PIVOTPOS

Complete Product, Operations, API and Deployment Guide

The working handbook for platform owners, business teams, developers and deployment engineers.

BUSINESS OPERATIONS

Retail, restaurant, hotel, pharmacy and services

OFFLINE + LAN

SQLite continuity, one branch host and ordered sync

TECHNICAL

Fastify API, Next.js web, Flutter BLoC and PostgreSQL

RELEASE

Local, managed cloud, VPS and device distribution

VERSION 1.0

July 2026 | Nextack Limited

How to use this manual

Use the web handbook for quick search and this document for setup, training, deployment and production acceptance.

OPERATORS	DEVELOPERS	RELEASE TEAMS
Sections 4-10	Sections 11-16	Sections 17-20

Security note: never place live secrets, passwords, payment keys, database URLs or Firebase service accounts in this manual or a client application.

Version 1.0 - July 2026

1. About this guide

This is the master operating guide for PivotPOS. It is written for:

- platform owners who operate the PivotPOS SaaS;
- business owners and administrators;
- cashiers, waiters, managers and inventory staff;
- developers, deployment engineers and support teams.

Use the following labels when reviewing capabilities:

- **Available**: implemented in the current repository.
- **Configuration required**: implemented, but depends on credentials, hardware or third-party setup.
- **Controlled rollout**: modeled or described, but requires business-specific validation before production use.

Never place real passwords, Paystack keys, SMTP app passwords, Firebase service accounts, database URLs or deployment tokens in documentation, screenshots, GitHub issues or the Flutter application. Rotate any credential that has been shared outside the deployment secret manager.

2. Product overview

PivotPOS is a multi-tenant, offline-first point-of-sale and business operations platform. A tenant is one business. Each tenant owns its branches, staff, roles, devices, catalog, customers, orders, stock, printers, settings and subscription. The API derives tenant identity from an authenticated session; clients do not choose another tenant by posting a tenant ID.

The system contains three applications:

Application	Technology	Main purpose
Business and platform web	Next.js 16, React 19, TypeScript	Onboarding, administration, reports, online storefront and platform control
API	Node.js, Fastify, TypeScript, Prisma	Authentication, tenancy, permissions, commerce, billing, sync and integrations
POS client	Flutter, Dart, BLoC, SQLite	Android, iOS/iPadOS, macOS and Windows selling, printing and offline operation

PostgreSQL is the central database. Each POS installation has a tenant-scoped SQLite database that stores the catalog, stock snapshots, orders, payments, printer profiles, outbox events and synchronization state needed for offline operation.

3. Current feature map

3.1 Shared business core

- Multi-tenant business isolation.
- Multiple branches and branch-scoped staff.
- Dynamic roles and granular permissions.
- Owner email/password sign-in and staff PIN unlock.
- Registered devices with stable device keys.
- One active session per account with previous-device takeover.
- Plans, trials, subscriptions, grace periods and feature limits.
- Products, categories, brands, SKU, barcode and stock tracking.
- Customers with name, email, phone and order history.
- Orders, payments, discounts, void controls and receipts.
- Cash, card and transfer payment selection with cash change calculation.
- Excel import for products, staff and historical orders.
- Dashboard, sales reports and operational notifications.
- Business themes, logo, cover image, receipt style and receipt fields.
- SMTP email, Firebase push registration and configurable error alerts.
- Public ordering website with catalog, checkout, reviews and reservations.

3.2 Restaurant, cafe, bar and lounge

- Floors/service areas and tables.
- Waiter-owned table orders and table locks.
- Cashier access to all tables.
- Quantity increment/decrement and preparation notes.
- Send-to-kitchen operation.
- Dynamic printer stations such as Kitchen, Bar, Lounge, Grill or VIP Bar.
- Split, merge, move, settle and authorized void operations.
- Business/trading sessions that may cross midnight or occur more than once per calendar day.
- Stable table QR paths with daily/session activation behind the same printed QR code.
- Online/table orders reviewed by staff before processing.
- Reservations with flat or per-person pricing and deposit configuration.
- Recipe and portion stock deduction for ingredients measured by unit, portion or volume.

