**Product Requirements Document (PRD) — Phase 1**

**Product:** Multi-Agent Investment Advisor  
**Phase 1 UI:** CLI (Phase 2 will be Streamlit)  
**Date:** Aug 26, 2025  
**Owner:** [You / Team]

**1) Summary & Objectives**

Build a **CLI-first**, multi-agent investment advisory system that:

* Orchestrates agents with **LangGraph** (state-machine/graph execution).
* Exposes tools via **Model Context Protocol (MCP)** for consistent, auditable access to web search and market data.
* Uses **Groq LLM API** for all reasoning and content generation.
* Delivers an end-to-end workflow: **gather company news → gather financials → analyze → recommend**.

The current prototype implements a similar workflow using **CrewAI**, **DuckDuckGo**, and **Yahoo Finance (yfinance)** with OpenAI models. It defines four agents (news/info, financial data, analyst, recommender) and analogous tasks executed sequentially.

**Success Criteria (Phase 1):**

* Single-command CLI produces:
  1. **Analysis.md** (company/stock analysis)
  2. **Recommendation.md** (Buy/Hold/Sell + rationale)
* All agent tool calls executed via **MCP tools** (search, financials).
* Orchestration via **LangGraph** (no CrewAI in runtime).
* **Groq** replaces OpenAI as the LLM provider.

**2) Scope (Phase 1)**

**In Scope**

* **Orchestration:** LangGraph graph with four agent nodes plus an orchestrator/supervisor.
* **Agents & Roles:**
  1. **News/Info Agent** – fetches latest company news.
  2. **Financial Data Agent** – retrieves snapshot/financials.
  3. **Analyst Agent** – synthesizes news + financials into analysis.
  4. **Recommendation Agent** – outputs investment stance & reasons.  
     *(Matches the roles in the prototype.)*
* **MCP Tools:**
  1. search.duckduckgo (company news & context).
  2. finance.yahoo.price (current price).
  3. finance.yahoo.company\_info (profile / snapshot).
  4. finance.yahoo.income\_statements (financials).  
     *(Prototype currently uses duckduckgo-search and yfinance directly.)*
* **LLM:** Groq API (e.g., llama-3.1-70b, mixtral-8x7b, or equivalent).
* **CLI UX:** single command (e.g., invest-advisor --ticker AAPL) + optional flags (market suffix, output dir, date override).
* **Outputs:** Analysis.md, Recommendation.md (as in prototype).
* **Config:** .env (Groq API key), YAML/JSON for MCP tool endpoints & LangGraph settings.
* **Basic Observability:** structured logs of each node transition, tool call, and token usage.

**Out of Scope (Phase 1)**

* Streamlit UI (planned for Phase 2).
* Portfolio optimization, backtesting, or trade execution.
* Coverage beyond equities (e.g., funds/crypto) unless trivially supported by data tool.
* Multi-ticker batch runs (Phase 1 is single-ticker per invocation).

**3) Users & Use Cases**

**Primary User:** Individual investor or analyst who wants a fast, explainable snapshot and recommendation from public web/news + fundamentals.

**Top Use Cases**

1. “Give me a **concise thesis** on $TICKER with the latest news and a clear buy/hold/sell view.”
2. “Compare **52-week levels and current price** with recent material events that may affect the thesis.”
3. “Provide **explainable reasons** and point to news sources (headlines + dates).”

**4) Functional Requirements**

**4.1 CLI**

* invest-advisor --ticker TICKER [--market-suffix .NS] [--out ./reports] [--date 2025-08-26] [--verbose]
* Runs LangGraph execution; saves Markdown outputs to disk.

