User Stories & On-Chain Requirements of RapidFlow a CLOB on Solana

Part A: Initial User & Function Mapping

1) Manual User Brainstorming

Direct Users:

- Retail traders (casual crypto traders)
- Professional/active traders (day traders, swing traders)
- Market makers (provide liquidity through limit orders)
- High-frequency traders (Bots/algorithmic trading systems)

Indirect Users/Beneficiaries:

- Liquidity providers (passive liquidity)
- Token projects (whose tokens get listed)
- DeFi protocol users (who benefit from better prices)
- Price oracle consumers (rely on accurate pricing data)

Administrators/Moderators:

- Platform developers (@Vdkk07 and @bytehas69)
- Market surveillance operators (monitor for manipulation)
- Token listing reviewers (approve new trading pairs)
- Risk managers (set trading limits, circuit breakers)

Stakeholders:

- RapidFlow token holders (governance participants)
- Institutional partners (potential integrators)
- Solana ecosystem projects (benefit from infrastructure)
- Auditors/security reviewers
- Liquidity mining participants

2) AI-Assisted User Prioritization

Professional/Active Traders

- **Rationale:** These are your core users who will stress-test the order book's speed, matching engine, and real-time execution. They need limit orders, market orders, and

order cancellations - the fundamental CLOB mechanics. Without them, you can't prove the "CEX-level performance" claim.

Market Makers

- **Rationale:** Essential for bootstrapping liquidity - the biggest challenge identified in your adversarial analysis. Market makers place both buy and sell orders, creating depth in the order book. A CLOB without liquidity is useless, so proving you can attract and support market makers is critical for POC viability.

Retail Traders

- **Rationale:** Represent the volume driver and validate the "low fees + instant execution" value prop for everyday users. They primarily execute market orders and simple limit orders. Proving that retail can benefit from your institutional-grade infrastructure demonstrates broad market appeal.

Platform Administrators

- **Rationale:** For POC, you need to demonstrate system initialization, trading pair creation, and basic operational controls. This proves the platform can be deployed, configured, and managed - foundational requirements before any trading happens.

3) Core Function Mapping

Professional/Active Traders

Key Functions

- Place limit orders (buy/sell at specific price)
- Place market orders (buy/sell at best available price)
- Cancel open orders
- View order book depth (bids and asks)
- Check order status (filled, partially filled, open)
- View trade history

Market Makers

Key Functions:

- Place multiple simultaneous limit orders (both sides of book)
- Monitor order fill rates
- View their active orders across the book
- Track profit/loss from fills

Retail Traders

Key Functions:

- Place simple market orders (instant buy/sell)
- Place basic limit orders
- Cancel pending orders
- View current market price
- Check wallet balance (available funds)
- View recent personal trades
- See basic order book (top few price levels)

Platform Administrators

Key Functions:

- Initialize the trading platform
- Create new trading pairs (e.g., SOL/USDC)
- Set trading parameters (tick size, lot size, fees)
- Collect platform fees
- Monitor overall system health
- Emergency halt trading (circuit breaker)

4) Deriving Core POC Requirements

Critical Interaction #1: Trader Places and Executes Orders

- A trader (retail/professional) places a limit order, another trader places a matching market order, and the trade executes with proper settlement.

Critical Interaction #2: Market Maker Provides Liquidity

- A market maker places multiple limit orders on both sides of the order book, creating depth and enabling price discovery.

