INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 5, May 2026 | ISSN: 2456-4184 | IJNRD.ORG

DECENTRALIZED EXCHANGE ON SOLANA

Dr. Manisha V. Pise (Guide), Deepak Dey (Student), Ninad Bomanwar (Student),
Abhinav Kale (Student), Chanksh Dubey (Student), Pratik Golder (Student).
Department of Computer Science and Engineering
,
Rajiv Gandhi College of Engineering, Research & Technology, Chandrapur, India

Abstract : ***We present Tradr, a Solana-deployed decentralized exchange that pairs a constant-product automated market maker (AMM) with a CLOB-style trading interface for price discovery, position management, and real-time charting. The on-chain program is written in Anchor 0.31.1 and enforces a uniform 9-decimal precision for both legs of every pool, eliminating decimal-normalization branches and making swap arithmetic robust against rounding. Key contributions are:***

- ***(i)*** ***An escrow-gated pool lifecycle in which every pool begins life in a Pending escrow state and is promoted to a live, unlocked pool only after admin approval, with initial liquidity returned to the creator on rejection.***
- ***(ii)*** ***A treasury-aware fee split that allocates a configurable share of every LP fee to a platform treasury wallet.***
- ***(iii)*** ***A two-tier real-time pipeline built on Redis pub/sub, Server-Sent Events (SSE), and a stateless WebSocket relay.***
- ***(iv)*** ***A CLOB-style order book and candlestick UI powered by Postgres-side OHLC aggregation and lightweight-charts.***

The system is deployed on Solana devnet (program ID 9n5ivbyJdXFxBDYyzHRrt1U8DKZJjMDCxBqjkwrTJoE4) with a Next.js 14 frontend and a Dockerised relay stack on AWS EC2. We organize this paper to conform to the IEEE A4 conference template.

Keywords: Solana, AMM, Constant-Product Curve, Anchor, Wrapped SOL, Escrow, Pyth, Redis Pub/Sub, Server-Sent Events, lightweight-charts, Prisma, EC2, Docker Compose.

INTRODUCTION

Solana's parallel transaction runtime is well-suited to both AMM swaps and high-frequency order management, but a large fraction of student DEX projects collapse under three forces undefined pool governance, mixed-decimal arithmetic, and fragile real-time pipelines. Tradr is a focused academic implementation that addresses each of these concerns through four design decisions.

First : all user tokens and the wrapped SOL leg are forced to nine decimals by the on-chain program, returning InvalidPrecision for any other configuration. This matches WSOL's native precision and removes an entire class of normalization bugs.

Second : every new pool is created through a two-phase escrow flow. The user's initial liquidity is locked into a program-derived escrow account in Pending state and only flows into the live pool vaults when an admin signs an approve_pool instruction; rejection refunds the creator atomically.

Third : fees are split: every basis point collected by the AMM is shared between LPs and a treasury wallet, with the treasury share tracked off-chain in a dedicated SwapTransaction ledger.

Fourth : the trading interface is presented as a CLOB-style screen with a candlestick chart, order book, recent trades, and a positions panel, even though execution settles through the AMM. This gives retail users one-click swaps while preserving an order-management surface for power users.

The objective is a system that is feasible for a classroom demonstration on devnet yet technically defensible. We document the architecture, on-chain program, real-time infrastructure, risk controls, and the devnet evaluation procedure.

RELATED WORK AND BACKGROUND.

DEX architectures on Solana fall into two families. AMMs maintain reserves and quote against a pricing curve, guaranteeing executable size for a quoted slippage; central limit order books expose explicit price levels with time-priority matching. Anchor provides a disciplined Rust DSL for program state, PDAs, and instruction-level constraint checking. Pyth publishes high-frequency price feeds on Solana that can be consumed on-chain for hard validation or off-chain for UX guards. The SPL Token Program and Associated Token Account (ATA) standard provide the building blocks for vault-style custody. The constant-product curve crate maintained by Dean Little provides a numerically stable reference implementation of the Uniswap-V2 invariant, which Tradr embeds directly in its swap and deposit instructions.

Tradr fits into this landscape as a constant-product AMM for SOL-token pairs, augmented with an off-chain order ledger and a CLOB-style frontend. Pyth is consumed off-chain for SOL/USD reference pricing alongside CoinGecko and Jupiter as fallback sources.

