Product Requirements Document

Agentic AI Stock Analysis App (MVP)

Version: 1.0
Date: December 10, 2025
Status: Draft

---

1. Product Overview

1.1 Purpose

An agentic AI-powered application that provides retail traders with instant technical analysis of Indian stocks by fetching real-time price data via Kite Connect API and generating easy-to-understand explanations of stock technicals.

1.2 Target Users

- Retail traders in India (beginner to intermediate level)
- Individuals looking for quick technical insights on NSE/BSE stocks
- Traders who want AI-powered analysis without manual chart reading

---

2. Core User Flow

1. User enters a stock symbol or name (e.g., "RELIANCE" or "RELIANCE.NS")
2. Agent validates the input and fetches current price data from Kite Connect API
3. Agent retrieves historical price data for candlestick chart generation
4. Agent displays:
    - Current price and basic stock info
    - Candlestick chart for the stock
    - AI-generated technical analysis explanation
5. User can enter another stock symbol to repeat the process

---

3. Functional Requirements

3.1 Stock Data Fetching

- **FR-001:** System shall accept stock symbol input from user (NSE/BSE format)
- **FR-002:** System shall fetch current price using Kite Connect API
- **FR-003:** System shall retrieve historical OHLC (Open, High, Low, Close) data for candlestick chart
- **FR-004:** System shall handle invalid stock symbols gracefully with error messages

3.2 Technical Analysis (Basic)

The AI agent shall analyze and explain:

- **FR-005:** Current price and price change (absolute and percentage)
- **FR-006:** Volume analysis (current volume vs average volume)
- **FR-007:** Support and resistance levels (recent key price levels)
- **FR-008:** Simple Moving Averages (SMA 20, SMA 50)
- **FR-009:** Price trend identification (uptrend, downtrend, sideways)

3.3 Visualization

- **FR-010:** System shall display candlestick chart showing recent price action (last 30-60 days)
- **FR-011:** Chart shall be interactive and clearly labeled with dates and prices
- **FR-012:** Chart shall include volume bars at the bottom

3.4 AI-Generated Insights

- **FR-013:** AI shall provide plain-language explanation of technical indicators
- **FR-014:** Explanation shall be structured and easy to understand for retail traders
- **FR-015:** AI shall avoid trading recommendations (disclaimer: for informational purposes only)

---

4. Non-Functional Requirements

4.1 Performance

- **NFR-001:** Stock price data shall be fetched within 3 seconds
- **NFR-002:** AI analysis shall be generated within 5 seconds of data retrieval

4.2 Usability

- **NFR-003:** Interface shall be clean and minimalist
- **NFR-004:** User shall be able to analyze a new stock without page refresh

4.3 Reliability

- **NFR-005:** System shall handle Kite Connect API rate limits gracefully
- **NFR-006:** System shall display appropriate error messages for API failures

---

5. Technical Constraints

5.1 API Integration

- Kite Connect API (user has existing API key)
- Indian markets only (NSE/BSE)

5.2 Market Coverage

- NSE (National Stock Exchange)
- BSE (Bombay Stock Exchange)

---

6. Out of Scope (MVP)

The following features are explicitly **not included** in the MVP:

- Advanced technical indicators (RSI, MACD, Bollinger Bands, Fibonacci)
- Multiple stock comparison
- Portfolio tracking
- Historical backtesting
- Real-time streaming data
- Trading execution capabilities
- Multi-timeframe analysis (intraday, weekly, monthly)
- Custom indicator configuration
- Alerts and notifications
- User authentication and saved preferences

---

7. Success Metrics

- User successfully retrieves stock price and analysis within 8 seconds
- AI explanations are clear and understandable (to be validated through user testing)
- System handles 95% of valid NSE/BSE stock symbols correctly

---

8. Future Enhancements (Post-MVP)

1. Add intermediate technical indicators (RSI, MACD)
2. Multi-stock comparison view
3. Historical trend analysis across different timeframes
4. Watchlist functionality
5. More advanced candlestick pattern recognition

