Devices Handling

- JikoXpress Pro Device & Staff Authentication Specificationge

JikoXpress Pro Device & Staff Authentication Specificationge

```
Device-first. PIN-based staff auth. Server is always the authority.
One device, one kitchen. Trust earned through physical registration.
```

Philosophy

- **Device identity first** — no device, no access. Period.
- **Server is always authority** — device only reacts, never decides
- **Minimal local storage** — only what is necessary lives on device
- **Progressive trust** — setup token → device token → staff token
- **Config as source of truth** — everything including kitchenId lives in one payload
- **Revocation is the kill switch** — no rotation complexity, owner revokes when needed
- **Device type drives auth requirements** — KIOSK needs less, POS needs more

---

Actor Overview

Actor	Who They Are	How They Auth
Platform Super Admin	QBIT SPARK team	Separate system — out of scope here
Kitchen Owner / Admin	Restaurant owner	Email + Password → Owner JWT
Device	Physical tablet / POS / kiosk	Setup flow → DEVICE_TOKEN
Staff	Kitchen employee	PIN on registered device → STAFF_TOKEN
Customer (Kiosk)	Walk-in customer	No auth — device auth is enough

---

Token Types

Token	Lifespan	Purpose	Issued At

`setupToken`	5 mins, single use	Binds physical device to claim attempt	`GET /devices/setup/token`
`DEVICE_TOKEN`	Permanent until revoked	Device identity on every request	After owner configures device
`STAFF_TOKEN`	8 hrs (one shift)	Staff session, device-bound	`POST /auth/staff/login`
`OWNER_TOKEN`	Standard session	Owner management access	`POST /auth/owner/login`

Device Types & Auth Requirements

Device Type	DEVICE_TOKEN	STAFF_TOKEN	Notes
`POS`	□ Required	□ Required	Full staff auth
`STORE_TABLET`	□ Required	□ Required	Full staff auth
`KIOSK`	□ Required	□ Not needed	Customer self-service
`KITCHEN_DISPLAY`	□ Required	□ Not needed	Passive display only

Device Type	DEVICE_TOKEN	STAFF_TOKEN	Notes
`POS`	□ Required	□ Required	Full staff auth
`STORE_TABLET`	□ Required	□ Required	Full staff auth
`KITCHEN_DISPLAY`	□ Required	□ Required	Station-bound staff auth
`KIOSK`	□ Required	□ Not needed	Customer self-service only

KIOSK is the only device type with no staff auth. Kitchen can have multiple KDS devices — one per station.

Config Payload — Single Source of Truth

Everything the device needs to know lives in one JSON payload. `kitchenId` is not stored separately — it is a field inside this payload.

```
{
  "deviceId": "dv_uuid",
  "deviceName": "Front Kiosk",
```

```
  "deviceType": "KIOSK",
  "kitchenId": "kt_uuid",
  "kitchenName": "Mama Pima Kitchen",
  "deviceStatus": "ACTIVE",
  "permissions": {
    "allowDineIn": true,
    "allowPickup": true,
    "allowDelivery": false,
    "allowPOS": false,
    "allowReports": false,
    "allowKitchenDisplay": true,
    "allowStoreAccess": false
  }
}
```

Config Hash

Hash is **always derived on the fly** from the full config payload. It is never stored as a static field.

```
hash(configPayload) → compare with hash in response
→ match   → config still fresh, proceed
→ mismatch → pull fresh config, overwrite local payload, re-derive
```

What Device Stores Locally

```
LOCAL DEVICE STORE
├── deviceToken      → permanent identity
└── configPayload    → full JSON blob above (kitchenId lives here)
```

Nothing else. No separate kitchenId field. No separate hash field.

Device Status Lifecycle

```
UNCONFIGURED   → claimed by owner, awaiting kitchen assignment
ACTIVE         → fully configured, operating normally
SUSPENDED      → kitchen subscription lapsed / unpaid
REVOKED        → decommissioned or compromised
```

Device Reaction Per Status

Status	Device Behaviour
`ACTIVE`	Check configHash → proceed normally
`SUSPENDED`	Lock screen → show "Kitchen subscription inactive. Contact admin."
`REVOKED`	Wipe all local storage → reset to setup/QR screen

Device does **not** need to know *why* it is suspended or revoked. It just reacts. The reason lives server-side.

Every API Response Envelope

Every response to a device — from boot config pull to mid-session requests — carries:

```
{
  "deviceStatus": "ACTIVE",
  "configHash": "a3f9c2d1...",
  "data": { }
}
```

Device reads `deviceStatus` and `configHash` on **every** response before processing `data`.

Device Fingerprint

Generated on the device from hardware/software attributes:

```
fingerprint = hash(
  deviceModel +
  osVersion +
  screenResolution +
  installationId     ← generated once on first app install, stored permanently
)
```

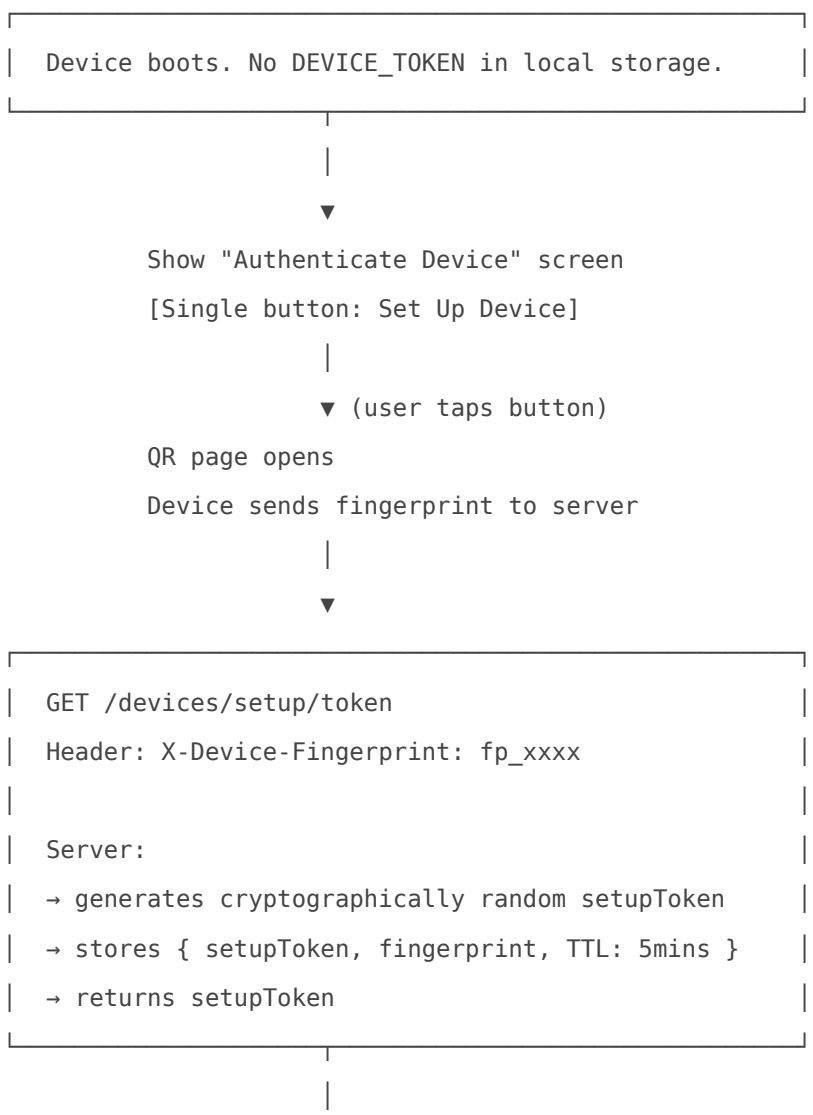
`installationId` is the anchor — generated once, never changes unless app is reinstalled.

