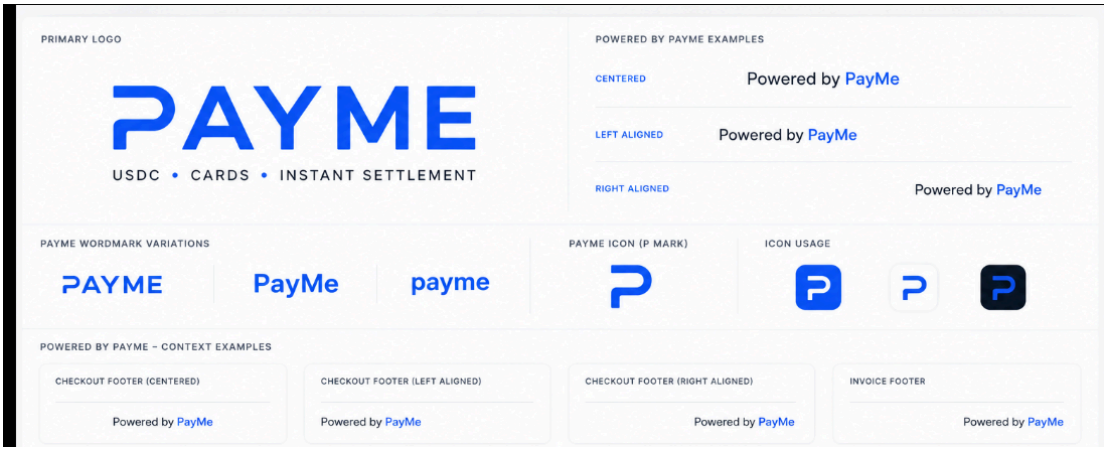
PAYME PRO

Full Code + Marketing Audit

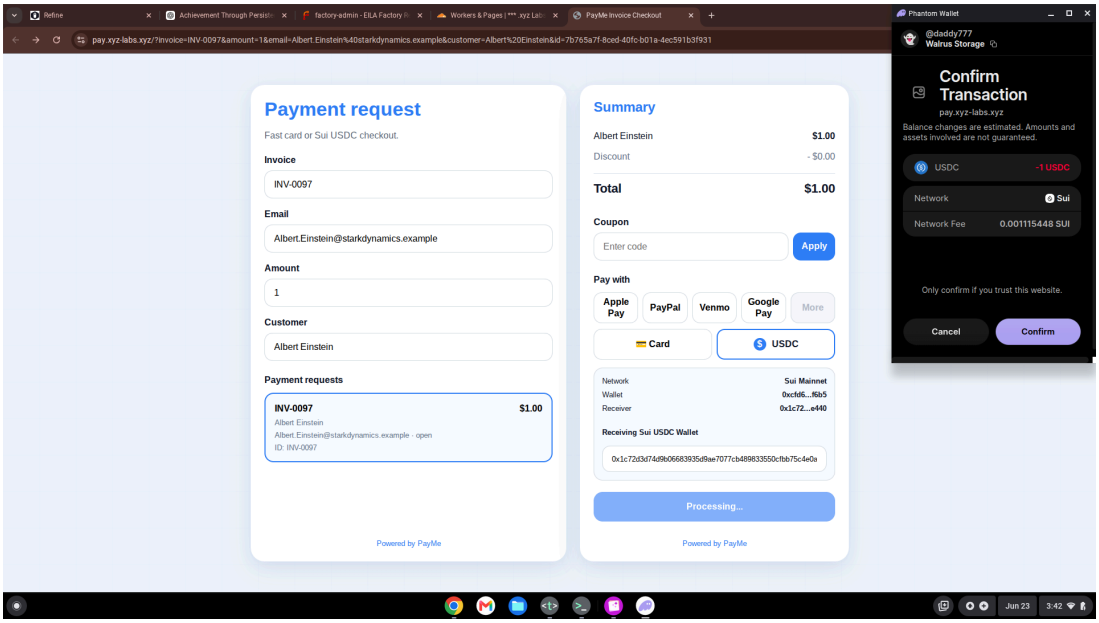
USDC • Cards • Instant Settlement

Scope: PayMePRO-Production-main repository, supplied production screenshots, checkout flow evidence, Cloudflare deployment references, and public-facing positioning assets.

Purpose: provide an investor-readable but technically grounded audit that explains what PayMe Pro does, what is deployed, what is production-ready, and what should be hardened before broader release.