SYSTEM OVERVIEW

User flow : connect wallet → create 9-decimal token (with optional metadata and image upload) → request a SOL-token pool by depositing initial liquidity into escrow → wait for admin approval → trade through the swap or trade screens.

Frontend : Next.js 14.2.35 (App Router) with TypeScript, Tailwind 4, shadcn/ui (Radix UI primitives), Solana wallet adapter, lightweight-charts for candlesticks, Zustand for trade state, and TanStack Query for data fetching.

On-chain programs (/anchor/programs/amm-anchor): A single Anchor program holding pool config, vaults, LP mints, and escrow state.

Backend (/server and Next.js API routes): a Node.js WebSocket relay containerised in Docker, a Redis pub/sub bus, and a set of Next.js Route Handlers acting as both REST endpoints and an event indexer that writes to Postgres via Prisma.

SMART-CONTRACT DESIGN

The program is declared at **9n5ivbyJdXFxBDYyzHRrt1U8DKZJjMDCxBqjkwrTJoE4** and exposes nine instructions: ***initialize, deposit, swap, withdraw, lock, unlock, create_pool_with_escrow, approve_pool, and reject_pool.***

A. Fixed-Precision Policy:

A single compile-time constant pins precision across the system:

rust
pub const DECIMALS: u8 = 9;

Both *mint_x.decimals* and *mint_y.decimals* are checked against this constant in initialize and *create_pool_with_escrow;* any mismatch produces *AMMError::InvalidPrecision*. The LP mint is also created with nine decimals *(mint::decimals = DECIMALS)*. This eliminates branchy normalization in the swap path and aligns the X-leg with WSOL's native nine-decimal representation.

B. Pool State and Admin Controls:

The Config PDA (seeds ["config", seed]) records the pool's two mints, fee in basis points, an Option<Pubkey> authority, a locked flag, a whitelisted flag, and a reserved 16-byte tail for forward compatibility. Pools are created with locked: true, whitelisted: false. The lock and unlock instructions verify config.authority == Some(user.key()) before flipping the flag, providing an emergency halt mechanism for live pools. Only pools with locked == false accept deposit, withdraw, and swap calls; PoolLocked is returned otherwise.

C. Constant-Product Curve and Fees:

Tradr uses the Uniswap-V2 invariant

$$x \cdot y = k \quad (1)$$

For an input *Δx* on the X-leg with fee f in basis points, the output is computed as

$$\Delta y = \frac{y \cdot \Delta x \cdot (10000 - f)}{x \cdot 10000 + \Delta x \cdot (10000 - f)} \quad (2)$$

Equation (2) is implemented in two places: the on-chain swap calls into Dean Little's constant-product-curve crate, and the frontend mirrors the same formula in lib/amm/swap-math.ts for pre-trade quoting and slippage display. Fees accumulate in the input vault and are realized by LPs on withdraw. A second, off-chain fee tier is computed by the API: a fixed share of each LP fee (300 BPS in the current configuration) is recorded against the platform treasury wallet *H3HdD17oX97vyhocLcg7vSZCvJPXENX9H5TiCQKTaeBT* in the swap_transactions ledger.

