# QR→Social Share Kiosk — Architecture & Step‑by‑Step Build Plan (Malaysia)

This document gives you a production‑ready blueprint: architecture, tech flow, platform constraints, and step‑by‑step tasks from frontend to backend. It assumes the goal is for **customers to scan a QR at your counter, choose a social app (XHS, Facebook, Instagram, WhatsApp, TikTok, etc.), then post content you’ve prepared**. Users may edit the caption and download the image before posting.

Key reality: **Most platforms will not allow truly automatic posting to a customer’s personal account.** You must use native **Share/Intent/Deep‑link flows**, and provide **Copy Caption** + **Download Image** as fallbacks. Posting to **your own brand Pages** (e.g., Facebook Page) can be automated via API, but that’s a different use case.

## 1) Customer Journey (Happy Path)

flowchart TD  
 A[Customer scans QR at counter] --> B[Opens landing page: share.yourbrand.my/?c=<campaign>&u=<scan\_id>]  
 B --> C[Shows prepared content: hero image + default caption]  
 C --> D[Customer may: Edit caption | Download image]  
 D --> E[Select app: XHS | Instagram | Facebook | WhatsApp | TikTok]  
 E --> F{Platform capability}  
 F -- Supports Web Share / Intent --> G[Open app via share sheet with media]  
 F -- No direct share --> H[Open app (deep link) + paste caption + select downloaded image]  
 G --> I[Customer taps Post]  
 H --> I[Customer taps Post]  
 I --> J[Thank‑you screen + optional reward QR/coupon]

## 2) High‑Level System Architecture

graph LR  
 subgraph Device[Customer Phone]  
 PWA[Landing Page (PWA)] -- Web Share API / Deep Links --> Apps[Installed Social Apps]  
 end  
  
 subgraph Cloud[Your Cloud]  
 CDN[(CDN)] --- S3[(Object Storage: images, OG thumbnails)]  
 API[(Backend API)] --- DB[(Database)]  
 CMS[Content CMS/Admin] --- API  
 QR[Dynamic QR Generator] --> API  
 end  
  
 Staff[Staff/Admin] --> CMS  
 PWA <-- content JSON / signed media URLs --> CDN  
 API --> Logs[(Analytics/Logs)]

**Suggested stack** - **Frontend**: React/Next.js (PWA), Tailwind; Clipboard API + Web Share API (v2 if available). - **Backend**: Laravel / Node.js (Express/NestJS). Rate limiting + signed URLs. - **Storage/CDN**: S3‑compatible (e.g., AWS S3) + CloudFront/Cloudflare. - **DB**: MySQL/PostgreSQL. - **Auth (Admin)**: Email+OTP or SSO; role‑based permissions. - **Analytics**: Server logs + UTM + optional GA4/Matomo.

## 3) End‑to‑End Technical Flow (Sequence)

sequenceDiagram  
 participant User as Customer  
 participant PWA as Landing Page (PWA)  
 participant API as Backend API  
 participant CDN as CDN/S3  
 participant CMS as Admin/CMS  
  
 CMS->>API: Create Campaign (caption variants, assets)  
 API->>CDN: Upload assets (images/videos) with cache headers  
 API->>API: Generate dynamic QR: /qr?c={campaign\_id}&loc={store\_id}  
  
 User->>PWA: Scan QR and open URL  
 PWA->>API: GET /api/campaigns/{campaign\_id}?scan={scan\_id}  
 API-->>PWA: JSON (caption, asset URLs, share options)  
 PWA->>CDN: Fetch image/og asset via signed or public URL  
 User->>PWA: Edit caption / Download image  
 User->>PWA: Tap “Share to Instagram / XHS / Facebook / WhatsApp / TikTok”  
 PWA->>User: Trigger Web Share / Deep Link / Copy Caption  
 PWA->>API: POST /api/events (share\_clicked, app=instagram)  
 User->>Apps: Completes post manually (one tap/paste)  
 PWA->>API: POST /api/events (share\_completed? optional via return CTA)

