray-600 dark:text-gray-300">
    { restaurant.address.zipCode }
  </span>
</div>
<div className="flex justify-between items-center py-1">
  <span className="font-medium text-gray-800 dark:text-white">Country:</span>
  <span className="text-sm text-gray-600 dark:text-gray-300">
    { restaurant.address.country }
  </span>
</div>
<div className="flex justify-between items-center py-1">
  <span className="font-medium text-gray-800 dark:text-white">Email:</span>
  <span className="text-sm text-gray-600 dark:text-gray-300">{ restaurant.email }</span>
</div>
<div className="flex justify-between items-center py-1">
  <span className="font-medium text-gray-800 dark:text-white">Business License:</span>
  <span className="text-sm text-gray-600 dark:text-gray-300">
    { restaurant.businessLicense ? (
      <a
        href={ restaurant.businessLicense }
        target="_blank"
        rel="noopener noreferrer"
        className="text-blue-500 underline"
      >
        License
      </a>
    ) : ( "N/A"
    )}
  </span>
</div>
<div className="flex justify-between items-center py-1">
  <span className="font-medium text-gray-800 dark:text-white">
```

Food Safety Cert:

 {restaurant.foodSafetyCert ? (
 <a
 href={restaurant.foodSafetyCert}
 target="_blank"
 rel="noopener noreferrer"
 className="text-blue-500 underline"
 >
 Cert

) : (
 "N/A"
)

</div>
<div className="flex justify-between items-center py-1">
 Exterior Photo:

 {restaurant.exteriorPhoto ? (
 <a
 href={restaurant.exteriorPhoto}
 target="_blank"
 rel="noopener noreferrer"
 className="text-blue-500 underline"
 >
 Photo

) : (
 "N/A"
)

</div>
<div className="flex justify-between items-center py-1">
 Logo:

 {restaurant.logo ? (
 <a
 href={restaurant.logo}
 target="_blank"
 rel="noopener noreferrer"
 className="text-blue-500 underline"
 > Logo

) : (
 "N/A"
)

</div>
<div className="flex justify-between items-center py-1">
 Status:


```

        {restaurant.status || "Pending"}
      </span>
    </div>
  </div>
  {/* Actions */}
  <div className="flex space-x-2 mt-4">
    <button
      onClick={() => handleApprove(restaurant._id, restaurant.restaurantName)}
      disabled={restaurant.status?.toLowerCase() !== "pending"}
      className={`px-3 py-1 rounded-lg text-white transition-colors ${{
        restaurant.status?.toLowerCase() !== "pending"
        ? "bg-gray-400 cursor-not-allowed"
        : "bg-green-500 hover:bg-green-600"
      }}`}
    >
      Approve
    </button>
    <button
      onClick={() => handleReject(restaurant._id, restaurant.restaurantName)}
      disabled={restaurant.status?.toLowerCase() !== "pending"}
      className={`px-3 py-1 rounded-lg text-white transition-colors ${{
        restaurant.status?.toLowerCase() !== "pending"
        ? "bg-gray-400 cursor-not-allowed"
        : "bg-red-500 hover:bg-red-600"
      }}`}
    >
      Reject
    </button>
  </div>
</div>
))}

</div>
 {/* No Results Message */}
{sortedRestaurants.length === 0 && (
  <div className="p-5 text-center text-gray-600 dark:text-gray-300">
    No pending restaurants found.
  </div>
)
</>
);
};

export default AdminResturentRequestTable;

```

src/components/admin/AdminResturentTable.tsx

```

import React, { useState } from "react";
import { deleteRestaurant, updateRestaurantStatus } from "../../utils/api";
import ConfirmationModal from "../UI/ConfirmationModal";

export interface Restaurant {
  _id: string;
  restaurantName: string;

```

```

contactPerson: string;
phoneNumber: string;
businessType: string;
cuisineType: string;
operatingHours: string;
deliveryRadius: string;
taxId: string;
address: {
  streetAddress: string;
  city: string;
  state: string;
  zipCode: string;
  country: string;
};
email: string;
businessLicense?: string | null;
foodSafetyCert?: string | null;
exteriorPhoto?: string | null;
logo?: string | null;
status?: string;
}

interface AdminResturentTableProps {
  headers: string[];
  data: Restaurant[];
  onDelete: (restaurantId: string) => void;
}

const AdminResturentTable: React.FC<AdminResturentTableProps> = ({ headers, data, onDelete }) => {
  const [restaurants, setRestaurants] = useState<Restaurant[]>(
    data.map((restaurant) => ({
      ...restaurant,
      status: restaurant.status ? restaurant.status.trim().toLowerCase() : "pending",
    }))
  );
  const [searchTerm, setSearchTerm] = useState<string>("");
  const [statusFilter, setStatusFilter] = useState<string>("all");
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [modalConfig, setModalConfig] = useState<{
    title: string;
    message: string;
    onConfirm: () => void;
  }>({ title: "", message: "", onConfirm: () => {} });

  // Determine if restaurant is blocked
  const isRestaurantBlocked = (restaurant: Restaurant) =>
    restaurant.status?.toLowerCase() === "blocked";

  // Open confirmation modal
  const openConfirmationModal = (title: string, message: string, onConfirm: () => void) => {
    setModalConfig({ title, message, onConfirm });
    setIsModalOpen(true);
  };
}

```

```

// Handle delete action
const handleDelete = (restaurantId: string, restaurantName: string) => {
  openConfirmationModal(
    "Confirm Delete",
    `Are you sure you want to delete ${restaurantName}? This action cannot be undone.`,
    async () => {
      try {
        await deleteRestaurant(restaurantId);
        setRestaurants((prev) => prev.filter((restaurant) => restaurant._id !== restaurantId));
        onDelete(restaurantId);
        setIsModalOpen(false);
      } catch (error: any) {
        alert(`Failed to delete restaurant: ${error.message}`);
        console.error("Delete error:", error);
      }
    }
  );
};

// Handle block/unblock action
const handleBlockToggle = (restaurantId: string, restaurantName: string, isBlocked: boolean) => {
  const newStatus = isBlocked ? "approved" : "blocked";
  openConfirmationModal(
    isBlocked ? "Confirm Unblock" : "Confirm Block",
    `Are you sure you want to ${isBlocked ? "unblock" : "block"} ${restaurantName}?`,
    async () => {
      try {
        await updateRestaurantStatus(restaurantId, newStatus);
        setRestaurants((prev) =>
          prev.map((restaurant) =>
            restaurant._id === restaurantId
              ? {
                  ...restaurant,
                  status: newStatus.charAt(0).toUpperCase() + newStatus.slice(1),
                }
              : restaurant
          )
        );
        setIsModalOpen(false);
      } catch (error: any) {
        alert(`${isBlocked ? "unblock" : "block"} restaurant: ${error.message}`);
        console.error("Block toggle error:", error);
      }
    }
  );
};

// Map headers to data fields
const getFieldValue = (restaurant: Restaurant, header: string) => {
  switch (header.toLowerCase()) {
    case "restaurant name":
      return restaurant.restaurantName;

```

```
case "contact person":  
    return restaurant.contactPerson;  
case "phone number":  
    return restaurant.phoneNumber;  
case "business type":  
    return restaurant.businessType;  
case "cuisine type":  
    return restaurant.cuisineType;  
case "operating hours":  
    return restaurant.operatingHours;  
case "delivery radius":  
    return restaurant.deliveryRadius;  
case "tax id":  
    return restaurant.taxId;  
case "street address":  
    return restaurant.address.streetAddress;  
case "city":  
    return restaurant.address.city;  
case "state":  
    return restaurant.address.state;  
case "zip code":  
    return restaurant.address.zipCode;  
case "country":  
    return restaurant.address.country;  
case "email":  
    return restaurant.email;  
case "business license":  
    return restaurant.businessLicense ? (  
        <a  
            href={restaurant.businessLicense}  
            target="_blank"  
            rel="noopener noreferrer"  
            className="text-blue-500 underline"  
        >  
            License  
        </a>  
    ) : (  
        "N/A"  
    );  
case "food safety certificate":  
    return restaurant.foodSafetyCert ? (  
        <a  
            href={restaurant.foodSafetyCert}  
            target="_blank"  
            rel="noopener noreferrer"  
            className="text-blue-500 underline"  
        >  
            Cert  
        </a>  
    ) : (  
        "N/A"  
    );  
case "exterior photo":
```

```

return restaurant.exteriorPhoto ? (
  <a
    href={restaurant.exteriorPhoto}
    target="_blank"
    rel="noopener noreferrer"
    className="text-blue-500 underline"
  >
    Photo
  </a>
) : (
  "N/A"
);

case "logo":
return restaurant.logo ? (
  <a
    href={restaurant.logo}
    target="_blank"
    rel="noopener noreferrer"
    className="text-blue-500 underline"
  >
    Logo
  </a>
) : (
  "N/A"
);

case "status":
return (
  <span className={getStatusStyles(restaurant.status || "Pending")}>
    {restaurant.status || "Pending"}
  </span>
);
default:
  return "";
}

// Filter restaurants by search term and status
const filteredRestaurants = restaurants.filter((restaurant) => {
  const matchesSearch = [
    restaurant.restaurantName,
    restaurant.contactPerson,
    restaurant.email,
    restaurant.cuisineType,
    restaurant.address.streetAddress,
    restaurant.address.city,
    restaurant.address.state,
    restaurant.address.country,
  ].some((value) => value.toLowerCase().includes(searchTerm.toLowerCase()));

  const matchesStatus =
    statusFilter === "all" ||
    restaurant.status?.toLowerCase() === statusFilter.toLowerCase();

```

```

    return matchesSearch && matchesStatus;
});

// Sort restaurants by restaurantName
const sortedRestaurants = [...filteredRestaurants].sort((a, b) => {
  const nameA = a.restaurantName.toLowerCase();
  const nameB = b.restaurantName.toLowerCase();
  return nameA.localeCompare(nameB);
});

// Get status color styles
const getStatusStyles = (status: string) => {
  const baseStyles = "px-2 py-1 text-xs font-semibold rounded-lg";
  switch (status.toLowerCase()) {
    case "approved":
      return `${baseStyles} bg-green-100 text-green-600 dark:bg-green-700 dark:text-green-200`;
    case "blocked":
      return `${baseStyles} bg-red-100 text-red-600 dark:bg-red-700 dark:text-red-200`;
    case "pending":
      return `${baseStyles} bg-yellow-100 text-yellow-600 dark:bg-yellow-700 dark:text-yellow-200`;
    case "rejected":
      return `${baseStyles} bg-gray-100 text-gray-600 dark:bg-gray-700 dark:text-gray-200`;
    default:
      return `${baseStyles} bg-gray-100 text-gray-600 dark:bg-gray-700 dark:text-gray-200`;
  }
};

return (
  <>
  {/* Confirmation Modal */}
  <ConfirmationModal
    isOpen={isModalOpen}
    onClose={() => setIsModalOpen(false)}
    onConfirm={modalConfig.onConfirm}
    title={modalConfig.title}
    message={modalConfig.message}
    confirmText="Confirm"
    cancelText="Cancel"
  />

  {/* Search and Filter Controls */}
  <div className="p-5 flex flex-col sm:flex-row gap-4 justify-between items-center">
    <input
      type="text"
      placeholder="Search restaurants..."
      value={searchTerm}
      onChange={(e) => setSearchTerm(e.target.value)}
      className="w-full sm:w-64 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
    />
    <select
      value={statusFilter}
      onChange={(e) => setStatusFilter(e.target.value)}>

```

```

    className="w-full sm:w-40 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  >
  <option value="all">All Statuses</option>
  <option value="approved">Approved</option>
  <option value="blocked">Blocked</option>
  <option value="pending">Pending</option>
  <option value="rejected">Rejected</option>
</select>
</div>

/* Desktop Table View with Horizontal Scroll */
<div className="hidden lg:block p-5 w-full rounded-md overflow-x-auto bg-white dark:bg-gray-
700">
  <table className="min-w-[1200px] w-full">
    <thead className="bg-gray-200 border-gray-200 dark:bg-gray-700 dark:border-gray-500">
      <tr>
        {headers.map((header, index) => (
          <th
            key={index}
            className="text-start py-4 px-6 font-semibold text-gray-600 dark:text-gray-200"
          >
            {header}
          </th>
        ))}
        <th className="text-start py-4 px-6 font-semibold text-gray-600 dark:text-gray-200">
          Actions
        </th>
      </tr>
    </thead>
    <tbody>
      {sortedRestaurants.map((restaurant, index) => (
        <tr
          key={restaurant._id}
          className={`border-b border-gray-200 dark:border-gray-500 ${{
            isRestaurantBlocked(restaurant)
              ? "bg-red-100 dark:bg-red-900"
              : index % 2 === 0
              ? "bg-gray-50 dark:bg-gray-600"
              : "bg-gray-100 dark:bg-gray-700"
            } hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors`}
        >
          {headers.map((header, idx) => (
            <td key={idx} className="py-4 px-6 text-gray-800 dark:text-white">
              {getFieldValue(restaurant, header)}
            </td>
          ))}
        <td className="py-4 px-6 flex space-x-2">
          <button
            onClick={() =>
              handleBlockToggle(
                restaurant._id,
                restaurant.restaurantName,

```

```

        isRestaurantBlocked(restaurant)
    )
}
className={`px-3 py-1 text-white rounded-lg transition-colors ${(
    isRestaurantBlocked(restaurant)
    ? "bg-green-500 hover:bg-green-600"
    : "bg-orange-500 hover:bg-orange-600"
)`}`}
disabled={restaurant.status?.toLowerCase() === "rejected"}
>
    {isRestaurantBlocked(restaurant) ? "Unblock" : "Block"}
</button>
<button
    onClick={() => handleDelete(restaurant._id, restaurant.restaurantName)}
    className="px-3 py-1 bg-red-500 text-white rounded-lg hover:bg-red-600 transition-colors"
>
    Delete
</button>
</td>
</tr>
))}
</tbody>
</table>
</div>

```

```

/* Mobile Grid View */
<div className="lg:hidden p-5 space-y-4 overflow-y-scroll h-full">
    {sortedRestaurants.map((restaurant) => (
        <div
            key={restaurant._id}
            className={`border rounded-md p-4 bg-white shadow-md hover:shadow-lg transition-shadow border-gray-200 dark:border-gray-500 ${(
                isRestaurantBlocked(restaurant)
                ? "bg-red-100 dark:bg-red-900"
                : "bg-gray-100 dark:bg-gray-600"
            )}`}
        >
            <div className="space-y-2">
                {headers.map((header, idx) => (
                    <div key={idx} className="flex justify-between items-center py-1">
                        <span className="font-medium text-gray-800 dark:text-white">{header}</span>
                        <span className="text-sm text-gray-600 dark:text-gray-300">
                            {getFieldValue(restaurant, header)}
                        </span>
                    </div>
                )))
            </div>
        <!-- Actions -->
        <div className="flex space-x-2 mt-4">
            <button
                onClick={() =>
                    handleBlockToggle(
                        restaurant._id,

```

```

        restaurant.restaurantName,
        isRestaurantBlocked(restaurant)
    )
}
className={`px-3 py-1 text-white rounded-lg transition-colors ${(
    isRestaurantBlocked(restaurant)
    ? "bg-green-500 hover:bg-green-600"
    : "bg-orange-500 hover:bg-orange-600"
)`}
disabled={restaurant.status?.toLowerCase() === "rejected"}
>
    {isRestaurantBlocked(restaurant) ? "Unblock" : "Block"}
</button>
<button
    onClick={() => handleDelete(restaurant._id, restaurant.restaurantName)}
    className="px-3 py-1 bg-red-500 text-white rounded-lg hover:bg-red-600 transition-colors"
>
    Delete
</button>
</div>
</div>
))}
</div>

/* No Results Message */
{sortedRestaurants.length === 0 && (
    <div className="p-5 text-center text-gray-600 dark:text-gray-300">
        No restaurants found.
    </div>
)
</>
);
};

export default AdminResturentTable;

```

src/components/admin/AdminUserPermissionTable.tsx

```
import React, { useState } from "react";
import { updateUserRole, updateUserStatus } from "../../utils/api";
import ConfirmationModal from "../UI/ConfirmationModal";

export interface User {
  _id: string;
  email: string;
  firstName: string;
  lastName: string;
  role: "CUSTOMER" | "RESTAURANT" | "DELIVERY" | "ADMIN";
  isActive: boolean;
  isVerified: boolean;
  address: {
    street: string;
    city: string;
    state: string;
    zipCode: string;
    country: string;
  };
  createdAt: string;
  updatedAt: string;
}

interface AdminUserPermissionProps {
  headers: string[];
  data: User[];
  setUsers: React.Dispatch<React.SetStateAction<User[]>>;
}

// Define role options
const ROLE_OPTIONS: { value: "CUSTOMER" | "RESTAURANT" | "DELIVERY" | "ADMIN"; label: string }[] = [
  { value: "CUSTOMER", label: "Customer" },
  { value: "RESTAURANT", label: "Restaurant" },
  { value: "DELIVERY", label: "Delivery" },
  { value: "ADMIN", label: "Admin" },
];

// Define status options
const STATUS_OPTIONS: { value: boolean; label: string }[] = [
  { value: true, label: "Active" },
  { value: false, label: "Inactive" },
];

// Define verification options
const VERIFICATION_OPTIONS: { value: boolean; label: string }[] = [
  { value: true, label: "Verified" },
  { value: false, label: "Unverified" },
];

// Define filter options for active and verification status
```

```

const ACTIVE_FILTER_OPTIONS: { value: string; label: string }[] = [
  { value: "all", label: "All Active Status" },
  { value: "true", label: "Active" },
  { value: "false", label: "Inactive" },
];

const VERIFICATION_FILTER_OPTIONS: { value: string; label: string }[] = [
  { value: "all", label: "All Verification Status" },
  { value: "true", label: "Verified" },
  { value: "false", label: "Unverified" },
];

const AdminUserPermissionTable: React.FC<AdminUserPermissionProps> = ({ headers, data, setUsers }) => {
  const [users, setLocalUsers] = useState<User[]>(data);
  const [searchTerm, setSearchTerm] = useState<string>("");
  const [roleFilter, setRoleFilter] = useState<string>"all";
  const [activeStatusFilter, setActiveStatusFilter] = useState<string>"all";
  const [verificationStatusFilter, setVerificationStatusFilter] = useState<string>"all";
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [modalConfig, setModalConfig] = useState<{
    title: string;
    message: string;
    onConfirm: () => void;
  }>({ title: "", message: "", onConfirm: () => {} });

  // Open confirmation modal
  const openConfirmationModal = (title: string, message: string, onConfirm: () => void) => {
    setModalConfig({ title, message, onConfirm });
    setIsModalOpen(true);
  };

  // Handle role change
  const handleRoleChange = (
    userId: string,
    userEmail: string,
    newRole: "CUSTOMER" | "RESTAURANT" | "DELIVERY" | "ADMIN"
  ) => {
    openConfirmationModal(
      "Confirm Role Change",
      `Are you sure you want to change the role of ${userEmail} to ${newRole}?`,
      async () => {
        try {
          await updateUserRole(userId, { role: newRole });
          const updateUser = (prev: User[]): User[] =>
            prev.map((user) => (user._id === userId ? { ...user, role: newRole } : user));
          setLocalUsers(updateUser);
          setUsers(updateUser);
          setIsModalOpen(false);
        } catch (error: any) {
          alert(`Failed to update role: ${error.message}`);
          console.error("Role change error:", error);
        }
      }
    );
  };
}

```

```

);
};

// Handle active status change
const handleActiveStatusChange = (userId: string, userEmail: string, isActive: boolean) => {
  openConfirmationModal(
    `Confirm ${isActive ? "Activate" : "Deactivate"} User`,
    `Are you sure you want to ${isActive ? "activate" : "deactivate"} ${userEmail}?`,
    async () => {
      try {
        await updateUserStatus(userId, { isActive });
        const updateUser = (prev: User[]): User[] =>
          prev.map((user) => (user._id === userId ? { ...user, isActive } : user));
        setLocalUsers(updateUser);
        setUsers(updateUser);
        setIsModalOpen(false);
      } catch (error: any) {
        alert(`Failed to update active status: ${error.message}`);
        console.error("Active status change error:", error);
      }
    }
  );
};

// Handle verification status change
const handleVerificationStatusChange = (userId: string, userEmail: string, isVerified: boolean) => {
  openConfirmationModal(
    `Confirm ${isVerified ? "Verify" : "Unverify"} User`,
    `Are you sure you want to ${isVerified ? "verify" : "unverify"} ${userEmail}?`,
    async () => {
      try {
        await updateUserStatus(userId, { isVerified });
        const updateUser = (prev: User[]): User[] =>
          prev.map((user) => (user._id === userId ? { ...user, isVerified } : user));
        setLocalUsers(updateUser);
        setUsers(updateUser);
        setIsModalOpen(false);
      } catch (error: any) {
        alert(`Failed to update verification status: ${error.message}`);
        console.error("Verification status change error:", error);
      }
    }
  );
};

// Map headers to data fields
const getFieldValue = (user: User, header: string) => {
  switch (header.toLowerCase()) {
    case "email":
      return user.email;
    case "first name":
      return user.firstName;
    case "last name":

```

```

    return user.lastName;
  case "role":
    return user.role;
  case "active status":
    return (
      <span
        className={`px-2 py-1 text-xs font-semibold rounded-lg ${(
          user.isActive
            ? "bg-green-100 text-green-600 dark:bg-green-700 dark:text-green-200"
            : "bg-red-100 text-red-600 dark:bg-red-700 dark:text-red-200"
        )}`}
      >
        {user.isActive ? "Active" : "Inactive"}
      </span>
    );
  case "verification status":
    return (
      <span
        className={`px-2 py-1 text-xs font-semibold rounded-lg ${(
          user.isVerified
            ? "bg-green-100 text-green-600 dark:bg-green-700 dark:text-green-200"
            : "bg-red-100 text-red-600 dark:bg-red-700 dark:text-red-200"
        )}`}
      >
        {user.isVerified ? "Verified" : "Unverified"}
      </span>
    );
  case "actions":
    return (
      <div className="flex space-x-2">
        <select
          value={user.role}
          onChange={(e) =>
            handleRoleChange(
              user._id,
              user.email,
              e.target.value as "CUSTOMER" | "RESTAURANT" | "DELIVERY" | "ADMIN"
            )
          }
        >
          {ROLE_OPTIONS.map((option) => (
            <option key={option.value} value={option.value}>
              {option.label}
            </option>
          )))
        </select>
        <select
          value={user.isActive.toString()}
          onChange={(e) => handleActiveStatusChange(user._id, user.email, e.target.value === "true")}
          className="px-2 py-1 rounded-lg border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
        >

```

```

    >
    {STATUS_OPTIONS.map((option) => (
      <option key={option.value.toString()} value={option.value.toString()}>
        {option.label}
      </option>
    )))
  </select>
<select
  value={user.isVerified.toString()}
  onChange={(e) =>
    handleVerificationStatusChange(user._id, user.email, e.target.value === "true")
  }
  className="px-2 py-1 rounded-lg border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
>
  {VERIFICATION_OPTIONS.map((option) => (
    <option key={option.value.toString()} value={option.value.toString()}>
      {option.label}
    </option>
  )))
  </select>
</div>
);
default:
  return "";
}
};

// Filter users by search term, role, active status, and verification status
const filteredUsers = users.filter((user) => {
  const matchesSearch = [user.email, user.firstName, user.lastName].some((value) =>
    value.toLowerCase().includes(searchTerm.toLowerCase())
  );
  const matchesRole = roleFilter === "all" || user.role.toLowerCase() === roleFilter.toLowerCase();
  const matchesActiveStatus =
    activeStatusFilter === "all" || user.isActive.toString() === activeStatusFilter;
  const matchesVerificationStatus =
    verificationStatusFilter === "all" || user.isVerified.toString() === verificationStatusFilter;
  return matchesSearch && matchesRole && matchesActiveStatus && matchesVerificationStatus;
});

// Sort users by email
const sortedUsers = [...filteredUsers].sort((a, b) => {
  const emailA = a.email.toLowerCase();
  const emailB = b.email.toLowerCase();
  return emailA.localeCompare(emailB);
});

return (

```

```

<>
 {/* Confirmation Modal */}
<ConfirmationModal
  isOpen={isModalOpen}
  onClose={() => setIsModalOpen(false)}
  onConfirm={modalConfig.onConfirm}
  title={modalConfig.title}
  message={modalConfig.message}
  confirmText="Confirm"
  cancelText="Cancel"
/>

 {/* Search and Filter Controls */}
<div className="p-5 flex flex-col sm:flex-row gap-4 justify-between items-center">
  <input
    type="text"
    placeholder="Search users..."
    value={searchTerm}
    onChange={(e) => setSearchTerm(e.target.value)}
    className="w-full sm:w-64 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
  <div className="flex flex-col sm:flex-row gap-4">
    <select
      value={roleFilter}
      onChange={(e) => setRoleFilter(e.target.value)}
      className="w-full sm:w-40 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
    >
      <option value="all">All Roles</option>
      {ROLE_OPTIONS.map((option) => (
        <option key={option.value} value={option.value.toLowerCase()}>
          {option.label}
        </option>
      ))}
    </select>
    <select
      value={activeStatusFilter}
      onChange={(e) => setActiveStatusFilter(e.target.value)}
      className="w-full sm:w-40 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
    >
      {ACTIVE_FILTER_OPTIONS.map((option) => (
        <option key={option.value} value={option.value}>
          {option.label}
        </option>
      ))}
    </select>
    <select
      value={verificationStatusFilter}
      onChange={(e) => setVerificationStatusFilter(e.target.value)}

```

```

    className="w-full sm:w-40 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  >
  {VERIFICATION_FILTER_OPTIONS.map((option) => (
    <option key={option.value} value={option.value}>
      {option.label}
    </option>
  )))
  </select>
</div>
</div>

/* Desktop Table View with Horizontal Scroll */
<div className="hidden lg:block p-5 w-full text-sm rounded-md overflow-x-auto bg-white dark:bg-gray-700">
  <table className="min-w-[1000px] w-full">
    <thead className="bg-gray-200 border-gray-200 dark:bg-gray-700 dark:border-gray-500">
      <tr>
        {headers.map((header, index) => (
          <th
            key={index}
            className="text-start py-4 px-6 font-semibold text-gray-600 dark:text-gray-200"
          >
            {header}
          </th>
        )))
      </tr>
    </thead>
    <tbody>
      {sortedUsers.map((user, index) => (
        <tr
          key={user._id}
          className={`border-b border-gray-200 dark:border-gray-500 ${index % 2 === 0 ? "bg-gray-50 dark:bg-gray-600" : "bg-gray-100 dark:bg-gray-700"} hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors`}
        >
          {headers.map((header, idx) => (
            <td key={idx} className="py-4 px-6 text-gray-800 dark:text-white">
              {getFieldValue(user, header)}
            </td>
          )))
        </tr>
      )))
    </tbody>
  </table>
</div>

/* Mobile Grid View */
<div className="lg:hidden p-5 space-y-4 overflow-y-scroll h-full">
  {sortedUsers.map((user) => (
    <div
      key={user._id}

```

```

    className="border rounded-md p-4 shadow-md hover:shadow-lg transition-shadow border-gray-200
dark:border-gray-500 bg-gray-100 dark:bg-gray-600"
  >
  <div className="space-y-2">
    {headers.map((header, idx) => (
      <div key={idx} className="flex justify-between items-center py-1">
        <span className="font-medium text-gray-800 dark:text-white">{header}</span>
        <span className="text-sm text-gray-600 dark:text-gray-300">
          {getFieldValue(user, header)}
        </span>
      </div>
    )))
  </div>
</div>
))}

</div>

/* No Results Message */
{sortedUsers.length === 0 && (
  <div className="p-5 text-center text-gray-600 dark:text-gray-300">
    No users found.
  </div>
)
</>
);
};

export default AdminUserPermissionTable;

```

src/components/admin/EarningsTable.tsx

```

import React, { useState } from "react";
import { MagnifyingGlassIcon } from "@heroicons/react/24/outline";
import { refundPayment } from "../../utils/api";

export interface Payment {
  _id: string;
  orderId: string;
  totalAmount: number;
  paymentMethod: "CREDIT_CARD" | "DEBIT_CARD";
  paymentStatus: "PENDING" | "COMPLETED" | "FAILED" | "REFUNDED";
  payhereTransactionId?: string;
  cardDetails: {
    maskedCardNumber?: string;
    cardHolderName?: string;
  };
  createdAt: string;
  refundedAt?: string;
  refundReason?: string;
  restaurantId: string;
  restaurantName?: string;
}

```

```

interface EarningsTableProps {
  headers: string[];
  data: Payment[];
  setPayments: React.Dispatch<React.SetStateAction<Payment[]>>;
}

const EarningsTable: React.FC<EarningsTableProps> = ({ headers, data, setPayments }) => {
  const [searchTerm, setSearchTerm] = useState<string>("");
}

const handleRefund = async (paymentId: string) => {
  try {
    const reason = prompt("Enter refund reason:");
    if (!reason) return;

    // Optimistic update
    setPayments((prev) =>
      prev.map((payment) =>
        payment._id === paymentId
          ? {
              ...payment,
              paymentStatus: "REFUNDED",
              refundedAt: new Date().toISOString(),
              refundReason: reason,
            }
          : payment
      )
    );
  }

  await refundPayment({ paymentId, reason });
} catch (err) {
  console.error("Refund error:", err);
  alert("Failed to process refund");
}
};

const filteredData = data.filter((payment) =>
  payment.orderId.toLowerCase().includes(searchTerm.toLowerCase())
);

// Map headers to payment fields for mobile view
const getFieldValue = (payment: Payment, header: string) => {
  switch (header.toLowerCase()) {
    case "date":
      return new Date(payment.createdAt).toLocaleDateString();
    case "amount":
      return `LKR ${payment.totalAmount.toFixed(2)}`;
    case "order id":
      return payment.orderId;
    case "resturent id":
      return payment.restaurantId;
    case "resturent name":
      return payment.restaurantName || "Unknown";
  }
};

```

```

case "method":
  return payment.paymentMethod.replace("_", " ");
case "status":
  return (
    <span
      className={`px-2 inline-flex text-xs leading-5 font-semibold rounded-full ${{
        payment.paymentStatus === "COMPLETED"
          ? "bg-green-100 text-green-800 dark:bg-green-700 dark:text-green-200"
          : payment.paymentStatus === "PENDING"
          ? "bg-yellow-100 text-yellow-800 dark:bg-yellow-700 dark:text-yellow-200"
          : payment.paymentStatus === "REFUNDED"
          ? "bg-blue-100 text-blue-800 dark:bg-blue-700 dark:text-blue-200"
          : "bg-red-100 text-red-800 dark:bg-red-700 dark:text-red-200"
      }}`}
    >
      {payment.paymentStatus}
    </span>
  );
case "transaction id":
  return payment.payhereTransactionId || "-";
default:
  return "";
}
};

return (
<div className="bg-white dark:bg-gray-800 rounded-lg shadow">
  {/* Search Input */}
  <div className="p-4 flex flex-col sm:flex-row gap-4 justify-between items-center">
    <div className="relative w-full sm:w-64">
      <input
        type="text"
        placeholder="Search by Order ID"
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        className="w-full pl-10 pr-4 py-2 rounded-full border border-gray-300 dark:border-gray-600 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
      />
      <MagnifyingGlassIcon className="w-5 h-5 absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-400" />
    </div>
  </div>
</div>

/* Desktop Table View */
<div className="hidden lg:block overflow-x-auto">
  <table className="min-w-full divide-y divide-gray-300 dark:divide-gray-700">
    <thead className="bg-gray-200 dark:bg-gray-900">
      <tr>
        {headers.map((header) => (
          <th
            key={header}
            className="px-6 py-3 text-left text-xs font-medium text-gray-500 dark:text-gray-300 uppercase tracking-wider"
        ))}
      </tr>
    </thead>
    <tbody>
      {rows.map((row) => (
        <tr>
          {row.map((cell) => (
            <td
              key={cell}
              className="px-6 py-4 text-sm text-gray-500 dark:text-gray-300 uppercase tracking-wider"
            )
          )}
        </tr>
      ))}
    </tbody>
  </table>
</div>

```

```

>
  {header}
</th>
)})
</tr>
</thead>
<tbody className="bg-white dark:bg-gray-800 divide-y divide-gray-200 dark:divide-gray-700">
  {filteredData.map((payment) => (
    <tr
      key={payment._id}
      className="hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors"
    >
      <td className="px-6 py-4 whitespace nowrap text-sm text-gray-900 dark:text-gray-100">
        {new Date(payment.createdAt).toLocaleDateString()}
      </td>
      <td className="px-6 py-4 whitespace nowrap text-sm text-gray-900 dark:text-gray-100">
        LKR {payment.totalAmount.toFixed(2)}
      </td>
      <td className="px-6 py-4 whitespace nowrap text-sm text-gray-900 dark:text-gray-100">
        ORD#{payment.orderId}
      </td>
      <td className="px-6 py-4 whitespace nowrap text-sm text-gray-900 dark:text-gray-100">
        RES#{payment.restaurantId}
      </td>
      <td className="px-6 py-4 whitespace nowrap text-sm text-gray-900 dark:text-gray-100">
        {payment.restaurantName || "Unknown"}
      </td>
      <td className="px-6 py-4 whitespace nowrap text-sm text-gray-900 dark:text-gray-100">
        {payment.paymentMethod.replace("_", " ")}
      </td>
      <td className="px-6 py-4 whitespace nowrap text-sm">
        <span
          className={`px-2 inline-flex text-xs leading-5 font-semibold rounded-full ${(
            payment.paymentStatus === "COMPLETED"
              ? "bg-green-100 text-green-800 dark:bg-green-700 dark:text-green-200"
              : payment.paymentStatus === "PENDING"
                ? "bg-yellow-100 text-yellow-800 dark:bg-yellow-700 dark:text-yellow-200"
                : payment.paymentStatus === "REFUNDED"
                  ? "bg-blue-100 text-blue-800 dark:bg-blue-700 dark:text-blue-200"
                  : "bg-red-100 text-red-800 dark:bg-red-700 dark:text-red-200"
            )}`}
        >
          {payment.paymentStatus}
        </span>
      </td>
      <td className="px-6 py-4 whitespace nowrap text-sm text-gray-900 dark:text-gray-100">
        {payment.payhereTransactionId || "-"}
      </td>
      <td className="px-6 py-4 whitespace nowrap text-sm">
        {payment.paymentStatus === "COMPLETED" && (
          <button
            onClick={() => handleRefund(payment._id)}
          >
        )}
      </td>
    