D. Escrow-Gated Liquidity Initialization:

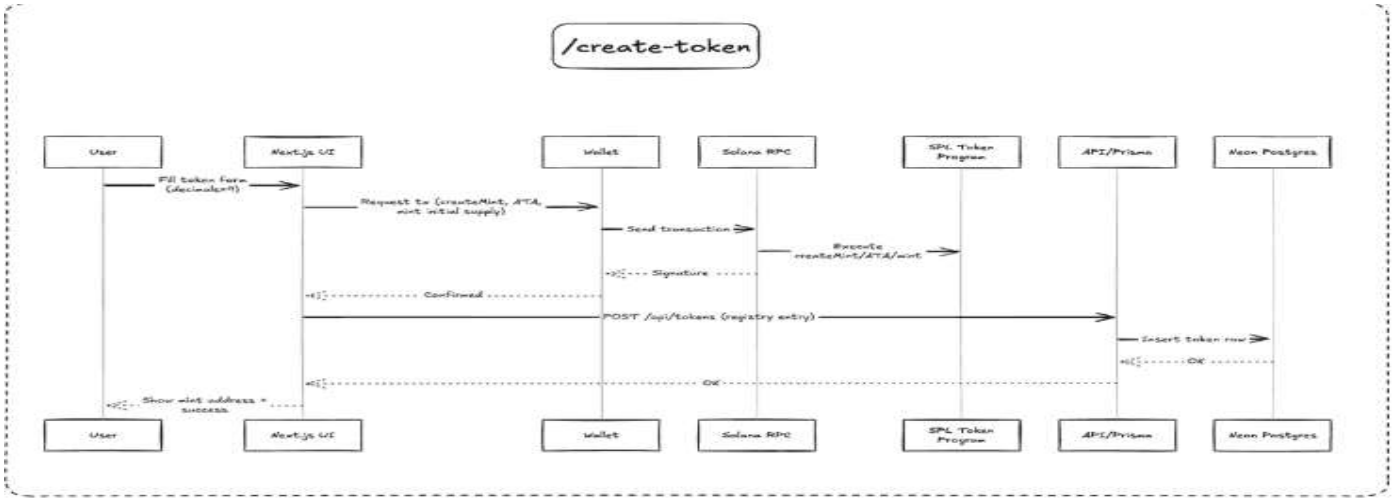
Initial liquidity is placed in a separate *EscrowState* PDA (seeds ["escrow", config]) carrying a Pending | Approved | Rejected status. On *approve_pool* the admin transfers escrowed tokens into the live vaults under the escrow PDA's signer seeds, mints LP tokens to the creator using the geometric-mean formula

$$L_0 = \lfloor \sqrt{x_0 \cdot y_0} \rfloor \quad (3)$$

implemented as an integer square root via Newton's method, and atomically sets *config.locked* = false, *config.whitelisted* = true, *escrow.status* = Approved. On *reject_pool* the deposits are returned to the creator and the escrow account is closed.

E. SOL and WSOL Handling:

The X-leg of every pool is wrapped SOL. The frontend transparently creates or reuses the user's WSOL ATA, transfers lamports, calls syncNative, submits the swap, and optionally closes the WSOL ATA at the end of the swap to refund the rent. From the user's perspective swaps are always SOL → token or token → SOL, with no manual wrapping.

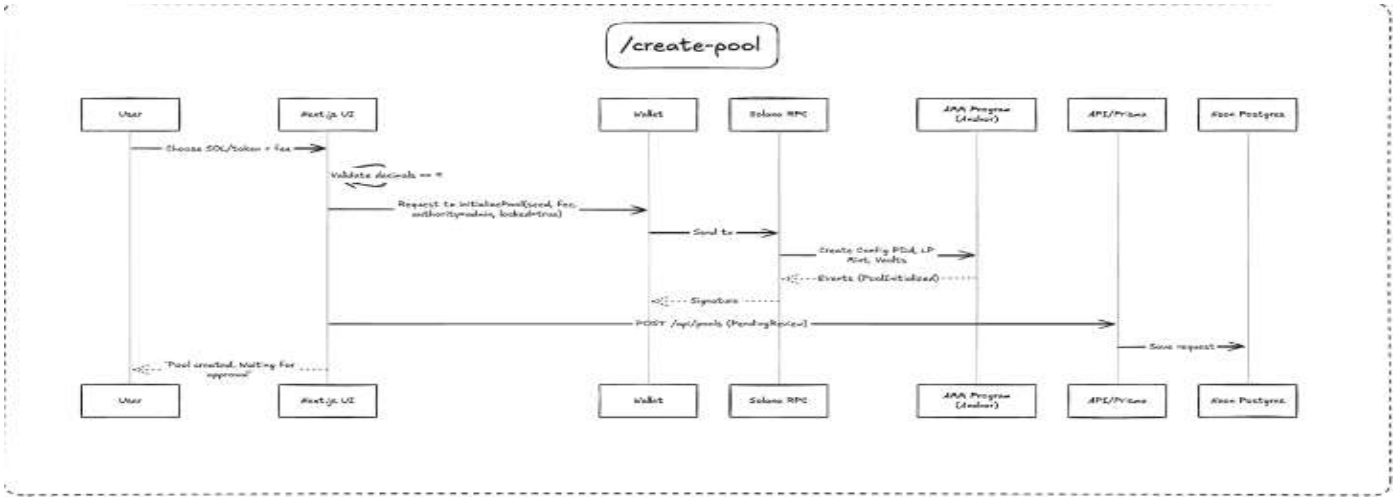


AMM POOL LIFECYCLE

The lifecycle is enforced jointly by the program and the off-chain PoolRequest table:

