MARKETING.MBA

End-to-end Documentation: LangGraph Marketing Intelligence Agent

(Confidential; For Marketing.MBA use only)

PREPARED BY	**NAVID BIN AHMED**
SUBMITTED TO	**SABRINA GROSS**
DOCUMENT STATUS	**CONFIDENTIAL**
DOCUMENT VERSION	**1.0**
REVISION VERSION	**0**
REVISED BY	**N/A**
LAST UPDATED	**OCT 31, 2025**
SUBMITTED ON	**OCT 31, 2025**

Table of Contents

Intelligent LangGraph Marketing Agent

1. Introduction

This is a detailed documentations for the full-stack LangGraph-based AI agent for performing marketing orchestration workflow, intended to develop for the **employer** : Marketing.MBA, Dubai, UAE, by the ensuing **Employee:** Navid Bin Ahmed, Full-Stack AI Engineer (Applied Data Scientist), Dhaka, Bangladesh – Head of AI's Right Hand.

---

2. Requirements Gathering

Title:

Design and Simulate a Mini Marketing Reality

Description:

Design and implement a mini LangGraph agent that performs a small, realistic "marketing orchestration" workflow.

Example Prompt:

"Analyze current market trends for AI-powered design tools and generate 3 positioning insights with supporting data."

The Agent Should:

- Take natural language input

- Call at least one external data source (Tavily, SerpAPI, or local mock dataset)

- Use structured state or subgraphs to separate "research", "analysis", and "summary" reasoning steps

- Return a structured JSON output with:

Insights (3 bullet points)

Sources (citation or dummy data)

Thought_trace (short reasoning summary)

Deliverables:

- GitHub rep with code + ReadME.md explaining:
  - Architecture (graph layout, node, state schema)
  - Design choices and trade-offs
  - What you'd improve for production (e.g., memory, parallelization, error handling)

- Optional short Loom / Video: 2-min walkthrough (bonus)

Constraints:

- You can use AI assistants: we encourage it. Make sure to document your approach in regards to which tool you use and why; how you prompt them and what context you give

