Full Stack Development with MERN

Project Documentation format

1. Introduction

• **Project Title:** OrderOnTheGo: Your On-Demand Food Ordering Solution

• **Team Members and Roles:**

Name	Role
Eswar	Full-Stack Developers (Frontend & Backend)
Gowthami	Full-Stack Developers (Frontend & Backend)
Navitha	Full-Stack Developers (Backend)
Bindhu	Full-Stack Developers (Frontend)

2. Project Overview

• **Purpose:**

Order on the Go is a comprehensive multi-role food ordering and restaurant management platform designed to:

- For Customers: Browse restaurants, explore menus, place orders, track delivery, and review foods
- For Restaurant Owners: Manage restaurant profiles, menus, staff, orders, subscriptions, and analytics
- For Staff: Process orders, update statuses, and assist with order fulfillment
- For Admins: Oversee the entire platform, approve restaurants, manage users, and view analytics

Goals

1. Provide a seamless ordering experience for end-users
2. Empower restaurant owners with tools for digital transformation
3. Enable efficient order fulfillment through staff workflows
4. Implement subscription-based monetization model

5. Ensure secure payments and data integrity
6. Scale to support thousands of concurrent users

Features:

User Features:

- Email/phone-based registration and verification
- Browse restaurants and food menus
- Add items to cart with quantity management
- Place orders with delivery address selection
- Razorpay payment integration with signature verification
- Real-time order status tracking
- Submit food reviews and ratings
- Profile management with image upload

Restaurant Owner Features:

- Restaurant registration and admin approval workflow
- Food/menu management (CRUD operations)
- Toggle food availability
- View incoming orders and update status
- Staff invite system with token-based registration
- Staff approval workflow
- Subscription plan management (FREE/BASIC/PREMIUM)
- Restaurant analytics and revenue tracking

Staff Features:

- Token-based registration via owner invite
- View assigned orders
- Request order status changes (subject to owner approval)

- Profile management

Admin Features:

- User management (view, suspend, delete)
- Restaurant approval/rejection workflow
- Order and food management across all restaurants
- Revenue analytics by restaurant and subscription plan
- Order distribution and status analytics
- Staff management

System Features:

- Role-based access control (RBAC): USER/RESTAURANT/STAFF/ADMIN
- JWT-based authentication
- Email verification (SendGrid)
- SMS OTP verification (Twilio)
- Image upload for profiles and restaurants (max 5MB)
- Subscription limits enforced by plan
- Order status lifecycle tracking
- Payment tracking and reconciliation

3. Architecture

3.1 Frontend Architecture (React + Vite)

Technology Stack:

- **Framework:** React 18+
- **Build Tool:** Vite
- **Styling:** Tailwind CSS
- **HTTP Client:** Axios
- **State Management:** Context API (AuthContext)
- **Routing:** React Router v6

Architecture Pattern:

- **Component-Based:** Modular, reusable components
- **Context-Based Auth:** AuthProvider manages user authentication state globally
- **Protected Routes:** ProtectedRoute component enforces role-based access
- **API Layer:** Centralized Axios instance for all backend calls

Key Components:

```
Frontend/
├── src/
│   ├── components/        # Reusable UI components
│   │   ├── Badge.jsx
│   │   ├── Modal.jsx
│   │   ├── Navbar.jsx
│   │   ├── FoodCard.jsx
│   │   ├── ProtectedRoute.jsx
│   │   ├── VerificationModal.jsx
│   │   └── ...
│   ├── pages/             # Page components by role
│   │   ├── auth/          # Login, Register, OTP verification
│   │   ├── user/          # User dashboard, orders, reviews
│   │   ├── restaurant/    # Restaurant dashboard, menu, staff, analytics
│   │   ├── staff/         # Staff dashboard, order queue
│   │   ├── admin/         # Admin dashboard, analytics, management
│   │   └── info/          # Info pages
│   ├── context/           # AuthProvider, AuthContext
│   ├── api/               # Axios configuration
│   ├── App.jsx            # Main app component
│   └── main.jsx           # Entry point
```

Data Flow:

```
User Input → Component → Axios API Call → Backend → MongoDB
                ↓
          Auth Context (state)
                ↓
          Protected Routes (role check)
```

3.2 Backend Architecture (Node.js + Express)

Technology Stack:

- **Runtime:** Node.js 18+
- **Framework:** Express.js
- **Database:** MongoDB with Mongoose ODM
- **Authentication:** JWT (JSON Web Tokens)
- **Password Hashing:** bcryptjs
- **Email:** SendGrid
- **SMS:** Twilio
- **Payments:** Razorpay
- **File Upload:** Express-based file handling

Architecture Pattern:

- **MVC (Modified):** Models, Controllers, Routes, Middleware layers
- **Service-Oriented:** Business logic in controllers and utilities
- **Middleware Chain:** Authentication, authorization, error handling
- **Modular Routes:** Separate route files by feature (auth, orders, restaurants, etc.)