3.3 Retail and general stores

- Barcode-ready catalog, categories and brands.
- Branch stock, reorder thresholds and auditable stock adjustments.
- Retail checkout and final receipt workflow.
- Customer capture and purchase history.
- Public ecommerce-style catalog and order approval.
- Product-level and category-level printer routing when labels or preparation stations are needed.

3.4 Hotel and hospitality

- Hotel business type and hospitality terminology.
- Rooms/service areas represented in the service-area workflow.
- Food, beverage and service orders through the POS and public ordering site.
- Reservations with room/table/service/event reservation types.
- Multi-outlet printer routing and business sessions.

PMS folio posting, room inventory, check-in/check-out, night audit and channel manager integrations are **controlled rollout** items and must not be treated as complete hotel PMS functions.

3.5 Pharmacy and health store

- Product catalog and stock.
- Batch/lot receipt, quantity and expiry date tracking.
- Customer records and order history.
- Low-stock and operational notifications.

Prescription upload, controlled-drug compliance, recalls and jurisdiction- specific pharmacy rules are **controlled rollout** and require legal review.

3.6 Salon, spa and service businesses

- Service products.
- Customers, staff and appointments.
- Staff assignment and overlap prevention for active appointments.
- Public service requests and reservations.
- Normal POS settlement and receipts.

Commission payroll, job cards, hourly timesheets and calendar-provider sync are not complete standalone modules.

4. Accounts, tenancy and access

4.1 Account types

Platform user operates PivotPOS itself. Platform users are separate from business staff and use `/platform/login`.

Business owner/admin uses the web admin to manage one tenant. Owners sign in with email and password and can be assigned across businesses through separate memberships.

POS staff includes cashiers, waiters, managers and inventory staff. A staff member signs in online once on a registered device, then can use a 4-6 digit PIN to unlock that known device.

4.2 Permission keys

Permissions are server-enforced:

Area	Permission keys
Sales	`sales.process`, `sales.discount`, `sales.void`
Orders	`orders.create`, `orders.view`, `orders.view_own`, `orders.update_own`, `orders.send_to_kitchen`, `orders.void`
Payments and receipts	`payments.process`, `receipts.print`
Tables and day control	`tables.view`, `operations.view_day`, `operations.start_day`, `operations.end_day`
Inventory	`inventory.view`, `inventory.manage`
Customers and reports	`customers.view`, `customers.manage`, `reports.view`
LAN sync	`sync.client.connect`, `sync.host.start`, `sync.host.stop`, `sync.host.takeover`, `sync.emergency_mode`
Administration	`staff.manage`, `settings.manage`

Suggested role profiles:

- **Waiter**: tables, own orders, create/update own orders and send to kitchen. Do not grant payment, receipt or day-close permissions unless required.
- **Cashier**: view/create orders, process payments, print receipts and view day status. Add start/end day only when the cashier is responsible for shifts.
- **Inventory officer**: inventory view/manage and reports view.

- **Manager**: order view, void, payments, reports, tables, day control and selected administration.
- **LAN host operator**: host start/stop plus the normal cashier permissions. Grant takeover only to a trusted manager.

4.3 Device behavior

Every POS installation creates a stable device key based on the installation. Register the key under **Business Admin > Devices** and assign the device to the correct branch. Editing a display name is safe. Changing the key breaks the trust relationship and should only be done as a deliberate replacement.

When the same user signs in on another registered device, PivotPOS allows the new login and invalidates the previous active session. The previous device will be asked to sign in again on its next authenticated request.

5. First-time business setup

1. Open the landing page and select **Start free**.
2. Choose the business type and subcategory.
3. Enter the business name, country, state, address, currency, timezone and language.
4. Add a valid business email and phone number.
5. Create the owner account and verify the email OTP.
6. Sign in and complete the business profile.
7. Upload the logo. The current limit is 300 KB. Upload a cover image up to 500 KB for the public ordering site.
8. Choose one of the ten business theme colors.
9. Configure tax, invoice prefix, receipt style, footer and custom fields.
10. Create branches and assign each branch a code, address and timezone.
11. Create roles, then add staff with branch assignment and PIN.
12. Create categories and brands before products, or import products and allow the importer to create missing category/brand names.
13. Add products, stock, barcode settings and printer station routing.
14. Add printer profiles under **Printers**.
15. Install the POS, register each device key and perform the first online sync.
16. For LAN mode, configure one branch host and test host/client communication.
17. Start a trading session and complete a cash test order.
18. Verify receipt, stock movement, dashboard totals and pending sync count.

