WALPER DOCS

Project documentation

Walper is the market for on-chain reputation: discover Solana traders, inspect their verified track record, and take a paper-money position on whether it holds up.

Version 0.1.0 Status Prototype Updated 23 September 2026

01

Overview

Walper turns Solana wallets into measurable identities. It ranks real wallets by on-chain trading performance, shows each wallet's verified record (PnL, win rate, activity), and lets anyone open a YES/NO position — with play money — on whether that wallet keeps performing over the next 7 or 30 days.

> **No real money.** Every balance, trade and payout in Walper is paper money. The project exists to test whether markets on trader reputation are a useful product before any conversation about real stakes.

Project status

Working prototype, deployed at walper.markets. It covers Phase 1 (wallet intelligence) and Phase 2 (paper markets) of the concept document. Phase 3 — real money, oracles, compliance — is intentionally out of scope.

Design principle

The hard part of this product is *defining performance* — realized vs. unrealized PnL, cost basis, benchmarking. Walper does not re-derive any of that. It is a thin application layer over data providers that already compute it for Solana, so the codebase stays small and the numbers stay auditable.

02

Features

FEATURE	WHAT IT DOES	ROUTE
Landing page	Marketing page explaining the product; one call to action into the app.	`/`
Discover	Leaderboard of real Solana wallets ranked by on-chain trading performance, with a timeframe filter.	`/app`
Wallet profile	Reputation Score with its breakdown, plus PnL, win rate, per-token breakdown, recent activity and first-seen date; create a market from here.	`/app/wallet/[address]`
Watchlist	Follow wallets and see their 7-day summaries in one place.	`/app/watchlist`
Wallet search	Paste any Solana wallet address in the app header to open its profile, score and — if eligible — create a market on it.	Header, every app page
Markets	Open and settled paper markets, filterable into featured (leaderboard) and community (user-created) markets.	`/app/markets`
Market detail	Implied probability, price history, explicit resolution rules and a trade panel.	`/app/markets/[id]`
Portfolio	Paper balance plus open and settled positions.	`/app/portfolio`
Docs	This document, also available as a PDF.	`/docs`

03

Getting started

Prerequisites

- Node.js 20.9 or later (required by Next.js 16) and npm.
- A Postgres database — the project uses Neon. Required for any page that touches markets.
- Free API keys for Vybe Network, Helius and Birdeye (optional to start — see below).

Install and run

The repository is private — ask the maintainer for access, then:

```
git clone <repository-url>
cd walper
npm install
cp .env.local.example .env.local   # then fill in your keys
npm run dev                        # http://localhost:3000
```

Without API keys the app still runs: every provider call falls back to clearly labelled demo data (an amber banner shows when this is happening), so the whole product can be explored before signing up for anything. The database, however, is required for markets and the portfolio.

Scripts

COMMAND	PURPOSE
`npm run dev`	Start the development server (Turbopack).
`npm run build`	Production build.
`npm run start`	Serve the production build.
`npm run lint`	Run ESLint.
`npm run docs:pdf`	Regenerate public/walper-docs.pdf from this page (dev server must be running).

04

Configuration

All configuration is through environment variables in `.env.local` (gitignored). The committed `.env.local.example` documents each one — never put real keys in it.

VARIABLE	REQUIRED	USED FOR
`DATABASE_URL`	Yes	Neon Postgres connection string: markets, positions, trades and paper balances. Without it, market pages throw.
`BIRDEYE_API_KEY`	For real data	Wallet PnL, win rate and per-token breakdown, and the data every market settles against. Without it, profiles show demo data and markets can't settle and are voided (refunded at cost) 7 days after they close.
`VYBE_API_KEY`	For real data	Top-traders leaderboard and SOL's 7-day price move (the benchmark for “outperform SOL” markets).
`HELIUS_API_KEY`	For real data	Human-readable activity feed and first-seen date on wallet profiles.

`DATABASE_URL` is marked sensitive on Vercel, so `vercel env pull` cannot export it — copy it from the Vercel Storage tab or the Neon dashboard.

05

Architecture

Next.js 16 (App Router) with TypeScript and Tailwind CSS v4, deployed on Vercel. Pages are React Server Components that call provider wrappers and the database directly; client components talk to a small set of route handlers so API keys never reach the browser.

