**# Primary objectives (in this order)**

1. Homepage graphs rewrite: replace all placeholder/mock chart data on the homepage graph components with real sentiment analysis data from the database/current sources. Build the end-to-end path:
   * Data: identify or implement sentiment metrics (e.g., score, label, confidence, trend), time bucketing, filters, and aggregation windows.
   * Backend/API: expose typed endpoints/queries that return just-enough chart-ready series with caching.
   * Frontend: update chart components to consume real data; add loading/skeleton, empty, and error states; keep visuals consistent; no placeholder data remains.
   * Ops: feature flag for safe rollout and an easy fallback to the old view (without placeholder data in production).
   * Docs/tests: document the data contract, add robust tests, and include before/after screenshots/gifs.
2. Code golf-first refactor/rewrite across the codebase: minimize code size/line count while preserving behavior and performance. Prefer shorter constructs, expression-level refactors, deduplication, and consolidation.
3. Developer empathy: add unusually thorough, junior-friendly comments and docstrings that explain the “why” and “how” line-by-line for tricky parts. Comments can be verbose even if code is golfed.
4. Reorganize the project for clarity and scalability: sensible module boundaries, folder structure, naming, and dependency hygiene. Provide a Refactor Map from old -> new.
5. Comprehensive documentation: top-level README, architecture overview, ADRs, API docs, module docs, setup/run/debug guides, contribution guidelines, changelog, and runbooks.
6. Test coverage and quality gates: unit/integration/e2e tests, coverage thresholds, CI automation.
7. Security, performance, accessibility, and observability reviews + fixes and recommendations.
8. Suggestions for improvements: produce a prioritized backlog with impact/effort and quick wins vs. epics.

**# Guardrails and principles**

* Preserve external behavior and public APIs by default. If a change is breaking, gate it behind a feature flag or propose a v2 plan.
* Code golf is prioritized, but not at the expense of correctness, security, or major performance regressions.
* Inline comments can be long and explanatory; code should be compact. Prefer single-expression functions, early returns, composable utilities, destructuring, concise conditionals/ternaries, compact iteration, and small helpers.
* Keep license headers and attributions intact.
* Secrets: remove from code; use env vars and document in .env.example.

**# Deliverables checklist**

* Execution plan: a short, actionable plan with phases and approval checkpoints.
* Refactor Map (refactor\_map.md): maps old files/symbols to new locations/names; lists removed/merged modules.
* Heavily-commented, golfed code: rewrite/refactor modules with compact code and rich comments/docstrings. Include rationale for non-obvious one-liners.
* Documentation:
  + README: getting started, scripts, env, architecture summary, common tasks.
  + Architecture: system overview, diagrams (Mermaid), module boundaries, data flow, dependency graph.
  + ADRs: key decisions and tradeoffs.
  + API docs: OpenAPI/TypeDoc/JSDoc/Pydoc (as appropriate).
  + CONTRIBUTING.md: branching, commit/PR style, code review, coding standards, code golf conventions.
  + CHANGELOG.md following Keep a Changelog.
  + RUNBOOKS: ops, incident handling, rollback, migrations.
  + SECURITY.md: threat model, dependency policy, secret management, SAST/DAST.
* Testing:
  + Add/upgrade unit, integration, and e2e tests with clear fixtures/mocks.
  + Achieve coverage threshold [target %] and protect critical paths.
  + Snapshot tests for UI components; contract tests for APIs.
* CI/CD:
  + Pipeline with lint/test/typecheck/build/security scan.
  + Caching and parallelization for speed.
* Performance & bundle:
  + Bundle analysis, size budgets, code-splitting, tree-shaking, lazy-loading, caching headers.
  + Profiling reports and perf test cases where relevant.
* Accessibility: ARIA, keyboard nav, focus management, color contrast, semantic markup; an a11y test checklist.
* Observability: structured logging, metrics, optional tracing hooks; error handling strategy; SLO/SLI suggestions.
* Security: secret scanning, dependency auditing, SSRF/XSS/CSRF protections, input validation, authz checks, HTTP headers.
* Database & migrations: schema review, migrations, seed data strategy, rollback plan.
* DevEx:
  + Formatter, linter, type checking (TS/mypy), pre-commit hooks, Makefile/Taskfile, devcontainer or Docker Compose.
  + Scripts for local dev: install, start, test, lint, typecheck, coverage, build, watch, storybook (if UI).
* Improvement backlog: a prioritized list with Impact (H/M/L), Effort (H/M/L), RICE score (optional), and rationale.
* Final report: what changed, why, risk, follow-ups, before/after metrics (size, startup, test time), and next steps.