Key Technical Requirements:

Based on these critical interactions, here are the POC requirements:

For Order Placement & Execution:

- System must accept and validate limit orders (price, quantity, side)
- System must accept and validate market orders
- Matching engine must match orders based on price-time priority
- System must execute trades when orders match
- System must transfer tokens between buyer and seller accounts

- System must update order book state after each trade
- System must handle partial fills (order partially executed)
- System must maintain order history and status

For Market Making & Liquidity:

- System must support multiple open orders per user
- System must efficiently cancel orders without affecting other orders
- Order book must maintain sorted price levels (bids/asks)
- System must handle rapid order placement/cancellation
- System must track available vs. locked balances (funds in open orders)

For Platform Operations:

- System must initialize trading pair accounts
- System must set and enforce trading parameters (fees, decimals)
- System must validate token accounts for trading pairs
- System must collect and distribute trading fees

For DeFi Integration:

- System must expose order book data (best bid/ask, depth)
- System must return execution results to calling programs
- System must handle cross-program invocations (CPI)

Part B — Adversarial Analysis & Granularity Check

1) Critique & Refine User Stories / Requirements:

1. **Coverage Strengths**
    - The user stories accurately represent all key personas (traders, market makers, administrators).
    - The POC focus on *order placement, execution, and liquidity provisioning* directly aligns with proving the project's core value proposition.
    - Technical requirements for matching, execution, and settlement demonstrate good initial granularity for on-chain program planning.

2. **Gaps & Granularity Concerns**
    - **Granularity of order states:** The requirements mention "maintain order history," but do not specify data structures or state transitions (e.g., *open → partially filled → filled/cancelled*). This needs refinement for database/account schema design.
    - **Missing failure scenarios:** Requirements do not specify what happens if an order fails validation or lacks sufficient balance.

3. **Suggestions for Refinement**
    - Add more precise descriptions of state transitions (open, matched, filled, cancelled).
    - Explicitly include system-level error handling (e.g., insufficient balance, expired order).
    - For market makers, focus on *order persistence and cancellation efficiency* instead of full PnL tracking (out of POC scope).

Refined User Stories

1. Trader / Market Maker (Unified Function Set)

- User places a new order specifying price, quantity, and side (buy/sell).
- System validates available balance and order parameters before acceptance.
- Order remains active until matched, cancelled, or expired.
- User can cancel or modify an existing order.
- System executes trades automatically upon a match and updates balances.
- User can view personal open orders and trade history.

2. Market Maker (Liquidity Focus

- User places multiple limit orders on both sides of the order book.
- System maintains active orders and allows batch cancellation.
- User can monitor order fill rates (tracked on-chain through order account state).

3. Administrator

- Initializes trading pair accounts and parameters (tick size, lot size, fees).
- Pauses/resumes trading for maintenance or emergencies.
- Collects and distributes fees to platform vault.

Refined Technical Requirements (Granular View)

Core Trading Logic

- `place_order()` instruction: accepts limit or market order type, validates inputs, checks balances, locks tokens, if matched any existing orders then free locked token and inserts into order book if no matches found.
- Order state transitions: *open* → *partially_filled* → *filled/cancelled*.
- Partial fill support — automatically updates remaining quantity.
- `cancel_order()` instruction: removes active orders, releases locked tokens.
- `settle_funds()` instruction: transfer the free tokens back to the user.

Order Book Management

- Maintain separate on-chain data structures for bid/ask queues sorted by price.
- Efficient insertion/removal via linked-list or heap structure.
- Store per-user order accounts with status tracking.

Token & Settlement System

- Validate token accounts via Solana Token Program.
- Transfer tokens on trade execution using CPI.
- Update vault and fee accounts after each trade.

Administrative Controls

- `initialize()` to create trading pair data account and parameters.
- `set_fee()` to adjust fees or tick sizes.

Rationale for Refinement

- **Clarity:** Removed redundant actions and merged similar user stories to make each atomic and testable.
- **Granularity:** Added detailed state transitions and explicit validation/error-handling requirements to enable technical mapping to Solana accounts and instructions.
- **Focus:** Scoped down aggregator functionality to essential composability proof.
- **POC Realism:** Prioritized features that can be fully demonstrated on-chain within POC constraints.