Layers

LAYER	PROVIDED BY
Blockchain data (parsed transactions)	Helius
Indexing and reputation ranking	Vybe Network
Performance engine (per-wallet PnL)	Birdeye
Market state, balances, trades	Postgres (Neon)
Application layer — UI, markets, settlement	This repository

Source layout

```
src/
  app/
    page.tsx              landing page
    docs/page.tsx         this documentation
    app/                  the product: Discover, wallet, watchlist,
                          markets, market detail, portfolio
    api/                  route handlers (see API reference)
  components/             UI pieces; landing/ holds landing-page sections
  lib/
    config.ts             env vars + hasVybe()/hasHelius()/hasBirdeye()
    types.ts              shared domain types
    providers/            vybe.ts, birdeye.ts, helius.ts
    amm/lmsr.ts           market-maker maths (pure, client + server safe)
    db/                   Postgres pool, schema, per-table queries
    market-kinds.ts       the four market templates
    market-seed.ts        keeps the featured-market pool topped up
    market-settlement.ts  settles or voids markets when their window closes
```

```
market-eligibility.ts which wallets can have a community market
solana-address.ts     wallet address validation (client + server)
session.ts            anonymous paper-trading session cookie
watchlist.ts          browser-local watchlist
mock-data.ts          demo-mode fallback data
```

Failure handling

Every provider call is wrapped in `try/catch`. A missing key or failed request never crashes a page: display paths fall back to demo data and log to the server console. Settlement is the exception — it never uses demo data, and leaves a market open rather than resolving it on missing or bad data.

06

Data providers

PROVIDER	ENDPOINTS USED	FREE TIER
Vybe Network	Top-traders leaderboard; token snapshot for SOL's 7-day price	60 requests/min, 25,000 credits/month
Birdeye	`/wallet/v2/pnl/details` — realized/unrealized PnL, win rate, per-token breakdown	“Standard” package; rate-limited on bursts
Helius	Enhanced (parsed) transaction history	Generous for development

Vybe was the original choice for everything, but its per-wallet PnL endpoint is only on the paid Developer plan (confirmed with a live 403 on a free key). Birdeye offers the equivalent on its free package, so wallet PnL comes from Birdeye and the leaderboard stays on Vybe.

Birdeye reports `win_rate` as a 0–1 fraction; Walper scales it to a percentage in one place (`providers/birdeye.ts`).

07

Reputation Score

Every wallet profile shows a **Reputation Score** from 0 to 100: one number summarising how well the wallet has traded over the last 30 days, with the breakdown shown next to it. It is built only from signals Walper can measure reliably, and the full method is published here.

> The score describes **past** trading. It is not a prediction, a rating of the person behind the wallet, or financial advice — and it never decides how a market settles. Markets settle on raw data only.

At a glance

PROPERTY	VALUE
Method version	`v0`
Window	Last 30 days, whichever timeframe tab is open on the profile
Data source	One Birdeye call (`/wallet/v2/pnl/details`) — the same one the profile already makes
Range	0–100, where 50 is average and a wallet with no evidence sits at 50
Freshness	Recomputed on every profile view; the underlying data is cached for up to 5 minutes

Design principles

- **Only reliable signals.** If an input can't be measured accurately for a wallet, that signal is marked “not measurable” and its weight is shared among the others. Nothing is estimated or filled in.
- **Fixed, published curves.** Each signal becomes 0–100 through a fixed formula, not a ranking against other wallets — so a wallet's score doesn't change because other wallets were viewed.
- **Evidence over luck.** Small samples are pulled toward average, so a handful of lucky trades can't produce a top score.
- **Realised gains count more.** Unrealised profit is marked at prices the wallet may not be able to sell at, so it counts at 50.0%.

The three signals

SIGNAL	WEIGHT	WHAT IT MEASURES	INPUT
Performance	45%	How much the wallet made relative to what it put in.	Return on capital: (realised PnL + ½ × unrealised PnL) ÷ total invested
Consistency	30%	How reliably the wallet wins, rather than relying on a few big hits.	Birdeye's win rate, adjusted for sample size
Risk	25%	Whether profit is spread across tokens or depends on one. Higher = less concentrated.	Best token's share of gross profit

How the score is calculated

```
1. return      = (realised + 0.5 × unrealised) ÷ invested
   Performance = 50 + 50 × tanh(return ÷ 0.4)

2. trades      = winning trades + losing trades
   adjusted    = (win rate × trades + 10 × 0.5) ÷ (trades + 10)
   Consistency = clamp((adjusted − 0.15) ÷ 0.7) × 100

3. share       = best token's PnL ÷ sum of all profitable tokens' PnL
   Risk        = clamp((1 − share) ÷ 0.8) × 100

4. blend       = weighted average of the measurable signals
                 (45% / 30% / 25%, re-normalised if one is missing)

5. score       = round(50 + (blend − 50) × trades ÷ (trades + 30))
```

Step 2 adds 10 imaginary trades at a 50% win rate, which barely moves a wallet with thousands of trades but stops a 9-for-10 streak reading as a 90% win rate. Step 5 does the same for the whole score: at 30 trades the score sits halfway between the blend and 50.