---

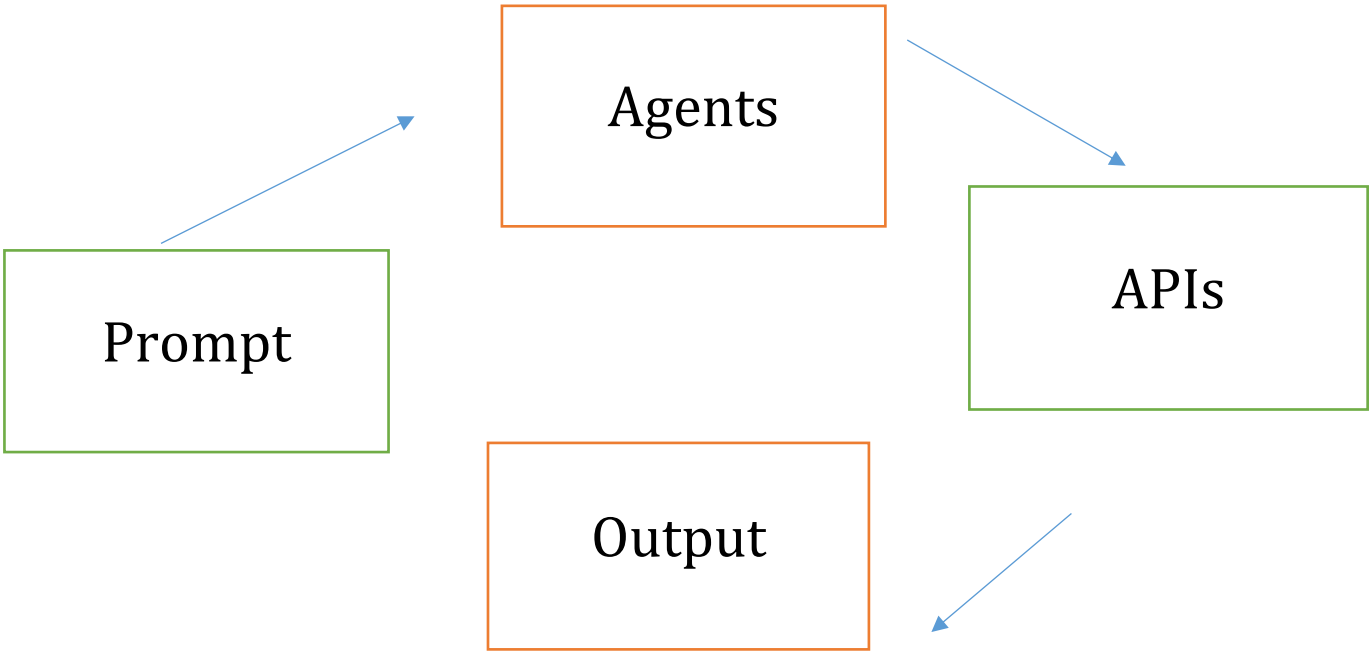
3. Working Procedure

- **Prompt:** User provides natural language prompt

- **External API:** Agents trigger SerpAPI to gather news and Google Gemini 2.5-flash API for LLM

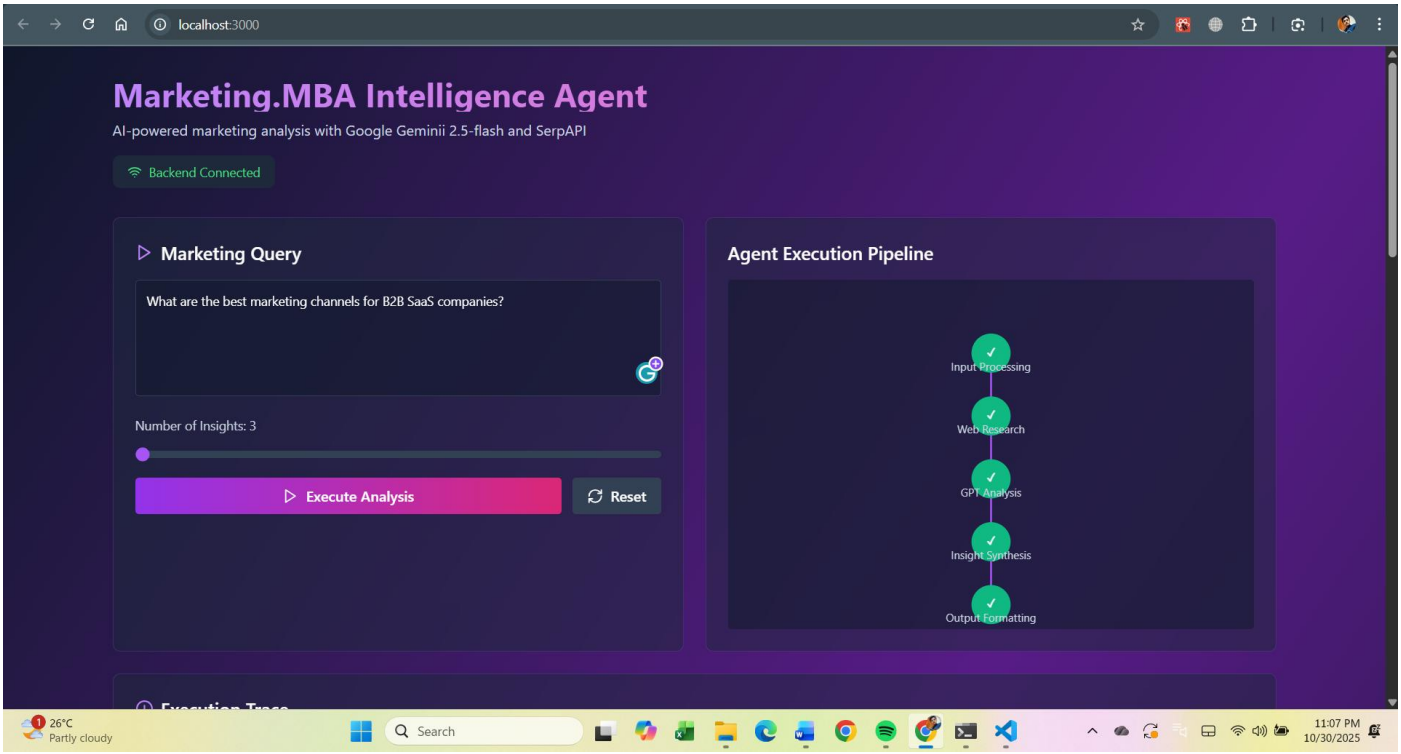
- **Agent** Subgraphs (explicit state & transitions): research → analysis → summary/insights