**# Process and working mode**

1. Discovery
   * Auto-detect stack, frameworks, build tooling, and architecture. Inventory the repo: dependencies, scripts, services, entry points, configs, env use, and dead code.
   * Identify critical paths and public APIs.
   * Output a short status report and any questions/assumptions; wait for approval if needed.
2. Plan
   * Propose phased plan (safe small PRs). Include a rollback strategy and test strategy. Request clarifications for ambiguous areas.
3. Execute in small batches
   * For each batch:
     + Present a summary of files to change and reasons.
     + Apply changes. Provide diffs or PR-ready changesets with commit messages.
     + Run tests/lint/typecheck/build; report results, sizes, and any new warnings.
     + Update docs incrementally.
4. Validate
   * Provide before/after metrics, bundle sizes, perf snapshots, coverage deltas.
   * Run smoke tests and critical-path e2e tests.
5. Handover
   * Produce final docs, Refactor Map, backlog, and release notes.

**# Code golf conventions (apply aggressively, with guardrails)**

* Prefer:
  + Expression-oriented code: ternaries, nullish coalescing, optional chaining, destructuring, short-circuiting.
  + Compact higher-order functions and comprehensions over verbose loops.
  + Small utility helpers to remove repetition.
  + Early returns, elided braces where safe, inlined constants when not repeated.
  + Functional composition over verbose intermediates.
  + Collapsing trivial wrappers, merging files with few exports.
* Avoid:
  + Obfuscation that harms security or makes debugging impractical.
  + Magic numbers without a comment or named constant nearby.
  + Performance regressions from overly clever constructs.
* Compensate with comments:
  + For every non-obvious one-liner, add a comment explaining it like you’re teaching a junior dev.
  + Include examples in comments for tricky functions.

**# Commenting and documentation style**

* Add module headers explaining purpose, inputs/outputs, invariants, and complexity notes.
* Use language-appropriate docstrings (e.g., JSDoc/TypeDoc for TS/JS; Pydoc for Python).
* For core algorithms/utilities, include “Why this approach?” and “Alternatives considered.”
* Where you compress logic, add line-by-line comments that expand the logic in plain English.
* Add TODOs only when also creating backlog items; link to the tracked issue.

**# Project organization and hygiene**

* Propose and implement an improved folder structure (explain it in docs).
* Remove dead code and unused deps (with justification in PR).
* Add .editorconfig, formatting, and lint rules aligning with golf goals (e.g., allow concise one-liners while enforcing safety).
* Create .env.example and document required env vars. Never commit secrets.
* If monorepo is appropriate, propose workspaces (pnpm/npm/yarn) + build orchestration (Turborepo/Nx) and justify.

**# Security and compliance**

* Add dependency audit step and lockfile updates.
* Secret scanning and banned patterns.
* Add secure headers, CSRF/XSS mitigations, strict input validation, and least-privilege for tokens/keys.
* Note any license or compliance concerns.

**# Performance and UX**

* Measure and reduce bundle size; document size budgets.
* Optimize critical rendering path, caching, and network requests.
* Add lazy loading, code-splitting, and image optimization where relevant.
* Accessibility fixes and test checklist.