---

9. Assumptions & Dependencies

Assumptions

- User has valid Kite Connect API credentials
- User has basic understanding of stock market terminology
- Market data is available during trading hours (9:15 AM - 3:30 PM IST)

Dependencies

- Kite Connect API availability and uptime
- AI model availability for generating technical analysis
- Chart rendering library for candlestick visualization

---

10. User Interface Requirements

10.1 Input Section

- Text input field for stock symbol

- Submit button to trigger analysis
- Clear placeholder text (e.g., "Enter stock symbol: RELIANCE, TCS, INFY")

10.2 Output Section

- Stock header: Name, symbol, current price, change %
- Candlestick chart (prominent, center of screen)
- Technical analysis text section below chart
- Option to analyze another stock

---

11. Technical Architecture

11.1 System Architecture Overview

The application follows a **client-server architecture** with Python backend for LangGraph orchestration:

Architecture Benefits:

- **Security:** API keys stored server-side
- **Stability:** Python LangChain 1.1.3 (stable release)
- **Observability:** Better LangSmith integration
- **Scalability:** Easier to add caching, rate limiting
- **Flexibility:** Can switch LLM providers without frontend changes

11.2 Component Breakdown

Frontend Layer (React)

- **Technology:** React 18+ with Vite
- **Charting Library:** Lightweight Charts by TradingView
- **UI Framework:** Tailwind CSS for styling
- **State Management:** React hooks (useState, useEffect)
- **HTTP Client:** Axios for API calls
- **Responsibilities:**
    - User input collection
    - Display stock charts and data
    - Show AI-generated analysis
    - Handle loading and error states

Backend Layer (Python FastAPI)

- **Framework:** FastAPI (modern, fast, async)
- **Python Version:** 3.10+
- **Web Server:** Uvicorn (ASGI server)
- **Responsibilities:**
    - Expose REST API endpoints
    - Authenticate requests (optional for MVP)
    - Orchestrate LangGraph agent
    - Handle errors and rate limiting
    - Return structured responses

Agentic AI Layer (LangGraph Python 1.1.3)

- **Framework:** LangGraph 0.2.x with LangChain Python 1.1.3
- **Purpose:** Coordinate workflow between stock data fetching and analysis
- **Agent Workflow:**
    1. Receive stock symbol from API
    2. Execute Stock Data Tool (Kite Connect)
    3. Process and calculate technical indicators

4. Execute LLM Analysis Tool
5. Return structured response to FastAPI

LLM Provider Layer (Model Abstraction)

- **Primary Model:** OpenAI GPT-4o-mini (cost-effective)
- **Abstraction:** LangChain's model interface for easy switching
- **Fallback/Alternative:** Claude Sonnet 4 (configurable)
- **Purpose:** Generate natural language technical analysis from stock data
- **Input:** OHLC data, volume, moving averages, support/resistance levels
- **Output:** Structured technical analysis explanation

Tool Layer

Stock Data Tool:

- Wraps Kite Connect API as a LangChain tool
- Functions: `fetch_quote()`, `fetch_historical_data()`
- Returns structured data for agent processing

Technical Calculation Tool:

- Calculate SMA, support/resistance, volume averages
- Returns processed indicators

External APIs

- **Kite Connect API:**
  - Endpoints: Quote API for current price, Historical API for OHLC data
  - Authentication: API key and access token
  - Rate limiting: Handle 3 requests/second limit
- **OpenAI API:**
  - Model: gpt-4o-mini (primary), gpt-4o (if higher quality needed)
  - Context: Stock data + technical indicators
  - Response: Technical analysis text
- **Claude API (Optional/Future):**
  - Model: claude-sonnet-4-20250514

    - Same interface through LangChain abstraction
- **LangSmith (Tracing):**
    - Track all LangChain/LangGraph calls
    - Monitor agent performance
    - Debug issues in production