Part C — Granularity & Clarity Refinement

1) Final Manual Review & Refinement

Professional/Active Trader

1. User places a *limit order* to buy or sell a token at a chosen price.
2. User places a *market order* to buy or sell instantly at the best available price.
3. User cancels an open order that has not been fully filled.
4. User checks the live *order book* to view current buy/sell prices.
5. User views *their active and past orders* to track execution history.

Market Maker

1. Market maker places *multiple limit orders* on both buy and sell sides.
2. Market maker cancels *individual or all* open orders.
3. Market maker monitors *filled and unfilled orders* to adjust strategy.
4. Market maker views *profit and loss summary* based on completed trades.

Retail Trader

1. Retail user places a *market order* to buy or sell tokens instantly.
2. Retail user places a *limit order* to set a specific buy/sell price.
3. Retail user cancels pending orders.
4. Retail user checks *available wallet balance* before trading.
5. Retail user views *basic order book summary* (top prices).
6. Retail user checks *recent trade history* to track completed transactions.

Platform Administrator

1. Admin initializes the *trading platform* and trading pairs (e.g., SOL/USDC).
2. Admin sets *trading parameters* such as tick size, fees, and decimals.
3. Admin collects *platform fees* from trades.
4. Admin monitors *system health and activity logs*.

Final Refined Technical Requirements (Simplified)

Order Placement & Matching

- System accepts and validates *limit* and *market* orders.
- Orders are matched by *price-time priority*.
- Trades automatically transfer tokens between buyer and seller.
- Order book updates in real time after each trade.
- System supports *partial fills* and updates order status.

- Keeps a *record* of all trades and order states.

Market Making

- Each user can keep multiple open orders.
- System allows *fast order placement and cancellation*.
- Orders are sorted by price (highest bid, lowest ask).
- System tracks *available vs. locked balances*.

Platform Operations

- Admin can *initialize trading pairs* and configure parameters.
- System *collects and distributes fees*.
- Validates *token accounts* for both sides of a trading pair.

DeFi Integration

- Order book data (best bid/ask, liquidity depth) is accessible via API or CPI.
- External programs can *place and settle trades* through CPI calls.
- Execution results are returned to the caller program for confirmation.

Appendix: Part C Refinement Log

Before (from Part B)	After (Refined)	Rationale for Change
"User places and executes limit orders (buy/sell at specific price)"	"User places a limit order to buy or sell a token at a chosen price."	Simplified phrasing and separated order placement from execution to maintain atomicity.
"User views order book depth (bids and asks)"	"User checks the live order book to view current buy/sell prices."	Replaced technical jargon ("depth") with a user-friendly phrase for clarity.
"Market maker adjusts spreads dynamically (cancel and replace orders)"	"Market maker cancels or replaces orders to adjust strategy."	Reworded for simplicity; kept the same functional meaning without trading jargon.

“DeFi Aggregator executes trades via API”	“Aggregator executes trades on behalf of users through the CLOB.”	Clarified action flow; replaced generic ‘API’ with explicit interaction context.
“System must handle cross-program invocations (CPI)”	“External programs can place and settle trades through CPI calls.”	Retained technical term (CPI) but added clarity on its functional purpose.
“System must handle partial fills (order partially executed)”	“System supports partial fills and updates order status.”	Simplified while preserving technical accuracy.
“Retail user places basic limit and market orders”	Split into two atomic actions: placing limit and market orders separately.	Enforced atomicity for cleaner mapping to instruction-level logic.
“Admin sets parameters and collects fees”	Split into two actions: “Admin sets parameters” and “Admin collects fees.”	Separation improves modular clarity for implementation and testing.

Part D: Defining Potential On-Chain Requirements

Professional/Active Trader

User Story 1: User places a limit order to buy or sell a token at a chosen price.
Potential On-Chain Requirements:

- Instruction to create a new *Order Account (PDA)* storing: trader, side (buy/sell), price, quantity, timestamp.
- Validation that trader has sufficient *available balance*.
- Lock equivalent tokens in escrow (buy: quote token; sell: base token).
- Insert order into on-chain order book structure (sorted by price-time).