```

```

        className="text-orange-600 hover:text-orange-900 dark:text-orange-400 dark:hover:text-indigo-200"
      >
    Refund
  </button>
)
)
</td>
</tr>
))
</tbody>
</table>
</div>

/* Mobile Grid View */
<div className="lg:hidden p-4 space-y-4 overflow-y-scroll">
  {filteredData.map((payment) => (
    <div
      key={payment._id}
      className="border rounded-md p-4 bg-white dark:bg-gray-600 shadow-md hover:shadow-lg transition-shadow border-gray-200 dark:border-gray-500"
    >
      <div className="space-y-2">
        {headers.map((header) =>
          header.toLowerCase() !== "actions" ? (
            <div key={header} className="flex justify-between items-center py-1">
              <span className="font-medium text-gray-800 dark:text-white">{header}</span>
              <span className="text-sm text-gray-600 dark:text-gray-300">
                {getFieldValue(payment, header)}
              </span>
            </div>
          ) : null
        )
      )
    </div>
    /* Actions */
    {payment.paymentStatus === "COMPLETED" && (
      <div className="mt-4">
        <button
          onClick={() => handleRefund(payment._id)}
          className="px-3 py-1 bg-orange-500 text-white rounded-lg hover:bg-orange-600 dark:bg-orange-600 dark:hover:bg-orange-700 transition-colors"
        >
          Refund
        </button>
      </div>
    )
  )
})
</div>
</div>

/* No Results Message */
{filteredData.length === 0 && (
  <div className="p-4 text-center text-gray-600 dark:text-gray-300">
    No payments found.

```

```

        </div>
    )}
</div>
);
};

export default EarningsTable;

```

src/components/admin/Header.tsx

```

import { FaMoon, FaSun } from "react-icons/fa";
import { HiOutlineMenuAlt2 } from "react-icons/hi";
import { FiUser } from "react-icons/fi";
import { MdOutlinePowerSettingsNew } from "react-icons/md";
import { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { useAuth } from "../../context/AuthContext"; // Adjust path as needed
import toast, { Toaster } from "react-hot-toast";

interface HeaderProps {
  toggleDarkMode: () => void;
  darkMode: boolean;
  toggleSidebar: () => void;
}

const Header = ({ toggleDarkMode, darkMode, toggleSidebar }: HeaderProps) => {
  const { user, isAuthenticated, logout } = useAuth();
  const navigate = useNavigate();
  const [isDropdownOpen, setIsDropdownOpen] = useState(false);

  const handleLogout = async () => {
    if (!isAuthenticated) {
      toast.error("You are not logged in.");
      return;
    }

    try {
      await logout();
      toast.success("Logged out successfully!");
      navigate("/");
    } catch (error) {
      console.error("Logout error:", error);
      toast.error("Failed to log out. Please try again.");
    } finally {
      setIsDropdownOpen(false);
    }
  };

  // Toggle dropdown visibility
  const toggleDropdown = () => {
    setIsDropdownOpen((prev) => !prev);
  };
}

```

```

// Close dropdown when clicking outside
const handleBlur = () => {
  setTimeout(() => setIsDropdownOpen(false), 200); // Delay to allow click on dropdown items
};

return (
  <>
  <Toaster position="top-right" reverseOrder={false} />
  <nav className="fixed top-0 z-50 w-full bg-white border-b border-gray-200 dark:bg-gray-800 dark:border-gray-700">
    <div className="px-3 py-3 lg:px-5 lg:pl-3">
      <div className="flex items-center justify-between">
        <div className="flex items-center justify-start rtl:justify-end">
          <button
            className="inline-flex items-center p-2 text-sm text-gray-500 rounded-lg sm:hidden hover:bg-gray-100 focus:ring-gray-200 focus:outline-none focus:ring-2 dark:text-gray-400 dark:hover:bg-gray-700 dark:focus:ring-gray-600"
            onClick={toggleSidebar}
            aria-label="Toggle sidebar"
          >
            <HiOutlineMenuAlt2 className="text-2xl" />
          </button>
        </div>
      </div>
      <div className="flex items-center space-x-4">
        <button
          onClick={toggleDarkMode}
          className="dark:bg-slate-50 dark:text-slate-700 rounded-full p-2"
          aria-label={darkMode ? "Switch to light mode" : "Switch to dark mode"}
        >
          {darkMode ? <FaSun /> : <FaMoon className="text-gray-500" />}
        </button>
        {isAuthenticated &&
          <div className="relative">
            <button
              onClick={toggleDropdown}
              onBlur={handleBlur}
              className="flex items-center space-x-2 p-2 text-gray-500 hover:bg-gray-100 dark:text-gray-400 dark:hover:bg-gray-700 rounded-full focus:outline-none"
              aria-label="User menu"
            >
              <FiUser className="text-xl" />
              <span className="hidden sm:inline text-sm font-medium dark:text-white">
                {user?.email || "Admin"}
              </span>
            </button>
            {isDropdownOpen &&
              <div className="absolute right-0 mt-2 w-48 bg-white dark:bg-gray-800 border border-gray-200 dark:border-gray-700 rounded-md shadow-lg">
                <Link
                  to="/admin-dashboard/profile"
                </Link>
              </div>
            }
          </div>
        }
      </div>
    </div>
  </nav>
)

```

```

    className="flex items-center px-4 py-2 text-sm text-gray-700 dark:text-gray-300 hover:bg-gray-100 dark:hover:bg-gray-700"
    onClick={() => setIsDropdownOpen(false)}
  >
  <FiUser className="mr-2" />
  Profile
</Link>
<button
  onClick={handleLogout}
  className="flex items-center w-full px-4 py-2 text-sm text-gray-700 dark:text-gray-300
  hover:bg-gray-100 dark:hover:bg-gray-700"
  >
  <MdOutlinePowerSettingsNew className="mr-2" />
  Logout
</button>
</div>
)
</div>
)
</div>
</div>
</nav>
</>
);
};

export default Header;

```

src/components/admin/Sidebar.tsx

```

import { Link, useLocation } from "react-router-dom";
import {
  FiCalendar,
  FiSettings,
  FiPieChart,
  FiUser,
  FiDivide,
  FiUsers,
  FiHome,
  FiSunset
} from "react-icons/fi";
import { MdPayments } from "react-icons/md";
import { useState, useMemo } from "react";

interface SubMenuItem {
  path: string;
  title: string;
}

interface MenuItem {
  path: string;

```

```
title: string;
icon: JSX.Element;
subItems?: SubMenuItem[];
}

const Sidebar = ({ isSidebarOpen }: { isSidebarOpen: boolean }) => {
  const location = useLocation();
  const [openMenus, setOpenMenus] = useState<string[]>([]);

  const menuItems: MenuItem[] = useMemo(() => [
    { path: "/admin-dashboard-overview", title: "Overview", icon: <FiPieChart /> },
    {
      path: "/admin-dashboard/restaurant",
      title: "Restaurant Management",
      icon: <FiSunset />,
      subItems: [
        { path: "/admin-dashboard/resturent", title: "All Restaurants" },
        { path: "/admin-dashboard/resturent/request", title: "Requests" },
        { path: "/admin-dashboard/resturent/add", title: "Create Resturent" },
      ],
    },
    {
      path: "/admin-dashboard/user-management",
      title: "User Management",
      icon: <FiUser />,
      subItems: [
        { path: "/admin-dashboard/user-management/all", title: "All Users" },
        { path: "/admin-dashboard/user-management/add", title: "Add New User" },
        { path: "/admin-dashboard/user-management/roles", title: "Roles & Permissions" },
        { path: "/admin-dashboard/user-management/settings", title: "User Settings" }
      ],
    },
    {
      path: "/admin-dashboard/earnings",
      title: "Earnings & Payments",
      icon: <MdPayments />,
      subItems: [
        { path: "/admin-dashboard/earnings/earan", title: "Daily/Weekly Earnings" },
        { path: "/admin-dashboard/earnings/payouts", title: "Payouts" },
      ],
    },
    {
      path: "/admin-dashboard/promotions",
      title: "Promotions & Discounts",
      icon: <FiDivide />,
      subItems: [
        { path: "/admin-dashboard/promotions/create", title: "Create Promotion" },
        { path: "/admin-dashboard/promotions/active", title: "Active Promotions" },
      ],
    },
    {
      path: "/admin-dashboard/customers",
      title: "Customers",
    }
  ], [location]);
  
```

```

icon: <FiUsers />,
subItems: [
  { path: "/admin-dashboard/customers/feedback", title: "Customer Feedback" },
  { path: "/admin-dashboard/customers/loyalty", title: "Loyalty Programs" },
],
},
{

path: "/admin-dashboard/settings",
title: "Business Settings",
icon: <FiHome />,
subItems: [
  { path: "/admin-dashboard/settings/profile", title: "Profile & Store Info" },
  { path: "/admin-dashboard/settings/delivery", title: "Delivery Settings" },
  { path: "/admin-dashboard/settings/staff", title: "Staff Management" },
],
},
{
path: "/admin-dashboard/reports",
title: "Reports & Analytics",
icon: <FiCalendar />,
subItems: [
  { path: "/admin-dashboard/reports/sales", title: "Sales Reports" },
  { path: "/admin-dashboard/reports/behavior", title: "Customer Behavior" },
],
},
{
path: "/admin-dashboard/support",
title: "Help & Support",
icon: <FiSettings />,
subItems: [
  { path: "/admin-dashboard/support/contact", title: "Contact Uber Eats Support" },
  { path: "/admin-dashboard/support/faq", title: "FAQs & Guides" },
],
},
{
path: "/admin-dashboard/profile", title: "Profile", icon: <FiUser /> },
],
[]);

const toggleMenu = (path: string) => {
  setOpenMenus((prev) =>
    prev.includes(path) ? prev.filter((item) => item !== path) : [...prev, path]
  );
};

return (
  <aside
    className={`fixed top-0 left-0 z-40 w-64 h-screen pt-20 bg-white border-r border-gray-200 sm:translate-x-0 dark:bg-gray-800 dark:border-gray-700 transition-transform ${isSidebarOpen ? "translate-x-0" : "-translate-x-full"}`}
    aria-label="Sidebar navigation"
  >
    <div className="h-full px-3 pb-4 overflow-y-auto">

```

```

<ul className="space-y-1 font-medium">
  {menuItems.map((item) => {
    const isActive = location.pathname === item.path ||
      item.subItems?.some((sub) => location.pathname === sub.path);

    return (
      <li key={item.path}>
        {item.subItems ? (
          <button
            className={`flex items-center w-full p-2 text-gray-900 rounded-lg dark:text-white hover:bg-gray-100 dark:hover:bg-gray-700 group ${{
              isActive ? "bg-gray-100 dark:bg-gray-700" : ""
            }}`}
            onClick={() => toggleMenu(item.path)}
            aria-expanded={openMenus.includes(item.path)}
            aria-controls={`submenu-${item.path}`}
          >
            <span
              className="mr-3 text-gray-500 dark:text-gray-400 group-hover:text-gray-900 dark:group-hover:text-white"
              title={item.title}
            >
              {item.icon}
            </span>
            <span className="flex-1 text-left">{item.title}</span>
            <svg
              className={`w-4 h-4 transition-transform ${{
                openMenus.includes(item.path) ? "rotate-180" : ""
              }}`}
              fill="none"
              stroke="currentColor"
              viewBox="0 0 24 24"
              aria-hidden="true"
            >
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M19 9l-7 7-7-7" />
            </svg>
          </button>
        ) : (
          <Link
            to={item.path}
            className={`flex items-center p-2 text-gray-900 rounded-lg dark:text-white hover:bg-gray-100 dark:hover:bg-gray-700 group ${{
              isActive ? "bg-gray-100 dark:bg-gray-700" : ""
            }}`}
            aria-current={isActive ? "page" : undefined}
          >
            <span
              className="mr-3 text-gray-500 dark:text-gray-400 group-hover:text-gray-900 dark:group-hover:text-white"
              title={item.title}
            >
              {item.icon}
            </span>
          </Link>
        )
      </li>
    )
  )}
</ul>

```

```

        <span className="flex-1">{item.title}</span>
        </Link>
    )}

{item.subItems && openMenus.includes(item.path) && (
    <ul id={`submenu-${item.path}`} className="pl-6 mt-1 space-y-1">
        {item.subItems.map((subItem) => (
            <li key={subItem.path}>
                <Link
                    to={subItem.path}
                    className={`${flex items-center p-2 text-sm text-gray-700 rounded-lg dark:text-gray-300
hover:bg-gray-100 dark:hover:bg-gray-700 ${{
                        location.pathname === subItem.path ? "bg-gray-100 dark:bg-gray-700" : ""
                    }}`}
                    aria-current={location.pathname === subItem.path ? "page" : undefined}
                >
                    {subItem.title}
                </Link>
            </li>
        )));
    </ul>
)
</li>
)});

);
)};

</ul>
</div>
</aside>
);

};

export default Sidebar;

```

src/components/admin/Stats.tsx

```

import ResturentStatsCard from "../UI/ResturentStatsCard";
import ResturentTitle from "../UI/ResturentTitle";
import {
    FiPieChart,
    FiClock,
    FiCheckCircle,
    FiXCircle,
    FiDollarSign,
    FiUsers,
    FiShoppingCart,
    FiStar,
    FiTruck,
    FiPercent,
    FiBox,
    FiUserCheck,
} from "react-icons/fi";

```

```
interface Status {
  title: string;
  icon: JSX.Element;
  count: number;
}

const menuItemsData: Status[] = [
  {
    title: "New Orders",
    icon: <FiPieChart />,
    count: 15,
  },
  {
    title: "Pending Orders",
    icon: <FiClock />,
    count: 8,
  },
  {
    title: "Completed Orders",
    icon: <FiCheckCircle />,
    count: 45,
  },
  {
    title: "Canceled Orders",
    icon: <FiXCircle />,
    count: 5,
  },
  {
    title: "Total Revenue",
    icon: <FiDollarSign />,
    count: 2450,
  },
  {
    title: "Active Customers",
    icon: <FiUsers />,
    count: 120,
  },
  {
    title: "Items Sold",
    icon: <FiShoppingCart />,
    count: 180,
  },
  {
    title: "Customer Ratings",
    icon: <FiStar />,
    count: 4,
  },
  {
    title: "Delivery Orders",
    icon: <FiTruck />,
    count: 35,
  },
]
```

```

    title: "Discounts Applied",
    icon: <FiPercent />,
    count: 12,
  },
{
  title: "Menu Items",
  icon: <FiBox />,
  count: 50,
},
{
  title: "Staff Online",
  icon: <FiUserCheck />,
  count: 6,
},
];
};

const Stats = () => {
  return (
    <div className="p-4">
      <ResturentTitle text="Restaurant Stats"/>

      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
        {menuItemsData.map((data, index) => (
          <ResturentStatsCard key={index} data={data} />
        )))
      </div>
    </div>
  );
};

export default Stats;

```

src/components/admin/UserSettingTable.tsx

```

import React, { useState } from "react";
import { getUserById, updateUser, deleteUser } from "../../utils/api";
import ConfirmationModal from "../UI/ConfirmationModal";

export interface User {
  _id: string;
  email: string;
  firstName: string;
  lastName: string;
  role: "CUSTOMER" | "RESTAURANT" | "DELIVERY" | "ADMIN";
  isActive: boolean;
  isVerified: boolean;
  address?: {
    street: string;
    city: string;
    state: string;
  }
}

```

```

zipCode: string;
country: string;
};
createdAt: string;
updatedAt: string;
}

interface UserSettingTableProps {
  headers: string[];
  data: User[];
  setUsers: React.Dispatch<React.SetStateAction<User[]>>;
}

const UserSettingTable: React.FC<UserSettingTableProps> = ({ headers, data, setUsers }) => {
  const [users, setLocalUsers] = useState<User[]>(data);
  const [searchTerm, setSearchTerm] = useState<string>("");
  const [isEditModalOpen, setIsEditModalOpen] = useState(false);
  const [isDeleteModalOpen, setIsDeleteModalOpen] = useState(false);
  const [selectedUser, setSelectedUser] = useState<User | null>(null);
  const [formData, setFormData] = useState({
    email: "",
    firstName: "",
    lastName: "",
    address: {
      street: "",
      city: "",
      state: "",
      zipCode: "",
      country: ""
    }
  });
  const [formError, setFormError] = useState<string | null>(null);

  // Open edit modal and fetch user data
  const openEditModal = async (userId: string) => {
    try {
      const response = await getUserId(userId);
      console.log("getUserId response:", response); // Log response for debugging
      const user = response.data.user; // Access nested user object
      setSelectedUser(user);
      setFormData({
        email: user.email || "",
        firstName: user.firstName || "",
        lastName: user.lastName || "",
        address: {
          street: user.address?.street || "",
          city: user.address?.city || "",
          state: user.address?.state || "",
          zipCode: user.address?.zipCode || "",
          country: user.address?.country || ""
        }
      });
      setIsEditModalOpen(true);
    }
  };
}

```

```

setFormError(null);
} catch (error: any) {
  alert(`Failed to fetch user: ${error.message}`);
  console.error("Fetch user error:", error);
}
};

// Handle form input changes
const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const { name, value } = e.target;
  if (name.includes("address.")) {
    const field = name.split(".")[1];
    setFormData((prev) => ({
      ...prev,
      address: { ...prev.address, [field]: value },
    }));
  } else {
    setFormData((prev) => ({ ...prev, [name]: value }));
  }
};

// Handle form submission
const handleFormSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  if (!selectedUser) return;

  try {
    const response = await updateUser(selectedUser._id, formData);
    console.log("updateUser response:", response); // Log response for debugging
    const updatedUser: User = response.data.user; // Access nested user object
    const updateUsers = (prev: User[]): User[] =>
      prev.map((user) =>
        user._id === selectedUser._id
          ? {
            ...user,
            email: updatedUser.email,
            firstName: updatedUser.firstName,
            lastName: updatedUser.lastName,
            address: updatedUser.address || {
              street: "",
              city: "",
              state: "",
              zipCode: "",
              country: "",
            },
          }
          : user
      );
    setLocalUsers(updateUsers);
    setUsers(updateUsers);
    setIsEditModalOpen(false);
    setSelectedUser(null);
    setFormError(null);
  } catch (error) {
    setFormError(error.message);
  }
};

```

```

} catch (error: any) {
  setFormError(error.message || "Failed to update user");
  console.error("Update user error:", error);
}
};

// Handle delete user
const handleDeleteUser = (userId: string, userEmail: string) => {
  openDeleteModal(
    "Confirm Delete User",
    `Are you sure you want to delete ${userEmail}?`,
    async () => {
      try {
        await deleteUser(userId);
        const updateUsers = (prev: User[]): User[] =>
          prev.filter((user) => user._id !== userId);
        setLocalUsers(updateUsers);
        setUsers(updateUsers);
        setIsDeleteModalOpen(false);
      } catch (error: any) {
        alert(`Failed to delete user: ${error.message}`);
        console.error("Delete user error:", error);
      }
    }
  );
};

// Open delete confirmation modal
const openDeleteModal = (title: string, message: string, onConfirm: () => void) => {
  setIsDeleteModalOpen(true);
  setModalConfig({ title, message, onConfirm });
};

const [modalConfig, setModalConfig] = useState<{
  title: string;
  message: string;
  onConfirm: () => void;
}>({ title: "", message: "", onConfirm: () => {} });

// Map headers to data fields
const getFieldValue = (user: User, header: string) => {
  switch (header.toLowerCase()) {
    case "email":
      return user.email;
    case "first name":
      return user.firstName;
    case "last name":
      return user.lastName;
    case "role":
      return user.role;
    case "active status":
      return (
        <span>

```

```

    className={`px-2 py-1 text-xs font-semibold rounded-lg ${

      user.isActive
        ? "bg-green-100 text-green-600 dark:bg-green-700 dark:text-green-200"
        : "bg-red-100 text-red-600 dark:bg-red-700 dark:text-red-200"
    }`}
  >
  {user.isActive ? "Active" : "Inactive"}
  </span>
);

case "verification status":
  return (
    <span
      className={`px-2 py-1 text-xs font-semibold rounded-lg ${

        user.isVerified
          ? "bg-green-100 text-green-600 dark:bg-green-700 dark:text-green-200"
          : "bg-red-100 text-red-600 dark:bg-red-700 dark:text-red-200"
      }`}
    >
    {user.isVerified ? "Verified" : "Unverified"}
    </span>
);
case "actions":
  return (
    <div className="flex space-x-2">
      <button
        onClick={() => openEditModal(user._id)}
        className="px-2 py-1 bg-indigo-500 text-white rounded-lg hover:bg-indigo-600 focus:outline-none
focus:ring-2 focus:ring-indigo-500"
      >
        Edit
      </button>
      <button
        onClick={() => handleDeleteUser(user._id, user.email)}
        className="px-2 py-1 bg-red-500 text-white rounded-lg hover:bg-red-600 focus:outline-none
focus:ring-2 focus:ring-red-500"
      >
        Delete
      </button>
    </div>
  );
default:
  return "";
}
};

// Filter users by search term
const filteredUsers = users.filter((user) =>
  [user.email, user.firstName, user.lastName].some((value) =>
    value.toLowerCase().includes(searchTerm.toLowerCase())
  )
);

// Sort users by email

```

```

const sortedUsers = [...filteredUsers].sort((a, b) =>
  a.email.toLowerCase().localeCompare(b.email.toLowerCase())
);

return (
  <>
  {/* Delete Confirmation Modal */}
  <ConfirmationModal
    isOpen={isDeleteModalOpen}
    onClose={() => setIsDeleteModalOpen(false)}
    onConfirm={modalConfig.onConfirm}
    title={modalConfig.title}
    message={modalConfig.message}
    confirmText="Delete"
    cancelText="Cancel"
  />

  {/* Edit Modal */}
  {isEditModalOpen && (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">
      <div className="bg-white dark:bg-gray-800 p-6 rounded-lg max-w-lg w-full max-h-[80vh] overflow-y-
      auto">
        <h2 className="text-xl font-semibold text-gray-800 dark:text-gray-200 mb-4">
          Edit User
        </h2>
        {formError && (
          <div className="text-red-500 mb-4">{formError}</div>
        )}
        <form onSubmit={handleFormSubmit} className="space-y-4">
          <div>
            <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
              Email
            </label>
            <input
              type="email"
              name="email"
              value={formData.email}
              onChange={handleInputChange}
              required
              className="w-full px-3 py-2 border border-gray-300 dark:border-gray-500 rounded-lg bg-white
              dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
            />
          </div>
          <div>
            <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
              First Name
            </label>
            <input
              type="text"
              name="firstName"
              value={formData.firstName}
              onChange={handleInputChange}
              required
            />
          </div>
        </form>
      </div>
    </div>
  )
)

```

```
    className="w-full px-3 py-2 border border-gray-300 dark:border-gray-500 rounded-lg bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
</div>
<div>
  <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
    Last Name
  </label>
  <input
    type="text"
    name="lastName"
    value={formData.lastName}
    onChange={handleInputChange}
    required
    className="w-full px-3 py-2 border border-gray-300 dark:border-gray-500 rounded-lg bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
</div>
<div>
  <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
    Street
  </label>
  <input
    type="text"
    name="address.street"
    value={formData.address.street}
    onChange={handleInputChange}
    className="w-full px-3 py-2 border border-gray-300 dark:border-gray-500 rounded-lg bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
</div>
<div>
  <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
    City
  </label>
  <input
    type="text"
    name="address.city"
    value={formData.address.city}
    onChange={handleInputChange}
    className="w-full px-3 py-2 border border-gray-300 dark:border-gray-500 rounded-lg bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
</div>
<div>
  <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
    State
  </label>
  <input
    type="text"
    name="address.state"
    value={formData.address.state}
    onChange={handleInputChange}
```

```
    className="w-full px-3 py-2 border border-gray-300 dark:border-gray-500 rounded-lg bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
</div>
<div>
  <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
    Zip Code
  </label>
  <input
    type="text"
    name="address.zipCode"
    value={formData.address.zipCode}
    onChange={handleInputChange}
    className="w-full px-3 py-2 border border-gray-300 dark:border-gray-500 rounded-lg bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
</div>
<div>
  <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
    Country
  </label>
  <input
    type="text"
    name="address.country"
    value={formData.address.country}
    onChange={handleInputChange}
    className="w-full px-3 py-2 border border-gray-300 dark:border-gray-500 rounded-lg bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
</div>
<div className="flex justify-end space-x-2">
  <button
    type="button"
    onClick={() => {
      setIsEditModalOpen(false);
      setSelectedUser(null);
      setFormError(null);
    }}
    className="px-4 py-2 bg-gray-300 text-gray-800 rounded-lg hover:bg-gray-400 focus:outline-none
focus:ring-2 focus:ring-gray-500"
  >
    Cancel
  </button>
  <button
    type="submit"
    className="px-4 py-2 bg-indigo-500 text-white rounded-lg hover:bg-indigo-600 focus:outline-none
focus:ring-2 focus:ring-indigo-500"
  >
    Save
  </button>
</div>
</form>
</div>
```

```

    </div>
)}
```

/* Search Bar */

```

<div className="p-5 flex justify-between items-center">
  <input
    type="text"
    placeholder="Search users..."
    value={searchTerm}
    onChange={(e) => setSearchTerm(e.target.value)}
    className="w-full sm:w-64 px-4 py-2 rounded-lg border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
</div>
```

/* Desktop Table View */

```

<div className="hidden lg:block p-5 w-full text-sm rounded-md overflow-x-auto bg-white dark:bg-gray-700">
  <table className="min-w-[1000px] w-full">
    <thead className="bg-gray-200 border-gray-200 dark:bg-gray-700 dark:border-gray-500">
      <tr>
        {headers.map((header, index) => (
          <th
            key={index}
            className="text-start py-4 px-6 font-semibold text-gray-600 dark:text-gray-200"
          >
            {header}
          </th>
        ))}
      </tr>
    </thead>
    <tbody>
      {sortedUsers.map((user, index) => (
        <tr
          key={user._id}
          className={`${`border-b border-gray-200 dark:border-gray-500 ${index % 2 === 0 ? "bg-gray-50 dark:bg-gray-600" : "bg-gray-100 dark:bg-gray-700`} `} hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors`}
        >
          {headers.map((header, idx) => (
            <td key={idx} className="py-4 px-6 text-gray-800 dark:text-white">
              {getFieldValue(user, header)}
            </td>
          )))
        </tr>
      ))}
    </tbody>
  </table>
</div>
```

/* Mobile Grid View */

```

<div className="lg:hidden p-5 space-y-4 overflow-y-scroll h-full">
  {sortedUsers.map((user) => (
```

```

<div
  key={user._id}
  className="border rounded-md p-4 shadow-md hover:shadow-lg transition-shadow border-gray-200
dark:border-gray-500 bg-gray-100 dark:bg-gray-600"
>
  <div className="space-y-2">
    {headers.map((header, idx) => (
      <div key={idx} className="flex justify-between items-center py-1">
        <span className="font-medium text-gray-800 dark:text-white">{header}</span>
        <span className="text-sm text-gray-600 dark:text-gray-300">
          {getFieldValue(user, header)}
        </span>
      </div>
    )))
  </div>
</div>
))}

</div>

{/* No Results Message */}
{sortedUsers.length === 0 && (
  <div className="p-5 text-center text-gray-600 dark:text-gray-300">
    No users found.
  </div>
)
</>
);
};

export default UserSettingTable;

```

src/components/Home/DeliverySection.tsx

```

import { motion } from "framer-motion";

function DeliverySection() {
  return (
    <section className="py-20 w-full bg-white overflow-hidden">
      <div className="max-w-7xl mx-auto px-4 grid md:grid-cols-2 gap-12 items-center">
        {/* Left side - Animated Illustration */}
        <motion.div
          initial={{ x: -100, opacity: 0 }}
          whileInView={{ x: 0, opacity: 1 }}
          viewport={{ once: true }}
          transition={{ duration: 0.8, ease: "easeOut" }}
          className="relative"
        >
          <motion.img
            src="/Home/delivery-boy.png"
            alt="Delivery Service"
            className="w-full h-auto max-w-lg mx-auto"
            animate={{

```

```

        y: [0, -20, 0],
    })
transition={ {
    duration: 4,
    repeat: Infinity,
    ease: "easeInOut",
}}
/>

{/* Floating elements */}
<motion.div
    className="absolute top-10 left-10 bg-orange-100 rounded-full p-4"
    animate={ {
        scale: [1, 1.1, 1],
        rotate: [0, 5, 0],
    }}
    transition={ {
        duration: 3,
        repeat: Infinity,
        ease: "easeInOut",
    }}
>
    <span className="text-3xl">📌</span>
</motion.div>

<motion.div
    className="absolute bottom-10 right-10 bg-green-100 rounded-full p-4"
    animate={ {
        scale: [1, 1.1, 1],
        rotate: [0, -5, 0],
    }}
    transition={ {
        duration: 3.5,
        repeat: Infinity,
        ease: "easeInOut",
        delay: 0.5,
    }}
>
    <span className="text-3xl">🎁</span>
</motion.div>
</motion.div>

{/* Right side - Content */}
<motion.div
    initial={ { x: 100, opacity: 0 } }
    whileInView={ { x: 0, opacity: 1 } }
    viewport={ { once: true } }
    transition={ { duration: 0.8, ease: "easeOut", delay: 0.2 } }
>
    <h2 className="text-4xl font-bold text-gray-900 mb-6">
        Lightning Fast
        <span className="text-orange-600"> Delivery</span>
    </h2>

```

```
<div className="space-y-6">
  <motion.div
    initial={{ opacity: 0, y: 20 }}
    whileInView={{ opacity: 1, y: 0 }}
    viewport={{ once: true }}
    transition={{ delay: 0.4 }}
    className="flex items-start space-x-4"
  >
    <div className="bg-orange-100 p-3 rounded-full">
      <span className="text-2xl">⚡</span>
    </div>
    <div>
      <h3 className="text-xl font-semibold mb-2">Quick Delivery</h3>
      <p className="text-gray-600">
        Get your food delivered in under 30 minutes or your money back
        guaranteed
      </p>
    </div>
  </motion.div>

  <motion.div
    initial={{ opacity: 0, y: 20 }}
    whileInView={{ opacity: 1, y: 0 }}
    viewport={{ once: true }}
    transition={{ delay: 0.6 }}
    className="flex items-start space-x-4"
  >
    <div className="bg-orange-100 p-3 rounded-full">
      <span className="text-2xl">📍</span>
    </div>
    <div>
      <h3 className="text-xl font-semibold mb-2">
        Real-time Tracking
      </h3>
      <p className="text-gray-600">
        Track your delivery in real-time with our advanced GPS system
      </p>
    </div>
  </motion.div>

  <motion.div
    initial={{ opacity: 0, y: 20 }}
    whileInView={{ opacity: 1, y: 0 }}
    viewport={{ once: true }}
    transition={{ delay: 0.8 }}
    className="flex items-start space-x-4"
  >
    <div className="bg-orange-100 p-3 rounded-full">
      <span className="text-2xl">❤️</span>
    </div>
    <div>
      <h3 className="text-xl font-semibold mb-2">
```

```

    Professional Service
  </h3>
  <p className="text-gray-600">
    Our trained delivery partners ensure your food arrives fresh
    and safe
  </p>
</div>
</motion.div>
</div>
</motion.div>
</div>
</section>
);
}

```

```
export default DeliverySection;
```

src/components/Home/HeroSection.tsx

```

import { useState, useEffect } from "react";
import CustomButton from "../UI/CustomButton";

const sliderData = [
  {
    id: 1,
    image: "/Home/slider_img_1.png",
    title: "Great Food. Tastes Good.",
    subtitle: "Fast Food & Restaurants",
    discount: "70% off",
    description:
      "Experience the best flavors from our kitchen to your table. Fresh, delicious, and made with love.",
  },
  {
    id: 2,
    image: "/Home/slider_img_2.png",
    title: "Fresh & Healthy.",
    subtitle: "Organic Food & Vegetables",
    discount: "50% off",
    description:
      "Discover our selection of fresh, healthy meals made with premium ingredients.",
  },
  {
    id: 3,
    image: "/Home/slider_img_3.png",
    title: "Quick Delivery.",
    subtitle: "Express Delivery Service",
    discount: "40% off",
    description:
      "Fast and reliable delivery to your doorstep. Hot food, right when you want it.",
  },
];

```

```

function HeroSection() {
  const [currentSlide, setCurrentSlide] = useState(0);

  useEffect(() => {
    const timer = setInterval(() => {
      setCurrentSlide((prev) => (prev + 1) % sliderData.length);
    }, 5000);

    return () => clearInterval(timer);
  }, []);

  return (
    <div className="relative lg:mt-0 h-[1000px] md:h-[1100px] lg:h-auto lg:min-h-screen w-full overflow-hidden bg-gray-50">
      {/* Background Layer with Pattern */}
      <div
        className="absolute inset-0 z-20 opacity-40"
        style={{
          backgroundImage: "url('/Home/banner_bg.jpg')",
          backgroundSize: "cover",
          backgroundPosition: "center",
          backgroundRepeat: "repeat",
        }}
      />

      {/* Grid Background */}
      <div className="absolute inset-0 z-15 overflow-hidden ">
        <div className="absolute inset-0 bg-[linear-gradient(to_right,#fb923c12_1px,transparent_1px),linear-gradient(to_bottom,#fb923c12_1px,transparent_1px)] bg-[size:24px_24px] [mask-image:radial-gradient(ellipse_80%_50%_at_50%_0%,#000_70%,transparent_110%)]"></div>
      </div>