PayMe wordmark and footer treatment exploration; shows final direction for minimal powered-by usage.



Live checkout screen with invoice/customer fields, card/USDC selection, wallet interaction, and processing state.

1. Audit Basis

Audit Item	Observed Evidence
Repository package	PayMePRO-Production-main.zip
Files inspected	108
Approximate code / content lines	18,193
Primary stack	React 18, Vite, wagmi, viem, Stripe API handlers, local checkout engine, Light CRM module
Payment rails observed	Card checkout, Stripe session generation, Base USDC transfer workflow, USDC transaction verification endpoint
Deployment evidence supplied	Cloudflare Workers service, KV binding, custom domain, PayMe checkout UI, wallet confirmation, Stripe comparison screenshot

2. Executive Summary

PayMe Pro is a payment request and checkout system that combines familiar checkout UX with stablecoin settlement workflows. The system is structured around invoice/payment request generation, customer capture, payment method selection, card checkout, and USDC transfer flows.

The supplied screenshots show a deployed checkout worker, a configured KV namespace binding, a custom domain, live checkout screens, and a wallet confirmation flow. These assets materially improve credibility because they demonstrate a working production environment instead of only a design prototype.

The codebase shows a modular payment engine, a separate Light CRM/invoicing layer, Stripe checkout session and webhook handlers, USDC transfer support through wagmi/viem, and local storage based administrative configuration. The current build reads as a functional early production system with strong marketing value and a clear hardening path.

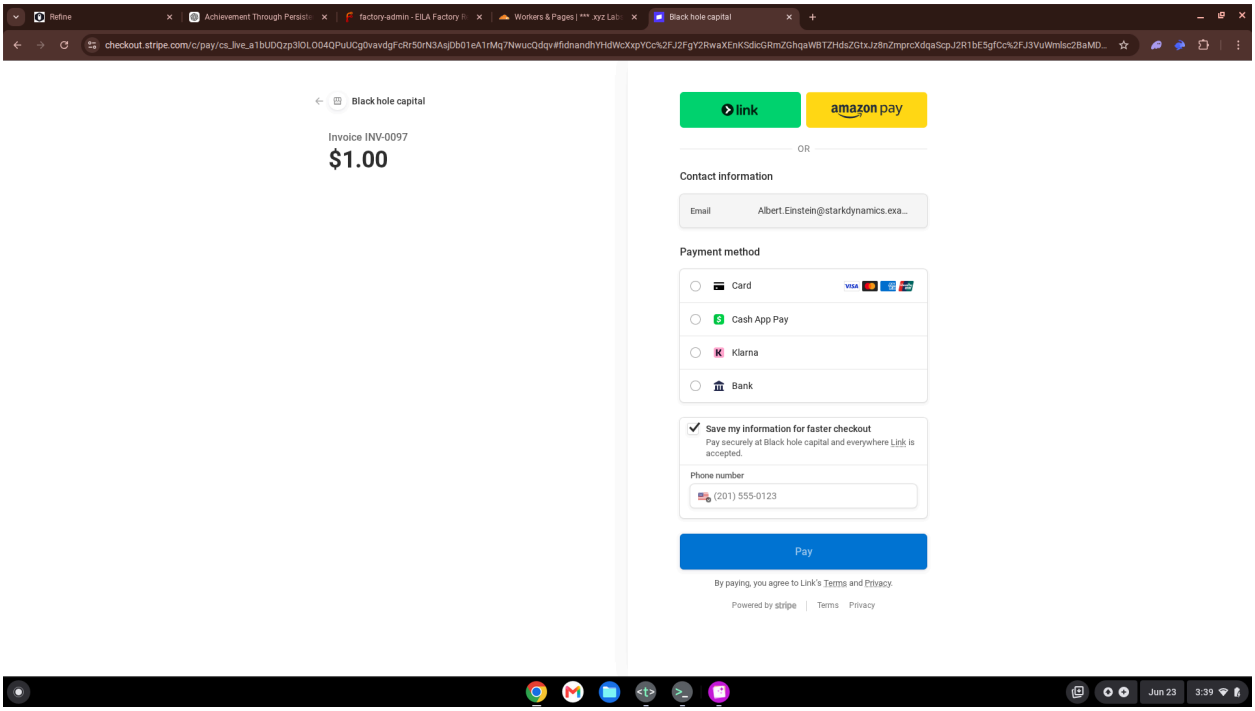
Rating Area	Assessment	Marketing Readiness
Product concept	Strong. Clear need: simple invoice checkout with card and USDC support.	High
Checkout UX	Clean, familiar, low-friction. PayMe footer and blue visual language are consistent.	High
Deployment proof	Cloudflare worker and bindings screenshots provide real infrastructure evidence.	High
Code maturity	Functional MVP/early production. Needs persistence and verification hardening.	Medium
Security posture	Good intent with webhook signature and on-chain receipt parsing; mock fallbacks should be gated.	Medium
Investor narrative	Strong once framed as payment infrastructure rather than only a checkout page.	High

