# Software Requirements Specification (SRS)

## 1. Introduction

### 1.1 Purpose

This SRS defines a **blended** version of your Stock Watch project that adds a **Real‑Time AI Stock Sentiment Dashboard**. The goal is to deliver a portfolio‑ready, end‑to‑end system that: - Tracks live prices and technical signals for user‑defined tickers - Ingests social + news data streams (Reddit, X/Twitter, financial outlets) - Runs **LLM-driven summarization** and **sentiment/emotion analysis** - Surfaces actionable **streaming alerts** and an interactive **React dashboard**

### 1.2 Scope

* **MVP**: Watchlist, price polling/streaming, threshold alerts, daily min/max & rolling averages, basic dashboard.
* **V1**: News/social ingestion, per‑ticker daily news summaries, sentiment tags, event→price correlation timelines.
* **V2**: Real‑time pipelines (Kafka/WebSockets), multi‑model sentiment ensemble, backtesting, paper trading, and explainable AI rationales.

### 1.3 Goals & Non‑Goals

**Goals** - Real‑time(ish) monitoring with sub‑10s price latency (where API allows) - Clear, contextual alerts (“%Δ + reasoning snippet”) - Modular NLP pipeline (swap models without rewriting plumbing) - Clean, maintainable codebase with tests, CI, and observability

**Non‑Goals** - Executing real trades with real capital (paper trading only) - Professional grade market data compliance tooling

### 1.4 Definitions

* **Signal**: Programmatically derived event (e.g., price break, surge in positive sentiment)
* **Sentiment**: Polarity score ∈ [−1, 1], **Emotion**: discrete labels (joy, fear, anger, etc.)
* **Event correlation**: Temporal alignment of content events (news/posts) to price movement windows

### 1.5 References

* Finance data APIs (e.g., Polygon, Alpha Vantage, IEX Cloud, Yahoo Finance/yfinance)
* Social/news APIs or firehoses, RSS/HTML scrapers (respect ToS)
* HF Transformers, LangChain, FastAPI, React, websockets, Kafka/Redpanda

## 2. Overall Description

### 2.1 Product Perspective

Micro‑services style system: 1) **Ingestion** → 2) **Stream/Batch Processing** → 3) **NLP** → 4) **Storage** → 5) **API** → 6) **Dashboard**.

### 2.2 Core Product Functions

* Manage watchlists and thresholds
* Fetch/stream price ticks & compute min/max/rolling averages
* Ingest Reddit/X/news; dedupe, normalize, and persist
* Summarize by ticker/day; sentiment + emotion tagging
* Generate signals from **price** and **sentiment** deltas
* Deliver alerts (UI toasts, email/Discord/Slack, or local notifications)
* Display interactive dashboard with filters and timelines

### 2.3 Users

* **You (dev)** initially; later **casual investors** & **power users**

### 2.4 Operating Environment

* **Backend**: Python 3.11+, FastAPI, LangChain, HF models, Kafka **or** Redis Pub/Sub/WebSockets
* **Frontend**: React + Vite (or Next.js), WebSocket client, Charting lib
* **Storage**: Postgres (OLTP), TimescaleDB (time‑series) or SQLite dev
* **Infra**: Docker Compose dev, GitHub Actions CI, optional cloud deploy

### 2.5 Constraints

* API rate limits and ToS
* No PII collection; comply with site scraping policies
* Keep infra cost near zero; use polling in early phases

### 2.6 Assumptions & Dependencies

* Free/low‑cost APIs are sufficient for demo
* Model inference fits within single‑GPU/CPU budget

## 3. System Features & Requirements

### 3.1 User Stories

* **US1**: As a user, I add tickers to a watchlist and set alert thresholds.
* **US2**: As a user, I see current price, intraday min/max, and 7/30‑day averages.
* **US3**: As a user, I receive an alert when price moves ±X% within Y minutes.
* **US4**: As a user, I see recent news/reddit/X posts relevant to a ticker.
* **US5**: As a user, I see sentiment & emotion tags and a one‑sentence LLM summary.
* **US6**: As a user, I view a timeline aligning content events to price changes.
* **US7**: As a user, I toggle model choices (HF, OpenAI) and thresholds.
* **US8**: As a user, I backtest signals against historical windows.
* **US9**: As a user, I run paper trades with PnL tracking.

