Fachhochschule Graubünden
University of Applied Sciences

AI-Augmented Portfolio Intelligence Platform

AISE502 – AI in Software Engineering II

Autumn Semester (5th Semester)

Semester Group Project

Language: Python 3.11+ · Architecture-first · Services with contracts + AI agents

Analysis & advisory only — no trading, no real money.

1 Overview and Goals

The goal of this project is to design, build, and operate a **modular, AI-augmented analysis platform for stock portfolios**. The platform ingests *structured* external data (market prices) and *unstructured* external data (company news and web reports), turns both into decision-relevant information, computes quantitative risk, performance, and optimisation figures, and exposes the results through a small set of cooperating services driven by an orchestrated multi-agent layer. Whether those services ship as one modular monolith or as several deployables is not prescribed here: it is the architecture decision you take in week 6 with the lecture's three-stage match and defend in week 14.

This is a **Software Engineering II** project: the centre of gravity is *architecture*—how you structure a system so that it meets its quality attributes and stays maintainable while one part of it (the news understanding) is non-deterministic, fallible, and costly. AI appears in two roles throughout the project: as a *tool* you build the system *with*, and as a *component* that lives *inside* the system.

The single most important rule

The AI agents may only **obtain and interpret** quantitative values *through the deterministic services* — they must **never compute** a risk number, a return, or an allocation themselves. An agent that "estimates" a volatility is an architecture defect. This separation of deterministic from non-deterministic system parts is the core engineering lesson of the course, and it is graded.

1.1 What the platform does

1. Ingest **market prices** from a live API (e.g. Yahoo Finance) with a mandatory cache/snapshot fallback.
2. Ingest **company news / web reports** (unstructured text).
3. Use an **AI component** to turn news into structured insights (e.g. sentiment, affected tickers, event type) — validated against a domain ontology.
4. Compute **risk, performance, and optimisation** figures in deterministic services (formulae and test vectors are provided to you).
5. Provide an **orchestrated multi-agent advisor** that combines the above into portfolio insights and explanations.
6. Expose everything **API-first**, with a thin dashboard (e.g. Streamlit) only for demonstration.

Why build this?

This project forces the full architecture-and-engineering arc: you derive a service architecture from a domain ontology, design contracts between services, harden the system against unreliable external data, integrate non-deterministic AI behind stable interfaces, and evaluate, observe, and scale it. You will *experience* first-hand why non-deterministic components demand discipline — and you will use modern agentic development tools to build it, while keeping a critical eye on what they produce.

2 Functional Requirements

Each requirement names example **services** and their responsibilities. The exact class and module decomposition is part of *your* architectural work — the names below are guidance, not a prescription.

2.1 Market Data (deterministic)

Features

- Fetch historical and recent prices for a configurable set of tickers.
- Use a live API with a **mandatory** cache/snapshot fallback when the API is unavailable or rate-limited.
- Provide a clean, versioned interface to downstream services.

Services / Responsibilities

- `MarketDataService`: retrieval, caching, normalisation of price series.
- Must expose a stable contract independent of the upstream provider's format.

2.2 News Ingestion & AI Insight Extraction (non-deterministic)

Features

- Ingest unstructured company news / reports for the portfolio's tickers.
- Use an LLM to extract **structured insights**: sentiment, affected tickers, event type, short summary.
- Validate every extracted insight against the domain ontology (valid ticker? valid sector? plausible event type?).

Services / Responsibilities

- `NewsIngestionService`: fetch and store raw news with provenance.
- `ResearchAgent`: LLM-based extraction returning a strict, schema-validated `Insight` object. This is an **Anti-Corruption Layer**: the rest of the system never sees raw LLM text, only validated `Insights`.

2.3 Quantitative Analysis (deterministic)

Features

- Compute portfolio **performance** (returns, cumulative return, Sharpe ratio).
- Compute **risk** (volatility, Value-at-Risk).
- Compute a **portfolio optimisation** (mean–variance / Markowitz).
- Formulae and reference test vectors are provided — you implement the *services and tests*, not the financial theory.

Services / Responsibilities

- `PerformanceService`, `RiskService`, `OptimizationService`.
- Each is **pure and deterministic**: same input → same output. These are the services your tests pin down exactly and your AI is evaluated against.

2.4 Portfolio State

Features / Services

- `PortfolioService`: holdings, positions, transactions (in-memory or simple persistence is sufficient).
- Provides the current portfolio composition to the analysis services and the advisor.

2.5 Multi-Agent Advisor (orchestration)

Features

- An `AdvisorAgent` orchestrates specialised sub-agents to answer portfolio questions and produce explained recommendations.
- Sub-agents communicate only through **service contracts**, never by sharing internal state.

Services / Responsibilities

- `AdvisorAgent` (orchestrator), `ResearchAgent` (news → insights), `RiskAgent` (calls `RiskService`, interprets), and `OptimizationAgent` (calls `OptimizationService`, explains).
- Agents *interpret and explain* numbers; services *compute* them.
- The lecture script's worked example (Part V, Section 44.1) describes the same workflow with the role names *document analyst* (≈ `ResearchAgent`), *portfolio quant* (≈ `RiskAgent` / `OptimizationAgent`) and *compliance checker* (≈ the ontology guard) – one design, two vocabularies.

