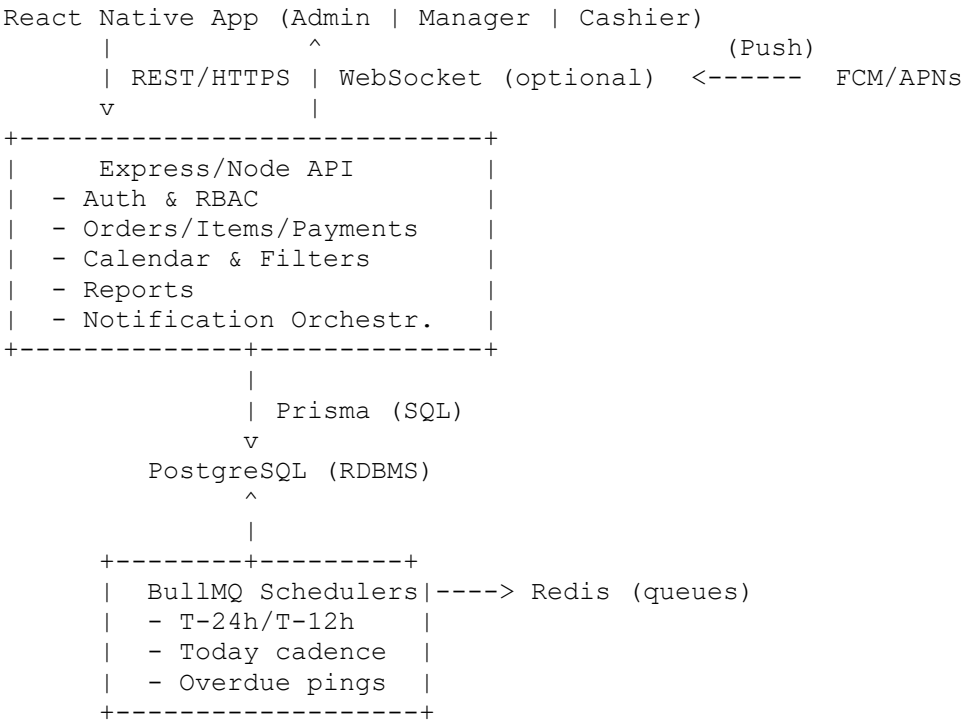
Architecture Summary

- **Mobile app (React Native, TypeScript)** for **Admin / Manager / Cashier**
- **Backend API (Express + Node.js, TypeScript)**
- **PostgreSQL** (report-friendly relational queries)
- **Job/Queue & Scheduler**: **BullMQ + Redis** for reminders & daily jobs
- **Push notifications**: **FCM/APNs** (or Expo Notifications) to staff devices
- **Auth & RBAC**: JWT (short-lived access, refresh rotation)
- **Hosting**: Render/Railway/Fly (API), Neon/Supabase (Postgres), Upstash (Redis)

High-Level Diagram (text)



Core Domains & Capabilities

1. **Order Intake (Phone → Cashier enters)**
   - Pickup only (date/time).
   - Dynamic pricing per item (cashier types item name, unit price, qty).
   - Payment states: **Paid**, **Advance Given (amount)**, **Unpaid**.

2. **Ops Views**
    - **Today / Tomorrow / This Week** lists.
    - Calendar selector (date & range).
    - Filters: status (Pending/Ready/Completed/Cancelled), payment (Paid/Advance/Unpaid), cashier, time windows.
3. **Notifications**
    - **Day-before (T-24h)** to Manager + Cashier.
    - Same-day cadence (configurable: e.g., T-12h, T-3h, T-1h, T-30m).
    - **Overdue**: repeat until Completed/Cancelled (e.g., every 30–60 min within shop hours).
    - Quiet hours window (configurable) + deduping.
4. **Reports**
    - Daily Sales (revenue, order count, paid/advance/unpaid split).
    - Popular Items (top N by quantity and revenue).
    - Pending Orders (not completed, or unpaid at/after deadline).

React Native (Staff App)

Key Screens

- **Login** (email/phone + password) → role-aware navigation.
- **Dashboards**
    - **Cashier**: Quick "New Order", **Today**, **Tomorrow**, **This Week**.
    - **Manager**: Today/Tomorrow/Week + **Deadline Alerts** feed + Reports.
    - **Admin**: Users & Settings, Reports.
- **Orders**
    - **New Order**: customer (name, phone), pickup date/time, **free-text items** (name, qty, unit price) with auto subtotal & total, payment status (Paid / Advance + amount / Unpaid), notes.
    - **Order Details**: countdown to pickup, payment/notes, change status (Pending → Ready → Completed / Cancelled).
    - **Order Lists**: Today/Tomorrow/Week; **Filters** (status, payment, cashier, time range), **Search** (customer/phone/order no).
    - **Calendar View**: month/week/day with counts & drill-down.
- **Reports**
    - **Daily Sales** (date picker): totals, avg ticket, paid/advance/unpaid.
    - **Popular Items** (date range): by qty & revenue.
    - **Pending Orders**: by date/time, with overdue highlight.
- **Notifications Inbox**: shows scheduled & sent reminders for transparency.
- **Settings (Admin/Manager)**: change reminder schedule & quiet hours.

