**Subtask Results**

The directory Refactor-Oct2025 has been created (if needed) and the file Refactor-Oct2025/batch1\_blueprint.md now contains the provided Markdown. The file content is exactly as specified:

**Batch 1 – Data Ingestion & Scheduler Hardening Blueprint**

1. Objective Summary

* Productionize Finviz headline ingestion with resilient retries, deduplication, and persistence to unlock downstream analytics.
* Harden the background scheduler so ingestion cadence can be controlled, monitored, and paused safely without redeployments.
* Instrument ingestion and scheduler activity with metrics/logging, aligning with Batch 1 acceptance criteria and setting the foundation for Batches 2–5.

2. Scheduler Lifecycle Enhancements

2.1 Task orchestration

* Extend [TaskScheduler.start()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:20) to:
  + Load feature flags from configuration and skip registering new tasks when INGESTION\_V2\_ENABLED is disabled.
  + Register the ingestion loop via asyncio.create\_task that delegates to the new ingestion service entry point (see §3).
  + Record task handles in self.tasks keyed by descriptive names ("headline\_ingestion\_v2", "cleanup\_v2").
* Replace the placeholder logic in [TaskScheduler.\_fetch\_headlines\_task()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:67) with:
  + A guard to short-circuit when the feature flag is off.
  + Context acquisition from AsyncSessionLocal and invocation of [FinvizClient.fetch\_portfolio\_headlines](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:63) through the ingestion service API for each configured portfolio ID.
  + Structured logging (task, portfolio\_id, duration\_ms, headline\_count).
* Update [TaskScheduler.\_cleanup\_task()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:80) to call a cleanup helper in the ingestion service that prunes expired cache entries, stale headlines, and marks stale analytics aggregates.

2.2 Error handling & retries

* Wrap each iteration of [TaskScheduler.\_run\_periodically()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:53) with exponential backoff (e.g., jittered sleep on failure capped at 5× interval) while preserving the base cadence.
* Annotate ingestion calls with rate-limit awareness:
  + Surface 429/timeout responses from [RateLimiter.acquire()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:748) as warnings and reschedule without crashing the loop.
  + Emit success/failure counters to Prometheus (see §5).
* Ensure cancellations are graceful by awaiting outstanding ingestion tasks during [TaskScheduler.stop()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:38).

2.3 Control surface & lifecycle

* Store scheduler state (running, last\_success\_at, consecutive\_failures) on the instance so the control API can query it.
* Persist a lightweight heartbeat in Redis with cache\_manager.set("scheduler:headline:last\_heartbeat", ...), allowing external monitors to detect stalls.
* Allow dynamic interval adjustment through feature-flag overrides (two environment variables or database-backed overrides).

3. Ingestion Service Module (backend/services/ingestion.py)

3.1 Responsibilities

* Provide a cohesive API to fetch, normalize, deduplicate, and persist Finviz headline payloads while coordinating cache invalidation.
* Abstract away direct Finviz client usage so routers and scheduler interact through a single facade.
* Surface post-ingestion signals (counts, dedupe stats) for telemetry.

3.2 Public API

* [ingest\_portfolios](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/ingestion.py): async entry point invoked by the scheduler; accepts a list of portfolio IDs, optional explicit API key, and async session handle. Internally iterates portfolios, acquiring headlines concurrently (bounded semaphore) and consolidating results.
* [ingest\_single\_portfolio](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/ingestion.py): helper returning a structured result (fetched, stored, deduped, errors) for one portfolio, leveraging [FinvizClient.fetch\_portfolio\_headlines](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:63).
* [cleanup\_expired\_records](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/ingestion.py): invoked by the hourly cleanup job to purge headlines older than configured retention and clear cache keys.
* [invalidate\_downstream\_caches](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/ingestion.py): central cache invalidation hook shared by write paths (see §4).

3.3 Workflow

1. Resolve runtime configuration (portfolio list, API key, dedupe thresholds) from [Settings](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/config.py:9) and persisted overrides (see [backend/models.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/models.py)).
2. Acquire the Finviz rate limiter via the client and fetch raw headlines.
3. Deduplicate using [HeadlineDeduplicator](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:781) before persistence.
4. Upsert records into Headline/related tables inside a single transaction (reuse AsyncSessionLocal from [database.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/database.py)).
5. Trigger cache invalidation once commits succeed.
6. Emit telemetry payload to the scheduler control bus (structlog + metrics).

3.4 Data integrity & transactions

* Use session-scoped transactions to write Headline, HeadlineSource, and any derived aggregates atomically.
* Guard against concurrent ingestion by locking on Redis key ingestion:portfolio:{id}:lock with short TTL to avoid double-processing.
* Capture dedupe hashes using the existing headline\_hash column, preventing duplicates before hitting the database.