      {/* Content Container */}
      <div className="absolute lg:top-20 inset-0 z-30 flex items-center justify-center">
        <div className="container mx-auto px-4 md:px-8 lg:px-16 py-12">
          <div className="grid grid-cols-1 lg:grid-cols-2 gap-12 items-center">
            {/* Text Content */}
            <div className="text-center lg:text-left space-y-6 max-w-xl mx-auto lg:mx-0">
              {/* Discount Badge */}
              <div className="bg-orange-600 text-white px-6 py-2 rounded-full text-sm font-semibold">
                {sliderData[currentSlide].discount}
              </div>
            </div>

            <h1 className="text-4xl md:text-5xl lg:text-6xl font-bold text-gray-900">
              {sliderData[currentSlide].title}
            </h1>

            <h2 className="text-2xl md:text-3xl text-orange-600 font-semibold">
              {sliderData[currentSlide].subtitle}
            </h2>
          </div>
        </div>
      </div>
    </div>
  );
}

```

```

<p className="text-gray-600 text-lg">
  {sliderData[currentSlide].description}
</p>

<div className="pt-4">
  <CustomButton
    title="Shop Now"
    bgColor="bg-orange-600"
    textColor="text-white"
    onClick={() => {}}
    style="hover:bg-orange-700 px-8 py-3 text-lg"
  />
</div>
</div>

/* Image Slider */
<div className="relative w-full max-w-md mx-auto">
  <div className="relative aspect-square w-full">
    {sliderData.map((slide, index) => (
      <div
        key={slide.id}
        className={`absolute inset-0 transition-all duration-700 transform ${{
          index === currentSlide
            ? "opacity-100 translate-x-0"
            : "opacity-0 translate-x-full"
        }}`}
      >
        <div className="w-full h-full rounded-full overflow-hidden">
          <img
            src={slide.image}
            alt={slide.title}
            className="w-full h-full object-cover"
          />
        </div>
      </div>
    )))
  </div>
</div>

/* Slider Dots */
<div className="absolute -bottom-12 left-1/2 transform -translate-x-1/2 flex space-x-3">
  {sliderData.map(_, index) => (
    <button
      key={index}
      onClick={() => setCurrentSlide(index)}
      className={`w-3 h-3 rounded-full transition-all duration-300 ${{
        index === currentSlide
          ? "bg-orange-600 w-6"
          : "bg-gray-300 hover:bg-gray-400"
      }}`}
      aria-label={`Go to slide ${index + 1}`}
    />
  )))
</div>

```

```
        </div>
    </div>
    </div>
    </div>
    </div>
);
}

export default HeroSection;
```

src/components/Home/RestaurantsSection.tsx

```
import Marquee from "react-fast-marquee";
import { motion } from "framer-motion";
```

```
interface Restaurant {
    id: number;
    name: string;
    image: string;
    rating: number;
    cuisine: string;
    deliveryTime: string;
}
```

```
const restaurants = [
    {
        id: 1,
        name: "Burger King",
        image: "/restaurants/burger-king.png",
        rating: 4.5,
        cuisine: "Fast Food",
        deliveryTime: "25-30 min",
    },
    {
        id: 2,
        name: "Pizza Hut",
        image: "/restaurants/pizza-hut.png",
        rating: 4.3,
        cuisine: "Italian",
        deliveryTime: "30-35 min",
    },
    {
        id: 3,
        name: "Subway",
        image: "/restaurants/subway.png",
        rating: 4.2,
        cuisine: "Sandwiches",
        deliveryTime: "20-25 min",
    },
    {
        id: 4,
        name: "KFC",
    }
]
```

```

image: "/restaurants/kfc.png",
rating: 4.4,
cuisine: "Chicken",
deliveryTime: "25-30 min",
},
{
id: 5,
name: "Taco Bell",
image: "/restaurants/taco-bell.png",
rating: 4.1,
cuisine: "Mexican",
deliveryTime: "20-25 min",
},
// Add more restaurants as needed
];

```

```

const RestaurantCard = ({ restaurant }: { restaurant: Restaurant }) => {
  return (
    <motion.div
      className="flex-shrink-0 w-72 mx-4 bg-white rounded-xl shadow-lg overflow-hidden hover:shadow-xl transition-shadow duration-300"
      whileHover={{ y: -5 }}
    >
      <div className="relative h-48">
        <img
          src={restaurant.image}
          alt={restaurant.name}
          className="w-full h-full object-cover"
        />
        <div className="absolute top-4 right-4 bg-white px-2 py-1 rounded-lg text-sm font-semibold text-gray-700">
          ★ ⚡ {restaurant.rating}
        </div>
      </div>
      <div className="p-4">
        <h3 className="text-xl font-bold text-gray-800 mb-2">
          {restaurant.name}
        </h3>
        <p className="text-gray-600 mb-2">{restaurant.cuisine}</p>
        <div className="flex items-center justify-between text-sm">
          <span className="text-gray-500">🕒 {restaurant.deliveryTime}</span>
          <span className="text-orange-600 font-semibold">Order Now</span>
        </div>
      </div>
    </motion.div>
  );
};

function RestaurantsSection() {
  return (
    <section className="relative py-16 w-full bg-gradient-to-b from-gray-50 to-gray-200">

```

```

<div className="max-w-7xl mx-auto px-4 mb-12">
  <motion.div
    initial={{ opacity: 0, y: 20 }}
    whileInView={{ opacity: 1, y: 0 }}
    viewport={{ once: true }}
    transition={{ duration: 0.8 }}>
    <h2 className="text-3xl md:text-4xl font-bold text-center text-gray-900 mb-4">
      Popular Restaurants
    </h2>
    <p className="text-gray-600 text-center max-w-2xl mx-auto">
      Discover the best food from over 1,000 restaurants and fast delivery
      to your doorstep
    </p>
  </motion.div>
</div>

<div className="mb-8">
  <Marquee gradient={false} speed={40} pauseOnHover={true}>
    {restaurants.map((restaurant) => (
      <RestaurantCard key={restaurant.id} restaurant={restaurant} />
    )))
  </Marquee>
</div>

<div className="mt-8">
  <Marquee
    gradient={false}
    speed={40}
    pauseOnHover={true}
    direction="right"
  >
    {restaurants.map((restaurant) => (
      <RestaurantCard key={restaurant.id} restaurant={restaurant} />
    )))
  </Marquee>
</div>
</section>
);
}

export default RestaurantsSection;

```

src/components/Home/ResturentGetStartedSection.tsx

```

import resturent from "../../assets/images/get-start-resturent.webp";
import { motion } from "framer-motion";

function ResturentGetStartedSection() {
  return (
    <section className="py-16 md:py-20 w-full bg-white overflow-hidden">
      <div className="max-w-7xl mx-auto px-4 flex flex-col md:flex-row gap-8 md:gap-12 items-center">

```

```

/* Left side - Animated Illustration */
<motion.div
  initial={{ x: -100, opacity: 0 }}
  whileInView={{ x: 0, opacity: 1 }}
  viewport={{ once: true }}
  transition={{ duration: 0.8, ease: "easeOut" }}
  className="w-full md:w-1/2 relative"
>
  <motion.img
    src={resturent}
    alt="Restaurant Partner Setup"
    className="w-full h-auto max-w-md mx-auto"
    animate={{
      y: [0, -15, 0],
    }}
    transition={{
      duration: 3.5,
      repeat: Infinity,
      ease: "easeInOut",
    }}
  />

  /* Floating elements */
  <motion.div
    className="absolute top-8 left-8 bg-orange-100 rounded-full p-3"
    animate={{
      scale: [1, 1.1, 1],
      rotate: [0, 5, 0],
    }}
    transition={{
      duration: 2.5,
      repeat: Infinity,
      ease: "easeInOut",
    }}
  >
    <span className="text-2xl">🍴</span>
  </motion.div>

  <motion.div
    className="absolute bottom-8 right-8 bg-green-100 rounded-full p-3"
    animate={{
      scale: [1, 1.1, 1],
      rotate: [0, -5, 0],
    }}
    transition={{
      duration: 3,
      repeat: Infinity,
      ease: "easeInOut",
      delay: 0.5,
    }}
  >
    <span className="text-2xl">กระเป๋า</span>
  </motion.div>

```

```

</motion.div>

/* Right side - Content */
<motion.div
  initial={{ x: 100, opacity: 0 }}
  whileInView={{ x: 0, opacity: 1 }}
  viewport={{ once: true }}
  transition={{ duration: 0.8, ease: "easeOut", delay: 0.2 }}
  className="w-full md:w-1/2"
>
  <h2 className="text-3xl md:text-4xl font-bold text-gray-900 mb-6">
    Get Started in Just{" "}
    <span className="text-orange-600">3 Steps</span>
  </h2>

  <div className="space-y-6">
    /* Step 1 */
    <motion.div
      initial={{ opacity: 0, y: 20 }}
      whileInView={{ opacity: 1, y: 0 }}
      viewport={{ once: true }}
      transition={{ delay: 0.4 }}
      className="flex items-start space-x-4"
    >
      <div className="bg-orange-100 p-3 rounded-full flex-shrink-0">
        <span className="text-2xl">1 □</span>
      </div>
      <div>
        <h3 className="text-xl font-semibold mb-2">Tell Us About Your Restaurant</h3>
        <p className="text-gray-600">
          Share basic details like your restaurant's name, location, cuisine type, and contact information. This
          helps us set up your profile and connect you with hungry customers in your area.
        </p>
      </div>
    </motion.div>

    /* Step 2 */
    <motion.div
      initial={{ opacity: 0, y: 20 }}
      whileInView={{ opacity: 1, y: 0 }}
      viewport={{ once: true }}
      transition={{ delay: 0.6 }}
      className="flex items-start space-x-4"
    >
      <div className="bg-orange-100 p-3 rounded-full flex-shrink-0">
        <span className="text-2xl">2 □</span>
      </div>
      <div>
        <h3 className="text-xl font-semibold mb-2">Upload Your Menu</h3>
        <p className="text-gray-600">
          Add your menu items, prices, and mouthwatering photos. Customize options and set availability to
          showcase what makes your restaurant special—our team can assist with optimization!
        </p>
      </div>
    </motion.div>
  </div>

```

```

    </div>
</motion.div>

/* Step 3 */
<motion.div
  initial={{ opacity: 0, y: 20 }}
  whileInView={{ opacity: 1, y: 0 }}
  viewport={{ once: true }}
  transition={{ delay: 0.8 }}
  className="flex items-start space-x-4"
>
  <div className="bg-orange-100 p-3 rounded-full flex-shrink-0">
    <span className="text-2xl">3</span>
  </div>
  <div>
    <h3 className="text-xl font-semibold mb-2">Access Restaurant Dashboard & Go Live</h3>
    <p className="text-gray-600">
      Once approved, log into your Restaurant Dashboard to manage orders, track performance, and adjust
      settings. You'll be live on FoodyX, ready to serve customers in no time!
    </p>
  </div>
</motion.div>
</div>
</motion.div>
</div>
</section>
);
}

export default ResturentGetStartedSection;

```

src/components/Home/ResturentGrowSection.tsx

```

import { motion } from "framer-motion";

const ResturentGrowSection = () => {
  return (
    <section className="py-20 w-full bg-gray-100 overflow-hidden">
      <div className="max-w-7xl mx-auto px-4">
        /* Header */
        <motion.div
          initial={{ y: -50, opacity: 0 }}
          whileInView={{ y: 0, opacity: 1 }}
          viewport={{ once: true }}
          transition={{ duration: 0.8, ease: "easeOut" }}
          className="text-center mb-12"
>
        <h2 className="text-4xl font-bold text-gray-900 mb-4">
          How FoodyX Works for{" "}
        <span className="text-orange-600">Restaurant Partners</span>
        </h2>
        <p className="text-gray-600 max-w-2xl mx-auto">

```

Partner with FoodyX to reach more customers and grow your business with our seamless delivery platform.

```
</p>
```

```
</motion.div>
```

```
{/* Cards Grid */}
```

```
<div className="grid md:grid-cols-3 gap-8">
```

```
  /* Card 1 - Customers Order */
```

```
<motion.div
```

```
  initial={{ opacity: 0, y: 50 }}
```

```
  whileInView={{ opacity: 1, y: 0 }}
```

```
  viewport={{ once: true }}
```

```
  transition={{ duration: 0.8, delay: 0.2 }}
```

```
  className="bg-white rounded-md p-6 shadow-lg"
```

```
>
```

```
  <motion.div
```

```
    className="bg-orange-100 w-16 h-16 rounded-full flex items-center justify-center mb-4 mx-auto"
```

```
    animate={{ scale: [1, 1.05, 1] }}
```

```
    transition={{ duration: 2, repeat: Infinity }}
```

```
>
```

```
  <span className="text-3xl">●□</span>
```

```
</motion.div>
```

```
  <h3 className="text-xl font-semibold text-center mb-3">Customers Order</h3>
```

```
  <p className="text-gray-600 text-center">
```

```
    A customer finds your restaurant and places an order through the Uber Eats app.
```

```
  </p>
```

```
</motion.div>
```

```
{/* Card 2 - You Prepare */}
```

```
<motion.div
```

```
  initial={{ opacity: 0, y: 50 }}
```

```
  whileInView={{ opacity: 1, y: 0 }}
```

```
  viewport={{ once: true }}
```

```
  transition={{ duration: 0.8, delay: 0.4 }}
```

```
  className="bg-white rounded-md p-6 shadow-lg"
```

```
>
```

```
  <motion.div
```

```
    className="bg-orange-100 w-16 h-16 rounded-full flex items-center justify-center mb-4 mx-auto"
```

```
    animate={{ scale: [1, 1.05, 1] }}
```

```
    transition={{ duration: 2, repeat: Infinity }}
```

```
>
```

```
  <span className="text-3xl">👤🕒</span>
```

```
</motion.div>
```

```
  <h3 className="text-xl font-semibold text-center mb-3">You Prepare</h3>
```

```
  <p className="text-gray-600 text-center">
```

```
    Your restaurant accepts and prepares the order.
```

```
  </p>
```

```
</motion.div>
```

```
{/* Card 3 - Delivery Partners Arrive */}
```

```
<motion.div
```

```
  initial={{ opacity: 0, y: 50 }}
```

```
  whileInView={{ opacity: 1, y: 0 }}
```

```

viewport={{ once: true }}
transition={{ duration: 0.8, delay: 0.6 }}
className="bg-white rounded-md p-6 shadow-lg"
>
<motion.div
  className="bg-orange-100 w-16 h-16 rounded-full flex items-center justify-center mb-4 mx-auto"
  animate={{ scale: [1, 1.05, 1] }}
  transition={{ duration: 2, repeat: Infinity }}
>
  <span className="text-3xl">🚚</span>
</motion.div>
<h3 className="text-xl font-semibold text-center mb-3">Delivery Partners Arrive</h3>
<p className="text-gray-600 text-center">
  Delivery people using the Uber platform pick up the order from your restaurant, then deliver it to the
customer.
</p>
</motion.div>
</div>
</div>
</section>
)
}

```

export default ResturentGrowSection

src/components/Home/ResturentRegistrationSection.tsx

```

import { useState, ChangeEvent } from "react";
import { Link, useNavigate } from "react-router-dom";
import { IoEyeOutline, IoEyeOffOutline } from "react-icons/io5";
import toast, { Toaster } from "react-hot-toast";
import CustomButton from "@/components/UI/CustomButton";
import resturent from "../../assets/images/resturent.webp";
import { registerRestaurant } from "@/utils/api";

interface FormData {
  restaurantName: string;
  contactPerson: string;
  phoneNumber: string;
  businessType: string;
  cuisineType: string;
  operatingHours: string;
  deliveryRadius: string;
  taxId: string;
  streetAddress: string;
  city: string;
  state: string;
  zipCode: string;
  country: string;
  email: string;
  password: string;
  agreeTerms: boolean;
}

```

```

businessLicense: File | null;
foodSafetyCert: File | null;
exteriorPhoto: File | null;
logo: File | null;
}

const ResturentRegistrationSection = () => {
  const navigate = useNavigate();

  const [formData, setFormData] = useState<FormData>({
    restaurantName: "",
    contactPerson: "",
    phoneNumber: "",
    businessType: "",
    cuisineType: "",
    operatingHours: "",
    deliveryRadius: "",
    taxId: "",
    streetAddress: "",
    city: "",
    state: "",
    zipCode: "",
    country: "",
    email: "",
    password: "",
    agreeTerms: false,
    businessLicense: null,
    foodSafetyCert: null,
    exteriorPhoto: null,
    logo: null,
  });

  const [errors, setErrors] = useState<Partial<Record<keyof FormData, string>>>({});

  const [confirmPassword, setConfirmPassword] = useState("");
  const [confirmPasswordError, setConfirmPasswordError] = useState("");
  const [passwordVisible, setPasswordVisible] = useState(false);
  const [confirmPasswordVisible, setConfirmPasswordVisible] = useState(false);
  const [isLoading, setIsLoading] = useState(false);

  // Handle input changes
  const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
    const { name, value, type, files, checked } = e.target;
    if (name === "confirmPassword") {
      setConfirmPassword(value);
      setConfirmPasswordError("");
    } else {
      setFormData((prev) => ({
        ...prev,
        [name]: type === "file" ? files?.[0] || null : type === "checkbox" ? checked : value,
      }));
      setErrors((prev) => ({ ...prev, [name]: "" }));
    }
  };
}

```

```

// Form validation
const validateForm = (): boolean => {
  const newErrors: Partial<Record<keyof FormData, string>> = { };
  let isValid = true;

  if (!formData.restaurantName.trim()) newErrors.restaurantName = "Restaurant name is required";
  if (!formData.contactPerson.trim()) newErrors.contactPerson = "Contact person is required";
  if (!formData.phoneNumber.trim()) newErrors.phoneNumber = "Phone number is required";
  if (!formData.businessType.trim()) newErrors.businessType = "Business type is required";
  if (!formData.cuisineType.trim()) newErrors.cuisineType = "Cuisine type is required";
  if (!formData.operatingHours.trim()) newErrors.operatingHours = "Operating hours are required";
  if (!formData.deliveryRadius.trim()) newErrors.deliveryRadius = "Delivery radius is required";
  if (!formData.taxId.trim()) newErrors.taxId = "Tax ID is required";
  if (!formData.streetAddress.trim()) newErrors.streetAddress = "Street address is required";
  if (!formData.city.trim()) newErrors.city = "City is required";
  if (!formData.state.trim()) newErrors.state = "State is required";
  if (!formData.zipCode.trim()) newErrors.zipCode = "ZIP code is required";
  if (!formData.country.trim()) newErrors.country = "Country is required";
  else if (formData.country.toLowerCase() !== "sri lanka")
    newErrors.country = "Only Sri Lanka is allowed";
  if (!formData.email.trim()) newErrors.email = "Email is required";
  else if (!/^[^s@]+@[^\s@]+\.[^\s@]+\$/test(formData.email))
    newErrors.email = "Invalid email address";
  if (!formData.password) newErrors.password = "Password is required";
  else if (formData.password.length < 6)
    newErrors.password = "Password must be at least 6 characters";
  if (!formData.agreeTerms) newErrors.agreeTerms = "You must agree to the terms";

  if (formData.password !== confirmPassword) {
    setConfirmPasswordError("Passwords do not match");
    isValid = false;
  } else {
    setConfirmPasswordError("");
  }

  setErrors(newErrors);
  if (Object.keys(newErrors).length > 0) isValid = false;
  return isValid;
};

// Handle form submission
const handleFormSubmit = async () => {
  if (!validateForm()) {
    toast.error("Please fix the errors in the form");
    return;
  }

  setIsLoading(true);
  try {
    const submitData = {
      ...formData,
      country: "Sri Lanka", // Enforce country as "Sri Lanka"
    }
  }
}

```

```

};

await registerRestaurant(submitData);
toast.success("Restaurant registered successfully! Please check your email for verification.");
navigate("/signin");
} catch (error: unknown) {
if (error instanceof Response) {
  const data = await error.json();
  if (data.errors && Array.isArray(data.errors)) {
    const newErrors: Partial<Record<keyof FormData, string>> = { };
    data.errors.forEach((err: string) => {
      if (err.includes("Street address")) newErrors.streetAddress = err;
      else if (err.includes("City")) newErrors.city = err;
      else if (err.includes("State")) newErrors.state = err;
      else if (err.includes("Zip code")) newErrors.zipCode = err;
      else if (err.includes("Country")) newErrors.country = err;
      else if (err.includes("Email")) newErrors.email = err;
      else if (err.includes("Password")) newErrors.password = err;
    });
    setErrors(newErrors);
    toast.error("Please fix the errors in the form");
  } else if (data.message) {
    toast.error(data.message);
  } else {
    toast.error("Failed to register restaurant");
  }
} else {
  toast.error("Failed to register restaurant");
}
} finally {
  setIsLoading(false);
}
};

return (
<div className="mx-auto mt-12 flex w-full max-w-[1920px] flex-col lg:flex-row">
  <Toaster position="top-right" reverseOrder={false} />

  {/* Left Image Section */}
  <div className="hidden lg:block lg:w-[55%]">
    <img src={resturent} alt="Background" className="h-full w-full object-cover" />
  </div>

  {/* Form Section */}
  <div className="flex w-full flex-col px-6 py-8 sm:px-8 md:px-12 lg:w-[45%] lg:px-16 lg:py-20 2xl:px-24">
    <div className="mb-6 hidden lg:block">
      <span className="text-4xl font-bold text-orange-600">FoodyX</span>
    </div>

    <div className="flex w-full flex-col">
      <h2 className="text-2xl font-bold leading-9 text-black lg:text-4xl">
        Register Your Restaurant
      </h2>
    </div>
  </div>
</div>

```

```
</h2>
<span className="mt-5 text-sm text-black lg:text-base">
  Provide your restaurant details to join FoodyX.
</span>

/* Form Sections */
<div className="mt-10 space-y-12">
  /* Business Details */
  <div>
    <h3 className="mb-6 text-xl font-semibold text-black">Business Details</h3>
    <div className="grid grid-cols-1 gap-8 sm:grid-cols-2">
      <div>
        <input
          type="text"
          name="restaurantName"
          value={formData.restaurantName}
          onChange={handleChange}
          placeholder="Restaurant Name"
          className="w-full border-b border-black text-sm focus:outline-none"
        />
        {errors.restaurantName && (
          <p className="mt-2 text-xs text-red-500">{errors.restaurantName}</p>
        )}
      </div>
      <div>
        <input
          type="text"
          name="contactPerson"
          value={formData.contactPerson}
          onChange={handleChange}
          placeholder="Contact Person"
          className="w-full border-b border-black text-sm focus:outline-none"
        />
        {errors.contactPerson && (
          <p className="mt-2 text-xs text-red-500">{errors.contactPerson}</p>
        )}
      </div>
      <div>
        <input
          type="tel"
          name="phoneNumber"
          value={formData.phoneNumber}
          onChange={handleChange}
          placeholder="Phone Number"
          className="w-full border-b border-black text-sm focus:outline-none"
        />
        {errors.phoneNumber && (
          <p className="mt-2 text-xs text-red-500">{errors.phoneNumber}</p>
        )}
      </div>
      <div>
        <input
          type="text"
```

```
name="businessType"
value={formData.businessType}
onChange={handleChange}
placeholder="Business Type (e.g., Restaurant, Cafe)"
className="w-full border-b border-black text-sm focus:outline-none"
/>
{errors.businessType && (
  <p className="mt-2 text-xs text-red-500">{errors.businessType}</p>
)}
</div>
<div>
<input
  type="text"
  name="cuisineType"
  value={formData.cuisineType}
  onChange={handleChange}
  placeholder="Cuisine Type (e.g., Italian, Mexican)"
  className="w-full border-b border-black text-sm focus:outline-none"
/>
{errors.cuisineType && (
  <p className="mt-2 text-xs text-red-500">{errors.cuisineType}</p>
)}
</div>
<div className="sm:col-span-2">
<input
  type="text"
  name="operatingHours"
  value={formData.operatingHours}
  onChange={handleChange}
  placeholder="Operating Hours (e.g., Mon-Sun 9 AM - 10 PM)"
  className="w-full border-b border-black text-sm focus:outline-none"
/>
{errors.operatingHours && (
  <p className="mt-2 text-xs text-red-500">{errors.operatingHours}</p>
)}
</div>
<div>
<input
  type="text"
  name="deliveryRadius"
  value={formData.deliveryRadius}
  onChange={handleChange}
  placeholder="Delivery Radius (e.g., 5 miles)"
  className="w-full border-b border-black text-sm focus:outline-none"
/>
{errors.deliveryRadius && (
  <p className="mt-2 text-xs text-red-500">{errors.deliveryRadius}</p>
)}
</div>
<div>
<input
  type="text"
  name="taxId"
```

```
value={formData.taxId}
onChange={handleChange}
placeholder="Tax ID"
className="w-full border-b border-black text-sm focus:outline-none"
/>
{errors.taxId && <p className="mt-2 text-xs text-red-500">{errors.taxId}</p>}
</div>
</div>
</div>

{/* Address Details */}
<div>
<h3 className="mb-6 text-xl font-semibold text-black">Address Details</h3>
<div className="grid grid-cols-1 gap-8 sm:grid-cols-2">
<div className="sm:col-span-2">
<input
  type="text"
  name="streetAddress"
  value={formData.streetAddress}
  onChange={handleChange}
  placeholder="Street Address"
  className="w-full border-b border-black text-sm focus:outline-none"
/>
{errors.streetAddress && (
  <p className="mt-2 text-xs text-red-500">{errors.streetAddress}</p>
)}
</div>
<div>
<input
  type="text"
  name="city"
  value={formData.city}
  onChange={handleChange}
  placeholder="City"
  className="w-full border-b border-black text-sm focus:outline-none"
/>
{errors.city && <p className="mt-2 text-xs text-red-500">{errors.city}</p>}
</div>
<div>
<input
  type="text"
  name="state"
  value={formData.state}
  onChange={handleChange}
  placeholder="State/Province"
  className="w-full border-b border-black text-sm focus:outline-none"
/>
{errors.state && <p className="mt-2 text-xs text-red-500">{errors.state}</p>}
</div>
<div>
<input
  type="text"
  name="zipCode"
```

```

        value={formData.zipCode}
        onChange={handleChange}
        placeholder="ZIP/Postal Code"
        className="w-full border-b border-black text-sm focus:outline-none"
      />
      {errors.zipCode && (
        <p className="mt-2 text-xs text-red-500">{errors.zipCode}</p>
      )}
    </div>
    <div>
      <input
        type="text"
        name="country"
        value={formData.country}
        onChange={handleChange}
        placeholder="Country"
        className="w-full border-b border-black text-sm focus:outline-none"
      />
      {errors.country && (
        <p className="mt-2 text-xs text-red-500">{errors.country}</p>
      )}
    </div>
  </div>
</div>

{/* Account Setup */}
<div>
  <h3 className="mb-6 text-xl font-semibold text-black">Account Setup</h3>
  <div className="grid grid-cols-1 gap-8 sm:grid-cols-2">
    <div className="sm:col-span-2">
      <input
        type="email"
        name="email"
        value={formData.email}
        onChange={handleChange}
        placeholder="Email Address"
        className="w-full border-b border-black text-sm focus:outline-none"
      />
      {errors.email && <p className="mt-2 text-xs text-red-500">{errors.email}</p>}
    </div>
    <div className="relative sm:col-span-2">
      <input
        type={passwordVisible ? "text" : "password"}
        name="password"
        value={formData.password}
        onChange={handleChange}
        placeholder="Create Password"
        className="w-full border-b border-black text-sm focus:outline-none"
      />
      <div
        className="absolute right-4 top-1/2 -translate-y-1/2 cursor-pointer"
        onClick={() => setPasswordVisible(!passwordVisible)}
      >

```

```

{passwordVisible ? (
  <IoEyeOutline size={20} color="#646464" />
) : (
  <IoEyeOffOutline size={20} color="#646464" />
)}
</div>
{errors.password && (
  <p className="mt-2 text-xs text-red-500">{errors.password}</p>
)}
</div>
<div className="relative sm:col-span-2">
<input
  type={confirmPasswordVisible ? "text" : "password"}
  name="confirmPassword"
  value={confirmPassword}
  onChange={handleChange}
  placeholder="Confirm Password"
  className="w-full border-b border-black text-sm focus:outline-none"
/>
<div
  className="absolute right-4 top-1/2 -translate-y-1/2 cursor-pointer"
  onClick={() => setConfirmPasswordVisible(!confirmPasswordVisible)}
>
  {confirmPasswordVisible ? (
    <IoEyeOutline size={20} color="#646464" />
  ) : (
    <IoEyeOffOutline size={20} color="#646464" />
  )}
</div>
{confirmPasswordError && (
  <p className="mt-2 text-xs text-red-500">{confirmPasswordError}</p>
)}
</div>
<div className="sm:col-span-2">
<label className="flex items-center text-sm">
<input
  type="checkbox"
  name="agreeTerms"
  checked={formData.agreeTerms}
  onChange={handleChange}
  className="mr-2"
/>
  I agree to the <Link to="/terms-condition"><span className="text-orange-500 hover:text-orange-700 mx-2"> Terms and Conditions </span></Link>
</label>
{errors.agreeTerms && (
  <p className="mt-2 text-xs text-red-500">{errors.agreeTerms}</p>
)}
</div>
</div>
</div>

{/* Documentation */}

```

```
<div>
  <h3 className="mb-6 text-xl font-semibold text-black">Documentation</h3>
  <div className="grid grid-cols-1 gap-8 sm:grid-cols-2">
    <div className="sm:col-span-2">
      <label className="text-sm">Business License (optional)</label>
      <input
        type="file"
        name="businessLicense"
        onChange={handleChange}
        className="mt-2 w-full text-sm"
      />
      {errors.businessLicense && (
        <p className="mt-2 text-xs text-red-500">{errors.businessLicense}</p>
      )}
    </div>
    <div className="sm:col-span-2">
      <label className="text-sm">Food Safety Certificate (optional)</label>
      <input
        type="file"
        name="foodSafetyCert"
        onChange={handleChange}
        className="mt-2 w-full text-sm"
      />
      {errors.foodSafetyCert && (
        <p className="mt-2 text-xs text-red-500">{errors.foodSafetyCert}</p>
      )}
    </div>
    <div className="sm:col-span-2">
      <label className="text-sm">Restaurant Exterior Photo (optional)</label>
      <input
        type="file"
        name="exteriorPhoto"
        onChange={handleChange}
        className="mt-2 w-full text-sm"
      />
      {errors.exteriorPhoto && (
        <p className="mt-2 text-xs text-red-500">{errors.exteriorPhoto}</p>
      )}
    </div>
    <div className="sm:col-span-2">
      <label className="text-sm">Logo (optional)</label>
      <input
        type="file"
        name="logo"
        onChange={handleChange}
        className="mt-2 w-full text-sm"
      />
      {errors.logo && <p className="mt-2 text-xs text-red-500">{errors.logo}</p>}
    </div>
  </div>
</div>
</div>
```

```

{/* Submit Button */}
<div className="mt-12">
  <CustomButton
    title={isLoading ? "Registering..." : "Register Restaurant"}
    bgColor="bg-orange-600"
    textColor="text-white"
    onClick={handleFormSubmit}
    style="hover:bg-orange-700"
    disabled={isLoading}
  />
</div>

{/* Sign In Link */}
<h1 className="mt-8 text-center text-sm text-gray-600">
  Already have an account?{" "}
  <span
    className="cursor-pointer font-semibold text-orange-600 hover:underline"
    onClick={() => navigate("/signin")}
  >
    Sign In
  </span>
</h1>
</div>

{/* Footer */}
<div className="mt-12 py-6 text-center text-xs text-black">
  2025 © All rights reserved. FoodyX
</div>
</div>
</div>
);

};

export default ResturentRegistrationSection;

```

src/components/Home/ReviewSection.tsx

```

import { Swiper, SwiperSlide } from "swiper/react";
import { Pagination, Autoplay } from "swiper/modules";
import { motion } from "framer-motion";
import "swiper/css";
import "swiper/css/pagination";

const reviews = [
  {
    id: 1,
    name: "Sarah Johnson",
    image: "Home/avatar1.png",
    rating: 5,
    review:
      "The delivery was super fast and the food arrived hot. Exactly what I was looking for!",
    role: "Food Enthusiast",
  }
];

```

```

},
{
  id: 2,
  name: "Michael Chen",
  image: "Home/avatar1.png",
  rating: 5,
  review:
    "Great variety of restaurants and excellent customer service. My go-to food delivery app!",
  role: "Regular Customer",
},
{
  id: 3,
  name: "Emily Rodriguez",
  image: "Home/avatar1.png",
  rating: 4,
  review:
    "The tracking feature is amazing! I always know exactly when my food will arrive.",
  role: "Food Blogger",
},
{
  id: 4,
  name: "David Kim",
  image: "Home/avatar1.png",
  rating: 5,
  review:
    "Outstanding service and the app is so easy to use. Highly recommended!",
  role: "Tech Professional",
},
];

```

```

function ReviewsSection() {
  return (
    <section className="py-10 w-full bg-gray-50">
      <div className="max-w-7xl mx-auto px-4">
        <motion.div
          initial={{ opacity: 0, y: 20 }}
          whileInView={{ opacity: 1, y: 0 }}
          viewport={{ once: true }}
          className="text-center mb-12"
        >
          <h2 className="text-4xl font-bold text-gray-900 mb-4">
            What Our Customers Say
          </h2>
          <p className="text-gray-600 max-w-2xl mx-auto">
            Read genuine reviews from our satisfied customers about their food
            delivery experience
          </p>
        </motion.div>

        <Swiper
          modules={[Pagination, Autoplay]}
          spaceBetween={30}
          slidesPerView={1}
        >

```