Scoring curves

Reference points, computed from the live formulas:

RETURN ON CAPITAL	-40%	-20%	0%	+10%	+20%	+40%	+80%
Performance	12	27	50	62	73	88	98

ADJUSTED WIN RATE	15%	30%	40%	50%	60%	70%	85%
Consistency	0	21	36	50	64	79	100

BEST TOKEN'S SHARE OF PROFIT	≤20%	40%	60%	80%	100%
Risk	100	75	50	25	0

Sample size and confidence

CLOSED TRADES	SHARE OF THE BLEND KEPT	CONFIDENCE LABEL
10	25%	Low
30	50%	Medium
100	77%	Medium
150	83%	High
1,000	97%	High

A wallet with no closed trades, or nothing invested in the window, is not scored — the profile says so instead of showing a number.

Tiers

SCORE	TIER
80–100	Exceptional
65–79	Strong
45–64	Average
30–44	Below average
0–29	Poor

Warnings

Shown under the breakdown. They don't change the score; they tell you how to read it.

WARNING	SHOWN WHEN
Likely automated	More than 3,000 trades in 30 days (100+ a day). Bots and market makers post high win rates by design.
Mostly unrealised	Unrealised gains exceed realised PnL — much of the profit exists only at marked prices.

Worked example

A wallet invested $100,000, realised $18,000, holds $6,000 unrealised, won 58.0% of 120 trades, and its best token made 40.0% of its profit.

STEP	WORKING	RESULT
Return on capital	(18,000 + 0.5 × 6,000) ÷ 100,000	+21.0%
Performance	50 + 50 × tanh(0.210 ÷ 0.4)	74
Adjusted win rate	(0.58 × 120 + 5) ÷ 130	57.4%
Consistency	(0.574 − 0.15) ÷ 0.7	61
Risk	(1 − 0.4) ÷ 0.8	75
Blend	74 × 0.45 + 61 × 0.3 + 75 × 0.25	70.2
Sample-size adjustment	50 + (70.2 − 50) × 120 ÷ 150	**66 — Strong**

What v0 leaves out, and why

SIGNAL	STATUS	REASON
Benchmark (vs. SOL)	Not yet	The free SOL price data covers 7 days; the score window is 30. Comparing mismatched windows would be misleading.
Specialization	Not yet	Needs token categories (memecoin, DeFi, …) that no current provider supplies.

SIGNAL	STATUS	REASON
Activity	Used as confidence	Trade count decides how far the score is trusted (step 5), rather than rewarding volume for its own sake.
Risk, for very active wallets	Often not measurable	Birdeye returns at most 100 tokens, ordered by most recent trade. When a wallet traded more tokens than that, its true best token may be missing, so Risk is skipped for that wallet.

Limitations

- Win rate and PnL are Birdeye's figures; Walper inherits their definitions (including a win-rate denominator that isn't exactly wins + losses).
- The weights and curve anchors are judgement calls, chosen to be easy to explain — not fitted to outcomes. They will be revisited once there is data on how scores relate to later performance.
- A 30-day window can't tell a lasting edge from a good month.
- Scores are computed on demand and not stored, so they aren't yet shown on the leaderboard.
- Any change to the method will increase the version number shown on each score and be recorded here.

08

Paper markets

A market asks a YES/NO question about one wallet's performance over a fixed window. Each share pays $1 if its side wins and $0 otherwise, so a price of $0.62 reads as a 62% implied probability.

Market kinds

KIND	QUESTION	WINDOW	RESOLVES YES WHEN
Outperform SOL	Will the wallet outperform SOL by 10%?	7 days	Wallet return − SOL return ≥ 10 percentage points
Absolute return	Will the wallet return more than 15%?	7 days	Wallet return ≥ 15%
Win rate	Will the win rate stay above 65%?	7 days	Win rate ≥ 65%
PnL target	Will PnL exceed $25,000?	30 days	Realized + unrealized PnL ≥ $25,000

Windows are limited to 7 or 30 days because those are the periods the providers return in a single call. Thresholds are fixed constants in `src/lib/market-kinds.ts`, which also holds each kind's question and resolution-rule text so creation, settlement and the UI never disagree.

