Pre-Search and Discovery Document for AgentForge

Project: AgentForge — Production-Ready Financial AI Agent for Ghostfolio
Domain: Finance (Wealth Management / Portfolio Analytics)
Repository: [ghostfolio/ghostfolio](https://github.com/ghostfolio/ghostfolio) (v2.242.0, AGPL-3.0)
Date: February 23, 2026
Author: AgentForge Team

Phase 1: Define Your Constraints

1. Domain Selection

Domain: Finance — Personal Wealth Management and Portfolio Analytics

Why Ghostfolio:

Ghostfolio is an open-source, privacy-first wealth management application built with Angular 21 (frontend) and NestJS 11 (backend) in a monorepo managed by Nx. It tracks portfolios across multiple asset classes (equities, bonds, cryptocurrencies, ETFs, mutual funds, commodities, real estate, and private equity) with real-time market data from multiple providers (Yahoo Finance, Alpha Vantage, CoinGecko, Financial Modeling Prep, and others). Ghostfolio already has a nascent AI endpoint (`/api/ai/prompt/:mode`) that generates portfolio analysis prompts via OpenRouter, but it lacks agentic capabilities — tool use, multi-step reasoning, verification, and evaluation. This gap is the perfect opportunity for AgentForge.

Specific Use Cases the Agent Will Support:

1. **Portfolio Analysis** — Analyze holdings, allocation percentages, sector/geography concentration, and risk exposure with natural language queries
2. **Transaction Categorization** — Classify and pattern-detect across BUY, SELL, DIVIDEND, FEE, INTEREST, and LIABILITY activities
3. **Market Data Retrieval** — Fetch and interpret current market data for symbols across multiple data sources (YAHOO, ALPHA_VANTAGE, COINGECKO, etc.)
4. **Tax Estimation** — Estimate capital gains/losses and dividend tax liability from activity history
5. **Compliance & Risk Checks** — Verify portfolio concentration limits, asset class diversification rules, and flag regulatory concerns
6. **Performance Benchmarking** — Compare portfolio performance against benchmarks over configurable date ranges

Verification Requirements for Finance Domain:

- All numerical claims (returns, allocations, balances) must be cross-referenced against Ghostfolio's own database — never hallucinated
- Investment advice must carry explicit disclaimers (not a licensed financial advisor)

- Tax estimates must note jurisdiction-dependence and recommend professional consultation
- Concentration warnings must use established thresholds (e.g., single holding > 20%, single sector > 40%)
- Currency conversions must use Ghostfolio's own exchange rate service for consistency

Data Sources:

- Ghostfolio's PostgreSQL database via Prisma ORM (primary source of truth)
- Ghostfolio REST API endpoints: `/portfolio/details`, `/portfolio/holdings`, `/portfolio/performance`, `/portfolio/dividends`, `/portfolio/investments`, `/portfolio/report`, `/order`, `/account`, `/exchange-rate`, `/symbol`, `/market-data`, `/benchmarks`
- External market data providers (via Ghostfolio's existing data service layer): Yahoo Finance, Alpha Vantage, CoinGecko, Financial Modeling Prep

2. Scale & Performance

Parameter	Target
Expected query volume (MVP)	50-100 queries/day
Expected query volume (production)	1,000-10,000 queries/day
Acceptable latency — single-tool	< 5 seconds
Acceptable latency — multi-step (3+ tools)	< 15 seconds
Concurrent users	10-50 simultaneous sessions
Cost constraints	< $0.05 per query average; < $500/month at 10K users

Assumptions for Cost Projections:

- Average query: ~2,000 input tokens + ~800 output tokens
- 70% of queries require 1 tool call, 25% require 2-3 tool calls, 5% require 4+ tool calls
- Verification overhead adds ~15% token cost
- Using Claude Sonnet 4.6 pricing: ~$3/M input, ~$15/M output tokens

Scale	Queries/Day	Monthly Est.
100 users	~300	~$45/mo
1,000 users	~3,000	~$450/mo
10,000 users	~30,000	~$4,500/mo
100,000 users	~300,000	~$45,000/mo

3. Reliability Requirements

Cost of a Wrong Answer:

- **High for numerical claims:** Incorrect portfolio values, wrong return calculations, or miscategorized transactions could lead users to make poor financial decisions. Financial losses are real and trust-destroying.