6. Business administration guide

6.1 Dashboard

The dashboard displays sales totals, order volume, pending online orders, business-session context and operational shortcuts. Use branch and date/session filters when reviewing a location. Online-order counts should be reviewed before closing a session.

6.2 Categories, brands and products

Create categories before products when printer routing matters. A restaurant category such as `Cocktails` can route to `BAR`; `Grills` can route to `KITCHEN` or `GRILL`. Category station values are names, so a business can create multiple custom stations.

Products support product type, SKU, barcode, price, cost, tax, category, brand, stock tracking, reorder level, active state and station routing. Barcode generation is useful only when the business has enabled barcode handling and a scanner or camera workflow.

Spreadsheet imports update records by ID/SKU where supported. Missing category or brand names are created for the current tenant. Review spelling before importing because `Soft Drinks` and `Soft Drink` become different categories.

6.3 Inventory

Do not edit stock quantity directly. Create an inventory adjustment with a reason. The API records an immutable movement and updates the stock projection. Use positive quantities for receipt/correction-in and negative quantities for waste, damage or correction-out.

Batch and expiry records are used for lots received with an expiry date. They help pharmacy, food and beverage businesses identify expiring stock. They do not replace regulatory compliance or automatic FEFO allocation.

Recipes connect one sellable product to ingredient products. Example: a cocktail can consume 40 ml of vodka and one mixer. Selling a full bottle is a separate sellable product or unit conversion. All recipes must use consistent base units so `750 ml`, `40 ml` and remaining stock reconcile.

6.4 Customers

Capture name, email and phone at the cashier or through online checkout. Email is the strongest matching key when present, but do not assume every customer has one. Review a customer record to see orders associated with that customer. Receipt emails require SMTP configuration and a valid customer email.

6.5 Staff, roles, branches and devices

Add the role first, assign permissions, then add staff. Keep users branch-scoped unless they genuinely need tenant-wide visibility. Deactivate departed staff instead of deleting transaction history. Register every physical terminal and use a recognizable device name such as `Main Bar Android 01`.

6.6 Printers and receipt design

Printer setup belongs in business administration. POS devices download the profiles for their tenant and branch. Receipt styles, logo, date/time, cashier name, payment method, currency code, custom fields and footer are controlled by business settings and plan features.

6.7 Online ordering

Complete the public profile, logo, cover, hero title, subtitle, about text and welcome message. Configure delivery fee, service fee, payment mode and order approval. Activate a general ordering session or a table session. Staff review pending orders under **Online orders** or in the POS online-order screen.

The stable public URLs are:

```
/order/<tenant-slug>
/order/<tenant-slug>/table/<table-id>
```

The printed table QR remains stable. Session activation changes whether the stable destination accepts orders, so a new physical QR does not need to be printed every day.

6.8 Reservations and appointments

An **appointment** schedules a service and staff member. A **reservation** holds a table, room, service slot or event capacity, may have guest count, deposit, expiry/grace rules and public booking.

Create reservation pricing rules before enabling public reservation checkout. A rule can be flat-price or per-person, with minimum/maximum guests and a deposit. Provider-backed online payment requires Paystack configuration.

6.9 Plans and billing

The seeded plans are Starter, Essentials, Growth and Scale, plus website add-on plans. Plan prices, currency, limits and features can be edited by the platform owner. The API enforces limits for users, branches and devices.

Do not delete data immediately on downgrade. Block creation/use of excess resources, show what exceeds the target plan, provide an export window and let the owner deactivate excess records. A grace period can warn the tenant before commerce endpoints are blocked.

7. Daily operating procedures

7.1 Open a trading session

1. Sign in on the host/cashier device.
2. Confirm the branch and current date/time.
3. Open **Business day**.
4. Start a session with the correct business date and optional note.
5. Confirm other devices show the open status after LAN/cloud sync.

A session can start at 08:00, 12:00 or any time, close after midnight, and a branch can have multiple sessions on one calendar date. Reports should filter by business-session IDs when operational shift totals are required.