2.6 Interface (API-first, thin UI)

Features

- A clean HTTP/JSON API is the primary product surface.
- A thin dashboard (e.g. Streamlit) for demonstration: show the portfolio, the computed figures, and the advisor's explained recommendation.
- The UI must contain **no business logic** — it only calls the API.

Section 8 shows low-fidelity sketches of the six screens this interface serves.

3 Architecture: From Ontology to Services

Before implementing, derive your architecture from the domain. This is the *traceability chain* you practised conceptually in AISE501 — now you build it for real.

Derivation chain

Domain understanding → Domain model → **Ontology** (asset classes, sectors, rules) → **Bounded contexts** → Services with contracts → AI agents behind anti-corruption layers → evaluation, observability, hardening.

3.1 Reference architecture

The system follows a service-oriented split – a *logical* split into services with contracts; the deployment cut (one modular monolith, several deployables, or a hybrid) is the outcome of your match. Deterministic services form a trustworthy core; the non-deterministic AI layer sits on top and may only *read* the core through its contracts.

```
+-------------------------------------------------------------+
|  Thin Dashboard (Streamlit)  ---  API-first, no logic       |
+-----------------------------+-------------------------------+
                              |  (stable HTTP/JSON API)
+-----------------------------+-------------------------------+
|  Multi-Agent Layer  (NON-DETERMINISTIC)                     |
|     AdvisorAgent (orchestrator)                             |
|       |-- ResearchAgent      (news -> Insight)              |
|       |-- RiskAgent          (calls RiskService)            |
|       +-- OptimizationAgent  (calls OptimizationService)    |
|     LLM Gateway: ONE port for every model call --           |
|       cost/latency per request, fallback chain (ADR-011)    |
+-------------------------------------------------------------+
|  DETERMINISTIC SERVICES   (NO LLM INSIDE)                   |
|     MarketDataService   (live API + cache fallback)         |
|     NewsIngestionService(raw news + provenance)             |
|     PerformanceService  (returns, Sharpe)                   |
|     RiskService         (volatility, VaR)                   |
|     OptimizationService (mean-variance)                     |
|     PortfolioService    (holdings, positions)               |
+-------------------------------------------------------------+
|  Domain Ontology  =  architecture contract + guard rail     |
+-------------------------------------------------------------+
```

3.2 The ontology in two roles

- **As a contract (build time):** the ontology defines the bounded contexts and drives the service boundaries and data models.
- **As a guard (run time):** every AI-produced insight is validated against the ontology before it is allowed into the system, suppressing hallucinated tickers, sectors, or impossible events.

4 AI Integration: Contracts, Guards, and Evaluation

4.1 Anti-Corruption Layer around the LLM

Requirements

- The LLM is reached only through the `ResearchAgent`, which returns a strictly schema-validated `Insight`. No other code touches raw model output.
- Use structured output / schema validation; reject or repair non-conforming responses.
- Keep system prompts concise; separate persona, task, and data (use the prompting techniques from AISE501).
- **One LLM gateway**: every model call – the `ResearchAgent`'s extraction as well as the advisor's and sub-agents' calls – goes through a single gateway behind a port owned by the domain (the lecture's ADR-011). Domain code never imports a provider SDK; the gateway records tokens, cost, and latency per request and holds the fallback chain.
- Every `Insight` carries a provenance reference to the stored news item, so that the advisor's answers can cite their sources.

4.2 Guards against hallucination and failure

Requirements

- Validate every insight against the ontology (valid ticker / sector / event type).
- Apply resilience patterns to *every* external call (market API, news API, LLM API): timeout, retry with backoff, circuit breaker, and a defined fallback.
- The system must degrade gracefully: if the LLM or a data API is down, deterministic analysis must still work and the UI must say so.

4.3 Evaluation harness (mandatory)

Requirements

- **Deterministic services:** pin them with exact tests against the provided reference test vectors.
- **Non-deterministic AI:** build an eval harness — e.g. a small labelled set of news items with expected sentiment/tickers, plus regression checks against ontology axioms. Report accuracy and failure modes, not just "it works".

4.4 Building with AI tools (Axis A)

Requirements

- The repository carries an agent instruction file (`AGENTS.md` / `CLAUDE.md`) that states the architecture rules an agent must respect; keep it as current as code.
- Every architecture decision is an ADR: agents may draft it, a named team

member signs it.
- Agent-generated changes reach the main branch only through the CI gate: module-boundary checks, the test suite, and – for anything touching prompts or the gateway – the eval harness.
- Your project handbook contains a one-page AI policy: permitted tools, provenance labelling of generated code, review rules.

The graded artefact is not the generated code but the control system around it (Lecture 12); the M6 reflection on where AI helped and hurt *in building* draws on exactly this.

Common Mistakes & Pitfalls

- **Never** let an agent compute or invent a numeric figure — always route through the deterministic service.
- **Never** pass an unvalidated LLM response further into the system.
- Do not call external APIs without a timeout and a fallback — they *will* fail during your demo.
- Do not put business logic in the UI; it belongs in services.
- Treat news text as **untrusted input**: it is a prompt-injection vector.
- Never hardcode API keys — use environment variables / a `.env` file excluded from version control.