### 3.2 Functional Requirements

**MVP** - FR‑M1: Create/read/update/delete (CRUD) watchlists and thresholds - FR‑M2: Fetch prices (poll every N sec) and compute min/max & rolling averages - FR‑M3: Trigger alerts on threshold breaches and persist alert history - FR‑M4: Expose REST endpoints & basic WebSocket for live updates - FR‑M5: Dashboard: table + sparkline + alert toasts

**V1 – AI Insights** - FR‑V1.1: Ingest news/reddit/X via API/RSS/scrape queue with dedup - FR‑V1.2: Classify relevance → per‑ticker - FR‑V1.3: Summarize daily news per ticker via LLM; cache results - FR‑V1.4: Sentiment (polarity) + Emotion multi‑label (HF pipeline or zero‑shot) - FR‑V1.5: Join content sentiments with price windows to emit **contextual signals**

**V2 – Streaming & Trading** - FR‑V2.1: Introduce Kafka/Redpanda **or** Redis Streams for event bus - FR‑V2.2: Replace polling with push where supported; fall back gracefully - FR‑V2.3: Signal backtesting module & paper trading execution engine - FR‑V2.4: Explanations: “Because [headline sentiment↑] within 10m before price↑”

### 3.3 Non‑Functional Requirements

* NFR‑1: Price latency target ≤10s (API‑dependent)
* NFR‑2: P95 dashboard render ≤1.5s for 50 tracked tickers
* NFR‑3: Unit/integration test coverage ≥70% in core services
* NFR‑4: Resilient to source outages; degrade gracefully with cached data
* NFR‑5: Observability: basic metrics, logs, health checks

### 3.4 External Interfaces

* **Finance**: yfinance/Polygon/IEX/Alpha Vantage
* **Social/News**: Reddit API + Pushshift‑like sources, X API (if available), RSS, web scrapers (respect robots.txt)
* **Notifications**: Email/Discord/Slack webhooks
* **Auth**: Simple JWT/session if multi‑user later

### 3.5 Data Model (initial)

* users(id, email, created\_at) (future)
* tickers(id, symbol)
* watchlist(id, user\_id?, name); watchlist\_items(watchlist\_id, ticker\_id, threshold\_pct)
* prices(id, symbol, ts, open, high, low, close, volume)
* content(id, source, url, ts, symbol?, text, lang, author, features\_json)
* nlp\_insights(content\_id, sentiment, emotions[], summary, model\_meta)
* signals(id, symbol, ts, kind, payload\_json)
* paper\_trades(id, symbol, side, qty, price, ts, reason, pnl?)

### 3.6 Security & Compliance

* Store only necessary content fields; avoid personal data
* Attribute sources; add cache TTLs & delete on takedown requests
* API keys via .env, never commit secrets

### 3.7 Error Handling

* Circuit breakers for failing sources
* Retry with exponential backoff; dead‑letter queue for bad records
* Fallback to cached summaries on model/api failure

### 3.8 Observability

* Health endpoints /healthz, /readyz
* App metrics: ingestion lag, model latency, alert rate, ws clients
* Structured logging with correlation IDs

### 3.9 Deployment

* Dev: Docker Compose (db, api, worker, ui, (optional) kafka)
* CI: lint, tests, type‑check, build images
* Prod (optional): Fly.io/Render single‑region, hobby tier

### 3.10 Risks

* API access changes / rate‑limits
* Model drift & hallucinations
* Legal/ToS restrictions on scraping

## 4. Architecture Overview

[Sources: Prices | Reddit | X | News]  
 │ │ │  
 ▼ ▼ ▼  
 Ingestion Jobs / Streams (pollers, webhooks, RSS, scrapers)  
 │  
 ▼  
 Message/Stream Bus (Kafka/Redpanda or Redis Streams) ← fallback: in‑proc queues  
 │  
 ├─► NLP Workers (HF/LangChain): relevance → sentiment → emotion → summary  
 │  
 ├─► Signal Engine: price deltas × sentiment spikes → alerts  
 │  
 └─► Storage (Postgres/Timescale)  
 │  
 ▼  
 FastAPI (REST + WebSockets)  
 │  
 ▼  
 React Dashboard

## 5. Milestones & Roadmap

* **Milestone 0 (Day 1–2)**: Repo, Docker Compose, FastAPI skeleton, Postgres schema
* **M1 (Week 1)**: Price polling + watchlist + threshold alerts + minimal UI
* **M2 (Week 2)**: Content ingestion (one source), dedupe, store raw text
* **M3 (Week 3)**: HF sentiment + LLM summarizer; per‑ticker daily digest
* **M4 (Week 4)**: Signals that combine price & sentiment; UI timelines
* **M5 (Week 5)**: WebSockets for live tiles; basic backtesting
* **M6 (Week 6)**: Paper trading & PnL; model switcher; deploy demo
* **Stretch**: Kafka stream, emotion multi‑label, explainability, multi‑tenancy

## 6. Acceptance Criteria (sample)

* AC‑1: Adding a ticker shows current price, day min/max, 7‑day average
* AC‑2: Price move ±X% within 30m generates an alert visible in UI and stored
* AC‑3: Daily per‑ticker summary visible with top 5 headlines and sentiment chart
* AC‑4: Timeline view shows content events aligned to price deltas
* AC‑5: WebSocket updates arrive for at least one widget in real‑time

# Learn‑as‑You‑Go Curriculum & Build Plan

Designed to minimize context‑switching and deliver visible wins weekly.

