

QR Ordering – Amendment: Mobile App & Print Flow (KOT / Invoice)

Status: Draft v0.1 – for review Scope: Staff mobile app (iOS/Android), guest web ordering, push-triggered printing Out of scope: Aegis internals (existing order management system, already integrated)

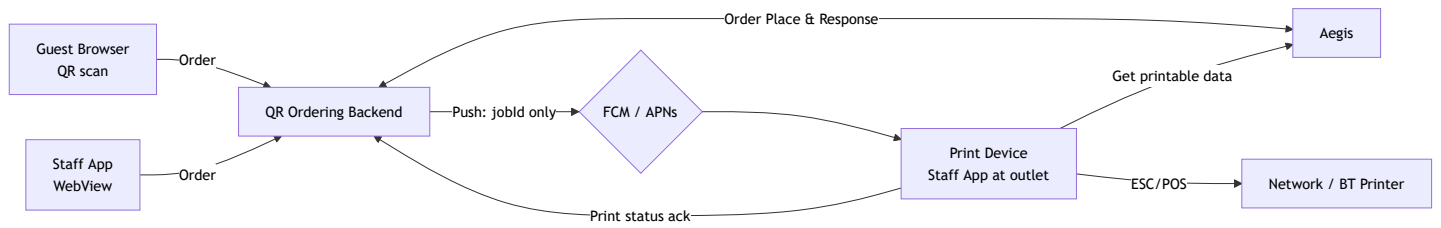
1. Summary

#	Change	Notes
1	Guest scans table/outlet QR and places Dine-in or Takeaway order	Phone camera → browser (no app install)
2	Staff can place the same orders on behalf of a guest	Same ordering web app, inside the staff app
3	iOS / Android app with a WebView that loads the same web app	Staff use the app; guests use camera + browser
4	KOT and Invoice printing from the app to Network (LAN) / Bluetooth printers	Triggered by push notification