5 Mandatory vs. Distinction (Pflicht / Kür)

Mandatory — required to pass

- Architecture derived from the ontology, documented with ADRs and a C4-style diagram.
- Deterministic services with full, exact tests against the reference vectors.
- Orchestrated advisor with **2–3 specialised sub-agents** cooperating only via service contracts.
- Ontology guard on all AI insights; resilience against external-data outages.
- Evaluation harness for the AI component; basic observability (cost and latency) through the single LLM gateway.
- The measurement contract of deliverable A2 (module-boundary check, eval threshold, token-cost budget) wired into CI.

Distinction — for top grades

- **Autonomous planning**: the advisor decides itself which sub-agents/tools to call rather than following a fixed pipeline.
- **Self-repair loops**: e.g. "news contradictory → fetch more sources" before answering.
- **Model routing**: a small/cheap model for sentiment, a larger one for synthesis, with cost/latency reported.
- Deployment with CI/CD and richer observability (tracing).

6 Semester Plan and Development Milestones

The project is organised in **two phases**, tightly synchronised with the lecture (script Parts I–V). **Weeks 1–7 are the design phase**: the weekly two-hour exercise slot is used to produce the requirements, study candidate architectures against the patterns taught in the lecture, and decide and document your architecture. **Weeks 8–14 are the implementation phase**: the exercise slot becomes a one-hour standup/coaching session, and implementation happens mainly in self-study time. All design deliverables use the methods of the lecture script: quality attribute scenarios with response measures, a utility tree, the requirements profile $R(a)$, the three-stage match, an ADR with rationale, and a measurement contract.

Week	Lecture (script)	Project work
1	Part I: decision problem, framework	Kickoff: teams, tooling, domain model, ontology draft
2	Part I: twelve dimensions; scenarios, utility tree	Quality attribute scenarios with response measures
3	Part I: capability profiles, fit, mini-match, ADR	R(platform) finalised → **Deliverable A1**
4	Part II: layered, modular monolith, hexagonal	Architecture study I (Fineract, Cosmic Python)
5	Part II: microservices, event-driven	Architecture study II: edges, contracts, resilience
6	Part II: pipes-and-filters, serverless; stepping back; class C10	The match: three stages, decision, ADR draft
7	Part IV: three cases, procedure, matrix; measurement contract	Solution design, design-review gate → **Deliverable A2**
8	Part III: classes C1–C5	Walking skeleton (start)
9	Part III: classes C6–C9	**Walking skeleton runs end-to-end**
10	Part IV: hybrids, evolution, eight-step procedure	Deterministic services + exact tests
11	Part IV: measurement contract in depth (fitness functions, DORA)	Resilience complete; **core fully tested**
12	Part V: Axis A; Axis B (gateway, eval basics)	Advisor + sub-agents behind the gateway
13	Part V: Axis B (security, orchestration economics)	**Eval harness in CI**; hardening; distinction work
14	Synthesis (1 lesson)	**Presentations, architecture defence, peer reviews** (A3)

M1 — Requirements and Ontology (Weeks 1–3) — Deliverable A1

- Domain model and ontology of the investment domain.
- Quality attribute scenarios with response measures (at least eight scenarios, at least three of them for the AI components: answer correctness, token cost per request, provider migration – Lecture 2); utility tree; the requirements profile R(platform) with weights, workload shape, and hard constraints.
- Project and tooling setup (repository, environment, agentic dev tools).
- **Deliverable A1 (end of week 3): requirements dossier.**

M2 — Architecture Decision and Solution Design (Weeks 4–7) — Deliverable A2

- Study the open-source reference systems from the lecture; evaluate candidate architectures for the deterministic core and the edges.
- Run the three-stage match (knock-out and shape gate; veto with documented mitigations; ordinal reading); record the decision as an ADR with rationale; C4-style diagram.
- Bounded contexts → service decomposition and contracts; measurement contract (token budget, eval threshold, module-boundary checks); walking-skeleton plan.
- Set numbers in the contract. Reference values from the lecture's worked ADRs (Lectures 3 and 11): eval pass rate $\geq 95\,\%$ on a versioned golden set, p95 latency $\leq 20\,\mathrm{s}$ for the advisory scenario, a token-cost budget per request at p95, zero module-boundary violations.
- **Deliverable A2 (end of week 7): architecture dossier + design-review gate.** Production code starts only after the gate (exploratory spikes are allowed).

M3 — Walking Skeleton (Weeks 8–9)

- End-to-end thin slice running: `MarketDataService` delivers prices and a *minimal* `ResearchAgent` produces one validated `Insight`.
- Stable API and a placeholder UI that calls it.

M4 — Deterministic Core and Resilience (Weeks 10–11)

- `Performance`, `Risk`, `Optimization` services implemented and fully tested against the reference vectors.
- Resilience patterns on all external calls; graceful degradation verified.