App Tech

- Expo or bare RN; React Navigation; React Query; React Hook Form + Zod;
- Local cache for lists; optional offline draft orders (store in SQLite).
- Push: FCM/APNs (or Expo Notifications for simpler setup).

Backend (Express/Node, TypeScript)

Modules

- `auth` (JWT, refresh, RBAC middleware)
- `users` (admin creates manager/cashier)
- `orders` (CRUD, status transitions, filters, calendar)
- `payments` (recording states & amounts)
- `reports` (daily, popular items, pending)
- `notifications` (policy config, token registry, push sender)
- `scheduling` (BullMQ producers/consumers, CRON)

Cross-cutting

- Validation (Zod), logging (pino), error handler, request ID, rate limits.
- OpenAPI (Swagger) docs; health checks & readiness.

Data Model (PostgreSQL via Prisma)

```
User
- id (uuid), name, email (unique), phone?, role (ADMIN|MANAGER|CASHIER)
- password_hash, is_active, created_at, updated_at

DeviceToken
- id, user_id (fk), platform (ios|android), token, last_seen_at

Order
- id (uuid), order_no (unique, e.g., CL-2025-000154)
- customer_name, customer_phone
- pickup_at (timestamptz),  // the deadline
- status (PENDING|READY|COMPLETED|CANCELLED)
- payment_status (PAID|ADVANCE|UNPAID)
- advance_amount (numeric, default 0)
- total_amount (numeric)     // sum of items; not derived at query-time for
speed
- notes text
- created_by (fk User), created_at, updated_at

OrderItem
```

```
- id, order_id (fk), item_name (text), qty (int), unit_price (numeric),
subtotal (numeric)

Payment
- id, order_id (fk), method (CASH|CARD|TRANSFER|OTHER)
- amount (numeric), paid_at (timestamptz), ref? text

OrderStatusHistory
- id, order_id, from_status, to_status, changed_by, changed_at

ReminderPolicy
- id (singleton row or tenant-scoped),    // admin configurable
- day_before_hours int[]      // e.g., [24] (or [24, 12])
- same_day_offsets_minutes int[]   // e.g., [180, 60, 30]
- overdue_repeat_minutes int DEFAULT 60
- quiet_hours_start, quiet_hours_end (local time)

Notification
- id, order_id, target_user_id, kind (DAY_BEFORE|SAME_DAY|OVERDUE),
- scheduled_for (timestamptz), sent_at (timestamptz?), status
(SCHEDULED|SENT|SKIPPED|FAILED),
- attempt_count int, error? text
```

Indexes

- `order(pickup_at)`, `order(status)`, `order(payment_status)`
- composite: `(pickup_at, status)` for time-window queries
- `order_item(order_id)`
- `notification(scheduled_for, status)` for queue pulls
- `device_token(user_id)`

API Design (key endpoints)

Auth & Users

- `POST /auth/login` → { accessToken, refreshToken, role }
- `POST /auth/refresh`
- `GET /users/me`
- `POST /users` (ADMIN)
- `POST /devices/token` → register/refresh push token

Orders

- `POST /orders`
  - body: customer, pickup_at, items[{ item_name, qty, unit_price }], payment_status, advance_amount?, notes
  - on create: compute totals, write status history, **schedule reminders**

- `GET /orders?date=YYYY-MM-DD` (Today/Tomorrow by param; supports `range_start`, `range_end`)
- `GET /orders/this-week?anchor=YYYY-MM-DD` // convenience
- `GET /orders/:id`
- `PATCH /orders/:id` (edit details, reschedule → **reschedule reminders**)
- `POST /orders/:id/status` { to: READY|COMPLETED|CANCELLED }
- `POST /orders/:id/payment` { status: PAID|ADVANCE|UNPAID, advance_amount? }

Filters & Search

- All list endpoints accept:
  - `status[]=PENDING&status[]=READY`
  - `payment[]=PAID|ADVANCE|UNPAID`
  - `cashier_id=...`
  - `from=ISO&to=ISO`
  - `q=customer_name_or_phone`

Reports

- `GET /reports/daily?date=YYYY-MM-DD`
  → total revenue, orders count, paid/advance/unpaid breakdown, avg ticket
- `GET /reports/popular-items?from=ISO&to=ISO&limit=10`
  → aggregated by `order_item.item_name` (top by qty & revenue)
- `GET /reports/pending?date=YYYY-MM-DD`
  → not completed by end of day; unpaid regardless of date

Notifications / Settings

- `GET /settings/reminders` (ADMIN/MANAGER)
- `PATCH /settings/reminders` (ADMIN)
- (Optional) `POST /notifications/test` (send to self)

Reminder & Notification Orchestration

Scheduling logic (on order create/update):

1. Read **ReminderPolicy**.
2. Compute timestamps relative to `pickup_at`:
   - **Day-before**: T-24h (and/or T-12h if configured).
   - **Same-day**: offsets (e.g., 3h, 1h, 30m before).
3. For each timestamp:
   - If inside **quiet hours**, shift to the next available time after quiet period.
   - Deduplicate: if a reminder already exists (same kind/time window), skip.
   - Create `Notification` rows with `SCHEDULED`.