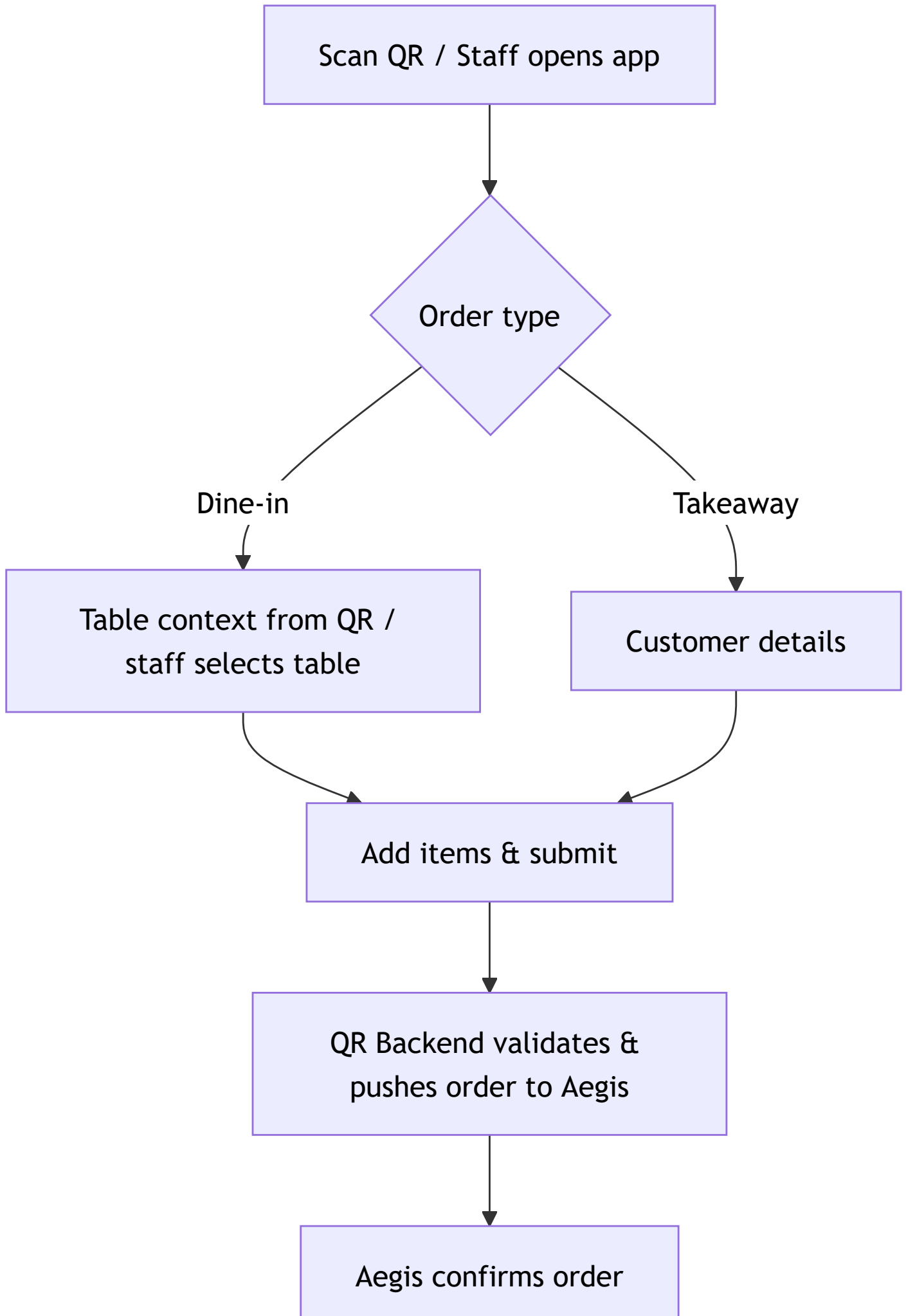
2. Actors & Components

Component	Responsibility
Guest Browser	Scan QR, order via web app
Staff App (iOS/Android)	WebView hosting the web app + native print layer + push receiver
QR Ordering Backend	Receives orders, forwards to Aegis, resolves print device(s), sends push, tracks print jobs
Aegis	Order management system. Source of truth for order data and printable data (KOT & Invoice)
FCM / APNs	Push delivery to Android / iOS
Printer	Network (ESC/POS over TCP 9100) or Bluetooth thermal printer