**# Observability and operations**

* Standardize error handling; ensure user-safe messages and developer-verbose logs.
* Add log levels, structured logs, and basic metrics/tracing hooks.
* Define SLO/SLI proposals and an error budget idea for future adoption.

**# Testing and CI/CD**

* Strengthen unit and integration tests; add e2e if missing.
* Add coverage thresholds in CI; fail on regressions.
* Lint/typecheck/test/build/security scan in CI with caching.
* Include smoke tests for deploys and rollback instructions.

**# Output format and patching**

* If you can write to the repo, make changes directly and include a running log of commits with messages.
* If not, produce unified diffs per file with context, ready to apply via patch, or provide PR-ready chunks.
* After each batch, include:
  + Files changed
  + Diffs
  + Test/lint/typecheck/build results
  + Bundle size deltas (frontend)
  + Notes for reviewers
* Always update the Refactor Map and docs as you go.

**# Assumptions to confirm**

* Supported runtime versions and browser targets.
* Current CI provider.
* Deployment method (containerized? serverless? PaaS?).
* Database migration policy and data retention requirements.
* Any third-party API constraints or SLAs.

**# Start now**

1. Scan the repository and output:
   * Detected stack summary
   * Risky hotspots and low-hanging fruit for golf/refactor
   * Proposed high-level plan with 3–6 batches
   * Any blocking questions
2. After approval, begin Batch 1 with minimal-blast-radius changes:
   * Set up lint/format/typecheck, basic CI, and code golf conventions
   * Add Refactor Map and initial docs skeleton
   * Refactor 1–2 core modules with golf + junior-friendly comments
   * Report results and propose Batch 2

**# Branching/commit/PR conventions**

* Branch: refactor/golf-[area]-[short-desc]
* Commits: Conventional Commits (feat, fix, refactor, docs, test, chore, perf, build, ci), present tense, scoped.
* PR: include summary, before/after, screenshots (if UI), metrics, checklists, and migration notes.

**# Notes**

* If any instruction conflicts, prioritize: correctness > security > performance > code golf > developer convenience > style.
* Explicitly document any tradeoffs chosen.

brākTrād Five-Batch Execution Plan

Clarified Operating Assumptions

* Scheduler cadence remains five-minute headline pulls and hourly hygiene, per [TaskScheduler.start()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:20) orchestrating [TaskScheduler.\_fetch\_headlines\_task()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:67) and [TaskScheduler.\_cleanup\_task()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:80).
* Finviz Elite sourcing stays automated through [FinvizClient](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:24) methods such as [FinvizClient.fetch\_portfolio\_headlines()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:63) and CSV fallback parsing, with dedupe handled by [HeadlineDeduplicator](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:781).
* Groq and OpenRouter model catalogs are fetched dynamically through [ModelFetcher.get\_groq\_models()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/model_fetcher.py:47) and [ModelFetcher.get\_openrouter\_models()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/model_fetcher.py:95), with persisted keys bootstrapped by [lifespan()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/main.py:50).
* Deployment remains Docker-first, leveraging [docker-compose.yml](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/index.html?id=3d70718a-d8d6-42e8-9546-e113a7e083bf&parentId=2&origin=5ad0dafb-ef35-4b14-811e-2d8c123a2651&swVersion=4&extensionId=kilocode.kilo-code&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView) for Postgres, Redis, backend, and frontend orchestration.
* The returns router stays active, with endpoints such as [get\_historical\_returns()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/routers/returns.py:62) and [get\_ticker\_sentiment\_returns()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/routers/returns.py:195) forming the analytics contract.

A screenshot of a computer

AI-generated content may be incorrect.