Featured and community markets

	FEATURED	COMMUNITY
Wallet	From the current top-25 leaderboard	Any wallet a user searches for, if eligible
Created by	The app, automatically	A user, from the wallet's profile page
Question	One of the four kinds, picked at random	The creator picks one of the four kinds
Labelled	Featured	Community
Sorted	By closing time	Most traded first (when filtered to Community)

Featured markets keep the page populated: whenever fewer than 20 featured markets are open, the markets page tops the pool back up. Community markets don't count toward that floor. Every wallet has at most one open market at a time — trying to create a second takes you to the existing one.

Creating a community market

Search a wallet address in the app header, open its profile, choose “Create market” and pick a question. Because a community market can be on any wallet, creation runs behind guardrails, enforced by the server (`POST /api/markets`), not just the page:

GUARDRAIL	RULE	WHY
Valid address	Must be a real Solana wallet address (32-byte base58). Invalid addresses are rejected before any lookup.	Garbage input used to create markets that could never settle.
Active wallet	At least 10 trades and $500 invested in the last 30 days.	A market on a wallet that barely trades can't be measured meaningfully.
Verified data	If a data key is configured but the wallet's data can't be fetched, creation is refused until it can be.	Never create a market on demo numbers.
Creation limit	5 new markets per session per rolling 24 hours. Being sent to an existing market doesn't count.	Stops spam and protects the providers' rate limits.

The wallet's profile shows the same eligibility check before you try, with the reason when a wallet doesn't qualify. Wallet ownership isn't verified, so someone can create a market on their own wallet and then trade to influence it — acceptable with play money, and labelled as a community market, but a hard blocker for any real-money phase.

Pricing

Prices come from the Logarithmic Market Scoring Rule (LMSR), the standard automated market maker for binary prediction markets without an order book. Its worst-case loss is bounded at `b · ln 2` — about $693 at the liquidity parameter `b = 1000` used here — which is what makes

running unbacked play markets sound. `b` is tuned for visible price movement on $25–500 trades, not validated for production.

Sessions and balances

No signup. On first use, an anonymous session id is stored in an httpOnly cookie (valid one year) and a $10,000 paper balance is created for it. Clearing cookies starts a new balance.

Settlement

Markets settle lazily: there is no scheduled job. The first request that reads a market after its window closes settles it against live data (Birdeye for the wallet, plus Vybe's SOL price for “Outperform SOL”) and pays out every open position in one transaction.

- Market detail pages and the trade endpoint settle their one market before doing anything else, so no one can trade on a decided market.
- List views settle at most 4 due markets per request, one at a time, sharing a single Birdeye lookup per wallet and window. This keeps a burst of due markets inside Birdeye's free-tier rate limit.
- After Birdeye returns HTTP 429, settlement skips Birdeye for 60 seconds instead of retrying into the limit.
- If data is missing, the market stays open and is retried on a later request — it is never settled on a guess. After 7 days it is voided instead (see below).

Void and refunds

A market is **voided**, and every position refunded at its cost basis, when:

- the data provider reports **no qualifying trades** for the wallet in the measured window — a win rate or return over zero trades has no answer, so it isn't resolved NO; or
- the wallet's data **can't be fetched for 7 days** after the window closed, so no market stays open forever.

Voided markets appear under the Settled tab marked “Void · refunded”, with the reason on the market page.

Known issue — late settlement measures the wrong window. The data provider only reports relative windows (“the last 7 days”), not arbitrary dates. Settlement is lazy, so a market settled days after it closed is measured on the most recent 7 or 30 days rather than its own. Found during testing: a market that closed on 2 September, settled on 23 September, was measured on 16–23 September. Scheduled settlement (see Roadmap) is the fix.

Every market page prints its exact resolution rule under “Resolution rules”.

09

Data model

Four Postgres tables, created automatically on first connection (`src/lib/db/client.ts`). Trades and payouts run in real transactions, which is why the app uses Neon's `Pool` rather than its stateless HTTP client.