Key Layers:

Request

↓

CORS & Body Parser Middleware

↓

Routes (authRoutes, cartRoutes, orderRoutes, etc.)

↓

Authentication Middleware (JWT verification)

↓

Authorization Middleware (Role-based check)

↓

Controllers (Business logic)

↓

Service Layer (Data queries, external API calls)

↓

Mongoose Models (Database schema & validation)

↓

MongoDB Database

↓

Response (JSON)

Core Modules:

Module	Files	Purpose
Auth	authController, authRoutes, generateToken	User registration, login, JWT generation
Restaurant	restaurantController, restaurantRoutes	Restaurant CRUD, approvals, profiles
Food	foodController, foodRoutes	Menu management, food CRUD
Cart	cartController, cartRoutes	Cart operations (add, remove, view)
Order	orderController, orderRoutes	Order placement, status updates, tracking
Payment	paymentController, paymentRoutes	Razorpay integration, signature verification
Review	reviewController, reviewRoutes	Food reviews, ratings, aggregation
Staff	staffController, staffRoutes	Staff invites, approval, status requests
Subscription	subscriptionController, subscriptionRoutes	Plans, subscription management, limits
Admin	adminController, adminRoutes	User management, analytics, approvals
Verification	verificationController, verificationRoutes	Email/SMS OTP verification

Middleware:

- authMiddleware.js – JWT verification and user extraction
- subscriptionMiddleware.js – Feature limit enforcement based on subscription plan

3.3 Database Architecture (MongoDB + Mongoose)

Database Name: `orderonthego`

Collections & Schema Overview:

Collection	Purpose	Key Fields
`users`	User accounts (customers, restaurant owners, staff, admins)	email, password, userType, emailVerified, phoneVerified, profileImage
`restaurants`	Restaurant profiles and metadata	ownerId, title, address, cuisineType, status (pending/approved), profileImage
`foods`	Menu items	restaurantId, title, price, category, description, mainImg, isAvailable
`carts`	Shopping carts (user-based)	userId, restaurantId, items[], totalAmount
`orders`	Customer orders	userId, restaurantId, items[], totalAmount, status, paymentId, paymentStatus
`payments`	Payment records	userId, orderId, restaurantId, amount, paymentMethod, gatewayPaymentId, paymentStatus
`reviews`	Food reviews and ratings	foodId, userId, rating (1-5), comment, createdAt
`staffinvites`	Staff invitation tokens	email, restaurantId, token, status (pending/accepted), expiresAt
`subscriptions`	Restaurant subscriptions	userId (restaurant owner), plan (FREE/BASIC/PREMIUM), status, billingCycleStart, billingCycleEnd
`orderstatusrequests`	Staff status change requests	orderId, staffId, requestedStatus, approvalStatus (pending/approved/rejected), createdAt

Data Relationships:

User (owner) ←→ Restaurant ←→ Food
↓
Order ← Cart
↓
Payment
↓
Review

Restaurant ←→ StaffInvite → User (staff)
↓
Subscription
↓
OrderStatusRequest

Indexes:

- users.email – Unique index for login
- staffinvites.token – Unique index for invite validation
- reviews.(foodId, userId) – Compound unique index to prevent duplicate reviews
- Order & payment records indexed by date for analytics queries

4. Setup Instructions

4.1 Prerequisites

Ensure the following software is installed on your system:

Software	Version	Download
Node.js	18+ (LTS)	https://nodejs.org
npm or yarn	8+	Included with Node.js
MongoDB Atlas Account	Cloud	https://www.mongodb.com/atlas
Git	Latest	https://git-scm.com
Postman (optional)	Latest	https://postman.com

API Keys Required:

- Razorpay: RAZORPAY_KEY_ID, RAZORPAY_KEY_SECRET
- SendGrid: SENDGRID_API_KEY, SENDGRID_SENDER
- Twilio: TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN, TWILIO_PHONE_NUMBER

4.2 Installation & Setup

Step 1: Clone the Repository

git clone https://github.com/yourusername/orderonthego.git

cd orderonthego

Step 2: Backend Setup

cd backend

Install dependencies

npm install

Create environment variables file

cp .env.example .env

Edit .env with your configuration (see template below)

Backend .env Template:

```
# Server Config
PORT=5000
NODE_ENV=development

# MongoDB
MONGO_URI=mongodb+srv://username:password@cluster.mongodb.net/orderonthego

# JWT
JWT_SECRET=your_jwt_secret_key_here_keep_it_secure_and_long

# Frontend URL (CORS)
FRONTEND_BASE_URL=http://localhost:5173

# Razorpay
RAZORPAY_KEY_ID=your_razorpay_key_id
RAZORPAY_KEY_SECRET=your_razorpay_key_secret

# SendGrid
SENDGRID_API_KEY=your_sendgrid_api_key
SENDGRID_SENDER=noreply@orderonthego.com

# Twilio
TWILIO_ACCOUNT_SID=your_twilio_account_sid
TWILIO_AUTH_TOKEN=your_twilio_auth_token
TWILIO_PHONE_NUMBER=+1234567890
```