7.2 Restaurant order flow

1. Waiter unlocks the registered client device.
2. Waiter selects an available table. The table becomes assigned.
3. Add products; repeated products increase quantity instead of creating unnecessary duplicate lines.
4. Add notes and send the required lines to preparation stations.
5. Kitchen/bar/lounge printers receive only their routed items.
6. Cashier opens the active order, prints a bill if the business workflow requires one, selects payment method and settles.
7. Final receipt prints with cashier name and payment method.
8. The host broadcasts the completed state and the table unlocks on clients.

Waiters should not settle or print customer receipts unless their role grants those permissions. Authorized users can void an unpaid order with a reason.

7.3 Retail order flow

1. Cashier scans/searches products.
2. Adjust quantities and attach a customer when available.
3. Select cash, card or transfer.
4. For cash, enter amount received and confirm change.
5. Settle once payment is approved.
6. Print or email the final receipt.

Retail does not require restaurant bill, table, merge or split terminology.

7.4 Close a trading session

1. Resolve open tables/orders and pending online orders.
2. Confirm pending local and cloud sync counts are zero.
3. Confirm payment totals by method.
4. Reconcile cash on hand.
5. Review printer failures and voids.
6. End the current session with a closing note.
7. Export/report the session totals as required.

8. Offline and LAN synchronization

8.1 Offline-first storage

The POS writes orders, order items, payments and an outbox event in one SQLite transaction. If the network is unavailable, the event remains pending. The catalog, stock snapshot, printer profiles and recent operations remain local. Never uninstall the app, clear application storage or delete the SQLite file while pending events exist.

The cloud API is idempotent: retries use the same client-generated UUID so the same accepted order does not reduce stock twice.

8.2 Single-host LAN mode

One registered device is the active host for a branch. The host can be an Android cashier terminal or desktop. Other devices communicate with the host over the router; only the host needs to push branch events to the cloud.

- LAN client tick: approximately every 3 seconds.
- Host configuration refresh: approximately every 15 seconds.
- Default host port: `8787`.
- Events are acknowledged by exact event ID.
- The host assigns an authoritative sequence; device clock differences do not decide event order.
- A branch access key and protocol version protect LAN requests.
- Duplicate events return the existing receipt instead of being applied twice.

If `Require host before orders` is enabled and the host is unavailable, emergency standalone behavior depends on the branch setting and `sync.emergency_mode` permission. Allowed emergency orders remain local until the host returns.

8.3 LAN setup

1. Put all devices on the same trusted router.
2. Avoid guest Wi-Fi/client isolation.
3. Register all devices to the same tenant and branch.
4. In **Business Admin > Local sync**, choose the primary host device.
5. Save the host address, normally the host's router IP, and port `8787`.
6. Add the local network CIDR, for example `192.168.1.0/24`.
7. Grant the host user `sync.host.start` and client staff `sync.client.connect`.
8. Start the host from the POS.
9. Verify the host-only connected-device page shows current user and last seen.
10. Place and settle a test order before disconnecting internet.

IP entry is not normally daily. Reserve the host IP in the router DHCP settings so the same device receives the same address. A changed router or subnet requires updating the saved host address/CIDR.

8.4 LAN recovery

If a client does not update:

1. Confirm both devices are on the same SSID/subnet.
2. Confirm the host app is open and host status is active.
3. Confirm the host IP and port `8787`.
4. Check whether the router uses guest/client isolation.
5. Check device registration and branch assignment.
6. Open debug logs and look for LAN push/pull, unauthorized key, protocol or timeout messages.
7. Do not clear data as a first fix; preserve pending events for diagnosis.

9. Thermal printing

Network ESC/POS printing is recommended for fixed Xprinter-style receipt, kitchen, bar and lounge printers. The device talks directly to the printer over TCP, normally port `9100`; internet is not required.

9.1 Network printer checklist

1. Connect printer Ethernet/Wi-Fi and POS device to the same router/subnet.
2. Print the printer self-test/configuration page to find its IP.
3. Reserve or set a static IP that belongs to the router subnet.
4. Test the port:

```
nc -vz 192.168.1.242 9100
```

1. Send a no-cut test:

```
printf '\x1b@\nPIVOTPOS TEST\nPrinter network is working\n\n\n\n' \
  | nc 192.168.1.242 9100
```

1. Add the profile in **Business Admin > Printers** and sync the POS.
2. Use the in-app test print.

If the Mac must be given a different manual IP to print and then loses internet, the printer is on the wrong subnet. Move the printer to the router's subnet instead of repeatedly changing the computer address.