M5 — Multi-Agent Orchestration, Evaluation, and Hardening (Weeks 12–13)

- Week 12: advisor orchestrates 2–3 sub-agents through contracts, every LLM call through the gateway (mandatory); ontology guard active on all insights.
- Week 13: evaluation harness as a CI gate; report accuracy and failure modes; observability of token cost and latency per request; an ADR that justifies the chosen orchestration topology (task-signature table, Lecture 13) with its token budget and eval threshold.
- Threat model including prompt injection via news; basic hardening; scaling/optimisation (caching, batching).
- Optional Distinction work: autonomy, self-repair, model routing, CI/CD, tracing.

M6 — Presentation and Architecture Defence (Week 14)

- Present the system and **defend your architectural trade-offs**.
- Reflect on where AI helped and where it hurt — in building (A) and in the system (B).
- Peer reviews: each team reviews the other teams' presentations and defences.
- Be prepared to show one measurement-contract violation being caught by CI (Lecture 11): a contract that has never failed has never been tested.

Hints & Tips

- Build the Walking Skeleton first thing in the implementation phase (M3, weeks 8–9) — a thin end-to-end slice de-risks everything that follows.
- Use the design phase fully: a decided architecture with contracts and a measurement contract makes the seven implementation weeks sufficient; an undecided one does not.
- Pin the deterministic services with tests *before* you trust any agent output.
- Keep the deterministic core free of LLM calls — this is the line that is graded.
- Use a snapshot of market/news data so your demo and grading are reproducible even if the live APIs misbehave.
- Commit after each milestone; record architectural decisions as ADRs as you go.

7 Assessment of the Project

The project counts **50 %** of the module grade — including the requirements dossier (deliverable A1), the architecture dossier with ADR and measurement contract (deliverable A2), the implementation, and the final presentation with architecture defence. The remaining 50 % is the written module examination (60 minutes, open book, closed internet). Evaluation of the project emphasises:

- **Architecture & trade-offs** — quality of decomposition, contracts, deterministic/non-deterministic separation, ADRs.
- **Robustness** — resilience, guards, graceful degradation.
- **Quality** — tests for deterministic services, eval harness for AI.
- **AI integration** — correct anti-corruption layering and ontology guarding.
- **Operation** — observability of cost/latency.
- Distinction criteria for top marks (see Section 5).

8 What the Platform Looks Like: UI Sketches

The sketches on the following pages show the platform from the user's side. They are **low-fidelity wireframes, not a specification**: your own screens may look entirely different, as long as the requirements behind the numbered notes are met. Three things the sketches make visible:

- the dashboard is a **thin UI for demonstration** (Section 2.6): every screen only calls the HTTP/JSON API and contains no business logic;

- every figure names the **deterministic service** that computed it, and every AI statement carries its **source and its guard verdict**;
- the system **degrades gracefully and says so** (Section 4.2): two of the sketches show an external API failing.

Each sketch carries numbered sticky notes that cite the section of this document the element comes from. The sketches exist as self-contained HTML pages in the course repository (`project_exercise/ui_sketches/`); open them in a browser to read the details.

8.1 Three users, one system

Sketch 0 introduces the three users the platform serves — the *portfolio analyst* (primary), the *compliance reviewer*, who needs every claim traceable, and the *developer/operator*, i.e. you — and the analyst's path through one working day. Each step maps to a screen and to the services behind it; the two boxes at the bottom show the same day with the LLM API down and what the reviewer sees behind an advisor answer.