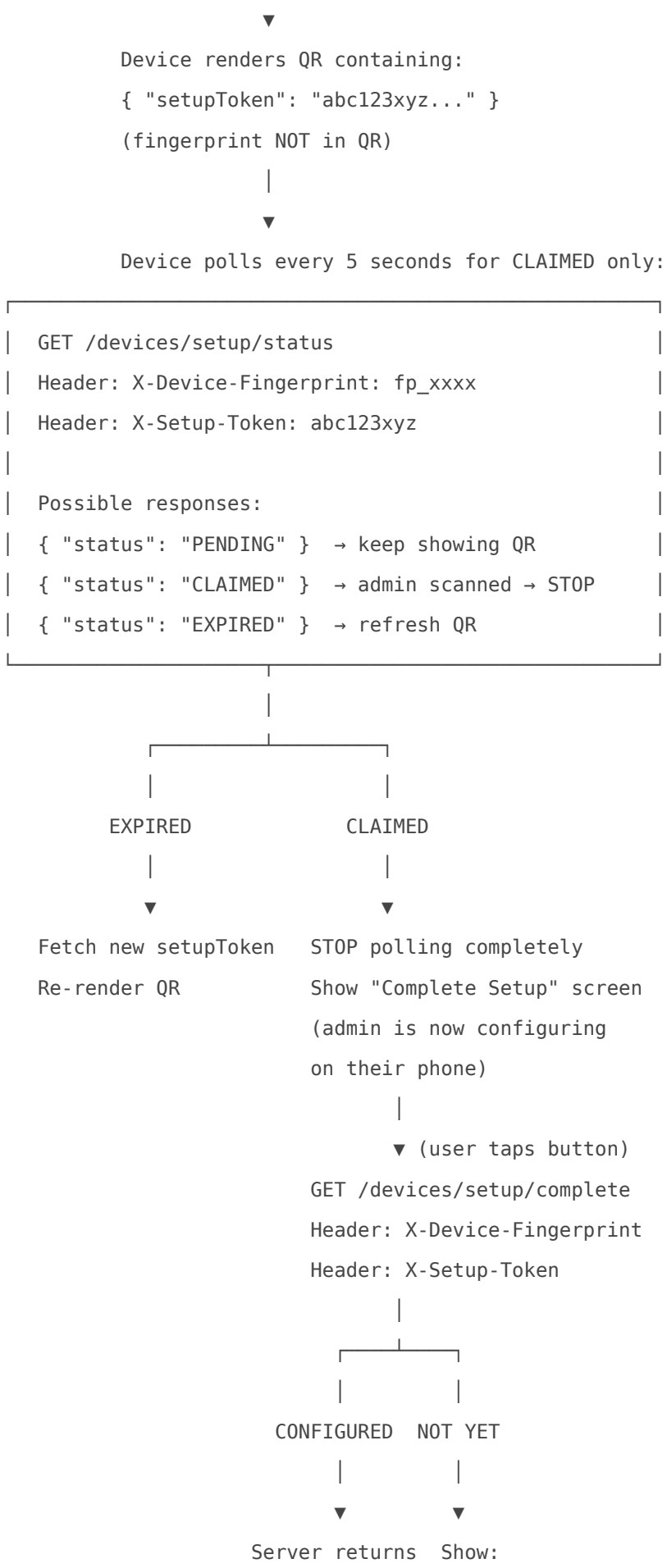
When Fingerprint Is Used

```
GET /devices/setup/token     → sent in header (binding moment)
Polling /setup/status        → sent while waiting for CLAIMED only
                               polling stops the moment CLAIMED detected

Everything after CLAIMED → fingerprint NOT needed
Everything after DEVICE_TOKEN is issued → fingerprint NOT needed
```

Fingerprint is **never** included in the QR code. It travels only in request headers, never exposed visually.

Flow 1 — Device Setup & Registration




Device renders QR containing:
{ "setupToken": "abc123xyz..." }
(fingerprint NOT in QR)
Device polls every 5 seconds for CLAIMED only:
GET /devices/setup/status
Header: X-Device-Fingerprint: fp_xxxx
Header: X-Setup-Token: abc123xyz
Possible responses:
{ "status": "PENDING" } → keep showing QR
{ "status": "CLAIMED" } → admin scanned → STOP
{ "status": "EXPIRED" } → refresh QR
EXPIRED
CLAIMED
Fetch new setupToken
Re-render QR
STOP polling completely
Show "Complete Setup" screen
(admin is now configuring
on their phone)
▼ (user taps button)
GET /devices/setup/complete
Header: X-Device-Fingerprint
Header: X-Setup-Token
CONFIGURED
NOT YET
Server returns
Show:

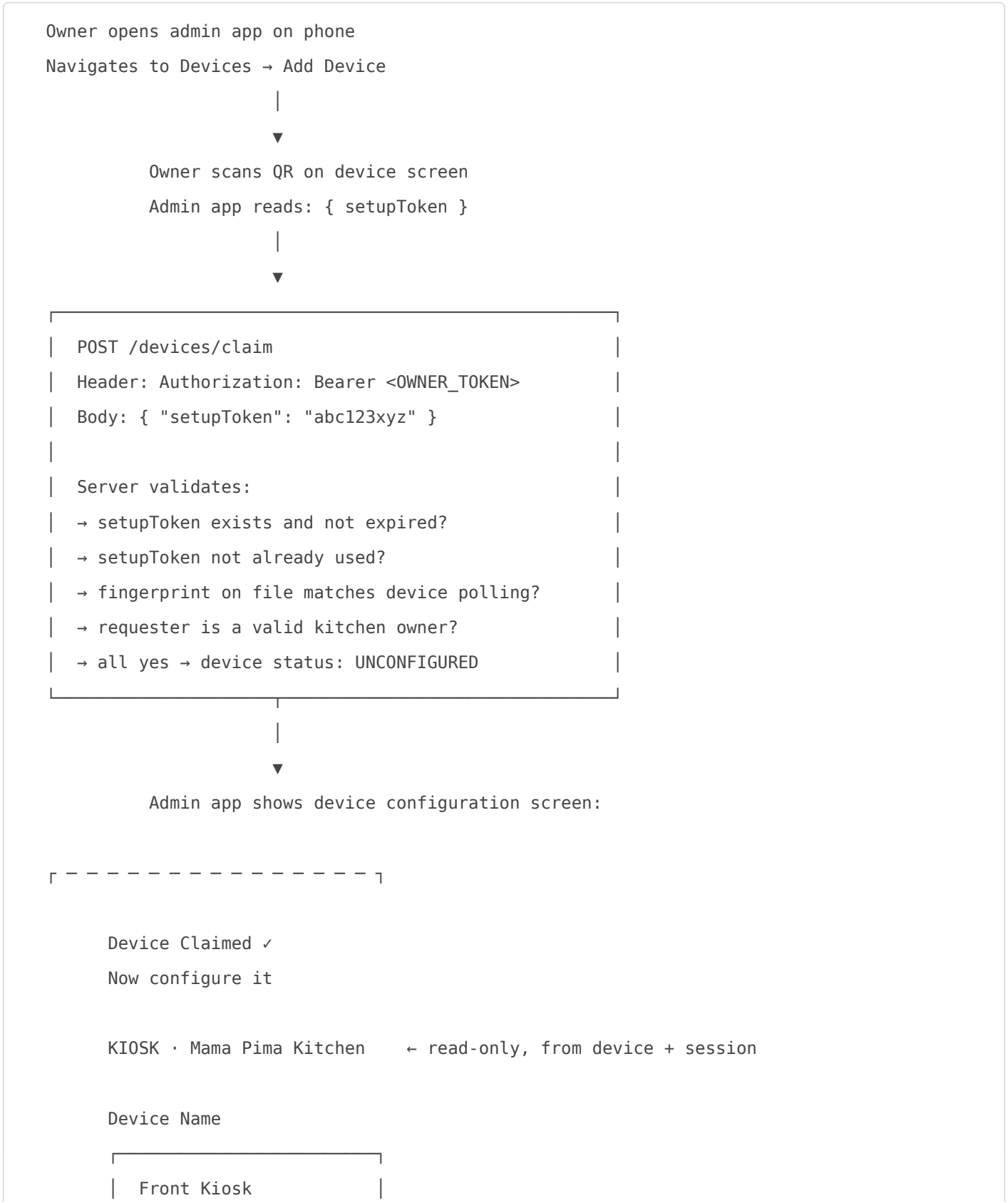
```
                    DEVICE_TOKEN    "Admin hasn't finished.
                    + configPayload  Try again in a moment."
                           |         [Try Again]
                           ▼
                    Device stores:
                    - DEVICE_TOKEN
                    - configPayload
                    Derives configHash
                    setupToken destroyed server-side
                           |
                           ▼
                    Navigate to PIN screen ✓
```

Incomplete Registration — Admin Abandons Mid-Flow

Three scenarios handled gracefully:

```
Scenario A — Admin never scanned (setupToken expired)
→ Device still on QR screen
→ Poll detects EXPIRED
→ Device auto-fetches new setupToken, re-renders QR
→ Admin can scan fresh anytime ✓

Scenario B — Admin scanned but never finished configuring, setupToken expired
→ Device on "Complete Setup" screen
→ User taps button next day
→ Server: setupToken expired → error returned
→ Device shows: "Setup session expired. Ask your admin to scan again."
             [Generate New QR]
→ Device returns to QR screen with fresh setupToken
→ Admin scans again, configures, done ✓

Scenario C — Admin fully configured, device never tapped "Complete Setup"
→ Device on "Complete Setup" screen
→ User taps button the next day
→ Server: device is CONFIGURED → returns DEVICE_TOKEN + configPayload
→ Works perfectly ✓
→ CONFIGURED state has no expiry — safe to complete anytime
```

Flow 2 — Owner Claims & Configures Device (Admin App)



```
      └──────────────────────────┘

      Permissions
      ──────────────────────────

      Dine In          [ ON  ]
      Pickup           [ ON  ]
      Delivery         [ OFF ]
      POS Access       [ OFF ]
      Reports          [ OFF ]
      Kitchen Display  [ ON  ]
      Store Access     [ OFF ]

      ┌──────────────────────┐
      │    Save & Activate   │
      └──────────────────────┘

└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
                    │
                    ▼
┌──────────────────────────────────────────────────────────┐
│  PUT /devices/{deviceId}/configure                       │
│  Header: Authorization: Bearer <OWNER_TOKEN>             │
│  Body: {                                                 │
│    "name": "Front Kiosk",                                │
│    "permissions": {                                      │
│      "allowDineIn": true,                                │
│      "allowPickup": true,                                │
│      "allowDelivery": false,                             │
│      "allowPOS": false,                                  │
│      "allowReports": false,                              │
│      "allowKitchenDisplay": true                         │
│    }                                                     │
│  }                                                       │
│                                                          │
│  Server:                                                 │
│  → kitchenId taken from owner session                    │
│  → deviceType taken from setupToken record               │
│  → generates DEVICE_TOKEN                                │
│  → builds configPayload                                  │
│  → status → ACTIVE                                       │
```

```
  │  → signals CONFIGURED on next poll                  │
  └─────────────────────────────────────────────────────┘
                          │
                          ▼
              Device polling detects CONFIGURED
              Fetches DEVICE_TOKEN + configPayload
              Stores locally, navigates to PIN screen ✓
```

Flow 3 — Device Boot (Already Registered)

```
  Device boots
      │
      ▼
  Check local storage
      │
  ┌───┴──┐
  │      │
NO TOKEN  TOKEN FOUND
  │       │
  ▼       ▼
Setup   GET /devices/{deviceId}/config
Flow    Header: X-Device-Token: <DEVICE_TOKEN>
            │
            ▼
        Response:
        {
          "deviceStatus": "ACTIVE",
          "configHash": "a3f9c2...",
          "config": { ...full configPayload... }
        }
            │
            ▼
        Device overwrites local configPayload
        Derives hash from fresh payload
            │
        ┌───┴──────┐
```

```
   |            |
ACTIVE       SUSPENDED / REVOKED
   |            |
   ▼            ▼
Show PIN     React accordingly
screen ✓     (lock / wipe)
```

