MERN Food Delivery App

Name : BiteNow

Created By : Ahmad Tariq

Technologies Used

3. Technologies to Use

Frontend

- **React.js** → UI library.
- **Redux Toolkit / Context API** → State management.
- **Axios/Fetch** → API calls.
- **React Router** → Routing.
- **CSS / SCSS / Material UI / Bootstrap** → Styling (no Tailwind).
- **Socket.IO Client** → Real-time chat + tracking.
- **Leaflet.js / Google Maps API** → Rider location tracking.

Backend

- **Node.js + Express.js** → REST API.
- **MongoDB + Mongoose** → Database.
- **JWT (jsonwebtoken)** → Authentication.
- **bcrypt.js** → Password hashing.
- **Socket.IO** → Real-time chat + rider tracking.
- **Stripe SDK / PayPal SDK** → Payment integration.
- **Multer / Cloudinary** → Image upload (restaurant logos, food pictures).

Extra Tools

- **Nodemon** → Auto-restart server during dev.
- **dotenv** → Environment variables.
- **Postman/Thunder Client** → API testing.
- **Git + GitHub** → Version control

Details

1) One-line vision

A clean, responsive MERN food-delivery web app (no Tailwind) where users register as **customer**, **restaurant owner**, or **rider** (plus system **admins** and one hard-coded **superadmin**). Restaurant owners create restaurant profiles and menus; customers order from restaurants; orders are handed to riders; customers can track rider location in real time and chat with rider/restaurant. Admins manage users and content. Payments are integrated.

2) Roles & permissions (high level)

- **Superadmin** (single account seeded in code/config): full system control — create/delete admins, manage all users and restaurants, view metrics.
- **Admin**: manage users (block/delete), view orders, moderate restaurants/menus.
- **Restaurant owner**: create/ edit their restaurant profile, create categories & menu items, accept/reject orders, mark orders ready for pickup, message customers and riders.
- **Rider**: accept delivery assignments, update live location, report delivery status.
- **Customer**: browse restaurants, place orders, pay, track rider, chat with restaurant/rider, rate orders.

3) Main user stories (concise)

- As a **new user**, I register and choose my user type (customer, restaurant owner, rider).
- As a **restaurant owner**, I create a restaurant profile, add categories & menu items, and manage incoming orders.
- As a **customer**, I browse restaurants, add items to cart, checkout with payment, and track my order/rider location.
- As a **rider**, I receive assigned deliveries, accept/decline, and broadcast location while delivering.
- As a **customer/restaurant owner/rider**, I can chat in real time about a specific order.
- As **superadmin/admin**, I manage users, delete accounts, and moderate restaurants.

4) Functional feature list (grouped)

Core flows

- Registration & login with role selection.
- Role-based dashboards (different UI & actions per role).
- Restaurant profile creation (name, address, phone, description, hours, images).
- Menu: owner-created **categories** (e.g., Burgers, Drinks) → add items to categories.

- Customer ordering: cart, checkout, create order for a specific restaurant.
- Payments: integrate Stripe (or similar) for card payments + support COD.
- Order lifecycle: placed → accepted (restaurant) → ready / picked (rider) → delivered / cancelled.
- Rider assignment flow: admin/restaurant auto-assign or choose from available riders; rider accepts.
- Real-time features: rider live location feed per order + order room chat (customer, owner, rider).
- Notifications: push/real-time alerts for order status changes (web sockets).
- Admin panel: user/restaurant/order management, metrics.

Support features

- Rate & review orders/restaurants.
- Order history for customers and riders.
- Basic search & filters for restaurants (cuisine, distance, rating).
- Simple analytics for owners (daily orders, top items).

5) Data model overview (entities & key fields, non-exhaustive)

- **User**: id, name, email, passwordHash, role, phone, createdAt, profilePhoto, restaurantId (if owner), isBlocked
- **Restaurant**: id, ownerId, name, address, phone, description, images, categories[], isActive
- **Category**: id, restaurantId, name
- **MenuItem**: id, restaurantId, categoryId, name, description, price, image, available
- **Order**: id, customerId, restaurantId, items[{menuItemId, qty, price}], total, paymentStatus, status, riderId, createdAt, deliveredAt
- **RiderLocation**: orderId, riderId, lat, lng, timestamp (real-time—persist minimal history)
- **ChatMessage**: orderId, fromUserId, toUserIds, text, timestamp
- **AdminLogs**: action, adminId, targetId, timestamp

6) API / surface area (summary, not code)

- **Auth**: register (with role), login, refresh token, logout.
- **Users**: get profile, update profile, admin endpoints: list users, block/delete user.
- **Restaurants**: create/edit/get/list, owner-only endpoints: add category, add item, update item.

- **Orders**: place order (customer), list orders (by role), update order status (owner/rider/admin), assign rider.
- **Payments**: create payment intent, webhook handler to mark order paid.
- **Realtime**: socket namespaces/rooms per order for location & chat.
- **Chat**: fetch message history for order, send message (via sockets and persisted via API).
- **Admin**: management endpoints (create admin, delete user, view reports).

