🚀 Financial Market Intelligence Agent - Complete Setup Guide

📋 Table of Contents

Prerequisites

Required Software

- Python 3.10 or higher
- Docker & Docker Compose
- Git
- AWS CLI (for deployment)
- Node.js 16+ (optional, for additional tools)

Required API Keys

1. **Google Gemini API Key** (Free tier available)
   - Visit: https://makersuite.google.com/app/apikey
   - Create a new API key
   - Free quota: 60 requests per minute
2. **Alpha Vantage API Key** (Free)
   - Visit: https://www.alphavantage.co/support/#api-key
   - Sign up for free API key

- Free quota: 5 requests per minute, 500 per day

3. **News API Key** (Optional but recommended)
   - Visit: https://newsapi.org/register
   - Free tier: 100 requests per day

Project Setup

1. Clone and Structure

```bash
# Create project directory
mkdir financial-intelligence-agent
cd financial-intelligence-agent

# Create directory structure
mkdir -p backend/app/{agents,data,rag,analysis,api}
mkdir -p frontend/components
mkdir -p deployment/aws
mkdir -p data/vector_db
mkdir -p tests

# Initialize git
git init
```

2. Create Configuration Files

Create `.env` file in the root directory:

```bash
# Copy from .env.example
cp deployment/.env.example .env

# Edit .env with your actual API keys
nano .env
```

Fill in your API keys:

```env
```

```
GOOGLE_API_KEY=AIzaSy...your_key_here
ALPHA_VANTAGE_API_KEY=your_alpha_vantage_key
NEWS_API_KEY=your_news_api_key  # Optional
```

3. Install Dependencies

Backend:

```bash
cd backend
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install -r requirements.txt
```

Frontend:

```bash
cd ../frontend
pip install -r requirements.txt
```

Local Development

Option 1: Run with Docker Compose (Recommended)

```bash
# From project root
docker-compose up --build

# Access:
# Frontend: http://localhost:8501
# Backend API: http://localhost:8000
# API Docs: http://localhost:8000/docs
```

Option 2: Run Manually

Terminal 1 - Backend:

```bash
```

```
cd backend
source venv/bin/activate
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

Terminal 2 - Frontend:

```
bash

cd frontend
streamlit run app.py --server.port 8501
```

4. Initialize RAG Database

```
bash

# Make API call to ingest initial documents
curl -X POST http://localhost:8000/api/ingest/documents
```

AWS Deployment

Step 1: Configure AWS CLI

```
bash

aws configure
# Enter your AWS Access Key ID
# Enter your AWS Secret Access Key
# Default region: us-east-1
# Default output format: json
```

Step 2: Set Environment Variables

Update `.env` with AWS configuration:

```
env

AWS_REGION=us-east-1
AWS_ACCOUNT_ID=123456789012
ECR_REGISTRY=123456789012.dkr.ecr.us-east-1.amazonaws.com
```

Step 3: Deploy to AWS

```
bash
```

```
# Make deploy script executable
chmod +x deployment/aws/deploy.sh

# Full deployment (recommended for first time)
./deployment/aws/deploy.sh deploy

# This will:
# 1. Check prerequisites
# 2. Create ECR repositories
# 3. Build Docker images
# 4. Push to ECR
# 5. Deploy CloudFormation stack
# 6. Output the application URL
```

Step 4: Verify Deployment

bash

```
# Check stack status
aws cloudformation describe-stacks \
  --stack-name financial-agent-stack \
  --region us-east-1

# Get application URL
aws cloudformation describe-stacks \
  --stack-name financial-agent-stack \
  --query 'Stacks[0].Outputs[?OutputKey==`LoadBalancerDNS`].OutputValue' \
  --output text
```

Updating Deployment

bash

```
# Update services with new images
./deployment/aws/deploy.sh update
```

Viewing Logs

bash

```
# View backend logs
./deployment/aws/deploy.sh logs

# Or use AWS Console:
# CloudWatch > Log groups > /ecs/production/backend
```

Cost Considerations

Estimated Monthly Costs:

- ECS Fargate (2 backend, 1 frontend): ~$50-70
- Application Load Balancer: ~$20
- CloudWatch Logs: ~$5
- Data Transfer: ~$10
- **Total: ~$85-105/month**

Cost Optimization:

- Use Fargate Spot for non-production
- Reduce task counts during off-hours
- Set up auto-scaling based on demand

Usage Examples

1. Using the Chat Interface

Frontend (Streamlit):