Flow 4 — Staff Login (POS / STORE_TABLET)

```
Device on PIN screen
Staff enters PIN
    |
    ▼
POST /auth/staff/login
Header: X-Device-Token: <DEVICE_TOKEN>
Body: { "pin": "1234" }
    |
    ▼
Server:
→ identifies kitchen from DEVICE_TOKEN
→ checks deviceType — is staff auth required?
→ finds staff with matching PIN hash in that kitchen
→ validates PIN
→ issues STAFF_TOKEN (8hrs, device-bound)
→ fetches permissionsHash for this staff member
    |
    ▼
Response:
{
  "deviceStatus": "ACTIVE",
  "configHash": "a3f9c2...",
  "staffToken": "eyJ...",
  "permissionsHash": "b7d1e4..."
}
    |
```

```
    ▼
Device fetches staff permissions once:
GET /staff/me/permissions
Header: X-Device-Token + X-Staff-Token
    │
    ▼
Device caches permissions locally for session
Effective permissions = staff permissions ∩ device permissions
    │
    ▼
Staff sees their dashboard ✓
```

PIN Security

```
Wrong PIN attempt → increment counter on device record
5 wrong attempts  → device locked for 15 mins (server-enforced)
Lockout is device-scoped, not global
Correct PIN → counter resets
```

Staff Session Rules

```
One active STAFF_TOKEN per device at any time
New PIN login → previous STAFF_TOKEN immediately invalidated
Staff logs out → STAFF_TOKEN invalidated → back to PIN screen
Token expires after 8hrs → back to PIN screen
```

Flow 5 — Mid-Session Config & Permissions Sync

```
Every API response carries:
{
  "deviceStatus": "ACTIVE",
  "configHash": "a3f9c2...",
  "permissionsHash": "b7d1e4..."   ← only when staff session active
}
```

```
Device on every response:

Step 1 — Check deviceStatus
→ ACTIVE     → continue
→ SUSPENDED  → lock screen immediately
→ REVOKED    → wipe local storage → setup screen

Step 2 — Check configHash
→ matches local derived hash  → config still fresh
→ mismatch                    → pull fresh config
                                GET /devices/{deviceId}/config

Step 3 — Check permissionsHash (if staff session active)
→ matches cached hash  → permissions still fresh
→ mismatch             → pull fresh permissions
                         GET /staff/me/permissions
                         recalculate effective permissions
```

Flow 6 — Manual Refresh (Refresh Button on Device)

```
Staff or admin taps Refresh button on device
    |
    ▼
GET /devices/{deviceId}/config
Header: X-Device-Token: <DEVICE_TOKEN>
    |
    ▼
Same as boot config pull
Device reacts to whatever server returns
(ACTIVE / SUSPENDED / REVOKED)
    |
    ▼
Config updated if hash changed ✓
```

Useful when owner changes permissions on dashboard and staff need it reflected immediately without waiting for next request.

Flow 7 — Device Revocation (Owner Dashboard)

```
DEVICE_TOKEN invalidated immediately
All active STAFF_TOKENs on this device killed
    │
    ▼
Device makes next request (any request or refresh)
Response carries: { "deviceStatus": "REVOKED" }
    │
    ▼
Device:
→ deletes DEVICE_TOKEN from local storage
→ deletes configPayload from local storage
→ clears all active staff sessions locally
→ clears any cached order/cart data
→ resets to setup/QR screen ✓

Device owner can now re-register the device
fresh through the QR setup flow
```

Two Revocation Paths

Source	Endpoint	Confirmation Required
Owner dashboard	`PATCH /devices/{id}/revoke`	Dashboard UI confirmation
Device itself	`POST /devices/self-revoke`	Are you sure + kitchen name typed

Both lead to the same outcome — token dead, device wiped, setup screen shown.

Flow 7b — Self-Revoke From Device

Device has a revoke option buried in settings — not on the main PIN screen.

```
Staff/admin navigates to Settings → Revoke Device
    │
    ▼
  Step 1 — Are You Sure? dialog

  ┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐
```


⚠ Revoke This Device?
This will immediately wipe
all data from this device.
It must be re-registered
to be used again.
Yes, Continue
Cancel
(taps Yes, Continue)
Step 2 — Type Kitchen Name to Confirm
Confirm Revocation
Type your kitchen name
to confirm:
Mama Pima Kitchen
Revoke Device
← red, disabled until name matches
Cancel

```
Revoke button enabled only when typed name matches
kitchenName from local configPayload exactly
(local string comparison — no server call needed for the check)
    |
    ▼ (taps Revoke Device)
POST /devices/self-revoke
Header: X-Device-Token: <DEVICE_TOKEN>
Body: { "kitchenName": "Mama Pima Kitchen" }
    |
    ▼
Server validates:
→ DEVICE_TOKEN valid?
→ kitchenName matches kitchen on record for this device?
→ yes → marks device REVOKED
       → invalidates DEVICE_TOKEN
       → kills all active STAFF_TOKENs
    |
    ▼
Device:
→ wipes all local storage
→ resets to setup/QR screen ✓
```

Flow 8 — Suspension (Subscription Lapsed)

```
Kitchen subscription expires
Platform marks kitchen as SUSPENDED server-side
All devices belonging to this kitchen → status SUSPENDED
    |
    ▼
Device makes next request
Response: { "deviceStatus": "SUSPENDED" }
    |
    ▼
Device:
→ does NOT wipe local storage
→ locks screen
```

```
→ shows: "Kitchen subscription inactive. Contact your admin."

When kitchen pays and subscription restored:
→ Platform marks kitchen ACTIVE
→ All devices → ACTIVE
→ Next device request returns ACTIVE
→ Device unlocks automatically ✓
```

DEVICE_TOKEN and configPayload are preserved during suspension. Device resumes instantly when restored — no re-registration needed.

