System Architecture Narrative

CredX Tech Inc

September 2026

Contents

Table 1: Document history

Date	Comment
Sep 3 2026	Rewritten for the embedded lending architecture
Sep 4 2026	Replaced with SOC 2 narrative v1.0
Sep 4 2026	Verified against source repositories
Sep 4 2026	Recorded PostgreSQL and S3 ca-central-1 detail

1 Purpose and Scope

This narrative describes the technical architecture of the CredX platform — the components the system is built from and how data moves between them — as the data-flow and system-composition counterpart to the Security Architecture Narrative, which addresses the controls protecting that architecture.

It focuses on **system composition and data flow**: what components exist, where they run, and how a transaction moves through them from order creation to ledger posting. Security controls protecting this architecture are described in depth in the Security Architecture Narrative and referenced here only for context.

Scope: Security, Availability, and Confidentiality. CredX is targeting SOC 2 Type I first, followed by Type II; this document describes the system as designed and, where noted, as implemented, and **does not represent a completed or issued SOC 2 report**.

2 Architecture Overview

This narrative describes the system at a **functional and environmental level**. Consistent with SOC 2 description practice, it does not name source repositories, modules, files, schema internals, or configuration detail — those change continuously, and exposing them adds exploitable detail without improving a reader's understanding of the controls.

The platform comprises five deployable components:

Component	Role	Environment
Backend API service	Order intake, webhook handling, credit-decisioning orchestration, loan draw-down, wallet credential issuance and validation, POS reconciliation, and the API surface for all clients	Heroku Private Spaces, Canadian region
Consumer web application	Customer-facing application: credit application flow, wallet and payment-credential presentation, scanning, transaction history, and profile management	Browser-delivered web application

Component	Role	Environment
Merchant mobile application	Merchant-facing application: order queue and in-store scanning; receives push and real-time order alerts	iOS and Android mobile devices
Administrative dashboard	Internal administration over the platform API	Browser-delivered web application, internal use only
POS device application	Native application deployed to merchant point-of-sale hardware	Clover POS terminals

An important distinction for the security model: **the consumer experience is delivered as a web application, while the mobile application is merchant-facing.** The mobile hardening controls described in the Security Architecture Narrative therefore protect the merchant application. The consumer credential-presentation path runs through a browser, where a different threat model applies.

The POS device application is a deployed component of the estate. Its scope, data handling, and release process should be assessed and either brought explicitly within the documented system boundary or carved out with a stated rationale.

2.1 Layers

The **client layer** comprises the consumer web application, the merchant mobile application, the POS device application, and Apple Wallet and Google Wallet passes carrying a dynamic, rotating barcode used as the in-store payment credential.

The **application layer** is a single backend API service exposing a GraphQL interface, hosted in Heroku Private Spaces in the Canadian region. Database migrations are applied automatically on release.

The **data layer** is a **PostgreSQL** relational database, co-located with the application layer inside the Canadian Private Space and encrypted at rest by the platform. **AWS S3 in `ca-central-1`** provides object storage for merchant logo images only.

A **real-time layer**, backed by **Redis**, drives live notifications over a websocket channel, with a managed push service delivering alerts to merchant devices.

The **integration layer** connects the backend to the external systems documented below.

Both wallet providers are implemented — Apple Wallet via PassKit with certificate-based pass signing and token-authenticated push updates, and Google

Wallet via service-account authentication.

3 Environment Segmentation

CredX separates development, staging, and production environments to limit the blast radius of any single environment's issues and to keep non-production work away from real customer and payment data. Isolation is achieved via **separate Heroku Private Spaces per environment** or, where a separate Private Space is not used, **strict network segmentation**.

All PII and payment-related workloads — production customer, merchant, and transaction data — are confined to the Canadian Private Space used for production. Development and staging environments are expected to operate on synthetic or de-identified data rather than live customer data; **production status of this practice should be confirmed prior to audit fieldwork [TBD]**.

Rationale:

- **Data residency** — hosting production in a Canadian Heroku Private Space keeps PII and payment data within Canada, supporting Alberta PIPA and PIPEDA obligations.
- **Blast-radius containment** — a misconfiguration, bug, or compromised credential in development or staging cannot directly reach production data or production external-partner credentials.
- **Environment-specific secrets** — per-environment key and credential separation is easier to enforce and audit when environments are already isolated at the infrastructure level.