TABLE	HOLDS	KEY COLUMNS
`users`	Anonymous sessions and paper balances	id, balance_usd
`markets`	One row per market, including LMSR state and settlement result. status is open, settled or void; created_by is “system-seed” for featured markets.	wallet_address, kind, threshold, start/end time, q_yes, q_no, liquidity_b, status, outcome, void_reason, created_by
`positions`	A user's shares in one market	(user_id, market_id), yes_shares, no_shares, cost_basis_usd, settled_payout_usd
`trades`	Append-only trade log; feeds the price-history chart	market_id, side, shares, cost_usd, price, created_at

API reference

Internal JSON endpoints used by the app's client components. They are not a public, versioned API. Endpoints that create a session set the session cookie.

METHOD & PATH	REQUEST	RESPONSE
`GET /api/me`	—	Ensures a session exists. `{ userId, balanceUsd }`
`GET /api/markets`	Query `status=open|settled|void` (optional)	Market list; settles up to 4 due markets first. `{ markets }`
`POST /api/markets`	`{ walletAddress, walletLabel?, kind? }`	Returns the wallet's open market, or creates a community market (random kind if `kind` is omitted). `{ market }`. Errors as `{ error }`: 400 invalid address or kind, 422 wallet not eligible, 429 creation limit reached
`GET /api/markets/[id]`	—	One market, settled first if due. `{ market }` or 404
`POST /api/markets/[id]/trade`	`{ side: "YES"|"NO", budgetUsd }`	`{ shares, newPriceYes, balanceUsd }`; 400 with `{ error }` on invalid input, a closed market or insufficient balance
`GET /api/wallet-summaries`	Query `addresses=a,b,c` (max 50)	7-day PnL summaries. `{ summaries }`

Design system

- Light-first, with a real light/dark toggle. One restrained accent, chartreuse `#C8FF3D`; green and red are reserved for performance, never decoration.
- Typography: Geist and Geist Mono.
- Tokens live in `src/app/globals.css` — defined on `:root` (light) and `.dark` — and are mapped into Tailwind utilities such as `bg-surface` and `text-text-dim`.
- An inline script in the root layout applies the saved or system theme before first paint, so there is no flash of the wrong theme.
- Motion (Framer Motion) is restrained and respects `prefers-reduced-motion`.

Deployment

- Hosted on Vercel (project `walper`), built with `next build`.
- Set `DATABASE_URL`, `BIRDEYE_API_KEY`, `VYBE_API_KEY` and `HELIUS_API_KEY` in the Vercel project's environment variables.
- The database must be hosted (Neon): serverless functions have a read-only, ephemeral filesystem, which is why an earlier local-SQLite version failed in production.
- After changing this page, run `npm run docs:pdf` and commit the regenerated PDF.

Known limitations

- Only four fixed market kinds; no user-defined thresholds, windows or market types.
- No accounts: the paper balance lives in a cookie and the watchlist in browser storage.
- Settlement only happens when someone views a market after its window, so quiet markets can stay open longer than their deadline — and a late settlement is measured on the most recent window, not the market's own (see Paper markets → Void and refunds).
- Search takes a full wallet address only; `.sol` names aren't resolved yet.
- Wallet ownership isn't verified, so community markets can be created on — and influenced by — the creator's own wallet.
- There is no way yet for a wallet's owner to ask for a community market on their wallet to be removed.
- Discover is a single leaderboard, not the concept document's six discovery categories.
- No wallet-vs-SOL performance chart yet (Birdeye's `/wallet/v2/pnl/chart` could provide it).
- Helius sometimes returns blank descriptions for unrecognised transactions; the feed falls back to “<source> transaction”.
- First-seen date is approximated from the latest 100 activity items, not a full history walk.
- Provider failures on display pages fall back to demo data silently (apart from the banner) rather than showing an error state.

Roadmap

NEXT STEP	WHY
Reputation Score v1	Add the SOL benchmark over a matching 30-day window, store scores so the leaderboard can show them, and tune weights against how scores predict later performance.
Scheduled settlement (e.g. Vercel Cron)	Now the top priority: settling within minutes of close is the only way to measure a market's own window, since providers only report relative windows. Also takes provider calls off page loads.
`.sol` name search	People know wallets by name (e.g. toly.sol), not by a 44-character address.
Share links with preview cards	Every created market becomes an invitation — the main traction loop for community markets.
Accounts and a server-side watchlist	Keep balances and follows across devices.
Market-authoring UI	Let users pick wallet, metric, threshold and window.
Wallet-vs-SOL performance chart	Complete the benchmark view the concept document calls for.
Discover categories	Rising, consistent, high-risk and other views once usage shows which matter.
Phase 3: real money	A separate decision requiring wallet curation, an oracle and dispute process, and legal review.