Flow 9 — Endpoint Access Control by Device Type

Server enforces device type restrictions on every request:

```
Request arrives
→ extract X-Device-Token
→ look up device record → read deviceType
→ check: is this deviceType permitted on this endpoint?
→ yes → proceed
→ no  → 403 Forbidden
```

Endpoint Permission Matrix

Endpoint	POS	STORE_TABLET	KIOSK	KITCHEN_DISPLAY
`GET /menu/public`	✅	✅	✅	✅
`POST /orders`	✅	✅	✅	✅
`GET /kitchen/display`	✅	✅	✅	✅
`POST /pos/cash-drawer`	✅	✅	✅	✅
`GET /reports`	✅	✅	✅	✅
`POST /kiosk/self-checkout`	✅	✅	✅	✅
`POST /auth/staff/login`	✅	✅	✅	✅

Request Structure Reference

Device-Only Request (KIOSK / KITCHEN_DISPLAY)

```
Headers:
  X-Device-Token: <DEVICE_TOKEN>
```

Device + Staff Request (POS / STORE_TABLET)

```
Headers:
  X-Device-Token: <DEVICE_TOKEN>
  X-Staff-Token:  <STAFF_TOKEN>

Server validates:
→ DEVICE_TOKEN valid and ACTIVE?
→ STAFF_TOKEN valid and not expired?
→ STAFF_TOKEN.deviceId === DEVICE_TOKEN.deviceId?
→ deviceType requires staff auth?
→ all yes → process request
```

Payload Shapes

DEVICE_TOKEN Payload (JWT)

```
{
  "deviceId": "dv_uuid",
  "kitchenId": "kt_uuid",
  "deviceType": "KIOSK"
}
```

STAFF_TOKEN Payload (JWT)

```
{
  "staffId": "st_uuid",
  "kitchenId": "kt_uuid",
  "deviceId": "dv_uuid",
  "expiresAt": "2026-05-05T18:00:00Z"
}
```

Permissions are **not** inside the token. They are fetched once per session and cached on device.

Staff Permissions Payload

```
{
  "permissionsHash": "b7d1e4...",
  "permissions": {
    "canViewOrders": true,
    "canManageOrders": true,
    "canViewReports": false,
    "canManageMenu": false,
    "canManageStaff": false,
    "canProcessRefunds": false
  }
}
```

Effective Permissions (Device ? Staff)

```
Device allowReports: false
Staff  canViewReports: true
Effective: CANNOT view reports on this device

Device allowPOS: true
Staff  canProcessRefunds: true
Effective: CAN process refunds on this device
```

Staff can never exceed what the device allows, regardless of their role.

Owner Dashboard — Device Management Screens

Screen 1 — Devices List

Screen 2 — Add Device (QR Scanner)

```
      QR code shown on the device

      ┌──────────────────────────┐
      │                          │
      │    [ Camera Viewfinder ]│
      │                          │
      │   ┌──┐           ┌──┐    │
      │   └──┘           └──┘    │
      │                          │
      │   ┌──┐           ┌──┐    │
      │   └──┘           └──┘    │
      └──────────────────────────┘

      Make sure the device is showing
      its setup QR code

└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

Screen 3 — Configure Device (After Scan)

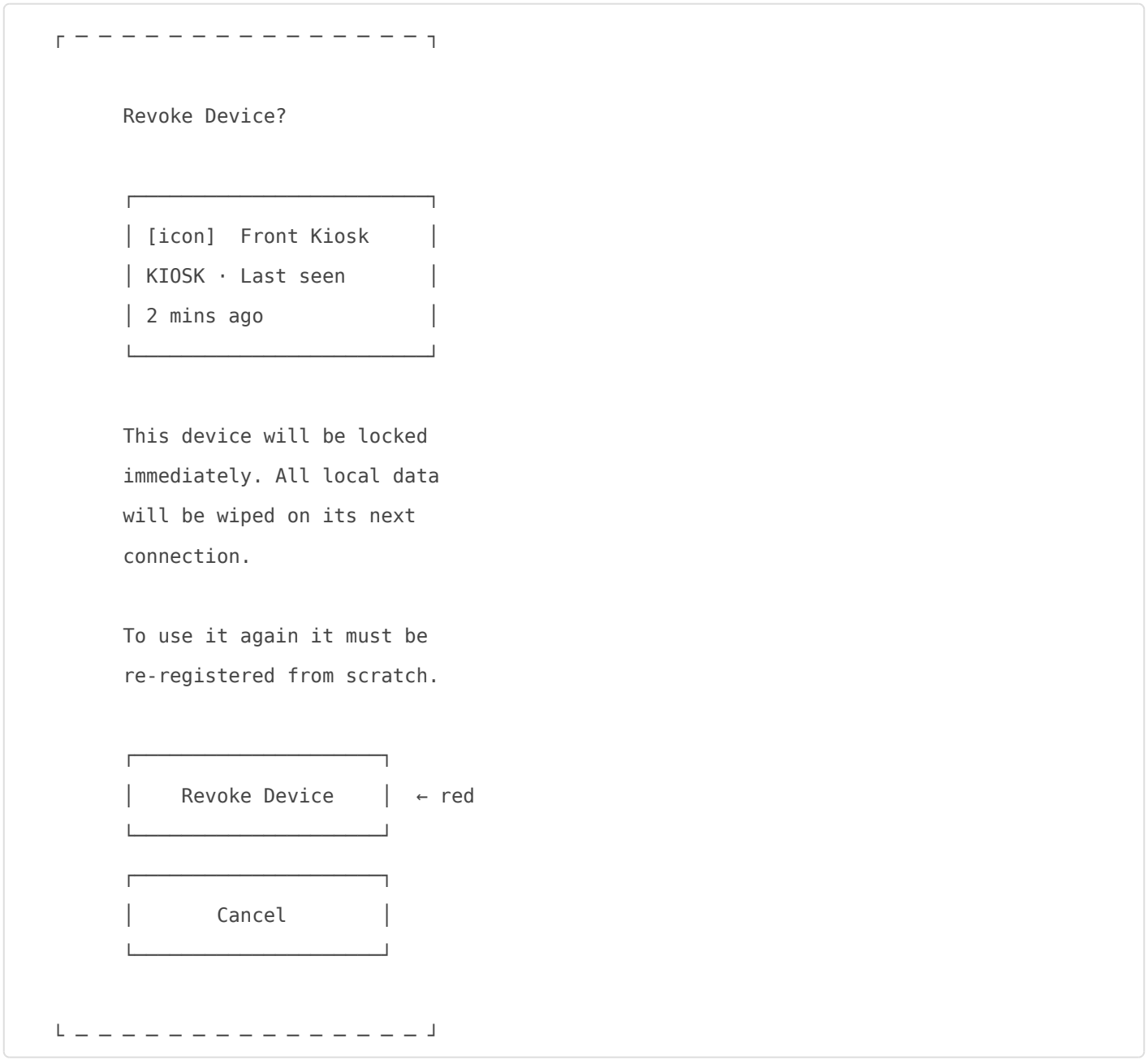
```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐

      Device Claimed ✓
      Now configure it

      KIOSK · Mama Pima Kitchen    ← read-only

      Device Name
      ┌──────────────────────────┐
      │  Front Kiosk             │
      └──────────────────────────┘