- **Medium for qualitative analysis:** Subjective observations about diversification or risk are less dangerous but still must be grounded in data.
- **Critical for tax estimates:** Incorrect tax guidance could lead to legal/compliance issues.

Non-Negotiable Verification:

1. **Numerical Accuracy** — Every number (balance, return, allocation %) must come from Ghostfolio's computed data, not LLM generation
2. **Source Attribution** — Every claim must reference the specific data source (e.g., "Based on your portfolio holdings as of [date]")
3. **Disclaimer Enforcement** — All tax, compliance, and investment advice responses must include appropriate disclaimers

Human-in-the-Loop Requirements:

- Tax estimates flag for professional review
- Portfolio optimization suggestions marked as "ideas" not "recommendations"
- Any response with confidence score < 0.7 should suggest the user verify with a financial professional

Audit/Compliance Needs:

- Full trace logging of every agent interaction (input, reasoning, tool calls, output)
- Token usage and cost tracking per request
- Immutable audit trail for regulatory review
- No storage of responses as financial advice (explicit disclaimers in every interaction)

4. Team & Skill Constraints

Constraint	Level	Notes
Agent framework familiarity	Moderate	Experience with LangChain/LangGraph patterns; notebook guide provides BaseAgent ReAct implementation
Ghostfolio domain experience	Learning	Need to deeply understand the NestJS API, Prisma schema, and portfolio calculation logic
Eval/testing frameworks	Moderate	Familiar with pytest/Jest patterns; notebook provides EvaluationFramework class
TypeScript/NestJS	Strong	Ghostfolio is TypeScript throughout
Python	Strong	For agent framework, eval, and observability tooling

---

Phase 2: Architecture Discovery

5. Agent Framework Selection

Decision: LangGraph (with LangChain tools)

Framework	Pros	Cons	Fit
LangGraph	State machines, cycles, complex multi-step flows, fine-grained control over tool routing, native LangSmith integration	Steeper learning curve than LangChain agents	**Best fit** — portfolio analysis requires multi-step reasoning with conditional branches
LangChain (agents)	Simpler API, extensive tool ecosystem, good docs	Less control over complex workflows, harder to debug multi-step chains	Good for MVP but limiting at scale
CrewAI	Multi-agent collaboration	Over-engineered for single-domain single-agent use case	Not needed
Custom (per notebook)	Full control, learning exercise	Reinventing the wheel for production features (retries, streaming, state)	Good for understanding, not for production

Justification:

- Portfolio analysis naturally maps to a **state graph**: user query → classify intent → select tools → execute → verify → format response
- LangGraph supports conditional edges (e.g., if confidence < threshold, route to human-in-the-loop)
- LangGraph's `StateGraph` pattern maps directly to the notebook's ReAct loop but with production-grade state management
- Native integration with LangSmith for observability

Architecture: Single Agent with Specialized Tool Nodes

- Not multi-agent — one agent with a well-defined tool registry is simpler, more debuggable, and sufficient for the use case
- State management via LangGraph's `TypedDict` state schema: conversation history, tool results, verification status, confidence scores

6. LLM Selection

Decision: Claude Sonnet 4.6 (primary) with Claude Haiku 4.5 (fallback for simple queries)

LLM	Pros	Cons	Role
Claude Sonnet 4.6	Excellent function calling, strong reasoning, good cost/performance ratio, large context window	Higher cost than Haiku	Primary reasoning engine
Claude Haiku 4.5	Very fast, very cheap	Less capable for complex multi-step reasoning	Simple lookups, classification
GPT-4o	Strong function calling	Different API, less cost-effective for this use case	Not selected
Open source (Llama 3)	Free inference	Weaker function calling, requires hosting	Not selected for MVP

Justification:

- Claude Sonnet 4.6 has strong structured output and function calling support
- The Anthropic SDK (Python) is mature and well-documented
- Context window (200K tokens) is more than sufficient for portfolio data + conversation history
- Cost: ~$3/M input, ~$15/M output — within budget constraints
- Ghostfolio's existing AI service uses OpenRouter, but we want direct provider integration for better control and lower latency

Function Calling Support: Claude's tool use is native and well-supported — structured JSON schemas for each tool, automatic parameter extraction, and reliable tool selection.