```

pagination={{
  clickable: true,
  bulletClass: "swiper-pagination-bullet custom-bullet",
  bulletActiveClass:
    "swiper-pagination-bullet-active custom-bullet-active",
}}
autoplay={{
  delay: 3000,
  disableOnInteraction: false,
}}
breakpoints={{
  640: {
    slidesPerView: 2,
  },
  1024: {
    slidesPerView: 3,
  },
}}
className="pb-12 custom-swiper"
>
{reviews.map((review) => (
  <SwiperSlide key={review.id}>
    <motion.div
      initial={{ opacity: 0, y: 20 }}
      whileInView={{ opacity: 1, y: 0 }}
      viewport={{ once: true }}
      className="bg-white p-8 rounded-2xl shadow-lg h-[250px]"
    >
      <div className="flex items-center mb-6">
        <img
          src={review.image}
          alt={review.name}
          className="w-12 h-12 rounded-full object-cover"
        />
        <div className="ml-4">
          <h3 className="font-semibold text-lg">{review.name}</h3>
          <p className="text-gray-500 text-sm">{review.role}</p>
        </div>
      </div>
      <div className="mb-4">
        {[...Array(review.rating)].map((_, i) => (
          <span key={i} className="text-yellow-400">
            ★
          </span>
        )))
      </div>
      <p className="text-gray-600 italic">"{review.review}"</p>
    </motion.div>
  </SwiperSlide>
))
</Swiper>
</div>
</section>

```

```
    );
}
```

```
export default ReviewsSection;
```

src/components/Payment/Button.tsx

```
// Button.tsx
import React from 'react';
import { cn } from '@/lib/utils';

type ButtonProps = {
  children: React.ReactNode;
  variant?: 'primary' | 'secondary' | 'outline';
  className?: string;
  onClick?: () => void;
  disabled?: boolean;
  type?: 'button' | 'submit' | 'reset';
};

export const Button = ({  
  children,  
  variant = 'primary',  
  className,  
  onClick,  
  disabled = false,  
  type = 'button',  
}: ButtonProps) => {  
  const baseStyles = 'px-4 py-2 rounded-lg font-semibold transition-all';  
  const variantStyles = {  
    primary: 'bg-orange-500 text-white hover:bg-orange-600',  
    secondary: 'bg-gray-200 text-black hover:bg-gray-300',  
    outline: 'border border-orange-500 text-orange-500 hover:bg-orange-100',  
  };  
  
  return (  
    <button  
      type={type}  
      onClick={onClick}  
      disabled={disabled}  
      className={cn(baseStyles, variantStyles[variant], className, {  
        'opacity-50 cursor-not-allowed': disabled,  
      })}  
    >  
    {children}  
  </button>  
);  
};
```

src/components/Payment/CardDetails.tsx

```
// CardDetails.tsx
import React, { useState } from 'react';
import { Button } from './Button';
import { useNavigate } from 'react-router-dom'; // Importing useNavigate from react-router-dom
import { FaCcVisa, FaCcMastercard, FaCcAmex, FaCcDiscover } from 'react-icons/fa'; // Importing credit card icons

const CardDetails = () => {
  const [cardNumber, setCardNumber] = useState("");
  const [expiryDate, setExpiryDate] = useState("");
  const [cvv, setCvv] = useState("");
  const [cardType, setCardType] = useState('visa'); // State for card type
  const [isValid, setIsValid] = useState(true); // State for form validation
  const navigate = useNavigate(); // Initialize useNavigate hook

  const handleCardTypeChange = (e: React.ChangeEvent<HTMLSelectElement>) => {
    setCardType(e.target.value);
  };

  const handleSubmit = () => {
    // Basic validation before submitting
    if (!cardNumber || !expiryDate || !cvv) {
      setIsValid(false);
      return;
    }

    console.log("Card details submitted:", { cardNumber, expiryDate, cvv, cardType });
    // Redirect to order confirmation page after submitting card details
    navigate("/order-confirmation"); // Use navigate for routing
  };
}

return (
  <div className="p-4">
    <h2 className="text-lg font-bold mb-4">Enter Card Details</h2>
    <div className="space-y-4">
      <label className="block text-sm font-medium text-gray-700">Card Number</label>
      <input
        type="text"
        className="w-full p-2 border border-gray-300 rounded-md"
        value={cardNumber}
        onChange={(e) => setCardNumber(e.target.value)}
        placeholder="Enter your card number"
      />

      <div className="flex space-x-4">
        <div className="w-1/2">
          <label className="block text-sm font-medium text-gray-700">Expiry Date</label>
          <input
            type="text"
            className="w-full p-2 border border-gray-300 rounded-md"
            value={expiryDate}
          />
        </div>
      </div>
    </div>
  </div>
)
```

```

        onChange={(e) => setExpiryDate(e.target.value)}
        placeholder="MM/YY"
      />
    </div>
    <div className="w-1/2">
      <label className="block text-sm font-medium text-gray-700">CVV</label>
      <input
        type="text"
        className="w-full p-2 border border-gray-300 rounded-md"
        value={cvv}
        onChange={(e) => setCvv(e.target.value)}
        placeholder="CVV"
      />
    </div>
  </div>

  <div>
    <label htmlFor="cardType" className="block text-gray-700 font-medium mb-2">
      Select Card Type
    </label>
    <select
      id="cardType"
      value={cardType}
      onChange={handleCardTypeChange}
      className="w-full p-3 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"
    >
      <option value="visa">Visa</option>
      <option value="mastercard">MasterCard</option>
      <option value="amex">American Express</option>
      <option value="discover">Discover</option>
    </select>
  </div>

  /* Card Icon Display */
  <div className="flex items-center gap-4 mt-4">
    {cardType === 'visa' && <FaCcVisa className="text-blue-600 text-3xl" />}
    {cardType === 'mastercard' && <FaCcMastercard className="text-red-600 text-3xl" />}
    {cardType === 'amex' && <FaCcAmex className="text-blue-500 text-3xl" />}
    {cardType === 'discover' && <FaCcDiscover className="text-orange-500 text-3xl" />}
  </div>

  /* Validation Message */
  {!isValid &&
    <div className="text-red-500 text-sm mt-2">
      Please ensure all fields are filled out correctly.
    </div>
  }
</div>

<Button
  className="mt-6 w-full bg-green-500"
  onClick={handleSubmit}
>
```

```
        Confirm Payment
    </Button>
</div>
);
};

export default CardDetails;
```

src/components/Payment/PaymentMethod.tsx

```
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { Button } from './Button';
import { FaCcVisa, FaCcMastercard, FaCcAmex, FaCcDiscover } from 'react-icons/fa';
import { FiShoppingCart, FiInfo } from 'react-icons/fi';

const PaymentMethod = () => {
    const [selectedPaymentMethod, setSelectedPaymentMethod] = useState('cash');
    const navigate = useNavigate();
    const [isOnlinePayment, setIsOnlinePayment] = useState(false);
    const [couponCode, setCouponCode] = useState("");
    const [tipAmount, setTipAmount] = useState(0);

    const handlePaymentMethodChange = (method: string) => {
        setSelectedPaymentMethod(method);
        if (method === 'payOnline') {
            navigate('/card-details');
        } else {
            setIsOnlinePayment(false);
        }
    };

    const handleApplyCoupon = () => {
        // Logic to apply coupon
        console.log(`Coupon Applied: ${couponCode}`);
    };

    return (
        <div className="flex flex-col md:flex-row gap-8 p-15 bg-gray-50 my-20 mx-5 mt-20">
            {/* Left Sidebar: Checkout/Order Details */}
            <div className="flex-1 bg-white shadow-lg p-6 rounded-xl">
                <div className="flex justify-between items-center mb-6">
                    <h2 className="text-2xl font-semibold text-gray-800">FoodX </h2>
                    <div className="flex items-center gap-4">
                        <div className="relative">
                            <FiShoppingCart className="text-xl text-gray-800" />
                            <span className="absolute top-0 right-0 bg-red-500 text-white text-xs rounded-full w-5 h-5">3</span>
                        </div>
                        <FiInfo className="text-xl text-gray-800" />
                    </div>
                </div>
            </div>
        </div>
    );
}
```

```

/* Payment Method Section */
<h3 className="text-xl font-semibold text-gray-800 mb-4">Payment Method</h3>
<div className="space-y-4">
  <label
    className={`flex items-center gap-3 p-4 border rounded-lg cursor-pointer
    ${selectedPaymentMethod === 'cash' ? 'border-blue-500 bg-blue-50' : 'border-gray-300'}`}>
    onClick={() => handlePaymentMethodChange('cash')}
  >
    <input
      type="radio"
      name="payment"
      checked={selectedPaymentMethod === 'cash'}
      className="h-5 w-5 text-blue-500"
    />
    <span className="text-lg font-medium text-gray-700">Cash</span>
  </label>
  <label
    className={`flex items-center gap-3 p-4 border rounded-lg cursor-pointer
    ${selectedPaymentMethod === 'cardAtPickup' ? 'border-blue-500 bg-blue-50' : 'border-gray-300'}`}>
    onClick={() => handlePaymentMethodChange('cardAtPickup')}
  >
    <input
      type="radio"
      name="payment"
      checked={selectedPaymentMethod === 'cardAtPickup'}
      className="h-5 w-5 text-blue-500"
    />
    <span className="text-lg font-medium text-gray-700">Card at Pickup Counter</span>
  </label>
  <label
    className={`flex items-center gap-3 p-4 border rounded-lg cursor-pointer
    ${selectedPaymentMethod === 'callBack' ? 'border-blue-500 bg-blue-50' : 'border-gray-300'}`}>
    onClick={() => handlePaymentMethodChange('callBack')}
  >
    <input
      type="radio"
      name="payment"
      checked={selectedPaymentMethod === 'callBack'}
      className="h-5 w-5 text-blue-500"
    />
    <span className="text-lg font-medium text-gray-700">Call Me Back</span>
  </label>

  /* Pay Online Option */
  <div className="border border-gray-300 rounded-lg p-4">
    <label
      className={`flex items-center gap-3 p-4 border rounded-lg cursor-pointer
      ${selectedPaymentMethod === 'payOnline' ? 'border-blue-500 bg-blue-50' : 'border-gray-300'}`}>
      onClick={() => handlePaymentMethodChange('payOnline')}
    >
      <input
        type="radio"

```

```

        name="payment"
        checked={selectedPaymentMethod === 'payOnline'}
        className="h-5 w-5 text-blue-500"
    />
    <span className="text-lg font-medium text-gray-700">Pay Online</span>
</label>
{isOnlinePayment && (
    <div className="flex gap-4 mt-4 text-2xl">
        <FaCcVisa className="text-blue-600" />
        <FaCcMastercard className="text-red-600" />
        <FaCcAmex className="text-blue-500" />
        <FaCcDiscover className="text-orange-500" />
    </div>
)
</div>

</div>

{/* Tip Amount */}
<div className="mt-4">
    <h4 className="text-lg font-semibold text-gray-800 mb-2">Tip?</h4>
    <input
        type="number"
        value={tipAmount}
        onChange={(e) => setTipAmount(Number(e.target.value))}
        placeholder="Enter tip amount"
        className="w-full p-3 border rounded-lg text-lg text-gray-700"
    />
</div>

<Button className="mt-6 w-full bg-green-500 text-white py-3 rounded-lg hover:bg-green-600">
    Save
</Button>
</div>

{/* Right Sidebar: Order Summary */}
<div className="flex-1 bg-white shadow-lg p-6 rounded-xl">
    <h3 className="text-xl font-semibold text-gray-800 mb-4">Order Summary</h3>

    {/* Order Items */}
    <div className="space-y-4 mb-4">
        <div className="flex justify-between text-gray-700">
            <span>1x Cheesecake (Strawberry)</span>
            <span>$4.99</span>
        </div>
        <div className="flex justify-between text-gray-700">
            <span>1x Chicken Wings (Lemon Pepper, Ranch Dressing)</span>
            <span>$9.99</span>
        </div>
        <div className="flex justify-between border-t pt-2 text-gray-700 font-semibold">
            <span>Sub-Total</span>
            <span>$14.98</span>
        </div>
    </div>
</div>
```

```

    </div>
    <div className="flex justify-between text-gray-700">
      <span>Tip</span>
      <span>${tipAmount}</span>
    </div>
    <div className="flex justify-between text-xl font-bold border-t pt-2 text-gray-800">
      <span>Total</span>
      <span>${(14.98 + tipAmount).toFixed(2)}</span>
    </div>
  </div>

  {/* Coupon Code */}
  <div className="mt-4">
    <input
      type="text"
      value={couponCode}
      onChange={(e) => setCouponCode(e.target.value)}
      placeholder="Enter Coupon Code"
      className="w-full p-3 border rounded-lg text-lg text-gray-700"
    />
    <Button
      onClick={handleApplyCoupon}
      className="mt-2 w-full bg-blue-500 text-white py-3 rounded-lg hover:bg-blue-600"
    >
      Apply Coupon
    </Button>
  </div>

  <div className="mt-6 text-center text-gray-600">
    <span className="text-sm">
      By placing your order, you agree to our <a href="#">Terms and Conditions</a> and <a href="#">Privacy Policy</a>.
    </span>
  </div>

  {/* Place Order Button */}
  <Button className="mt-6 w-full bg-orange-500 text-white py-3 rounded-lg hover:bg-orange-600">
    Place Pickup Order Now
  </Button>
</div>
</div>
);

};

export default PaymentMethod;

```

src/components/restaurants/CategoryTable.tsx

```
import React, { useState } from 'react';
import { addCategory } from '../utils/api';
import Message from '../UI/FormSuccessMessage';
import FormErrorMessage from '../UI/FormErrorMessage';

export interface Category {
  _id: string;
  name: string;
  description: string;
  restaurantId: string;
  createdAt: string;
  updatedAt: string;
}

interface CategoryTableProps {
  headers: string[];
  data: Category[];
  setCategories: React.Dispatch<React.SetStateAction<Category[]>>;
  restaurantId: string;
}

const AddCategories: React.FC<CategoryTableProps> = ({ headers, data, setCategories, restaurantId }) => {
  const [categories, setLocalCategories] = useState<Category[]>(data);
  const [searchTerm, setSearchTerm] = useState<string>("");
  const [newCategory, setNewCategory] = useState({ name: "", description: "" });
  const [formErrors, setFormErrors] = useState<Record<string, string>>({});
  const [submitError, setSubmitError] = useState<string | null>(null);

  // Handle new category input changes
  const handleNewCategoryChange = (e: React.ChangeEvent<HTMLInputElement | HTMLTextAreaElement>)
  => {
    const { name, value } = e.target;
    setNewCategory((prev) => ({ ...prev, [name]: value }));
    setFormErrors((prev) => ({ ...prev, [name]: "" }));
  };

  // Validate form fields
  const validateForm = (): boolean => {
    const errors: Record<string, string> = {};

    if (!newCategory.name.trim()) {
      errors.name = 'Name is required';
    }
    if (newCategory.description && !newCategory.description.trim()) {
      errors.description = 'Description cannot be empty';
    }
    setFormErrors(errors);
    return Object.keys(errors).length === 0;
  };

  // Handle adding a new category
  const handleAddCategory = () => {
    addCategory(newCategory).then((category) => {
      setLocalCategories((prev) => [...prev, category]);
      setNewCategory({ name: "", description: "" });
      setFormErrors({});
      setSubmitError(null);
      Message.show(`Category ${category.name} added successfully!`);
    }).catch((error) => {
      setSubmitError(error.message);
      setFormErrors({});
    });
  };
}
```

```

const handleAddCategory = async (e: React.FormEvent) => {
  e.preventDefault();
  if (!validateForm()) {
    return;
  }
  try {
    const response = await addCategory(restaurantId, newCategory);
    const addedCategory: Category = {
      _id: response.category._id,
      name: response.category.name,
      description: response.category.description || '',
      restaurantId: response.category.restaurantId,
      createdAt: response.category.createdAt,
      updatedAt: response.category.updatedAt,
    };
    const updateCategories = (prev: Category[]) => [...prev, addedCategory];
    setLocalCategories(updateCategories);
    setCategories(updateCategories);
    setNewCategory({ name: '', description: '' });
    setFormErrors({});
    setSubmitError(null);
  } catch (error: any) {
    if (error.response?.status === 400 && error.response.data.errors) {
      const backendErrors: Record<string, string> = {};
      error.response.data.errors.forEach(({ param, msg }: { param: string; msg: string }) => {
        backendErrors[param] = msg;
      });
      setFormErrors(backendErrors);
      setSubmitError('Please correct the errors in the form.');
    } else {
      setSubmitError(error.message || 'Failed to add category');
      console.error('Add category error:', error);
    }
  }
};

// Map headers to data fields
const getFieldValue = (category: Category, header: string) => {
  switch (header.toLowerCase()) {
    case 'name':
      return category.name;
    case 'description':
      return category.description || '-';
    default:
      return '';
  }
};

// Filter categories by search term
const filteredCategories = categories.filter((category) =>
  [category.name, category.description].some((value) =>
    value.toLowerCase().includes(searchTerm.toLowerCase())
)
);

```

```

);

// Sort categories by name
const sortedCategories = [...filteredCategories].sort((a, b) =>
  a.name.toLowerCase().localeCompare(b.name.toLowerCase())
);

return (
  <>
  {/* Add Category Form */}
  <div className="p-5 max-w-lg mx-auto">
    {submitError && (
      <Message
        type="error"
        message={submitError}
        onClose={() => setSubmitError(null)}
      />
    )}
    <form onSubmit={handleAddCategory} className="space-y-4">
      <div>
        <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
          Name
        </label>
        <input
          type="text"
          name="name"
          value={newCategory.name}
          onChange={handleNewCategoryChange}
          required
          className={`w-full px-3 py-2 border rounded-lg bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-orange-500 ${
            formErrors.name ? 'border-red-500' : 'border-gray-300 dark:border-gray-500'
          }`}
        />
        <FormErrorMessage error={formErrors.name} />
      </div>
      <div>
        <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
          Description
        </label>
        <textarea
          name="description"
          value={newCategory.description}
          onChange={handleNewCategoryChange}
          className={`w-full px-3 py-2 border rounded-lg bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-orange-500 ${
            formErrors.description ? 'border-red-500' : 'border-gray-300 dark:border-gray-500'
          }`}
          rows={4}
        />
        <FormErrorMessage error={formErrors.description} />
      </div>
      <div className="flex justify-end">

```

```

<button
  type="submit"
  className="px-4 py-2 bg-orange-500 text-white rounded-lg hover:bg-orange-600 focus:outline-none
focus:ring-2 focus:ring-orange-500">
  <>
    Add Category
  </button>
</div>
</form>
</div>

{/* Search Bar */}
<div className="p-5 flex justify-between items-center">
<div className="relative w-full sm:w-64">
  <input
    type="text"
    placeholder="Search categories..."
    value={searchTerm}
    onChange={(e) => setSearchTerm(e.target.value)}
    className="w-full pl-10 pr-4 py-2 rounded-full border border-gray-300 dark:border-gray-500 bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
  />
  <svg
    className="absolute left-3 top-1/2 transform -translate-y-1/2 h-5 w-5 text-gray-400 dark:text-gray-300"
    fill="none"
    stroke="currentColor"
    viewBox="0 0 24 24"
    xmlns="http://www.w3.org/2000/svg"
  >
    <path
      strokeLinecap="round"
      strokeLinejoin="round"
      strokeWidth="2"
      d="M21 21l-6-6m2-5a7 7 0 11-14 0 7 7 0 0114 0z"
    />
  </svg>
</div>
</div>

{/* Desktop Table View */}
<div className="hidden lg:block p-5 w-full text-sm rounded-md overflow-x-auto bg-white dark:bg-gray-
700">
  <table className="min-w-[1000px] w-full">
    <thead className="bg-gray-200 border-gray-200 dark:bg-gray-700 dark:border-gray-500">
      <tr>
        {headers.map((header, index) => (
          <th
            key={index}
            className="text-start py-4 px-6 font-semibold text-gray-600 dark:text-gray-200"
          >
            {header}
          </th>
        )))
      </tr>
    </thead>
    <tbody>
      {rows.map((row, index) => (
        <tr key={index}>
          {row}
        </tr>
      )))
    </tbody>
  </table>
</div>

```

```

        </tr>
    </thead>
    <tbody>
        {sortedCategories.map((category, index) => (
            <tr
                key={category._id}
                className={`${`border-b border-gray-200 dark:border-gray-500 ${{
                    index % 2 === 0 ? 'bg-gray-50 dark:bg-gray-600' : 'bg-gray-100 dark:bg-gray-700'
                }} hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors`}
            >
                {headers.map((header, idx) => (
                    <td key={idx} className="py-4 px-6 text-gray-800 dark:text-white">
                        {getFieldValue(category, header)}
                    </td>
                )))
            </tr>
        )))
    </tbody>
</table>
</div>
/* Mobile Grid View */
<div className="lg:hidden p-5 space-y-4 overflow-y-scroll h-full">
    {sortedCategories.map((category) => (
        <div
            key={category._id}
            className="border rounded-md p-4 shadow-md hover:shadow-lg transition-shadow border-gray-200 dark:border-gray-500 bg-white dark:bg-gray-600"
        >
            <div className="space-y-2">
                {headers.map((header, idx) => (
                    <div key={idx} className="flex justify-between items-center py-1">
                        <span className="font-medium text-gray-800 dark:text-white">{header}</span>
                        <span className="text-sm text-gray-600 dark:text-gray-300">
                            {getFieldValue(category, header)}
                        </span>
                    </div>
                )))
            </div>
        </div>
    )))
</div>
/* No Results Message */
{sortedCategories.length === 0 && (
    <div className="p-5 text-center text-gray-600 dark:text-gray-300">
        No categories found.
    </div>
)}
</>
);
};

export default AddCategories;

```

src/components/restaurants/EditMenuItemModal.tsx

```
import { useState } from 'react';
import { updateMenuItem } from '../utils/api';
import CategorySelect from '../UI/CategorySelect';
import ErrorMessage from '../UI/FormErrorMessage';

interface MenuItem {
  id: string;
  name: string;
  description: string;
  price: number;
  category: string; // Category _id
  imageUrl?: string;
  isAvailable: boolean;
}

interface EditMenuItemModalProps {
  item: MenuItem;
  restaurantId: string;
  onClose: () => void;
  onSave: (updatedItem: MenuItem) => void;
}

const EditMenuItemModal = ({ item, restaurantId, onClose, onSave }: EditMenuItemModalProps) => {
  const [formData, setFormData] = useState({
    name: item.name,
    description: item.description,
    price: item.price,
    category: item.category,
    isAvailable: item.isAvailable,
  });
  const [mainImage, setMainImage] = useState<File | null>(null);
  const [thumbnailImage, setThumbnailImage] = useState<File | null>(null);
  const [errors, setErrors] = useState<Partial<Record<keyof typeof formData | 'mainImage' | 'thumbnailImage', string>>>({ });
  const [submitError, setSubmitError] = useState<string | null>(null);

  const validateField = (name: string, value: any): string | undefined => {
    switch (name) {
      case 'name':
        if (!value.trim()) return 'Name is required';
        break;
      case 'description':
        if (!value.trim()) return 'Description is required';
        break;
      case 'price':
        if (value === "" || isNaN(value) || value < 0) return 'Price must be a positive number';
        break;
      case 'category':
        if (!value) return 'Category is required';
        break;
      case 'mainImage':
```

```

if (value) {
  const validTypes = ['image/png', 'image/jpeg', 'image/jpg'];
  if (!validTypes.includes(value.type)) return 'Main image must be a PNG, JPEG, or JPG';
  if (value.size > 5242880) return 'Main image must be less than 5MB';
}
break;
case 'thumbnailImage':
if (value) {
  const validTypes = ['image/png', 'image/jpeg', 'image/jpg'];
  if (!validTypes.includes(value.type)) return 'Thumbnail image must be a PNG, JPEG, or JPG';
  if (value.size > 5242880) return 'Thumbnail image must be less than 5MB';
}
break;
default:
break;
}
return undefined;
};

const validateForm = (): boolean => {
const newErrors: Partial<Record<keyof typeof formData | 'mainImage' | 'thumbnailImage', string>> = { };

// Validate text fields
Object.entries(formData).forEach(([name, value]) => {
  const error = validateField(name, value);
  if (error) newErrors[name as keyof typeof formData] = error;
});

// Validate file fields
if (mainImage) {
  const error = validateField('mainImage', mainImage);
  if (error) newErrors.mainImage = error;
}
if (thumbnailImage) {
  const error = validateField('thumbnailImage', thumbnailImage);
  if (error) newErrors.thumbnailImage = error;
}

setErrors(newErrors);
return Object.keys(newErrors).length === 0;
};

const handleInputChange = (
e: React.ChangeEvent<HTMLInputElement | HTMLTextAreaElement | HTMLSelectElement>
) => {
  const { name, value } = e.target;
  setFormData((prev) => ({ ...prev, [name]: value }));
}

// Validate on change and clear error if valid
const error = validateField(name, value);
setErrors((prev) => ({
  ...prev,
  [name]: error,
})

```

```
}));

};

const handleCheckboxChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const { name, checked } = e.target;
  setFormData((prev) => ({ ...prev, [name]: checked }));
};

const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>, type: 'mainImage' | 'thumbnailImage') => {
  const file = e.target.files?.[0];
  if (file) {
    if (type === 'mainImage') setMainImage(file);
    else setThumbnailImage(file);

    // Validate file and update errors
    const error = validateField(type, file);
    setErrors((prev) => ({
      ...prev,
      [type]: error,
    }));
  } else {
    // Clear file and error if no file is selected
    if (type === 'mainImage') setMainImage(null);
    else setThumbnailImage(null);
    setErrors((prev) => ({
      ...prev,
      [type]: undefined,
    }));
  }
};

const handleBlur = (e: React.FocusEvent<HTMLInputElement | HTMLTextAreaElement | HTMLSelectElement>) => {
  const { name, value } = e.target;
  const error = validateField(name, value);
  setErrors((prev) => ({
    ...prev,
    [name]: error,
  }));
};

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  if (!validateForm()) {
    setSubmitError('Please fix the errors in the form.');
    return;
  }

  try {
    const updatedData = await updateMenuItem(restaurantId, item.id, formData, {
      mainImage: mainImage || undefined,
      thumbnailImage: thumbnailImage || undefined,
    });
  }
};
```

```

});

const updatedItem: MenuItem = {
  id: updatedData.menuItem._id,
  name: updatedData.menuItem.name,
  description: updatedData.menuItem.description,
  price: updatedData.menuItem.price,
  category: updatedData.menuItem.category,
  imageUrl: updatedData.menuItem.mainImage || 'https://via.placeholder.com/300x200',
  isAvailable: updatedData.menuItem.isAvailable,
};

onSave(updatedItem);
 onClose();
} catch (err: any) {
  setSubmitError(err.message || 'Failed to update menu item');
}
};

const handleCloseKeyDown = (e: React.KeyboardEvent) => {
if (e.key === 'Enter' || e.key === ' ') {
  e.preventDefault();
  onClose();
}
};

return (
<div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">
  <div className="bg-white dark:bg-gray-800 p-6 rounded-lg max-w-lg w-full relative">
    /* Cancel Icon */
    <div
      role="button"
      aria-label="Close modal"
      tabIndex={0}
      className="absolute top-4 right-4 text-orange-500 dark:text-gray-300 hover:text-orange-700
dark:hover:text-gray-100 cursor-pointer"
      onClick={onClose}
      onKeyDown={handleCloseKeyDown}
    >
      <svg
        className="h-6 w-6"
        fill="none"
        stroke="currentColor"
        viewBox="0 0 24 24"
        xmlns="http://www.w3.org/2000/svg"
      >
        <path
          strokeLinecap="round"
          strokeLinejoin="round"
          strokeWidth="2"
          d="M6 18L18 6M6 6l12 12"
        />
      </svg>
    
```

```
</div>
<h2 className="text-xl font-semibold mb-4 dark:text-white">Update Your: <span className='text-orange-500'>{formData.name}</span> </h2>
{submitError && <div className="text-red-600 mb-4">{submitError}</div>}
<form onSubmit={handleSubmit}>
<div className="mb-4">
  <label className="block text-sm font-medium dark:text-gray-300">Name</label>
  <input
    type="text"
    name="name"
    value={formData.name}
    onChange={handleInputChange}
    onBlur={handleBlur}
    className={`w-full p-2 border rounded dark:bg-gray-700 dark:text-white ${errors.name ? 'border-red-500' : ''}`}
    required
  />
  <ErrorMessage error={errors.name} />
</div>
<div className="mb-4">
  <label className="block text-sm font-medium dark:text-gray-300">Description</label>
  <textarea
    name="description"
    value={formData.description}
    onChange={handleInputChange}
    onBlur={handleBlur}
    className={`w-full p-2 border rounded dark:bg-gray-700 dark:text-white ${errors.description ? 'border-red-500' : ''}`}
    required
  />
  <ErrorMessage error={errors.description} />
</div>
<div className="mb-4">
  <label className="block text-sm font-medium dark:text-gray-300">Price</label>
  <input
    type="number"
    name="price"
    value={formData.price}
    onChange={handleInputChange}
    onBlur={handleBlur}
    className={`w-full p-2 border rounded dark:bg-gray-700 dark:text-white ${errors.price ? 'border-red-500' : ''}`}
    step="0.01"
    required
  />
  <ErrorMessage error={errors.price} />
</div>
<div className="mb-4">
  <label className="block text-sm font-medium dark:text-gray-300">Category</label>
  <CategorySelect
    restaurantId={restaurantId}
    value={formData.category}
    onChange={handleInputChange}>
```

```

    required
  />
  <ErrorMessage error={errors.category} />
</div>
<div className="mb-4 flex flex-row">

  <input
    type="checkbox"
    name="isAvailable"
    checked={formData.isAvailable}
    onChange={handleCheckboxChange}
    className="h-4 w-4 text-orange-600"
  /> <label className="block text-sm font-medium dark:text-gray-300 ml-3">Available</label>
</div>
<div className="mb-4">
  <label className="block text-sm font-medium dark:text-gray-300">Main Image</label>
  <input
    type="file"
    accept="image/*"
    onChange={(e) => handleFileChange(e, 'mainImage')}
    className={`w-full p-2 border rounded dark:bg-gray-700 dark:text-white ${errors.mainImage ? 'border-red-500' : ''}`}
  />
  <ErrorMessage error={errors.mainImage} />
</div>
<div className="mb-4">
  <label className="block text-sm font-medium dark:text-gray-300">Thumbnail Image</label>
  <input
    type="file"
    accept="image/*"
    onChange={(e) => handleFileChange(e, 'thumbnailImage')}
    className={`w-full p-2 border rounded dark:bg-gray-700 dark:text-white ${errors.thumbnailImage ? 'border-red-500' : ''}`}
  />
  <ErrorMessage error={errors.thumbnailImage} />
</div>
<div className="flex gap-4">
  <button
    type="submit"
    className="flex-1 py-2 bg-orange-500 text-white rounded hover:bg-orange-600"
  >
    Save
  </button>
  <button
    type="button"
    onClick={onClose}
    className="flex-1 py-2 bg-gray-300 text-gray-800 rounded hover:bg-gray-400"
  >
    Cancel
  </button>
</div>
</form>
</div>

```

```
</div>
);
};

export default EditMenuItemModal;
```

src/components/restaurants/Header.tsx

```
import { FaMoon, FaSun, FaPowerOff } from "react-icons/fa";
import { HiOutlineMenuAlt2 } from "react-icons/hi";
import { FiUser } from "react-icons/fi";
import { MdOutlinePowerSettingsNew } from "react-icons/md";
import { useState, useEffect } from "react";
import { Link, useNavigate } from "react-router-dom";
import { useAuth } from "@/context/AuthContext";
import toast, { Toaster } from "react-hot-toast";
import { getRestaurantById, updateRestaurantAvailability } from "@/utils/api";

interface HeaderProps {
  toggleDarkMode: () => void;
  darkMode: boolean;
  toggleSidebar: () => void;
}

const Header = ({ toggleDarkMode, darkMode, toggleSidebar }: HeaderProps) => {
  const { user, isAuthenticated, logout } = useAuth();
  const navigate = useNavigate();
  const [isDropdownOpen, setIsDropdownOpen] = useState(false);
  const [logo, setRestaurantLogo] = useState<string | null>(null);
  const [availability, setAvailability] = useState<boolean>(true);
  const [isToggling, setIsToggling] = useState<boolean>(false);

  // Fetch restaurant logo and availability
  useEffect(() => {
    const fetchRestaurant = async () => {
      try {
        if (!user || !user.id) {
          throw new Error("User not authenticated or missing ID");
        }
        const response = await getRestaurantById(user.id);
        if (!response) {
          throw new Error("Restaurant data not found in response");
        }
        setRestaurantLogo(response.logo || null);
        setAvailability(response.availability ?? true);
      } catch (error: any) {
        console.error('Failed to fetch restaurant data:', error);
        toast.error(error.message || 'Failed to fetch restaurant data');
        setRestaurantLogo(null);
        setAvailability(true); // Fallback to default availability
      }
    };
  });
}
```

```
if (isAuthenticated && user) {
  fetchRestaurant();
}
}, [isAuthenticated, user]);

// Handle availability toggle with optimistic update
const handleToggleAvailability = async () => {
  if (!isAuthenticated || !user) {
    toast.error("You must be logged in to update availability.");
    return;
  }

  setIsToggling(true);
  // Optimistically update the UI
  const previousAvailability = availability;
  const newAvailability = !availability;
  setAvailability(newAvailability);

  try {
    const response = await updateRestaurantAvailability(newAvailability);
    if (!response) {
      throw new Error("Restaurant data not found in response");
    }
    // Confirm server state (optional, since UI is already updated)
    setAvailability(response.availability);
    toast.success(`Restaurant is now ${newAvailability ? 'available' : 'unavailable'}.`);
  } catch (error: any) {
    // Revert to previous state on error
    setAvailability(previousAvailability);
    console.error('Toggle availability error:', error);
    toast.error(error.message || 'Failed to update availability');
  } finally {
    setIsToggling(false);
  }
};

const handleLogout = async () => {
  if (!isAuthenticated) {
    toast.error("You are not logged in.");
    return;
  }

  try {
    await logout();
    toast.success("Logged out successfully!");
    navigate("/");
  } catch (error) {
    console.error("Logout error:", error);
    toast.error("Failed to log out. Please try again.");
  } finally {
    setIsDropdownOpen(false);
  }
};
```

```

// Toggle dropdown visibility
const toggleDropdown = () => {
  setIsDropdownOpen((prev) => !prev);
};