      Permissions
      ──────────────────────────

      Dine In          [ ON  ]
      Pickup           [ ON  ]
      Delivery         [ OFF ]
      POS Access       [ OFF ]
      Reports          [ OFF ]
```

```
      Kitchen Display  [ ON  ]
      Store Access     [ OFF ]

      ┌──────────────────────┐
      │   Save & Activate    │
      └──────────────────────┘

└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

Screen 4 — Device Detail

```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐

      ← Front Kiosk

      Status      □□ACTIVE
      Type        KIOSK
      Kitchen     Mama Pima Kitchen
      Last Seen   2 minutes ago
      Registered  1 May 2026

      Permissions
      ──────────────────────────

      Dine In          [ ON  ]
      Pickup           [ ON  ]
      Delivery         [ OFF ]
      POS Access       [ OFF ]
      Reports          [ OFF ]
      Kitchen Display  [ ON  ]

      ┌──────────────────────┐
      │   Edit Permissions   │
      └──────────────────────┘

      ┌──────────────────────┐
      │   Revoke Device      │  ← red, destructive
      └──────────────────────┘

└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

Screen 5 — Revoke Confirmation



Device Screens (Dotted)

Screen A — Authenticate Device (Unregistered)

```
        This device is not registered

        Ask your kitchen admin to
        scan the setup code

        ┌──────────────────────┐
        │   Set Up Device      │
        └──────────────────────┘
```

Screen B — QR Code Display

```
        [JikoXpress Logo]

        Scan this code with the
        JikoXpress admin app

        ┌──────────────────────┐
        │                      │
        │   ▄▄▄▄▄  ▄▄ ▄▄▄▄▄   │
        │   ▄▄ ▄▄ ▄▄▄▄ ▄▄ ▄▄   │
        │   ▄▄▄▄▄ ▄▄  ▄▄▄▄▄   │
        │     ▄▄  ▄▄ ▄▄       │
        │   ▄▄▄▄▄ ▄▄▄ ▄▄▄▄▄   │
        │                      │
        └──────────────────────┘

        Refreshes in  03:47

        Waiting for admin to scan...
```

QR refreshes every 5 minutes. Old setupToken is burned on each refresh. Polling runs every 5 seconds **for CLAIMED status only** — stops the moment admin scans.

Screen C — Admin Scanned (Complete Setup)

```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐

      [JikoXpress Logo]

      Code scanned ✓

      Your admin is configuring
      this device on their phone.

      Tap below once they are done:

      ┌──────────────────────┐
      │   Complete Setup     │
      └──────────────────────┘

      ← Back (generate new QR)

└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

No polling on this screen. User taps "Complete Setup" manually when admin signals done.

Screen C2 — Complete Setup — Session Expired

```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐

      [JikoXpress Logo]

      ⚠  Setup session expired

      Ask your admin to scan
      the device again.

      ┌──────────────────────┐
      │   Generate New QR    │
      └──────────────────────┘
```

```
└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

Screen C3 — Complete Setup — Admin Not Done Yet

```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐

      [JikoXpress Logo]

      Admin hasn't finished yet

      Ask your admin to complete
      the configuration on their
      phone then try again.

      ┌──────────────────────┐
      │      Try Again       │
      └──────────────────────┘

└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

Screen D — PIN Login (Registered Device)

```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐

      Mama Pima Kitchen
      Front Kiosk

      Enter your PIN

      ┌──┐ ┌──┐ ┌──┐ ┌──┐
      │ ● │ │ ● │ │  │ │  │
      └──┘ └──┘ └──┘ └──┘

      ┌──┐ ┌──┐ ┌──┐
      │ 1 │ │ 2 │ │ 3 │
      ├──┤ ├──┤ ├──┤
```

```
        | 4 | | 5 | | 6 |
        ├───┤ ├───┤ ├───┤
        | 7 | | 8 | | 9 |
        ├───┤ ├───┤ ├───┤
        | ⌫ | | 0 | | ✓ |
        └───┘ └───┘ └───┘

        [↻ Refresh]

└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

Screen E — Suspended

```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐

        [JikoXpress Logo]

        ⚠

        Kitchen subscription inactive

        This device has been
        suspended. Please contact
        your kitchen admin.

        ┌──────────────────────┐
        |      Refresh         |
        └──────────────────────┘

└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

Screen F — Revoked

```
┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐

        [JikoXpress Logo]

        
```

```
        Device access revoked

        This device has been
        deactivated by your admin.

        Contact your admin to
        re-register this device.

        ┌──────────────────────┐
        │   Set Up Device      │
        └──────────────────────┘

└ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
```

Endpoint Reference

Device Setup — No Auth

Method	Endpoint	Headers	Body	Returns
GET	`/devices/setup/token`	`X-Device-Fingerprint`	—	`setupToken`
GET	`/devices/setup/status`	`X-Device-Fingerprint`, `X-Setup-Token`	—	`{ status }`
GET	`/devices/setup/complete`	`X-Device-Fingerprint`, `X-Setup-Token`	—	`DEVICE_TOKEN + configPayload`

Device Claim & Config — Owner Auth

Method	Endpoint	Headers	Body	Returns
POST	`/devices/claim`	`Authorization: Bearer <OWNER_TOKEN>`	`{ setupToken }`	`{ deviceId, status: UNCONFIGURED }`
PUT	`/devices/{id}/configure`	`Authorization: Bearer <OWNER_TOKEN>`	name, permissions	`{ success }`
GET	`/devices`	`Authorization: Bearer <OWNER_TOKEN>`	—	All devices for owner
PATCH	`/devices/{id}/revoke`	`Authorization: Bearer <OWNER_TOKEN>`	—	`{ success }`
PATCH	`/devices/{id}/reactivate`	`Authorization: Bearer <OWNER_TOKEN>`	—	`{ success }`

Method	Endpoint	Headers	Body	Returns
PUT	`/devices/{id}/permissions`	`Authorization: Bearer <OWNER_TOKEN>`	permissions object	`{ success }`

Device Runtime — Device Auth

Method	Endpoint	Headers	Body	Returns
GET	`/devices/{id}/config`	`X-Device-Token`	—	`deviceStatus + configHash + configPayload`
POST	`/devices/self-revoke`	`X-Device-Token`	`{ kitchenName }`	`{ status: REVOKED }`

Staff Auth — Device Auth

Method	Endpoint	Headers	Body	Returns
POST	`/auth/staff/login`	`X-Device-Token`	`{ pin }`	`STAFF_TOKEN + permissionsHash`
GET	`/staff/me/permissions`	`X-Device-Token`, `X-Staff-Token`	—	`permissionsHash + permissions`
POST	`/auth/staff/logout`	`X-Device-Token`, `X-Staff-Token`	—	`{ success }`

