SPEC-1 PixPax Architecture

SPEC-1-PixPax API Stabilization & Concord-Style Receipts Architecture

Generated: 2026-02-24

Background

The current `ternent-api` codebase has grown organically (PixPax, Stickerbook, Accounts, Billing, WS, EJS pages) and now feels unstable and hard to extend safely.

PixPax's core principle is **append-only receipts**: the canonical truth is the receipt stream; any database is a convenience/projection that can be rebuilt from receipts at any time.

You deploy on **DigitalOcean Kubernetes**, run **Postgres in-cluster**, and use **DigitalOcean Spaces** as managed object storage.

Requirements (MoSCoW)

Must
- Canonical, **append-only receipt streams** stored in Spaces.
- Receipts are **cryptographically verifiable** (ECDSA P-256 signatures + hash chaining).
- **Explicit behavior**: no hidden writes. "Save" creates a receipt; autosave is opt-in client policy.
- **Stable API**: consistent error envelope, validation, versioning.
- **Security**: authz required for all sensitive operations; remove/lock public admin/billing endpoints.
- **Rebuildable Postgres** projection with deterministic replay.

Should
- In-process synchronous projection for "save" path (simple MVP) + background projector for repair/rebuild.
- Observability: structured logs, request IDs, metrics, basic tracing hooks.
- Idempotency for commands to safely retry.

Could
- OpenAPI + generated SDKs.
- Event compaction via snapshots.
- Async jobs/queue (issuance batches, reconciliation, email).

Won't (MVP refactor)
- Microservices split. Start as a modular monolith.

Method

Concord ethos alignment
- **Spaces receipts are the only source of truth.**
- **Postgres is a projection** (drop/rebuild anytime).
- **Receipts are ordered and tamper-evident** via `prevHash` + `hash` + signature.
- **Idempotent commands** prevent duplicate receipts on retries.
- **No magic**: receipts only created by explicit command endpoints.

Target architecture (modular monolith on Kubernetes)

Core components
- **API Service (Node)**: HTTP + explicit WS subscriptions; routes are thin.
- **Vault Transit (ECDSA P-256)**: custodial signing keys (non-exportable by default).

- **Spaces**: canonical receipt store + derived snapshots.
- **Postgres**: projections for fast reads and lookups (replayable).

See PlantUML:
- `diagrams/component.puml`
- `diagrams/domain-storage.puml`
- `diagrams/save-sequence.puml`

Identity model
- **Account**: workspace/billing boundary.
- **Identity**: an ECDSA P-256 signing key + metadata.
- One account has multiple identities.
- **Pixbook is bound to exactly one identity** (owner/collector). Individual cards may be traded via receipts.

Receipt schema (canonical event)

```json
{
  "eventId": "01H...ULID",
  "stream": { "type": "pixbook", "accountId": "...", "bookId": "..." },
  "type":
"PIXBOOK_CREATED|PIXBOOK_SAVE|CARD_ADDED|CARD_REMOVED|CARD_TRADE_*|PACK_ISSUED|PACK_OPENI
  "payload": {},
  "createdAt": "2026-02-24T12:34:56.000Z",
  "issuer": { "identityId": "...", "publicKeyId": "..." },
  "idempotencyKey": "uuid",
  "prevEventId": "…",
  "prevHash": "sha256:…",
  "hash": "sha256:…",
  "signature": "ecdsa-p256:base64…",
  "schemaVersion": 1
}
```

Canonical JSON
- Use RFC 8785 JSON Canonicalization Scheme (JCS) for deterministic hashing.

Spaces layout
- `pixpax/pixbooks/{accountId}/{bookId}/events/{eventId}.json`
- `pixpax/pixbooks/{accountId}/{bookId}/snapshot.json` (derived)

Command vs Query API
- Queries (read-only): `GET /v1/pixbooks/:id`, `GET /v1/pixbooks/:id/receipts?...`
- Commands (writes): `POST /v1/pixbooks/:id/commands/save`, `POST /v1/trades/:id/commands/accept`, etc.
- Commands return `{"eventId": "..."}` when the receipt has been appended successfully.

Projection strategy (simple MVP)
- On command success, API **applies projection synchronously** for the affected stream (pixbook) to keep UX predictable.
- A separate **projector/replayer** job can:
  - rebuild all projections from receipts,
  - repair drift,
  - backfill new projections.

Data model (projection)
See `api/schema.sql` for concrete tables:

- `identities`
- `pixbooks`
- `pixbook_ledger_heads`
- `projector_offsets`

Security
- AuthN via better-auth sessions/passkeys.
- AuthZ via account membership + identity ownership.
- Vault policies restrict signing to authorized workloads (K8s auth) and internal service accounts.

Implementation

Phase 1 — Stabilize current API (no behavior change)
- Add global error middleware + consistent envelopes.
- Add request IDs + structured logging.
- Lock down admin/billing endpoints.
- Harden WS message parsing and auth handshake.

Phase 2 — Modular refactor
- Split routes into modules (accounts, pixbooks, stickerbook, pixpax).
- Introduce Zod validation and typed service layer.
- Introduce OpenAPI skeleton (this pack includes one).

Phase 3 — Receipts + Vault signing
- Implement receipt writer (append-only).
- Implement hashing/canonical JSON + ECDSA P-256 signing via Vault Transit.
- Implement verification utilities (server + client).

Phase 4 — Projection & replay
- In-process projection on command.
- Replay job for full rebuild; snapshot generation.

Milestones
- M1: Stability + security patch shipped.
- M2: Modular boundaries + validation shipped.
- M3: Receipt streams + Vault signing shipped.
- M4: Projection replay + operational readiness shipped.

Gathering Results
- 5xx rate, p95 latency, WS disconnect rate.
- Receipt verification failures (should be ~0).
- Projection drift (rebuild yields same results).
- Mean time to add a new endpoint + contract test.