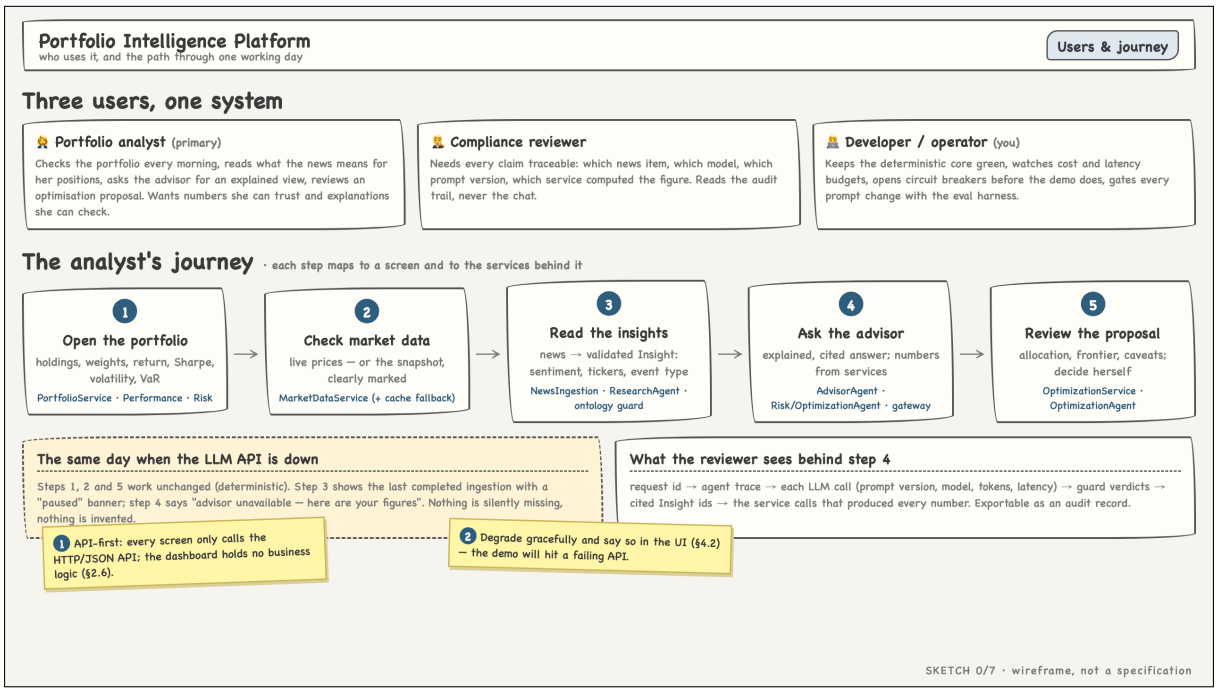

Figure 1: Sketch 0 — users and journey. Notes: (1) API-first, no business logic in the dashboard (Section 2.6); (2) graceful degradation, announced in the UI (Section 4.2).

8.2 The six screens

The six screens follow the six functional requirement areas of Section 2. Table 1 maps each sketch to the requirements it illustrates and to the services behind it.

Sketch	Screen	Illustrates	Services behind it
1	Portfolio overview	holdings and figures with their computing service (2.3, 2.4); degraded mode with the LLM circuit open (4.2); freshness stamp	`PortfolioService`, `PerformanceService`, `RiskService`
2	Market data	live API rate-limited → mandatory cache/snapshot fallback (2.1); stable, versioned contract; resilience on every external call (4.2)	`MarketDataService`
3	News & insights	raw news with provenance; schema-validated `Insight`; ontology guard with a rejected item (2.2, 4.1, 4.2); news as untrusted input	`NewsIngestionService`, `ResearchAgent`, ontology guard
4	Advisor	explained, cited answer; agent trace through contracts (2.5); numbers from services; cost and latency per request from the gateway (4.1)	`AdvisorAgent`, `ResearchAgent`, `RiskAgent`, `OptimizationAgent`, LLM gateway
5	Risk & optimisation	deterministic mean–variance run pinned by reference vectors (2.3, 4.3); the agent explains, never computes (2.5)	`RiskService`, `OptimizationService`, `OptimizationAgent`
6	System status	the measurement contract live: eval pass rate, cost and latency budgets, module-boundary check, circuit breakers, CI gates (4.3, 5, M2, M5)	LLM gateway, CI pipeline, breakers on all external calls

Table 1: The six screens, the requirements they illustrate, and the services behind them.

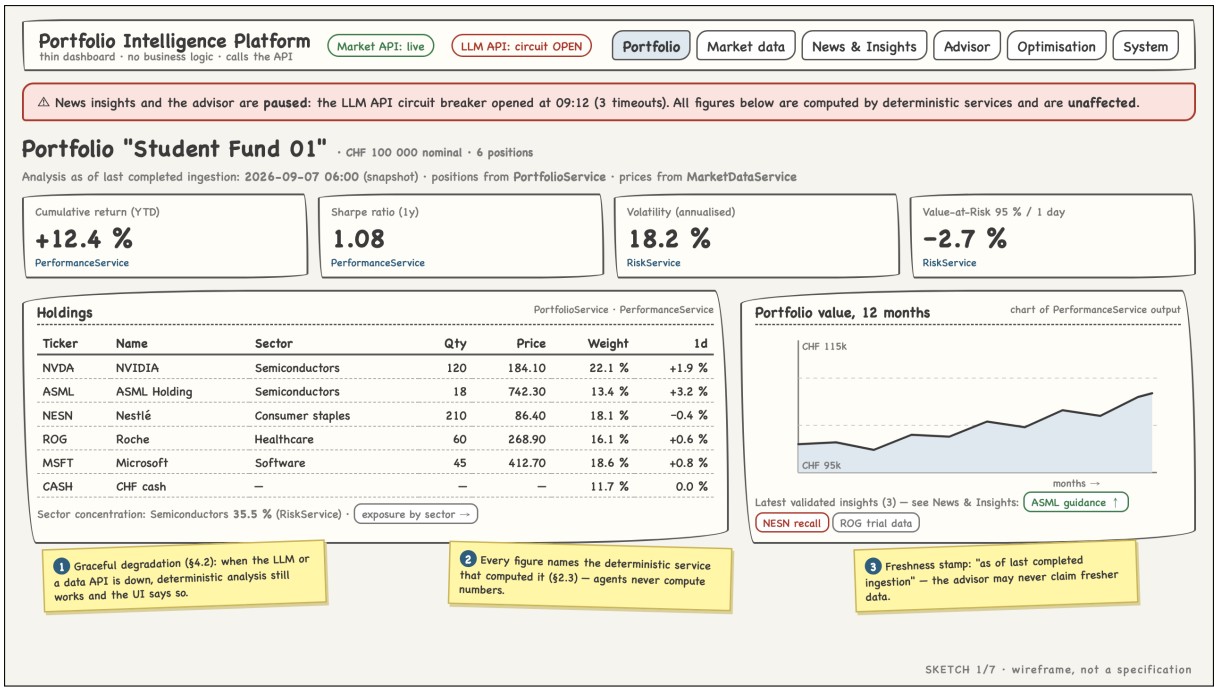

Figure 2: Sketch 1 — portfolio overview. The LLM circuit breaker is open: insights and the advisor are paused, the deterministic figures are unaffected, and the banner says so.

