Project SOW — Real-time Multi-Asset Indicator Dashboard (Optimized & Interactive)

Project title

Real-time Indicator & Prediction Dashboard — SOL, BTC, ETH, Gold (XAU)
(with performance-first architecture, lightweight UI, accessible charts and downloadable cumulative PDF)

Overview

Build a highly-performant, lightweight responsive web application showing real-time candlesticks and indicators for SOL, BTC, ETH, XAU. Emphasize low-latency streaming (<1s from ingest), minimized CPU/ memory on clients, progressive loading, and clear, easily-understood chart visuals. Provide an interactive "Bid Simulator" and a single downloadable cumulative PDF report collating charts, signals, and simulator snapshots.

Performance & Lightweight Principles (applied across stack)

• Server-side precompute: compute indicators on backend; stream minimal payloads to clients (latest candle deltas + signals).
• Delta & compressed transport: use binary websocket frames (MessagePack or Protobuf) and gzip brotli for fallback to reduce bandwidth.
• Efficient time-series DB: TimescaleDB with hypertables + continuous aggregates for downsampled retention.
• Client: React+TypeScript with lean bundle (code-splitting, tree-shaking, esbuild/ Vite), no heavy runtime libs.
• Chart library: TradingView Lightweight Charts (tiny footprint) or Canvas-based custom renderer; avoid heavyweight DOM SVG toolkits for main chart.
• Web Workers / Wasm: run any remaining indicator computation in Web Workers; use WASM for CPU-heavy signal computations only if needed.
• Virtualization & lazy load: virtualize long lists (signals, trades), lazy-load optional panels (orderbook, advanced metrics).
• CDN & edge caching: serve static assets and prebuilt downsampled historical tiles from CDN/edge.
• Throttling & coalescing: coalesce UI updates to ~200–500ms for visible renders; do not re-render on every tick.
• Battery & mobile friendly: reduce animation rate and polling for mobile; respect reduced-motion.

UI / Chart UX (interactive & immediately understandable)

• Clean baseline: dark/light theme, high contrast, large fonts for prices/time.
• Single-page layout: main chart center, right-side compact controls (signal badge, simulator, recent signals).
• Chart clarity:
- Minimal default overlays (e.g., EMA(8,34) + Volume) with toggle to add more.
- Use clear color palette with accessible contrasts and small legends.
- Tooltips: show OHLCV, indicator values, signal explanation on hover.
- Projection fan: translucent cone overlay showing median and ±1σ bands for selected horizon.
- Small sparkline heatmap above selector for quick cross-asset comparison.
• Interactive simulator:
- Inline input validation, presets (■1000, 1% slippage), quick toggle between fiat & units.
- Instant feedback: show projected P&L;, probability, and pop-out detail with distribution histogram.
- “What-if” slider on chart for entry price / leverage — dynamically updates fan & P&L.;
• Accessibility:
- Keyboard navigation for asset/timeframe selectors.
- ARIA labels, high-contrast mode, text alternatives for charts (CSV export).
• UX polish:
- Smooth transitions (CSS transforms), avoid expensive layout operations.
- Responsive breakpoints: stacked layout for mobile; hide heavy panels by default.

Architecture & Dataflow (optimized)

• Ingest: exchange websockets → ingest service (Normalize, enrich) → indicator engine → TSDB (TimescaleDB) + streaming buffer (Redis Streams).
• Indicator engine: batch compute for closed candles, incremental updates for partial candle (if required) using efficient numeric libs.
• WebSocket Gateway: lightweight Node Fastify or Go gateway handling binary encode (MessagePack/Protobuf) and client subscriptions; uses Redis for scaling.
• REST API: FastAPI/Express for queries and simulation endpoint. Simulation uses cached precomputed distributions for speed.
• ML (optional): separate Python microservice (for inference only), cache predictions, serve probabilities as

additional fields.
• PDF generation: server-side on-demand renderer that snapshots selected charts & tables into a cumulative PDF (HTML-to-PDF using headless Chromium or server rendering via reportlab wkhtmltopdf).

Optimizations specific to charting & simulator
• Tile-based historical delivery: deliver precomputed downsampled tiles for ranges; client stitches tiles for smooth pan/zoom.
• Aggregation tiers: 1m raw, 5m/15m/1h continuous aggregates; client requests nearest tier for viewport.
• Sparse updates for candles: push only newest candle or small delta; clients append and only re-render current viewport.
• Client-side cache: IndexedDB for recent tiles to enable instant reloads and offline snapshots for the PDF.
• Simulation performance: use precomputed conditional return histograms (binned) per asset/timeframe/signal/volatility bucket — computing projections becomes a table lookup + simple algebra, very fast.
• Memory budget: free old data in client after 1-3 sessions; limit in-memory indicator history.

APIs & Contracts (kept small & fast)
• Binary websocket frames for high-frequency streams.
• REST for heavy queries & PDF generation: POST /api/v1/report/pdf {asset, timeframe, start, end, include_simulations: []} → returns PDF binary.
• Use concise JSON schemas; add pagination for large results.

Security & Ops (concise)
• HTTPS, JWT, backend-only secrets, CORS locked to known origins.
• Rate limiting at gateway; token-based subscription for WS as needed.
• Observability: shard metrics (ingest lag, WS throughput), SLOs for <1s latency on "hot" subscriptions.

Acceptance Criteria (performance-focused)
• Live candle + signal delivery latency <1s under test load (N users baseline test).
• Initial page Time-to-Interactive <2.5s on 4G for desktop simplified view.
• Main bundle size (JS) < 300KB gzipped for core app (defer extra features).
• Simulator response (POST /simulate) < 300ms for rule-based mode.
• PDF generation latency acceptable (server-side): <10s for full-report; provide async job if very large ranges are requested.

Testing & QA additions (performance & UX)
• Synthetic load tests for WS concurrency and ingest throughput (k6, Gatling).
• Lighthouse audits (bundle, accessibility, performance).
• Cross-device test matrix (desktop/mid-low mobile).
• Visual regression tests for chart rendering (pixel-compare snapshots).
• Automated checks confirming indicator values against reference library.

Deliverables & PDF
• Updated lightweight React app (core charts + simulator).
• Backend services and OpenAPI spec.
• Server-side PDF generator + REST entry for cumulative downloadable PDF.
• Seed data & demo scripts.

Developer Handoff notes (quick)
• Prioritize backend precompute + compact binary streaming before client-side indicator work.
• Provide compact typed contracts (TypeScript types / OpenAPI) early to unblock frontend.
• Prototype: first deliver trading view with 1 asset and simulator using cached histograms.

Optional fast wins (1–2 week)
• Use TradingView Lightweight Charts, precompute EMA/SMA server-side, stream only EMA points.
• Implement conditional histogram caching for simulator for immediate sub-300ms responses.
• Use Vite for extremely fast dev builds and small prod bundles.