3. Product Architecture Overview

PayMe Pro separates the buyer-facing checkout experience from internal request management, CRM/invoicing, and payment rail integrations.

Layer	Repository Evidence	Role
Buyer Checkout UI	payme-checkout-engine/components/PayMeCheckout.tsx, SinglePaymentPage.tsx, PaymentLanding.jsx	Presents payment request, amount, customer information, card/USDC method selection, and PayMe footer branding.
Payment Request Engine	payme-checkout-engine/lib/paymentRequests.ts	Creates, lists, updates, and transitions payment requests through open, pending, paid, and expired states.
Coupon / Discount Engine	payme-checkout-engine/lib/coupons.ts, CouponField.tsx	Validates coupon codes, tracks usage, supports percent and fixed discounts.

Stripe Rail	lib/stripe.ts, pages/api/stripe/create-checkout-session.ts, webhook.ts	Builds hosted Stripe checkout payloads, creates sessions, and validates webhook signatures.
USDC Rail	src/services/usdcTransfer.js, lib/usdc.ts, pages/api/usdc/verify-payment.ts	Connects wallet, switches to Base, executes USDC transfer, and verifies ERC-20 transfer logs.
Light CRM / Invoicing	modules/light-crm/*, lib/spine/*	Manages customers, invoices, recurring flags, invoice numbers, history, and payment link generation.
Configuration / Branding	config/branding.ts, lib/storage.ts, src/styles/*	Centralizes default PayMe brand settings, company information, wallet, and UI appearance.



Stripe checkout reference screenshot for category comparison and customer expectation benchmarking.

4. Repository Inventory

Category	Count / Notes
Total files	108
JS / JSX / TS / TSX lines	6,173
CSS lines	2,654
Docs / reports	README, integration notes, stage reports, hardening notes
Large visual assets	drop.png, wallpaper assets, screenshots supplied separately

5. Code Audit Findings

ID	Area	Status	Finding
P-01	Buyer checkout flow	Pass	Checkout UI supports invoice/payment request display, amount, customer email, coupon, card/USDC selection, and visible PayMe branding.
P-02	Payment request state machine	Partial	Open/pending/paid state transitions exist. Server-side persistence should replace or

			augment localStorage for production records.
P-03	Stripe session creation	Pass with hardening	Server API creates Stripe Checkout Sessions. Current coupon enforcement comments identify the correct hardening point for server-side validation.
P-04	Stripe webhook security	Pass	Webhook handler disables body parsing and uses Stripe signature verification before accepting completion events.
P-05	USDC execution	Pass	Wallet connection, Base chain selection, and ERC-20 USDC transfer function are implemented using wagmi/viem.
P-06	USDC verification	Partial / High priority	Receipt parser checks Base mainnet, USDC contract, transfer event, destination wallet, transaction status, and amount. RPC failure currently mock-verifies; this should be disabled in production.
P-07	Data persistence	Needs hardening	Settings, coupons, and requests are currently localStorage based in the checkout engine. Good for demos/admin prototype; production should use KV/D1 or a signed backend store.
P-08	CRM / invoice module	Pass	Light CRM module includes company profile, customers, invoice preview, recurring billing, open invoices, and history flow.
P-09	Branding integration	Pass	Powered by PayMe, companyName, accentColor, and checkout labels are centralized through branding/storage configs.
P-10	Environment config	Partial	Sensitive Stripe keys are read server-side in API handlers. Avoid VITE-prefixed secret fallbacks in production because VITE variables are intended for client exposure.