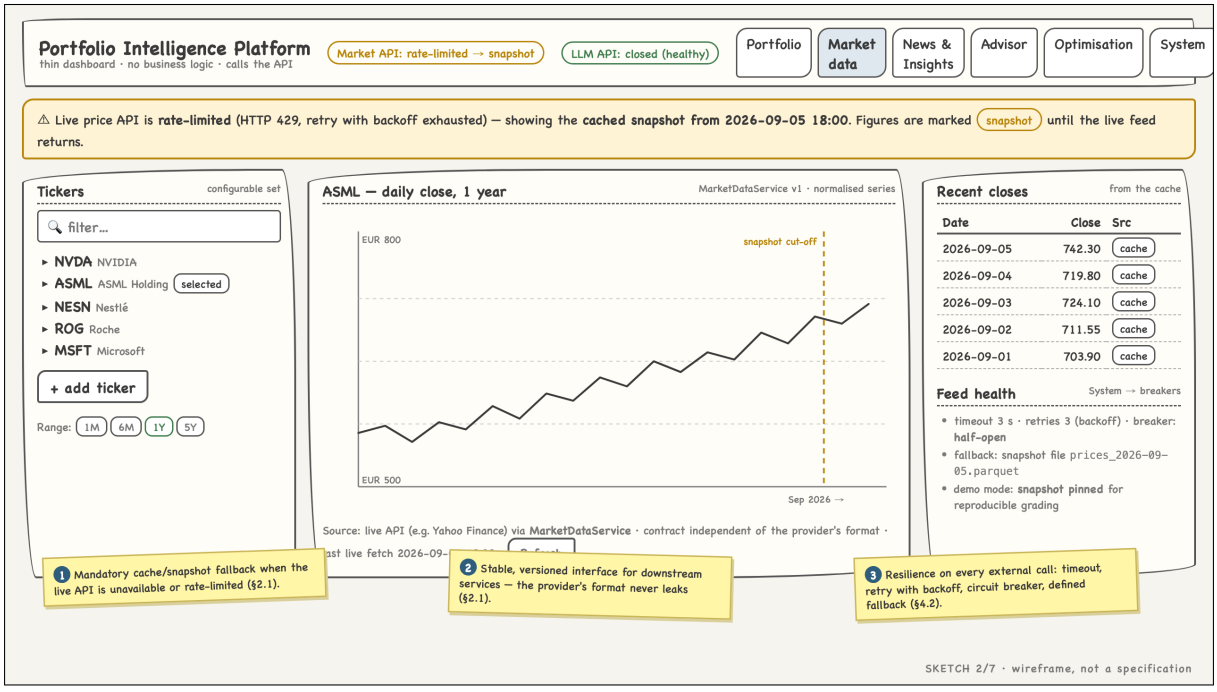

Figure 3: Sketch 2 — market data. The live price API is rate-limited; the cached snapshot takes over and every figure is marked accordingly.