11.3 Data Flow

1. **User Input → React Frontend**
    - User enters stock symbol
    - Frontend validates format
    - Sends POST request to backend: `/api/analyze`
2. **Frontend → FastAPI Backend**
    - Receives stock symbol
    - Validates request
    - Triggers LangGraph agent
3. **FastAPI → LangGraph Agent Initialization**
    - Agent receives stock symbol
    - Plans execution: fetch data → calculate indicators → generate analysis
4. **Agent → Stock Data Tool → Kite Connect API**
    - Tool fetches current quote (price, volume, change)
    - Tool fetches historical data (last 60 days OHLC)
    - Returns structured data to agent
5. **Agent → Technical Calculation Tool**
    - Calculate SMA 20 and SMA 50
    - Identify support/resistance levels
    - Calculate volume averages
    - Returns processed indicators to agent
6. **Agent → LLM Provider (OpenAI)**
    - Agent constructs prompt with structured data
    - Sends to GPT-4o-mini via LangChain abstraction
    - Receives technical analysis generation
    - (Optional) Logs to LangSmith for tracing

7. **Agent → FastAPI → Frontend**
    - Agent returns complete response to FastAPI
    - FastAPI formats response as JSON:
        - Stock info (symbol, price, change)
        - Historical OHLC data (for chart)
        - Calculated indicators
        - AI-generated analysis text
    - Frontend receives response
8. **Frontend Rendering**
    - Display price information
    - Render candlestick chart
    - Show AI-generated analysis
    - Handle loading/error states

11.4 Key Technical Decisions

Why React + Python Backend Architecture?

- **Security:** API keys stored server-side, not exposed in browser
- **Stability:** Python LangChain 1.1.3 is mature and stable
- **Better Ecosystem:** Python has richer LangChain tooling and community
- **Professional Pattern:** Industry-standard client-server architecture
- **Easier Scaling:** Can add caching, database, authentication later

Why FastAPI for Backend?

- **Modern & Fast:** Async support, high performance
- **Easy to Use:** Simple, intuitive API design
- **Auto Documentation:** Built-in Swagger/OpenAPI docs
- **Type Safety:** Pydantic models for validation
- **Production Ready:** Used by major companies (Uber, Netflix)

Why LangGraph with Python 1.1.3?

- **Structured Workflow:** Clear state machine for agent execution
- **Tool Integration:** Built-in support for custom tools (Kite Connect)

- **Error Handling:** Robust retry logic and fallback mechanisms
- **Observability:** Easy to debug and trace agent decisions
- **Mature Ecosystem:** Python 1.1.3 is stable, well-tested
- **LangSmith Integration:** Seamless tracing and monitoring

Why LangChain Model Abstraction?

- **Vendor Independence:** Switch between OpenAI, Claude, or other providers with minimal code changes
- **Consistent Interface:** Same API regardless of underlying model
- **Easy Testing:** Can swap in different models for A/B testing
- **Cost Optimization:** Start with cheaper models, upgrade if needed

Why GPT-4o-mini as Primary Model?

- **Cost-Effective:** ~~94% cheaper than GPT-4o (~~$0.15/$0.60 per million tokens vs $2.50/$10)
- **Sufficient Quality:** Good enough for technical analysis explanations
- **Fast Response:** Lower latency than larger models
- **Scalable:** Can handle more requests within budget
- **Easy Upgrade Path:** Switch to GPT-4o or Claude if quality issues arise

Chart Library Selection

Option 1: Lightweight Charts (TradingView) ✅ Recommended

- Pros: Professional-grade, performant, feature-rich
- Cons: Slightly larger bundle size

Option 2: Recharts

- Pros: React-native, simpler API, smaller bundle
- Cons: Less specialized for financial charts

11.5 Technical Stack Summary

Layer	Technology	Purpose
Frontend	React 18 + Vite	User interface
Frontend Styling	Tailwind CSS	UI components and styling