2025-10-27 Batch 1 Platform Stabilization

2025-10-29

2025-10-31

2025-11-01

2025-11-03 Batch 1 Batch 2 Intelligence Pipeline

2025-11-05

2025-11-07 Batch 2 Batch 3 Analytics Activation

2025-11-09

2025-11-11

2025-11-13 Batch 3 Batch 4 Experience & Rollout

2025-11-15

2025-11-17

2025-11-19

2025-11-21 Batch 4 Batch 5 Release Hardening

2025-11-23

2025-11-25

2025-11-27 Batch 5

2025-11-29

2025-12-01

2025-12-03

Batch 1 – Data Ingestion & Scheduler Hardening

**Goals & Deliverables**

* Productionize headline ingestion with retriable Finviz pulls, resilient parsing, and Redis-aware dedupe caching.
* Formalize scheduler control plane (pause/resume, metrics, alerting) while preserving 5-minute/60-minute cadences.
* Establish ingestion observability dashboards and alert thresholds.

**Key Changes**

* Extend [TaskScheduler.\_fetch\_headlines\_task()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:67) to orchestrate Finviz retrieval via [FinvizClient.fetch\_portfolio\_headlines()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:63) and store results through a new ingestion service module.
* Wire cache invalidation hooks inside [CacheManager](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/cache.py:13) for headline, analytics, and ticker caches.
* Add ingestion failure telemetry and rate-limit metrics in [FinvizClient](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:24) and [RateLimiter.acquire()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:748).
* Introduce scheduler management endpoints in [backend/main.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/main.py:50) (e.g., /api/system/scheduler) with feature-flag toggles persisted through [Settings](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/config.py:9).

**Feature Flags / Rollout**

* INGESTION\_V2\_ENABLED (env + persisted in UserSettings) to gate the new Finviz ingestion flow.
* SCHEDULER\_CONTROL\_API flag to expose scheduler administration UI only in staging initially.

**Risks & Rollback**

* Finviz throttling spikes: mitigate with adaptive back-off and fallback scrapes; rollback by toggling INGESTION\_V2\_ENABLED false.
* Redis downtime: fallback to in-memory cache with capped history; scheduler can be paused via new control API.

**Tests & Documentation**

* Contract tests for ingestion service mocking Finviz responses (success, empty, rate-limited).
* Integration test verifying scheduler start/stop semantics.
* Update runbooks with scheduler operations and ingestion SLOs.

Batch 2 – Sentiment Pipeline & Dynamic Model Configuration

**Goals & Deliverables**

* Guarantee Groq/OpenRouter catalog freshness and fail-safe fallbacks while keeping user-specific keys reconciled.
* Harden sentiment analysis concurrency and background job safety.
* Document model-selection UX and provide administrative overrides.

**Key Changes**

* Refine [ModelFetcher.refresh\_all\_models()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/model_fetcher.py:190) to use shared async client pools, TTL-aware caching, and instrumentation.
* Enhance [ensure\_settings\_loaded()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/routers/sentiment.py:22) and [analyze\_headline\_task()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/routers/sentiment.py:97) to re-resolve models on each batch run, including partial failure handling.
* Persist feature toggles for preferred models through [Settings.selected\_models](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/config.py:322) and add new endpoints in [backend/routers/settings.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/routers/settings.py:1) (to be updated) for admin-specified defaults.
* Update Docker env templates to surface Groq/OpenRouter key sourcing during container bootstrap.

**Feature Flags / Rollout**

* MODEL\_CATALOG\_DYNAMIC\_REFRESH enabling real-time catalog updates.
* SENTIMENT\_QUEUE\_HARDENING gating the new concurrency controls before production rollout.

**Risks & Rollback**

* Misconfigured user keys causing empty catalogs: provide staged validation endpoint; fallback to existing cached lists by disabling MODEL\_CATALOG\_DYNAMIC\_REFRESH.
* Background job saturation: guard with asyncio semaphore; rollback by reverting to previous single-threaded execution path.