1. **Draft** — POST /api/pools/create records the request in Postgres with token pair, fee, and amounts.
2. **Pending** — create_pool_with_escrow initializes the Config and EscrowState PDAs and transfers the creator's initial liquidity into the escrow vaults.
3. **Approved / Listed** — approve_pool moves liquidity from escrow into the live pool vaults, mints LP tokens to the creator, and atomically sets config.locked = false, config.whitelisted = true, escrow.status = Approved.
4. **Rejected** — reject_pool returns the deposits to the creator and closes the escrow account.
5. **Paused** — lock (signed by config.authority) sets config.locked = true, halting deposits, withdrawals, and swaps.
6. **Resumed** — unlock flips config.locked back to false.

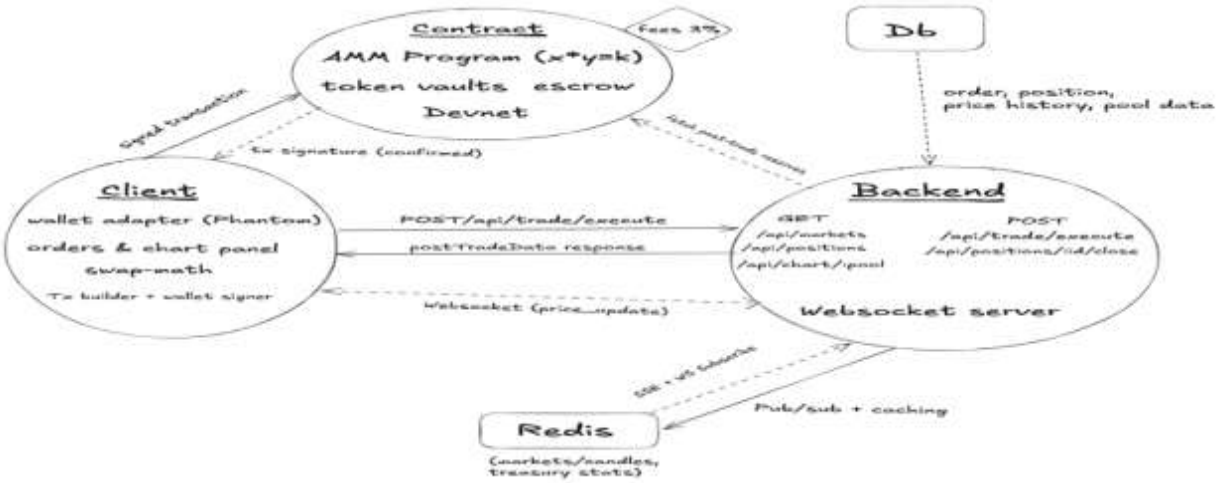
Deposits, withdrawals, and swaps revert with PoolLocked while the pool is Paused or before approval. This prevents user funds from being stranded in pools that have not yet been reviewed and provides a one-call emergency halt for production incidents.



CLOB-STYLE TRADING INTERFACE

The /trade route presents a market-style interface composed of a MarketSelector, a ChartPanel rendering OHLC candles, an OrderForm with buy/sell sides and market/limit order types, an OrdersPanel showing open orders and positions, and a top-bar reading mark price, X-reserve, Y-reserve, fee, and SOL/USD. Orders are mediated by the /api/trade/execute endpoint, which atomically creates an Order row, a Position row, a PriceHistory sample, and a treasury SwapTransaction record inside a Prisma transaction at Serializable isolation, with a single retry on P2034 serialization conflicts and idempotency keyed on the on-chain signature.

A small MakerBot table (one row per pool) is reserved for an off-chain spread-quoting bot that periodically rebalances visible inventory; it is currently scaffolded rather than active.



ORACLE INTEGRATION WITH PYTH

A SOL/USD reference price is exposed by /api/sol-price with a 15-second in-memory cache and a three-source fallback chain: CoinGecko, Pyth Hermes *(hermes.pyth.network/v2/updates/price/latest?ids[]=ef0d8b6f...)*, and Jupiter. The first source returning a finite positive number wins; on total failure the last cached value is served. The price is rendered in the trade page top bar and used by the swap UX to display USD-equivalents and to issue deviation warnings when the AMM-implied price diverges by more than a configured threshold from the oracle. Pyth integration is currently off-chain only; a future iteration will attest Pyth prices on-chain inside the swap instruction.