Layer	Technology	Purpose
Charting	Lightweight Charts	Candlestick visualization
HTTP Client	Axios	API communication
Backend Framework	FastAPI	REST API server
Backend Language	Python 3.10+	Server-side logic
Web Server	Uvicorn	ASGI server
Agent Framework	LangGraph 0.2.x	Agentic workflow orchestration
LLM Core	LangChain Python 1.1.3	LLM abstraction and tooling
Primary LLM	OpenAI GPT-4o-mini	Technical analysis generation
Alternative LLM	Claude Sonnet 4	Fallback/upgrade option
Stock Data API	Kite Connect	Real-time & historical data
Tracing/Monitoring	LangSmith	Agent observability
Frontend Deployment	Vercel/Netlify	Static site hosting
Backend Deployment	Railway/Render/AWS	Python backend hosting

11.6 API Endpoints

Backend API Specification

Base URL: `http://localhost:8000` (development) or `https://api.yourapp.com` (production)

Endpoint 1: Analyze Stock

```
POST /api/analyze
Content-Type: application/json

Request Body:
{
  "symbol": "RELIANCE",
  "exchange": "NSE"  // Optional, defaults to NSE
}

Response (200 OK):
{
  "status": "success",
  "data": {
    "stock_info": {
      "symbol": "RELIANCE",
      "name": "Reliance Industries Ltd.",
      "current_price": 2456.75,
      "change": 29.30,
      "change_percent": 1.21,
      "volume": 5234567,
      "last_updated": "2025-12-10T15:30:00Z"
    },
    "historical_data": [
      {
        "date": "2025-11-10",
        "open": 2430.00,
        "high": 2465.50,
        "low": 2420.00,
        "close": 2456.75,
        "volume": 5234567
      },
      // ... more historical data (60 days)
    ],
    "technical_indicators": {
      "sma_20": 2420.50,
      "sma_50": 2380.20,
      "support_levels": [2400.00, 2350.00],
      "resistance_levels": [2500.00, 2550.00],
      "avg_volume_10d": 4500000,
      "trend": "uptrend"
    },
    "analysis": {
      "summary": "The stock is currently trading above its 20-day moving average...",
      "generated_at": "2025-12-10T15:30:05Z",
      "model_used": "gpt-4o-mini"
    }
```

```
  }
}

Error Response (400/500):
{
  "status": "error",
  "message": "Invalid stock symbol",
  "error_code": "INVALID_SYMBOL"
}
```

Endpoint 2: Health Check

```
GET /api/health

Response (200 OK):
{
  "status": "healthy",
  "timestamp": "2025-12-10T15:30:00Z",
  "version": "1.0.0"
}
```

11.7 LLM Provider Configuration

Model Switching Strategy

The application uses LangChain's model abstraction to allow seamless switching:

```
python
```

```python
# Python backend configuration
from langchain_openai import ChatOpenAI
from langchain_anthropic import ChatAnthropic
import os

# Configuration-based model selection
MODEL_CONFIG = {
    "provider": os.getenv("LLM_PROVIDER", "openai"),  # or "claude"
    "openai": {
        "model": "gpt-4o-mini",
        "temperature": 0.3,
        "api_key": os.getenv("OPENAI_API_KEY")
    },
    "claude": {
        "model": "claude-sonnet-4-20250514",
        "temperature": 0.3,
        "api_key": os.getenv("ANTHROPIC_API_KEY")
    }
}

# Initialize model based on config
def get_llm():
    provider = MODEL_CONFIG["provider"]
    if provider == "openai":
        return ChatOpenAI(**MODEL_CONFIG["openai"])
    elif provider == "claude":
        return ChatAnthropic(**MODEL_CONFIG["claude"])
    else:
        raise ValueError(f"Unknown provider: {provider}")
```

Cost Comparison (Estimated per 1000 analyses)

Model	Input Cost	Output Cost	Total Est. Cost
GPT-4o-mini	$0.15	$0.60	~$1.50
GPT-4o	$2.50	$10.00	~$25.00
Claude Sonnet 4	$3.00	$15.00	~$30.00