6. Technical Risk Register

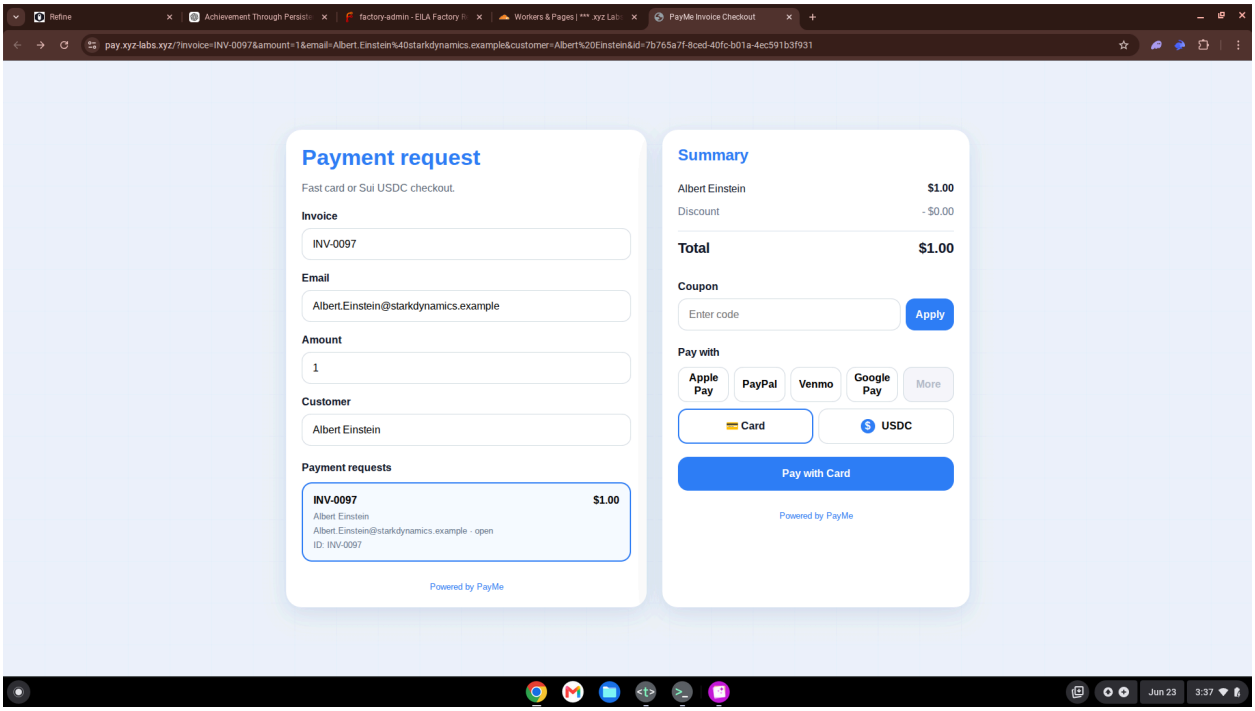
Severity	Risk	Evidence	Recommended Action
High	Mock verification fallback	/api/usdc/verify-payment returns verified:true if RPC fails.	Gate mock mode behind explicit DEV flag; production should return pending/unverified on RPC failure.
High	Client/localStorage persistence	Payment requests, settings, coupons, and admin state are stored in browser localStorage.	Move authoritative state to Cloudflare KV/D1/Durable Objects; keep localStorage only as UI cache.
Medium	VITE secret fallback	API handlers read VITE_STRIPE_SECRET_KEY and VITE_STRIPE_WEBHOOK_SECRET.	Use server-only environment variable names. Remove VITE secret fallback before public production.
Medium	Server-side coupon validation	Comment notes future KV/D1 validation; current session payload trusts frontend amount.	Enforce coupon lookup and final amount calculation on server before Stripe session creation.
Medium	Wallet/network coverage	Code targets Base USDC while screenshot shows Sui confirmation as separate flow.	Clearly label supported networks by environment; avoid mixed-network confusion in buyer UX.
Low	Logging	Some console logs include session metadata and email.	Use structured redacted logs for production.
Low	Dependency hygiene	Modern React/Vite/wagmi stack. No lockfile audit was executed in this environment.	Run npm audit / SCA scan before enterprise sales process.