Formal, production-verified evidence of this segmentation — a reviewed Private Space inventory and documented network policies — should be gathered as part of SOC 2 Type I evidence collection. As of this writing the approach is **specified** in CredX's architecture and security documentation, with production implementation to be verified.

4 Application Components

4.1 Backend Structure

The backend is organized into discrete functional domains — merchant management, orders, payments, refunds, lending, wallet and credential issuance, webhooks, messaging, and audit — rather than as a single undifferentiated codebase. The relational data model is defined declaratively and version-controlled, and serves as the authoritative schema definition and migration source.

4.2 Merchant Onboarding and POS Connection

Merchants onboard directly with CredX and then connect their point-of-sale system by one of two integration paths, reflecting the two supported POS providers:

- **Clover** — connected via OAuth 2.0, with CredX storing an access token and refresh token and handling automatic token refresh and recovery.
- **Square** — connected via a **static access token entered by an administrator during onboarding** rather than an OAuth flow. This token has **no built-in refresh, expiry, or revocation-detection mechanism** on the CredX side, a tracked integration gap.

Once connected, a merchant's own software authenticates to CredX using a merchant-specific API key. These keys are the credential of record for attributing merchant-initiated calls and are **stored as salted hashes**, in contrast to the POS integration credentials described in the Security Architecture Narrative, which are stored unencrypted.

4.3 Lender Records

Lenders are represented inside CredX by a **thin reference record** — sufficient to associate a loan with a lender identifier. Underwriting, credit decisioning, and lending-policy logic live entirely with Ownest, CredX's affiliated underwriting partner; CredX does not replicate underwriting logic locally.

This design keeps CredX's own database **free of applicant financial data**, since applicant details are submitted by the customer directly to Ownest's hosted questionnaire rather than passing through CredX's backend.

5 The Transaction Lifecycle

Step	What happens
1. Order appears	The POS provider sends a webhook when an order is created or updated at the point of sale. CredX verifies the webhook signature, fetches the full order back from the provider to confirm its contents, and mirrors it into the CredX database
2. Merchant device notified	CredX pushes a notification to the merchant's device over both the push service and the real-time channel, carrying the order identifier, merchant, location, and total. **No customer data is included in this event**

Step	What happens
3. Customer presents pass	The customer presents their Apple Wallet or Google Wallet pass; the scan submits the rotating barcode value to the backend, which validates the time-bound token and resolves it to a specific user and their active loan
4. Loan drawn down	The loan is drawn down against the order total. The payment record, the transaction record, and the loan ledger entry are **written atomically together**, so a failure partway through cannot leave the financial data model inconsistent
5. POS reconciled	CredX writes the payment back to the POS provider. For **Square**, this records a payment collected outside Square's own tender flow. For **Clover**, the payment is created through a customer-facing tender call and then read back to confirm it landed

6 Data Model

The financial core of the schema — the **money data model** — is organized around six related record types: **orders, order line items, payments, refunds, transactions, and the loan ledger**. Together they capture the full lifecycle of an order from POS mirror through loan draw-down to refund.

The remaining record types cover identity and access, merchants and their devices and locations, lending, the wallet and payment-credential lifecycle, and the audit log.

This narrative describes the data model **at a functional level only**. Schema internals, column names, and configuration detail are deliberately not disclosed, consistent with SOC 2 description practice of describing architecture and controls without exposing exploitable detail.

6.1 Arithmetic and Integrity Guarantees

- **Precise decimal arithmetic.** Every monetary value is stored and computed using fixed-precision decimal types. **No floating-point monetary values exist anywhere in the schema**, avoiding rounding drift across payments, refunds, and ledger entries.
- **Atomic transactions.** The draw-down path wraps the loan debit — which writes both the transaction and the ledger entry — together with the order status update and the payment status update in a single database transaction, with rollback handling that fails the payment and the order together on error.

