Project Proposal: Thook - Hyperlocal Instant Commerce Platform

1. Executive Summary

Thook is a scalable, hyperlocal instant commerce platform engineered for rapid delivery and real-time inventory synchronization. The system utilizes a modern, decoupled three-tier architecture to minimize operational costs while maximizing performance. By leveraging serverless infrastructure and asynchronous API communication, Thook ensures high availability, strict ACID compliance for transactions, and a frictionless user experience.

2. System Architecture & Technology Stack

The platform separates client-side rendering from backend business logic and database management, ensuring secure and scalable operations.

- **Customer Application (Mobile):** Flutter (iOS/Android). Compiles to native ARM machine code for high-performance rendering. State management is handled via Riverpod for predictable, unidirectional data flow.
- **Admin Dashboard (Web):** ReactJS. A component-driven single-page application (SPA) optimized for complex data grid rendering and real-time state updates.
- **Core API Backend:** Python (FastAPI). A high-performance, asynchronous web framework. It acts as the secure middleware, processing RESTful requests, executing business logic, and handling database transactions via SQLAlchemy and Asyncpg.
- **Database:** Neon (Serverless PostgreSQL). Provides connection pooling and copy-on-write branching. Compute scales to zero during idle periods, optimizing resource utilization.
- **Authentication Engine:** Firebase Auth. Manages identity verification and issues stateless JSON Web Tokens (JWTs) for secure API endpoint authorization.

3. Core Technical Modules

Module A: Mobile Client Operations

- **Stateless Authentication:** Implements OTP-based verification, securely storing session tokens locally (via Flutter Secure Storage) to authorize subsequent API calls.
- **Data Caching & Synchronization:** Implements local caching of the product catalog to reduce redundant network requests and database read operations.
- **Transactional Cart State:** Maintains isolated local cart state, calculating totals and delivery parameters before constructing the final order payload for backend validation.

Module B: Administrative Web Console

- **Role-Based Access Control (RBAC):** Restricts dashboard access strictly to verified administrative JWTs.

- **Inventory Mutation:** Provides full CRUD (Create, Read, Update, Delete) interfaces for the product catalog, triggering immediate database updates.
- **Order Lifecycle Management:** Visualizes the orders table in real-time, allowing administrators to update execution states (Pending $\rightarrow$ Accepted $\rightarrow$ Dispatched $\rightarrow$ Fulfilled).

Module C: API & Backend Infrastructure

- **JWT Verification Middleware:** Intercepts all incoming requests to validate Firebase signatures, preventing unauthorized database access.
- **Concurrency Handling:** Utilizes PostgreSQL row-level locking (e.g., SELECT ... FOR UPDATE) during the checkout sequence to prevent race conditions and inventory double-booking when processing simultaneous orders.

4. Relational Database Schema (DDL Blueprint)

The system relies on a strictly typed relational schema to enforce data integrity:

- users
  - id (UUID, Primary Key)
  - firebase_uid (String, Unique, Indexed)
  - phone_number (String, Unique)
  - created_at (Timestamp)
- products
  - id (UUID, Primary Key)
  - name (String)
  - category (String, Indexed)
  - price (Numeric)
  - stock_count (Integer)
  - image_url (String)
  - is_active (Boolean, Default: True)
- orders
  - id (UUID, Primary Key)
  - user_id (UUID, Foreign Key $\rightarrow$ users.id)
  - total_amount (Numeric)
  - status (Enum: PENDING, ACCEPTED, DELIVERED, FAILED)
  - created_at (Timestamp, Indexed)
- order_items
  - id (UUID, Primary Key)
  - order_id (UUID, Foreign Key $\rightarrow$ orders.id)
  - product_id (UUID, Foreign Key $\rightarrow$ products.id)
  - quantity (Integer)
  - price_at_purchase (Numeric)