**Tests & Documentation**

* Async unit tests for model fetcher covering TTL expiry and fallback lists.
* Load test of sentiment batch endpoint to confirm queue back-pressure.
* Architecture doc updates describing sentiment model selection flow with a sequence diagram.

Batch 3 – Returns & Analytics Activation

**Goals & Deliverables**

* Activate returns API auto-recalculation triggered by ingestion events, ensuring analytics dashboards pull live data.
* Normalize return calculations and sentiment comparisons for homepage graphs, replacing placeholder datasets.
* Provide return-specific caching and hydration routines to support chart queries.

**Key Changes**

* Extend [calculate\_sentiment\_returns()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/routers/returns.py:248) to integrate with ingestion events and persist deltas via [calculate\_returns()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/market_data.py:70).
* Introduce a scheduler job hook invoking returns recomputation nightly using [TaskScheduler](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:11) with new cadence configurations.
* Update analytics endpoints in [backend/routers/analytics.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/routers/analytics.py:1) to support richer filters (horizon, confidence buckets) and ensure caching with [CacheManager.cached\_result](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/cache.py:169).
* Create a homepage returns aggregation endpoint under [backend/routers/analytics.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/routers/analytics.py:148) (extended) to replace placeholder data.

**Feature Flags / Rollout**

* RETURNS\_ROUTER\_V2 toggling the new aggregation contract for frontend consumers.
* ANALYTICS\_CACHE\_LAYER controlling cache-backed responses for safe rollback.

**Risks & Rollback**

* Calculation drift due to inconsistent market data: schedule validation job; disable RETURNS\_ROUTER\_V2 to revert to legacy behavior.
* Cache staleness: leverage TTL monitoring; rollback by disabling ANALYTICS\_CACHE\_LAYER.

**Tests & Documentation**

* Snapshot tests for returns endpoints using seeded market data.
* Regression tests comparing computed returns across horizons.
* Documentation updates for API contracts and data schemas consumed by the frontend.

Batch 4 – Frontend Integration & Experience Enhancements

**Goals & Deliverables**

* Replace placeholder charts with live sentiment/returns data, ensuring responsive UX, skeleton states, and error boundaries.
* Establish feature-flag-driven rollout across views (homepage, portfolio, returns dashboards).
* Improve accessibility, real-time updates, and cross-device performance.

**Key Changes**

* Wire homepage charts in [frontend/src/routes/+page.svelte](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/frontend/src/routes/+page.svelte:1) to consume new analytics endpoints with proper suspense states.
* Update returns dashboard in [frontend/src/routes/returns/+page.svelte](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/frontend/src/routes/returns/+page.svelte:1) to support horizon filtering and confidence thresholds.
* Implement global fetch utilities and model-selection UI snippets under a new shared module (e.g., frontend/src/lib/api) fed by backend flags.
* Connect WebSocket live updates via [websocket\_endpoint](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/main.py:270) for sentiment and returns deltas, gated by config.

**Feature Flags / Rollout**

* UI\_RETURNS\_V2 flag to progressively enable live charts.
* LIVE\_UPDATES\_ENABLED controlling WebSocket-driven updates.

**Risks & Rollback**

* Frontend regressions in SSR/CSR hydration: add staging smoke tests; rollback by disabling UI\_RETURNS\_V2.
* WebSocket overload: throttle updates and fallback to polling when LIVE\_UPDATES\_ENABLED is off.

**Tests & Documentation**

* Cypress/Playwright UI smoke tests verifying chart rendering, loading, and error states.
* Visual regression snapshots for key pages.
* Update UX playbooks describing new data states and accessibility improvements.

Batch 5 – Deployment, Observability & Operational Readiness

**Goals & Deliverables**

* Harden Docker-first deployment, extend CI/CD, and document rollback/runbooks.
* Set up observability baselines (metrics, logs, alerts) for ingestion, sentiment, and returns pipelines.
* Deliver comprehensive documentation, Refactor Map, and release artifacts.