// Close dropdown when clicking outside
const handleBlur = () => {
  setTimeout(() => setIsDropdownOpen(false), 200);
};

return (
  <>
  <Toaster position="top-right" reverseOrder={false} />
  <nav className="fixed top-0 z-50 w-full bg-white border-b border-gray-200 dark:bg-gray-800 dark:border-gray-700">
    <div className="px-3 py-3 lg:px-5 lg:pl-3">
      <div className="flex items-center justify-between">
        <div className="flex items-center justify-start rtl:justify-end">
          <button
            className="inline-flex items-center p-2 text-sm text-gray-500 rounded-lg sm:hidden hover:bg-gray-100 focus:ring-gray-200 focus:outline-none focus:ring-2 dark:text-gray-400 dark:hover:bg-gray-700 dark:focus:ring-gray-600"
            onClick={toggleSidebar}
            aria-label="Toggle sidebar"
          >
            <HiOutlineMenuAlt2 className="text-2xl" />
          </button>
          <Link to="/resturent-dashboard/overview" className="flex items-center ms-2 md:me-24">
            <img
              src={logo || '/images/default-logo.png'}
              alt="Restaurant Logo"
              className="h-8 w-8 rounded-full object-cover mr-2 shrink-0 border border-gray-200 dark:border-gray-600"
            />
          </Link>
        </div>
        <div className="flex items-center space-x-4">
          {/* Availability Toggle Button */}
          {isAuthenticated && user && (
            <button
              onClick={handleToggleAvailability}
              disabled={isToggling}
              className={`flex items-center space-x-2 p-2 rounded-full ${{
                availability
                  ? 'bg-green-100 text-green-600 dark:bg-green-700 dark:text-green-200'
                  : 'bg-red-100 text-red-600 dark:bg-red-700 dark:text-red-200'
                } ${isToggling ? 'opacity-50 cursor-not-allowed' : 'hover:bg-gray-100 dark:hover:bg-gray-700'} `}
              aria-label={availability ? 'Set restaurant unavailable' : 'Set restaurant available'}
            >
              <FaPowerOff className="text-lg" />
              <span className="text-sm font-medium">
                {isToggling ? 'Updating...' : availability ? 'Available' : 'Unavailable'}
              </span>
            </button>
          )}
        </div>
      </div>
    </div>
  </nav>
)

```

```

        </span>
    </button>
)
/* Dark Mode Toggle */
<button
    onClick={toggleDarkMode}
    className="dark:bg-slate-50 dark:text-slate-700 rounded-full p-2"
    aria-label={darkMode ? "Switch to light mode" : "Switch to dark mode"}
>
    {darkMode ? <FaSun /> : <FaMoon className="text-gray-500" />}
</button>
/* User Dropdown */
{isAuthenticated && user && (
    <div className="relative">
        <button
            onClick={toggleDropdown}
            onBlur={handleBlur}
            className="flex items-center space-x-2 p-2 text-gray-500 hover:bg-gray-100 dark:text-gray-400
dark:hover:bg-gray-700 rounded-full focus:outline-none"
            aria-label="User menu"
        >
            <FiUser className="text-xl" />
            <span className="hidden sm:inline text-sm font-medium dark:text-white">
                {user.email || "User"}
            </span>
        </button>
        {isDropdownOpen && (
            <div className="absolute right-0 mt-2 w-48 bg-white dark:bg-gray-800 border border-gray-200
dark:border-gray-700 rounded-md shadow-lg">
                <Link
                    to="/resturent-dashboard/profile"
                    className="flex items-center px-4 py-2 text-sm text-gray-700 dark:text-gray-300 hover:bg-gray-
100 dark:hover:bg-gray-700"
                    onClick={() => setIsDropdownOpen(false)}
                >
                    <FiUser className="mr-2" />
                    Profile
                </Link>
                <button
                    onClick={handleLogout}
                    className="flex items-center w-full px-4 py-2 text-sm text-gray-700 dark:text-gray-300
hover:bg-gray-100 dark:hover:bg-gray-700"
                >
                    <MdOutlinePowerSettingsNew className="mr-2" />
                    Logout
                </button>
            </div>
        )
    )
    </div>
)
</div>
</div>
</div>

```

```
</nav>
</>
);
};

export default Header;
```

src/components/restaurants/ManageCategoryTable.tsx

```
import React, { useState } from 'react';
import { updateCategory, deleteCategory } from '../utils/api';
import ConfirmationModal from '../UI/ConfirmationModal';
import Message from '../UI/FormSuccessMessage';
import FormErrorMessage from '../UI/FormErrorMessage';

export interface Category {
  _id: string;
  name: string;
  description: string;
  restaurantId: string;
  createdAt: string;
  updatedAt: string;
}

interface ManageCategoryTableProps {
  headers: string[];
  data: Category[];
  setCategories: React.Dispatch<React.SetStateAction<Category[]>>;
  restaurantId: string;
}

const ManageCategoryTable: React.FC<ManageCategoryTableProps> = ({ headers, data, setCategories, restaurantId }) => {
  const [categories, setLocalCategories] = useState<Category[]>(data);
  const [searchTerm, setSearchTerm] = useState<string>('');
  const [isEditModalOpen, setIsEditModalOpen] = useState(false);
  const [isDeleteModalOpen, setIsDeleteModalOpen] = useState(false);
  const [selectedCategory, setSelectedCategory] = useState<Category | null>(null);
  const [formData, setFormData] = useState({ name: '', description: '' });
  const [formErrors, setFormErrors] = useState<Record<string, string>>({ });
  const [submitError, setSubmitError] = useState<string | null>(null);

  // Open edit modal and populate form data
  const openEditModal = (category: Category) => {
    setSelectedCategory(category);
    setFormData({
      name: category.name,
      description: category.description || '',
    });
    setFormErrors({});
    setSubmitError(null);
    setIsEditModalOpen(true);
  };

  const handleEdit = () => {
    if (!selectedCategory) return;

    const updatedCategory = { ...selectedCategory, ...formData };
    const newCategories = categories.map((category) => category._id === selectedCategory._id ? updatedCategory : category);

    setLocalCategories(newCategories);
    setCategories(newCategories);
    setSubmitError(null);
    setIsEditModalOpen(false);
  };

  const handleDelete = () => {
    if (!selectedCategory) return;

    deleteCategory(selectedCategory._id).then(() => {
      const newCategories = categories.filter((category) => category._id !== selectedCategory._id);
      setLocalCategories(newCategories);
      setCategories(newCategories);
      setSubmitError(null);
      setIsDeleteModalOpen(false);
    });
  };

  const handleSearch = () => {
    const filteredData = data.filter((category) => category.name.toLowerCase().includes(searchTerm));
    setLocalCategories(filteredData);
  };
}
```

```

};

// Handle modal close
const handleCloseModal = () => {
  setIsEditModalOpen(false);
  setSelectedCategory(null);
  setFormErrors({ });
  setSubmitError(null);
};

// Handle form input changes
const handleInputChange = (e: React.ChangeEvent<HTMLInputElement | HTMLTextAreaElement>) => {
  const { name, value } = e.target;
  setFormData((prev) => ({ ...prev, [name]: value }));
  setFormErrors((prev) => ({ ...prev, [name]: " " }));
};

// Validate form fields
const validateForm = (): boolean => {
  const errors: Record<string, string> = { };

  if (!formData.name.trim()) {
    errors.name = 'Name is required';
  }
  if (formData.description && !formData.description.trim()) {
    errors.description = 'Description cannot be empty';
  }

  setFormErrors(errors);
  return Object.keys(errors).length === 0;
};

// Handle form submission for editing
const handleFormSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  if (!selectedCategory) return;

  if (!validateForm()) {
    return;
  }

  try {
    const response = await updateCategory(restaurantId, selectedCategory._id, formData);
    const updatedCategory: Category = {
      _id: response.category._id,
      name: response.category.name,
      description: response.category.description || "",
      restaurantId: response.category.restaurantId,
      createdAt: response.category.createdAt,
      updatedAt: response.category.updatedAt,
    };
    const updateCategories = (prev: Category[]) =>
      prev.map((cat) => (cat._id === selectedCategory._id ? updatedCategory : cat));
  }
}

```

```

setLocalCategories(updateCategories);
setCategories(updateCategories);
setIsEditModalOpen(false);
setSelectedCategory(null);
setFormErrors({ });
setSubmitError(null);
} catch (error: any) {
if (error.response?.status === 400 && error.response.data.errors) {
  const backendErrors: Record<string, string> = { };
  error.response.data.errors.forEach(({ param, msg }: { param: string; msg: string }) => {
    backendErrors[param] = msg;
  });
  setFormErrors(backendErrors);
  setSubmitError('Please correct the errors in the form.');
} else {
  setSubmitError(error.message || 'Failed to update category');
  console.error('Update category error:', error);
}
}
};

// Handle delete category
const handleDeleteCategory = (categoryId: string, categoryName: string) => {
openDeleteModal(
  'Confirm Delete Category',
  `Are you sure you want to delete ${categoryName}?`,
  async () => {
    try {
      await deleteCategory(restaurantId, categoryId);
      const updateCategories = (prev: Category[]) =>
        prev.filter((cat) => cat._id !== categoryId);
      setLocalCategories(updateCategories);
      setCategories(updateCategories);
      setIsDeleteModalOpen(false);
    } catch (error: any) {
      setSubmitError(`Failed to delete category: ${error.message}`);
      console.error('Delete category error:', error);
    }
  }
);
};

// Open delete confirmation modal
const openDeleteModal = (title: string, message: string, onConfirm: () => void) => {
  setIsDeleteModalOpen(true);
  setModalConfig({ title, message, onConfirm });
};

const [modalConfig, setModalConfig] = useState<{
  title: string;
  message: string;
  onConfirm: () => void;
}>({ title: "", message: "", onConfirm: () => {} });

```

```

// Map headers to data fields
const getFieldValue = (category: Category, header: string) => {
  switch (header.toLowerCase()) {
    case 'name':
      return category.name;
    case 'description':
      return category.description || '-';
    case 'actions':
      return (
        <div className="flex space-x-2">
          <button
            onClick={() => openEditModal(category)}
            className="px-2 py-1 bg-orange-500 text-white rounded-lg hover:bg-orange-600 focus:outline-none
focus:ring-2 focus:ring-orange-500"
          >
            Edit
          </button>
          <button
            onClick={() => handleDeleteCategory(category._id, category.name)}
            className="px-2 py-1 bg-red-500 text-white rounded-lg hover:bg-red-600 focus:outline-none
focus:ring-2 focus:ring-red-500"
          >
            Delete
          </button>
        </div>
      );
    default:
      return "";
  }
};

// Filter categories by search term
const filteredCategories = categories.filter((category) =>
  [category.name, category.description].some((value) =>
    value.toLowerCase().includes(searchTerm.toLowerCase())
  )
);

// Sort categories by name
const sortedCategories = [...filteredCategories].sort((a, b) =>
  a.name.toLowerCase().localeCompare(b.name.toLowerCase())
);

return (
  <>
    {/* Delete Confirmation Modal */}
    <ConfirmationModal
      isOpen={isDeleteModalOpen}
      onClose={() => setIsDeleteModalOpen(false)}
      onConfirm={modalConfig.onConfirm}
      title={modalConfig.title}
      message={modalConfig.message}
)

```

```

confirmText="Delete"
cancelText="Cancel"
/>>

{/* Edit Modal */}
{isEditModalOpen && (
  <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">
    <div className="relative bg-white dark:bg-gray-800 p-6 rounded-lg max-w-lg w-full max-h-[80vh] overflow-y-auto">
      <button
        onClick={handleCloseModal}
        className="absolute top-4 right-4 text-gray-500 dark:text-gray-300 hover:text-gray-700 dark:hover:text-gray-100 focus:outline-none focus:ring-2 focus:ring-orange-500"
        aria-label="Close"
      >
        <svg
          className="h-5 w-5 text-orange-500 hover:text-orange-700"
          fill="none"
          stroke="currentColor"
          viewBox="0 0 24 24"
          xmlns="http://www.w3.org/2000/svg"
        >
          <path
            strokeLinecap="round"
            strokeLinejoin="round"
            strokeWidth="2"
            d="M6 18L18 6M6 6l12 12"
          />
        </svg>
      </button>
      <h2 className="text-xl font-semibold text-gray-800 dark:text-gray-200 mb-4">
        Edit Category: <span className='text-orange-500'>{formData.name}</span>
      </h2>
      {submitError && (
        <Message
          type="error"
          message={submitError}
          onClose={() => setSubmitError(null)}
        />
      )}
      <form onSubmit={handleFormSubmit} className="space-y-4">
        <div>
          <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
            Name
          </label>
          <input
            type="text"
            name="name"
            value={formData.name}
            onChange={handleInputChange}
            required
            className={`w-full px-3 py-2 border rounded-lg bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-orange-500 ${

```

```

    formErrors.name
      ? 'border-red-500'
      : 'border-gray-300 dark:border-gray-500'
    }`}
  />
<FormErrorMessage error={formErrors.name} />
</div>
<div>
  <label className="block text-sm font-medium text-gray-700 dark:text-gray-300">
    Description
  </label>
  <textarea
    name="description"
    value={formData.description}
    onChange={handleInputChange}
    className={`w-full px-3 py-2 border rounded-lg bg-white dark:bg-gray-700 text-gray-800
dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-orange-500 ${

      formErrors.description
      ? 'border-red-500'
      : 'border-gray-300 dark:border-gray-500'
    }`}
    rows={4}
  />
<FormErrorMessage error={formErrors.description} />
</div>
<div className="flex justify-end space-x-2">
  <button
    type="button"
    onClick={handleCloseModal}
    className="px-4 py-2 bg-gray-300 text-gray-800 rounded-lg hover:bg-gray-400 focus:outline-none
focus:ring-2 focus:ring-gray-500"
  >
    Cancel
  </button>
  <button
    type="submit"
    className="px-4 py-2 bg-orange-500 text-white rounded-lg hover:bg-orange-600 focus:outline-none
focus:ring-2 focus:ring-orange-500"
  >
    Save
  </button>
</div>
</form>
</div>
</div>
)}`

/* Search Bar */
<div className="p-5 flex justify-between items-center">
  <div className="relative w-full sm:w-64">
    <input
      type="text"
      placeholder="Search categories...">
  </div>
</div>

```

```

value={searchTerm}
onChange={(e) => setSearchTerm(e.target.value)}
className="w-full pl-10 pr-4 py-2 rounded-full border border-gray-300 dark:border-gray-500 bg-white
dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
/>
<svg
  className="absolute left-3 top-1/2 transform -translate-y-1/2 h-5 w-5 text-gray-400 dark:text-gray-300"
  fill="none"
  stroke="currentColor"
  viewBox="0 0 24 24"
  xmlns="http://www.w3.org/2000/svg"
>
  <path
    strokeLinecap="round"
    strokeLinejoin="round"
    strokeWidth="2"
    d="M21 21l-6-6m2-5a7 7 0 11-14 0 7 7 0 0114 0z"
  />
</svg>
</div>
</div>

/* Desktop Table View */
<div className="hidden lg:block p-5 w-full text-sm rounded-md overflow-x-auto bg-white dark:bg-gray-700">
  <table className="min-w-[1000px] w-full">
    <thead className="bg-gray-300 border-gray-300 dark:bg-gray-700 dark:border-gray-500">
      <tr>
        {headers.map((header, index) => (
          <th
            key={index}
            className="text-start py-4 px-6 font-semibold text-gray-600 dark:text-gray-200"
          >
            {header}
          </th>
        ))}
      </tr>
    </thead>
    <tbody>
      {sortedCategories.map((category, index) => (
        <tr
          key={category._id}
          className={`${`border-b border-gray-200 dark:border-gray-500 ${{
            index % 2 === 0 ? 'bg-gray-50 dark:bg-gray-600' : 'bg-gray-100 dark:bg-gray-700'
          }} hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors`}
        >
          {headers.map((header, idx) => (
            <td key={idx} className="py-4 px-6 text-gray-800 dark:text-white">
              {getFieldValue(category, header)}
            </td>
          )))
        </tr>
      ))}
    </tbody>
  </table>
</div>

```

```

        </tbody>
    </table>
</div>

/* Mobile Grid View */
<div className="lg:hidden p-5 space-y-4 overflow-y-scroll h-full">
  {sortedCategories.map((category) => (
    <div
      key={category._id}
      className="border rounded-md p-4 shadow-md hover:shadow-lg transition-shadow border-gray-200 dark:border-gray-500 bg-white dark:bg-gray-600"
    >
      <div className="space-y-2">
        {headers.map((header, idx) => (
          <div key={idx} className="flex justify-between items-center py-1">
            <span className="font-medium text-gray-800 dark:text-white">{header}</span>
            <span className="text-sm text-gray-600 dark:text-gray-300">
              {getFieldValue(category, header)}
            </span>
          </div>
        )))
      </div>
    </div>
  )));
</div>

/* No Results Message */
{sortedCategories.length === 0 && (
  <div className="p-5 text-center text-gray-600 dark:text-gray-300">
    No categories found.
  </div>
)
</>
);
};

export default ManageCategoryTable;

```

src/components/restaurants/MenuItemTable.tsx

```

import { useState } from 'react';

interface MenuItem {
  id: string;
  name: string;
  description: string;
  price: number;
  category: string; // Category _id
  imageUrl?: string;
  thumbnailImage?: string;
  isAvailable: boolean;
}

```

```

}

interface Category {
  _id: string;
  name: string;
}

interface MenuItemsTableProps {
  data: MenuItem[];
  categories?: Category[];
}

const MenuItemsTable: React.FC<MenuItemsTableProps> = ({ data, categories = [] }) => {
  const [menuItems] = useState<MenuItem[]>(data);
  const [categoryFilter, setCategoryFilter] = useState<string>('all');

  // Table headers
  const headers: string[] = ['Image', 'Thumbnail Image', 'Name', 'Description', 'Price', 'Category', 'Availability'];

  // Dynamic categories from data
  const dynamicCategories = ['all', ...Array.from(new Set(menuItems.map(item => item.category))).sort()];

  // Map category _id to name
  const categoryMap = new Map<string, string>(categories.map(cat => [cat._id, cat.name]));

  // Map headers to data fields
  const getFieldValue = (item: MenuItem, header: string) => {
    switch (header.toLowerCase()) {
      case 'image':
        return (
          <img
            src={item.imageUrl || 'https://via.placeholder.com/50x50'}
            alt={item.name}
            className="w-12 h-12 object-cover rounded"
          />
        );
      case 'thumbnail image':
        return (
          <img
            src={item.thumbnailImage || 'https://via.placeholder.com/50x50'}
            alt={`${item.name} thumbnail`}
            className="w-12 h-12 object-cover rounded"
          />
        );
      case 'name':
        return item.name;
      case 'description':
        return <span className="line-clamp-2">{item.description}</span>;
      case 'price':
        return `$$ {item.price.toFixed(2)} `;
      case 'category':
        return categoryMap.get(item.category) || `Category ${item.category}`; // Fallback to _id
      case 'availability':
        return item.availability;
    }
  };
}

```

```

return (
  <span className={getStatusStyles(item.isAvailable)}>
    {item.isAvailable ? 'Available' : 'Not Available'}
  </span>
);
default:
  return "";
}
};

// Filter items by category
const filteredItems = menuItems.filter((item) => {
  return categoryFilter === 'all' || item.category.toLowerCase() === categoryFilter.toLowerCase();
});

// Sort items by name
const sortedItems = [...filteredItems].sort((a, b) => {
  const nameA = a.name.toLowerCase();
  const nameB = b.name.toLowerCase();
  return nameA.localeCompare(nameB);
});

// Get status color styles
const getStatusStyles = (status: boolean) => {
  const baseStyles = 'px-2 py-1 text-xs font-semibold rounded-lg';
  return status
    ? `${baseStyles} bg-green-100 text-green-600 dark:bg-green-700 dark:text-green-200`
    : `${baseStyles} bg-red-100 text-red-600 dark:bg-red-700 dark:text-red-200`;
};

return (
  <>
  {/* Category Filter Controls */}
  <div className="p-5 flex flex-col sm:flex-row gap-4 justify-between items-center">
    <select
      value={categoryFilter}
      onChange={(e) => setCategoryFilter(e.target.value)}
      className="w-full sm:w-40 px-3 py-2 rounded-full bg-gray-100 border border-gray-300 dark:border-gray-500 dark:bg-gray-800 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-orange-500"
    >
      {dynamicCategories.map((category) => (
        <option key={category} value={category.toLowerCase()}>
          {category === 'all' ? 'All' : categoryMap.get(category) || `Category ${category}`}
        </option>
      ))}
    </select>
  </div>

  {/* Desktop Table View with Horizontal Scroll */}
  <div className="hidden lg:block p-5 w-full text-sm rounded-md overflow-x-auto bg-white dark:bg-gray-700">
    <table className="min-w-[1000px] w-full">

```

```

<thead className="bg-gray-300 border-gray-200 dark:bg-gray-800 dark:border-gray-500">
  <tr>
    {headers.map((header, index) => (
      <th
        key={index}
        className="text-start py-4 px-6 font-semibold text-gray-700 dark:text-gray-200"
      >
        {header}
      </th>
    )))
  </tr>
</thead>
<tbody>
  {sortedItems.map((item, index) => (
    <tr
      key={item.id}
      className={`border-b border-gray-200 dark:border-gray-500 ${(
        index % 2 === 0 ? 'bg-gray-50 dark:bg-gray-600' : 'bg-gray-100 dark:bg-gray-700'
      )} hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors`}
    >
      {headers.map((header, idx) => (
        <td key={idx} className="py-4 px-6 text-gray-800 dark:text-white">
          {getFieldValue(item, header)}
        </td>
      )))
    </tr>
  )))
</tbody>
</table>
</div>

/* Mobile Grid View */
<div className="lg:hidden p-5 space-y-4 overflow-y-scroll h-full">
  {sortedItems.map((item) => (
    <div
      key={item.id}
      className="border rounded-md p-4 shadow-md hover:shadow-lg transition-shadow border-gray-200 dark:border-gray-500 bg-white dark:bg-gray-600"
    >
      <div className="space-y-2">
        {headers.map((header, idx) => (
          <div key={idx} className="flex justify-between items-center py-1">
            <span className="font-medium text-gray-800 dark:text-white">{header}</span>
            <span className="text-sm text-gray-600 dark:text-gray-300">
              {getFieldValue(item, header)}
            </span>
          </div>
        )))
      </div>
    </div>
  )))
</div>

```

```

    /* No Results Message */
    {sortedItems.length === 0 && (
      <div className="p-5 text-center text-gray-600 dark:text-gray-300">
        No menu items found.
      </div>
    )}
  </>
);
};

export default MenuItemsTable;

```

src/components/restaurants/ResturentItemCard.tsx

```

import { useState } from 'react';
import { deleteMenuItem } from '../../utils/api';
import ConfirmationModal from '../../components/UI/ConfirmationModal';

interface MenuItem {
  id: string;
  name: string;
  description: string;
  price: number;
  category: string;
  imageUrl?: string;
  isAvailable: boolean;
}

interface ResturentItemCardProps {
  data: MenuItem;
  restaurantId: string;
  categoryName: string;
  onEdit: (item: MenuItem) => void;
  onDelete: (id: string) => void;
}

const ResturentItemCard = ({ data, restaurantId, categoryName, onEdit, onDelete }: ResturentItemCardProps)
=> {
  const [isDeleteModalOpen, setIsDeleteModalOpen] = useState(false);

  const handleOpenDeleteModal = () => {
    setIsDeleteModalOpen(true);
  };

  const handleCloseDeleteModal = () => {
    setIsDeleteModalOpen(false);
  };

  const handleConfirmDelete = async () => {
    try {
      await deleteMenuItem(restaurantId, data.id);
      onDelete(data.id);
    }
  };
}

```

```

setIsDeleteModalOpen(false);
} catch (error) {
  console.error('Delete menu item error:', error);
  alert('Failed to delete menu item');
}
};

return (
  <>
  <div className="bg-white shadow-md dark:bg-gray-800 rounded-md overflow-hidden transition-transform hover:scale-105">
    <img
      src={data.imageUrl || 'https://via.placeholder.com/300x200'}
      alt={data.name}
      className="w-full h-48 object-cover rounded-t-xl p-2"
    />
    <div className="p-4">
      <h2 className="font-bold text-lg text-gray-800 dark:text-white">{data.name}</h2>
      <div className="mt-1">
        <span className="text-sm font-semibold text-gray-800 dark:text-gray-300">Category: </span>
        <span className="inline-block bg-blue-100 text-gray-800 dark:bg-gray-500 dark:text-blue-100 px-2 py-0.5 rounded-full text-xs font-medium">
          {categoryName}
        </span>
      </div>
      <p className="text-sm text-gray-600 dark:text-gray-300 mt-2 line-clamp-2">
        {data.description}
      </p>
      <div className="mt-3 flex justify-between items-center">
        <span className="text-lg font-bold text-green-600 dark:text-green-400">
          Rs. {data.price.toFixed(2)}
        </span>
        <span
          className={`text-xs font-medium px-2 py-1 rounded-md ${data.isAvailable
            ? 'bg-green-100 text-green-600 dark:bg-green-400 dark:text-green-100'
            : 'bg-red-100 text-red-600 dark:bg-red-500 dark:text-red-100'
          }`}>
          {data.isAvailable ? 'Available' : 'Not Available'}
        </span>
      </div>
      <div className="mt-4 flex gap-2">
        <button
          className="flex-1 py-2 rounded-md bg-orange-500 text-white hover:bg-orange-600 dark:bg-orange-500 dark:hover:bg-orange-600 transition-colors"
          onClick={() => onEdit(data)}
        >
          Edit
        </button>
        <button
          className="flex-1 py-2 rounded-md bg-red-500 text-white hover:bg-red-600 dark:bg-red-500 dark:hover:bg-red-600 transition-colors"
        >

```

```

    onClick={handleOpenDeleteModal}
  >
  Delete
  </button>
</div>
</div>
</div>
<ConfirmationModal
  isOpen={isDeleteModalOpen}
  onClose={handleCloseDeleteModal}
  onConfirm={handleConfirmDelete}
  title="Confirm Deletion"
  message={`Are you sure you want to delete "${data.name}"? This action cannot be undone.`}
  confirmText="Delete"
  cancelText="Cancel"
/>
</>
);
};

export default ResturentItemCard;

```

src/components/restaurants/ResturentOrderTable.tsx

```

import { Order } from '@/pages/restaurants/Order';
import React, { useState } from 'react';
import { updateOrderStatus } from '../../utils/api';
import ConfirmationModal from '../../components/UI/ConfirmationModal';

interface ResturentTableProps {
  headers: string[];
  data: Order[];
  defaultStatus?: string; // Optional status filter
}

const ResturentOrderTable: React.FC<ResturentTableProps> = ({ headers, data, defaultStatus }) => {
  const [orders, setOrders] = useState<Order[]>(data);
  const [searchTerm, setSearchTerm] = useState<string>('');
  const [sortOrder, setSortOrder] = useState<'asc' | 'desc'>('desc');
  const [error, setError] = useState<string | null>(null);
  const [isModalOpen, setIsModalOpen] = useState<boolean>(false);
  const [pendingStatusChange, setPendingStatusChange] = useState<{
    orderId: string;
    newStatus: string;
  } | null>(null);

  // Status options
  const statusOptions = [
    'PENDING',
    'CONFIRMED',
    'PREPARING',
    'READY',
  ];

```

```

'OUT_FOR_DELIVERY',
'DELIVERED',
'CANCELLED'
};

// Handle status change initiation
const initiateStatusChange = (orderId: string, newStatus: string) => {
  setPendingStatusChange({ orderId, newStatus });
  setIsModalOpen(true);
};

// Confirm status change
const confirmStatusChange = async () => {
  if (!pendingStatusChange) return;
  const { orderId, newStatus } = pendingStatusChange;
  try {
    await updateOrderStatus(orderId, newStatus as any);
    setOrders(prevOrders =>
      prevOrders.map(order =>
        order.orderId === orderId ? { ...order, status: newStatus } : order
      )
    );
    setError(null);
  } catch (err: any) {
    setError(`Failed to update status: ${err.message}`);
    console.error("Update status error:", err);
  } finally {
    setIsModalOpen(false);
    setPendingStatusChange(null);
  }
};

// Cancel status change
const cancelStatusChange = () => {
  setIsModalOpen(false);
  setPendingStatusChange(null);
};

// Filter orders by search term and default status
const filteredOrders = orders.filter(order =>
  (defaultStatus ? order.status === defaultStatus : true) &&
  (order.orderId.toLowerCase().includes(searchTerm.toLowerCase()) ||
  order.items.toLowerCase().includes(searchTerm.toLowerCase()) ||
  order.deliveryAddress.toLowerCase().includes(searchTerm.toLowerCase()))
);

// Sort orders by date
const sortedOrders = [...filteredOrders].sort((a, b) => {
  const dateA = new Date(a.orderTime).getTime();
  const dateB = new Date(b.orderTime).getTime();
  return sortOrder === 'asc' ? dateA - dateB : dateB - dateA;
});

```

```

// Toggle sort order
const toggleSortOrder = () => {
  setSortOrder(prev => prev === 'asc' ? 'desc' : 'asc');
};

// Status styles
const getStatusStyles = (status: string) => {
  const baseStyles = 'w-fit px-2 py-1 text-xs font-semibold border rounded-lg focus:outline-none focus:ring-2 focus:ring-indigo-500';
  switch (status) {
    case 'PENDING':
      return `${baseStyles} bg-gray-100 text-blue-600 border-blue-600 dark:bg-gray-500 dark:text-blue-200 dark:border-blue-400`;
    case 'CONFIRMED':
      return `${baseStyles} bg-gray-100 text-yellow-600 border-yellow-600 dark:bg-gray-500 dark:text-yellow-200 dark:border-yellow-400`;
    case 'PREPARING':
      return `${baseStyles} bg-gray-100 text-orange-600 border-orange-600 dark:bg-gray-500 dark:text-orange-200 dark:border-orange-400`;
    case 'READY':
      return `${baseStyles} bg-gray-100 text-green-600 border-green-600 dark:bg-gray-500 dark:text-green-200 dark:border-green-400`;
    case 'OUT_FOR_DELIVERY':
      return `${baseStyles} bg-gray-100 text-purple-600 border-purple-600 dark:bg-gray-500 dark:text-purple-200 dark:border-purple-400`;
    case 'DELIVERED':
      return `${baseStyles} bg-gray-100 text-teal-600 border-teal-600 dark:bg-gray-500 dark:text-teal-200 dark:border-teal-400`;
    case 'CANCELLED':
      return `${baseStyles} bg-gray-100 text-red-600 border-red-600 dark:bg-gray-500 dark:text-red-200 dark:border-red-400`;
    default:
      return `${baseStyles} bg-gray-100 text-gray-600 border-gray-600 dark:bg-gray-500 dark:text-gray-200 dark:border-gray-400`;
  }
};

return (
  <>
  {error && <p className="text-red-600 dark:text-red-300 mb-4">{error}</p>}
  <div className="p-5 flex flex-col sm:flex-row gap-4 justify-between items-center">
    <div className="relative w-full sm:w-64">
      <input
        type="text"
        placeholder="Search orders..."
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        className="w-full pl-10 pr-4 py-2 rounded-full border border-gray-300 dark:border-gray-500 bg-white dark:bg-gray-700 text-gray-800 dark:text-gray-200 focus:outline-none focus:ring-2 focus:ring-indigo-500"
      />
      <svg
        className="absolute left-3 top-1/2 transform -translate-y-1/2 h-5 w-5 text-gray-500 dark:text-gray-400"
        fill="none"

```

```

stroke="currentColor"
viewBox="0 0 24 24"
xmlns="http://www.w3.org/2000/svg"
>
<path
  strokeLinecap="round"
  strokeLinejoin="round"
  strokeWidth="2"
  d="M21 21l-6-6m2-5a7 7 0 11-14 0 7 7 0 0114 0z"
/>
</svg>
</div>
<button
  onClick={toggleSortOrder}
  className="w-full sm:w-auto px-4 rounded-full py-2 bg-gray-200 border-gray-200 dark:bg-gray-800
dark:border-gray-500 transition-colors"
>
  Sort by Date {sortOrder === 'asc' ? '↑' : '↓'}
</button>
</div>
<div className="hidden lg:block p-5 w-md text-sm rounded-2xl overflow-hidden bg-white dark:bg-gray-
600">
<table className="w-full">
  <thead className="bg-gray-300 border-gray-200 dark:bg-gray-700 dark:border-gray-500">
    <tr>
      {headers.filter(header => header !== 'Action').map((header, index) => (
        <th key={index} className="text-start py-4 px-6 font-semibold text-gray-600 dark:text-gray-200">
          {header}
        </th>
      ))}
    </tr>
  </thead>
  <tbody>
    {sortedOrders.map((order, index) => (
      <tr
        key={order.orderId}
        className={`border-b border-gray-200 dark:border-gray-500 ${index % 2 === 0
          ? 'bg-gray-50 dark:bg-gray-600'
          : 'bg-gray-100 dark:bg-gray-700'
        } hover:bg-gray-200 dark:hover:bg-gray-500 transition-colors`}
      >
        <td className="py-4 px-6">
          <span className="font-mono text-gray-800 dark:text-white">ORD#{order.orderId}</span>
        </td>
        <td className="py-4 px-6 text-orange-600 font-bold dark:text-gray-300">{order.items}</td>
        <td className="py-4 px-6">
          <span className="font-semibold text-green-600 dark:text-green-300">{order.totalAmount}</span>
        </td>
        <td className="py-4 px-6 text-gray-500 font-bold dark:text-gray-400">{order.orderTime}</td>
        <td className="py-4 px-6 text-gray-600 dark:text-gray-300">{order.deliveryAddress}</td>
        <td className="py-4 px-6">
          <select
            value={order.status}>

```