9.2 Routing

- `RECEIPT` prints full customer receipts.
- Preparation stations print only matching item lines.
- `ALL` is a deliberate backup and receives every station job.
- Multiple printers can share the same station.
- Custom station names are supported when product/category routing uses the same exact name.

Print jobs are local. A hardware failure must not change whether the financial order is accepted. The local print queue records errors and retries.

10. Email, push and payments

10.1 SMTP

Gmail testing uses `smtp.gmail.com`, port `587`, `SMTP_SECURE=false`, the Gmail address and a Google App Password. Do not use the normal Google password. Business invitation, password reset, verification, online-order notification and receipt email behavior also depends on the business email settings.

10.2 Push notifications

Firebase Admin credentials belong only on the API. Android/iOS Firebase client files belong in their platform projects. Push is used for online orders and selected operational alerts. Android and Apple platforms require correct Firebase/APNs setup; Windows/macOS support must be validated against the chosen desktop notification transport before promising background delivery.

10.3 Paystack

PivotPOS owns the Paystack integration. Businesses provide settlement account details; they do not enter secret keys or split codes. The admin retrieves a bank list, resolves account number to account name and stores the provider recipient/subaccount identifiers needed for settlement.

Use test keys until the full webhook, split, refund, reconciliation and settlement flow passes UAT. Keys are set through protected platform settings or deployment secrets, never Flutter.

11. Local development

11.1 Requirements

- Git.
- Docker Desktop and Docker Compose.
- Node.js 22 LTS and npm.
- Flutter stable and Dart.
- Android Studio for Android.
- Xcode/CocoaPods for iOS and macOS.
- Visual Studio 2022 with Desktop development with C++ for Windows.

Run:

```
node --version
npm --version
docker --version
docker compose version
flutter doctor -v
```

11.2 Docker database plus local applications

From the repository:

```
docker compose up -d postgres

cd apps/api
cp .env.example .env
npm install
npm run db:generate
npm run db:migrate
npm run db:seed
npm run dev
```

In another terminal:

```
cd apps/admin
cp .env.example .env.local
npm install
npm run dev
```

Addresses:

- Web: `http://localhost:3000`
- API: `http://127.0.0.1:8000`
- Health: `http://127.0.0.1:8000/health`
- PostgreSQL: `127.0.0.1:5433`

11.3 Complete Docker stack

```
export JWT_SECRET="$(openssl rand -hex 32)"
docker compose up --build
docker compose exec api npm run db:seed
```

11.4 Flutter local run

macOS:

```
cd apps/pos
flutter pub get
./scripts/run_macos_local.sh
```

Physical Android:

```
cd apps/pos
./scripts/run_android_local.sh DEVICE_SERIAL
```

Manual Android command:

```
flutter run -d DEVICE_SERIAL \
  --dart-define=API_URLS=http://127.0.0.1:8000,http://YOUR_MAC_IP:8000,http://10.0.2.2:8000
\
  --dart-define=API_LOGS=true
```

`127.0.0.1` on a physical phone means the phone itself. Use Mac LAN IP or `adb reverse tcp:8000 tcp:8000`. When ADB reports more than one device, pass `-s DEVICE_SERIAL` to `adb reverse` or use the helper script.

11.5 Local verification

```
cd apps/api
npm run typecheck
npm test
npm run test:integration
npm run build

cd ../admin
npm run lint
npm run build

cd ../pos
dart analyze
flutter test
```

The API package intentionally has no `lint` script; use `npm run typecheck`.

12. Managed staging deployment

The current staging pattern is:

- Next.js web: Vercel.
- Fastify API: Render.
- PostgreSQL: Neon.
- DNS/proxy: Cloudflare when a custom domain is added.

12.1 Database

Create a Neon PostgreSQL database, copy its pooled/standard connection URL and set it as `DATABASE_URL` on the API service. Never expose it in client code. Run:

```
npm run db:generate
npm run db:deploy
npm run db:seed
```

Use `db:deploy`, not `db:migrate`, in staging/production.