3. High-Level Architecture



4. Ordering Flow (Context)



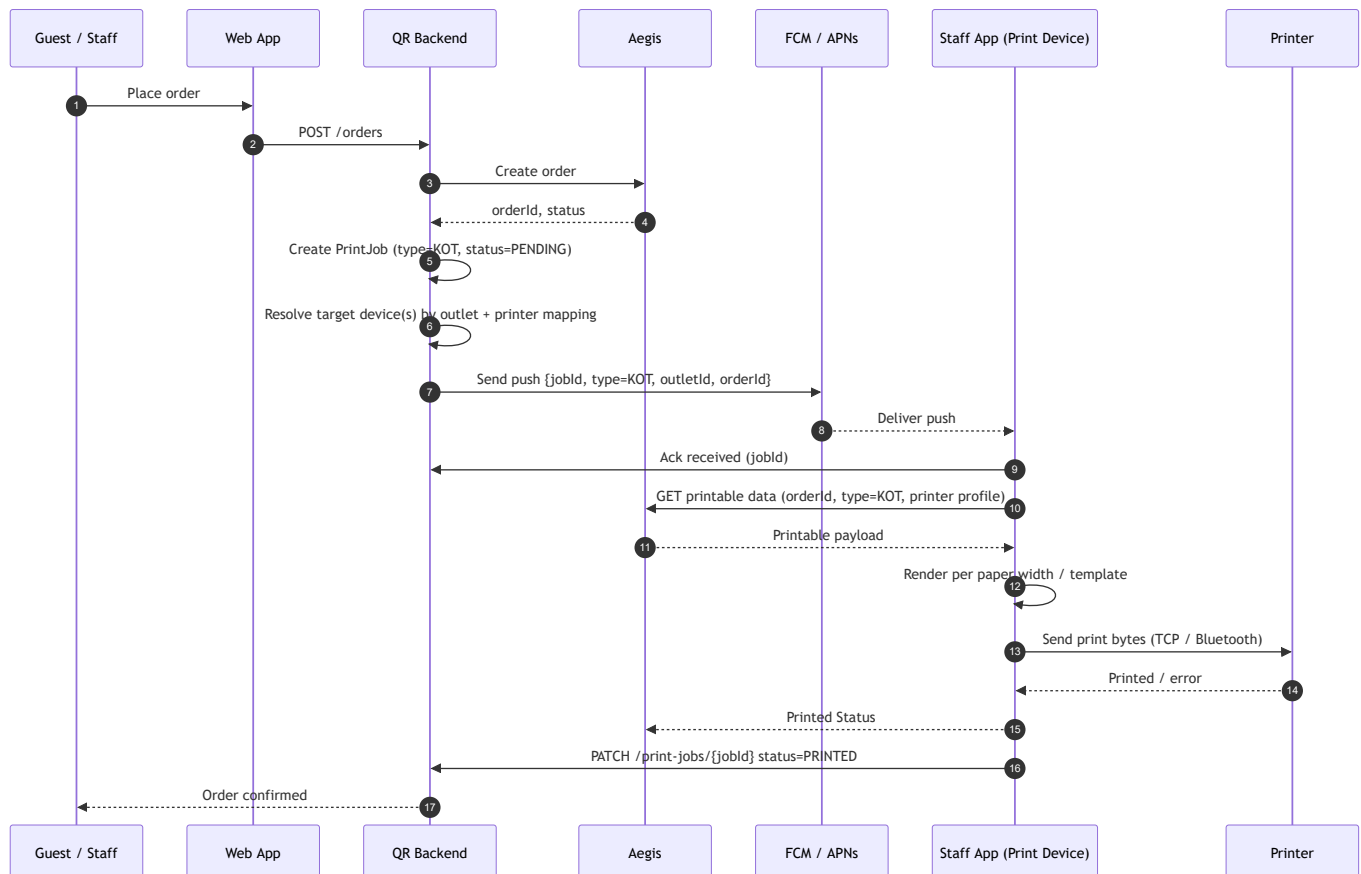
Print Trigger - see section 5

5. Print Flow

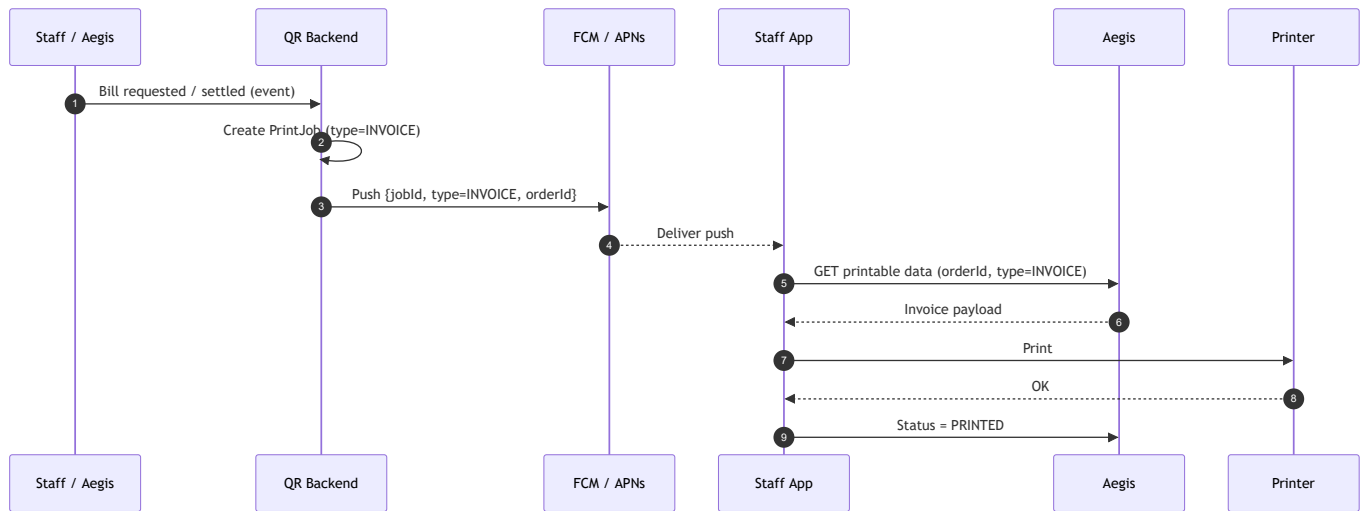
5.1 Rules

1. When an order is punched (guest / staff / Aegis), the **QR backend** sends a push notification to the correct device(s) in that outlet.
2. Android → **FCM**. iOS → **APNs**.
3. On receiving the push, the app calls the **Aegis API** to fetch the printable data.
4. Aegis returns the data; the device prints it.
5. **Paper size, printer type and design are dynamic** (not hardcoded in the app).
6. Two formats: **KOT** and **Invoice**.
7. Printable data for both formats always comes from the Aegis API, **called after** the push is received.
8. The push carries **only identifiers** (no order/PII data).
9. After printing, it sends printed status to **Aegis API**

5.2 Main Sequence – KOT



5.3 Main Sequence – Invoice



6. Push Payload (Proposed)

```

{
  "jobId": "pj_8f3a...",
  "type": "KOT",
  "outletId": "OUT_101",
  "orderId": "ORD_55021",
  "printerId": "PRN_KITCHEN_1",
  "createdAt": "2026-10-07T12:30:00Z"
}
  
```

- Android: **FCM data message**, priority: high .
- iOS: **APNs** with content-available: 1 (see risk R1 – likely needs a visible alert as well).
- No customer data or items in the push.