```

        onChange={(e) => initiateStatusChange(order.orderId, e.target.value)}
        className={getStatusStyles(order.status)}
    >
    {statusOptions.map((status) => (
        <option
            key={status}
            value={status}
            className="bg-gray-100 text-gray-800 dark:bg-gray-600 dark:text-gray-200"
        >
            {status}
        </option>
    )))
    </select>
    </td>
    </tr>
))
</tbody>
</table>
</div>
<div className="lg:hidden p-5 space-y-4 overflow-y-scroll h-3/4">
    {sortedOrders.map((order) => (
        <div
            key={order.orderId}
            className="border rounded-md p-4 bg-white shadow-md hover:shadow-lg transition-shadow border-gray-200 dark:bg-gray-600 dark:border-gray-500"
        >
            <div className="space-y-2">
                <div className="flex justify-between items-start">
                    <div className="font-mono text-gray-600 font-bold dark:text-indigo-300">ORD#{order.orderId}</div>
                    <select
                        value={order.status}
                        onChange={(e) => initiateStatusChange(order.orderId, e.target.value)}
                        className={getStatusStyles(order.status)}
                    >
                        {statusOptions.map((status) => (
                            <option
                                key={status}
                                value={status}
                                className="bg-gray-100 text-gray-800 dark:bg-gray-600 dark:text-gray-200"
                            >
                                {status}
                            </option>
                        )))
                    </select>
                </div>
                <div className="text-sm text-orange-600 font-bold dark:text-gray-300">Items: {order.items}</div>
                <div className="text-sm text-gray-600 dark:text-gray-300">Address: {order.deliveryAddress}</div>
                <div className="flex justify-between items-center">
                    <div className="text-sm text-green-600 font-semibold dark:text-green-300">
                        {order.totalAmount}
                    </div>
                    <div className="text-xs text-gray-500 dark:text-gray-400">{order.orderTime}</div>
                </div>
            </div>
        </div>
    ))

```

```

        </div>
        </div>
        </div>
    ))
</div>
<ConfirmationModal
  isOpen={isModalOpen}
  onClose={cancelStatusChange}
  onConfirm={confirmStatusChange}
  title="Confirm Status Change"
  message={
    pendingStatusChange
    ? `Are you sure you want to change the status of order ${pendingStatusChange.orderId} to
    ${pendingStatusChange.newStatus}?
    :
  }
  confirmText="Confirm"
  cancelText="Cancel"
/>
</>
);
};

export default ResturentOrderTable;

```

src/components/restaurants/Sidebar.tsx

```

import { Link, useNavigate } from "react-router-dom";
import {
  FiCalendar,
  FiBook,
  FiSettings,
  FiPieChart,
  FiUser,
  FiDivide,
  FiUsers,
  FiHome,
  FiSunset,
} from "react-icons/fi";
import { MdPayments, MdOutlinePowerSettingsNew, MdOutlineRestaurantMenu } from "react-icons/md";
import { useState } from "react";
import { useAuth } from "@/context/AuthContext";
import toast, { Toaster } from "react-hot-toast";

interface SubMenuItem {
  path: string;
  title: string;
}

interface MenuItem {
  path?: string; // Made optional since Logout doesn't need a path
  title: string;
}

```

```

icon: JSX.Element;
subItems?: SubMenuItem[];
onClick?: () => void; // Added for actions like logout
}

const menuItems: MenuItem[] = [
  { path: "/resturent-dashboard/overview", title: "Overview", icon: <FiPieChart /> },
  {
    path: "/resturent-dashboard/orders",
    title: "Orders",
    icon: <FiSunset />,
    subItems: [
      { path: "/resturent-dashboard/orders/new", title: "New Orders" },
      { path: "/resturent-dashboard/orders/confirme", title: "Confirme Order" },
      { path: "/resturent-dashboard/orders/preparing", title: "Preparing Orders" },
      { path: "/resturent-dashboard/orders/ready", title: "Ready Orders" },
      { path: "/resturent-dashboard/orders/out-for-delivery", title: "Out For Delivery" },
      { path: "/resturent-dashboard/orders/completed", title: "Complete Orders" },
      { path: "/resturent-dashboard/orders/canceled", title: "Canceled Orders" },
    ],
  },
  {
    path: "/resturent-dashboard/menu-management",
    title: "Menu Management",
    icon: <FiBook />,
    subItems: [
      { path: "/resturent-dashboard/menu-management/all", title: "All Menu Item" },
      { path: "/resturent-dashboard/menu-management/manage", title: "Manage Menu Item" },
      { path: "/resturent-dashboard/menu-management/add", title: "Add New Menu Item" },
    ],
  },
  {
    path: "/resturent-dashboard/category",
    title: "Menu Category Management",
    icon: <MdOutlineRestaurantMenu />,
    subItems: [
      { path: "/resturent-dashboard/category/manage", title: "Manage Category" },
      { path: "/resturent-dashboard/category/add", title: "Add New Category" },
    ],
  },
  {
    path: "/resturent-dashboard/promotions",
    title: "Promotions & Discounts",
    icon: <FiDivide />,
    subItems: [
      { path: "/resturent-dashboard/promotions/create", title: "Create Promotion" },
      { path: "/resturent-dashboard/promotions/active", title: "Active Promotions" },
    ],
  },
  {
    path: "/resturent-dashboard/customers",
    title: "Customers",
    icon: <FiUsers />,
  }
]

```

```

subItems: [
  { path: "/resturent-dashboard/customers/feedback", title: "Customer Feedback" },
  { path: "/resturent-dashboard/customers/loyalty", title: "Loyalty Programs" },
],
},
{
path: "/resturent-dashboard/earnings",
title: "Earnings & Payments",
icon: <MdPayments />,
subItems: [
  { path: "/resturent-dashboard/earnings/daily", title: "Daily/Weekly Earnings" },
  { path: "/resturent-dashboard/earnings/payouts", title: "Payouts" },
],
},
{
path: "/resturent-dashboard/settings",
title: "Business Settings",
icon: <FiHome />,
subItems: [
  { path: "/resturent-dashboard/settings/profile", title: "Profile & Store Info" },
  { path: "/resturent-dashboard/settings/delivery", title: "Delivery Settings" },
  { path: "/resturent-dashboard/settings/staff", title: "Staff Management" },
],
},
{
path: "/resturent-dashboard/reports",
title: "Reports & Analytics",
icon: <FiCalendar />,
subItems: [
  { path: "/resturent-dashboard/reports/sales", title: "Sales Reports" },
  { path: "/resturent-dashboard/reports/behavior", title: "Customer Behavior" },
],
},
{
path: "/resturent-dashboard/support",
title: "Help & Support",
icon: <FiSettings />,
subItems: [
  { path: "/resturent-dashboard/support/contact", title: "Contact Uber Eats Support" },
  { path: "/resturent-dashboard/support/faq", title: "FAQs & Guides" },
],
},
{
path: "/resturent-dashboard/profile",
title: "Profile", icon: <FiUser />,
{
  title: "Logout", // Removed path since it's an action, not a route
  icon: <MdOutlinePowerSettingsNew />,
  onClick: () => {}, // Placeholder, will be set in the component
},
];
}

interface SidebarProps {
  isSidebarOpen: boolean;
}

```

```

const Sidebar = ({ isSidebarOpen }: SidebarProps) => {
  const [openMenus, setOpenMenus] = useState<string[]>([]);
  const { isAuthenticated, logout } = useAuth();
  const navigate = useNavigate();

  const handleLogout = async () => {
    if (!isAuthenticated) {
      toast.error("You are not logged in.");
      return;
    }

    try {
      await logout(); // Call the logout function from AuthContext
      toast.success("Logged out successfully!");
      navigate("/"); // Redirect to home page using useNavigate
    } catch (error) {
      console.error("Logout error:", error);
      toast.error("Failed to log out. Please try again.");
    }
  };
};

const toggleMenu = (path: string) => {
  setOpenMenus((prev) =>
    prev.includes(path) ? prev.filter((item) => item !== path) : [...prev, path]
  );
};

// Update the Logout menu item with the handleLogout function
const updatedMenuItems = menuItems.map((item) =>
  item.title === "Logout" ? { ...item, onClick: handleLogout } : item
);

return (
  <>
  <Toaster position="top-right" reverseOrder={false} />
  <aside
    className={`fixed top-0 left-0 z-40 w-64 h-screen pt-20 bg-white border-r border-gray-200 sm:translate-x-0 dark:bg-gray-800 dark:border-gray-700 transition-transform ${isSidebarOpen ? "translate-x-0" : "-translate-x-full"}`}
  >
    <div className="h-full px-3 pb-4 overflow-y-auto">
      <ul className="space-y-2 font-medium">
        {updatedMenuItems.map((item) => (
          <li key={item.title}>
            {item.subItems ? (
              <div
                className="flex items-center p-2 text-gray-900 rounded-lg dark:text-white hover:bg-gray-100 dark:hover:bg-gray-700 group cursor-pointer"
                onClick={() => toggleMenu(item.path!)}
              >
            
```

```

<span className="mr-3 text-gray-500 dark:text-gray-400 group-hover:text-gray-900 dark:group-
hover:text-white">
  {item.icon}
</span>
<span className="flex-1">{item.title}</span>
<svg
  className={`w-4 h-4 transition-transform ${
    openMenus.includes(item.path!) ? "rotate-180" : ""
  }`}
  fill="none"
  stroke="currentColor"
  viewBox="0 0 24 24"
>
  <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M19 9l-7 7-7-7" />
</svg>
</div>
) : (
<div
  onClick={item.onClick} // Handle clicks for items with onClick (e.g., Logout)
  className={`${`flex items-center p-2 text-gray-900 rounded-lg dark:text-white hover:bg-gray-100
dark:hover:bg-gray-700 group ${(
    item.onClick ? "cursor-pointer" : ""
  )}`}
>
  {item.path ? (
    <Link
      to={item.path}
      className="flex items-center w-full"
    >
      <span className="mr-3 text-gray-500 dark:text-gray-400 group-hover:text-gray-900
dark:group-hover:text-white">
        {item.icon}
      </span>
      <span className="flex-1">{item.title}</span>
    </Link>
  ) : (
    <>
      <span className="mr-3 text-gray-500 dark:text-gray-400 group-hover:text-gray-900
dark:group-hover:text-white">
        {item.icon}
      </span>
      <span className="flex-1">{item.title}</span>
    </>
  )}
</div>
)}
{item.subItems && openMenus.includes(item.path!) && (
  <ul className="pl-6 mt-1 space-y-1">
    {item.subItems.map((subItem) => (
      <li key={subItem.path}>
        <Link
          to={subItem.path}

```

```

        className="flex items-center p-2 text-sm text-gray-700 rounded-lg dark:text-gray-300
        hover:bg-gray-100 dark:hover:bg-gray-700"
      >
        {subItem.title}
      </Link>
    </li>
  ))}
</ul>
)
)
</li>
)}
</ul>
</div>
</aside>
</>
);
};

export default Sidebar;

```

src/components/restaurants/Stats.tsx

```

import ResturentStatsCard from "../UI/ResturentStatsCard";
import ResturentTitle from "../UI/ResturentTitle";
import {
  FiPieChart,
  FiClock,
  FiCheckCircle,
  FiXCircle,
  FiDollarSign,
  FiUsers,
  FiShoppingCart,
  FiStar,
  FiTruck,
  FiPercent,
  FiBox,
  FiUserCheck,
} from "react-icons/fi";

interface Status {
  title: string;
  icon: JSX.Element;
  count: number;
}

const menuItemsData: Status[] = [
  {
    title: "New Orders",
    icon: <FiPieChart />,
    count: 15,
  },
  {

```

```
title: "Pending Orders",
icon: <FiClock />,
count: 8,
},
{
title: "Completed Orders",
icon: <FiCheckCircle />,
count: 45,
},
{
title: "Canceled Orders",
icon: <FiXCircle />,
count: 5,
},
{
title: "Total Revenue",
icon: <FiDollarSign />,
count: 2450,
},
{
title: "Active Customers",
icon: <FiUsers />,
count: 120,
},
{
title: "Items Sold",
icon: <FiShoppingCart />,
count: 180,
},
{
title: "Customer Ratings",
icon: <FiStar />,
count: 4,
},
{
title: "Delivery Orders",
icon: <FiTruck />,
count: 35,
},
{
title: "Discounts Applied",
icon: <FiPercent />,
count: 12,
},
{
title: "Menu Items",
icon: <FiBox />,
count: 50,
},
{
title: "Staff Online",
icon: <FiUserCheck />,
count: 6,
```

```

    },
];

const Stats = () => {
  return (
    <div className="p-4">
      <ResturentTitle text="Restaurant Stats"/>

      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
        {menuItemsData.map((data, index) => (
          <ResturentStatsCard key={index} data={data} />
        )))
      </div>
    </div>
  );
};

export default Stats;

```

src/components/UI/Button.tsx

```

import { Link } from "react-router-dom";
import { motion } from "framer-motion";
import { IconType } from "react-icons";

interface ButtonProps {
  to?: string;
  onClick?: () => void;
  children: React.ReactNode;
  icon?: IconType;
  variant?: "primary" | "secondary";
  className?: string;
}

const Button = ({ to, onClick, children, icon, variant = "primary", className = "" }: ButtonProps) => {
  const baseStyles = "inline-flex items-center gap-2 px-8 py-3 rounded-lg text-lg font-semibold transition-all duration-300";
  const variants = {
    primary: "bg-orange-600 text-white hover:bg-orange-700 hover:scale-105",
    secondary: "bg-white text-orange-600 hover:bg-gray-100 hover:scale-105"
  };

  const buttonContent = (
    <>
      {Icon && <Icon className="text-xl" />}
      {children}
    </>
  );

  if (to) {

```

```

return (
  <motion.div
    whileHover={{ scale: 1.05 }}
    whileTap={{ scale: 0.95 }}
  >
  <Link
    to={to}
    className={`${baseStyles} ${variants[variant]} ${className}`}
  >
    {buttonContent}
  </Link>
</motion.div>
);
}

return (
  <motion.button
    onClick={onClick}
    className={`${baseStyles} ${variants[variant]} ${className}`}
    whileHover={{ scale: 1.05 }}
    whileTap={{ scale: 0.95 }}
  >
    {buttonContent}
  </motion.button>
);
};

export default Button;

```

src/components/UI/card.tsx

```

// Card.tsx
import React from 'react';
import { cn } from '@/lib/utils'; // Utility function for conditional classNames

type CardProps = {
  children: React.ReactNode;
  className?: string;
};

export const Card = ({ children, className }: CardProps) => (
  <div className={cn("border rounded-lg shadow-sm bg-white p-4", className)}>
    {children}
  </div>
);

export const CardContent = ({ children, className }: CardProps) => (
  <div className={cn("p-4", className)}>
    {children}
  </div>
);

```

src/components/UI/CategorySelect.tsx

```
import { useState, useEffect } from 'react';
import { getCategories } from '../../utils/api';

interface Category {
  _id: string;
  name: string;
  description: string;
  restaurantId: string;
  createdAt: string;
  updatedAt: string;
}

interface CategorySelectProps {
  restaurantId: string;
  value: string;
  onChange: (e: React.ChangeEvent<HTMLSelectElement>) => void;
  required?: boolean;
}

const CategorySelect: React.FC<CategorySelectProps> = ({ restaurantId, value, onChange, required }) => {
  const [categories, setCategories] = useState<Category[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    const fetchCategories = async () => {
      try {
        setLoading(true);
        const response = await getCategories(restaurantId);
        setCategories(response);
        setError(null);
      } catch (err: any) {
        setError(err.message || "Failed to load categories.");
        console.error("Fetch categories error:", err);
      } finally {
        setLoading(false);
      }
    };
    if (restaurantId) {
      fetchCategories();
    }
  }, [restaurantId]);

  return (
    <select
      name="category"
      value={value}
      onChange={onChange}
      required={required}
  
```

```

disabled={loading || !!error}
className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white
dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-indigo-500"
>
<option value="">{loading ? 'Loading...' : error ? 'Error' : 'Select a category'}</option>
{ !loading && !error && categories.map(category => (
  <option key={category._id} value={category._id}>
    {category.name}
  </option>
))
</select>
);
};

```

```
export default CategorySelect;
```

src/components/UI/ConfirmationModal.tsx

```
import React from 'react';
```

```
interface ConfirmationModalProps {
  isOpen: boolean;
  onClose: () => void;
  onConfirm: () => void;
  title: string;
  message: string;
  confirmText?: string;
  cancelText?: string;
}
```

```
const ConfirmationModal: React.FC<ConfirmationModalProps> = ({
  isOpen,
  onClose,
  onConfirm,
  title,
  message,
  confirmText = 'Confirm',
  cancelText = 'Cancel',
}) => {
  if (!isOpen) return null;

  return (
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black bg-opacity-50">
      <div className="bg-white dark:bg-gray-700 rounded-lg shadow-lg p-6 w-full max-w-md">
        <h2 className="text-lg font-semibold text-gray-800 dark:text-gray-200 mb-4">{title}</h2>
        <p className="text-gray-600 dark:text-gray-300 mb-6">{message}</p>
        <div className="flex justify-end space-x-4">
          <button
            onClick={onClose}
            className="px-4 py-2 bg-gray-300 dark:bg-gray-600 text-gray-800 dark:text-gray-200 rounded-lg
            hover:bg-gray-400 dark:hover:bg-gray-500 transition-colors"
          >
            {cancelText}
          </button>
        </div>
      </div>
    </div>
  );
}
```

```

        </button>
        <button
          onClick={onConfirm}
          className="px-4 py-2 bg-red-500 text-white rounded-lg hover:bg-red-600 transition-colors"
        >
          {confirmText}
        </button>
      </div>
    </div>
  </div>
);
};

export default ConfirmationModal;

```

src/components/DriverRoute.tsx

```

import React from "react";
import { Navigate } from "react-router-dom";
import { useAuth } from "../context/AuthContext";

interface DriverRouteProps {
  children: React.ReactNode;
}

const DriverRoute: React.FC<DriverRouteProps> = ({ children }) => {
  console.log("DriverRoute component rendered");
  const { user } = useAuth();
  console.log("Current user in DriverRoute:", user);

  if (!user) {
    console.log("No user found, redirecting to signin");
    return <Navigate to="/signin" replace />;
  }

  if (user.role !== "DELIVERY") {
    console.log("User role is not DELIVERY, redirecting to home");
    return <Navigate to="/" replace />;
  }

  console.log("User is authorized, rendering children");
  return <>{children}</>;
};

export default DriverRoute;

```

src/components/Modal.tsx

```

import React from "react";
import { FaExclamationTriangle } from "react-icons/fa";

```

```
interface ModalProps {
  isOpen: boolean;
  onClose: () => void;
  onConfirm: () => void;
  title: string;
  message: string;
  confirmText?: string;
  cancelText?: string;
}

const Modal: React.FC<ModalProps> = ({  
  isOpen,  
  onClose,  
  onConfirm,  
  title,  
  message,  
  confirmText = "Continue",  
  cancelText = "Cancel",  
) => {  
  if (!isOpen) return null;  
  
  return (  
    <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">  
      <div className="bg-white rounded-lg p-6 max-w-md w-full mx-4 shadow-xl">  
        <div className="flex items-center justify-center mb-4">  
          <div className="bg-yellow-100 p-3 rounded-full">  
            <FaExclamationTriangle className="text-yellow-500 text-2xl" />  
          </div>  
        </div>  
  
        <h2 className="text-xl font-semibold text-center mb-2">{title}</h2>  
        <p className="text-gray-600 text-center mb-6">{message}</p>  
  
        <div className="flex justify-center space-x-4">  
          <button  
            onClick={onClose}  
            className="px-4 py-2 border border-gray-300 rounded-lg text-gray-700 hover:bg-gray-50 transition-colors">  
            {cancelText}  
          </button>  
          <button  
            onClick={onConfirm}  
            className="px-4 py-2 bg-orange-500 text-white rounded-lg hover:bg-orange-600 transition-colors">  
            {confirmText}  
          </button>  
        </div>  
      </div>  
    </div>  
  );  
};
```

```
export default Modal;
```

src/components/OrderDetails.tsx

```
import React from "react";

interface OrderDetailsModalProps {
  isOpen: boolean;
  onClose: () => void;
  title: string;
  children: React.ReactNode;
}

const OrderDetailsModal: React.FC<OrderDetailsModalProps> = ({  
  isOpen,  
  onClose,  
  title,  
  children,  
) => {  
  if (!isOpen) return null;  
  
  return (  
    <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">  
      <div className="bg-white rounded-lg p-6 max-w-2xl w-full mx-4 shadow-xl">  
        <div className="flex justify-between items-center mb-4">  
          <h2 className="text-xl font-semibold">{title}</h2>  
          <button  
            onClick={onClose}  
            className="text-gray-500 hover:text-gray-700"  
          >  
            ×  
          </button>  
        </div>  
        <div className="max-h-[70vh] overflow-y-auto">{children}</div>  
      </div>  
    </div>  
  );  
};  
  
export default OrderDetailsModal;
```

src/context/AuthContext.ts

```
import {  
  createContext,  
  useContext,  
  useState,  
  useEffect,  
  ReactNode,  
} from "react";
```

```
interface User {
  id: string;
  email: string;
  firstName: string;
  lastName: string;
  role: "CUSTOMER" | "RESTAURANT" | "ADMIN" | "DELIVERY";
}

interface AuthContextType {
  user: User | null;
  isAuthenticated: boolean;
  login: (userData: User) => void;
  logout: () => void;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider = ({ children }: { children: ReactNode }) => {
  const [user, setUser] = useState<User | null>(null);
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  useEffect(() => {
    // Check if user is logged in on initial load
    const token = localStorage.getItem("token");
    const userData = localStorage.getItem("user");
    if (token && userData) {
      setUser(JSON.parse(userData));
      setIsAuthenticated(true);
    }
  }, []);

  const login = (userData: User) => {
    setUser(userData);
    setIsAuthenticated(true);
    localStorage.setItem("user", JSON.stringify(userData));
  };

  const logout = () => {
    setUser(null);
    setIsAuthenticated(false);
    localStorage.removeItem("token");
    localStorage.removeItem("user");
  };

  return (
    <AuthContext.Provider value={{ user, isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => {
  const context = useContext(AuthContext);
```

```
if (context === undefined) {
  throw new Error("useAuth must be used within an AuthProvider");
}
return context;
};
```

src/context/CartContext.ts

```
import React, { createContext, useContext, useState, useEffect } from "react";
import {
  getCart,
  addToCart,
  updateCartItem,
  removeCartItem,
  clearCart,
  createCart,
  getMenuItemById,
} from "../utils/api";
import { toast } from "react-toastify";
import { useAuth } from "./AuthContext";
```

```
interface User {
  id: string;
  email: string;
  role: "CUSTOMER" | "RESTAURANT" | "ADMIN";
}
```

```
interface CartItem {
  _id: string;
  menuItemId: string;
  restaurantId: string;
  name: string;
  price: number;
  quantity: number;
  mainImage?: string;
  thumbnailImage?: string;
  size?: string;
  pieces?: number;
  description?: string;
  category?: string;
}
```

```
interface CartContextType {
  cartId: string | null; // Added cartId
  cartItems: CartItem[];
  cartTotal: number;
  isCartOpen: boolean;
  toggleCart: () => void;
  addItemToCart: (
    menuItemId: string,
    quantity: number,
```

```

restaurantId: string
) => Promise<void>;
updateItemQuantity: (itemId: string, quantity: number) => Promise<void>;
removeItemFromCart: (itemId: string) => Promise<void>;
clearCartItems: () => Promise<void>;
}

const CartContext = createContext<CartContextType | undefined>(undefined);

export const CartProvider: React.FC<{ children: React.ReactNode }> = ({

  children,
}) => {
  const [cartId, setCartId] = useState<string | null>(null); // Added cartId state
  const [cartItems, setCartItems] = useState<CartItem[]>([]);
  const [cartTotal, setCartTotal] = useState(0);
  const [isCartOpen, setIsCartOpen] = useState(false);
  const { user } = useAuth() as { user: User | null };

  useEffect(() => {
    if (user?.id) {
      console.log("User ID found, fetching cart:", user.id);
      fetchCart();
    } else {
      console.log("No user ID found");
      setCartId(null);
      setCartItems([]);
      setCartTotal(0);
    }
  }, [user]);
}

const fetchCart = async () => {
  try {
    if (!user?.id) {
      console.log("No user ID in fetchCart");
      return;
    }

    const token = localStorage.getItem("token");
    if (!token) {
      console.log("No token found in fetchCart");
      return;
    }

    console.log("Fetching cart for user:", user.id);
    const cart = await getCart(user.id);
    console.log("Fetched cart data:", cart);

    if (cart && Array.isArray(cart.items)) {
      console.log("Raw cart items:", cart.items);
      // Validate cart items before setting them
      const validItems = cart.items.filter((item: CartItem) => {
        console.log("Validating item:", item);
        const isValid =

```

```

item &&
item.menuItemId &&
item.restaurantId &&
item.name &&
typeof item.price === "number" &&
typeof item.quantity === "number";
console.log("Item validation result:", isValid);
return isValid;
});

console.log("Valid cart items:", validItems);
setCartId(cart._id); // Save cartId
setCartItems(validItems);
calculateTotal(validItems);
} else {
  console.log("No valid items in cart, creating new cart");
  const newCart = await createCart(user.id);
  if (newCart && Array.isArray(newCart.items)) {
    const validItems = newCart.items.filter((item: CartItem) => {
      console.log("Validating new cart item:", item);
      const isValid =
        item &&
        item.menuItemId &&
        item.restaurantId &&
        item.name &&
        typeof item.price === "number" &&
        typeof item.quantity === "number";
      console.log("New item validation result:", isValid);
      return isValid;
    });
    console.log("New cart created with valid items:", validItems);
    setCartId(newCart._id); // Save cartId
    setCartItems(validItems);
    calculateTotal(validItems);
  } else {
    console.log("No valid items in new cart");
    setCartId(null);
    setCartItems([]);
    setCartTotal(0);
  }
}
} catch (error) {
  console.error("Error in fetchCart:", error);
  setCartId(null);
  setCartItems([]);
  setCartTotal(0);
}

};

const calculateTotal = (items: CartItem[]) => {
  const total = items.reduce((sum, item) => {
    if (typeof item.price !== "number" || typeof item.quantity !== "number") {

```

```
        console.warn("Invalid item data:", item);
        return sum;
    }
    return sum + item.price * item.quantity;
}, 0);

console.log("Calculated total:", total);
setCartTotal(total);
};

const toggleCart = () => {
    setIsCartOpen(!isCartOpen);
};

const addItemToCart = async (
    menuItemId: string,
    quantity: number,
    restaurantId: string
) => {
    try {
        if (!user?.id) {
            console.log("No user found, user:", user);
            toast.error("Please login to add items to cart");
            return;
        }
    }

    const token = localStorage.getItem("token");
    if (!token) {
        console.log("No token found");
        toast.error("Please login to add items to cart");
        return;
    }

    // Get menu item details
    const menuItem = await getMenuItemById(restaurantId, menuItemId);
    if (!menuItem) {
        throw new Error("Menu item not found");
    }

    console.log("Menu item details:", menuItem);

    const cartItem = {
        menuItemId: menuItem._id,
        restaurantId: restaurantId,
        name: menuItem.name,
        price: menuItem.price,
        quantity: quantity,
        mainImage: menuItem.mainImage
            ? menuItem.mainImage
            : "https://via.placeholder.com/500",
        thumbnailImage: menuItem.thumbnailImage
            ? menuItem.thumbnailImage
            : "https://via.placeholder.com/200",
    };
}
```

```
};

console.log("Adding item to cart:", {
  ...cartItem,
  userId: user.id,
  token: token.substring(0, 10) + "...",
});

await addToCart(user.id, cartItem);
await fetchCart();
toast.success("Item added to cart successfully!");
} catch (error) {
  console.error("Add to cart error:", error);
  if (error instanceof Error) {
    toast.error(error.message || "Failed to add item to cart");
  } else {
    toast.error("Failed to add item to cart");
  }
}
};

const updateItemQuantity = async (menuItemId: string, quantity: number) => {
try {
  if (!user?.id) {
    console.log("No user found");
    toast.error("Please login to update cart");
    return;
  }

  console.log("Updating item quantity:", {
    userId: user.id,
    menuItemId: menuItemId,
    quantity: quantity,
  });
}

await updateCartItem(user.id, menuItemId, quantity);
await fetchCart();
toast.success("Cart updated successfully!");
} catch (error) {
  console.error("Update quantity error:", error);
  toast.error(
    error instanceof Error
      ? error.message
      : "Failed to update item quantity"
  );
}
};

const removeItemFromCart = async (menuItemId: string) => {
try {
  if (!user?.id) {
    console.log("No user found");
    toast.error("Please login to remove items from cart");
  }
}
```

```
    return;
}

await removeCartItem(user.id, menuItemId);
await fetchCart();
toast.success("Item removed from cart");
} catch (error) {
  console.error("Remove item error:", error);
  toast.error(
    error instanceof Error
      ? error.message
      : "Failed to remove item from cart"
  );
}
};

const clearCartItems = async () => {
try {
  if (!user?.id) {
    console.log("No user found");
    toast.error("Please login to clear cart");
    return;
  }

  await clearCart(user.id);
  setCartId(null);
  setCartItems([]);
  setCartTotal(0);
  toast.success("Cart cleared");
} catch (error) {
  console.error("Clear cart error:", error);
  toast.error(
    error instanceof Error ? error.message : "Failed to clear cart"
  );
}
};

return (
<CartContext.Provider
  value={{
    cartId, // Expose cartId
    cartItems,
    cartTotal,
    isCartOpen,
    toggleCart,
    addItemToCart,
    updateItemQuantity,
    removeItemFromCart,
    clearCartItems,
  }}
>
  {children}
</CartContext.Provider>
```

```
);

};

export const useCart = () => {
  const context = useContext(CartContext);
  if (context === undefined) {
    throw new Error("useCart must be used within a CartProvider");
  }
  return context;
};
```

src/context/DriverContext.ts

```
import React, { createContext, useContext, useState, useEffect } from "react";
import { toast } from "react-toastify";
import {
  registerDriver,
  updateDriverLocation,
  updateDriverAvailability,
  getCurrentDriver,
  assignDelivery,
  completeDelivery,
} from "../utils/api";
import { useAuth } from "./AuthContext";

interface Driver {
  _id: string;
  userId: string;
  vehicleType: "BIKE" | "CAR" | "VAN";
  vehicleNumber: string;
  location: {
    type: string;
    coordinates: [number, number];
  };
  isAvailable: boolean;
  currentDelivery: string;
  rating: number;
  totalDeliveries: number;
  createdAt: string;
  updatedAt: string;
  __v: number;
}

interface Delivery {
  _id: string;
  orderId: string;
  status: "PENDING" | "ASSIGNED" | "PICKED_UP" | "DELIVERED" | "CANCELLED";
  pickupLocation: [number, number];
  deliveryLocation: [number, number];
  customerName: string;
}
```

```

customerPhone: string;
restaurantName: string;
restaurantAddress: string;
items: Array<{
  name: string;
  quantity: number;
  price: number;
}>;
}

interface DriverContextType {
  driver: Driver | null;
  currentDelivery: Delivery | null;
  isAvailable: boolean;
  location: [number, number] | null;
  registerDriver: (data: {
    vehicleType: "BIKE" | "CAR" | "VAN";
    vehicleNumber: string;
    location: [number, number];
  }) => Promise<void>;
  updateLocation: (location: [number, number]) => Promise<void>;
  updateAvailability: (isAvailable: boolean) => Promise<void>;
  acceptDelivery: (deliveryId: string) => Promise<void>;
  completeDelivery: () => Promise<void>;
}

```

```
const DriverContext = createContext<DriverContextType | undefined>(undefined);
```

```

export const DriverProvider: React.FC<{ children: React.ReactNode }> = ({ 
  children,
}) => {
  const { user } = useAuth();
  const [driver, setDriver] = useState<Driver | null>(null);
  const [currentDelivery, setCurrentDelivery] = useState<Delivery | null>(null);
  const [isAvailable, setIsAvailable] = useState(false);
  const [location, setLocation] = useState<[number, number] | null>(null);

  useEffect(() => {
    if (user?.role === "DELIVERY") {
      fetchDriverDetails();
    } else if (user?.role) {
      console.warn(
        `User role ${user.role} is not authorized to access driver features`
      );
    }
  }, [user]);
}

```

```

const fetchDriverDetails = async () => {
  try {
    if (!user?.id) {
      throw new Error("User not authenticated");
    }
  }
}

```

```
const response = await getCurrentDriver(user.id);
const driverData = response.data;

if (!driverData) {
  throw new Error("Driver data not found");
}

setDriver(driverData);
setIsAvailable(driverData.isAvailable);

if (driverData.location?.coordinates) {
  setLocation(driverData.location.coordinates);
} else {
  console.warn("Driver location not found in response");
  setLocation(null);
}
} catch (error) {
  console.error("Fetch driver details error:", error);
  toast.error(
    error instanceof Error
      ? error.message
      : "Failed to fetch driver details"
  );
}
};

const registerDriverHandler = async (data: {
  vehicleType: "BIKE" | "CAR" | "VAN";
  vehicleNumber: string;
  location: [number, number];
}) => {
  try {
    if (!user?.id) {
      throw new Error("User not authenticated");
    }
  }

  const driverData = await registerDriver({
    userId: user.id,
    ...data,
  });

  setDriver(driverData);
  setIsAvailable(driverData.isAvailable);
  setLocation(driverData.location.coordinates);
  toast.success("Driver registered successfully!");
}
} catch (error) {
  console.error("Register driver error:", error);
  toast.error(
    error instanceof Error ? error.message : "Failed to register driver"
  );
  throw error;
}
};
```