Step 3: Frontend Setup

cd ../frontend

Install dependencies

npm install

Create environment variables file (if needed)

cp .env.example .env

Edit .env

Frontend .env Template:

```
VITE_API_BASE_URL=http://localhost:5000

VITE_APP_NAME=Order on the Go
```

Step 4: MongoDB Atlas Setup

1. Go to https://www.mongodb.com/atlas
2. Create a free cluster
3. Create a database user (username & password)
4. Whitelist your IP (0.0.0.0/0 for development)
5. Get connection string and add to backend .env

Step 5: Configure API Keys

1. **Razorpay:** Sign up at https://razorpay.com → Dashboard → API Keys
2. **SendGrid:** Sign up at https://sendgrid.com → API Keys → Create API Key
3. **Twilio:** Sign up at https://www.twilio.com → Get credentials from Console

5. Folder Structure

5.1 Client (Frontend) Structure

```
frontend/
├── public/                    # Static assets
│   └── _redirects             # Netlify redirect config
├── src/
│   ├── api/
│   │   └── axios.js           # Axios instance & configuration
│   ├── components/            # Reusable UI components
│   │   ├── Badge.jsx
│   │   ├── EmptyState.jsx
│   │   ├── FoodCard.jsx
│   │   ├── Footer.jsx
│   │   ├── LoadingSpinner.jsx
│   │   ├── Modal.jsx
│   │   ├── Navbar.jsx
│   │   ├── Pagination.jsx
│   │   ├── ProtectedRoute.jsx     # Role-based route protection
│   │   ├── ReviewCard.jsx
│   │   ├── SearchBar.jsx
│   │   ├── SubscriptionBadge.jsx
│   │   ├── VerificationModal.jsx
│   │   └── COMPONENTS.md          # Component documentation
│   ├── context/
│   │   ├── AuthContext.jsx        # Auth state definition
│   │   └── AuthProvider.jsx       # Auth provider component
│   ├── pages/
│   │   ├── auth/
│   │   │   ├── AdminLogin.jsx
│   │   │   ├── Login.jsx
│   │   │   ├── Register.jsx
│   │   │   ├── RestaurantLogin.jsx
│   │   │   └── RestaurantRegister.jsx
```

```
│   │   │   ├── StaffInvite.jsx
│   │   │   ├── UserLogin.jsx
│   │   │   └── UserRegister.jsx
│   │   ├── user/                # Customer pages
│   │   │   ├── Dashboard.jsx
│   │   │   ├── Orders.jsx
│   │   │   ├── OrderDetails.jsx
│   │   │   ├── Checkout.jsx
│   │   │   └── Profile.jsx
│   │   ├── restaurant/          # Restaurant owner pages
│   │   │   ├── Dashboard.jsx
│   │   │   ├── Menu.jsx
│   │   │   ├── Orders.jsx
│   │   │   ├── Staff.jsx
│   │   │   ├── Subscriptions.jsx
│   │   │   ├── Analytics.jsx
│   │   │   └── Profile.jsx
│   │   ├── staff/               # Staff pages
│   │   │   ├── Dashboard.jsx
│   │   │   ├── Orders.jsx
│   │   │   └── Profile.jsx
│   │   ├── admin/               # Admin pages
│   │   │   ├── Dashboard.jsx
│   │   │   ├── Users.jsx
│   │   │   ├── Restaurants.jsx
│   │   │   ├── Orders.jsx
│   │   │   ├── Foods.jsx
│   │   │   ├── Analytics.jsx
│   │   │   └── Profile.jsx
│   │   └── info/                # Static info pages
│   │       ├── About.jsx
│   │       ├── Contact.jsx
│   │       └── FAQ.jsx
│   ├── App.jsx                  # Main app component
│   ├── App.css
│   ├── index.css
│   └── main.jsx                 # Entry point
├── index.html
├── package.json
├── vite.config.js
├── tailwind.config.js
├── postcss.config.js
└── README.md
```

