Below is a single comprehensive prompt you can paste into **Cursor** (Create/Chat/Agent). It instructs Cursor to scaffold a production-ready MVP for a hyper-local home-food marketplace with rider delivery. Adjust placeholders in ALL_CAPS before running.

SYSTEM ROLE / HIGH-LEVEL GOAL

You are a senior full-stack engineer. Build a **minimal, production-ready MVP** for a hyper-local **HomeFood + Rider** Micro SaaS: - **Frontend:** Flutter (Android-first), Riverpod for state, GoRouter for navigation. - **Backend:** FastAPI + PostgreSQL via Supabase (Auth, DB, Storage), SQLAlchemy 2.x, Pydantic v2, Alembic. - **Notifications:** Firebase Cloud Messaging (FCM) plumbing (stubs for now). - **Maps:** Use OpenStreetMap/OSM or Google Maps wrapper; abstract behind a service. - **Payments (MVP):** Cash on delivery / wallet placeholder; no gateway integration in v1. - **I18N:** English + Urdu ready (string ARB + RTL support on Flutter). Default Urdu for PK locale. - **Target:** Small town networks using WhatsApp today. Keep flows lean, offline-tolerant.

Deliver: 1) A working monorepo with frontend `app/` and backend `server/` + infra. 2) SQL schema + Alembic migrations. 3) Seed scripts and minimal fixtures. 4) Sample data + mock images. 5) README with run commands. 6) Basic unit tests where valuable (domain logic, assignment).

Follow the **MVP scope** and **acceptance criteria** precisely.

FUNCTIONAL SCOPE (MVP)

Roles: Buyer, Seller (home chef), Rider, Admin.

Buyer - Browse today’s menu (nearby sellers; filter by category/time). - Item detail → add to cart → checkout. - Choose Pickup or Delivery (via rider). - Place order (Cash / Local wallet placeholder). - Track status: Pending → Accepted → Rider Assigned → Picked Up → Delivered.

Seller - Manage menu (CRUD, availability toggles, photos). - Incoming orders: Accept/Reject. - If delivery required: Auto-assign closest available rider (with fallback to manual pick from list). - Mark “Handed Over.”

Rider - Toggle availability (Available/Busy) with last seen and approximate location. - See assigned tasks, open task detail, call buyer/seller, start navigation link. - Mark Picked-Up / Delivered.

Admin (minimal) - List users, orders, deliveries; update statuses for dispute resolution.

DATA MODEL (Postgres on Supabase)

Create tables with appropriate FKs, indexes, RLS (later toggle), soft-delete columns optional.

```
users (
  id uuid pk default gen_random_uuid(),
  role text check (role in ('buyer','seller','rider','admin')) not null,
  phone text,
  name text not null,
  avatar_url text,
  rating_avg numeric(3,2) default 0,
  locale text default 'en',
  created_at timestamptz default now()
);

addresses (
  id uuid pk default gen_random_uuid(),
  user_id uuid references users(id) on delete cascade,
  label text,
  line1 text,
  area text,
  city text,
  lat double precision,
  lng double precision,
  is_default boolean default false
);

menu_items (
  id uuid pk default gen_random_uuid(),
  seller_id uuid references users(id) on delete cascade,
  name text not null,
  photo_url text,
  category text,
  price_pk integer not null,
  portion text,
  is_available boolean default true,
  created_at timestamptz default now()
);

orders (
  id uuid pk default gen_random_uuid(),
  buyer_id uuid references users(id),
  seller_id uuid references users(id),
  address_id uuid references addresses(id),
  status text check (status in (

'pending','accepted','rider_requested','rider_assigned','picked_up','delivered','ready_for_pic
  )) default 'pending',
  delivery_required boolean default false,
  payment_method text check (payment_method in ('cash','wallet')) default
'cash',
  total_amount integer not null,
  created_at timestamptz default now()
);
```

```
order_items (
  id uuid pk default gen_random_uuid(),
  order_id uuid references orders(id) on delete cascade,
  menu_item_id uuid references menu_items(id),
  qty integer not null,
  unit_price integer not null
);

rider_presence (
  rider_id uuid primary key references users(id) on delete cascade,
  is_available boolean default false,
  last_seen_at timestamptz default now(),
  lat double precision,
  lng double precision
);

deliveries (
  id uuid pk default gen_random_uuid(),
  order_id uuid references orders(id) on delete cascade,
  rider_id uuid references users(id),
  status text check (status in (

'requested','assigned','en_route_pickup','picked_up','en_route_drop','delivered','confirmed'
  )) default 'requested',
  pickup_lat double precision,
  pickup_lng double precision,
  drop_lat double precision,
  drop_lng double precision,
  distance_km numeric(6,2),
  fee_pk integer default 0,
  created_at timestamptz default now()
);

payouts (
  id uuid pk default gen_random_uuid(),
  user_id uuid references users(id),
  role text,
  amount_pk integer not null,
  method text,
  external_ref text,
  status text default 'pending',
  created_at timestamptz default now()
);

create index idx_users_role on users(role);
create index idx_menu_items_seller on menu_items(seller_id, is_available);
create index idx_orders_seller_status on orders(seller_id, status);
create index idx_orders_buyer_status on orders(buyer_id, status);
create index idx_deliveries_rider_status on deliveries(rider_id, status);
```

```
create index idx_rider_presence on rider_presence(is_available,
last_seen_at);
```