7. Marketing Audit

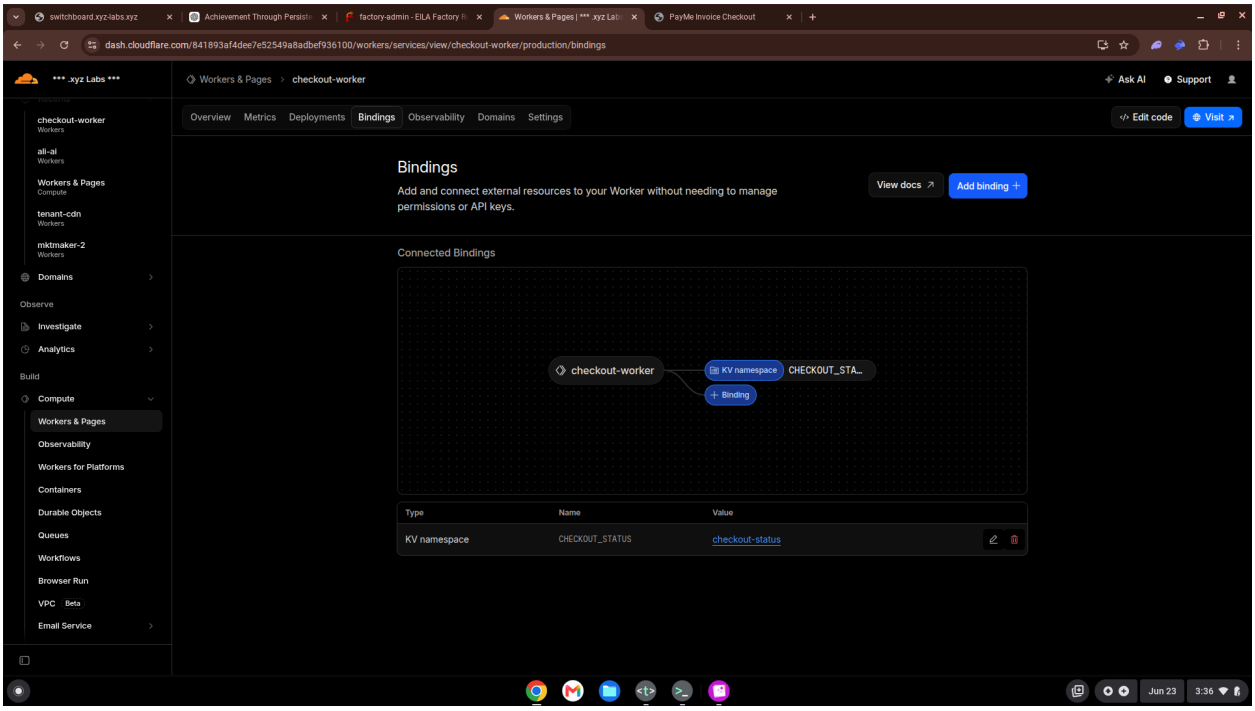
Recommended public positioning: PayMe Pro is enterprise payment infrastructure for invoices, checkout links, and stablecoin settlement. The cleanest customer-facing line remains: USDC • Cards • Instant Settlement.

Asset / Message	What Works	Recommended Use
Powered by PayMe	Simple, familiar, high trust. Mirrors established payment processor footer patterns.	Keep at the bottom of checkout, invoices, and email receipts.
USDC • Cards • Instant Settlement	Clear, specific, and non-hype. Explains the value proposition in five words.	Use under the PayMe wordmark and in cut sheets.
Cloudflare Screenshots	Shows real deployed infrastructure, not mockups.	Feature prominently in investor and partner materials.
Checkout screenshots	Demonstrates familiar UX and payment method parity with mainstream processors.	Use as product proof in one-pagers.
Wallet confirmation screenshot	Shows actual crypto settlement path and buyer-side confirmation.	Use as technical evidence, not hero marketing.
Stripe comparison screenshot	Validates market category and user expectation.	Use internally or in investor context; avoid implying affiliation.

