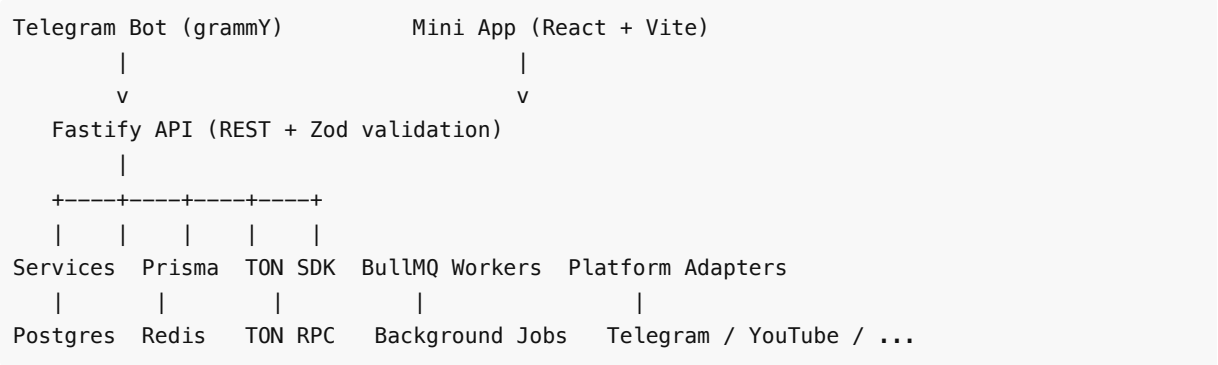
AdVault -- Project Overview

Telegram channel advertising runs on trust. Advertisers wire money and hope the post goes live. Channel owners deliver content and hope the payment clears. There's no escrow, no verification, no recourse. AdVault replaces trust with TON-based smart contract escrow, automated post verification, and a privacy-preserving fund relay.

Architecture



Server -- Fastify handles REST routes for the Mini App while grammY runs the Telegram bot. Both share the same service layer, database, and worker infrastructure. A single process boots everything.

Workers -- Seven BullMQ workers drive deal progression: payment detection (30s), auto-posting (30s), post verification (10min), deal timeouts (5min), stats refresh (6h), data purge (60min), and failed transfer recovery (2min). Redis-backed with distributed locks, retries, and dead-letter queues.

Frontend -- React 19 + TanStack Query + `@telegram-apps/telegram-ui`. Runs as a Telegram Mini App (WebApp SDK) with graceful fallback for browser development.

Blockchain -- Each deal gets a unique TON WalletContractV4 escrow wallet. Funds move through a two-hop relay (escrow -> master -> recipient) to break on-chain linkage between parties.

Key Decisions

Why TON over other chains? Native Telegram ecosystem. Users already have TON wallets via Telegram's built-in wallet. No bridge, no MetaMask, no chain switching. Plus low fees (~0.01 TON per transaction) and 5-second finality.

Why per-deal escrow wallets? Each deal gets its own wallet address. No commingling of funds, no accounting complexity, fully transparent on-chain. Anyone can verify the escrow balance for a specific deal by checking its wallet address on TonScan.

Why two-hop fund relay? Privacy. A direct transfer from escrow to the channel owner's wallet creates a public on-chain link between advertiser and channel owner. The two-hop relay (escrow -> master wallet -> recipient) breaks this linkage. The master wallet aggregates many transfers, making individual deal parties unlinkable.

Why AES-256-GCM for creative content? Ad creatives and escrow mnemonics contain sensitive business data. At-rest encryption means a database breach doesn't expose content. GCM mode provides both confidentiality and integrity verification. The purge worker destroys encrypted content after 30 days (GDPR-aware retention).

Why the platform adapter pattern? The deal/escrow/payment logic is identical regardless of where the ad runs. The `IPlatformAdapter` interface (`fetchChannelInfo` , `publishPost` , `verifyPostExists` , etc.) abstracts platform differences. Adding Instagram or Twitter means implementing 6 methods -- zero changes to the deal state machine, escrow flow, or payment logic.

Why BullMQ over cron? Cron jobs are fire-and-forget with no retry, no backpressure, and no visibility. BullMQ provides: distributed locks (prevents duplicate processing across instances), configurable retries with exponential backoff, dead-letter queues for failed jobs, Redis-backed persistence (survives restarts), and per-job metadata for debugging.

Why grammY over node-telegram-bot-api? TypeScript-first with full type inference on context objects. Middleware ecosystem (conversations plugin for multi-step flows, auto-retry, flood control). Built-in session management. The conversations plugin handles the `/addchannel` and `/createcampaign` multi-step flows cleanly without hand-rolled state machines.

Future Thoughts

- **TON mainnet deployment** -- Real USDT escrow with production wallet management and key rotation
- **Dispute arbitration** -- Community DAO or TON governance for resolving disputed deals, replacing manual admin resolution
- **Reputation scoring** -- On-chain reputation based on deal completion history, cancel rate, and verification pass rate
- **Multi-chain support** -- Ethereum L2 escrow (Base, Arbitrum) for non-Telegram platforms where TON isn't the native ecosystem
- **Real-time WebSocket updates** -- Push deal status changes to the Mini App instead of polling
- **Advertiser analytics dashboard** -- Impressions, clicks, and ROI tracking per deal with platform-specific metrics
- **Fiat on-ramp** -- TON Connect payment channels or third-party fiat-to-TON conversion so users don't need to pre-hold TON

Known Limitations

- **Testnet only** -- No real funds flow. All TON transactions happen on testnet. Mainnet deployment requires audited key management and regulatory review.
- **YouTube is proof-of-concept** -- Manual posting only (owner uploads video, submits URL). Basic verification checks that the video exists but doesn't validate content match.
- **Instagram/Twitter/TikTok adapters are stubs** -- Interface implemented, methods throw "not yet supported". Ready for integration when platform APIs are available.
- **`getChatStatistics` requires 50+ members** -- Telegram's API restriction. Channels under 50 members show estimated stats only.
- **Render free tier sleeps after 15min** -- Mitigated with UptimeRobot pinging `/api/health` every 5 minutes. Not an issue in production on a paid tier.
- **No fiat on-ramp** -- Users must already hold TON in a wallet. Fiat conversion is out of scope for the current build.
- **Bot polling mode only** -- Webhook mode requires a public HTTPS endpoint with a valid certificate. Polling works everywhere but adds ~1s latency to bot responses.

AI Code Percentage

~85% AI-generated code.

~17,800 lines across the codebase: server source (6,080 lines), server tests (7,642 lines), and web frontend (4,137 lines).

AI generated: scaffolding, service implementations, workers, test suites, frontend pages, Prisma schema, Docker configuration, deployment setup.

Human directed: architecture decisions, feature requirements, state machine design, security model, review and iteration on every generated output, bug fixes from manual testing, deployment configuration and troubleshooting.