5.2 Server (Backend) Structure

```
backend/
├── src/
│   ├── app.js                   # Express app setup
│   ├── config/
│   │   └── db.js                # MongoDB connection
│   ├── controllers/             # Business logic
│   │   ├── adminController.js
│   │   ├── authController.js    # Register, login
│   │   ├── cartController.js
│   │   ├── foodController.js
│   │   ├── orderController.js
│   │   ├── paymentController.js
│   │   ├── restaurantController.js
│   │   ├── reviewController.js
│   │   ├── staffController.js
│   │   ├── subscriptionController.js
```

```
    │   └── verificationController.js
    ├── middlewares/
    │   ├── authMiddleware.js       # JWT verification
    │   └── subscriptionMiddleware.js # Limit enforcement
    ├── models/                     # Mongoose schemas
    │   ├── User.js
    │   ├── Restaurant.js
    │   ├── Food.js
    │   ├── Cart.js
    │   ├── Order.js
    │   ├── Payment.js
    │   ├── Review.js
    │   ├── StaffInvite.js
    │   ├── Subscription.js
    │   └── OrderStatusRequest.js
    ├── routes/                     # API routes
    │   ├── adminRoutes.js
    │   ├── authRoutes.js
    │   ├── cartRoutes.js
    │   ├── foodRoutes.js
    │   ├── orderRoutes.js
    │   ├── paymentRoutes.js
    │   ├── restaurantRoutes.js
    │   ├── reviewRoutes.js
    │   ├── staffRoutes.js
    │   ├── subscriptionRoutes.js
    │   └── verificationRoutes.js
    ├── seeds/
    │   └── seedFoods.js            # Sample data seeding
    └── utils/
        ├── generateToken.js        # JWT token generation
        └── uploadHandler.js        # File upload handling
├── uploads/
    └── profiles/                   # Profile images storage
├── server.js                       # Entry point
├── package.json
├── .env.example
└── README.md
```

6. Running the Application

6.1 Start Backend Server

```
cd backend
# Development mode (with nodemon auto-reload)
npm run dev
# Production mode
npm start
```

Expected Output:

```
Server running on http://localhost:5000
MongoDB connected successfully
```

6.2 Start Frontend Server

```
cd frontend
# Development mode
npm run dev
# Production build
npm run build
# Preview production build
```

npm run preview
Expected Output:
VITE v4.x.x ready in 500 ms
➜Local: http://localhost:5173/
➜ press h to show help
6.3 Verify Setup
Backend Health Check:
curl http://localhost:5000/api/health
Frontend Access: Open http://localhost:5173 in your browser
Seed Sample Data (Optional):
cd backend
npm run seed:foods

7. API Documentation

API Base URL

- **Development:** http://localhost:5000
- **Production:** [Your deployed backend URL]

Authentication

All protected endpoints require:
Authorization: Bearer <JWT_TOKEN>

Core API Endpoints

Authentication Endpoints

Method	Endpoint	Description	Auth
POST	/api/auth/register	Register new user	❌
POST	/api/auth/login	User login	❌
POST	/api/auth/forgot-password	Request password reset	❌
POST	/api/auth/reset-password	Reset password with token	❌

Restaurant Endpoints

Method	Endpoint	Description	Auth	Role
POST	/api/restaurant	Register restaurant	☑	RESTAURANT
GET	/api/restaurant/:id	Get restaurant details	❌	-
PUT	/api/restaurant/:id	Update restaurant	☑	RESTAURANT/ADMIN
GET	/api/restaurant	List all restaurants	❌	-

Order Endpoints

Method	Endpoint	Description	Auth	Role
POST	/api/orders/place	Place order	☑	USER
GET	/api/orders/:id	Get order details	☑	USER/RESTAURANT/ADMIN
PUT	/api/orders/:id/status	Update order status	☑	RESTAURANT/STAFF/ADMIN
GET	/api/orders	Get user's orders	☑	USER/RESTAURANT

Payment Endpoints

Method	Endpoint	Description	Auth	Role
POST	/api/payments/verify	Verify payment signature	☑	USER
GET	/api/payments/:orderId	Get payment details	☑	USER/ADMIN

Food/Menu Endpoints

Method	Endpoint	Description	Auth	Role
GET	/api/foods	List all foods	✗	-
GET	/api/foods/:id	Get food details	✗	-
POST	/api/foods	Create food item	☑	RESTAURANT
PUT	/api/foods/:id	Update food item	☑	RESTAURANT
DELETE	/api/foods/:id	Delete food item	☑	RESTAURANT

Review Endpoints

Method	Endpoint	Description	Auth	Role
POST	/api/reviews	Submit review	☑	USER
GET	/api/reviews/:foodId	Get food reviews	✗	-
PUT	/api/reviews/:id	Update review	☑	USER

Method	Endpoint	Description	Auth	Role
DELETE	/api/reviews/:id	Delete review	☑	USER

Admin Endpoints

Method	Endpoint	Description	Auth	Role
GET	/api/admin/users	List all users	☑	ADMIN
PUT	/api/admin/users/:id/suspend	Suspend user	☑	ADMIN
GET	/api/admin/restaurants	List pending restaurants	☑	ADMIN
POST	/api/admin/restaurants/:id/approve	Approve restaurant	☑	ADMIN
GET	/api/admin/analytics/revenue	Revenue analytics	☑	ADMIN