7. Tool Design

Required Tools (7 total — exceeds minimum of 5):

#	Tool Name	Description	Ghostfolio API Endpoint	Error Handling
1	`portfolio_analysis`	Get holdings, allocation %, sector/asset class breakdown, total value	`GET /portfolio/details`	Return cached data if API timeout; empty portfolio message if no holdings
2	`portfolio_performance`	Get performance metrics (TWR, max drawdown) over date range	`GET /portfolio/performance`	Validate date range; fallback to YTD if invalid range
3	`market_data_lookup`	Fetch current and historical prices for symbols	`GET /symbol/:dataSource/:symbol` + market-data endpoints	Handle unknown symbols gracefully; suggest alternatives
4	`transaction_history`	Retrieve and categorize activities (BUY/SELL/DIVIDEND/FEE/INTEREST/LIABILITY)	`GET /order`	Paginate large result sets; handle empty history
5	`tax_estimate`	Calculate estimated capital gains/losses and dividend income from activity history	Custom calculation on order data	Always append tax disclaimer; note

				jurisdiction limitations
6	`compliance_check`	Check portfolio against diversification rules and concentration limits	Custom rules on portfolio details	Configurable thresholds; warn vs. block
7	`benchmark_comparison`	Compare portfolio performance against benchmark indices	`GET /portfolio/performance` + benchmarks endpoint	Handle missing benchmark data; note time period alignment

Tool Design Principles (from notebook):

- Each tool does ONE thing (atomic)
- Each returns a structured `ToolResult` with status, data, message, and execution_time
- Each is safe to retry (idempotent — all are read-only GET operations)
- Each has clear documentation and JSON schema for the LLM
- Each handles errors gracefully with meaningful messages

External API Dependencies:

- Ghostfolio's own REST API (primary — all tools go through this)
- No direct external API calls — all market data flows through Ghostfolio's data provider abstraction

Mock vs. Real Data:

- Development: Mock data via Ghostfolio's seed database + custom test fixtures
- Testing: Deterministic mock responses for each tool to ensure reproducible eval results
- Production: Live Ghostfolio API with authenticated requests

8. Observability Strategy

Decision: Langfuse (primary) + Custom structured logging (fallback)

Tool	Pros	Cons	Decision
LangSmith	Native LangChain/LangGraph integration, excellent UI	Vendor lock-in, cost at scale	Runner-up
Langfuse	Open source, self-hostable, LangChain integration, traces + evals + datasets + prompts, free tier generous	Slightly less polished than LangSmith	**Selected** — open source aligns with project values; self-hosting avoids data privacy concerns for financial data
Braintrust	Good evals, CI integration	Less tracing capability	Not selected
Helicone	Proxy-based, easy setup	Limited eval features	Not selected

What We Track:

Metric	Implementation
Full request traces	Langfuse trace per conversation turn: input → LLM reasoning → tool calls → verification → output
Latency breakdown	Langfuse spans: `llm_call`, `tool_execution`, `verification`, `total_response`
Error tracking	Langfuse events with error category, stack trace, and conversation context
Token usage	Per-request input/output tokens + cumulative cost tracking
Eval results	Langfuse datasets + scores for regression detection
User feedback	Thumbs up/down captured in Langfuse as scores on traces
Cost tracking	Per-trace cost annotation (tokens × model pricing)

Real-time Monitoring:

- Langfuse dashboard for live trace inspection
- Alerts on: error rate > 5%, p95 latency > 15s, cost per query > $0.10, hallucination flags