Assuming ~1000 input tokens and ~1000 output tokens per analysis

When to Switch Models

- **Start with:** GPT-4o-mini (cost-effective MVP)

- **Upgrade to GPT-4o if:** Analysis quality insufficient
- **Switch to Claude if:** Need better reasoning for complex patterns
- **A/B Test:** Run both models on sample stocks to compare quality

11.8 Security Considerations

- **API Key Storage:** All API keys (OpenAI, Kite Connect, LangSmith) stored server-side in environment variables
- **Frontend Security:** No sensitive credentials exposed in React app
- **CORS Configuration:** Properly configured CORS headers in FastAPI
- **Rate Limiting:** Implement rate limiting on backend endpoints to prevent abuse
- **Input Validation:** Validate stock symbols and sanitize inputs
- **Error Handling:** Never expose API keys or sensitive data in error messages
- **HTTPS Only:** Enforce HTTPS in production
- **Authentication (Future):** Add user authentication for production use

11.9 Scalability Considerations (Future)

The current architecture is designed to scale. Future enhancements:

- **Caching Layer:** Redis for frequently requested stocks (reduce API calls)
- **Database:** PostgreSQL for storing historical analysis and user preferences
- **Background Jobs:** Celery for async processing of multiple stocks
- **WebSocket Integration:** Real-time data streaming for live updates
- **Load Balancing:** Multiple backend instances behind load balancer
- **CDN:** CloudFront/Cloudflare for static assets
- **Monitoring:** Prometheus + Grafana for metrics
- **Agent Execution History:** Store LangGraph traces in database

11.10 Development Environment

Frontend Setup

- **Package Manager:** npm or yarn
- **Build Tool:** Vite (fast HMR and builds)
- **Node Version:** 18+ LTS
- **Dependencies:**

- `react`, `react-dom`
- `axios` - HTTP client
- `lightweight-charts` - Charting
- `tailwindcss` - Styling

Backend Setup

- **Python Version:** 3.10+
- **Package Manager:** pip or poetry
- **Virtual Environment:** venv or conda
- **Dependencies:**
  - `fastapi` - Web framework
  - `uvicorn` - ASGI server
  - `langchain==1.1.3` - LLM orchestration
  - `langgraph` - Agent graphs
  - `langchain-openai` - OpenAI integration
  - `langchain-anthropic` - Claude integration (optional)
  - `langsmith` - Tracing
  - `kiteconnect` - Stock data API
  - `pydantic` - Data validation
  - `python-dotenv` - Environment variables

Development Tools

- **Version Control:** Git
- **Code Quality:**
  - Frontend: ESLint, Prettier
  - Backend: Black, Flake8, mypy
- **Testing:**
  - Frontend: Vitest + React Testing Library
  - Backend: pytest
- **API Documentation:** FastAPI auto-generates Swagger docs

11.11 Deployment Strategy

Frontend Deployment

- **Platform:** Vercel or Netlify (recommended)
- **Build Command:** `npm run build`
- **Output Directory:** `dist/`
- **Environment Variables:**
  - `VITE_API_URL` - Backend API URL
- **Cost:** Free tier sufficient for MVP

Backend Deployment

- **Platform Options:**
  - **Railway** (recommended for MVP) - Easy Python deployments
  - **Render** - Free tier available
  - **AWS EC2/ECS** - More control, higher complexity
  - **Google Cloud Run** - Serverless containers
- **Deployment Method:**
  - Docker container (recommended)
  - Or direct Python deployment
- **Environment Variables:**
  - `OPENAI_API_KEY`
  - `KITE_API_KEY`
  - `KITE_API_SECRET`
  - `KITE_ACCESS_TOKEN`
  - `LANGCHAIN_TRACING_V2=true`
  - `LANGCHAIN_API_KEY`
  - `LANGCHAIN_PROJECT=stock-analysis-agent`
  - `LLM_PROVIDER=openai`
- **Cost:** ~$5-10/month for basic tier