4. Cache Invalidation & Refresh Strategy

* Generate cache keys with [CacheManager.cache\_key()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/cache.py:20) to ensure consistent hashing for analytics/ticker responses.
* After successful persistence, call [invalidate\_downstream\_caches](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/ingestion.py) to:
  + Remove headline-related keys via [CacheManager.delete\_pattern()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/cache.py:87) (e.g., headlines:*, analytics:summary:*, returns:ticker:\*).
  + Optionally prewarm hot caches by reusing [CacheManager.get\_or\_set()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/cache.py:130) with fresh data loaders.
* Expose cache hit/miss metrics to validate improvements (see §5).

5. Telemetry & Observability

* Emit structured logs through structlog in:
  + Ingestion success/failure paths (status, portfolio\_id, count, duration\_ms).
  + Scheduler control events (action, actor, interval).
* Register Prometheus metrics in [backend/main.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/main.py:208):
  + Counter: braktrad\_ingestion\_fetch\_total{status, source}.
  + Histogram: braktrad\_ingestion\_duration\_seconds{portfolio\_id}.
  + Gauge: braktrad\_scheduler\_last\_success\_timestamp.
* Surface Redis heartbeat timestamp in a debug endpoint to assist operators.

6. Feature Flags

* INGESTION\_V2\_ENABLED:
  + Declared in [Settings](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/config.py:9) with default False in production.
  + Persist overrides via a JSON column on the user settings record (see [backend/models.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/models.py)).
  + Checked inside [TaskScheduler.\_fetch\_headlines\_task()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:67) and ingestion service entry points.
* SCHEDULER\_CONTROL\_API:
  + Controls exposure of admin endpoints and UI toggles.
  + Evaluated during FastAPI router registration inside [backend/main.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/main.py:148) before including the control router.
  + Document expected propagation through environment variables (ENVIRONMENT, FEATURE\_FLAGS).

7. Scheduler Control API Surface

* Introduce backend/routers/system\_scheduler.py with endpoints:
  + POST /api/system/scheduler/start – triggers [TaskScheduler.start()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:20) when stopped.
  + POST /api/system/scheduler/stop – calls [TaskScheduler.stop()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:38).
  + GET /api/system/scheduler/status – returns state (running, tasks, last\_success\_at, heartbeat value).
* Authenticate using existing admin JWT middleware (reuse utilities from [backend/routers/settings.py](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/routers/settings.py) if present).
* Inject the scheduler instance via Depends on request.app.state.task\_scheduler to avoid circular imports.
* Protect routes behind the SCHEDULER\_CONTROL\_API flag and return 404 when disabled.

8. Data Model & Migrations

* No new migrations required. Existing headline tables and settings storage support the additional metadata.

9. Testing & Validation

* Unit tests:
  + Scheduler: fake ingestion service to validate retry/backoff logic for [TaskScheduler.\_run\_periodically()](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/scheduler.py:53).
  + Ingestion service: mock [FinvizClient.fetch\_portfolio\_headlines](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/finviz_client.py:63) to ensure dedupe and persistence flows.
  + Cache utilities: verify [invalidate\_downstream\_caches](vscode-webview://0pibngqu41ts2d5lel6qvpc8vsoeo1c8o7vpdg6tde3bt3gghjab/backend/services/ingestion.py) deletes expected keys via a fake Redis client.
* Integration tests:
  + Spin up FastAPI app with in-memory Redis to validate control API start/stop paths.
  + Use seeded headlines to confirm cache invalidation updates analytics endpoints under INGESTION\_V2\_ENABLED.
* Telemetry validation:
  + Assert Prometheus collectors emit expected samples after ingestion (scrape /metrics and parse).
  + Snapshot structlog JSON to ensure operational metadata is present.

10. External Dependency Risks & Mitigation

| **Risk** | **Impact** | **Mitigation** |
| --- | --- | --- |
| Finviz rate limiting/outage | High | Exponential backoff, dual-path fetch (export + scrape), manual toggle via INGESTION\_V2\_ENABLED, alert on consecutive failures. |
| Redis unavailable or degraded | Medium | Fallback to in-memory locks, skip cache invalidation with warnings, heartbeat stored in app memory, alert on cache errors. |
| Database contention during bulk ingest | Medium | Batch inserts, limit concurrency per portfolio, use short-lived locks. |
| Scheduler runaway tasks | Medium | Heartbeat watchdog, status endpoint, ability to pause via control API. |

11. Implementation Sequence & Acceptance Criteria

1. Implement ingestion service module with unit tests.
2. Wire scheduler to new service and add control API (feature-flagged).
3. Add telemetry/metrics and verify via integration tests.
4. Document operator runbooks (scheduler start/stop, cache operations).
5. Enable INGESTION\_V2\_ENABLED in staging; verify ingestion SLOs and cache invalidation.
6. Promote to production after 24h of healthy metrics.

Acceptance Criteria:

* Scheduler can be paused/resumed through the control API with feature flag on.
* Ingestion runs every five minutes, retries gracefully, and deduplicates headlines before persistence.
* Cache keys invalidated immediately after ingestion; analytics endpoints reflect fresh data.
* Metrics and structured logs expose ingestion health for dashboards and alerts.

The file accurately reflects the specified blueprint.