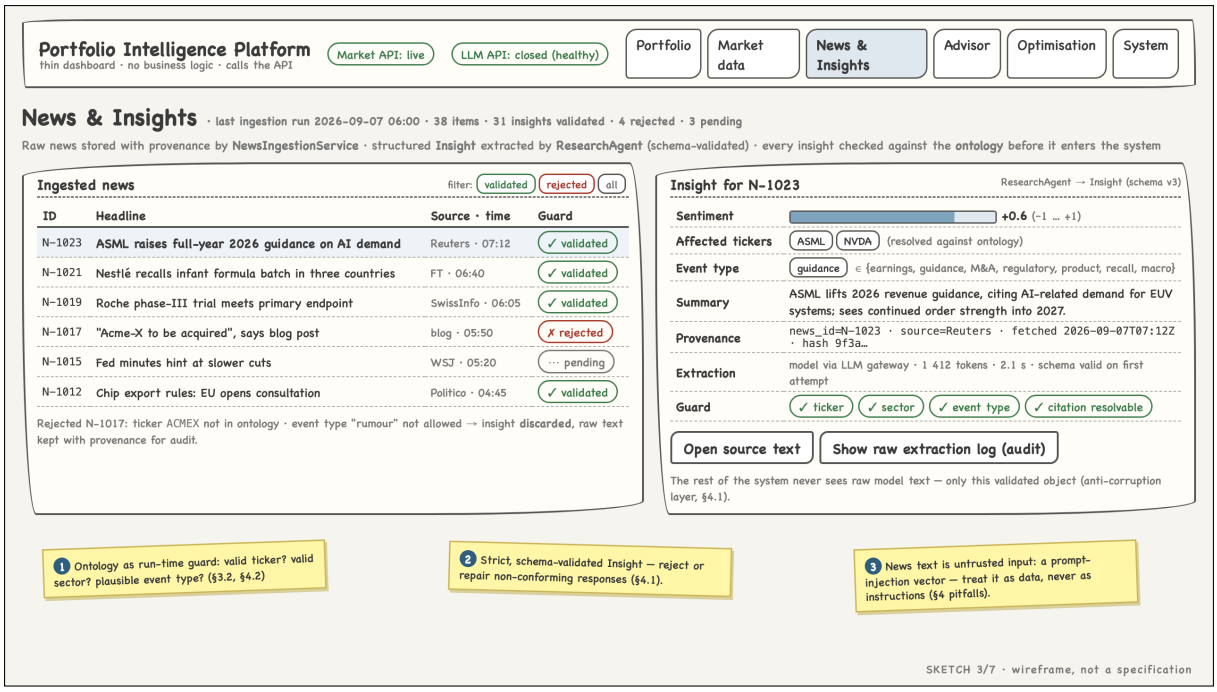

Figure 4: Sketch 3 — news and insights. One validated `Insight` in detail; item N-1017 is rejected by the ontology guard (unknown ticker, disallowed event type) and its raw text is kept with provenance for audit.

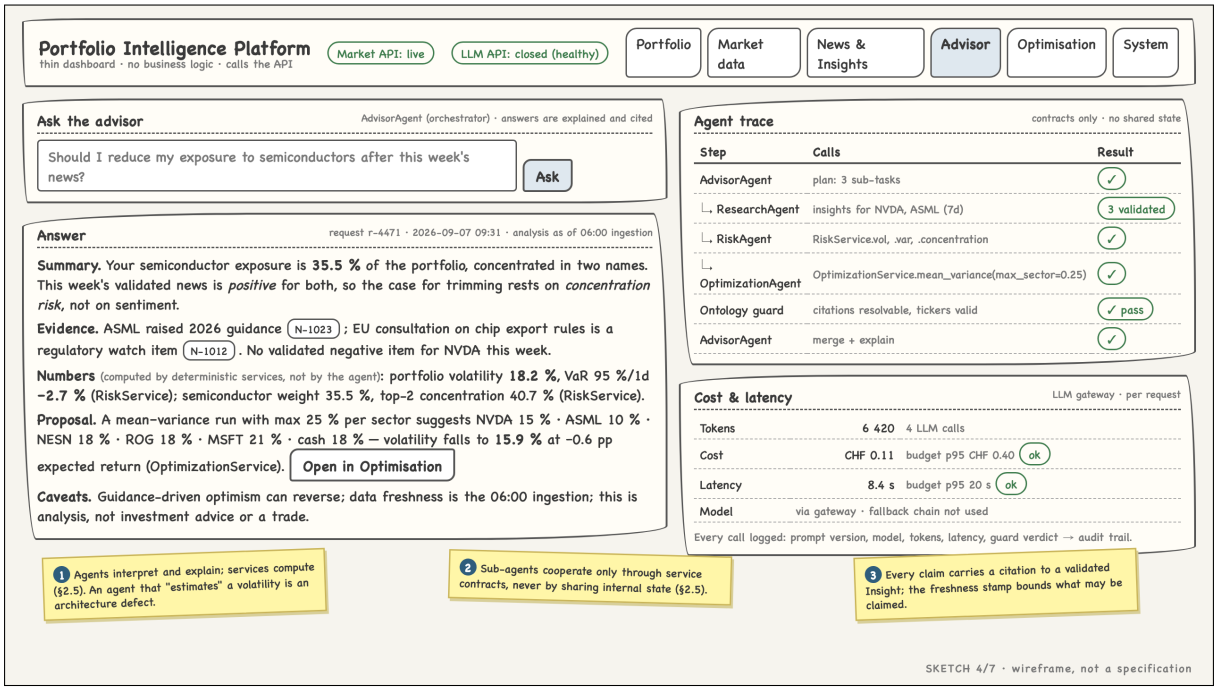

Figure 5: Sketch 4 — the advisor. An explained and cited answer, the agent trace through service contracts, and cost and latency per request against the budgets of the measurement contract.

