AgentForge

System Architecture

Production-Ready Financial AI Agent for Ghostfolio

February 2026 • v0.1.0

1. Overview

AgentForge is a production-ready financial AI agent that integrates with Ghostfolio, an open-source portfolio management platform. It provides natural-language portfolio analysis through a chat interface powered by Claude Sonnet 4.6 and a LangGraph reasoning loop.

The system consists of four layers:

- Frontend — Angular 21 chat widget deployed to Vercel
- API Gateway — FastAPI REST endpoints on Railway
- Agent Core — LangGraph state machine with verification pipeline
- Data Layer — Ghostfolio REST API backed by PostgreSQL and Redis

Component	Technology	Deployment
Chat Widget	Angular 21, ngx-markdown, SCSS	Vercel
Agent API	FastAPI, Uvicorn, Python 3.12	Railway (Docker)
LLM	Claude Sonnet 4.6 (primary), Haiku 4.5 (fallback)	Anthropic API
Reasoning	LangGraph state machine	Embedded in agent
Data Source	Ghostfolio v2.243.0	Railway (Docker)
Database	PostgreSQL 16	Railway (managed)
Cache	Redis 7	Railway (managed)
Observability	Langfuse	Langfuse Cloud

2. System Architecture

The request lifecycle flows through four tiers. Each component communicates over HTTP, with internal Railway services using private DNS (*.railway.internal).

```
┌──────────────────────────────────────────────────────────────┐
│                      FRONTEND  (Vercel)                      │
│                                                              │
│   Angular 21 SPA  →  AgentService (Signals + RxJS)           │
│   Components: HeaderBar, ChatContainer, ChatMessage,         │
│               ChatInput, MessageMetadata, TypingIndicator    │
└─────────────────────────┬────────────────────────────────────┘
                          │ HTTPS POST /api/agent/chat
```

```
┌──────────────────────────────▼───────────────────────────────┐
│                     API GATEWAY  (Railway)                   │
│                                                              │
│  FastAPI + CORS  →  ChatRequest / ChatResponse (Pydantic)    │
│  Endpoints: POST /api/agent/chat, GET /api/agent/health      │
└──────────────────────────────────────────────────────────────┘
                               │
                               │ run_agent()
┌──────────────────────────────▼───────────────────────────────┐
│                  AGENT CORE  (LangGraph)                     │
│                                                              │
│  classify → call_model → should_continue ─┬→ verify → END    │
│       ▲                                   │                  │
│       └─ collect ← tools (ToolNode) ◄─────┘                  │
│                                                              │
│  7 Tools: portfolio_analysis, portfolio_performance,         │
│           market_data_lookup, transaction_history,           │
│           tax_estimate, compliance_check, benchmark_comparison│
└──────────────────────────────────────────────────────────────┘
                               │
                               │ GhostfolioClient (httpx)
┌──────────────────────────────▼───────────────────────────────┐
│                    DATA LAYER  (Railway)                     │
│                                                              │
│  Ghostfolio REST API  ↔  PostgreSQL 16  +  Redis 7           │
└──────────────────────────────────────────────────────────────┘
```

3. Agent Reasoning Graph

The core of AgentForge is a LangGraph compiled state machine. Each node is a pure function that takes the current AgentState and returns an updated state. The graph executes asynchronously via ainvoke().

3.1 State Schema

Field	Type	Purpose
messages	list[BaseMessage]	Full conversation history (LangChain message protocol)
tool_results	list[dict]	Parsed results from tool executions
iterations	int	LLM invocation count (capped at MAX_ITERATIONS=10)
confidence	float	Verification confidence score (0.0–1.0)
verification_passed	bool	Whether all verification checks passed
query_type	str	Classified query type: general | tax | advice | compliance
total_input_tokens	int	Accumulated input token count across all LLM calls
total_output_tokens	int	Accumulated output token count across all LLM calls

3.2 Graph Nodes

classify_query — Classifies the user query using keyword matching into one of four types: general, tax, advice, or compliance. This determines verification behavior (e.g., tax queries require a disclaimer).

call_model — Prepends the system prompt and invokes Claude Sonnet 4.6 with all 7 tools bound. Enforces the iteration limit (default 10). Extracts token usage from response_metadata and accumulates it in the graph state.

should_continue (conditional edge) — Routes to 'tools' if the AI response contains tool_calls, otherwise routes to 'verify'. This creates the agent loop: the LLM can call tools multiple times before producing a final answer.

tools (ToolNode) — LangGraph's built-in ToolNode executes the tool calls from the AI message. Each tool is a @tool-decorated async function that calls the GhostfolioClient.

collect_tool_results — Parses JSON from tool response messages and accumulates structured results for the verification step.

verify_response — Runs four verification checks on the final response: fact-checking against tool data, hallucination detection, domain constraint compliance, and completeness. Computes the confidence score and optionally appends a disclaimer for tax/advice queries.

4. Tool Layer

AgentForge exposes 7 domain-specific tools to the LLM. Each tool is a LangChain @tool-decorated async function that wraps calls to the GhostfolioClient. Tools return a standardized ToolResult with status, data, message, and execution_time.

Tool	Description	Ghostfolio Endpoints
portfolio_analysis	Holdings, allocation, sector breakdown, total value	/v1/portfolio/details
portfolio_performance	TWR, max drawdown, net/gross returns by date range	/v2/portfolio/performance
market_data_lookup	Current price, asset class, sectors for a symbol	/v1/symbol/{source}/{symbol}
transaction_history	Activity log with type counts and fees	/v1/order
tax_estimate	Capital gains/losses, dividend income estimate	/v2/portfolio/performance + /v1/portfolio/dividends
compliance_check	Concentration/diversification rule violations	/v1/portfolio/details
benchmark_comparison	Portfolio vs benchmark return comparison	/v2/portfolio/performance + /v1/benchmarks

All tools share a common error-handling pattern: catch HTTP and connection errors, return a ToolResult with status='error', and let the LLM compose an appropriate user-facing message.

