**Designation:** Senior Backend Engineer / Database Architect

**Project:** Mealora (Catering Service Progressive Web Application)

**Project Objectives:** The primary objective is the design and implementation of a resilient, scalable backend Application Programming Interface (API) for "Mealora," a contemporary catering Progressive Web Application (PWA). As the frontend architecture has been finalized and anticipates a RESTful JSON API, the backend system must be engineered to adhere to stringent reliability constraints. Specifically, the architecture must facilitate advanced scheduling, precise inventory management, and the processing of high-volume bulk orders. The proposed solution must be architected to accommodate future scalability, adhering to clean code methodologies and implementing comprehensive error-handling mechanisms.

**Technical Specifications:**

1. **Runtime Environment:** Node.js (Latest Long-Term Support version) – Selected for its non-blocking I/O capabilities, ensuring optimal performance during concurrent order processing.
2. **Framework:** Express.js – Utilized for its modular structure, facilitating robust routing, error handling, and request validation middleware.
3. **Database & Authentication:** Supabase (PostgreSQL) – Chosen to leverage relational data integrity, Row Level Security (RLS), and integrated JSON Web Token (JWT) authentication.
4. **Security & Stability Measures:**
    - Implementation of `helmet` for HTTP header security reinforcement.
    - Configuration of `cors` policies strictly aligned with the PWA's domain.
    - Utilization of `express-validator` or `joi` for rigorous input validation.
    - Deployment of `express-rate-limit` to mitigate potential API abuse.

**Functional and Non-Functional Requirements:**

1. **Database Schema Architecture (Supabase / PostgreSQL):**
    - **User Profiles:** Extension of the default Supabase authentication schema to incorporate a public `profiles` table. This table shall house catering-specific data points, including `company_name`, `contact_phone`, and `billing_address`.
    - **Menu Management:** A dynamic structure capable of supporting daily menu rotations is required. Essential fields include:
        - `available_date` (Indexed for optimized temporal queries).
        - `stock_quantity` (Integer type to enforce inventory limits).
        - `dietary_tags` (Array of Strings; e.g., ['Vegan', 'Gluten-Free', 'Halal']).
        - `nutritional_info` (JSONB format for flexible data storage, such as caloric content and macronutrients).
        - `category` (Enumerated type: 'Breakfast', 'Lunch', 'Event Platter').
    - **Order Lifecycle:** The order system must function as a deterministic state machine.
        - Status Enumeration: `pending_payment`, `confirmed`, `preparing`, `out_for_delivery`, `delivered`, `cancelled`.

- Temporal tracking must include `delivery_window_start` and `delivery_window_end`.

- **Order Items:** A relational entity linking orders to menus, explicitly recording the `unit_price` at the moment of transaction to maintain historical financial accuracy, independent of future price fluctuations.

2. **API Endpoints & Architectural Design:**
   - **Global Middleware:** Implementation of authentication middleware is mandatory to decode Supabase JWTs and inject the `user` context into the request object. Furthermore, centralized error-handling middleware must be established to ensure standardized JSON error responses.
   - **Menu Retrieval and Management:**
     - `GET /api/v1/menu`: Shall accept query parameters for `date`, `dietary_tags`, and `category`. The endpoint must handle empty result sets gracefully (e.g., during holidays).
     - *Administrative Access:* `POST /api/v1/menu` for the publication of daily offerings.
   - **Order Processing:**
     - `POST /api/v1/orders`: Requires comprehensive validation logic. Atomic checks of stock levels are imperative to prevent inventory discrepancies during periods of high traffic.
     - `PATCH /api/v1/orders/:orderId/cancel`: Facilitates user-initiated cancellations, provided they fall within the authorized cancellation window.
   - **User History & Tracking:**
     - `GET /api/v1/orders`: Returns a paginated history of orders for the authenticated user.
     - `GET /api/v1/orders/:orderId`: Provides a granular view of a specific order, including line items and current status.
3. **PWA Optimization & Performance:**
   - **Caching Strategy:** `Cache-Control` headers must be implemented for GET requests (e.g., menu data may be cached for one hour, whereas user profile data requires strict non-caching policies).
   - **Payload Optimization:** All API responses must be compressed (utilizing gzip or brotli) and formatted as strictly typed JSON to minimize latency on mobile networks.
   - **Offline Synchronization:** To ensure data consistency, the API must support "Idempotency Keys" in POST headers, preventing duplicate transaction processing in the event of network interruptions and subsequent automatic retries.
4. **Advanced Business Logic Implementation:**
   - **The 24-Hour Lead Time Protocol:** Given the nature of catering preparation, middleware must automatically reject any order wherein the `delivery_date` falls within 24 hours of the current timestamp ( `now()` ).
   - **Dynamic Pricing Engine:** Total costs must be calculated server-side. The logic shall include provisions for bulk volume discounts (e.g., a 10% reduction for orders exceeding $500).
   - **Inventory Control:** Upon order creation, the `stock_quantity` within the `menus` table must be decremented. Conversely, order cancellations must trigger a stock replenishment. Database transactions are required to guarantee atomicity and data integrity.

**Deliverables:**

1. `server.js` **:** A comprehensive, modular Express server file encompassing connection logic, middleware configuration, and route definitions.
2. **SQL Configuration Script:** A detailed SQL script facilitating the creation of tables, the establishment of Foreign Key constraints, the definition of Performance Indexes, and the implementation of Row Level Security (RLS) policies.