12.2 API service

Render settings:

```
Root directory: apps/api
Build command: npm ci && npm run db:generate && npm run build
Start command: npm start
Health path: /health
```

Required variables include `DATABASE_URL`, `JWT_SECRET`, `ADMIN_ORIGIN`, platform credentials and configured integration secrets.

12.3 Web service

Vercel root directory is `apps/admin`. Set:

```
API_URL=https://YOUR-API-HOST
API_DOCS_OWNER_EMAIL=YOUR-PRIVATE-DOCS-OWNER
```

`ADMIN_ORIGIN` on the API must be the deployed application origin, for example `https://pivotpos.example`, not the Vercel account/dashboard URL.

12.4 Flutter staging build

```
flutter build apk --release \
  --dart-define=API_URL=https://YOUR-API-HOST \
  --dart-define=API_LOGS=false
```

The API URL is a build-time value. A hot reload cannot replace a value compiled into an already installed release APK.

13. VPS deployment

Use Ubuntu 24.04 LTS with at least 2 vCPU, 4 GB RAM and 60 GB SSD for a small production start. Prefer managed PostgreSQL; if PostgreSQL shares the VPS, increase memory and implement off-server backups.

13.1 Prepare the server

```
sudo apt update
sudo apt install -y ca-certificates curl git ufw
curl -fsSL https://get.docker.com | sudo sh
sudo usermod -aG docker "$USER"
sudo ufw allow OpenSSH
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable
```

Log out/in after adding the Docker group.

13.2 Deploy with Compose

```
git clone YOUR_PRIVATE_REPOSITORY_URL pivotpos
cd pivotpos
cp .env.production.example .env.production
openssl rand -hex 32
```

Fill secrets, then:

```
docker compose --env-file .env.production \
  -f docker-compose.yml \
  -f docker-compose.prod.yml \
  up --build -d
```

Use a reverse proxy such as Caddy or Nginx for TLS. Example Caddyfile:

```
app.example.com {
  reverse_proxy 127.0.0.1:3000
}

api.example.com {
  reverse_proxy 127.0.0.1:8000
}
```

Do not publish PostgreSQL port `5432/5433` to the internet. Restrict API/admin container ports to localhost or the proxy network.

13.3 VPS release procedure

1. Verify an off-server database backup.
2. Pull the reviewed release tag.
3. Build new images.
4. Run `prisma migrate deploy` once.
5. Restart API, then web.
6. Check `/health`, login, catalog, test order and sync.
7. Keep the previous image/tag for rollback.

14. Production environment reference

Core API:

```
NODE_ENV=production
PORT=8000
DATABASE_URL=postgresql://...
JWT_SECRET=<64-or-more-random-hex-characters>
ADMIN_ORIGIN=https://app.example.com
```

```
PLATFORM_ADMIN_EMAIL=...
PLATFORM_ADMIN_PASSWORD=...
```

Mail:

```
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_SECURE=false
SMTP_USER=...
SMTP_PASS=...
SMTP_FROM_NAME=PivotPOS
SMTP_FROM_EMAIL=...
```

Notifications/integrations:

```
FIREBASE_PROJECT_ID=...
FIREBASE_SERVICE_ACCOUNT_JSON=...
PAYSTACK_SECRET_KEY=...
ERROR_ALERT_TO=...
ERROR_ALERT_MIN_STATUS=500
ERROR_ALERT_CATEGORIES=server
```

Use the platform settings screen for settings designed to be dynamic. Keep bootstrap secrets and database credentials in the host secret manager.

15. API guide

15.1 Basics

Local base URL: `http://127.0.0.1:8000`

Staging/production base URL: the deployed HTTPS API origin.

Protected requests use:

```
Authorization: Bearer <access-token>
Content-Type: application/json
```

Access tokens are short-lived. Refresh tokens rotate. On `401`, the Flutter client should refresh once and retry the original request. Repeated `401` means the session is invalid and the user must sign in again.

