Product Requirements Document (PRD): Market Intelligence Swarm

1. Overview

Product Name

Market Intelligence Swarm - An AI-powered competitive intelligence platform

Vision Statement

To democratize market intelligence by providing real-time, actionable insights through automated data aggregation and AI analysis, enabling businesses to make informed strategic decisions without expensive consulting services.

Problem Statement

Businesses struggle to:

- Monitor competitors effectively in real-time
- Process vast amounts of market data efficiently
- Separate signal from noise in financial news and social media
- Access affordable, comprehensive market intelligence
- Generate actionable insights from unstructured data

Solution

A dual-architecture platform combining:

1. **Web Dashboard**: Real-time market sentiment visualization
2. **AI Agent System**: On-demand competitive analysis

2. Product Architecture

2.1 Dual-Mode System

```
Market Intelligence Swarm
├── MODE 1: Continuous Monitoring (Flask App)
│   ├── Real-time data streams (RSS, Reddit, Financial APIs)
│   ├── Sentiment analysis engine
│   ├── Dashboard visualization
│   └── REST API
│
└── MODE 2: On-Demand Analysis (LangGraph System)
    ├── AI agent swarm (Researcher + Analyst)
    ├── Web search integration
    ├── Local LLM processing
    └── Strategic report generation
```

2.2 Core Components

Component A: Flask Web Application

- **Purpose**: Continuous market monitoring
- **Key Features**:
    - Real-time sentiment tracking
    - Trending stock identification
    - Multi-source data aggregation
    - Historical data caching
    - Web dashboard interface

Component B: LangGraph Agent System

- **Purpose**: Deep-dive competitor analysis
- **Key Features**:
    - Multi-agent workflow orchestration
    - Live web research capability
    - Strategic analysis generation
    - Local LLM integration (Ollama)
    - Structured report output

3. User Personas

Primary User: Small/Medium Business Owner

- **Needs**: Competitive insights without large budgets
- **Use Case**: Weekly competitor check, market entry decisions
- **Technical Level**: Basic

Secondary User: Financial Analyst

- **Needs**: Quick sentiment analysis, trend spotting
- **Use Case**: Daily market scanning, report supplementation
- **Technical Level**: Intermediate

Tertiary User: Startup Founder

- **Needs**: Market positioning insights
- **Use Case**: Investor pitch preparation, competitive landscaping
- **Technical Level**: Basic-Intermediate

4. Functional Requirements

4.1 Core Features

FR-001: Real-time Market Monitoring

Description: Continuous collection of market data from multiple sources
Acceptance Criteria:

- Collects from ≥3 RSS feeds every 5 minutes
- Monitors ≥4 Reddit communities
- Tracks major market indices (S&P 500, NASDAQ, DJIA)
- Caches data for 10 minutes minimum
- Processes ≥100 articles per update

FR-002: Sentiment Analysis Engine

Description: Analyze sentiment from text content
Acceptance Criteria:

- Classifies sentiment as positive/negative/neutral
- Tracks sentiment per stock symbol
- Calculates percentage positive sentiment
- Processes both news and social media content
- Updates sentiment in real-time

FR-003: Trending Stock Detection

Description: Identify frequently mentioned stocks
Acceptance Criteria:

- Extracts stock symbols from text
- Ranks stocks by mention frequency
- Displays top 20 trending stocks
- Filters invalid symbols (<6 characters)
- Updates trending list every 5 minutes

FR-004: AI Agent Research System

Description: On-demand competitor analysis
Acceptance Criteria:

- Accepts company name input
- Searches web for latest information
- Processes raw data through LLM
- Generates structured report
- Provides strategic recommendations

FR-005: Web Dashboard

Description: User interface for data visualization
Acceptance Criteria:

- Displays real-time sentiment metrics
- Shows trending stocks table
- Presents top news articles
- Visualizes market index performance
- Provides force-refresh capability

4.2 API Requirements

FR-006: REST API Endpoints

- GET /api/intelligence - Get current market intelligence
- GET /api/intelligence/refresh - Force data refresh
- GET /api/health - System health check
- **Response Format**: JSON with timestamp, data, metadata

5. Non-Functional Requirements

NFR-001: Performance

- Dashboard loads in <3 seconds
- API response time <2 seconds
- Supports 100 concurrent users
- 99% uptime for data collection

NFR-002: Scalability

- Modular architecture for easy expansion
- Configurable worker count (default: 5)
- Adjustable update intervals
- Cache system for reduced API calls

NFR-003: Cost Efficiency

- Uses only free-tier APIs
- No paid data source dependencies
- Local LLM processing (Ollama)
- Open-source stack