User Story 2: User places a market order to buy or sell instantly at the best available price.
Potential On-Chain Requirements:

- Instruction to process *market order* without creating a persistent order account.
- Matching logic that consumes existing limit orders from the opposite side.
- Update affected limit orders (reduce or close).

User Story 3: User cancels an open order.
Potential On-Chain Requirements:

- Instruction to mark *Order Account* as cancelled.
- Release locked tokens back to trader's wallet.
- Remove order from order book state.

User Story 4: User checks the live order book.
Potential On-Chain Requirements:

- Read-only access to the *Order Book PDA* for price levels and order lists.
- On-chain account storing sorted bid/ask levels (possibly using linked-list PDAs or array-based structure).

User Story 5: User views their active and past orders.
Potential On-Chain Requirements:

- *User Order Index PDA* that references all orders linked to a user.
- Stored fields for order status (open, filled, cancelled, partially filled).

Market Maker

User Story 1: Market maker places multiple limit orders on both buy and sell sides.
Potential On-Chain Requirements:

- Support for batch instruction: multiple order creations in one transaction.
- Efficient account management for multiple open orders (array or PDA seeds).
- Gas-efficient insertion into the order book for each order.

User Story 2: Market maker cancels individual or all open orders.
Potential On-Chain Requirements:

- Batch cancel instruction to close multiple order accounts.
- Logic to release all associated locked funds.

User Story 3: Market maker monitors filled/unfilled orders.
Potential On-Chain Requirements:

- PDA or account field tracking fill percentage.
- Read-only access to order book and user order index.

User Story 4: Market maker views profit/loss summary**.**
Potential On-Chain Requirements:

- On-chain trade record PDA storing execution prices and sizes.
- Off-chain calculation of realized/unrealized PnL (on-chain PnL optional).

Retail Trader

User Story 1: Retail user places a market order to buy or sell instantly.
Potential On-Chain Requirements:

- Simple market order instruction with minimal parameters.
- Matching engine matches best available opposite orders.
- Settlement logic ensures instant token transfer.

User Story 2: Retail user places a limit order.
Potential On-Chain Requirements:

- Same as professional limit order logic (simplified validation).
- Order stored in PDA and locked funds tracked.

User Story 3: Retail user cancels pending orders.
Potential On-Chain Requirements:

- Cancel instruction that unlocks remaining tokens.
- Updates order status and removes from active book.

User Story 4: Retail user checks wallet balance.
Potential On-Chain Requirements:

- Query SPL token accounts for available and locked balances.
- Possibly use a PDA that tracks locked balances for clarity.

User Story 5: Retail user views basic order book summary.
Potential On-Chain Requirements:

- Read-only query to the *Order Book PDA* (top N bids/asks).
- Optionally, an off-chain API cache for speed.

User Story 6: Retail user checks recent trade history.
Potential On-Chain Requirements:

- On-chain trade log or event emission system storing recent fills.
- Off-chain indexer can read from program logs for analytics.

Platform Administrator

User Story 1: Admin initializes the trading platform and pairs.
Potential On-Chain Requirements:

- Instruction to create *Market PDA* for each trading pair.
- Store metadata: base mint, quote mint, tick size, fee rate.

User Story 2: Admin sets trading parameters.
Potential On-Chain Requirements:

- Instruction to update fields on the *Market PDA* (admin-only).
- Access control via admin signer check.

User Story 3: Admin collects platform fees.
Potential On-Chain Requirements:

- Accumulator field in Market PDA for total fees.
- Withdraw instruction allowing admin to collect to a treasury wallet.

User Story 4: Admin monitors system health.
Potential On-Chain Requirements:

- Event logs for system activity (executions, cancellations).
- Optional heartbeat or validator reporting mechanism.