```
Navigate to http://localhost:8501
1. Click "Chat Agent" in sidebar
2. Enter: "Analyze Tesla's performance and compare with Ford"
3. View agent reasoning steps and recommendations
```

API (curl):

```
bash
```

```
curl -X POST http://localhost:8000/api/chat \
  -H "Content-Type: application/json" \
  -d '{
    "message": "Analyze AAPL stock",
    "session_id": "user123",
    "include_sources": true
  }'
```

2. Stock Analysis

Full Analysis:

bash

```
curl -X POST http://localhost:8000/api/analyze \
  -H "Content-Type: application/json" \
  -d '{
    "symbol": "AAPL",
    "analysis_type": "full",
    "timeframe": "3mo"
  }'
```

Technical Analysis Only:

bash

```
curl -X POST http://localhost:8000/api/analyze \
  -H "Content-Type: application/json" \
  -d '{
    "symbol": "TSLA",
    "analysis_type": "technical",
    "timeframe": "1mo"
  }'
```

3. Price Predictions

bash

```
curl -X POST http://localhost:8000/api/predict/NVDA?days=7
```

4. Stock Comparison

bash

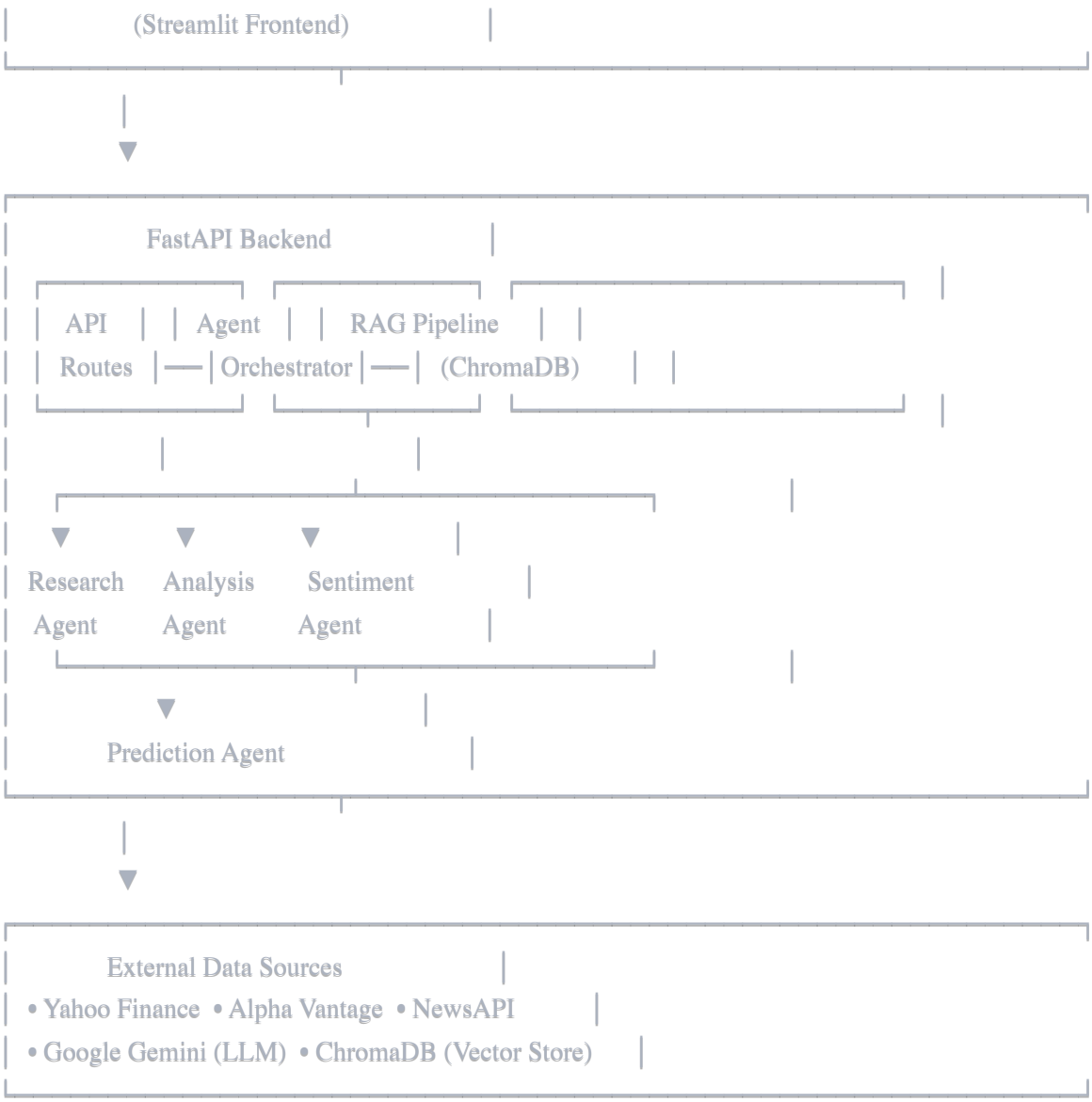
```
curl http://localhost:8000/api/compare?symbols=AAPL,MSFT,GOOGL
```

5. Python Integration