9. Eval Approach

Decision: Hybrid — Langfuse Evals + Custom Python eval framework (inspired by notebook's EvaluationFramework)

How We Measure Correctness:

1. **Ground Truth Comparison** — For numerical outputs (allocations, returns, balances), compare agent response values against direct Ghostfolio API call results. Tolerance: ±0.1% for percentages, ±$0.01 for currency values.
2. **Tool Selection Accuracy** — For each test case, verify the agent selected the expected tool(s). Scored as exact match on tool name set.
3. **LLM-as-Judge** — For qualitative responses (analysis, recommendations), use a separate LLM call to score relevance, accuracy, and completeness on a 1-5 scale.
4. **Safety Checks** — Verify disclaimers are present, harmful requests are refused, and no hallucinated numbers appear.

Ground Truth Data Sources:

- Ghostfolio's seed database (deterministic portfolio data)
- Pre-computed expected outputs from direct API calls on test accounts
- Manual expert-reviewed expected responses for qualitative test cases

Automated vs. Human Evaluation:

- Automated: Correctness (numerical), tool selection, safety, latency, consistency (80% of eval)
- LLM-as-judge: Qualitative analysis quality (15% of eval)
- Human: Adversarial review, edge case validation (5% of eval — pre-launch)

CI Integration:

- Eval suite runs on every PR via GitHub Actions
- Regression detection: fail CI if pass rate drops below 80%
- Nightly full eval run with 50+ test cases

Test Suite Structure (50+ test cases):

Category	Count	Examples
Happy path	20+	"What is my portfolio allocation?", "Show my performance YTD", "What dividends did I receive this year?"
Edge cases	10+	Empty portfolio, single holding, unknown symbol, very large portfolio (100+ holdings), mixed currencies
Adversarial	10+	Prompt injection attempts ("ignore instructions and..."), requests for specific stock picks, attempts to bypass disclaimers, social engineering for account data
Multi-step	10+	"Compare my tech allocation to the S&P 500 and suggest rebalancing", "Calculate my tax liability and check if I'm over-concentrated in any sector"

10. Verification Design

Implement 4 verification types (exceeds requirement of 3+):

Verification	Implementation	Trigger
Fact Checking	Cross-reference every numerical claim against Ghostfolio API response data. If agent states "Your portfolio is worth $X", verify $X matches `portfolio/details` total value.	Every response containing numbers
Hallucination Detection	Parse agent response for numerical values and symbol names; verify each exists in the tool result data. Flag any number not traceable to a tool output.	Every response
Confidence Scoring	Score 0-1 based on: (1) tool call success rate in this response, (2) data freshness (market data age), (3) query complexity match to available tools. Threshold: 0.7.	Every response
Domain Constraints	Enforce: disclaimer presence on all advice, concentration limit warnings (>20% single holding, >40% single sector), tax estimate caveats.	Responses tagged as advice/tax/compliance

Escalation Triggers:

- Confidence score < 0.7 → Suggest user verify with professional
- Tax-related queries → Always append professional consultation disclaimer
- Portfolio changes suggested → Mark as "ideas for discussion" not "recommendations"
- Any verification failure → Log to Langfuse, return safe fallback response

---

Phase 3: Post-Stack Refinement

11. Failure Mode Analysis

Failure Mode	Probability	Impact	Mitigation
Tool API timeout	Medium	Medium	5-second timeout per tool call; retry once; return partial results with "data may be incomplete" warning
Ambiguous query	High	Low	Ask clarifying question before tool selection; use intent classification as first LangGraph node
LLM hallucination	Medium	High	Verification layer catches numerical hallucinations; source attribution required for all claims
Rate limiting (external APIs)	Low	Medium	Ghostfolio's caching layer (Redis) absorbs most; agent respects rate limits with exponential backoff
Invalid tool parameters	Medium	Low	JSON schema validation before tool execution; LLM retry with corrected params
Empty/null data	Medium	Medium	Each tool handles null gracefully; agent trained to say "no data available" rather than guess
Concurrent request overload	Low	Medium	Queue-based processing via Bull (Ghostfolio already uses Bull); max concurrent agent sessions = 10

Graceful Degradation:

1. If all tools fail → Return: "I'm unable to access your portfolio data right now. Please try again in a moment."
2. If LLM fails → Return: "I'm experiencing difficulties. Please try rephrasing your question."
3. If verification fails → Return the response but flag: "This response could not be fully verified. Please double-check the numbers."