- **Output Response:** structured JSON with `insights` (n numbers>=3), `sources`, and `thought_trace` (short reasoning steps)



---

4. Demo-graph



4.1 Sample Response: Output

Execution Trace

Json```

INPUT 9:50:13 PM

{
"query": "What are the best marketing channels for B2B SaaS companies?",
"max_results": 3,
"status": "Processing query parameters"
}

RESEARCH 9:50:14 PM

{
"status": "Searching web with SerpAPI",
"search_queries": [
"What are the best marketing channels for B2B SaaS companies?"
],

"progress": "Fetching search results..."
}

ANALYSIS 9:50:17 PM

{
"status": "Analyzing with Google Gemini",
"model": "2.5-flash",
"progress": "Generating insights..."
}

SYNTHESIS 9:50:21 PM

{
"status": "Synthesizing insights",
"progress": "Formatting results..."
}

OUTPUT 9:50:22 PM

{
"status": "Preparing final output",
"progress": "Complete"
}

Analysis Results

- **Insights**
- **Sources**
- **Thought Trace: Metrics**

```json
{
"research_phase": "Retrieved 3 sources",
"analysis_phase": "Generated 3 insights",
```

"synthesis_phase": "Average confidence: 0.89",
"quality_metrics": {
"total_insights": 3,
"processing_time": 20.15,
"timestamp": "2025-10-30T15:50:33.529482"
}
}

---

5. Tech-Stacks & Software Used

Tech Used:

- Langgraph
- FastAPI
- React.js
- SerpAPIs & Google Gemini 2,5-Flash API

Cloud and software tools used:

- Render
- VS Code
- GitHub
- Mermaid
- **AI Assistants:** ChatGPT, Claude, YouTube, Stack OverFlow

5.1 Usefulness of AI Assistants: Speed, Modernization, and Reliability

- Quick error handling and finding solution. Multiple solutions are provided from which the most acceptable solution was applied for a robust and fastest development

- Check for security vulnerabilities. SQL injection and other threats may occur and so a thorough security analysis was done based on the best practices and advanced tentative solutions. However, most of which is associated to 'further extension', in a later section

- Embracing what is out of idea. Learning is life-long. AI Assistants help gain a speedy learning curve easily, to the point, and on-the-go. Better than boring video tutorials or books.

- **Example prompt 1:** while working with a paid old OpenAI API

I use paid openai api which is no longer accessible. Kindly share alternative free apis that should work.

- **Example prompt 2:** while interfacing and validating output on the React UI

In FastAPI Full Stack project, one API associated with research agent works fine. But another agent's output - analysis marketing insights/summary associated with another OpenAI API is missing. Suggest what causes the issue and recommend solution?

5.2 Reasons for Choosing Specific AI Assistants

- **ChatGPT:** easier to access, has a freemium version. Useful for everyday use-cases. I love it because it knows about me more than that I do. (Laughing out loud)

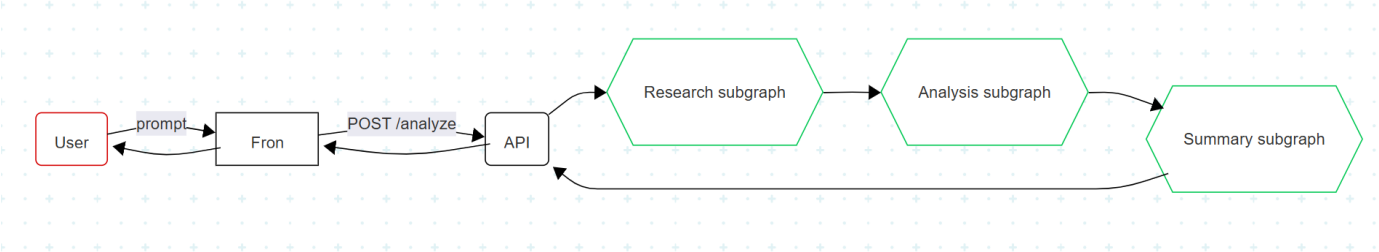
- **Claude.AI:** Better for software dev, as it illustrates a solution graphically. Easier to catch the underlying concepts on Claude. But very limited access in freemium version.

- **Other Tools/Sites:** Google is the best friend! Search results came up with the personalized solutions in AI mode version.