```python
import requests

# Initialize client
BASE_URL = "http://localhost:8000"

# Analyze stock
response = requests.post(
    f"{BASE_URL}/api/analyze",
    json={
        "symbol": "AAPL",
        "analysis_type": "full",
        "timeframe": "3mo"
    }
)

analysis = response.json()
print(f"Risk Level: {analysis['risk_level']}")
print(f"Recommendations: {analysis['recommendations']}")

# Chat with agent
response = requests.post(
    f"{BASE_URL}/api/chat",
    json={
        "message": "What's the sentiment around NVIDIA?",
        "session_id": "my-session"
    }
)

chat = response.json()
print(f"Agent Response: {chat['response']}")
```

Architecture Overview

```
┌─────────────────────────────────────────────────────────────────────┐
│                          User Interface                             │
```



Troubleshooting

Common Issues

1. API Key Errors

```
Error: GOOGLE_API_KEY not configured
Solution: Check .env file exists and contains valid API key
```

2. ChromaDB Connection Issues

```
Error: Cannot connect to ChromaDB
Solution: Ensure data/vector_db directory exists and has write permissions
chmod -R 755 data/vector_db
```

3. Docker Build Failures

```
Error: Docker image build failed
Solution:
- Clear Docker cache: docker system prune -a
- Rebuild: docker-compose build --no-cache
```

4. Port Already in Use

```
Error: Port 8000 already in use
Solution:
- Kill process: lsof -ti:8000 | xargs kill -9
- Or change port in docker-compose.yml
```

5. AWS Deployment Failures

```
Error: CloudFormation stack creation failed
Solution:
- Check AWS credentials: aws sts get-caller-identity
- Verify IAM permissions for CloudFormation, ECS, ECR
- Check CloudFormation events for specific error
```

Performance Optimization

1. Reduce API Latency

```python
# In app/config.py
CACHE_TTL = 300  # Cache responses for 5 minutes
CONCURRENT_REQUESTS = 5  # Limit concurrent API calls
```

2. Optimize ChromaDB

```python
# Use batch operations
documents = [...]
await rag_pipeline.add_documents(documents)  # Batch insert
```

3. Rate Limiting

```python
# Add to FastAPI
from slowapi import Limiter, _rate_limit_exceeded_handler
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)
app.state.limiter = limiter
app.add_exception_handler(RateLimitExceeded, _rate_limit_exceeded_handler)

@app.post("/api/chat")
@limiter.limit("10/minute")
async def chat(request: Request, data: ChatRequest):
    ...
```

Next Steps

Enhancements

1. **Add authentication** (JWT, OAuth)
2. **Implement caching** (Redis)
3. **Add real-time updates** (WebSockets)
4. **Integrate more data sources**
5. **Add portfolio tracking**
6. **Implement backtesting**

Production Readiness

1. **Set up monitoring** (CloudWatch, Datadog)
2. **Add error tracking** (Sentry)
3. **Implement CI/CD** (GitHub Actions)
4. **Add automated testing**
5. **Set up alerts**

Advanced Features

1. **Multi-language support**
2. **Custom ML models**

3. **Advanced charting**
4. **Social media sentiment**
5. **Earnings call transcripts**

Resources

- **Google Gemini Docs**: https://ai.google.dev/docs
- **LangChain Docs**: https://python.langchain.com/docs/
- **FastAPI Docs**: https://fastapi.tiangolo.com/
- **Streamlit Docs**: https://docs.streamlit.io/
- **AWS ECS Docs**: https://docs.aws.amazon.com/ecs/

Support

For issues or questions:

1. Check the troubleshooting section
2. Review API documentation at `/docs`
3. Check application logs
4. Open an issue on GitHub

License

MIT License - Feel free to use this for your portfolio!