**4.2 LangGraph Orchestration**

**Graph Nodes (Agents) & Edges**

* **Start → NewsAgent → FinancialAgent → AnalystAgent → RecoAgent → End**
* Retry edge on transient tool/HTTP errors (configurable backoff).
* Orchestrator handles **context passing** (news → analysis; financials → analysis; analysis → recommendation).  
  *This mirrors the prototype’s Task context chain (financials+news → analysis → recommendation).*

**4.3 Agents (Prompts & Constraints)**

* **News/Info Agent**
  + Goal: “Gather and summarize latest company news and business information.”
  + Tools: search.duckduckgo
  + Requirements: include **headline, date, brief impact**. Deduplicate similar stories.
* **Financial Data Agent**
  + Goal: “Gather financial snapshot (price, market cap, PE/EPS, 52-week range, moving averages, cash/EBITDA if available) and income statements.”
  + Tools: finance.yahoo.price, finance.yahoo.company\_info, finance.yahoo.income\_statements
  + Notes: handle suffix (e.g., .NS) toggle, as prototype attempts both with/without suffix.
* **Analyst Agent**
  + Goal: “Produce a **comprehensive analysis** combining news and fundamentals; call out risks, valuation context, trend direction.”
  + Output file: Analysis.md (prototype writes analysis to a file).
  + Formatting: sections for **Overview, Financials, Recent News, Risks, View**. (Remove India-specific “lakh/crore” unless locale flag is set; prototype uses Indian units.)
* **Recommendation Agent**
  + Goal: “Recommend **Buy/Hold/Sell** with rationale, time horizon, and key assumptions.”
  + Tool: finance.yahoo.price (for current price confirmation pre-decision).
  + Output: Recommendation.md (as in prototype).

**4.4 MCP Tooling (Phase 1)**

* Implement MCP servers that wrap:
  + **DuckDuckGo**: query, return JSON list {title, url, snippet, date?}.
  + **Yahoo Finance** (yfinance under the hood):
    - company\_info(symbol) → cleaned JSON subset similar to prototype (shortName, symbol, regularMarketPrice, marketCap, sector, industry, eps, pe, 52w low/high, ma50, ma200, cash, ebitda, margins, etc.).
    - income\_statements(symbol) → JSON (prototype returns DataFrame.to\_json(orient="index")).
    - current\_price(symbol) → float/string.
* Each tool returns **typed JSON** with minimal, documented schema.
* Agents call tools only via MCP.

**4.5 LLM (Groq)**

* Configurable model name, temperature, max tokens.
* System prompts enforce **citations for news** (include URLs) and **clear reasoning → concise outputs**.

**4.6 Output Artifacts**

* Analysis.md: structured analysis with sections, bullet points, and a short **TL;DR**.
* Recommendation.md: stance + **3–5 supporting reasons**, explicit **price context**, and top **watch-outs**.
* Include **“Data Freshness”** line (UTC timestamp of tool calls).

**5) Non-Functional Requirements**

* **Reliability:** Retries with exponential backoff on MCP tool timeouts; graceful skips if a single tool fails (still produce a report with caveats).
* **Performance:** End-to-end CLI run completes in **≤ 45s** for common tickers on a healthy network.
* **Security:** API keys only via environment variables. No PII. Respect robots/ToS of data sources.
* **Observability:**
  + Structured logs: node transitions, tool calls (name, latency), token usage.
  + Optional --verbose echoes each step summary to console (no raw prompt tokens).
* **Portability:** Minimal assumptions so Phase 2 UI can reuse the same LangGraph+MCP core.

**6) Data Sources & Contracts**

* **DuckDuckGo Search (News):** keyword: <COMPANY or TICKER> + time filter (last 7–14 days).
* **Yahoo Finance (via yfinance):**
  + company\_info: returns cleaned subset used in prototype (price, cap, sector, margins, etc.).
  + income\_statements: JSON oriented by index (as in prototype).
  + current\_price: numeric.