Security Summary

Threat	Mitigation
Stranger scans QR	Claim requires valid OWNER_TOKEN — scan alone is useless
QR replay attack	setupToken expires in 5 mins, single use, burns on claim
Wrong device claims token	Fingerprint registered at token fetch — mismatch on claim → rejected
Stolen DEVICE_TOKEN	Owner revokes from dashboard → dead on next request
Accidental self-revoke on device	Two-step confirmation + kitchen name typed to confirm
Malicious self-revoke attempt	Server re-validates kitchenName against device record server-side
Staff PIN brute force	5 wrong attempts → 15 min lockout, device-scoped
Stolen STAFF_TOKEN	Device-bound — useless on any other device

Threat	Mitigation
Device deactivated mid-session	deviceStatus in every response → device reacts immediately
Subscription lapse	All kitchen devices suspended automatically — no data lost
Wrong device type hits endpoint	Server reads deviceType from record — 403 if mismatch
Config changed while device offline	Boot always pulls fresh config — guaranteed fresh start
Rooted / jailbroken device	App detects on boot and refuses to run
Token extracted from storage	OS keychain hardware-backed — not extractable even on rooted devices (with detection)

Secure Token Storage — Per Platform

DEVICE_TOKEN must never be stored in plaintext, localStorage, or unencrypted files. Every platform has a secure OS-managed storage mechanism. JikoXpress follows the same pattern used by Spotify, Slack, Discord, and 1Password.

Storage Mechanism Per Platform

Platform	Storage Mechanism	Security Level
Android	Android Keystore System	□□□□□ hardware-backed
iOS	Keychain + Secure Enclave	□□□□□ hardware-backed
Windows	DPAPI / Credential Manager	□□□□ strong
Mac	macOS Keychain	□□□□□ hardware on Apple Silicon
Linux (with keychain)	libsecret → GNOME Keyring / KWallet	□□□□ strong
Linux (headless / no keychain)	Encrypted file, machine-derived key	□□□ acceptable
Web	httpOnly cookie only	□□□ acceptable — never localStorage

Cross-Platform Desktop (Electron)

For any Electron-based desktop client, use `keytar` — the same library used by Spotify, Slack, and VS Code:

```
// Store
await keytar.setPassword('JikoXpress', 'deviceToken', token)
```

```
// Retrieve
await keytar.getPassword('JikoXpress', 'deviceToken')

// Delete (on revoke/wipe)
await keytar.deletePassword('JikoXpress', 'deviceToken')
```

`keytar` wraps the OS keychain on all three platforms — one API, three platforms handled transparently.

Linux Fallback (No Keychain Available)

```
Derive encryption key from:
  hash(machineId + username + appSalt)
→ encrypt DEVICE_TOKEN with derived key
→ store encrypted blob in ~/.config/jikoxpress/
→ not hardware-backed but significantly better than plaintext
```

This is the same approach VS Code uses on headless Linux.

What Is Never Acceptable

```
 Plain text files on filesystem
 localStorage or sessionStorage (web)
 Unencrypted local SQLite
 Hardcoded in config files
 Environment variables on disk
 Shared storage accessible by other apps
```

Root / Jailbreak Detection

On app boot, before loading any stored token:

```
Android → RootBeer library → detect root
iOS     → DTTJailbreakDetection → detect jailbreak
→ detected → refuse to run
→ show: "This device does not meet security requirements.
         Contact your admin."
```

Compromised OS → secure storage guarantees are void → app must not run.

Managed Device Enforcement

Since JikoXpress devices are business-owned and managed:

```
App enforces on boot:
→ screen lock must be enabled
→ if no screen lock → refuse to run
→ show: "Please enable screen lock to use JikoXpress"
```

Physical access without screen lock is the simplest attack vector. Eliminating it costs nothing.

Real-Time Architecture — WebSocket + STOMP

What Needs Real-Time

```
Kitchen devices (KDS, POS, STORE_TABLET):
→ New order arrived (from any channel)
→ Order status changed
→ Order ready for pickup/delivery

Customer app:
→ Order confirmed
→ Order being prepared
→ Driver assigned
→ Driver location (live map)
→ Order delivered

Driver app:
→ New delivery assigned
→ Route updates

Platform dashboard (super admin):
→ Live orders across all kitchens
```

```
→ Live driver positions
→ Alerts on stuck orders
```

Protocol Decision

Protocol	Order Events	Driver Location	Verdict
STOMP over WebSocket	✅ perfect fit	⚠️ works but heavy	Use for events
Raw WebSocket	✅ works	✅ perfect fit	Use for location
MQTT	❌ overkill	❌ strong but needs broker + separate auth per client	Not used

MQTT rejected — every mobile client (driver app, customer app) would need separate MQTT broker credentials, creating a second auth system alongside JikoXpress auth. Not worth it.

```
Final stack:
→ STOMP over WebSocket  → order events, notifications, status updates
→ Raw WebSocket         → driver location streaming (high frequency)
→ Auth for both         → existing DEVICE_TOKEN / user JWT in headers
→ No MQTT               → credential complexity not justified
```

Topic Design (STOMP)

```
/topic/kitchen/{kitchenId}/orders       → KDS, POS, STORE_TABLET
/topic/kitchen/{kitchenId}/order/{id}   → specific order status updates
/topic/order/{orderId}/tracking         → customer tracks their order
/topic/platform/dashboard               → super admin live view
```

Server assigns topic subscriptions based on `deviceType` — device cannot self-subscribe to unauthorized topics.

Raw WebSocket (Driver Location)

```
Driver app → /ws/location (raw WebSocket)
→ sends GPS coordinates every 3 seconds
→ socket server broadcasts to:
   /topic/order/{orderId}/tracking  → customer
   kitchen dashboard                → kitchen
   platform dashboard               → super admin
```

Scaling Architecture (Phased)

```
Phase 1 — v1 (now):
→ STOMP embedded directly in main Spring Boot
→ kitchen order events only
→ zero extra infrastructure

Phase 2 — delivery launches:
→ extract dedicated Socket Server (separate Spring Boot service)
→ RabbitMQ between main backend and socket server
→ main backend publishes events → RabbitMQ → socket server → devices
→ raw WebSocket for driver location on same socket server
→ STOMP for all other events

Phase 3 — scale:
→ socket server scales horizontally
→ RabbitMQ fans out across all socket server instances
→ device clients unchanged — same STOMP protocol
```

```
Phase 2+ architecture:


┌──────────────────┐         ┌──────────────┐        ┌──────────────────────┐
│   Main Backend   │─publish─▶   RabbitMQ   │─consume─▶   Socket Server     │
│  (Spring Boot)   │         │              │        │  STOMP → /ws/stomp   │
└──────────────────┘         └──────────────┘        │  Raw WS → /ws/location│
                                                     └──────────┬───────────┘
                                                                │
                                                ┌───────────────┼───────────────┐
                                                ▼               ▼               ▼
                                          Kitchen Devices   Customer App    Driver App
                                          KDS / POS / Tablet Order tracking Location push
```

Missed Message Recovery

STOMP does not retain messages. On reconnect each device fetches missed events:

```
KDS reconnects
→ GET /kitchen/orders?since={lastReceivedAt}&station=GRILL
```

```
POS reconnects
→ GET /kitchen/pos/orders?since={lastReceivedAt}

Customer app reconnects
→ GET /orders/{orderId}/status
```

No messages lost. Gap filled on reconnect via REST fallback.