Add initial **seed** users (2 sellers, 1 rider, 2 buyers), 6 menu items, and 3 orders across states.

BACKEND (FastAPI)

Structure (inside `server/`):

```
server/
  app/
    main.py
    core/config.py
    core/security.py
    db/session.py
    db/base.py
    models/*.py
    schemas/*.py
    api/
      deps.py
      v1/
        routes_users.py
        routes_menu.py
        routes_orders.py
        routes_riders.py
        routes_deliveries.py
    services/
      assignment.py
      notifications.py
      maps.py (abstraction; OSM/Google stub)
    tests/
    alembic/
  pyproject.toml
  alembic.ini
  README.md
```

Key Endpoints (v1) - `POST /auth/signup` (Supabase handles auth client-side; server trusts JWT via middleware; include stub if needed) - **Menu**: `GET /menu?seller_id=...`, `POST /menu`, `PATCH /menu/{id}` - **Orders (Buyer)**: `POST /orders` (create), `GET /orders/{id}`, `GET /orders?buyer_id=...` - **Orders (Seller)**: `GET /seller/orders?status=...`, `POST /seller/orders/{id}/accept`, `POST /seller/orders/{id}/reject` - **Delivery**: `POST /orders/{id}/request_rider` (auto), `POST /orders/{id}/assign_rider?rider_id=...` (manual) - **Rider**: `POST /rider/presence` (toggle + heartbeat), `GET /rider/tasks`, `POST /deliveries/{id}/status` → (`en_route_pickup|picked_up|en_route_drop|delivered`)

Assignment Service (services/assignment.py) - Input: order_id (with seller location ↔ pickup point, buyer drop address coords). - Query `rider_presence where is_available=true`, sort by distance then last_seen_at. - Atomic assign: DB transaction inserts into `deliveries` with `status='assigned'`, flips order→`rider_assigned`, and sets `rider_presence.is_available=false` (until delivery complete). - Retry: if none available, set order→`rider_requested` and return 202.

Notifications stub: write functions to enqueue FCM payloads (no keys committed). Log to console in dev.

Maps stub: build `maps.distance_km(a_lat,a_lng,b_lat,b_lng)` using haversine; allow external provider later.

Security: Validate Supabase JWTs via middleware. Provide `get_current_user()` dependency and `role_required([...])` decorator.

Migrations: Alembic autogenerate baseline from SQLAlchemy models that mirror schema above.

FRONTEND (Flutter)

Project: `app/` with Flutter 3.22+.

Packages - `flutter_riverpod`, `go_router`, `dio`, `freezed`, `json_serializable`, `intl`, `shared_preferences`, `firebase_messaging`, `geolocator`, `url_launcher`.

Folders

```
app/lib/
  main.dart
  app_router.dart
  core/
    env.dart
    theme.dart
    i18n/
      l10n.dart (English/Urdu ARB files)
  features/
    auth/
    buyer/
      home_screen.dart
      item_detail_screen.dart
      cart_screen.dart
      checkout_screen.dart
      order_status_screen.dart
    seller/
      dashboard_screen.dart
      menu_form_screen.dart
      orders_inbox_screen.dart
```

```
      rider_assign_sheet.dart
    rider/
      tasks_screen.dart
      task_detail_screen.dart
      earnings_screen.dart
    account/
      profile_screen.dart
  data/
    models/ (freezed classes)
    services/
      api_client.dart
      menu_service.dart
      order_service.dart
      rider_service.dart
      presence_service.dart
```

Routes (GoRouter)

```
/ → role gate
/buyer/home → /buyer/item/:id → /buyer/cart → /buyer/checkout → /buyer/
order/:id
/seller/dashboard → /seller/menu/new → /seller/menu/:id → /seller/orders → /
seller/order/:id
/rider/tasks → /rider/task/:id → /rider/earnings
/account
```

State: Riverpod providers for auth user, cart, orders stream, rider presence.