## 4) Platform Capabilities & UX Design

| Platform | What works well | What you **cannot** do | Recommended UX |
| --- | --- | --- | --- |
| **Instagram** | Share to **Stories** via OS share (image) or app intents. | Prefill caption for **Feed** posts; true auto‑post to personal accounts. | **Download image** → button to **Open Instagram** (deep link) → user selects image; **Copy Caption** button shown. For Stories, use Web Share if supported; otherwise show instructions. |
| **Facebook** | Share a **link** with OG image via share dialog; can open app. | Prefill user’s message text via URL params (blocked). Sharing raw image without user action. | Host a landing link (with OG image/description). Provide **Copy Caption**. |
| **XiaoHongShu (小红书)** | No public consumer posting API. | True auto‑post to user’s account. | **Download image** + **Copy Caption** + **Open XHS** (attempt deep link; fallback to app store). |
| **WhatsApp** | Text share via URL; media via OS share sheet. | Directly post to Status without user interaction. | Use **Web Share API** to attach image+caption when supported; else **Copy Caption** + Download. |
| **TikTok** | App accepts shares via OS share; users must confirm. | Prefill caption reliably via web across all OS versions; true auto‑post. | Use **Web Share** to pass the media; else **download + open app** pattern. |

Golden rule: Always provide **both** (1) a one‑tap share if supported on that device, and (2) a **clear fallback**: “Copy caption → Download image → Open app”.

## 5) Frontend (PWA) Spec

**Routes** - /s?c=<campaign\_id>&u=<scan\_id> — share page (minimal UI, fast) - /t/<scan\_id> — thank‑you/reward page (optional)

**UI Blocks** 1. **Hero**: Image preview (pinch‑zoom), file size label, download button. 2. **Caption Editor**: Textarea showing default caption; **Copy** button; character counter; hashtag chips. 3. **App Picker**: Big buttons (Instagram, XHS, Facebook, WhatsApp, TikTok). Detect platform (iOS/Android) to choose best method. 4. **Help Sheet**: 3‑step instructions for each app (collapsible). 5. **Privacy Notice**: “We do not collect your personal data—only anonymous share events.”

**Key Functions (pseudocode)**

async function shareMedia(imageUrl: string, caption: string) {  
 // Try Web Share with files first  
 if (navigator.canShare) {  
 const blob = await fetch(imageUrl).then(r => r.blob());  
 const file = new File([blob], 'post.jpg', { type: blob.type });  
 const data: ShareData = { files: [file], text: caption };  
 if (navigator.canShare(data)) {  
 await navigator.share(data); // Opens native share sheet  
 return { method: 'webshare' };  
 }  
 }  
 // Fallbacks  
 await navigator.clipboard.writeText(caption);  
 // Trigger image download (anchor with download attr)  
 const a = document.createElement('a');  
 a.href = imageUrl; a.download = 'post.jpg'; document.body.appendChild(a); a.click(); a.remove();  
 return { method: 'fallback' };  
}

**Deep‑link helpers (examples)** - **Facebook share dialog (link)**: https://www.facebook.com/sharer/sharer.php?u=<encodedLandingUrl> - **WhatsApp text**: https://api.whatsapp.com/send?text=<encodedCaption+link> - **Open app attempts**: try app scheme (if known) then fallback to store; always show instructions.

**Performance** - Preload hero image; compress to < 1 MB; WebP/AVIF with JPEG fallback. - PWA install banner for repeat users (staff testing).

## 6) Backend API Spec

**Endpoints** - POST /api/campaigns (admin) — create campaign; payload includes caption variants, tags, assets. - GET /api/campaigns/{id} — returns public share payload: { imageUrlSigned, captions: {default, xhs, ig, fb}, maxChars, hashtags, tracking }. - POST /api/events — logs events (scan\_opened, share\_clicked, copy\_caption, download\_image, webshare\_success, fallback\_flow). - GET /qr?c=<campaign\_id>&loc=<store\_id> — returns a **PNG/SVG QR** (server‑generated) pointing to /s?c=...&u=<uuid>.

**Security** - Signed URLs for media (time‑bound); or public+cache with randomized filenames. - Rate‑limit events; validate origin. - CORS locked to your domain.

**Data Model (simplified)**

Campaign(id, name, start\_at, end\_at, status)  
Asset(id, campaign\_id, url, mime, width, height, size)  
Caption(id, campaign\_id, platform, text, max\_chars)  
Scan(id, campaign\_id, store\_id, created\_at)  
Event(id, scan\_id, type, platform, meta\_json, created\_at)  
Store(id, code, address)  
User(id, role, email, password\_hash)

## 7) CMS/Admin Requirements

* Create/edit campaigns with **live preview** of share page.
* Upload **1:1, 4:5, 9:16** variants; auto‑generate crops.
* Caption templates per platform; hashtag library; **per‑platform char counters**.
* Toggle: **Allow customer to edit caption?** (on/off).
* Generate and download **counter QR** for printing (with store/location tag).
* Analytics dashboard: scans, shares by platform, completion rate, top stores, export CSV.