---

6. Architecture

6.1 Architecture: Full-Stack System Architecture



6.1.1 How It Works (End-to-End)

1. Frontend (React) sends POST `/analyze` with a natural language prompt.
2. FastAPI backend receives the prompt and runs the `LangGraphAgent`.
3. Agent performs three subgraphs (explicit state):

   - **research**: call SerpAPI and produce `raw_results` + `sources`.

   - **analysis**: call Google Gemini 2.5-flash API and prduce LLM output - `insights`.

- **summary**: map signals to 3 concise insights; produce `thought_trace`.

4. Backend returns structured JSON with `insights`, `sources`, `thought_trace`, and `raw_research`.

5. Frontend displays results.

6.2 Architecture: Multi-Agent Architecture

Multi-Agent Horizontal (Linear) Unidirectional Architecture

Some major takeaways:

- Each node operates on the Shared State model.

- The graph enforces deterministic execution in linear and forwarding order.

6.2.1 Agent Nodes & Responsibilities

Input Processing (root):

- Receives validated input from user, orchestrates subgraphs, maintains explicit `state` dict and `thought_trace`. Handles concurrency control for external calls.

Web Research subgraph (ReserachNode):

- Node: Query generator, Parallel search workers (bounded concurrency), Raw results aggregator.

- Output: `raw_results` (list of search hits).

GPT Analysis subgraph (AnalysisNode):

- Node: Candidate extractor (LLM), Scorer, Deduplicator / Selector, Source linker.

- Output: `candidate_points`, `point_scores`, `selected_points`.

Insight Synthesis/ Summary subgraph (SummaryNode):

- Node: Formatter (insights), Confidence estimator, packaging outputs. - Output: final `insights`, `sources`, `thought_trace`, `confidence_score`

Output:

- Displays output on the UI

6.2.2 Agent State schema (explicit)

`state` is a nested dict recorded across subgraphs:

```json
{
"input": { "prompt": "...", "max_results": ... },
  "research": { "queries": [...], "raw_results": [...] },
  "analysis": { "candidate_points": [...], "point_scores": [...] },
  "summary": { "insights": [...], "sources": [...] }
}```
```

7. Design Choices & Trade-offs

a. **Architecture:** For agent, a simple linear horizontal architecture suits the purpose as the requirement is pretty straightforward with no backward dependencies.
Full-stack microservice architecture was preferred for the end-end development as this gives flexibility for debugging and easier to implement. I focus on full-stack AI also because, it gives me control on the complete product.
Used separate React components for listing sources, making summarization, and loading spinners.

Trade-off: As of now, the architectural choices are standard and industry-ready. With scaling, we can go for hybrid or vertical architecture for agents.

b. **APIs**: SerpAPI for Real-world data from web search results and Google Gemini 2.5-flash API for LLM that summarizes insights based on the search results were preferred.
External requests are async and capped (bounded concurrency) to avoid overloading API or hitting rate limits.

Trade-off: Both are freemium. We can use paid APIs for an extended time period and more polished output.

c. **Explicit subgraphs** (Research/Analysis/Summary): Help visibility into intermediate state and debugging. Mirrors how LangGraph splits concerns and progresses between agents enabling transparency & debugging; each subgraph writes into state.

Trade-off: bigger payload and more surface area to manage; mitigated by concise, purposeful logging user Python Logger.

d. **Modular subgraphs:** Separate responsibilities (research/analysis/summary) → easier to replace components

Trade-off: overhead of interfaces between subgraphs.

e. **Reasoning visibility:** thought_trace captures high-level decisions (e.g., query formation, failures, selection), makes it easier to trace where the possible issues lie, if any.

State captures raw inputs/outputs per subgraph — this is critical for auditability & for later storing as memory.

Trade-off: As of now, the reasoning is fine for this use case.

f. **Error handling & resilience:** Graceful fallback to mock sources if external calls fail.

Trade-off: Incorporated exception-handling and with extended edge cases it can be matured.

---

8. Strong Aspects of Current Version: Full-Stack Multi-Agent Intelligence

a. Production-grade Full-Stack Code: modular, clean and reusable code. Attention was given to address exceptions gracefully.

Full-stack development (end-to-end) has been carried out maintaining high caliber, enthusiasm and standard.

b. Test Scripts: includes test scripts with detailed edge cases for API (Gemini)

c. External API integrated: For web research and LLM result generation, official APIs have been used. Users will experience a genuine and real-time data with richer summarization and confidence calibration.

d. Industry Standard Maintained: Sufficient exception handling considering the use case, full-stack development with visually appealing UI.

e. Security & Best Practices Maintained: Secrets via environment variables (.env) for dev environment (recommended advanced secrets manager), and input validation with Pydantic (AgentInput) have been maintained.