REAL-TIME DATA AND INDEXER

Real-time updates flow over a hybrid Redis–SSE–WebSocket pipeline:

1. The /api/trade/execute route, after committing to Postgres, publishes a price_update message to the Redis channel tradr:trades containing the post-trade pool price, reserves, side, signature, and token symbols.

2. The Next.js process subscribes to tradr:trades and re-emits per-pool events through a Server-Sent Events stream at /api/sse/trades/[poolAddress]. Each SSE connection carries a 30-second heartbeat to survive proxy timeouts.

3. The standalone WebSocket relay (server/src/ws-server.ts) provides an alternative fan-out path: clients send a set_pool message to subscribe to a single pool, the server holds a WeakMap of liveness flags, and Redis messages are routed via per-pool channels. A 30-second ping/pong heartbeat with terminate() on missed pongs maintains a clean client set.

4. Candlesticks are served from /api/chart/[poolAddress]?timeframe={1m,5m,15m,1h,4h,1d} using Postgres-side OHLC aggregation:

```
sql
SELECT
  (EXTRACT(EPOCH FROM "timestamp")::bigint
    / :interval * :interval)::int AS bucket_time,
  (array_agg(price ORDER BY "timestamp" ASC))[1]  AS open,
  MAX(price) AS high,
  MIN(price) AS low,
  (array_agg(price ORDER BY "timestamp" DESC))[1] AS close,
  SUM(volume) AS volume
FROM price_history
WHERE "poolAddress" = :pool AND timestamp >= :since
GROUP BY bucket_time
ORDER BY bucket_time ASC;
```

Aggregated payloads are cached in Redis with a per-timeframe TTL (10s for 1m candles, scaling up to 600s for daily) and invalidated by deleting chart:{pool}:* keys on each new trade. Each trade/execute call writes both a pre-trade and a post-trade PriceHistory row so candle bodies reflect the actual swap range rather than only the final mark.

BACKEND AND DEPLOYMENT

The deployment target is a single AWS EC2 instance running Docker Compose. The compose file defines a Redis 7 container and a ws-server container built from a multi-stage Node 20 Alpine Dockerfile that copies frontend/prisma/schema.prisma, runs prisma generate, and compiles the relay with tsc. The Next.js application is deployed independently and connects to the same managed Postgres (Neon-compatible) over DATABASE_URL and to the EC2 Redis over REDIS_URL. Environment variables hold the program ID, RPC URL, treasury pubkey, and admin pubkey.

SECURITY AND RISK CONTROLS

- **Escrow gate:** pools cannot serve liquidity until approve_pool flips locked and whitelisted. Rejection refunds creator atomically.
- **Authority checks:** lock and unlock require config.authority == Some(signer.key()); approve_pool and reject_pool validate that escrow mints, escrow creator, and config mints are consistent before any token movement.
- **Precision discipline:** AMMError::InvalidPrecision rejects any token with non-nine decimals at pool creation.
- **Slippage protection:** swap accepts a minimum-output parameter that aborts with SlippageExceeded when the curve cannot satisfy it; the frontend computes this minimum from a user-controlled basis-point tolerance.
- **Idempotent indexing:** every /api/trade/execute is idempotent on the Solana signature, runs in a Serializable Prisma transaction, and retries once on P2034 serialization failures.
- **Caching defenses:** Redis is treated as best-effort; every Redis call is wrapped in a try/catch and the route falls back to direct Postgres reads or skips broadcasts if the bus is unavailable.
- **Heartbeats:** SSE streams emit a comment heartbeat every 30 seconds; the WebSocket relay terminates clients that miss two pings.
- **Admin gate:** frontend/lib/config/admins.ts hard-codes a single admin pubkey (4po7zwTPcRGx9wRCwd94AgajWHXLCszw3YqF7DVALv2h) for UI gating; on-chain admin authority is held by the program's upgrade authority.

EVALUATION ON DEVNET

Tradr was validated on Solana devnet through three test families.