UI: Simple Material 3; Urdu RTL flip if locale=ur_PK.

Offline: cache menu and cart locally; queue non-idempotent requests if offline and replay when online.

I18N: Provide en/ur ARB with ~50 strings (nav labels, statuses, prompts).

SEED & FIXTURES

- Script to upload 6 sample dish images to Supabase Storage `menu/` and fill `menu_items`.
- Seed 2 sellers, 2 buyers, 1 rider with phone numbers.
- Create 3 orders: (pending, rider_assigned, delivered) for demo flows.

README S & RUN COMMANDS

- Root README: explain monorepo, prerequisites, environment variables.
- Backend README: uvicorn run, Alembic migrate, unit tests.

- Frontend README: `flutter run`, env file template, how to toggle locales.

Backend Run

```
python -m venv .venv && source .venv/bin/activate
pip install -e .
export SUPABASE_JWT_SECRET=...  # or local bypass for dev
alembic upgrade head
uvicorn app.main:app --reload --port 8000
```

Frontend Run

```
flutter pub get
flutter run -d chrome   # or Android device
```

ENV & CONFIG

Create `.env.example` files: - **server**: `DATABASE_URL`, `SUPABASE_JWT_SECRET`, `FCM_SERVER_KEY` (optional), `MAPS_PROVIDER`. - **app**: `API_BASE_URL`, `SUPABASE_URL`, `SUPABASE_ANON_KEY`.

ACCEPTANCE CRITERIA (MVP)

1) Buyer can browse menu, add item, checkout with cash, see order status transitions. 2) Seller can add/toggle menu items, receive order, accept/reject. 3) On accept with delivery=true, system **auto-assigns available rider** or returns rider_requested; manual assignment works. 4) Rider can set availability, see assigned task, mark picked_up → delivered; order status updates accordingly. 5) All roles persist to DB; refresh survives app restarts. 6) Basic notifications stubs are called (console logs in dev). 7) Urdu/English strings compile; RTL works for Urdu. 8) CI step runs `flake8/ruff` + `pytest -q` for server and `flutter analyze` for app.

TESTING

- Unit test `services/assignment.py`:
- chooses nearest available rider by haversine distance.
- handles no available riders → order=rider_requested.
- transactional integrity (rider busy during assignment).
- Snapshot tests for order state transitions.

CODING CONVENTIONS

- Type-safe models (Pydantic/Freezed), no magic strings for statuses; use enums.
- Error handling: return 4xx with machine-readable codes, human message for UI.
- No secrets committed. Use `.env` files.

NICE-TO-HAVE (stretch after MVP)

- Live GPS tracking, fee estimator (base + per-km), ratings, favorites, in-app wallet, receipts, teacher/admin web.

TASK LIST FOR CURSOR (execute in order)

1) Create monorepo structure with `app/` and `server/` + workspace settings. 2) Implement FastAPI models, schemas, routes, assignment service, Alembic. 3) Seed script + sample data + storage upload helper. 4) Flutter skeleton with routes/screens and Riverpod providers. 5) Implement menu list, item detail, cart, checkout, order tracker. 6) Implement seller dashboard + menu CRUD + orders inbox + rider assign sheet. 7) Implement rider availability + tasks list + task detail + status actions. 8) Wire API client services; add simple error toasts. 9) Add i18n (en/ur) and RTL support. 10) Write READMEs and basic tests; ensure clean `analyze` and `pytest`.

PLACEHOLDERS TO FILL BEFORE RUNNING

- SUPABASE_URL=...
- SUPABASE_ANON_KEY=...
- API_BASE_URL=http://10.0.2.2:8000 (Android emulator) or http://localhost:8000
- FCM keys (optional for dev)

OUTPUT

Generate all files with correct imports and working build. Where details are ambiguous, choose the simplest implementation that satisfies acceptance criteria and document decisions in README.