NFR-004: Privacy & Security

- No user data storage required
- API key management via environment variables
- CORS enabled for web access
- Input validation on all endpoints

NFR-005: Usability

- Simple command-line interface for agent system
- Intuitive web dashboard
- Clear error messages
- Comprehensive logging

6. Technical Specifications

6.1 Technology Stack

Backend Framework: Flask (Python)

AI Framework: LangGraph

LLM Integration: Ollama (llama3.2)

Web Scraping: BeautifulSoup4, feedparser

Financial Data: yfinance

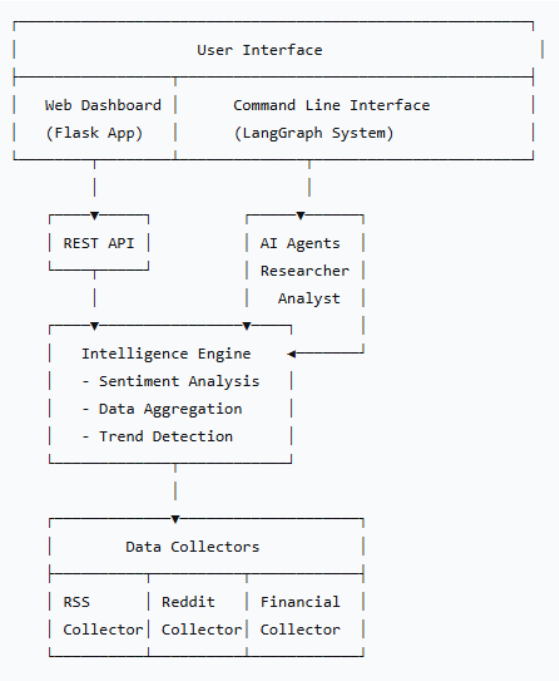
Async Processing: asyncio, aiohttp

Data Processing: pandas, numpy

6.2 Data Sources

- **News**: Yahoo Finance, CNBC, Reuters, CNN RSS feeds
- **Social**: Reddit (r/stocks, r/investing, etc.)
- **Financial**: yfinance (free), Alpha Vantage (optional)
- **Web Search**: DuckDuckGo (via search API)

6.3 System Architecture Diagram



7. User Flows

Flow 1: Continuous Monitoring (Passive User)

User opens dashboard → System displays current intelligence

→ User views sentiment metrics

→ User checks trending stocks

→ User reads top news

→ (Optional) User refreshes data

Flow 2: Deep Analysis (Active User)

User runs agent system → Enters competitor name

→ Researcher agent searches web

→ Analyst agent processes data

→ System generates report

→ User receives strategic analysis

8. Success Metrics

Quantitative Metrics

- **Data Freshness**: <5 minutes for market data
- **Sentiment Accuracy**: >85% human-aligned classification
- **System Uptime**: >99% availability
- **Processing Speed**: <30 seconds for agent analysis
- **API Latency**: <2 second response time

Qualitative Metrics

- User satisfaction with insights quality
- Ease of setup and use
- Actionability of generated reports
- Dashboard clarity and usability

9. Future Roadmap

Phase 2 (Next 3 Months)

- Additional data sources (Twitter/X, SEC filings)
- Advanced sentiment models (BERT-based)
- Alert system for significant events
- Historical data analysis
- Export functionality (PDF/CSV)

Phase 3 (Next 6 Months)

- Predictive analytics
- Custom watchlists
- Multi-company comparison
- API key for external integration
- Mobile responsive design

Phase 4 (Next 12 Months)

- Industry-specific modules
- Advanced visualization (charts, graphs)
- Team collaboration features
- Scheduled report generation
- Plugin architecture

10. Constraints & Assumptions

Constraints

- Must use free APIs where possible
- Local deployment focus (not cloud-native)
- Limited to English language content
- No real-time stock trading recommendations
- Educational/research purposes only

Assumptions

- Users have basic Python knowledge for setup

- Local LLM (Ollama) is installed and configured
- Stable internet connection available
- Free API limits won't be exceeded
- Users understand financial market risks

11. Risks & Mitigations

Risk	Impact	Probability	Mitigation
API rate limiting	High	Medium	Implement caching, rotate sources
Data quality issues	Medium	High	Data validation, multiple sources
LLM hallucination	Medium	Medium	Fact-checking prompts, source citation
System complexity	Low	High	Clear documentation, simple setup
Legal compliance	High	Low	Educational disclaimer, no financial advice

12. Open Questions

1. Should we add user authentication?
2. How to handle non-US stock markets?
3. What's the optimal cache duration?
4. How to scale for enterprise use?
5. Should we add database persistence?

Document Version: 1.0
Status: Draft for Review
Stakeholders: Product Team, Engineering, Design, QA