Functional:

- Creation of a 9-decimal SPL token with Metaplex metadata.
- Pool creation via create_pool_with_escrow with verification of Pending escrow status.
- Admin approval flipping the pool to locked == false, whitelisted == true and minting LP tokens by integer-sqrt geometric mean.
- Swaps executed in both directions with automatic WSOL wrapping.
- Liquidity addition and removal with proportional LP minting and burning.
- CLOB-style order placement through /api/trade/execute with concurrent submissions exercising the Serializable retry path.

Usability:

- Single-click wallet connection and swap.
- Pre-trade slippage and minimum-received display computed from lib/amm/swap-math.ts.
- SOL/USD price visible on the trade page with sub-15-second freshness.
- Deviation warning when AMM mid-price diverges from the Pyth/CoinGecko/Jupiter reference by more than the configured threshold.

Reliability:

- SSE streams reconnect cleanly through proxies thanks to the 30-second heartbeat.
- WebSocket relay survives Redis restarts: clients remain connected and resubscribe their pool channel after the bus reconnects.
- Chart endpoint serves correctly under cold cache by falling through to Postgres-side aggregation; warm-cache reads are sub-50 ms.

LIMITATIONS

- **Spot-only execution :** There is no leverage, funding rate, or on-chain liquidation; the Position model approximates positions as one row per fill.
- **Off-chain oracle :** Pyth is consumed only as one of three off-chain price sources for SOL/USD; deviation guards are advisory rather than program-enforced.
- **Single-admin governance :** Both the program upgrade authority and the UI admin gate resolve to a single pubkey.
- **Single-region deployment :** The Redis bus and WebSocket relay run on a single EC2 instance; horizontal scaling would require a managed Redis cluster and stateless relay replicas.
- **No CLOB matching engine :** Although the UI is order-book-shaped, all execution is settled through the AMM; resting limit orders are tracked off-chain only.

FUTURE WORK

- On-chain Pyth attestation inside the swap instruction for hard deviation checks.
- Multisig governance (e.g., Squads) replacing the single-admin pubkey for both the upgrade authority and pool approval.
- An active maker bot driving the MakerBot table to provide two-sided quotes on illiquid pools.
- Cross-program routing that selects between AMM curves and any future on-chain order book for best execution.
- Native support for additional token decimal configurations (6 and 8) with rigorous normalization and property tests.
- Activation of the on-chain treasury split currently tracked off-chain, by adding a treasury vault to the AMM Config PDA.

CONCLUSION

Tradr demonstrates that a tightly scoped DEX can deliver an authentic Solana trading experience under academic constraints. The 9-decimal precision contract removes a class of integer-arithmetic bugs and aligns naturally with WSOL. The escrow-gated lifecycle improves pool quality without giving the admin custody of user funds beyond the review window. The CLOB-style frontend over an AMM execution engine satisfies both retail users (one-click swaps) and power users (charts, order books, positions). A Redis–SSE–WebSocket pipeline provides sub-second trade and price propagation while remaining trivially deployable on a single EC2 instance, and Postgres-side OHLC aggregation keeps charts responsive without pre-computed candle stores. The result is a stack that is comfortable to demonstrate, cheap to host, and structured to scale incrementally.

ACKNOWLEDGMENT

We thank Dr. Manisha V. Pise for guidance and feedback throughout this project, and the open-source authors of Anchor, the constant-product-curve crate, lightweight-charts, and Prisma.

REFERENCES

I. Solana Labs, "Solana Documentation," online manual, accessed 2026.
II. Anchor Framework, "The Anchor Book," v0.31.1, online manual, accessed 2026.
III. D. Little, "constant-product-curve," Rust crate, github.com/deanmlittle/constant-product-curve, accessed 2026.
IV. Pyth Network, "Hermes Price Feeds and On-chain Programs," developer documentation, accessed 2026.
V. SPL Token Library, "Token Program and Associated Token Accounts," developer documentation, accessed 2026.
VI. Vercel, "Next.js 14 App Router," framework documentation, accessed 2026.
VII. Prisma, "PostgreSQL Connector and Transactions," ORM documentation, accessed 2026.
VIII. Redis Ltd., "Redis Pub/Sub," documentation, accessed 2026.
IX. TradingView, "lightweight-charts," v5 documentation, accessed 2026.

AUTHOR INFORMATION

Abhinav Kale, Ninad Bomanwar, Pratik Golder, Deepak Dey, and Chanksh Dubey.

Department of Computer Science and Engineering, Rajiv Gandhi College of Engineering, Research & Technology, Chandrapur, Maharashtra, India