File Structure — `kitchen_service/devices/`

Base package: `org.qbitspark.jikoxpresspro`

```
kitchen_service/
└── devices/
    │
    ├── device_mng/
    │   ├── entity/
    │   │   └── DeviceEntity.java
    │   ├── enums/
    │   │   ├── DeviceType.java
    │   │   └── DeviceStatus.java
    │   ├── payload/
    │   │   ├── DeviceSetupTokenResponse.java
    │   │   ├── DeviceSetupStatusResponse.java
    │   │   ├── DeviceClaimRequest.java
    │   │   ├── DeviceConfigureRequest.java
    │   │   ├── DeviceConfigPayload.java
    │   │   ├── DevicePermissionsPayload.java
    │   │   ├── DeviceResponse.java
    │   │   └── DeviceSummaryResponse.java
    │   ├── repo/
    │   │   └── DeviceRepo.java
    │   ├── service/
    │   │   ├── DeviceService.java
    │   │   └── impl/
    │   │       └── DeviceServiceImpl.java
```

```
    │   ├── controller/
    │   │   ├── DeviceSetupController.java       ← setup/claim/complete (no auth)
    │   │   └── DeviceManagementController.java  ← owner dashboard endpoints
    │   └── mapper/
    │       └── DeviceMapper.java
    │
    ├── device_auth/
    │   ├── entity/
    │   │   └── StaffSessionEntity.java
    │   ├── enums/
    │   │   └── StaffTokenStatus.java
    │   ├── payload/
    │   │   ├── StaffLoginRequest.java
    │   │   ├── StaffLoginResponse.java
    │   │   ├── StaffPermissionsPayload.java
    │   │   └── DeviceAuthResponse.java          ← wrapper with deviceStatus + configHash
    │   ├── repo/
    │   │   └── StaffSessionRepo.java
    │   ├── service/
    │   │   ├── DeviceAuthService.java
    │   │   └── impl/
    │   │       └── DeviceAuthServiceImpl.java
    │   └── controller/
    │       └── DeviceAuthController.java
    │
    └── device_config/
        ├── payload/
        │   └── DeviceRuntimeResponse.java       ← envelope every response carries
        └── service/
            ├── DeviceConfigService.java
            └── impl/
                └── DeviceConfigServiceImpl.java
```

Package Responsibilities

```
device_mng    → device lifecycle — setup, claim, configure, revoke, list
device_auth   → staff PIN login, session management, permissions
device_config → shared runtime envelope (deviceStatus + configHash)
                also handles boot config pull
```

`DeviceAuthResponse` is the shared wrapper every device endpoint returns — not just auth endpoints. It wraps the endpoint `data` alongside `deviceStatus` and `configHash`.

Future Scope — KDS Advanced Configuration

> ⚠ This section is documented for memory and future planning. Not part of v1 build scope.

Multiple KDS devices are fully supported by the current auth design — each KDS is a separate registered device with its own `DEVICE_TOKEN` and staff auth. What changes in future scope is the **configuration depth** of what each KDS shows and how it behaves.

KDS Station Assignment (v1 — Basic)

Currently `deviceName` (e.g. "Grill Station") is the only station identifier. Enough for v1.

KDS Advanced Config (Future)

Each KDS will have a `kdsConfig` block inside its configPayload covering:

What to show:

```
station          → GRILL / FRYER / COLD / DRINKS / BAKERY / EXPO / PASS
showByCategories → ["MAINS", "STARTERS"] — filter by menu category
showByFulfillment→ DINE_IN / DELIVERY / PICKUP / ALL
showByChannel    → ALL or specific channels (App, POS, Kiosk, WhatsApp)
```

Display algorithm:

```
sortAlgorithm    → FIFO / PRIORITY / BY_TABLE / FULFILLMENT_FIRST
gridColumns      → 4 / 6 / 8 (how many orders visible at once)
cardDetail       → COMPACT / FULL (item name only vs full modifiers + notes)
fontSize         → SMALL / MEDIUM / LARGE
```

Timing & alerts:

```
alertYellowAfterMins → order turns yellow after X mins
alertRedAfterMins    → order turns red after Y mins
soundOnNewOrder      → true / false
soundOnCritical      → true / false
```

Station coordination:

```
expoMode                  → this KDS sees ALL stations, coordinates completion
showSiblingStationStatus  → show mini progress of other stations on same order
autoBumpOnAllStationsDone → auto-move order to READY when all stations done
```

Future Config Payload Shape (KDS)

```
{
  "deviceType": "KITCHEN_DISPLAY",
  "deviceName": "Grill Station",
  "station": "GRILL",
  "kdsConfig": {
    "showByCategories": ["MAINS", "STARTERS"],
    "showByFulfillment": ["DINE_IN", "PICKUP"],
    "showByChannel": ["ALL"],
    "sortAlgorithm": "FIFO",
    "gridColumns": 4,
    "cardDetail": "FULL",
    "fontSize": "MEDIUM",
    "alertYellowAfterMins": 8,
    "alertRedAfterMins": 15,
    "soundOnNewOrder": true,
    "soundOnCritical": true,
    "expoMode": false,
    "showSiblingStationStatus": true,
    "autoBumpOnAllStationsDone": false
  }
}
```

All of this will live inside configPayload — hash-monitored, server-controlled, no app update needed when owner changes KDS settings from dashboard.

Real World KDS Setups

```
Small kitchen (1 KDS)
→ One screen, shows everything, chef manages all

Medium kitchen (2-3 KDS)
→ Grill Station  → grill items only
→ Fryer Station  → fryer items only
→ Cold Station   → salads, desserts, drinks

Large kitchen (4+ KDS)
→ All of above plus:
→ Expo Screen    → head chef sees ALL orders, coordinates
→ Pass Screen    → confirms order complete before handoff to runner
```