CI/CD Pipeline

- **GitHub Actions:** Automated testing and deployment

- **Workflow:**
    1. Push to `main` branch
    2. Run tests (frontend + backend)
    3. Build Docker image (backend)
    4. Deploy to Railway/Render
    5. Deploy frontend to Vercel

---

12. Project Folder Structure

12.1 Overview

The project follows a **monorepo structure** with separate frontend (React) and backend (Python) applications:

```
stock-analysis-agent/
├── frontend/                # React application
├── backend/                 # Python FastAPI application
├── .gitignore
├── README.md
├── docker-compose.yml       # Local development setup
└── .github/
    └── workflows/
        └── ci-cd.yml        # GitHub Actions CI/CD
```

12.2 Frontend Structure (`frontend/`)

```
frontend/
├── src/
│   ├── components/          # React components
│   │   ├── ui/              # Reusable UI components
│   │   │   ├── Input.jsx    # Input field component
│   │   │   ├── Button.jsx   # Button component
│   │   │   ├── Card.jsx     # Card container
│   │   │   └── Loader.jsx   # Loading spinner
│   │   ├── StockInput.jsx   # Stock symbol input with validation
│   │   ├── StockChart.jsx   # Candlestick chart display
│   │   ├── StockInfo.jsx    # Price and stock information
│   │   ├── Analysis.jsx     # AI analysis text display
│   │   └── ErrorBoundary.jsx # Error handling component
│   │
│   ├── services/            # API service layer
│   │   └── api.js           # Axios instance + API call functions
│   │
```

```
│   ├── hooks/                # Custom React hooks
│   │   └── useStockAnalysis.js # Hook for stock analysis logic
│   │
│   ├── utils/                # Utility functions
│   │   ├── formatters.js     # Data formatting (currency, dates)
│   │   └── validators.js     # Input validation
│   │
│   ├── App.jsx               # Main application component
│   ├── main.jsx              # React entry point
│   └── index.css             # Global styles (Tailwind imports)
│
├── public/
│   └── favicon.ico
│
├── tests/                    # Frontend tests
│   └── components/
│       └── StockInput.test.jsx
│
├── .env.example              # Example environment variables
├── .env                      # Actual environment variables (gitignored)
├── package.json              # Dependencies and scripts
├── vite.config.js            # Vite configuration
├── vitest.config.js          # Test configuration
├── tailwind.config.js        # Tailwind CSS configuration
└── eslint.config.js          # ESLint rules
```

Key Frontend Files Explained:

- **components/ui/**: Generic, reusable UI components that follow design system
- **services/api.js**: Centralized API calls to backend (POST /api/analyze)
- **hooks/useStockAnalysis.js**: Encapsulates analysis logic (loading, error states)
- **utils/formatters.js**: Format prices (₹2,456.75), dates, percentages
- **utils/validators.js**: Validate stock symbols (NSE/BSE format)