12. Security Considerations

Threat	Mitigation
Prompt injection	System prompt hardening; input sanitization; never include raw user input in tool parameters without validation; LangGraph's structured state prevents prompt leakage
Data leakage	Agent only accesses authenticated user's data via JWT; no cross-user data access; Ghostfolio's existing permission system (`HasPermissionGuard`) enforced
API key management	LLM API keys stored as environment variables, never in code; Ghostfolio's `PropertyService` for runtime config; secrets in deployment platform (Vercel/Railway)
Audit logging	Every agent interaction logged to Langfuse with user ID (pseudonymized), timestamp, full trace; logs retained for 90 days
Financial advice liability	Every response includes disclaimer; agent refuses to give specific buy/sell recommendations; agent identifies as AI assistant, not financial advisor
Token exhaustion attack	Max iterations = 10; max tokens per response = 4096; cost limit per query = $0.10; circuit breaker for repeated failures (from notebook pattern)

13. Testing Strategy

Test Type	Tool	Coverage Target
Unit tests for tools	Jest (matching Ghostfolio's existing test infra)	Each of 7 tools: happy path, error path, edge cases = ~35 tests
Integration tests for agent flows	Custom Python eval framework + Langfuse	End-to-end flows: query → tool selection → execution → verification → response = ~20 flows
Adversarial testing	Custom test cases + manual red teaming	Prompt injection, data exfiltration attempts, unsafe advice requests = 10+ cases
Regression testing	GitHub Actions CI	Full eval suite on every PR; nightly run; fail on < 80% pass rate
Performance testing	Custom latency benchmarks	p50 < 3s, p95 < 10s, p99 < 15s for single-tool queries
Load testing	k6 or Artillery	50 concurrent users sustained for 5 minutes without degradation

14. Open Source Planning

What We Release:

1. **AgentForge Finance Package** — Reusable NPM package containing: 7 finance tools with schemas, verification layer, eval framework, and Langfuse integration. Published to npm as `@agentforge/ghostfolio-agent`.
2. **Eval Dataset** — 50+ test cases with expected outcomes, published as a public dataset (JSON format) on GitHub for others to benchmark financial agents.

Licensing: AGPL-3.0 (matching Ghostfolio's license for compatibility)

Documentation:

- README with setup guide, architecture diagram, and quickstart
- API documentation for each tool
- Eval dataset documentation with test case format specification
- Contributing guide

Community Engagement:

- Open PR to Ghostfolio for AI agent integration
- Share on X/LinkedIn with demo video
- Respond to issues/PRs within 48 hours during project week

15. Deployment & Operations

Layer	Technology	Notes
Agent backend	Python/FastAPI service	Separate microservice; communicates with Ghostfolio API over HTTP
Hosting	Vercel (frontend) + Railway (agent backend)	Vercel for the Ghostfolio Angular app; Railway for Python agent service with auto-scaling
CI/CD	GitHub Actions	Lint → Test → Eval → Build → Deploy pipeline

Monitoring	Langfuse (traces) + Railway metrics (infra)	Dashboard for agent health, latency, error rates, costs
Rollback	Git-based deployment rollback via Railway	One-click rollback to previous deployment
Secrets	Railway environment variables	ANTHROPIC_API_KEY, LANGFUSE_PUBLIC_KEY, LANGFUSE_SECRET_KEY, GHOSTFOLIO_API_URL, GHOSTFOLIO_API_TOKEN

16. Iteration Planning

Feedback Collection:

- Langfuse trace-level scoring (thumbs up/down on each response)
- Optional free-text feedback field in UI
- Automated detection of user reformulations (same intent, different phrasing = likely poor first response)