```

const updateLocationHandler = async (newLocation: [number, number]) => {
  try {
    if (!driver) {
      throw new Error("Driver not registered");
    }

    await updateDriverLocation(driver._id, newLocation);
    setLocation(newLocation);
    toast.success("Location updated successfully!");
  } catch (error) {
    console.error("Update location error:", error);
    toast.error(
      error instanceof Error ? error.message : "Failed to update location"
    );
    throw error;
  }
};

const updateAvailabilityHandler = async (newAvailability: boolean) => {
  try {
    if (!driver) {
      throw new Error("Driver not registered");
    }

    if (!driver._id) {
      throw new Error("Driver ID is missing");
    }

    await updateDriverAvailability(driver._id, newAvailability);
    setIsAvailable(newAvailability);
    toast.success(
      `You are now ${
        newAvailability ? "available" : "unavailable"
      } for deliveries`
    );
  } catch (error) {
    console.error("Update availability error:", error);
    toast.error(
      error instanceof Error ? error.message : "Failed to update availability"
    );
    throw error;
  }
};

const acceptDeliveryHandler = async (deliveryId: string) => {
  try {
    if (!driver) {
      throw new Error("Driver not registered");
    }

    const delivery = await assignDelivery(driver._id, deliveryId);
    setCurrentDelivery(delivery);
  } catch (error) {
    console.error("Accept delivery error:", error);
    toast.error(error.message);
  }
};

```

```

setIsAvailable(false);
toast.success("Delivery accepted successfully!");
} catch (error) {
  console.error("Accept delivery error:", error);
  toast.error(
    error instanceof Error ? error.message : "Failed to accept delivery"
  );
  throw error;
}
};

const completeDeliveryHandler = async () => {
try {
  if (!driver || !currentDelivery) {
    throw new Error("No active delivery");
  }
}

await completeDelivery(driver._id);
setCurrentDelivery(null);
setIsAvailable(true);
toast.success("Delivery completed successfully!");
} catch (error) {
  console.error("Complete delivery error:", error);
  toast.error(
    error instanceof Error ? error.message : "Failed to complete delivery"
  );
  throw error;
}
};

return (
<DriverContext.Provider
value={{
  driver,
  currentDelivery,
  isAvailable,
  location,
  registerDriver: registerDriverHandler,
  updateLocation: updateLocationHandler,
  updateAvailability: updateAvailabilityHandler,
  acceptDelivery: acceptDeliveryHandler,
  completeDelivery: completeDeliveryHandler,
}}
>
{children}
</DriverContext.Provider>
);
};

export const useDriver = () => {
  const context = useContext(DriverContext);
  if (context === undefined) {
    throw new Error("useDriver must be used within a DriverProvider");
  }
};

```

```
    }
    return context;
};
```

src/lib/utils.ts

```
import { type ClassValue, clsx } from "clsx"
import { twMerge } from "tailwind-merge"

export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs))
}
```

src/utils/api.ts

```
const BASE_URL = "http://localhost:3010/api";
import { jwtDecode } from "jwt-decode";
```

```
//----- Auth APIs -----
```

```
interface FormData {
  email: string;
  password: string;
  firstName: string;
  lastName: string;
  role: string;
  address: {
    street: string;
    city: string;
    state: string;
    zipCode: string;
    country: string;
  };
}
interface UserStatusData {
  isActive?: boolean;
  isVerified?: boolean;
}

interface UserRoleData {
  role: "CUSTOMER" | "RESTAURANT" | "DELIVERY" | "ADMIN";
}
```

```
export const login = async (email: string, password: string) => {
  try {
    const response = await fetch(`${BASE_URL}/auth/login`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ email, password }),
    });
  }
```

```
if (!response.ok) {
  throw new Error("Login failed");
}

const data = await response.json();
return data;
} catch (error) {
  console.error("Login error:", error);
  throw error;
}
};

export const register = async (userData: FormData) => {
  try {
    const response = await fetch(` ${BASE_URL}/auth/register`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(userData),
    });

    if (!response.ok) {
      throw response;
    }

    return await response.json();
  } catch (error) {
    console.error("Registration error:", error);
    throw error;
  }
};

export const getProfile = async () => {
  try {
    const response = await fetch(` ${BASE_URL}/auth/profile`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${localStorage.getItem("token")}`,
      },
    });

    if (!response.ok) {
      throw response;
    }

    return await response.json();
  } catch (error) {
    console.error("Profile fetch error:", error);
    throw error;
  }
};
```

```
};

export const updateProfile = async (userData: Partial<FormData>) => {
  try {
    const response = await fetch(`${BASE_URL}/auth/profile`, {
      method: "PUT",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${localStorage.getItem("token")}`,
      },
      body: JSON.stringify(userData),
    });

    if (!response.ok) {
      throw response;
    }

    return await response.json();
  } catch (error) {
    console.error("Profile update error:", error);
    throw error;
  }
};

// get all users (for admin)
export const getAllUsers = async () => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(`${BASE_URL}/auth/users`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to fetch users");
    }

    return await response.json();
  } catch (error) {
    console.error("Fetch all users error:", error);
    throw error;
  }
};

// get user by id (for admin)
export const getUserById = async (userId: string) => {
```

```
try {
  const token = localStorage.getItem("token");
  if (!token) {
    throw new Error("No authentication token found");
  }

  const response = await fetch(`#${BASE_URL}/auth/users/${userId}`, {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
  });

  if (!response.ok) {
    const errorData = await response.json();
    throw new Error(errorData.message || "Failed to fetch user");
  }

  return await response.json();
} catch (error) {
  console.error("Fetch user by ID error:", error);
  throw error;
}
};

// update user (for admin)
export const updateUser = async (
  userId: string,
  userData: Partial<FormData & { password?: string; confirmPassword?: string }>
) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(`#${BASE_URL}/auth/users/${userId}`, {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(userData),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to update user");
    }

    return await response.json();
  } catch (error) {
    console.error("Update user error:", error);
  }
};
```

```
    throw error;
  }
};

// delete user (for admin)
export const deleteUser = async (userId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/auth/users/${userId}` , {
      method: "DELETE",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to delete user");
    }

    return await response.json();
  } catch (error) {
    console.error("Delete user error:", error);
    throw error;
  }
};

// get update user status (for admin)
export const updateUserStatus = async (userId: string, statusData: UserStatusData) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/auth/users/${userId}/status` , {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(statusData),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to update user status");
    }

    return await response.json();
  }
};
```

```

} catch (error) {
  console.error("Update user status error:", error);
  throw error;
}
};

// update user role (for admin)
export const updateUserRole = async (userId: string, roleData: UserRoleData) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/auth/users/${userId}/role`, {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(roleData),
    });

    if (!response.ok) {
      const responseData = await response.json();
      throw new Error(responseData.message || "Failed to update user role");
    }

    return await response.json();
  } catch (error) {
    console.error("Update user role error:", error);
    throw error;
  }
};

//----- Restaurant APIs -----


interface RestaurantFormData {
  restaurantName: string;
  contactPerson: string;
  phoneNumber: string;
  businessType: string;
  cuisineType: string;
  operatingHours: string;
  deliveryRadius: string;
  taxId: string;
  streetAddress: string;
  city: string;
  state: string;
  zipCode: string;
  country: string;
  email: string;
  password?: string;
  agreeTerms?: boolean;
}

```

```

businessLicense: File | null;
foodSafetyCert: File | null;
exteriorPhoto: File | null;
logo: File | null;
}

// Register restaurant
export const registerRestaurant = async (restaurantData: RestaurantFormData) => {
  try {
    const formData = new FormData();
    formData.append("restaurantName", restaurantData.restaurantName);
    formData.append("contactPerson", restaurantData.contactPerson);
    formData.append("phoneNumber", restaurantData.phoneNumber);
    formData.append("businessType", restaurantData.businessType);
    formData.append("cuisineType", restaurantData.cuisineType);
    formData.append("operatingHours", restaurantData.operatingHours);
    formData.append("deliveryRadius", restaurantData.deliveryRadius);
    formData.append("taxId", restaurantData.taxId);
    // Send address fields individually instead of nesting
    formData.append("streetAddress", restaurantData.streetAddress);
    formData.append("city", restaurantData.city);
    formData.append("state", restaurantData.state);
    formData.append("zipCode", restaurantData.zipCode);
    formData.append("country", restaurantData.country);
    formData.append("email", restaurantData.email);
    if (restaurantData.password) {
      formData.append("password", restaurantData.password);
    }
    if (typeof restaurantData.agreeTerms !== "undefined") {
      formData.append("agreeTerms", String(restaurantData.agreeTerms));
    }
    if (restaurantData.businessLicense) {
      formData.append("businessLicense", restaurantData.businessLicense);
    }
    if (restaurantData.foodSafetyCert) {
      formData.append("foodSafetyCert", restaurantData.foodSafetyCert);
    }
    if (restaurantData.exteriorPhoto) {
      formData.append("exteriorPhoto", restaurantData.exteriorPhoto);
    }
    if (restaurantData.logo) {
      formData.append("logo", restaurantData.logo);
    }

    const response = await fetch(` ${BASE_URL}/restaurants/register`, {
      method: "POST",
      body: formData,
    });

    if (!response.ok) {
      throw response;
    }
  }
}

```

```

return await response.json();
} catch (error) {
  console.error("Restaurant registration error:", error);
  throw error;
}
};

// Fetch restaurant by userId
export const getRestaurantByUserId = async () => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('No token found. Please log in.');
    }

    const response = await fetch(` ${BASE_URL}/restaurants`, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
    });
  }

  if (!response.ok) {
    const errorData = await response.json();
    throw new Error(errorData.message || 'Failed to fetch restaurant');
  }

  return await response.json();
} catch (error) {
  console.error('Fetch restaurant by userId error:', error);
  throw error;
}
};

// Update restaurant
export const updateRestaurant = async (restaurantId: string, data: { [key: string]: any }, files?: { [key: string]: File | null }) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('No token found. Please log in.');
    }

    const formData = new FormData();

    // Append text fields
    Object.entries(data).forEach(([key, value]) => {
      if (key === 'address') {
        // Handle nested address object
        Object.entries(value as { [key: string]: string }).forEach(([addrKey, addrValue]) => {
          formData.append(`address[${addrKey}]`, addrValue);
        });
      }
    });
  }
}

```

```

    } else {
      formData.append(key, value as string);
    }
  });

// Append files if provided
if (files) {
  Object.entries(files).forEach(([key, file]) => {
    if (file) {
      formData.append(key, file);
    }
  });
}

const response = await fetch(`#${BASE_URL}/restaurants/${restaurantId}`, {
  method: 'PUT',
  headers: {
    Authorization: `Bearer ${token}`,
    // Do NOT set Content-Type; fetch will set it automatically for multipart/form-data
  },
  body: formData,
});

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || 'Failed to update restaurant');
}

return await response.json();
} catch (error) {
  console.error('Update restaurant error:', error);
  throw error;
}
};

// Get all restaurants
export const getAllRestaurants = async (page = 1, limit = 10) => {
  try {
    const response = await fetch(`#${BASE_URL}/restaurants/all?page=${page}&limit=${limit}`, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || 'Failed to fetch restaurants');
    }

    return await response.json();
  } catch (error) {
    console.error('Fetch all restaurants error:', error);
  }
};

```

```
        throw error;
    }
};

// get restaurant details by id
export const getRestaurantById = async (restaurantId: string) => {
    try {
        const response = await fetch(` ${BASE_URL}/restaurants/${restaurantId}` , {
            method: 'GET',
            headers: {
                'Content-Type': 'application/json',
                Authorization: `Bearer ${localStorage.getItem("token")}`,
            },
        });
    }

    if (!response.ok) {
        const errorData = await response.json();
        throw new Error(errorData.message || 'Failed to fetch restaurant details');
    }
    return await response.json();
} catch (error) {
    console.error('Fetch restaurant details error:', error);
    throw error;
}
};

// Delete a restaurant by ID
export const deleteRestaurant = async (restaurantId: string) => {
    try {
        const token = localStorage.getItem('token');
        const response = await fetch(` ${BASE_URL}/restaurants/${restaurantId}` , {
            method: 'DELETE',
            headers: {
                'Content-Type': 'application/json',
                Authorization: `Bearer ${token}`,
            },
        });
    }

    if (!response.ok) {
        const errorData = await response.json();
        throw new Error(errorData.message || 'Failed to delete restaurant');
    }
    return await response.json();
} catch (error) {
    console.error('Delete restaurant error:', error);
    throw error;
}
};

// Update restaurant status
export const updateRestaurantStatus = async (restaurantId: string, status: string) => {
    try {
```

```
const token = localStorage.getItem('token');
if (!token) {
  throw new Error('No authentication token found');
}

const response = await fetch(`#${BASE_URL}/restaurants/${restaurantId}`, {
  method: 'PATCH',
  headers: {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${token}`,
  },
  body: JSON.stringify({ status }),
});

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || 'Failed to update restaurant status');
}

return await response.json();
} catch (error) {
  console.error('Update restaurant status error:', error);
  throw error;
}
};

//Update restaurant availability
export const updateRestaurantAvailability = async (availability: boolean) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('No authentication token found');
    }

    const response = await fetch(`#${BASE_URL}/restaurants/availability`, {
      method: 'PATCH',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify({ availability }),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || 'Failed to update restaurant availability');
    }

    return await response.json();
  } catch (error) {
    console.error('Update restaurant availability error:', error);
    throw error;
  }
}
```

```
}
```

```
//----- Menu items api's -----
```

```
// Get menu items by restaurant ID (public or authenticated)
export const getMenuItemsByRestaurantId = async (restaurantId: string, isAuthenticated: boolean = false) => {
  try {
    const headers: HeadersInit = {
      'Content-Type': 'application/json',
    };

    if (isAuthenticated) {
      const token = localStorage.getItem('token');
      if (!token) {
        throw new Error('No authentication token found');
      }
      headers['Authorization'] = `Bearer ${token}`;
    }

    const response = await fetch(`#${BASE_URL}/restaurants/${restaurantId}/menu-items`, {
      method: 'GET',
      headers,
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || 'Failed to fetch menu items');
    }

    return await response.json();
  } catch (error) {
    console.error('Fetch menu items error:', error);
    throw error;
  }
};

// Fetch a specific menu item by ID
export const getMenuItemById = async (restaurantId: string, menuItemId: string) => {
  try {

    const response = await fetch(
      `#${BASE_URL}/restaurants/${restaurantId}/menu-items/${menuItemId}`,
    {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
      },
    );
  }

  if (!response.ok) {
    const errorData = await response.json();
  }
}
```

```

    throw new Error(errorData.message || 'Failed to fetch menu item');
}

return await response.json();
} catch (error) {
  console.error('Fetch menu item error:', error);
  throw error;
}
};

// Update a menu item
export const updateMenuItem = async (
  restaurantId: string,
  menuItemId: string,
  data: Partial<{
    name: string;
    description: string;
    price: number;
    category: string;
    isAvailable: boolean;
  }>,
  files?: { mainImage?: File; thumbnailImage?: File }
) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('No authentication token found');
    }

    const formData = new FormData();
    // Append text fields
    if (data.name) formData.append('name', data.name);
    if (data.description) formData.append('description', data.description);
    if (data.price !== undefined) formData.append('price', data.price.toString());
    if (data.category) formData.append('category', data.category);
    if (data.isAvailable !== undefined) formData.append('isAvailable', data.isAvailable.toString());

    // Append files if provided
    if (files?.mainImage) formData.append('mainImage', files.mainImage);
    if (files?.thumbnailImage) formData.append('thumbnailImage', files.thumbnailImage);

    const response = await fetch(
      `${BASE_URL}/restaurants/${restaurantId}/menu-items/${menuItemId}`,
      {
        method: 'PATCH',
        headers: {
          Authorization: `Bearer ${token}`,
          // Do NOT set Content-Type; fetch sets it automatically for FormData
        },
        body: formData,
      }
    );
  }
}

```

```
if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || 'Failed to update menu item');
}

return await response.json();
} catch (error) {
  console.error('Update menu item error:', error);
  throw error;
}
};

// Delete a menu item
export const deleteMenuItem = async (restaurantId: string, menuItemId: string) => {
  try {
    const token = localStorage.getItem('token');
    if (!token) {
      throw new Error('No authentication token found');
    }

    const response = await fetch(
      `${BASE_URL}/restaurants/${restaurantId}/menu-items/${menuItemId}`,
      {
        method: 'DELETE',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
      }
    );

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || 'Failed to delete menu item');
    }

    return await response.json();
  } catch (error) {
    console.error('Delete menu item error:', error);
    throw error;
  }
};

// add menu item
export const addMenuItem = async (
  restaurantId: string,
  data: {
    name: string;
    description: string;
    price: number;
    category: string;
    isAvailable: boolean;
  },
)
```

```

files?: {
  mainImage?: File;
  thumbnailImage?: File;
}
) => {
  const token = localStorage.getItem('token');
  const formData = new FormData();

  // Append text fields
  formData.append('name', data.name);
  formData.append('description', data.description);
  formData.append('price', data.price.toString());
  formData.append('category', data.category);
  formData.append('isAvailable', data.isAvailable.toString());

  // Append files if provided
  if (files?.mainImage) {
    formData.append('mainImage', files.mainImage);
  }
  if (files?.thumbnailImage) {
    formData.append('thumbnailImage', files.thumbnailImage);
  }

  const response = await fetch(` ${BASE_URL}/restaurants/${restaurantId}/menu-items`, {
    method: 'POST',
    headers: {
      ...(token && { 'Authorization': `Bearer ${token}` }),
      // Do not set Content-Type for FormData; browser sets it with boundary
    },
    body: formData,
  });

  if (!response.ok) {
    const errorMessage = await response.json();
    throw new Error(errorMessage.message || 'Failed to add menu item');
  }

  return response.json();
};

//----- Category APIs -----


interface CategoryData {
  name: string;
  description?: string;
}

// Get all categories for a restaurant
export const getCategories = async (restaurantId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }
  }
}

```

```

}

const response = await fetch(`${BASE_URL}/restaurants/${restaurantId}/categories`, {
  method: "GET",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`,
  },
});

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || "Failed to fetch categories");
}

return await response.json();
} catch (error) {
  console.error("Fetch categories error:", error);
  throw error;
}
};

// Add a new category
export const addCategory = async (restaurantId: string, data: CategoryData) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(`${BASE_URL}/restaurants/${restaurantId}/categories`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to add category");
    }

    return await response.json();
  } catch (error) {
    console.error("Add category error:", error);
    throw error;
  }
};

// Update an existing category
export const updateCategory = async (

```

```
restaurantId: string,
categoryId: string,
data: Partial<CategoryData>
) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(`#${BASE_URL}/restaurants/${restaurantId}/categories/${categoryId}`, {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to update category");
    }

    return await response.json();
  } catch (error) {
    console.error("Update category error:", error);
    throw error;
  }
};

// Delete a category
export const deleteCategory = async (restaurantId: string, categoryId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(`#${BASE_URL}/restaurants/${restaurantId}/categories/${categoryId}`, {
      method: "DELETE",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to delete category");
    }

    return await response.json();
  }
};
```

```

} catch (error) {
  console.error("Delete category error:", error);
  throw error;
}
};

// Get category by ID (public)
// Get category by ID (public)
export const getCategoryById = async (restaurantId: string, categoryId: string) => {
  try {
    const response = await fetch(` ${BASE_URL}/restaurants/${restaurantId}/categories/${categoryId}` , {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    });
  }

  if (!response.ok) {
    const errorData = await response.json();
    throw new Error(errorData.message || "Failed to fetch category");
  }

  const data = await response.json();
  return data.data; // Extract data field
} catch (error) {
  console.error("Fetch category by ID error:", error);
  throw error;
}
};

//----- Cart APIs -----


interface CartItem {
  _id?: string;
  menuItemId: string;
  restaurantId: string;
  name: string;
  price: number;
  quantity: number;
  mainImage?: string;
  thumbnailImage?: string;
}

// Get cart
export const getCart = async (cartId: string) => {
  try {
    const response = await fetch(` ${BASE_URL}/carts/${cartId}` , {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${localStorage.getItem("token")}`,
      },
    });
  }

```

```

if (!response.ok) {
  throw new Error("Failed to fetch cart");
}

return await response.json();
} catch (error) {
  console.error("Get cart error:", error);
  throw error;
}
};

// Add item to cart
export const addToCart = async (cartId: string, item: CartItem) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const requestBody = {
      menuItemId: item.menuItemId,
      restaurantId: item.restaurantId,
      name: item.name,
      price: item.price,
      quantity: item.quantity,
      mainImage: item.mainImage
        ? item.mainImage
        : "https://via.placeholder.com/500",
      thumbnailImage: item.thumbnailImage
        ? item.thumbnailImage
        : "https://via.placeholder.com/200",
    };
  }

  console.log("Sending cart request:", {
    url: `${BASE_URL}/carts/${cartId}/items`,
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token.substring(0, 10)}...`,
    },
    body: requestBody,
  });

  const response = await fetch(`${BASE_URL}/carts/${cartId}/items`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify(requestBody),
  });
}

```

```

const responseText = await response.text();
console.log("Raw response:", responseText);

let data;
try {
  data = JSON.parse(responseText);
} catch {
  data = { message: responseText };
}

if (!response.ok) {
  console.error("Cart API error:", {
    status: response.status,
    statusText: response.statusText,
    error: data,
    requestBody: requestBody,
  });
  throw new Error(data.message || "Failed to add item to cart");
}

console.log("Cart API success:", data);
return data;
} catch (error) {
  console.error("Add to cart error:", error);
  throw error;
}
};

// Update cart item
export const updateCartItem = async (
  userId: string,
  menuItemId: string,
  quantity: number
) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }
  }

  // First, get the cart to verify the item exists
  const cartResponse = await fetch(`${BASE_URL}/carts/${userId}`, {
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });

  if (!cartResponse.ok) {
    throw new Error("Cart not found");
  }

  const cart = await cartResponse.json();
  const itemExists = cart.items.some(

```

```

(item: CartItem) => item.menuItemId === menuItemId
);

if (!itemExists) {
  throw new Error("Item not found in cart");
}

const response = await fetch(
`"${BASE_URL}/carts/${userId}/items/${menuItemId}`,
{
  method: "PATCH",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`,
  },
  body: JSON.stringify({ quantity }),
}
);

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || "Failed to update cart item");
}

return await response.json();
} catch (error) {
  console.error("Update cart item error:", error);
  throw error;
}
};

// Remove cart item
export const removeCartItem = async (userId: string, menuItemId: string) => {
try {
  const token = localStorage.getItem("token");
  if (!token) {
    throw new Error("No authentication token found");
  }

  const response = await fetch(
`"${BASE_URL}/carts/${userId}/items/${menuItemId}`,
{
  method: "DELETE",
  headers: {
    Authorization: `Bearer ${token}`,
  },
}
);

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || "Failed to remove cart item");
}
}

```

```
return await response.json();
} catch (error) {
  console.error("Remove cart item error:", error);
  throw error;
}
};

// Clear cart
export const clearCart = async (userId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }
  }

  const response = await fetch(`${BASE_URL}/carts/${userId}`, {
    method: "DELETE",
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });

  if (!response.ok) {
    const responseData = await response.json();
    throw new Error(responseData.message || "Failed to clear cart");
  }

  return await response.json();
} catch (error) {
  console.error("Clear cart error:", error);
  throw error;
}
};

// Create a new cart
export const createCart = async (userId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }
  }

  console.log("Creating cart for user:", userId);

  const response = await fetch(`${BASE_URL}/carts`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify({ userId }),
  });
}
```

```

// Log the raw response for debugging
const responseText = await response.text();
console.log("Raw create cart response:", responseText);

let data;
try {
  data = JSON.parse(responseText);
} catch {
  data = { message: responseText };
}

if (!response.ok) {
  console.error("Create cart error:", {
    status: response.status,
    statusText: response.statusText,
    error: data,
  });
  throw new Error(data.message || "Failed to create cart");
}

console.log("Cart created successfully:", data);
return data;
} catch (error) {
  console.error("Create cart error:", error);
  throw error;
}
};

//----- order api's -----
interface OrderItem {
  menuItemId: string;
  name: string;
  price: number;
  quantity: number;
}

interface DeliveryAddress {
  street: string;
  city: string;
  state: string;
  zipCode: string;
  country: string;
}

interface CreateOrderData {
  userId: string;
  restaurantId: string;
  items: OrderItem[];
  deliveryAddress: DeliveryAddress;
  paymentMethod: "CREDIT_CARD" | "CASH" | "ONLINE";
}

interface OrderStatus {

```

```

status:
| "PENDING"
| "CONFIRMED"
| "PREPARING"
| "READY_FOR_PICKUP"
| "ON_THE_WAY"
| "DELIVERED"
| "CANCELLED";
}

export const createOrder = async (orderData: CreateOrderData) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/orders`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(orderData),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to create order");
    }

    return await response.json();
  } catch (error) {
    console.error("Create order error:", error);
    throw error;
  }
};

// Get orders by restaurant ID
// Fetches orders for the restaurant ID derived from the JWT token's userId
export const getOrdersByRestaurantId = async () => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    // Decode JWT token to extract userId (used as restaurantId)
    const decoded: { userId: string } = jwtDecode(token);
    const restaurantId = decoded.userId;
    if (!restaurantId) {
      throw new Error("Restaurant ID not found in token");
    }
  }
};

```

```

console.log("resturent id eka thamai apu meka ",restaurantId)

const response = await fetch(` ${BASE_URL}/orders/restaurant/${restaurantId}` , {
  method: "GET",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`,
  },
});

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || "Failed to fetch restaurant orders");
}

return await response.json();
} catch (error) {
  console.error("Get restaurant orders error:", error);
  throw error;
}
};

// Get orders by user ID
export const getOrdersByUserId = async (userId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/orders/user/${userId}` , {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to fetch orders");
    }

    return await response.json();
  } catch (error) {
    console.error("Get orders error:", error);
    throw error;
  }
};

// Get order by ID

```

```
export const getOrderById = async (orderId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/orders/${orderId}`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to fetch order");
    }

    return await response.json();
  } catch (error) {
    console.error("Get order error:", error);
    throw error;
  }
};

// Update order status
export const updateOrderStatus = async (
  orderId: string,
  status: OrderStatus["status"]
) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/orders/${orderId}/status`, {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify({ status }),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to update order status");
    }

    return await response.json();
  }
};
```

```
    } catch (error) {
      console.error("Update order status error:", error);
      throw error;
    }
  };

// Delete order by ID
export const deleteOrder = async (orderId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(`#${BASE_URL}/orders/${orderId}`, {
      method: "DELETE",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) {
      const responseData = await response.json();
      throw new Error(responseData.message || "Failed to delete order");
    }

    // For 204 No Content, we don't need to parse JSON
    if (response.status === 204) {
      return null;
    }

    // For other success statuses, parse JSON
    return await response.json();
  } catch (error) {
    console.error("Delete order error:", error);
    throw error;
  }
};

// Cancel order by ID
export const cancelOrder = async (orderId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(`#${BASE_URL}/orders/${orderId}/cancel`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });
  }
};
```

```

    },
    body: JSON.stringify({ status: "CANCELLED" }),
  });

if (!response.ok) {
  const errorText = await response.text();
  let errorMessage = "Failed to cancel order";

  try {
    const eventData = JSON.parse(errorText);
    errorMessage = eventData.message || errorMessage;
  } catch (e) {
    // If response is not JSON, use the raw text
    console.log("Error parsing JSON:", e);
    errorMessage = errorText || errorMessage;
  }

  throw new Error(errorMessage);
}

return await response.json();
} catch (error) {
  console.error("Cancel order error:", error);
  throw error;
}
};

//----- Payment APIs -----


interface PaymentData {
  userId: string;
  cartId: string;
  orderId: string;
  restaurantId: string;
  items: OrderItem[];
  totalAmount: number;
  paymentMethod: "CREDIT_CARD" | "DEBIT_CARD";
  cardDetails: {
    cardNumber: string;
    cardHolderName: string;
  };
}

/**
 * Initiates a payment process by calling the payment service's /process endpoint.
 * Returns the PayHere payload and hash for redirecting to the PayHere payment page.
 * @param paymentData - Data required to initiate the payment
 * @returns Promise resolving to the payment initiation response
 * @throws Error if the request fails or token is missing
 */
export const initiatePayment = async (paymentData: PaymentData) => {
  try {
    const token = localStorage.getItem("token");
  }

```

```

if (!token) {
  throw new Error("No authentication token found");
}

const response = await fetch(`${BASE_URL}/payments/process`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`,
  },
  body: JSON.stringify(paymentData),
});

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || "Failed to initiate payment");
}

return await response.json();
} catch (error) {
  console.error("Initiate payment error:", error);
  throw error;
}
};

// get all payments
export const getAllPayments = async (params: {
  status?: string;
  restaurantId?: string;
  startDate?: string;
  endDate?: string;
  page?: number;
  limit?: number;
}) => {
  try {
    const query = new URLSearchParams();
    if (params.status) query.set("status", params.status);
    if (params.restaurantId) query.set("restaurantId", params.restaurantId);
    if (params.startDate) query.set("startDate", params.startDate);
    if (params.endDate) query.set("endDate", params.endDate);
    if (params.page) query.set("page", params.page.toString());
    if (params.limit) query.set("limit", params.limit.toString());

    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(`${BASE_URL}/payments/all?${query.toString()}`, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });
  }
}

```

```

});

if (!response.ok) {
  const eventData = await response.json();
  throw new Error(eventData.message || "Failed to fetch payments");
}

return await response.json();
} catch (error) {
  console.error("Get all payments error:", error);
  throw error;
}
};

// refund payments
export const refundPayment = async ({ paymentId, reason }: { paymentId: string; reason: string }) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/payments/refund`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify({ paymentId, reason }),
    });

    const data = await response.json();
    if (!response.ok) {
      console.error("Refund payment failed:", { status: response.status, data });
      throw new Error(data.message || "Failed to refund payment");
    }

    // Normalize response to match frontend expectation
    return {
      success: data.success,
      data: {
        paymentId: data.paymentId,
        paymentStatus: data.paymentStatus,
        refundReason: reason,
      },
      message: data.message || "Payment refunded successfully",
    };
  } catch (error) {
    console.error("Refund payment error:", error);
    throw error;
  }
};

```

----- Driver APIs -----

```
interface DriverFormData {
  userId: string;
  vehicleType: "BIKE" | "CAR" | "VAN";
  vehicleNumber: string;
  location: [number, number];
}

// Register driver
export const registerDriver = async (driverData: DriverFormData) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/drivers/register`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(driverData),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to register driver");
    }

    return await response.json();
  } catch (error) {
    console.error("Register driver error:", error);
    throw error;
  }
};

// Update driver location
export const updateDriverLocation = async (
  driverId: string,
  location: [number, number]
) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/drivers/${driverId}/location`, {
      method: "PUT",
      headers: {
        "Content-Type": "application/json",
      },
    });
  }
};
```

```
    Authorization: `Bearer ${token}`,
},
body: JSON.stringify({ location }),
});

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || "Failed to update location");
}

return await response.json();
} catch (error) {
  console.error("Update location error:", error);
  throw error;
}
};

// Update driver availability
export const updateDriverAvailability = async (
  driverId: string,
  isAvailable: boolean
) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    if (!driverId) {
      throw new Error("Driver ID is required");
    }

    const response = await fetch(
      `${BASE_URL}/drivers/${driverId}/availability`,
      {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({
          isAvailable,
          driverId, // Add driverId to the request body
        }),
      }
    );

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || "Failed to update availability");
    }

    return await response.json();
  }
}
```

```

} catch (error) {
  console.error("Update availability error:", error);
  throw error;
}
};

// Get available drivers
export const getAvailableDrivers = async (
  latitude: number,
  longitude: number,
  maxDistance: number = 5000
) => {
  try {
    const response = await fetch(
      `${BASE_URL}/drivers/available?latitude=${latitude}&longitude=${longitude}&maxDistance=${maxDistance}`,
      {
        method: "GET",
        headers: {
          "Content-Type": "application/json",
        },
      }
    );
  }

  if (!response.ok) {
    const errorMessage = await response.json();
    throw new Error(errorMessage.message || "Failed to fetch available drivers");
  }

  return await response.json();
} catch (error) {
  console.error("Get available drivers error:", error);
  throw error;
}
};

// Get driver details
export const getDriverDetails = async (driverId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }
    const response = await fetch(`${BASE_URL}/drivers/${driverId}`, {
      method: "GET",
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
  }

  if (!response.ok) {
    const errorMessage = await response.json();
    throw new Error(errorMessage.message || "Failed to fetch driver details");
  }
};

```

```

}

return await response.json();
} catch (error) {
  console.error("Get driver details error:", error);
  throw error;
}
};

// Get current driver details
export const getCurrentDriver = async (userId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    console.log("userId", userId);

    const response = await fetch(` ${BASE_URL}/drivers/me?userId=${userId}`, {
      method: "GET",
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) {
      const responseData = await response.json();
      throw new Error(responseData.message || "Failed to fetch driver details");
    }

    return await response.json();
  } catch (error) {
    console.error("Get current driver error:", error);
    throw error;
  }
};

// Assign delivery to driver
export const assignDelivery = async (driverId: string, deliveryId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }

    const response = await fetch(` ${BASE_URL}/drivers/${driverId}/assign`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify({ deliveryId }),
    });
  }
}

```

```

});

if (!response.ok) {
  const errorData = await response.json();
  throw new Error(errorData.message || "Failed to assign delivery");
}

return await response.json();
} catch (error) {
  console.error("Assign delivery error:", error);
  throw error;
}
};

// Complete delivery
export const completeDelivery = async (driverId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }
  }

  const response = await fetch(`${BASE_URL}/drivers/${driverId}/complete`, {
    method: "POST",
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });

  if (!response.ok) {
    const errorData = await response.json();
    throw new Error(errorData.message || "Failed to complete delivery");
  }

  return await response.json();
} catch (error) {
  console.error("Complete delivery error:", error);
  throw error;
}
};

// Delivery Service API
export const assignDeliveryDriver = async (
  orderId: string,
  customerLocation: [number, number]
): Promise<{ status: string; message?: string }> => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }
  }
}

```

```
// Send the actual customer location coordinates [longitude, latitude]
const requestBody = {
  orderId,
  customerLocation,
};

console.log("Assigning delivery driver with data:", requestBody);

const response = await fetch(` ${BASE_URL}/delivery/assign`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`,
  },
  body: JSON.stringify(requestBody),
});

const data = await response.json();
console.log("Delivery assignment response:", data);

if (response.status === 404) {
  return {
    status: "error",
    message: "No available drivers found in the area",
  };
}

if (!response.ok) {
  throw new Error(data.message || "Failed to assign delivery driver");
}

return {
  status: "success",
  message: data.message,
};