12.3 Backend Structure (**backend/**)

```
backend/
├── app/
│   ├── __init__.py
│   ├── main.py               # FastAPI app entry point and configuration
│   │
│   ├── api/                  # API routes and endpoints
│   │   ├── __init__.py
│   │   └── routes.py         # Defines /api/analyze, /api/health
```

```
│   │
│   ├── agent/              # LangGraph agent logic
│   │   ├── __init__.py
│   │   ├── graph.py        # StateGraph definition and compilation
│   │   ├── state.py        # Agent state schema (MessagesAnnotation)
│   │   └── nodes.py        # Individual node functions (fetch, calculate, analyze)
│   │
│   ├── tools/              # LangChain tools
│   │   ├── __init__.py
│   │   ├── stock_data_tool.py # Tool wrapper for Kite Connect API
│   │   └── technical_tool.py  # Tool for technical calculations (SMA, S/R)
│   │
│   ├── services/           # External service integrations
│   │   ├── __init__.py
│   │   ├── kite_service.py    # Kite Connect API wrapper
│   │   └── llm_service.py     # LLM provider abstraction (OpenAI/Claude)
│   │
│   ├── utils/              # Utility functions
│   │   ├── __init__.py
│   │   ├── calculations.py    # SMA, support/resistance calculations
│   │   └── validators.py      # Input validation and sanitization
│   │
│   ├── config/             # Application configuration
│   │   ├── __init__.py
│   │   └── settings.py        # Pydantic Settings (environment variables)
│   │
│   └── models/             # Pydantic models for API
│       ├── __init__.py
│       ├── request.py         # Request schemas (AnalyzeStockRequest)
│       └── response.py        # Response schemas (AnalyzeStockResponse)
│
├── tests/                  # Backend tests (pytest)
│   ├── __init__.py
│   ├── test_api.py         # API endpoint tests
│   ├── test_agent.py       # LangGraph agent tests
│   └── test_tools.py       # Tool functionality tests
│
├── .env.example            # Example environment variables
├── .env                    # Actual environment variables (gitignored)
├── requirements.txt        # Python dependencies (pip)
├── pyproject.toml          # Poetry configuration (optional)
└── Dockerfile              # Docker container definition
```

Key Backend Files Explained:

- **main.py**: FastAPI app initialization, CORS setup, router inclusion

- **api/routes.py**: REST API endpoints that call the LangGraph agent
- **agent/graph.py**: LangGraph StateGraph with nodes and edges defined
- **agent/nodes.py**: Pure functions for each agent step (fetch → calculate → analyze)
- **tools/stock_data_tool.py**: LangChain Tool that wraps Kite Connect API
- **tools/technical_tool.py**: LangChain Tool for technical indicator calculations
- **services/kite_service.py**: Low-level Kite Connect API interactions
- **services/llm_service.py**: Model selection logic (OpenAI vs Claude)
- **config/settings.py**: Pydantic Settings for environment variable management
- **models/request.py**: Input validation schemas (symbol, exchange)
- **models/response.py**: Structured response formats

12.4 Root Level Files

.gitignore

```
# Python
__pycache__/
*.py[cod]
*$py.class
.env
venv/
.pytest_cache/

# Node
node_modules/
dist/
.env
.env.local

# IDE
.vscode/
.idea/
*.swp
```

docker-compose.yml

```yaml
```

```yaml
version: '3.8'

services:
  backend:
    build: ./backend
    ports:
      - "8000:8000"
    env_file:
      - ./backend/.env
    volumes:
      - ./backend:/app

  frontend:
    build: ./frontend
    ports:
      - "5173:5173"
    environment:
      - VITE_API_URL=http://localhost:8000
    volumes:
      - ./frontend:/app
    depends_on:
      - backend
```

README.md (Root level project documentation)

- Project overview
- Setup instructions for frontend and backend
- Environment variable configuration
- Running locally with Docker
- Deployment guide
- API documentation link

12.5 Environment Files

frontend/.env.example

```bash
VITE_API_URL=http://localhost:8000
```

backend/.env.example

```bash
```

```
# OpenAI
OPENAI_API_KEY=your_openai_api_key_here

# Kite Connect
KITE_API_KEY=your_kite_api_key_here
KITE_API_SECRET=your_kite_api_secret_here
KITE_ACCESS_TOKEN=your_kite_access_token_here

# LangSmith
LANGCHAIN_TRACING_V2=true
LANGCHAIN_API_KEY=your_langsmith_api_key_here
LANGCHAIN_PROJECT=stock-analysis-agent

# LLM Provider
LLM_PROVIDER=openai
```

12.6 Design Principles

1. **Separation of Concerns**: Frontend and backend are completely separate
2. **Modularity**: Each folder has a specific responsibility
3. **Testability**: Tests live alongside code they test
4. **Scalability**: Easy to add new tools, services, or components
5. **Configuration**: All secrets in environment variables
6. **Type Safety**: Pydantic models for backend, PropTypes/Type