## 8) Analytics & Tracking

* Add ?src=qr&store=<code>&c=<campaign\_id>&u=<scan\_id> to landing.
* Fire POST /api/events on: page load, copy, download, share click, share success (if Web Share returns), open app.
* Optional: GA4 event mirroring (without personal identifiers).

## 9) Legal/Compliance (Malaysia)

* Show concise PDPA notice; don’t collect personal identifiers by default.
* Only store aggregate events; if running rewards, gather minimal info (e.g., phone/email) with explicit consent.
* Provide Terms/Privacy links on landing.

## 10) Step‑by‑Step Build Plan

**Phase 0 — Foundations (Day 1‑2)** 1. Register domain yourbrand.my, enable HTTPS (Let’s Encrypt/Cloudflare). 2. Create S3 bucket + CDN; set cache policies; CORS for your domain.

**Phase 1 — Backend (Day 3‑6)** 3. Scaffold API (Laravel/Node). Add Campaign, Asset, Caption, Scan, Event models. 4. Implement POST /api/campaigns, GET /api/campaigns/{id}, POST /api/events. 5. Add signed URL helper for asset downloads; upload pipeline with image compression. 6. Build QR generator endpoint that embeds scan\_id + store\_id.

**Phase 2 — Frontend PWA (Day 7‑10)** 7. Build /s page: fetch campaign JSON → render hero, caption editor, app picker, copy/download buttons. 8. Implement **Web Share** first, with file support; log method used. 9. Implement fallbacks: clipboard copy, image download, per‑app instruction sheets; attempt deep links where safe.

**Phase 3 — CMS/Admin (Day 11‑13)** 10. Admin UI: create campaign, upload images, write captions per platform, preview share page. 11. Generate printable QR posters (PDF export with store code, campaign name, expiry date).

**Phase 4 — Analytics & QA (Day 14‑15)** 12. Events dashboard; funnel (Scans → Opens → Shares). Device/OS breakdown. 13. QA matrix: iOS Safari/Chrome, Android Chrome, low network; test each platform flow.

**Launch**: Print QR posters, brief staff (30‑sec talk track), monitor dashboard.

## 11) Edge Cases & Fallbacks

* **iOS older versions** without file sharing: force fallback flow (copy+download+open app).
* **No app installed**: detect failure → show App Store/Play link.
* **Image too large**: serve smaller variant automatically by UA detection.
* **Offline after scan**: PWA caches assets with Workbox so image & page still load.

## 12) Optional Enhancements

* **Rewards/Referral**: after sharing, show a thank‑you page with a rotating coupon QR.
* **A/B Testing**: Serve different captions or image crops by scan\_id.
* **Multi‑language**: auto‑detect device language (EN/中文) with toggle.
* **Device‑specific assets**: portrait 9:16 for Stories, square for feed.
* **Kiosk Mode**: add a small animated guide and progress steps.

## 13) Minimal Code Snippets (Reference)

**Copy Caption + Toast**

async function copyCaption(text: string){  
 try { await navigator.clipboard.writeText(text); showToast('Caption copied'); }  
 catch { showToast('Copy failed — long‑press to copy'); }  
}

**Attempt Facebook Link Share**

function openFacebookShare(landingUrl: string){  
 const url = `https://www.facebook.com/sharer/sharer.php?u=${encodeURIComponent(landingUrl)}`;  
 window.location.href = url;  
}

**Print‑ready QR Poster** - Generate PDF server‑side with campaign name, store code, expiry, tracking QR. - Include short URL and backup numeric code for manual entry.

## 14) What You Need to Prepare (Checklist)

* Brand domain + SSL
* S3/CDN bucket
* Backend app (Laravel/Node) + DB
* Admin/CMS (campaigns, captions, assets)
* PWA share page with Web Share + fallbacks
* QR poster template (PDF)
* Analytics dashboard
* PDPA‑friendly privacy text

**Deliverable summary**: With this plan, you can ship a compliant, fast QR→Share kiosk that lets customers **edit captions**, **download images**, and **post** across XHS/Instagram/Facebook/WhatsApp/TikTok with the least friction allowed by each platform.