Timeouts and retries on external calls (tenacity), and minimal CORS configuration have been enabled (recommended OAuth (JWT/API key) for production).

---

9. Scope of Improvement for Production: Possible Scaling

a. DataBase, Memory Storage & Caching Integration:

- yes, cache results from APIs for repeated prompts can be used in DB.
- persisted agent state or a vector DB for long-term memory can be used.
- Rate limiting can be used.
- Persist state and thought_trace to a DB (Postgres) or vector DB for later recall and personalization can further be used.
- Redis could have been used here, but it comes with 30 MB storage and Render DB is free for one month.

b. Parallelization and event-driven orchestration:

- Fetch SerpAPI along with other news/data sources in parallel (fetch multiple APIs for a single agent).
- Run analysis pipelines concurrently for multiple signals.
- Use a task queue (Celery/RQ) or serverless functions to scale heavy tasks, and return job IDs for long-running runs.

c. Robustness & error handling:

- Retry/backoff for external calls, timeouts, partial success handling.
- Input validation, rate-limiting and quotas, observability (tracing, metrics).

d. Virtualization and Containerization

- Integration of CI/CD pipelines and virtualizion using Docker and Kubernetes could have been implemented.

f. Robust deduplication & clustering

- Use semantic embeddings + clustering to merge similar insights.

g. Extended Security & infra

- Add JWT / OAuth2 authentication for user-level access and per-user quotas.

- Rate-limiting, monitoring (Prometheus), structured logging, extended error tracking (Sentry).

- Use secrets manager (AWS Secrets Manager / HashiCorp Vault) for SerpAPI and Google Gemini API keys instead of .env for development

- Dependency safety by keeping dependencies pinned and run vulnerability scans (Snyk/bandit).

h. Extended Testing with More Edge Cases

- Unit tests for orchestrator logic, integration tests that mock SerpAPI and Google Gemini API, end-to-end tests (Playwright) for UI.

i. Advanced Observability

- Add tracing (OpenTelemetry) across service calls and subgraph transitions.

j. Possible Feature Extension

- PDF extractor

- Secure sign-in to use the platform for limiting unauthorized access

- Save previous prompts and enable auto-suggestions

---

10. Run the Project Locally

clone the repo:

```
git clone https://github.com/NavidBinAhmed/AI-agent-marketing.git
```

Backend:

```
cd backend
```

```
python -m venv mmba && source .mmba/source/activate
```

```
pip install -r requirements.txt
```

```
uvicorn app.main:app --reload (open http://localhost:8000)
```

Frontend:

```
cd frontend
```

```
npm install
```

```
npm run dev (open http://localhost:3000)
```

---

11. Similar Product Ideas to Strategize Traditional Sales and Marketing Division

a. AI Agent for lead generation, qualification, and enrichment of sales, using conversational lead management

b. Sales operation and production Agent, using automated CRM, meeting preparation, note-taker, drafting proposal, and contract creation

c. Sales engagement and coaching Agent, using real-time voice-over calls, personalized email outreach, and analyze output

d. Agent for content creation and optimization, using automated visual asset generation and SEO optimization

e. Agent for campaign management and personalization, using ad optimization and automated campaign execution following user demands

f. Agent for market and competitive intelligence, using autonomous tracking and alert along with reputation or sentiment analysis and tracking

---

12. Issues Encountered: Trial & Error

- Tailwind CSS setup: upgraded from v3 to v4
- API integration: Switched to Gemini from OpenAI
- Insights generation issue
- Sources listing issue,

and more

---

13. Usage & Cost Analysis

API	Free Requests Limit	Used (Oct -31)	Price
Gemini 2.5-flash	250/day, 10/min	15	Free
SerpAPI	250/month	11	Free

_____ ** _____

About Marketing.MBA

Marketing.MBA, a subsidiary of Creator Group Holdings is a UAE-based organization that acts as a strategic marketing coach and business development consultancy for marketing professionals and corporate marketing teams. Marketing.MBA produce bold marketing ideas, implement strategic digital performance sales processes, and train individuals and enterprise brands to optimize their digital acquisition channels for maximum returns on their marketing investments.