## Phase 0 – Foundations (1–2 days)

* Set up **mono‑repo**: /api, /workers, /web
* Docker Compose (db, api, worker, ui). Add Makefile & pre‑commit hooks
* Pick **data API** (yfinance to start), seed tickers table

**Deliverable**: GET /tickers, GET /prices?symbol=TSLA returns mocked/polled data

## Phase 1 – Price MVP (Week 1)

* FastAPI endpoints: watchlist CRUD, thresholds
* Poll prices every 15s; compute min/max & rolling averages server‑side
* Alert engine v0 (threshold only) → persist to signals
* React dashboard v0: table, sparkline, toast on alert, WebSocket for live rows

**Deliverable**: Demo video of watchlist + live alerts

## Phase 2 – Content Ingestion (Week 2)

* One source first (RSS from reputable finance site) → content
* Normalization & dedupe (hash title+url+ts)
* Basic relevance tagging by ticker (symbol mentions + fuzzy match)

**Deliverable**: /content?symbol=AAPL returns fresh items

## Phase 3 – NLP Insights (Week 3)

* HF pipeline for sentiment (distilbert‑based) + zero‑shot emotion (NLI)
* LangChain summarizer for daily per‑ticker digest (cache results)
* Store nlp\_insights; add model metadata & version

**Deliverable**: Dashboard card shows **Sentiment meter** + **1‑line summary**

## Phase 4 – Correlated Signals (Week 4)

* Join **content sentiment spikes** with **price windows** (±10–30m)
* New signal types: SENTIMENT\_SURGE, NEWS\_BREAK\_WITH\_PRICE\_MOVE
* Timeline component aligning events→price

**Deliverable**: Timeline view with filters (source, sentiment, strength)

## Phase 5 – Realtime UX (Week 5)

* Introduce WebSockets broadly; optional **Kafka/Redpanda** (local image) later
* Push updates to widgets; show ingestion lag metric in footer

**Deliverable**: Live‑updating dashboard tiles under 10s lag

## Phase 6 – Backtesting & Paper Trading (Week 6)

* Backtest runner over historical prices + recorded signals
* Paper trading execution engine (market orders) + PnL chart

**Deliverable**: Backtest report + PnL for paper strategy

## Phase 7 – Polish & Deploy (Week 7)

* Auth (local), RBAC later; rate‑limit endpoints
* Add tests (pytest), type hints (mypy), CI checks (GitHub Actions)
* Deploy to Render/Fly or single VPS; add /healthz and uptime checks

## Study Guide by Week

**Week 1 – FastAPI & WebSockets** - CRUD patterns, background tasks, pydantic models, basic WS broadcast

**Week 2 – Data Modeling & Time‑Series** - Postgres + SQLModel/SQLAlchemy, indices for (symbol, ts), materialized views

**Week 3 – NLP Basics** - HF pipelines, tokenization, batching, caching; prompt templates for summaries

**Week 4 – Streaming Concepts** - Event buses (Kafka vs Redis), idempotency, consumer groups, backpressure

**Week 5 – Frontend Data‑Viz** - React query, WS client, virtualization for large tables, charting (area, bar)

**Week 6 – Quant 101** - Rolling stats, event study windows, evaluation metrics (precision/recall of signals)

**Week 7 – Ops** - Logging/metrics, health checks, rate limiting, secret management

## Task Breakdown (High‑Level Kanban)

* **Backlog**: APIs selection, schema finalize, UI wireframes, alert rules
* **Doing**: Price poller, watchlist CRUD, threshold engine
* **Review**: WS push, sentiment pipeline, timeline join
* **Done**: MVP table/sparkline, alerts storage, content normalization

## Testing Strategy

* Unit: utilities, NLP scoring functions, alert rules
* Integration: ingestion→storage→API→UI
* E2E: Cypress/Playwright smoke (add ticker, force price change, see alert)

## Future Enhancements

* Multi‑tenant auth, user preferences
* Ensemble sentiment (finetuned finance models + zero‑shot)
* Anomaly detection on volume & volatility
* LLM‑authored **explanations** with source links and confidence
* Notebook exports for analysts