Eval-Driven Improvement Cycle:

1. Run nightly eval suite → identify failing test cases
2. Analyze failure patterns in Langfuse traces
3. Fix: prompt tuning, tool schema refinement, or verification rule adjustment
4. Add new test cases for discovered failure modes
5. Re-run eval → confirm improvement without regression

Feature Prioritization (Post-MVP):

1. Streaming responses for better UX
2. Multi-language support (Ghostfolio already supports i18n)
3. Proactive alerts (e.g., "Your portfolio concentration exceeded 40% in tech")
4. Integration with Ghostfolio's existing AI prompt system
5. Support for "what-if" portfolio simulations

Long-Term Maintenance:

- Monthly dependency updates
- Quarterly eval dataset expansion
- Model version upgrades tracked via Langfuse prompt versioning
- Community PRs reviewed and merged on rolling basis

Summary: Technology Stack Decision Matrix

Layer	Selection	Rationale
Agent Framework	**LangGraph**	State machine control for multi-step financial reasoning; conditional routing for verification

LLM	**Claude Sonnet 4.6** (primary) / **Claude Haiku 4.5** (simple queries)	Best function calling + reasoning at reasonable cost
Observability	**Langfuse**	Open source, self-hostable, excellent tracing + evals; aligns with project's open-source values
Evals	**Langfuse Evals + Custom Python Framework**	Hybrid: automated numerical checks + LLM-as-judge for qualitative
Backend	**Python / FastAPI**	LangGraph is Python-native; FastAPI for high-performance async API
Frontend Integration	**Ghostfolio Angular App** (existing)	Chat widget integrated into Ghostfolio's existing UI
Database	**PostgreSQL** (Ghostfolio's existing via Prisma)	Agent reads from Ghostfolio's database through its API layer
Deployment	**Vercel + Railway**	Vercel for frontend; Railway for Python agent microservice
Testing	**Jest (tools) + Custom eval (agent) + GitHub Actions (CI)**	Match Ghostfolio's existing test infra + add agent-specific eval

---

Ghostfolio Codebase Key Reference

Repository Structure:

```
ghostfolio/ ├── apps/ │ ├── api/ # NestJS backend (REST API) │ │ └── src/app/ │ │ ├── portfolio/ # Core: holdings, performance, dividends │ │ ├── order/ # Activities (BUY, SELL, DIVIDEND, etc.) │ │ ├── account/ # User accounts and balances │ │ ├── endpoints/ │ │ │ ├── ai/ # Existing AI prompt generation │ │ │ ├── benchmarks/ │ │ │ ├── market-data/ │ │ │ └── watchlist/ │ │ ├── exchange-rate/ # Currency conversion │ │ ├── symbol/ # Asset symbol lookup │ │ └── auth/ # JWT + OAuth + WebAuthn │ └── client/ # Angular 21 frontend ├── libs/ │ ├── common/ # Shared types, interfaces, permissions │ └── ui/ # Shared UI components ├── prisma/ │ └── schema.prisma # Database schema (14 models, 9 enums) └── docker/ # Docker deployment config
```

Key Prisma Models: User, Account, Order (activities), SymbolProfile, MarketData, AccountBalance, Tag, Platform, Access, Subscription

Key Enums: AssetClass (6 types), AssetSubClass (10 types), DataSource (9 providers), Type (6 activity types: BUY, SELL, DIVIDEND, FEE, INTEREST, LIABILITY)

Existing AI Integration: `apps/api/src/app/endpoints/ai/ai.service.ts` — Generates portfolio analysis prompts via OpenRouter; provides foundation to build upon.

Authentication: JWT-based with Google OAuth, OIDC, and WebAuthn support. Permission guard system (`HasPermissionGuard`) enforces role-based access.

This document fulfills the Pre-Search checklist requirements (Phases 1-3) from the AgentForge project specification. It is based on systematic exploration of the Ghostfolio repository, the AI_Agents_Complete_Guide notebook, and the AgentForge project brief.