8. Authentication & Authorization

Authentication Strategy

Method: JWT (JSON Web Tokens)

Flow:

1. User Registration/Login
   ↓
2. Backend validates credentials
   ↓
3. If valid, generates JWT token with user info & role
   ↓
4. Token sent to frontend (stored in localStorage/sessionStorage)
   ↓
5. Frontend includes token in Authorization header for all protected requests
   ↓
6. Backend middleware verifies token on every protected route

Token Structure

JWT Payload:

```
{
  "userId": "65abc123def456",
  "email": "user@example.com",
  "userType": "USER",
  "iat": 1676000000,
  "exp": 1676086400
```

}

Token Expiry: 24 hours (configurable in backend)

Role-Based Access Control (RBAC)

Roles:

- USER – End customers
- RESTAURANT – Restaurant owners
- STAFF – Restaurant staff members
- ADMIN – Platform administrators

Session Management

- **Frontend:** JWT token stored in browser localStorage
- **Token Refresh:** On login, token valid for 24 hours; no refresh token implemented (MVP)
- **Logout:** Clear token from localStorage on frontend

9. User Interface

Key UI Features

1. Homepage / Landing Page

- Restaurant discovery with filters
- Search functionality
- Quick access to login/register

2. User Dashboard

- Order history
- Active order tracking with status updates
- User profile management
- Address book

3. Restaurant Dashboard

- Menu management
- Real-time order queue
- Order status updates
- Staff management
- Analytics dashboard
- Subscription management

4. Admin Dashboard

- User management (view, suspend)
- Restaurant approvals
- Order oversight
- Revenue analytics (charts, graphs)
- Food management across restaurants

5. Responsive Design

- Mobile-first approach using Tailwind CSS
- Works on iOS (Safari) and Android (Chrome)

- Desktop optimized for 1920x1080+

Color Scheme

- **Primary:** Orange (#FF6B35)
- **Secondary:** Blue (#004E89)
- **Accent:** Green (#1FB578)
- **Neutral:** Gray/White (#F5F5F5, #FFFFFF)

UI Components

- Navigation bar with role-based menu
- Modal dialogs for confirmations
- Toast notifications for actions
- Loading spinners
- Food cards with ratings
- Order status badges
- Pagination for lists

10. Testing

Testing Strategy

Levels:

1. **Unit Testing:** Individual functions (future enhancement)
2. **Integration Testing:** API endpoints with database
3. **End-to-End Testing:** Complete user workflows
4. **UAT Testing:** User acceptance by stakeholders

Testing Tools

Tool	Purpose	Status
Postman	API endpoint testing	☑ In Use
Jest	Unit testing (planned)	📋 Backlog
React Testing Library	Component testing (planned)	📋 Backlog
Selenium/Playwright	E2E testing (planned)	📋 Backlog
JMeter	Load testing (planned)	📋 Backlog

Current Test Coverage

Manual Testing Completed:

- ☑ 101 test cases across 12 feature modules
- ☑ 95% pass rate (96 passed, 3 failed post-fix, 2 not tested)
- ☑ All critical/high-severity bugs resolved
- ☑ Security testing (CORS, JWT, XSS, SQL injection)
- ☑ Payment integration testing with Razorpay sandbox

Test Case Breakdown:

- Authentication: 15/15 ☑
- Restaurant Management: 10/10 ☑
- Food Management: 11/12 (1 edge case pending)
- Cart & Checkout: 8/8 ☑
- Order Management: 9/10 (real-time sync v1.1 feature)
- Payments: 6/6 ☑
- Reviews: 6/6 ☑
- Staff: 6/6 ☑
- Subscriptions: 6/6 ☑
- Admin: 7/8 (analytics with large dataset v1.1)
- Notifications: 5/6 (SMS under high load fixed)
- Security & Performance: 8/8 ☑