- **Idempotency keys.** Both payments and refunds carry a unique idempotency key, and both paths return the existing record rather than reprocessing when a key is replayed — for example on webhook retry.
- **Credential attribution.** Payment and refund records are each stamped with the merchant API credential that authorized them, giving credential-level attribution for every money-moving event.

6.2 Identifier Hygiene

Several record types share a **commonly-named external identifier field**, and each refers to a **different external partner's identifier namespace**. This is a known identifier-hygiene consideration rather than a defect: any join or lookup involving one of these fields must first establish which partner namespace it refers to before the result can be trusted. Engineers and auditors reviewing data relationships should treat these identifiers as **context-dependent, not globally unique**.

Note separately that the loan identifier itself serves as the correlation reference passed to Ownest, so that an application completed on Ownest's hosted questionnaire can be correlated back to the originating loan without applicant financial data crossing the API boundary.

7 External System Integrations

The CredX backend integrates with **eleven external systems**. Full sub-service-organization detail, including what data each partner receives, is in the System Description and should be treated as the authoritative source for vendor-risk and privacy review; this table supports architecture and integration-pattern review specifically.

System	Role	Auth model	Integration pattern
Clover	POS system — order source and payment tender destination	OAuth 2.0, automatic token refresh + recovery	Webhook (inbound orders) + OAuth-authenticated REST calls (outbound tender/read-back)
Square	POS system — order source and payment record destination	Static, admin-pasted access token	Webhook (inbound orders) + token-authenticated REST calls (outbound EXTERNAL payment write)

System	Role	Auth model	Integration pattern
Ownest	Credit underwriting and loan-terms decisioning (affiliated company)	OAuth 2.0 client credentials (machine-to-machine)	Webhook (application status back to CredX) + OAuth M2M API calls (proposal requests); **correlation ID only, no applicant financial data**
Apple Wallet / APNs	Payment instrument — issues and updates the wallet pass carrying the rotating barcode	Pass certificates + APNs token authentication	Pass issuance/update API + push notification for pass updates
Google Wallet	Payment instrument — Android equivalent of the Apple Wallet pass	Service-account JWT	Service-account-authenticated API calls for pass issuance/update
Sign in with Apple / Google	Customer identity — authentication at account creation/login	JWKS validation / OAuth 2.0 ID token	OAuth2/OIDC identity-token exchange
Twilio	SMS delivery — login codes / OTP	Account SID + auth token	API key-authenticated REST calls (outbound send)
SendGrid	Email delivery — OTP and administrative links	API key	API key-authenticated REST calls (outbound send)
Expo Push	Mobile push delivery — merchant order alerts	Push token	Push-token-addressed notification API (outbound)
AWS S3 (`ca-central-1`)	Object storage — merchant logo images only; no other data reaches this store	Access key	Access-key-authenticated object storage API. Uploads rely on the bucket's default server-side encryption rather than requesting it explicitly

System	Role	Auth model	Integration pattern
Redis	Internal real-time notification backbone; not a partner data recipient	Connection URL	In-Space service connection; no durable partner data stored

Two integration-pattern gaps are worth flagging architecturally: **Square's static admin-pasted token** has no built-in refresh, expiry, or revocation-detection mechanism on the CredX side, and **Clover's OAuth flow does not support partial refunds** on payments tendered externally — that is, through the loan-funded flow rather than Clover's own tender UI.

8 Known Architectural Limitations

The following are **confirmed conditions of the current system**, established by internal review rather than hypothetical risks. They are described functionally; the specific implementation locations are held in CredX's internal engineering records rather than in this narrative.

Limitation	Description
Refunds are amount-level, not item-level	Refunds carry no per-item allocation, and the reason field on refund records is present but not populated. Reduces the granularity of refund reporting and root-cause analysis for partial refunds
Order quantity cannot express fractions	Line-item quantity is stored as a whole number, so fractional or weighted quantities — items sold by weight — are not supported and could fail or be misrecorded
No currency dimension; CAD assumed	The data model carries no currency attribute. Acceptable while CredX is Canada-only, but a hard blocker for the planned U.S. expansion, which requires multi-currency support
Funding lender not reconciled back onto the loan	A loan is created against a default lender reference, and the record is not updated once the funding lender is confirmed, so a placeholder persists. Limits accurate reporting on which lender funded a given loan without manual cross-reference
Clover partial refunds on externally-tendered payments	Where a payment was tendered outside Clover's own flow — that is, loan-funded — a partial refund **silently fails** to reach Clover's books, leaving the provider's order record out of sync with CredX's