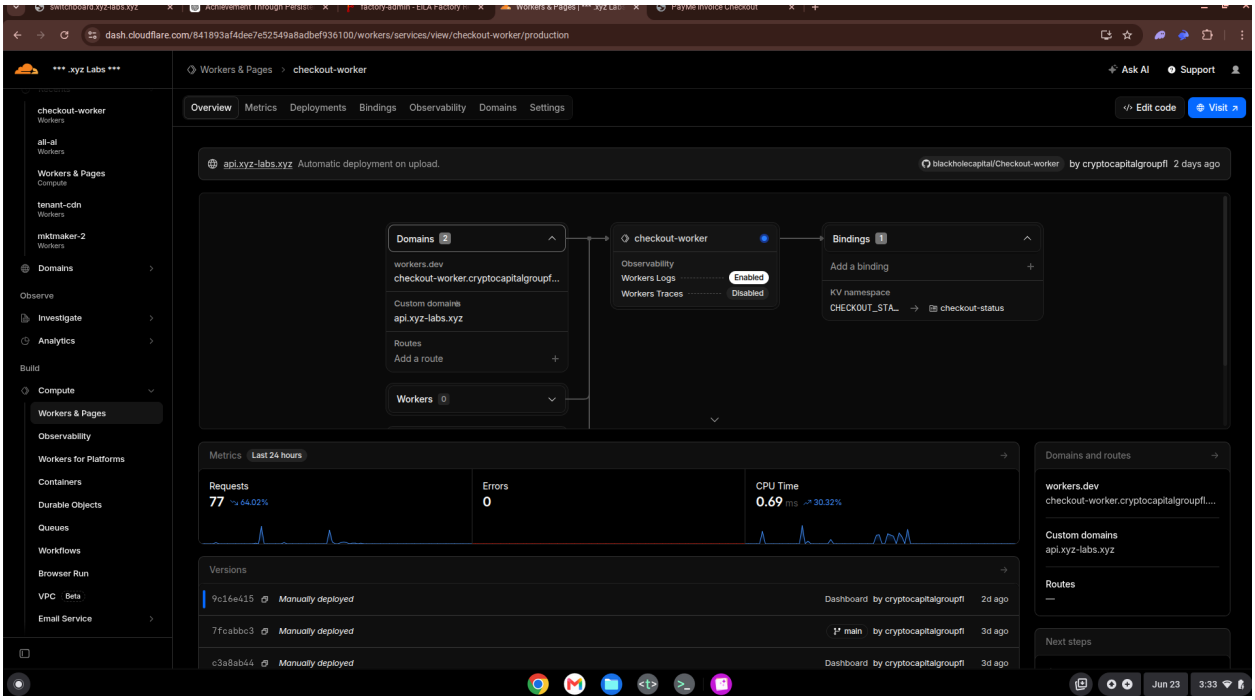
8. Production Screenshot Evidence



Buyer checkout screen showing PayMe footer, card flow, and invoice form.



Cloudflare Worker bindings showing checkout-worker connected to KV namespace CHECKOUT_STATUS.



Cloudflare Worker overview with custom domain, service binding graph, metrics, and deployment versions.

9. Investor-Readable Technical Specification

Spec	Current Evidence	Production Message
Checkout Modes	Card and USDC flows are present in UI and code.	PayMe Pro supports familiar checkout UX and stablecoin rails.
Settlement Network	Base USDC constants and ERC-20 transfer logic are implemented.	Designed for low-friction USDC settlement on Base.
Infrastructure	Cloudflare Worker, custom domain, and KV binding are shown in screenshots.	Runs at the edge with globally distributed infrastructure.
Customer / Invoice Layer	Light CRM creates customer records, invoice previews, invoice numbers, and payment links.	Built for payment requests, invoices, and lightweight B2B collection workflows.
Verification	USDC receipt validation parses logs against expected wallet and amount.	Can verify on-chain transfer evidence; production hardening should remove mock fallback.
Brand Layer	PayMe footer and default branding are centralized.	Brand can be embedded without overtaking merchant checkout experience.

10. Recommended Hardening Roadmap

Phase	Action	Outcome
Phase 1	Disable mock verification in production, remove VITE secret fallbacks, redact logs.	Safer public launch posture.
Phase 2	Move requests/coupons/settings to Cloudflare KV/D1, enforce server-side coupon calculation.	Authoritative production data layer.
Phase 3	Add admin authentication, audit logs, payment event IDs, replay protection.	Enterprise-grade operational controls.
Phase 4	Add webhooks to CRM/email/notification services and receipt delivery.	Closed-loop payment operations.
Phase 5	Add monitoring dashboard: transactions, settlement latency, failed verifications, worker errors.	Investor and customer-facing operational visibility.

11. Final Audit Conclusion

PayMe Pro has the bones of a credible payment infrastructure product: clear checkout UX, real deployment evidence, on-chain payment logic, Stripe compatibility, lightweight CRM/invoicing, and a brandable powered-by footer. The product should be marketed as early production infrastructure with a defined hardening path, not as a speculative concept.

The most important next step is to make the server side authoritative: persistent storage, production-only verification behavior, secure secrets, and audited payment state. Once those are completed, the product can be presented as a more mature alternative payment layer for invoices, AI agents, services, and embedded commerce.

12. Reference QR Codes

XYZ Labs

PayMe Pro