4. **BullMQ**:
    - A CRON worker (runs every minute) pulls due `Notification` rows (`scheduled_for <= now`, status = SCHEDULED).
    - Publishes push via FCM/APNs to **Manager + Cashier** (their registered `DeviceToken`s).
    - Marks `SENT`, records attempts & errors.
5. **Overdue loop** (if `now > pickup_at` and order not `COMPLETED|CANCELLED`):
    - Schedule a repeating job at `overdue_repeat_minutes` within shop hours until state changes.
    - Stop immediately when order becomes `COMPLETED|CANCELLED`.

Reliability

- Idempotent send (store provider message ID).
- Exponential backoff on provider errors.
- Dead-letter queue for repeated failures.

State Machine (Orders)

```
PENDING  --(ready for pickup)-->  READY  --(picked up & settled)--> COMPLETED
   |                 |                                  ^
   +--(cancel)--> CANCELLED  <--------------------------+
```

- Payment status can change anytime (UNPAID ↔ ADVANCE ↔ PAID).
- Status history is appended on every transition.

Performance & Scale Notes

- **Large orders (200+ items)**:
    - Keep items in a separate table; compute `total_amount` at write time to avoid heavy reads.
    - Use pagination/virtualized lists in RN for long orders.
- **List queries**: filtered by `pickup_at` ranges with proper indexes → fast **Today/Tomorrow/Week** views.
- **Reporting**: aggregate queries rely on indexed columns and date ranges. For month/year, consider materialized daily snapshots later.

Security & Ops

- **JWT** (short TTL access, refresh rotation), **RBAC** middleware.
- **Helmet**, CORS locked to app, rate limit auth endpoints.
- **Audit logs** for cancellations and payment changes.
- **Backups**: daily DB snapshots (7–30 day retention).
- **Monitoring**: Health checks, uptime, logs (pino → Logtail/BetterStack).
- **Secrets** in platform env vars; never in repo.

CI/CD & Environments

- **Monorepo** (optional): `apps/mobile`, `apps/api`, `packages/shared` (types/schemas).
- **CI**: GitHub Actions → lint, typecheck, unit tests; API deploy to Render/Railway; EAS/fastlane for app builds.
- **Envs**: dev / staging / prod with isolated DB/Redis.

MVP Backlog (2–3 sprints)

Sprint 1

- Auth & RBAC, Users
- Create Order (dynamic items, payments)
- Today/Tomorrow/Week lists + filters
- Basic Daily Sales report
- Device token registration

Sprint 2

- ReminderPolicy + Notification scheduling (T-24h + same-day)
- Overdue repeat + quiet hours
- Popular Items & Pending Orders reports
- Order status transitions & history

Sprint 3

- Calendar month view + range filters
- Exports (CSV/PDF for daily sales)
- Error handling, audit logs, polish

Example Contracts

Create Order

```
POST /orders
{
  "customer": { "name": "Ayesha", "phone": "+94XXXXXXXXX" },
  "pickupAt": "2025-08-21T15:30:00+05:30",
  "items": [
    { "itemName": "Chocolate Gateau 1kg", "qty": 1, "unitPrice": 6500 },
    { "itemName": "Eclairs", "qty": 50, "unitPrice": 120 }
  ],
  "payment": { "status": "ADVANCE", "advanceAmount": 3000 },
  "notes": "Write 'Happy 21st' in gold."
}
```

201 →

```
{
  "orderId": "ord_123",
  "orderNo": "CL-2025-000154",
  "status": "PENDING",
  "totalAmount": 12500
}
```

List This Week

```
GET /orders?from=2025-08-18T00:00:00+05:30&to=2025-08-24T23:59:59+05:30
  &status[]=PENDING&status[]=READY
  &payment[]=ADVANCE
  &q=ayesha
```

Update Payment

```
POST /orders/ord_123/payment
{ "status": "PAID", "advanceAmount": 0 }
```

Cost-conscious Hosting (typical)

- API: Render/Railway free/low-tier (US$0–7)
- Postgres: Neon/Supabase free tier
- Redis (queues): Upstash free tier
- Push: FCM (free), APNs (free), or Expo (free tier good to start)