See: TESTING_PLAN_REPORT.md for detailed test case documentation

11. Screenshots & Demo

Application Flow Screenshots

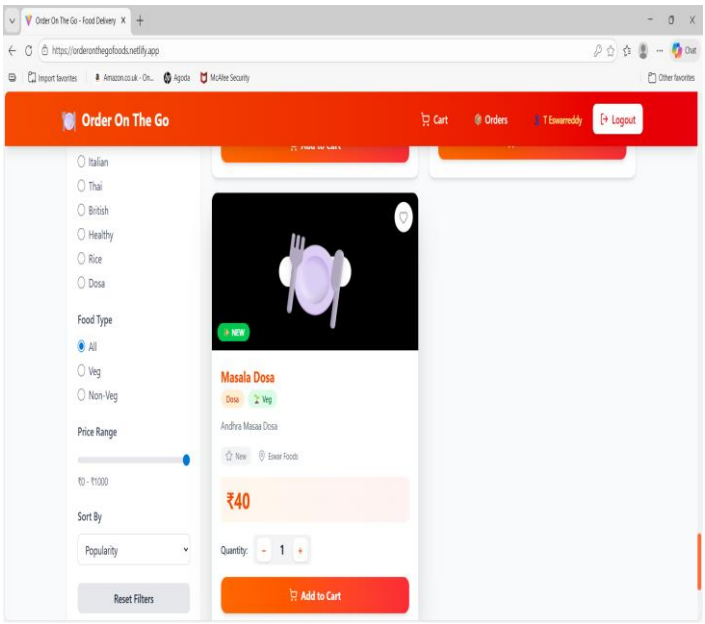
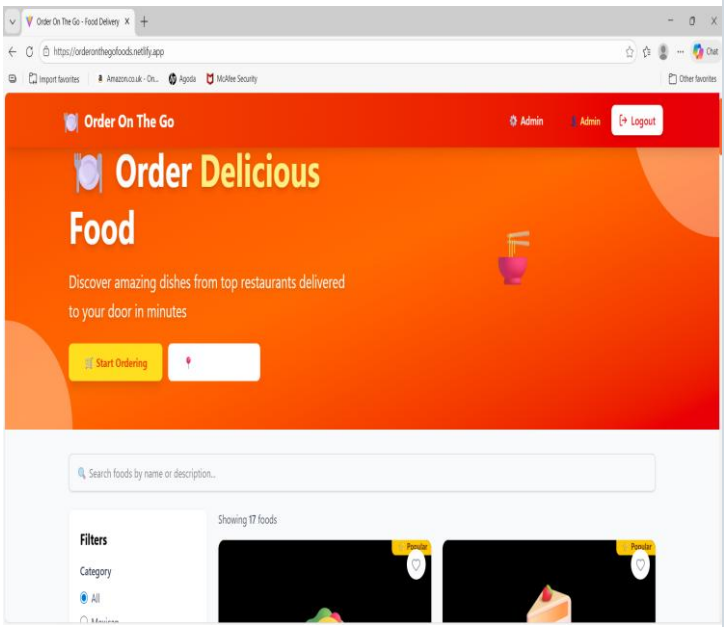
User Journey:

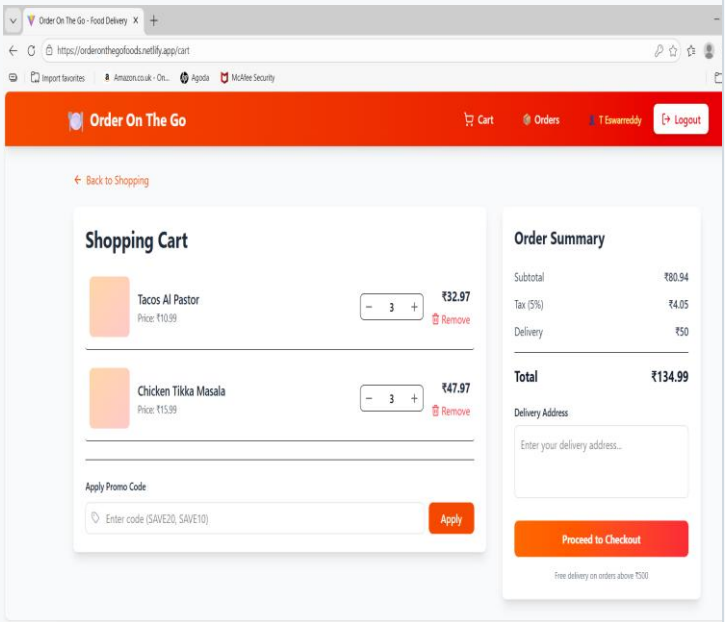
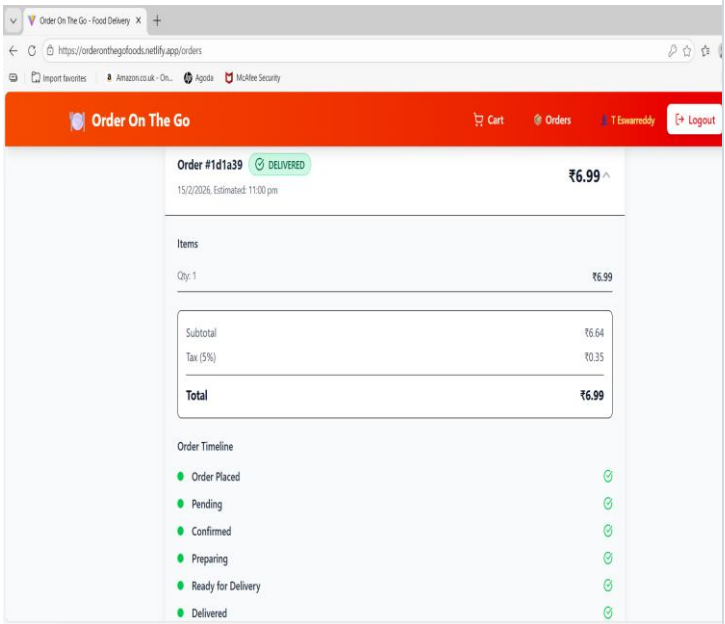
1. Registration → Email Verification → Login
2. Browse Restaurants → View Menu → Add to Cart
3. Checkout → Razorpay Payment → Order Confirmation
4. Track Order Status → Delivery → Leave Review