**Key Changes**

* Update docker-compose overrides and production Dockerfiles (e.g., [backend/Dockerfile](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/Dockerfile:1), [frontend/Dockerfile](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/frontend/Dockerfile:1)) for multi-stage builds, health checks, and minimal images.
* Integrate CI pipeline steps for lint/test/build across Node and Python using scripts defined in [package.json](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/index.html?id=3d70718a-d8d6-42e8-9546-e113a7e083bf&parentId=2&origin=5ad0dafb-ef35-4b14-811e-2d8c123a2651&swVersion=4&extensionId=kilocode.kilo-code&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView) and [frontend/package.json](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/frontend/package.json:1).
* Enhance observability by exposing Prometheus metrics in [backend/main.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/main.py:208) with pipeline-specific counters and logs.
* Finalize documentation set: README, architecture diagrams, ADRs, runbooks, backlog.

**Feature Flags / Rollout**

* OBSERVABILITY\_ENHANCED to enable expanded metrics/logging pipelines.
* DOCKER\_BUILD\_OPTIMIZED to toggle new build args until validated across environments.

**Risks & Rollback**

* CI failures due to environment drift: maintain baseline pipeline as fallback; disable new CI steps via configuration toggle.
* Docker image compatibility issues: keep previous tags accessible; revert DOCKER\_BUILD\_OPTIMIZED for immediate rollback.

**Tests & Documentation**

* CI smoke pipelines validating lint/unit/integration/e2e for both stacks.
* Load/perf testing scripts capturing ingestion and sentiment throughput.
* Comprehensive documentation push including Refactor Map, runbooks, security hardening notes.

Risk Register

| **Risk** | **Impact** | **Mitigation** | **Batch** |
| --- | --- | --- | --- |
| Finviz API rate-limit or schema change | High | Adaptive retry, fallback scraping, feature-flag rollback | Batch 1 |
| Redis/cache outages impacting ingestion | Medium | Graceful degradation + cache bypass fallback | Batch 1 |
| Dynamic model catalog returning empty list | Medium | Pre-flight validation endpoint, fallback lists | Batch 2 |
| Sentiment job backlog causing latency | High | Async semaphore, queue metrics, auto-pause control | Batch 2 |
| Returns recalculation inconsistency | High | Data validation jobs, revert via flag | Batch 3 |
| Frontend rollout causing UX regressions | Medium | Feature-flag gating, beta environment, visual tests | Batch 4 |
| Docker/CI deployment regressions | Medium | Parallel pipelines, retain prior image tags | Batch 5 |

Dependency Checklist

* **Batch 1 prerequisites**: Valid Finviz credentials stored via [Settings](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/config.py:9); Redis service healthy per [docker-compose.yml](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/index.html?id=3d70718a-d8d6-42e8-9546-e113a7e083bf&parentId=2&origin=5ad0dafb-ef35-4b14-811e-2d8c123a2651&swVersion=4&extensionId=kilocode.kilo-code&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView).
* **Batch 2 prerequisites**: Groq/OpenRouter keys persisted in UserSettings and database migrations up to [20251010\_add\_horizon\_vote](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/alembic/versions/20251010_add_horizon_vote.py:1).
* **Batch 3 prerequisites**: Market data ingestion up-to-date, returns feature flag seeded, scheduler control from Batch 1 live.
* **Batch 4 prerequisites**: API contracts from Batches 2–3 stabilized, frontend env variables (e.g., VITE\_API\_PROXY\_TARGET) aligned, WebSocket endpoint authenticated.
* **Batch 5 prerequisites**: All feature flags configured, CI secrets/registries provisioned, observability stack endpoints (Prometheus/Grafana) available.

Summary: Delivered five detailed batches covering ingestion hardening, sentiment model orchestration, returns analytics activation, frontend rollout, and deployment hardening; provided corresponding feature flags, risk register, and dependency checklist for stakeholder approval.