Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MINI PROJECT

Programme	:	B. Tech Computer Science & Engineering
Course Code / Course Name	:	10211CS207 / Database Management Systems
Year / Semester	:	2025-26 / Summer
Faculty Name	:	Dr. Krishnaveni.N
Slot	:	S7L1
Title	:	**ONLINE FOOD DELIVERY SYSTEM**
Name of the students	:	

1) P.TARUN -VTU29624

2) SK. MASTHAN VALI - VTU29717

3) Sk. RASOOL -VTU29782

4) P. CHAITRA NAGA SRI -VTU29807

5) PASUPULETI TEJA -VTU29852

6) SK. USMAN -VTU29869

7) SRI HARI CHARAN C S -VTU29876

8) CH. BHANU PRAKASH - VTU29904

9) SAKE ASHOK -VTU29911

10) B. AKSHAYA CHANDANA -VTU29938

ONLINE FOOD DELIVERY SYSTEM

Problem statement:

Customers need a fast, reliable way to browse nearby restaurants, place orders, and track deliveries. Restaurants need an efficient order management channel. Delivery partners need clear pickup/delivery instructions and optimized routing. The goal is to build an integrated system that connects customers, restaurants, and delivery partners while ensuring order accuracy, real-time tracking, payment handling, and scalability.

Abstract:

This project presents an Online Food Delivery System that connects customers, restaurants, and delivery partners through a responsive web and mobile application. The system allows customers to discover restaurants, view menus, place orders, choose delivery or pickup, and pay via multiple methods. Restaurants receive real-time orders, update menu items and availability, and manage order status through an intuitive dashboard. Delivery partners accept assignments, navigate optimized routes, and update delivery status with live location reporting. The backend handles order orchestration, inventory availability, pricing, promotional offers, payment processing, and notifications. Key functional components include user authentication, restaurant and menu management, cart and checkout flow, order management, real-time tracking, ratings & reviews, and admin analytics. The architecture follows a microservice-compatible design with RESTful APIs, a relational database for core transactional data, and a push/real-time layer (WebSockets or Firebase) for status updates and location tracking. Security considerations (HTTPS, JWT for auth, secure payment integration) and non-functional requirements (scalability, availability, latency targets) are addressed. The methodology includes requirement analysis, UI/UX design, database modelling, API design, implementation (Node.js/Express or Django REST), frontend development (React / React Native), testing (unit, integration, user acceptance), and deployment (Docker, Kubernetes, cloud). Expected contributions include a reliable prototype demonstrating end-to-end order flow, improved delivery visibility via live tracking, and metrics-driven dashboards for restaurants and admins. The system can be extended with ML-powered recommendations, demand forecasting, and dynamic delivery assignment to further optimize user experience and operational efficiency.

Objectives:

1. Provide a seamless ordering experience for customers (search, order, pay, track).
2. Enable restaurants to manage menu, accept/reject orders, and view analytics.
3. Provide delivery partners with assignment, navigation, and status update tools.
4. Ensure secure payment processing and data privacy.

5. Build a scalable, maintainable architecture suitable for future extensions (ML recommendations, surge pricing).

Actors:

- Customer (end user)
- Restaurant Admin (manager/staff)
- Delivery Partner (rider)
- System Administraton
- Payment Gateway (external)

Primary use cases (brief):

User registration / login (email/phone/social).

1. Browse restaurants & search by location/cuisine.
2. View menu, customize items, add to cart.
3. Checkout: choose address, delivery time, payment method, apply offers.
4. Place Order — receive order confirmation.
5. Restaurant receives order and updates status (Accepted → Preparing → Ready).
6. Delivery partner assigned, picks up order, and updates status (Picked up → In transit → Delivered).
7. Real-time tracking: customer sees rider location and ETA.
8. Ratings & reviews for food and delivery.
9. Admin: view dashboards, manage users/restaurants, handle disputes.

Typical success scenario (Place order & deliver):

1. Customer searches and selects restaurant.
2. Adds items to cart, checks out, pays.
3. System creates order, notifies restaurant.
4. Restaurant accepts and starts preparing.
5. System assigns delivery partner (auto or manual).
6. Delivery partner picks up and delivers.
7. Customer confirms receipt and leaves feedback.

Functional Requirements (high-level):

- User authentication and profile management.
- Restaurant onboarding and menu management.
- Cart, order creation, order history.

- Real-time order & delivery tracking (WebSocket/Firebase).
- Payment integration (Stripe/Paytm/Razorpay).
- Push notifications & SMS alerts.
- Rating and review system.
- Admin dashboards and report

.

Non Functional Requirements:

- Availability: 99.5% (target).
- Scalability: horizontal scaling for order API and real-time services.
- Latency: API response < 200–500 ms for core endpoints.
- Security: HTTPS, encrypted sensitive data, JWT with refresh tokens.
- Data retention & backups daily.

Data model (entities):

- Users (customers, roles)
- Restaurants (profile, location, opening hours)
- MenuItems (name, price, options, availability)
- Orders (items, total, status, timestamps)
- OrderItems (quantity, customizations)
- DeliveryPartners (status, current_location)
- Payments (transaction id, status)
- Ratings & Reviews

Architecture (high-level):

- Client: React (web) + React Native (mobile).
- Backend: RESTful API (Node.js/Express or Django REST).
- Real-time: WebSocket server or Firebase Realtime/Firestore for live updates.
- DB: PostgreSQL (relational) + Redis (caching, queues).
- Task queue: RabbitMQ / Celery or Bull (Node).
- External: Payment gateway, Maps API (Google/Mapbox) for routing.
- Deployment: Docker containers, deploy on cloud (AWS/GCP/Azure), optional Kubernetes.

Sequence / Flow (EX:order placement):

Customer UI → POST /orders → Orders Service writes to DB → Notify Restaurant (push/WebSocket) → Restaurant updates status → System assigns delivery partner (match service) → Delivery partner receives job → Live location updates published → Customer polls/subscribes to updates.

Technology stack (suggested):

- Frontend: React, React Native, Redux.
- Backend: Node.js + Express or Python + Django REST Framework.
- DB: PostgreSQL.
- Realtime: Socket.IO or Firebase.
- Auth: JWT, OAuth for social login.
- Payment: Stripe / Razorpay integration.
- DevOps: Docker, GitHub Actions, AWS/GCP.

Testing & Evaluation:

- Unit tests for services & components.
- Integration tests for API endpoints.
- End-to-end tests (Cypress / Appium).
- Load testing (k6 / JMeter).
- UAT with sample users and restaurants.

Implementation plan (milestones):

1. Requirements & wireframes (1 week).
2. Database & API design (1 week).
3. Basic auth, restaurant & menu CRUD (1 week).
4. Cart, checkout, orders (1 week).
5. Restaurant dashboard & order flow (1 week).
6. Delivery partner app & assignment logic (1 week).
7. Real-time tracking, payments, notifications (1 week).

8. Testing, bugfix, deploy (1–2 weeks).

Extensions & future work:

- ML-based recommendations and personalization.
- Dynamic delivery-partner allocation using optimization.
- Predictive ETAs using historical data.
- Loyalty & subscription models.

Conclusion :

The proposed Online Food Delivery System provides a full-stack, production-minded solution connecting customers, restaurants, and delivery partners with real-time tracking, secure payments, and analytics. The design emphasizes modularity and scalability so the system can be extended with ML recommendations and optimization algorithms in future work.