5. Verification Pipeline

The ResponseVerifier runs four independent checks after the LLM produces its final answer. Each check contributes equally to the confidence score.

Check	What It Detects	Result
Fact Verification	Numbers in response not found in tool data	Warning if < 50% verified
Hallucination Detection	Buy/sell recommendations, guaranteed claims, risk-free assertions	Error if detected
Domain Constraints	Missing disclaimers on tax/advice responses	Error (tax) or Warning (advice)
Completeness	Response too short, tool errors not acknowledged	Warning if incomplete

Confidence Formula: confidence = (checks_passed / 4) × 0.7 + tool_success_rate × 0.3

A response passes verification if confidence ≥ 0.5 and there are no errors.

6. Chat Widget Frontend

The frontend is a standalone Angular 21 application using signal-based reactivity (zoneless). It communicates with the agent API via HttpClient and renders responses as markdown.

6.1 Component Hierarchy

```
AppComponent (theme state, shell)
  ├── HeaderBarComponent (title, connection status, theme toggle, clear)
  └── ChatContainerComponent (message list, empty state, error banner)
        ├── ChatMessageComponent (bubble, markdown, metadata)
        │     ├── MarkdownComponent (ngx-markdown rendering)
        │     ├── MessageMetadataComponent (expandable tools/confidence/cost)
        │     └── TypingIndicatorComponent (3-dot bounce animation)
        └── ChatInputComponent (auto-resize textarea, send button)
```

6.2 Data Flow

- User types in ChatInputComponent → emits messageSent event
- ChatContainerComponent calls AgentService.sendMessage()
- AgentService adds user message + loading placeholder to messages signal
- HTTP POST to /api/agent/chat → on success, replaces placeholder with response
- Signal reactivity triggers re-render of ChatMessageComponent
- ChatMessageComponent renders markdown via <markdown [data]="...">
- MessageMetadataComponent displays expandable tools/confidence/duration/cost panel

6.3 Theming

The widget supports light and dark themes via CSS custom properties defined in _variables.scss. The theme-dark class is toggled on the root element. The initial theme is set from the user's prefers-color-scheme media query.

7. Observability

Every agent invocation is traced to Langfuse with:

- Trace: conversation_id, input/output messages, tools used, confidence
- Generation: model name, token counts, cost
- Score: confidence value attached to trace

Langfuse integration is lazy-initialized and fails silently if credentials are not configured, ensuring the agent works without observability in development.

8. Deployment Topology

Service	Platform	URL / Domain	Image
Chat Widget	Vercel	widget-blond-tau.vercel.app	Angular static build
Agent API	Railway	agentforge-production-3f34.up.railway.app	python:3.12-slim + Uvicorn
Ghostfolio	Railway	ghostfolio.railway.internal:3333	ghostfolio/ghostfolio:latest
PostgreSQL	Railway	postgres.railway.internal:5432	postgres:16 (managed)
Redis	Railway	redis.railway.internal:6379	redis:7 (managed)

Internal Railway services communicate over private DNS (*.railway.internal). Only the Agent API and Chat Widget have public-facing domains. The Dockerfile uses a shell-form CMD to respect Railway's injected PORT environment variable.

9. Evaluation Framework

AgentForge includes 50 test cases across four categories:

Category	Count	Purpose
Happy Path	20	Standard queries — portfolio analysis, performance, market data
Edge Cases	10	Invalid inputs, non-existent symbols, extreme parameters
Adversarial	10	Prompt injection, unauthorized access, financial advice requests
Multi-Step	10	Queries requiring multiple tool calls in sequence

Each test case specifies expected tools, expected outcome keywords, and custom pass criteria (must_contain_disclaimer, must_refuse, max_latency). The evaluation runner exits with code 1 if the pass rate falls below 80%.

Performance Targets

- End-to-end latency (single tool): < 5 seconds
- Multi-step latency (3+ tools): < 15 seconds
- Tool success rate: > 95%
- Eval pass rate: > 80%
- Hallucination rate: < 5%

10. Configuration

The agent is configured via environment variables:

Variable	Required	Default	Purpose
ANTHROPIC_API_KEY	Yes	—	Claude API authentication
GHOSTFOLIO_API_URL	No	http://localhost:3333/api	Ghostfolio base URL
GHOSTFOLIO_API_TOKEN	No	—	JWT token for Ghostfolio auth
LANGFUSE_PUBLIC_KEY	No	—	Langfuse tracing (public key)
LANGFUSE_SECRET_KEY	No	—	Langfuse tracing (secret key)
LANGFUSE_HOST	No	https://cloud.langfuse.com	Langfuse endpoint
AGENT_MAX_ITERATIONS	No	10	Max LLM reasoning loops
AGENT_TIMEOUT_SECONDS	No	30	Request timeout
AGENT_MAX_COST_USD	No	0.10	Per-request cost limit

11. Cost Model

Token usage and cost are tracked per request across all LLM invocations in the agent loop. Pricing is based on Claude Sonnet 4.6 rates:

- Input tokens: $3.00 / million tokens
- Output tokens: $15.00 / million tokens
- Typical single-tool query: ~5,000–7,000 tokens, $0.02–$0.04
- Multi-step query: ~10,000–15,000 tokens, $0.05–$0.10

The AGENT_MAX_COST_USD configuration (default $0.10) provides a safety cap per request.