15.2 Status codes

Code	Meaning
`200/201`	Success/created

Code	Meaning
`400`	Validation or business-rule input error
`401`	Missing, invalid or expired authentication
`402`	Subscription/plan access is unavailable
`403`	Permission, device, branch or network policy denied
`404`	Route or record not found
`409`	State conflict or duplicate/idempotency conflict
`429`	Rate limited
`500`	Unexpected server failure

Do not blindly retry every `4xx`. Retry network failures and token refresh. Resolve validation, permission and conflict responses according to the payload.

15.3 Endpoint groups

The current API exposes:

- Authentication and verification.
- Business settings, branches, roles, staff and devices.
- Catalog, inventory, batches and recipes.
- Orders, settlement, split/merge/table movement and business sessions.
- Customers, appointments, reservations and reviews.
- Printers, notifications, online ordering and public storefront.
- Offline cloud pull and LAN-host control.
- Plans, checkout and provider webhooks.
- Separate platform-owner endpoints.

The source-verified route inventory is in `docs/API.md`. The detailed core machine-readable contract is `docs/openapi.yaml`. The private in-app reference is `/api-docs` and is restricted by owner identity.

15.4 Login example

```
curl -X POST http://127.0.0.1:8000/v1/auth/login \
  -H 'content-type: application/json' \
  -d '{"email":"owner@example.com","password":"Strong-password-123"}'
```

15.5 Idempotency and sync

Offline orders, items, payments and adjustments use client UUIDs. Replaying an already accepted event must return the existing state or a recognized idempotent response. A same-table conflict that returns the same active order ID is treated as acknowledgement; a different active order needs reconciliation.

16. Security and privacy

- Use HTTPS everywhere outside local development.
- Use a 32+ byte random JWT secret and rotate compromised secrets.
- Keep refresh tokens in secure HTTP-only cookies on web and secure storage on Flutter.
- Hash passwords and PINs with Argon2id.
- Restrict staff with least-privilege roles and branch scope.
- Register devices; deactivate lost or replaced devices.
- Keep platform accounts separate from tenant accounts.
- Redact passwords, PINs, tokens, cookies and authorization headers in logs and email alerts.
- Apply rate limits to login, PIN and recovery endpoints.
- Validate request bodies with schemas.
- Verify payment webhook signatures.
- Do not trust a client-supplied tenant ID.
- Back up PostgreSQL and periodically test restoration.
- Add PostgreSQL row-level security before high-risk enterprise production as a second isolation layer.
- Publish privacy policy, terms, retention schedule and breach procedure before onboarding external businesses.

17. Backups, monitoring and support

17.1 Database backup

Daily managed backups plus weekly off-provider exports are recommended. Example:

```
pg_dump "$DATABASE_URL" --format=custom --file=pivotpos-$(date +%F).dump
```

Test restore in a separate database:

```
createdb pivotpos_restore_test
pg_restore --clean --if-exists --dbname=pivotpos_restore_test BACKUP.dump
```

17.2 Monitoring

Monitor:

- API `/health` uptime and latency.
- HTTP `500` rate.
- login/PIN failures and `429` rate.
- subscription webhook failures.
- outbox pending age/count per device.
- LAN host heartbeat/last seen.
- print queue failures.
- PostgreSQL connections, storage and slow queries.
- email and push delivery failures.

Expected validation or subscription errors should not flood owner email. Use platform error-alert categories and a sensible minimum status.

17.3 Support data

When reporting a POS issue include:

- tenant slug and branch;
- device name/key suffix and app version;
- user/role;
- timestamp with timezone;
- operation and order ID;
- pending count;
- sanitized API/LAN error payload;
- printer profile name/IP/port when relevant.

Never request a user's password, PIN, refresh token or payment secret.

18. Release builds and updates

18.1 Android

```
flutter build appbundle --release \
  --dart-define=API_URL=https://api.example.com \
  --dart-define=API_LOGS=false
```

Use a backed-up upload keystore and increase `version`/build number for every release. Play Store distribution provides normal update delivery. Direct APK users must install a newer APK signed by the same key unless an in-app update service is added.

18.2 iOS and macOS

Build on macOS with Apple Developer signing:

```
flutter build ios --release --dart-define=API_URL=https://api.example.com
flutter build macos --release --dart-define=API_URL=https://api.example.com
```

The current Firebase iOS dependency requires deployment target 15.0 or later. Use App Store/TestFlight for iOS and signed/notarized DMG or Mac App Store distribution for macOS.