7) UI pages & components (high level)

- Public:
    - Landing / restaurant listing / search
    - Restaurant page (menu, categories)
- Auth:
    - Register (select role at signup)
    - Login / Forgot password
- Customer:
    - Home / search / restaurant page
    - Cart / Checkout / Payment
    - Order tracking page (map + rider coordinates + chat)
    - Profile / order history
- Restaurant owner dashboard:
    - Restaurant profile editor
    - Menu manager (categories + items)
    - Orders list (incoming) with accept/reject and mark ready
    - Chat / messages
    - Simple analytics
- Rider dashboard:
    - Available deliveries list
    - Active delivery view (start/pause/update location/send status)
    - Earnings / history
- Admin dashboard:

    - User management (filter by role), restaurant moderation, order oversight, logs

8) Real-time & mapping design

- Use **Socket.IO** for real-time communication (location updates and chat).
- Each order has a socket room (e.g., order_<id>). Customer/owner/rider join that room.
- Rider emits riderLocation events periodically (e.g., every 3–5 seconds) while delivering.
- Customer order tracking page subscribes to room and renders the rider's location on a map.
- For maps: use **Leaflet + OpenStreetMap** (no billing) — it's simple and free.

9) Payments & order security

- Use **Stripe Payment Intents** for card payments (PCI considerations handled by Stripe).
- Flow: create PaymentIntent server-side, return client secret → frontend completes payment → webhook marks order as paid.
- Support Cash-on-Delivery (COD) as an alternate paymentStatus.
- Validate payment server-side before setting order status to paid.
- Idempotency keys for payment webhooks & order creation to prevent double charges.

10) Authentication, authorization & security

- JWT for auth (access token + optional refresh tokens).
- Protect all API endpoints; check role-based access at controller/middleware level.
- Input validation & rate-limiting on critical endpoints (auth, payments).
- Hard-code or seed one **superadmin** account from configuration/env; only superadmin can create other admin accounts.
- Store passwords hashed (bcrypt) and secrets in environment variables.

11) Acceptance criteria & basic tests (per major feature)

- **Registration**: user can register and login; selected role is stored; invalid email rejected.
- **Restaurant creation**: owner can create and edit only their restaurant; restaurant appears in public listing.
- **Menu management**: owner can add category and items; items show up on restaurant page with correct prices.

- **Ordering & payment**: customer can checkout, be charged (Stripe success), and order recorded with paid status.
- **Order lifecycle**: owner accepts order; rider assigned; rider location updates are visible to customer in near real time.
- **Chat**: sending a message appears for all participants of order room and persists to DB.
- **Admin functions**: superadmin can create/delete admin; admin can block/delete users and restaurants.
- **Security**: blocked users cannot log in (or access protected endpoints).

12) MVP scope (recommended — keep it small to finish faster)

MVP features

1. Registration/login with role selection
2. Restaurant owner: create restaurant + manage menu categories/items
3. Customer: browse restaurants, add to cart, checkout (Stripe + COD)
4. Order lifecycle: place → accept → mark ready → assign rider → mark delivered
5. Rider: accept/deliver and emit location; customer can track on a simple map
6. Simple order chat (order-level messages)
7. Superadmin seeded account + admin user management UI
8. Basic CSS styles (no Tailwind) for clean look

Defer to v1+

- Advanced search & recommendation
- Ratings & complex analytics
- Push notifications / mobile apps
- Multi-restaurant promotions & coupons

13) Prioritized milestone plan (4 sprints suggested)

- **Sprint 1 (Auth + Basic models + Restaurant management)**
  Registration/login/role selection, create restaurant + menu categories, basic UI for owner & restaurant listing.
- **Sprint 2 (Customer ordering & payments)**
  Cart, checkout, Stripe integration, order model & status flow.
- **Sprint 3 (Rider flow + real-time tracking)**
  Rider assignment, rider UI, Socket.IO location updates, customer tracking UI with Leaflet.

- **Sprint 4 (Chat + Admin + polish)**
  Order chat, superadmin/admin functions, CSS polish, basic testing & deployment.

14) UX / visual suggestions (clean look, no Tailwind)

- Use a neutral palette: soft off-white background, dark text, 1 accent color (e.g., blue/green) for CTAs.
- Cards for restaurants and menu items with soft shadows and rounded corners.
- Responsive grid: 1 column mobile, 2–3 columns on desktop.
- Clear, prominent role selection during signup (radio or select).
- Order tracking page: top shows order status timeline, map below, chat docked to the side or bottom on mobile.
- Minimal animations (fade/slide) for status changes for perceived responsiveness.

15) Non-functional & operational considerations

- **Scalability**: Socket.IO scaling needs sticky sessions or a redis adapter for multiple instances.
- **Logging & monitoring**: store admin logs & use basic server logs/health endpoints.
- **Backups**: backup MongoDB regularly for orders and chats.
- **Deployment**: separate frontend static hosting (Netlify/Vercel) or serve from Node; backend on Heroku/Render/DigitalOcean; use HTTPS.

16) Open questions (I made reasonable defaults)

I assumed:

- Stripe as payment provider.
- Leaflet/OpenStreetMap for maps.
- JWT for tokens and Socket.IO for real-time.

If you want different choices (e.g., Razorpay or PayPal, Google Maps), tell me and I'll adjust the plan.

17) Final checklist to hand to devs (copy/paste)

- Seed superadmin from .env (email + password).
- Enforce role on registration and store it on the user record.
- Only restaurant.owner creates/edits their restaurant and menus.
- Orders are tied to restaurant + items with computed totals server-side.
- Payment: server creates PaymentIntent and verifies webhook before marking order paid.
- Socket rooms order_<id> for location & chat; throttle rider location events (e.g., max 1 per 3s).

- Admin routes protected by role('superadmin','admin').
- Basic CSS: utility classes and component-level CSS (no Tailwind).