**7) Execution Flow (Happy Path)**

1. **CLI** parses args → initializes LangGraph + loads MCP tools + Groq.
2. **NewsAgent** calls search.duckduckgo → returns top N recent items → summarizes.
3. **FinancialAgent** calls company\_info, income\_statements, and current\_price (tries symbol then symbol+suffix if configured) → builds a structured snapshot (price, PE/EPS, ranges, margins, cash/EBITDA). *(Prototype attempts with/without “.NS”.)*
4. **AnalystAgent** merges results → writes Analysis.md. *(Prototype writes to file via Task output\_file.)*
5. **RecommendationAgent** checks current\_price → outputs Recommendation.md with stance + reasons. *(Prototype mirrors this step.)*

**Error Paths:**

* Missing news → continue with financials; add caveat in Analysis.
* Missing financials → proceed with news; recommendation must state “insufficient fundamentals.”
* Tool failure → retry; after retries, skip and log.

**8) Acceptance Criteria**

1. **CLI:** invest-advisor --ticker AAPL produces **both** Analysis.md and Recommendation.md with timestamps and tool citations (URLs for news).
2. **Agents & Orchestration:** Execution runs inside **LangGraph** (unit test asserts graph path taken).
3. **MCP Only:** All external calls come through **MCP tools** (grep/log validation).
4. **Groq:** All model calls use Groq API (integration test with GROQ\_API\_KEY).
5. **Suffix Handling:** When --market-suffix .NS is supplied (e.g., RELIANCE), system attempts with and without suffix and succeeds if either returns data. *(Mirrors prototype behavior.)*
6. **Runtime:** Typical run ≤ 45 seconds on a standard connection.
7. **Resilience:** If one tool fails, outputs still generated with clear caveats.

**9) Milestones (Phase 1)**

* **M1 – Project Skeleton (LangGraph + Groq wired):** 3–4 days
* **M2 – MCP Servers (search & finance) + Contracts:** 4–5 days
* **M3 – Agents & Prompts + Graph Edges:** 3 days
* **M4 – CLI + Config + Logging:** 2–3 days
* **M5 – Tests (unit/integration) + Docs:** 3 days
* **M6 – UAT & Sign-off:** 1–2 days

**10) Risks & Mitigations**

* **Data API Instability (yfinance)** → Implement retries; cache last known stable snapshot.
* **News Quality/Timeliness** → Use filters (recent only), dedup by URL/domain.
* **LLM Cost/Latency (Groq)** → Set sensible token limits; allow --fast mode with shorter summaries.
* **Market Suffix Ambiguity (.NS)** → Attempt both forms; surface which one succeeded (prototype does similar).

**11) Open Questions**

1. **Model choice on Groq** (default & fallback): e.g., llama-3.1-70b primary, mixtral-8x7b fallback?
2. **News date filtering**: last 7 vs 14 days?
3. **Locale/Units**: Prototype uses Indian number units (“lakh/crore”). Keep global/US format by default with optional --locale IN?
4. **Compliance/Attribution**: Required disclaimer language for recommendations?

**12) Deliverables (Phase 1)**

* **Code:**
  + LangGraph graph (graph.py) with 4 agent nodes + orchestrator.
  + MCP servers (mcp\_search.py, mcp\_finance.py).
  + CLI (cli.py): invest-advisor entrypoint.
  + Config: .env.sample, config.yaml (model, tools).
* **Docs:** README.md (install, env, run), **Architecture diagram**, **Tool schemas**.
* **Outputs:** Analysis.md, Recommendation.md.
* **Tests:** unit tests for nodes/tool contracts; end-to-end CLI test.

**Prototype → Target Mapping (for implementers)**

* **CrewAI (agents/tasks/crew)** → **LangGraph** (nodes/edges/state)
* **DuckDuckGoSearchRun tool** → **MCP: search.duckduckgo**
* **yfinance tools (price/info/income)** → **MCP: finance.yahoo.\***
* **OpenAI model fields** → **Groq LLM config** (model name, temp, tokens)
* **Output files (Analysis.md / Recommendation.md)** → **Same** (Phase 1 deliverables)

If you’d like, I can also provide:

* a **LangGraph state diagram** (mermaid),
* **MCP tool JSON schemas**, and
* a **starter folder structure** with stubbed modules.