7. Printable Data – Dynamic Design

Aegis returns the printable data; the device only renders it. Recommended contract:

Request

```
GET /print/data?orderId=ORD_55021&type=KOT&paperWidth=80&printerId=PRN_KITCHEN_1
```

Response (structured, preferred)

```

{
  "type": "KOT",
  "paper": { "widthMm": 80, "charsPerLine": 48, "codepage": "UTF-8" },
  "blocks": [
    { "t": "text", "v": "KOT #123", "align": "center", "size": "large", "bold": true },
    { "t": "kv", "k": "Table", "v": "T-12" },
    { "t": "line" },
    { "t": "item", "qty": 2, "name": "Chicken Momo", "mods": ["Extra spicy"] },
    { "t": "qr", "v": "https://..." },
    { "t": "cut" }
  ]
}

```

Aspect	Approach
Paper size	58 mm / 80 mm configured per printer profile; sent as a request param and/or defined in Aegis
Printer type	Per-printer profile: connection (LAN IP:port / BT MAC), protocol (ESC/POS), codepage
Design	Aegis owns the template. App renders blocks → ESC/POS bytes
Non-Latin text	If printer lacks the font (e.g. Devanagari), render as bitmap/raster
Formats	KOT and INVOICE – same renderer, different payload

Open item: Aegis returns (a) structured blocks, (b) raw ESC/POS bytes, or (c) an image. See Question Q3.

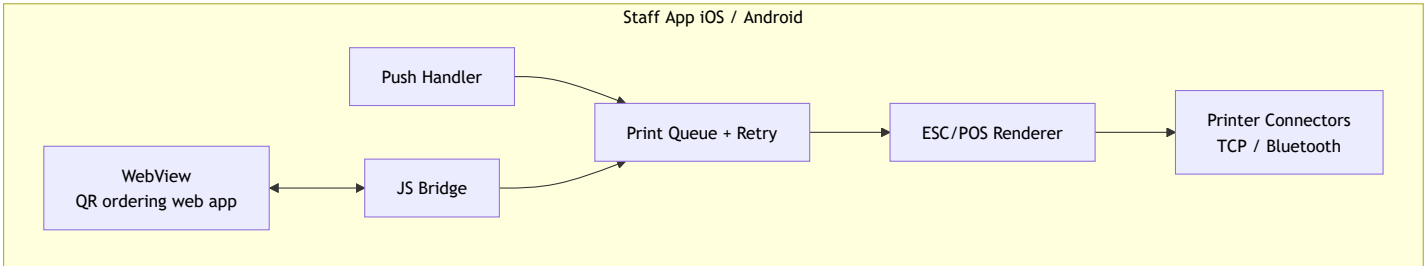
8. Device & Printer Registry

Backend needs to know *which device prints what*.

Table	Fields
print_devices	deviceId, outletId, platform, fcm/apns token, appVersion, lastSeen, isPrintStation
printers	printerId, outletId, name, connection (LAN/BT), address, paperWidth, role (KITCHEN / BAR / BILLING)
device_printer_map	deviceId ↔ printerId (which device can reach which printer)
print_jobs	jobId, orderId, type, printerId, targetDeviceId, status, attempts, timestamps, error

App registration flow: login → select outlet → register push token → (if print station) discover & save printers → heartbeat.

9. App Architecture (WebView Shell)



- The web app can also trigger a **manual reprint** via the JS bridge.
- Guests never use the app: the browser version is the same web app **without** the bridge.

10. Risks & Design Concerns

ID	Risk	Mitigation
R1	iOS silent push is not reliable. APNs may throttle or drop it, and it will not wake an app the user force-quit. KOT is time-critical.	Use a visible alert push + Notification Service Extension where possible; keep a WebSocket/SSE or short-poll fallback while the app is open; recommend a dedicated, always-open print tablet
R2	Android Doze / OEM battery killers can delay FCM.	High-priority data messages, foreground service on print station, battery-optimization exemption
R3	Duplicate prints if several devices in an outlet receive the push.	One designated print station per printer, or a "claim job" API (first device wins)
R4	Printed twice on push retry.	Idempotent <code>jobId</code> ; device ignores already-PRINTED jobs
R5	Aegis credentials on the device.	Short-lived token issued by QR backend, or QR backend proxies the Aegis call
R6	Printer offline / out of paper.	Local queue, retry with backoff, visible print-queue screen, manual reprint
R7	Bluetooth on iOS requires BLE-compatible printers (classic BT SPP is not supported on iOS).	Use LAN printers on iOS, or confirm BLE printer models
R8	Push payload data leakage.	IDs only; fetch data over HTTPS