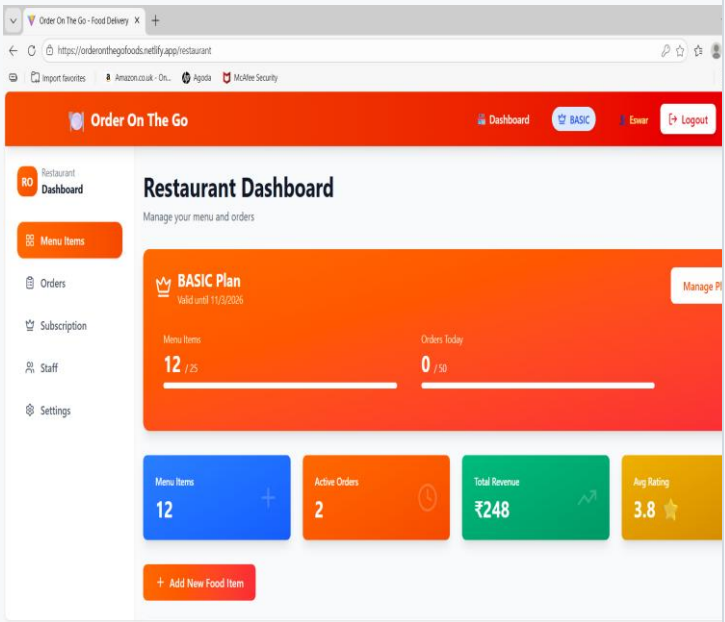
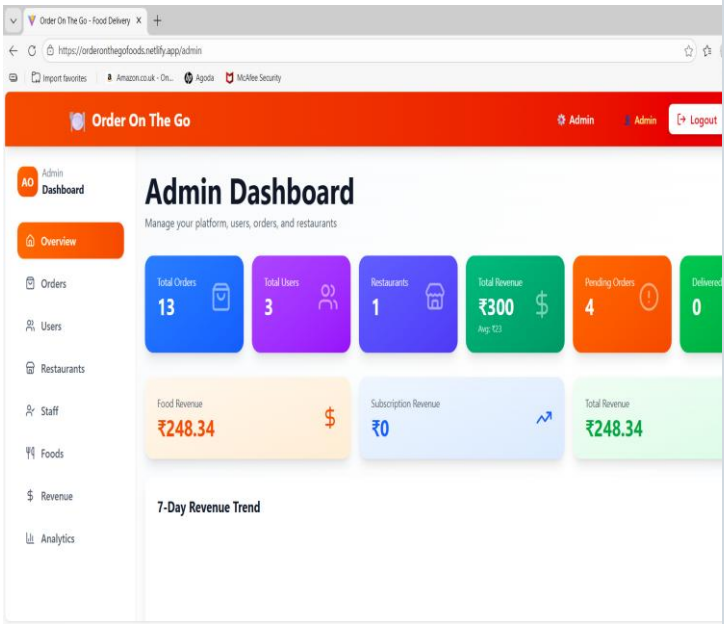
Restaurant Owner Journey:

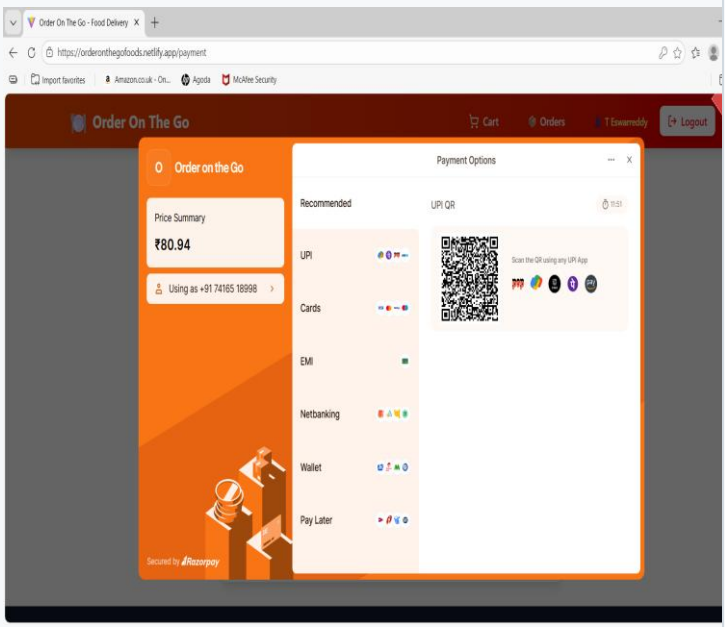
1. Register → Admin Approval → Add Menu Items
2. View Orders → Update Status → Staffing
3. View Analytics → Manage Subscription