18.3 Windows

Build on Windows with Visual Studio Desktop C++:

```
flutter build windows --release --dart-define=API_URL=https://api.example.com
```

Package the complete Release folder in an installer/MSIX and code-sign it. Do not distribute only the `.exe`; Flutter desktop requires adjacent DLL/data files.

18.4 Update strategy

Web and API fixes deploy centrally. Flutter UI/native/offline database changes require an application update. Keep API changes backward-compatible for at least the supported client versions and use the LAN protocol minimum-version gate to block unsafe mixed versions.

19. Troubleshooting

Cannot reach PivotPOS server

- Physical devices cannot use Mac `127.0.0.1`.
- Check `/health` from another machine on the same LAN.
- Confirm API listens on `0.0.0.0:8000`.
- Use the current Mac LAN IP or ADB reverse.
- Check firewall, hotspot isolation and VPN.

Device is not registered

Compare the exact stable device key with **Devices**, confirm it is active and assigned to the same branch. Reinstalling/clearing app storage may create a new installation identity.

Invalid PIN

Confirm tenant slug spelling, membership status, branch, registered device and that the PIN belongs to that tenant membership. The same email can have different memberships/PINs.

Products are missing

Confirm product active state, branch assignment/stock, tenant, plan/module and initial catalog sync. On the host, synchronize cloud data after login before clients pull the host snapshot.

Day appears open on one device and closed on another

Confirm both devices use the same branch and host. Check LAN snapshot pulls and pending session events. Do not let a stale cloud pull overwrite a newer host sequence.

Pending count does not clear

Read the oldest outbox error first. Events are processed in order. Fix the first validation, permission, device or conflict error; later events may depend on it. Do not delete pending events until their business effect is reconciled.

Printer timeout

Confirm printer and POS are on the same subnet, port `9100` is open, the IP is not stale and router client isolation is off. A successful Mac terminal print does not prove an Android device can route to that subnet.

Prisma column does not exist

The application was deployed before its migration. Run `npm run db:deploy` against the correct database, restart the API and verify the migration table.

20. Production acceptance checklist

Business setup

- Business email, logo, address, currency and timezone are correct.
- Branches, staff, roles and devices are approved.
- Plan and grace period are correct.
- Tax, receipt and payment methods are verified.

Operations

- Trading session opens/closes across midnight.
- Cash/card/transfer settlement and change work.
- Void permissions and audit reason work.
- Customer and email receipt work.
- Online order approval and notification work.

Restaurant/hotel

- Table lock, waiter ownership and cashier override work.
- Same-item quantities merge correctly.
- Kitchen/bar/lounge/custom routing works.
- Split/merge/move and settlement unlock work.
- QR ordering session and reservations work.

Offline/LAN

- First sync completes with zero pending.
- Client order reaches host in seconds.
- Host settlement unlocks client table.
- Internet-off order is preserved and syncs after restoration.
- Duplicate replay does not duplicate stock/payment.
- Host takeover and version mismatch behavior are tested.

Printing

- Receipt, bill and preparation layouts fit 58/80 mm as configured.
- Cashier name, date/time, currency code and payment method print.
- Footer has enough trailing feed before cut.
- Printer-off error is visible and retry succeeds after power returns.

Production

- Secrets rotated and stored outside Git.
- Database migration and seed applied.
- Backups and restore test complete.
- HTTPS, CORS, email, push and Paystack test webhooks pass.
- Android/iOS/macOS/Windows packages use the production API.
- Monitoring, support contact, privacy policy and incident procedure are live.

21. Documentation map

- `docs/COMPLETE_GUIDE.md`: master handbook.
- `docs/API.md`: developer-focused API reference.
- `docs/openapi.yaml`: OpenAPI contract.
- `docs/LOCAL_SETUP.md`: local developer setup.
- `docs/DEPLOYMENT.md`: managed and container deployment.
- `docs/OFFLINE.md`: offline/cloud synchronization.
- `docs/PRINTING.md`: thermal printer setup.
- `docs/ARCHITECTURE.md`: system and security boundaries.
- `docs/PLATFORM_ADMIN.md`: software-owner operations.
- `/docs`: searchable web handbook.
- `/api-docs`: private API reference for the configured owner.