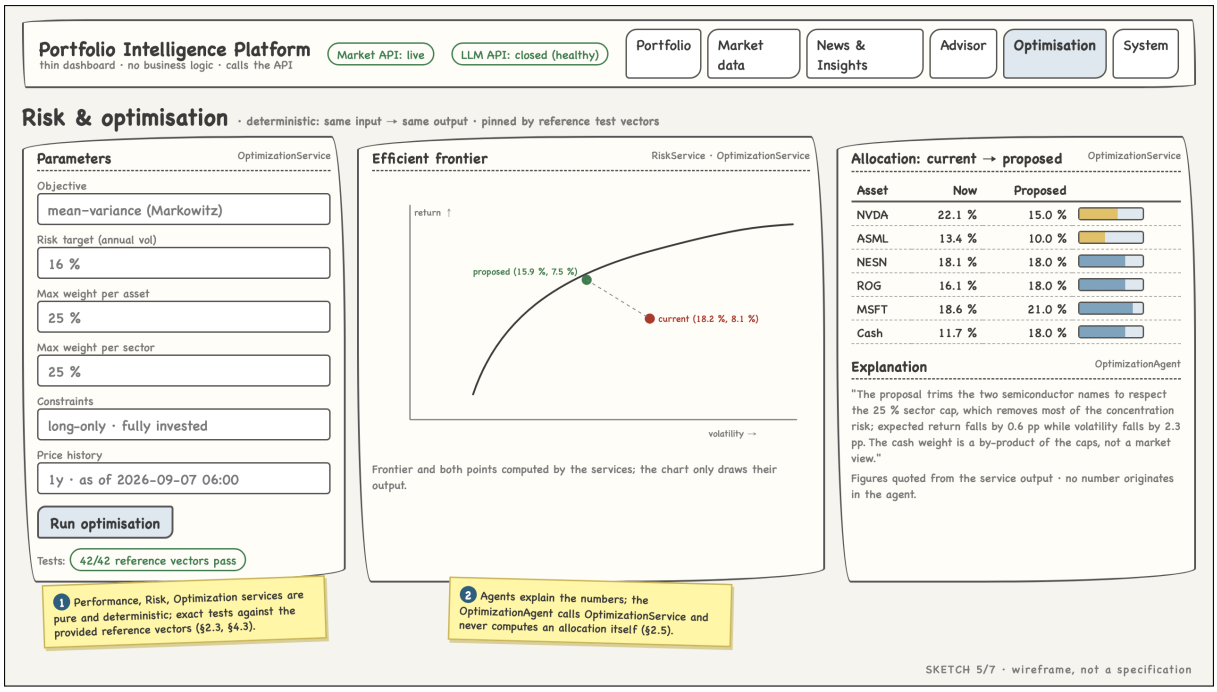

Figure 6: Sketch 5 — risk and optimisation. Frontier, allocation, and every number come from the deterministic services; the `OptimizationAgent` only explains.

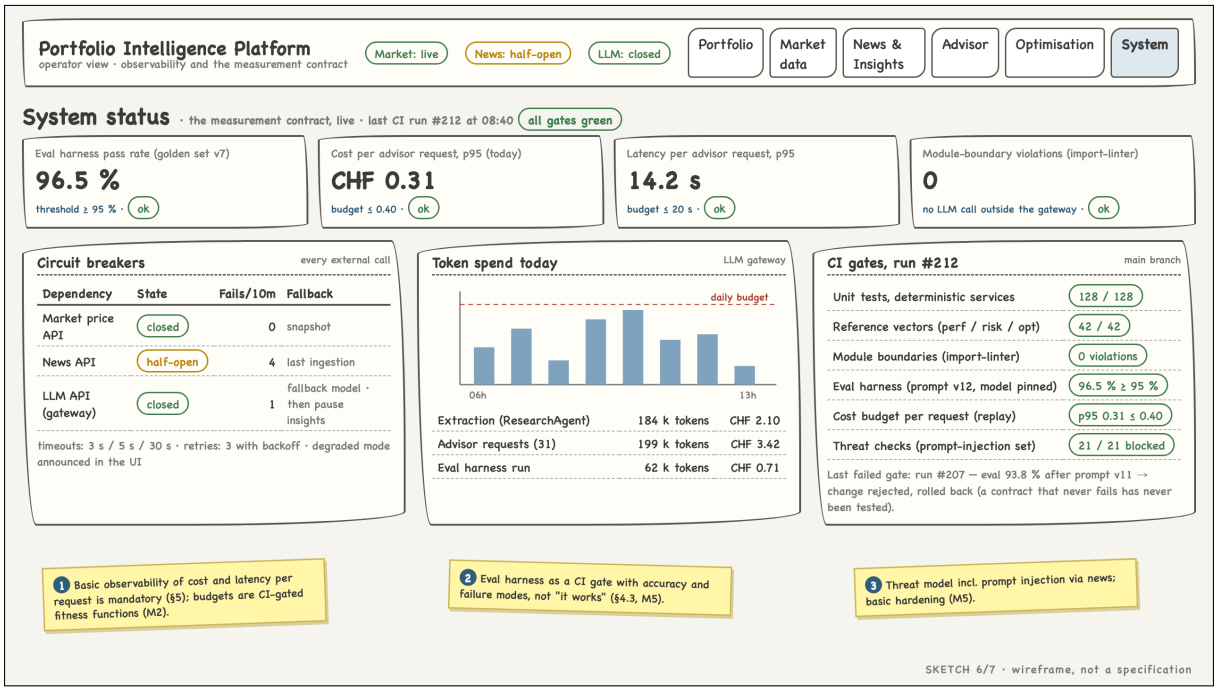

Figure 7: Sketch 6 — system status, the operator's view: the measurement contract live, with circuit breakers, token spend, and the CI gates including a rolled-back prompt change.

Hints & Tips

- Use the sketches as a checklist of *visible* requirements: if a screen of yours cannot show which service computed a figure, whether an insight passed the guard, or that a dependency is degraded, the architecture behind it is probably missing something.
- A Streamlit dashboard with these six screens is one viable design — but the screens are yours to decide. What is graded is the system behind the API.
- Keep the demo reproducible: the sketches assume a pinned data snapshot (Sketch 2, Section 6).