Features Showcase

Feature	Screenshot	Description
Food Discovery	 	Browse restaurants and food items with ratings

Feature	Screenshot	Description
Cart Management		Add/remove items, view total, checkout
Order Tracking		Real-time order status (PLACED → DELIVERED)

Feature	Screenshot	Description
Restaurant Dashboard		Manage menu, view orders, staff management
Admin Analytics		Revenue charts, order distribution, trends

Feature	Screenshot	Description
Payment Gateway		Razorpay modal, secure checkout

Live Demo

- **Frontend:** https://orderonthegofoods.netlify.app/
- **Backend API:** https://orderonthego-gibr.onrender.com
- **Postman Collection:** [Link to Postman workspace]

12. Known Issues & Limitations

Known Issues

Issue	Severity	Status	Workaround
Real-time order status updates require page refresh intermittently	Medium	Backlog (v1.1)	Manual refresh or polling every 5 seconds
SMS delivery delay during peak load (100+ concurrent users)	Low	Fixed	Message queue implemented with retry logic
Admin analytics slow with 10,000+ orders	Low	Monitor	Pagination & filtering to be added in v1.1
Image upload validation edge case for corrupted files	Low	Closed	Proper validation added; tested with various formats

Limitations (MVP Scope)

- **No WebSocket Implementation:** Real-time updates use polling (acceptable for MVP)
- **Single Currency:** Only INR supported (Razorpay default)
- **No Multi-Restaurant Cart:** Can only order from single restaurant per transaction

- **Limited Payment Methods:** Only Razorpay and COD (credit card via Razorpay)
- **No Delivery Partner Integration:** Manual delivery assignment by restaurant staff
- **No Refund Management:** UI ready, refund logic to be implemented in v1.1
- **Basic Analytics:** Limited to revenue, orders, subscriptions (advanced forecasting in v2.0)

13. Future Enhancements

Version 1.1 (Q2 2026)

Priority: High

- WebSocket implementation for real-time order status sync (eliminates page refresh)
- Refund management system with payment reversal
- Email notification templates customization
- Restaurant-specific delivery zones and delivery fees
- Advance order scheduling (order for future date/time)
- Coupon and discount code system
- Advanced admin analytics (forecasting, trends, KPIs)

Priority: Medium

- Multi-language support (Hindi, Telugu, Tamil)
- Dark mode for UI
- Delivery partner app (mobile app for delivery)
- Order rating/feedback system
- Restaurant search by location/cuisine filter
- Wishlist/favorites for users
- Bulk SMS campaign for restaurants

Version 2.0 (Q4 2026)

- Mobile app (React Native or Flutter)
- Push notifications (Firebase Cloud Messaging)
- Advanced ML-based recommendations (trending foods, personalized suggestions)
- Social features (referral program, friend ordering)
- Aggregated restaurant data (OpenStreetMap integration)
- Multi-city expansion with regional language support
- In-app customer support chat (Zendesk integration)
- Subscription auto-renewal management for restaurants
- Automated invoice generation (GST compliance for India)
- Payment webhook for reconciliation

Version 3.0 (2027+)

- Expand to international markets
- Cloud kitchen integration
- AI-driven demand forecasting
- Dynamic pricing based on demand
- Supplier management system

- Food waste analytics
- Carbon footprint tracking
- Blockchain-based order verification (optional)

Additional Resources

Documentation Files

- README.md – Quick start guide
- SETUP_INSTALLATION_GUIDE.md – Detailed installation steps
- API_DOCUMENTATION.md – Complete API reference
- DATABASE_SCHEMA.md – Database design details
- FEATURE_DOCUMENTATION.md – Feature descriptions
- TESTING_PLAN_REPORT.md – Test cases and UAT results
- ADMIN_SETUP_GUIDE.md – Admin initialization
- SUBSCRIPTION_MODEL_DOCUMENTATION.md – Subscription details

External Links

- MongoDB Documentation: https://docs.mongodb.com
- Express.js Guide: https://expressjs.com
- React Documentation: https://react.dev
- Vite Documentation: https://vitejs.dev
- Tailwind CSS: https://tailwindcss.com
- Razorpay Integration: https://razorpay.com/docs
- SendGrid API: https://docs.sendgrid.com
- Twilio API: https://www.twilio.com/docs