} catch (error) {
  console.error("Delivery assignment error:", error);
  return {
    status: "error",
    message:
      error instanceof Error
        ? error.message
        : "Failed to assign delivery driver",
  };
}

export const updateDeliveryStatus = async (
  deliveryId: string,
  status: string,
  location: [number, number]
) => {
```

```

const response = await fetch(` ${BASE_URL}/delivery/${deliveryId}/status` , {
  method: "PUT",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    status,
    location,
  }),
});
return response.json();
};

export const getDeliveryStatus = async (deliveryId: string) => {
  const response = await fetch(` ${BASE_URL}/delivery/${deliveryId}/status`);
  return response.json();
};

export const getDriverLocation = async (deliveryId: string) => {
  const response = await fetch(` ${BASE_URL}/delivery/${deliveryId}/location`);
  return response.json();
};

export const getDeliveryStatusbyOrderId = async (orderId: string) => {
  try {
    const token = localStorage.getItem("token");
    if (!token) {
      throw new Error("No authentication token found");
    }
  }

  const response = await fetch(` ${BASE_URL}/delivery/order/${orderId}` , {
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });

  if (!response.ok) {
    throw new Error("Failed to fetch delivery status");
  }

  return response.json();
} catch (error) {
  console.error("Get delivery status error:", error);
  throw error;
}
};

```

src/App.tsx

```

import { BrowserRouter, Navigate, Route, Routes } from "react-router-dom";
import { AuthProvider } from "@/context/AuthContext";
import { CartProvider } from "@/context/CartContext";

```

```

import { DriverProvider } from "@/context/DriverContext";
import { ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import Home from "./pages/Home";
import Layout from "./components/UI/Layout";
import SignIn from "./pages/SignIn";
import SignUp from "./pages/SignUp";
import PaymentMethod from "@/components/Payment/PaymentMethod";
import CardDetails from "@/components/Payment/CardDetails";
import OrderConfirmation from "@/pages/OrderConfirmation";
import RestaurantMenu from "./pages/RestaurantMenu";
import ViewMenuItem from "./pages/ViewMenuItem";
import Order from "./pages/Order";
import Cart from "./pages/Cart";
import ResturentRegister from "./pages/ResturentRegister";
import Profile from "./pages/Profile";
import DriverDashboard from "./pages/Drivers/DriverDashboard";
import TermsAndConditions from "./pages/TermsAndConditions";

//----- admin-----
import AdminLayout from "./pages/admin/AdminLayout";
import AllResturent from "./pages/admin/AllResturent";
import RequestResturent from "./pages/admin/RequestResturent";
import AddResturent from "./pages/admin/AddResturent";
import AllUsers from "./pages/admin/AllUsers";
import UserPermissions from "./pages/admin/UserPermissions";
import UserSetting from "./pages/admin/UserSetting";
import AddUser from "./pages/admin/AddUser";
import Earnings from "./pages/admin/Earnings";

const AdminRoutes = () => {
  return (
    <Routes>
      <Route path="/" element={<AdminLayout />}>
        <Route index element={<Navigate to="overview" replace />} />{" "}
        {/* Relative path */}
        <Route path="overview" element={<Overview />} />
        <Route path="profile" element={<Profile />} />
        {/* Order-related routes grouped under /orders */}
        <Route path="resturent">
          <Route path="" element={<AllResturent />} />
          <Route path="request" element={<RequestResturent />} />
          <Route path="add" element={<AddResturent />} />
        </Route>
        {/* user-related routes grouped under /orders */}
        <Route path="user-management">
          <Route path="all" element={<AllUsers />} />
          <Route path="roles" element={<UserPermissions />} />
          <Route path="add" element={<AddUser />} />
          <Route path="settings" element={<UserSetting />} />
        </Route>
        {/* payment-related routes grouped under /orders */}
      </Route>
    </Routes>
  );
}

```

```

<Route path="earnings">
  <Route path="earan" element={<Earnings />} />
  <Route path="roles" element={<UserPermissions />} />
  <Route path="add" element={<AddUser />} />
  <Route path="settings" element={<UserSetting />} />
</Route>

  {/* Add more routes as needed to match sidebar */}
</Route>
</Routes>
);
};

//----- resturent-----
import ResturentLayout from "./pages/restaurants/ResturentLayout";
import Overview from "./pages/restaurants/Overview";

import PendingOrders from "./pages/restaurants/PendingOrders";
import PreparingOrder from "./pages/restaurants/PreparingOrder";
import CancelledOrder from "./pages/restaurants/CanceledOrder";
import ConfirmedOrders from "./pages/restaurants/ConfirmedOrders";
import ReadyOrders from "./pages/restaurants/ReadyOrders";
import CompletedOrders from "./pages/restaurants/CompletedOrders";
import OutForDeliveryOrders from "./pages/restaurants/OutForDeliveryOrders";
import ManageMenuItems from "./pages/restaurants/ManageMenuItems";
import AddNewMenuItem from "./pages/restaurants/AddNewMenuItem";
import AllMenuItems from "./pages/restaurants/AllMenuItems";
import AddCategories from "./pages/restaurants/AddCategories";
import CategoryManagement from "./pages/restaurants/CategoryManagement";
import FoodHomePage from "./pages/FoodHomePage";
import Restaurant from "./pages/Restaurant";
import ResturentProfile from "./pages/restaurants/ResturentProfile";
import MobileBottomNav from "./components/UI/MobileBottomNav";
import Checkout from "./pages/Checkout";
import OrdersList from "./pages/OrdersList";

const ResturentRoutes = () => {
  return (
    <Routes>
      <Route path="/" element={<ResturentLayout />}>
        <Route index element={<Navigate to="overview" replace />} />{" "}
        {/* Relative path */}
        <Route path="overview" element={<Overview />} />
        <Route path="profile" element={<ResturentProfile />} />
        {/* Order-related routes grouped under /orders */}
        <Route path="orders">
          <Route path="new" element={<PendingOrders />} />
          <Route path="confirme" element={<ConfirmedOrders />} />
          <Route path="preparing" element={<PreparingOrder />} />
          <Route path="ready" element={<ReadyOrders />} />
          <Route path="out-for-delivery" element={<OutForDeliveryOrders />} />
          <Route path="completed" element={<CompletedOrders />} />
          <Route path="canceled" element={<CancelledOrder />} />
        </Route>
      </Route>
    </Routes>
  );
};

```

```

</Route>
 {/* menu-related routes grouped under /menu-management */}
<Route path="menu-management">
  <Route path="all" element={<AllMenuItems />} />
  <Route path="manage" element={<ManageMenuItems />} />
  <Route path="add" element={<AddNewMenuItem />} />
</Route>
 {/* category-related routes grouped under /menu-management */}
<Route path="category">
  <Route path="manage" element={<CategoryManagement />} />
  <Route path="add" element={<AddCategories />} />
</Route>
 {/* Add more routes as needed to match sidebar */}
</Route>
</Routes>
);
};

//----- driver-----
const DriverRoutes = () => {
  console.log("DriverRoutes component rendered");
  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        <Route index element={<DriverDashboard />} />
      </Route>
    </Routes>
  );
};

function App() {
  console.log("App component rendered");
  return (
    <BrowserRouter>
      <AuthProvider>
        <CartProvider>
          <DriverProvider>
            <ToastContainer
              position="top-right"
              autoClose={3000}
              hideProgressBar={false}
              newestOnTop
              closeOnClick
              rtl={false}
              pauseOnFocusLoss
              draggable
              pauseOnHover
              theme="light"
            />
            <Routes>
              <Route path="/" element={<Layout />}>
                <Route path="/" element={<Home />} />
                <Route path="/signin" element={<SignIn />} />
              </Routes>
            </DriverProvider>
          </CartProvider>
        </AuthProvider>
      </BrowserRouter>
    );
}

```

```

<Route path="/signup" element={<SignUp />} />
<Route
  path="/resturent-signup"
  element={<ResturentRegister />}
/>
<Route path="/restaurants" element={<Restaurant />} />
<Route path="/payment-method" element={<PaymentMethod />} />
<Route path="/card-details" element={<CardDetails />} />
<Route
  path="/order-confirmation"
  element={<OrderConfirmation />}
/>
<Route path="/orders" element={<OrdersList />} />
<Route path="/order/:orderId" element={<Order />} />
<Route path="/cart" element={<Cart />} />
<Route path="/checkout" element={<Checkout />} />
<Route path="/account" element={<Profile />} />
<Route
  path="/restaurant/:restaurantId/menu"
  element={<RestaurantMenu />}
/>
<Route
  path="/restaurant/:restaurantId/menu/:menuItem"
  element={<ViewMenuItem />}
/>
<Route path="menu" element={<FoodHomePage />} />
</Route>
<Route
  path="/resturent-dashboard/*"
  element={<ResturentRoutes />}
/>
<Route path="/admin-dashboard/*" element={<AdminRoutes />} />
<Route path="/driver-dashboard/*" element={<DriverRoutes />} />
</Routes>
<MobileBottomNav />
</DriverProvider>
</CartProvider>
</AuthProvider>
</BrowserRouter>
);
}

```

export default App;

src/main.tsx

```

import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.tsx'
import './index.css'

createRoot(document.getElementById('root')!).render(

```

```
<StrictMode>
  <App />
</StrictMode>,
)
```

src/pages/admin/AddRestaurant.tsx

```
import { useState } from "react";
import ResturentTitle from "../../components/UI/ResturentTitle";
import { registerRestaurant } from "../../utils/api";
```

```
interface RestaurantFormData {
```

```
  restaurantName: string;
  contactPerson: string;
  phoneNumber: string;
  businessType: string;
  cuisineType: string;
  operatingHours: string;
  deliveryRadius: string;
  taxId: string;
  streetAddress: string;
  city: string;
  state: string;
  zipCode: string;
  country: string;
  email: string;
  password?: string;
  agreeTerms?: boolean;
  businessLicense: File | null;
  foodSafetyCert: File | null;
  exteriorPhoto: File | null;
  logo: File | null;
```

```
}
```

```
const AddResturent = () => {
```

```
  const [formData, setFormData] = useState<RestaurantFormData>({
    restaurantName: "",
    contactPerson: "",
    phoneNumber: "",
    businessType: "",
    cuisineType: "",
    operatingHours: "",
    deliveryRadius: "",
    taxId: "",
    streetAddress: "",
    city: "",
    state: "",
    zipCode: "",
    country: "",
    email: "",
    password: ""
```

```
agreeTerms: false,
businessLicense: null,
foodSafetyCert: null,
exteriorPhoto: null,
logo: null,
});
const [loading, setLoading] = useState<boolean>(false);
const [error, setError] = useState<string | null>(null);
const [success, setSuccess] = useState<string | null>(null);

// Options for dropdowns
const businessTypes = ["Restaurant", "Fast Food Chain", "Cafe", "Food Truck", "Catering"];
const cuisineTypes = ["Italian", "American", "Mexican", "Chinese", "Indian", "Other"];

// Handle input changes for text, select, and checkbox
const handleChange = (
  e: React.ChangeEvent<HTMLInputElement | HTMLSelectElement | HTMLTextAreaElement>
) => {
  const { name, value, type } = e.target;
  const checked = (e.target as HTMLInputElement).checked;
  setFormData((prev) => ({
    ...prev,
    [name]: type === "checkbox" ? checked : value,
  }));
};

// Handle file input changes
const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const { name, files } = e.target;
  if (files && files[0]) {
    // Validate file type (e.g., PDF or image)
    const validTypes = ["application/pdf", "image/jpeg", "image/png"];
    if (!validTypes.includes(files[0].type)) {
      setError("Please upload a PDF or image (JPEG/PNG) file.");
      return;
    }
    // Validate file size (e.g., max 5MB)
    if (files[0].size > 5 * 1024 * 1024) {
      setError("File size must be less than 5MB.");
      return;
    }
    setFormData((prev) => ({
      ...prev,
      [name]: files[0],
    }));
  } else {
    // Allow clearing file input
    setFormData((prev) => ({
      ...prev,
      [name]: null,
    }));
  }
};
```

```
// Handle form submission
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  setError(null);
  setSuccess(null);

// Validate required fields
const requiredFields = [
  "restaurantName",
  "contactPerson",
  "phoneNumber",
  "businessType",
  "cuisineType",
  "operatingHours",
  "deliveryRadius",
  "taxId",
  "streetAddress",
  "city",
  "state",
  "zipCode",
  "country",
  "email",
  "password",
];
for (const field of requiredFields) {
  if (!formData[field as keyof RestaurantFormData]) {
    setError(`Please fill in the ${field} field.`);
    return;
  }
}
if (!formData.agreeTerms) {
  setError("You must agree to the terms and conditions.");
  return;
}

// Validate email format
const emailRegex = /^[^@\s]+@[^\s@]+\.[^\s@]+$/;
if (!emailRegex.test(formData.email)) {
  setError("Please enter a valid email address.");
  return;
}

// Validate password length
if (formData.password && formData.password.length < 8) {
  setError("Password must be at least 8 characters long.");
  return;
}

// Log formData for debugging
console.log("Submitting Form Data:", {
  ...formData,
  businessLicense: formData.businessLicense ? formData.businessLicense.name : null,
```

```
foodSafetyCert: formData.foodSafetyCert ? formData.foodSafetyCert.name : null,
exteriorPhoto: formData.exteriorPhoto ? formData.exteriorPhoto.name : null,
logo: formData.logo ? formData.logo.name : null,
});

 setLoading(true);
try {
 await registerRestaurant(formData);
setSuccess("Restaurant registered successfully!");
// Reset form
setFormData({
  restaurantName: "",
  contactPerson: "",
  phoneNumber: "",
  businessType: "",
  cuisineType: "",
  operatingHours: "",
  deliveryRadius: "",
  taxId: "",
  streetAddress: "",
  city: "",
  state: "",
  zipCode: "",
  country: "",
  email: "",
  password: "",
  agreeTerms: false,
  businessLicense: null,
  foodSafetyCert: null,
  exteriorPhoto: null,
  logo: null,
});
} catch (err: any) {
// Extract error message from response
let errorMessage = "Failed to register restaurant. Please try again.";
if (err instanceof Response) {
  try {
    const eventData = await err.json();
    errorMessage = eventData.message || errorMessage;
  } catch (jsonError) {
    console.error("Error parsing response:", jsonError);
  }
} else {
  errorMessage = err.message || errorMessage;
}
setError(errorMessage);
console.error("Registration error details:", err);
} finally {
  setLoading(false);
}
};

return (
```

```

<>
<div className="p-4">
  <ResturentTitle text="Add New Restaurant" />
</div>

<div className="mx-auto w-full max-w-full sm:max-w-4xl lg:max-w-6xl">
  {success && (
    <div className="mb-4 p-4 bg-green-100 text-green-700 dark:bg-green-700 dark:text-green-200 rounded-md">
      {success}
    </div>
  )}
  {error && (
    <div className="mb-4 p-4 bg-red-100 text-red-700 dark:bg-red-700 dark:text-red-200 rounded-md">
      {error}
    </div>
  )}
</form>
  onSubmit={handleSubmit}
  className="flex flex-col gap-6 bg-white dark:bg-gray-700 p-4 sm:p-6 rounded-md"
>
<div className="grid grid-cols-1 md:grid-cols-2 gap-4">
  {/* Left Column */}
  <div className="flex flex-col gap-4">
    {/* Restaurant Name */}
    <div className="flex flex-col gap-2">
      <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
        Restaurant Name
      </label>
      <input
        type="text"
        name="restaurantName"
        value={formData.restaurantName}
        onChange={handleChange}
        required
        className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
        placeholder="Enter restaurant name"
      />
    </div>

    {/* Phone Number */}
    <div className="flex flex-col gap-2">
      <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
        Phone Number
      </label>
      <input
        type="tel"
        name="phoneNumber"
        value={formData.phoneNumber}
        onChange={handleChange}
        required
      />
    </div>
  </div>
</div>

```

```
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
      placeholder="+1-123-456-7890"
    />
</div>

/* Business Type */
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Business Type
  </label>
  <select
    name="businessType"
    value={formData.businessType}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
  >
    <option value="">Select business type</option>
    {businessTypes.map((type) => (
      <option key={type} value={type}>
        {type}
      </option>
    )))
  </select>
</div>

/* Operating Hours */
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Operating Hours
  </label>
  <input
    type="text"
    name="operatingHours"
    value={formData.operatingHours}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="e.g., 10 AM - 10 PM"
  />
</div>

/* Tax ID */
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Tax ID
  </label>
```

```
<input
  type="text"
  name="taxId"
  value={formData.taxId}
  onChange={handleChange}
  required
  className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
  placeholder="Enter tax ID"
/>
</div>

{/* Street Address */}
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Street Address
  </label>
  <input
    type="text"
    name="streetAddress"
    value={formData.streetAddress}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="Enter street address"
  />
</div>

{/* State */}
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    State
  </label>
  <input
    type="text"
    name="state"
    value={formData.state}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="Enter state"
  />
</div>
</div>

{/* Right Column */}
<div className="flex flex-col gap-4">
  {/* Contact Person */}
```

```
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Contact Person
  </label>
  <input
    type="text"
    name="contactPerson"
    value={formData.contactPerson}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="Enter contact person"
  />
</div>

{/* Email */}
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Email
  </label>
  <input
    type="email"
    name="email"
    value={formData.email}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="Enter email"
  />
</div>

{/* Password */}
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Password
  </label>
  <input
    type="password"
    name="password"
    value={formData.password}
    onChange={handleChange}
    required
    minLength={8}
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="Enter password"
  />
</div>
```

```
/* Cuisine Type */


<label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Cuisine Type
  </label>
  <select
    name="cuisineType"
    value={formData.cuisineType}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
  >
    <option value="">Select cuisine type</option>
    {cuisineTypes.map((type) => (
      <option key={type} value={type}>
        {type}
      </option>
    )))
  </select>



/* Delivery Radius */


<label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Delivery Radius
  </label>
  <input
    type="text"
    name="deliveryRadius"
    value={formData.deliveryRadius}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="e.g., 5 miles"
  />



/* City */


<label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    City
  </label>
  <input
    type="text"
    name="city"
    value={formData.city}
    onChange={handleChange}
    required
  >


```

```
        className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
            placeholder="Enter city"
        />
    </div>

    {/* Zip Code */}
    <div className="flex flex-col gap-2">
        <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
            Zip Code
        </label>
        <input
            type="text"
            name="zipCode"
            value={formData.zipCode}
            onChange={handleChange}
            required
            pattern="\d{5}(-\d{4})?"
            className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
            placeholder="Enter zip code"
        />
    </div>

    {/* Country */}
    <div className="flex flex-col gap-2">
        <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
            Country
        </label>
        <input
            type="text"
            name="country"
            value={formData.country}
            onChange={handleChange}
            required
            className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
            placeholder="Enter country"
        />
    </div>
</div>
</div>

    {/* File Uploads */}
    <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
        <div className="flex flex-col gap-2">
            <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
                Business License (PDF/Image)
            </label>
            <input
```

```
        type="file"
        name="businessLicense"
        onChange={handleFileChange}
        accept=".pdf,image/jpeg,image/png"
        className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
      >
    </div>
    <div className="flex flex-col gap-2">
      <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
        Food Safety Certificate (PDF/Image)
      </label>
      <input
        type="file"
        name="foodSafetyCert"
        onChange={handleFileChange}
        accept=".pdf,image/jpeg,image/png"
        className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
      >
    </div>
    <div className="flex flex-col gap-2">
      <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
        Exterior Photo (Image)
      </label>
      <input
        type="file"
        name="exteriorPhoto"
        onChange={handleFileChange}
        accept="image/jpeg,image/png"
        className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
      >
    </div>
    <div className="flex flex-col gap-2">
      <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
        Logo (Image)
      </label>
      <input
        type="file"
        name="logo"
        onChange={handleFileChange}
        accept="image/jpeg,image/png"
        className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
      >
    </div>
  </div>
```

```

/* Agree Terms */
<div className="flex items-center gap-2">
  <input
    type="checkbox"
    name="agreeTerms"
    checked={formData.agreeTerms}
    onChange={handleChange}
    className="h-4 w-4 text-orange-600 focus:ring-orange-500 border-gray-300 dark:border-gray-600 rounded" />
  <label className="text-sm text-gray-700 dark:text-gray-200">
    I agree to the{" "}
    <a href="/terms" className="text-orange-600 underline">
      terms and conditions
    </a>
  </label>
</div>

/* Submit Button */
<button
  type="submit"
  disabled={loading}
  className={`w-full py-2 px-6 text-white rounded-md transition-colors text-center ${loading ? "bg-gray-400 cursor-not-allowed" : "bg-orange-600 hover:bg-orange-700 dark:bg-orange-500 dark:hover:bg-orange-600"}`}>
  {loading ? "Submitting..." : "Add Restaurant"}
</button>
</form>
</div>
</>
);
};

export default AddResturent;

```

src/pages/admin/AddUser.tsx

```

import { useState, useEffect } from "react";
import RestaurantTitle from "../../components/UI/ResturentTitle"; // Adjusted to RestaurantTitle
import { register } from "../../utils/api";

interface UserFormData {
  firstName: string;
  lastName: string;
  email: string;
  password: string;
  role: "CUSTOMER" | "RESTAURANT" | "DELIVERY" | "ADMIN" | "";
  street: string;
}

```

```

city: string;
state: string;
zipCode: string;
country: string;
agreeTerms: boolean;
}

const AddUser = () => {
  const [formData, setFormData] = useState<UserFormData>({
    firstName: "",
    lastName: "",
    email: "",
    password: "",
    role: "",
    street: "",
    city: "",
    state: "",
    zipCode: "",
    country: "",
    agreeTerms: false,
  });
  const [loading, setLoading] = useState<boolean>(false);
  const [error, setError] = useState<string | null>(null);
  const [success, setSuccess] = useState<string | null>(null);

  // Options for role dropdown
  const roleOptions = ["CUSTOMER", "RESTAURANT", "DELIVERY", "ADMIN"];

  // Generate random password
  const generatePassword = () => {
    const length = 12;
    const charset =
      "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*";
    let password = "";
    for (let i = 0; i < length; i++) {
      const randomIndex = Math.floor(Math.random() * charset.length);
      password += charset[randomIndex];
    }
    return password;
  };

  // Set initial password on component mount
  useEffect(() => {
    setFormData((prev) => ({ ...prev, password: generatePassword() }));
  }, []);

  // Handle input changes for text, select, and checkbox
  const handleChange = (
    e: React.ChangeEvent<HTMLInputElement | HTMLSelectElement>
  ) => {
    const { name, value, type } = e.target;
    const checked = (e.target as HTMLInputElement).checked;
    setFormData((prev) => ({

```

```
...prev,
  [name]: type === "checkbox" ? checked : value,
});
};

// Handle form submission
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  setError(null);
  setSuccess(null);

  // Validate required fields
  const requiredFields = [
    "firstName",
    "lastName",
    "email",
    "password",
    "role",
    "street",
    "city",
    "state",
    "zipCode",
    "country",
  ];
  for (const field of requiredFields) {
    if (!formData[field as keyof UserFormData]) {
      setError(`Please fill in the ${field} field.`);
      return;
    }
  }
  if (!formData.agreeTerms) {
    setError("You must agree to the terms and conditions.");
    return;
  }

  // Validate email format
  const emailRegex = /^[^@\s]+@[^\s@]+\.[^\s@]+$/;
  if (!emailRegex.test(formData.email)) {
    setError("Please enter a valid email address.");
    return;
  }

  // Validate password length
  if (formData.password.length < 8) {
    setError("Password must be at least 8 characters long.");
    return;
  }

  // Prepare data for register API
  const registerData = {
    email: formData.email,
    password: formData.password,
    firstName: formData.firstName,
```

```
lastName: formData.lastName,
role: formData.role as "CUSTOMER" | "RESTAURANT" | "DELIVERY" | "ADMIN",
address: {
  street: formData.street,
  city: formData.city,
  state: formData.state,
  zipCode: formData.zipCode,
  country: formData.country,
},
};

// Log formData for debugging
console.log("Submitting Form Data:", registerData);

 setLoading(true);
try {
  // Register user
  await register(registerData);

  // // Send welcome email
  // await sendWelcomeEmail({
  //   email: formData.email,
  //   password: formData.password,
  // });
}

setSuccess("User registered successfully! A welcome email has been sent.");
// Reset form with new password
setFormData({
  firstName: "",
  lastName: "",
  email: "",
  password: generatePassword(),
  role: "",
  street: "",
  city: "",
  state: "",
  zipCode: "",
  country: "",
  agreeTerms: false,
});
} catch (err: any) {
  // Extract error message from response
  let errorMessage = "Failed to register user. Please try again.";
  if (err instanceof Response) {
    try {
      const errorData = await err.json();
      errorMessage = errorData.message || errorMessage;
    } catch (jsonError) {
      console.error("Error parsing response:", jsonError);
    }
  } else {
    errorMessage = err.message || errorMessage;
  }
}
```

```

setError(errorMessage);
console.error("Registration error details:", err);
} finally {
  setLoading(false);
}
};

return (
<>
<div className="p-4">
  <RestaurantTitle text="Add New User" />
</div>

<div className="mx-auto w-full max-w-full sm:max-w-4xl lg:max-w-6xl">
  {success && (
    <div className="mb-4 p-4 bg-green-100 text-green-700 dark:bg-green-700 dark:text-green-200 rounded-md">
      {success}
    </div>
  )}
  {error && (
    <div className="mb-4 p-4 bg-red-100 text-red-700 dark:bg-red-700 dark:text-red-200 rounded-md">
      {error}
    </div>
  )}
</form>
  <form
    onSubmit={handleSubmit}
    className="flex flex-col gap-6 bg-white dark:bg-gray-700 p-4 sm:p-6 rounded-md">
    <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
      {/* Left Column */}
      <div className="flex flex-col gap-4">
        {/* First Name */}
        <div className="flex flex-col gap-2">
          <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
            First Name
          </label>
          <input
            type="text"
            name="firstName"
            value={formData.firstName}
            onChange={handleChange}
            required
            className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
            placeholder="Enter first name"
          />
        </div>
        {/* Email */}
        <div className="flex flex-col gap-2">
          <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">

```

```
Email
</label>
<input
  type="email"
  name="email"
  value={formData.email}
  onChange={handleChange}
  required
  className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
  placeholder="Enter email"
/>
</div>

{/* Role */}
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Role
  </label>
  <select
    name="role"
    value={formData.role}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
  >
    <option value="">Select role</option>
    {roleOptions.map((role) => (
      <option key={role} value={role}>
        {role}
      </option>
    ))}
  </select>
</div>

{/* Street */}
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Street Address
  </label>
  <input
    type="text"
    name="street"
    value={formData.street}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="Enter street address"
```

```
/>
</div>

{/* State */
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    State
  </label>
  <input
    type="text"
    name="state"
    value={formData.state}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="Enter state"
  />
</div>
</div>

{/* Right Column */
<div className="flex flex-col gap-4">
  {/* Last Name */}
  <div className="flex flex-col gap-2">
    <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
      Last Name
    </label>
    <input
      type="text"
      name="lastName"
      value={formData.lastName}
      onChange={handleChange}
      required
      className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
      placeholder="Enter last name"
    />
  </div>

  {/* Password (Display Only) */}
  <div className="flex flex-col gap-2">
    <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
      Generated Password
    </label>
    <input
      type="text"
      name="password"
      value={formData.password}
      readOnly
    />
  </div>
</div>
```

```
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-gray-100 dark:bg-gray-600 border-gray-300 dark:border-gray-600 focus:outline-none"
      placeholder="Generated password"
    />
</div>

/* City */
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    City
  </label>
  <input
    type="text"
    name="city"
    value={formData.city}
    onChange={handleChange}
    required
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="Enter city"
  />
</div>

/* Zip Code */
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Zip Code
  </label>
  <input
    type="text"
    name="zipCode"
    value={formData.zipCode}
    onChange={handleChange}
    required
    pattern="\d{5}(-\d{4})?"
    className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-orange-500"
    placeholder="Enter zip code"
  />
</div>

/* Country */
<div className="flex flex-col gap-2">
  <label className="block text-sm font-bold text-gray-700 dark:text-gray-200">
    Country
  </label>
  <input
    type="text"
    name="country"
    value={formData.country}
    onChange={handleChange}
```

```
required
  className="w-full py-2 px-3 border rounded-md shadow-sm text-gray-700 dark:text-gray-200 bg-
white dark:bg-gray-700 border-gray-300 dark:border-gray-600 focus:outline-none focus:ring-2 focus:ring-
orange-500"
    placeholder="Enter country"
  />
</div>
</div>
</div>

{/* Agree Terms */}
<div className="flex items-center gap-2">
  <input
    type="checkbox"
    name="agreeTerms"
    checked={formData.agreeTerms}
    onChange={handleChange}
    className="h-4 w-4 text-orange-600 focus:ring-orange-500 border-gray-300 dark:border-gray-600
rounded"
  />
  <label className="text-sm text-gray-700 dark:text-gray-200">
    I agree to the{" "}
    <a href="/terms" className="text-orange-600 underline">
      terms and conditions
    </a>
  </label>
</div>

{/* Submit Button */}
<button
  type="submit"
  disabled={loading}
  className={`w-full py-2 px-6 text-white rounded-md transition-colors text-center ${{
    loading
      ? "bg-gray-400 cursor-not-allowed"
      : "bg-orange-600 hover:bg-orange-700 dark:bg-orange-500 dark:hover:bg-orange-600"
  }}`}
>
  {loading ? "Submitting..." : "Add User"}
</button>
</form>
</div>
</>
);
};

export default AddUser;
```

src/pages/admin/AllRestaurent.tsx

```
import React, { useEffect, useState } from "react";
import AdminResturentTable from "../../components/admin/AdminResturentTable";
import ResturentTitle from "../../components/UI/ResturentTitle";
import { getAllRestaurants } from "../../utils/api";

export interface Restaurant {
  _id: string;
  restaurantName: string;
  contactPerson: string;
  phoneNumber: string;
  businessType: string;
  cuisineType: string;
  operatingHours: string;
  deliveryRadius: string;
  taxId: string;
  address: {
    streetAddress: string;
    city: string;
    state: string;
    zipCode: string;
    country: string;
  };
  email: string;
  businessLicense?: string | null;
  foodSafetyCert?: string | null;
  exteriorPhoto?: string | null;
  logo?: string | null;
  status?: string;
}

// Table headers for restaurant details
const tableHeaders: string[] = [
  "Restaurant Name",
  "Contact Person",
  "Phone Number",
  "Business Type",
  "Cuisine Type",
  "Operating Hours",
  "Delivery Radius",
  "Tax ID",
  "Street Address",
  "City",
  "State",
  "Zip Code",
  "Country",
  "Email",
  "Business License",
  "Food Safety Certificate",
  "Exterior Photo",
  "Logo",
  "Status",
]
```

```
];
const RestaurantDetailsPage: React.FC = () => {
  const [restaurants, setRestaurants] = useState<Restaurant[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);
  const [page, setPage] = useState<number>(1);
  const [totalPages, setTotalPages] = useState<number>(1);
  const limit = 10;

  // Fetch restaurants on component mount or page change
  useEffect(() => {
    const fetchRestaurants = async () => {
      try {
        setLoading(true);
        const response = await getAllRestaurants(page, limit);
        const formattedRestaurants: Restaurant[] = response.data.map((item: any) => ({
          _id: item._id,
          restaurantName: item.restaurantName,
          contactPerson: item.contactPerson,
          phoneNumber: item.phoneNumber,
          businessType: item.businessType,
          cuisineType: item.cuisineType,
          operatingHours: item.operatingHours,
          deliveryRadius: item.deliveryRadius,
          taxId: item.taxId,
          address: {
            streetAddress: item.address.streetAddress,
            city: item.address.city,
            state: item.address.state,
            zipCode: item.address.zipCode,
            country: item.address.country,
          },
          email: item.email,
          businessLicense: item.businessLicense || null,
          foodSafetyCert: item.foodSafetyCert || null,
          exteriorPhoto: item.exteriorPhoto || null,
          logo: item.logo || null,
          status: item.status ? item.status.trim().toLowerCase() : "pending",
        }));
        setRestaurants(formattedRestaurants);
        setTotalPages(response.pagination.totalPages);
        setError(null);
      } catch (err: any) {
        setError(err.message || "Failed to load restaurants. Please try again.");
        console.error(err);
      } finally {
        setLoading(false);
      }
    };
    fetchRestaurants();
  }, [page]);
```

```

// Handle delete action from table
const handleDelete = (restaurantId: string) => {
  setRestaurants((prev) => prev.filter((restaurant) => restaurant._id !== restaurantId));
};

// Handle page change
const handlePageChange = (newPage: number) => {
  if (newPage >= 1 && newPage <= totalPages) {
    setPage(newPage);
  }
};

return (
  <div className="p-4">
    <ResturentTitle text="Restaurant Details" />
    {loading && <div className="text-center p-4 text-gray-600 dark:text-gray-300">Loading...</div>}
    {error && <div className="text-center p-4 text-red-500">{error}</div>}
    {!loading && !error && (
      <>
        <AdminResturentTable headers={tableHeaders} data={restaurants} onDelete={handleDelete} />
        <div className="flex justify-center mt-4 space-x-2">
          <button
            onClick={() => handlePageChange(page - 1)}
            disabled={page === 1}
            className={`px-4 py-2 rounded-lg ${page === 1
              ? "bg-gray-300 cursor-not-allowed"
              : "bg-indigo-500 text-white hover:bg-indigo-600"
            }`}
          >
            Previous
          </button>
          <span className="px-4 py-2 text-gray-800 dark:text-gray-200">
            Page {page} of {totalPages}
          </span>
          <button
            onClick={() => handlePageChange(page + 1)}
            disabled={page === totalPages}
            className={`px-4 py-2 rounded-lg ${page === totalPages
              ? "bg-gray-300 cursor-not-allowed"
              : "bg-indigo-500 text-white hover:bg-indigo-600"
            }`}
          >
            Next
          </button>
        </div>
      </>
    )}
  </div>
);

```

```
export default RestaurantDetailsPage;
```