Limitation	Description
Integration failures absent from the audit log	Token refresh failures, rejected webhooks, and failed provider calls are written only to shorter-retention application logs, not the audit log, limiting forensic visibility into integration failures over time
Payment-confirmation push disabled	The push notification for payment confirmation is disabled, so merchant applications rely solely on the real-time websocket channel with no push-based fallback if that connection drops

8.1 Additional Findings

Two conditions surfaced during internal review that appear in **no prior CredX compliance document**:

The POS device application is undocumented A native application deployed to merchant point-of-sale hardware exists in the estate but appears in no system description, boundary definition, or sub-service organization inventory. It should be assessed and either brought inside the documented system boundary or explicitly carved out with a rationale.

Client application roles were described backwards Prior narratives attributed a consumer-facing native mobile application to CredX. The consumer experience is in fact a **web** application, and the **mobile application is merchant-facing**. This matters for the security narrative: the mobile hardening controls described there — application hardening, device-integrity detection, certificate pinning, platform-backed secure storage — apply to the **merchant** application, while the consumer credential presentation path runs through a **browser**, where those controls do not apply and a different threat model governs.

9 Scalability and Availability Considerations

At the product level, CredX's Security Requirements Specification calls for **rate limiting and throttling** on the API surface, along with **health checks and automated restarts** for the backend service. DDoS protection and regular, tested backups are also specified as availability-supporting controls.

CredX's Recovery Time and Recovery Point Objectives are documented against the confirmed Heroku Private Spaces environment — Tier 1: 4 hours weekdays / 12 hours weekends RTO with a 24-hour RPO; Tier 2: 8 hours / 24 hours; Tier 3: 48 hours / 48 hours. The backup policy is likewise confirmed in production. **No restore test has yet been performed against either**, so recovery is a validated design rather than demonstrated capability. Note that the 24-hour Tier 1 RPO admits up to a day of lost order, payment,

and loan-ledger data; the Business Continuity Plan records the reasoning for accepting that window at current volume. The confirmed backup policy is daily automated backups, encrypted at rest, retained at least 90 days, with restore testing at least twice yearly. **The restore testing requirement has not yet been exercised.**

Because the application layer and data layer are **co-located within a single Heroku Private Space per environment**, availability of the production Canadian Private Space is currently **the single largest infrastructure-level availability dependency** for the platform. This is consistent with the architecture's current stage — a single-region deployment appropriate for a pre-revenue, Canada-only platform — and is expected to be revisited as CredX scales.

10 Architecture Roadmap

CredX's Security Architecture Narrative describes a three-phase **Layered Security Roadmap**: Phase 1 (Baseline/MVP — the current phase), Phase 2 (Enterprise Security, closing the live-monitoring/SIEM gap), and Phase 3 (Carrier-Grade, introducing HSMs and zero-trust networking). This narrative is written against **Phase 1**.

The architectural changes flagged above should be sequenced against this same roadmap rather than treated as one-off fixes:

- The **currency dimension** should be added before, or as part of, any U.S. expansion effort, since multi-currency support is a hard prerequisite for operating outside Canada.
- The **fractional-quantity** and **lender-reconciliation** gaps should be resolved ahead of expanding into merchant verticals such as grocery or weight-based retail, or lending-partner arrangements where these gaps would create merchant-facing or financial-reporting issues.
- The **audit-log** and **disabled-push-notification** gaps are of lower architectural complexity and are candidates for near-term closure.

11 References

11.1 Narratives

- System Description
- Security Architecture Narrative
- Control Environment Narrative
- Organizational Narrative
- Products and Services Narrative
- Business Continuity Plan Narrative

11.2 Policies

- Application Security Policy
- Artificial Intelligence Governance Policy
- Availability Policy
- Data Classification Policy
- Datacenter Policy
- Encryption Policy
- Log Management Policy
- Processing Integrity Policy
- System Change Policy
- Vulnerability Management Policy