Act as a senior engineer finalizing Phase 7 and preparing the codebase for the next phase. Phase 7 (Payment Gateway Integration) is complete and fully functional in sandbox mode with Remita as primary provider and Flutterwave as fallback. The implementation includes an extended database schema (payments with provider fields, metadata, audit trail; payment\_events for state changes; receipts with content hash; v\_payment\_summary view), a provider-agnostic architecture (IPaymentProvider interface, provider registry and factory with env-based configuration), full Remita and Flutterwave integrations (initiation with RRR or unique transaction reference; HMAC-SHA256 webhook signature verification; status mapping; sandbox support), a robust service layer (idempotency scaffolding, provider selection and fallback, webhook processing, receipt management), secured and documented API endpoints (POST /payments/init, GET /payments/:id, POST /payments/:id/verify, GET /payments/:id/receipt, POST /payments/webhook/remita, POST /payments/webhook/flutterwave, GET /payments/providers/status), security hardening (HMAC signature verification, timestamp validation and replay protection, structured logging), and comprehensive types and OpenAPI documentation. Monorepo builds (pnpm --recursive build) and TypeScript compilation pass across all packages; endpoints are mounted and documented; the database schema is updated; and docs (README.md, TODO.md, DEVELOPMENT\_STATUS.md, OpenAPI) are current. A prior runtime initialization issue was resolved via explicit module initialization with dependency injection, enabling full end-to-end payment flow, structured logging, and shared TypeScript types across the monorepo. Local dev stack is verified: API health at http://localhost:4000/health returns 200; web runs at http://localhost:5173; environment variables load from .env; payment routes are mounted at /payments; Zod validation works; /payments/providers/status reports both providers.

Your immediate goal is to finish idempotency enforcement so duplicate prevention is correct and race-safe, then proceed to the next phase as outlined in TODO.md while adhering to ARCHITECTURE.md, DEVELOPMENT\_GUIDE.md, PROPOSAL.md, README.md, and the OpenAPI and sequence diagrams as the sources of truth. Deliver a minimal, well-structured patch with tests, logs, and documentation updates.

Implement robust idempotency for payment initiation and webhook processing. Enforce idempotency for POST /payments/init using an Idempotency-Key header. Treat repeated requests with the same Idempotency-Key and identical request body as safe replays that return the original result; treat the same key with a different body as a conflict. Ensure concurrency safety under load and across multiple instances. Prefer a database-first approach for atomicity using Postgres constraints and upsert patterns: either add a globally unique constraint on payments.idempotency\_key or introduce a dedicated idempotency\_keys table keyed by idempotency\_key with request\_hash, payment\_id, response\_snapshot, status\_code, created\_at, and updated\_at. If you keep the logic in the payments table, add request\_hash (SHA-256 of canonicalized request payload), response\_snapshot jsonb, first\_request\_at, last\_request\_at, and replay\_count, and create a unique index on idempotency\_key. Implement atomic creation with INSERT ... ON CONFLICT DO NOTHING and recover the existing row on conflict, returning the stored response snapshot. For strictness, if an existing idempotency\_key has a different request\_hash, return 422 Unprocessable Entity with an error explaining the mismatch and do not mutate state. Ensure the service layer acquires a short-lived lock to prevent thundering herds, either via INSERT conflict semantics alone or with pg\_advisory\_xact\_lock on a key derived from the idempotency\_key. Persist response snapshots and status codes so replays are fast and consistent. Update structured logs to include idempotency\_key, request\_hash, payment\_id, and replay\_count for observability. Document the Idempotency-Key contract in OpenAPI, including examples and error semantics.

Extend idempotency to webhook processing to ensure de-duplication. Use provider event identifiers and signatures to deduplicate delivery: record provider\_event\_id and signature\_hash in payment\_events with a unique constraint so repeated webhooks do not re-apply state changes. Ensure webhook handlers are idempotent and safe under retries and out-of-order delivery. Maintain HMAC-SHA256 signature verification, timestamp checks, and replay protection as implemented. Update logs to record provider\_event\_id, computed\_signature, and validation results.

Do not introduce a migration framework; update infra/db/001\_schema.sql with minimal changes that preserve backwards compatibility, adjust indexes as required, and keep v\_payment\_summary accurate. Update packages/types/src/payment.ts and packages/types/src/payment-providers.ts to reflect any new fields. Keep the provider pattern intact and avoid leaking provider-specific details beyond the provider interface.

Add or update tests to cover unit, integration, and e2e paths. For PaymentService, test that the first POST /payments/init with an Idempotency-Key persists payment and response snapshot, a second identical request returns the same response without creating a new payment or events, a second request with the same key but different payload returns 422, and concurrent identical requests result in a single payment. For webhooks, test that duplicate deliveries for the same provider\_event\_id are ignored after the first successful application, signature verification rejects invalid payloads, and out-of-order events maintain correct final state. For API integration, test all endpoints in sandbox mode, including GET /payments/providers/status. Ensure tests run deterministically with pnpm test and work with the local Postgres in docker compose. Add structured log assertions where appropriate.

Update OpenAPI to document Idempotency-Key usage for POST /payments/init, the 422 error for key/body mismatches, and any new response fields like idempotency\_key and request\_hash. Update README.md with configuration steps for sandbox credentials, idempotency usage guidelines, and operational notes. Update TODO.md to mark Phase 7 finalized and to reflect the next phase items you implement. Update DEVELOPMENT\_STATUS.md to reflect idempotency as complete. Ensure the development guide documents running pnpm dev, environment variable expectations, and endpoints.

Acceptance criteria: duplicate POST /payments/init with the same Idempotency-Key and identical body returns the original response with 200/201 and does not create additional payments or events; the same key with a different body returns 422 with an explanatory error; concurrent identical initiations create exactly one payment; webhook handlers are idempotent and deduplicate on provider\_event\_id; payment\_events enforce uniqueness to prevent double-application; logs include idempotency\_key, request\_hash, payment\_id, and provider\_event\_id as applicable; all existing endpoints continue to function; OpenAPI and README reflect the new idempotency behavior; pnpm --recursive build and typecheck pass; tests cover the new behavior and pass locally.

Work within the existing code layout: infra/db/001\_schema.sql, packages/types/src/payment.ts, packages/types/src/payment-providers.ts, the payments module (providers, service, controller, routes), and documentation files (README.md, TODO.md, DEVELOPMENT\_STATUS.md, OpenAPI). Preserve clean architecture, type safety, and provider-agnostic design. Keep the patch minimal, focused, and well-tested. When done, provide a concise summary of changes, how to run tests, and any operational caveats.