Finance Trading Assistant API

A sophisticated LangServe API that provides AI-powered trading strategy consultation and automated PineScript code generation for TradingView. Built with FastAPI and powered by Azure OpenAI.

🚀 Features

- **AI Trading Consultant**: Get expert advice on trading strategies with comprehensive explanations
- **PineScript Code Generation**: Automatically generate PineScript v5 code for your trading strategies
- **Conversation Context**: Maintains conversation summaries for contextual responses
- **RESTful API**: Easy-to-use endpoints with automatic documentation
- **Risk Management**: Built-in risk management considerations in all strategies
- **Educational Focus**: Detailed explanations of market conditions, indicators, and best practices

📋 Prerequisites

- Python 3.8+
- Azure OpenAI API access
- Virtual environment (recommended)

🛠️ Installation

1. **Clone the repository**

   bash

   ```
   git clone <your-repo-url>
   cd finance/api_backend
   ```

2. **Create and activate virtual environment**

   bash

   ```
   python3 -m venv venv
   source venv/bin/activate  # On Windows: venv\Scripts\activate
   ```

3. **Install dependencies**

   bash

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

4. **Set up environment variables** Create a `.env` file in the project root:

env

```
# Azure OpenAI Configuration
AZURE_OPENAI_API_KEY=your-api-key-here
AZURE_OPENAI_ENDPOINT=https://your-resource.openai.azure.com/
AZURE_OPENAI_DEPLOYMENT=your-deployment-name
AZURE_OPENAI_API_VERSION=2024-02-15-preview
```

🚦 Running the Application

1. **Start the server**

   bash

   ```
   uvicorn main:app --reload
   ```

   Or with custom settings:

   bash

   ```
   uvicorn main:app --reload --host 0.0.0.0 --port 8000 --log-level debug
   ```

2. **Access the API**
   - API Documentation: http://localhost:8000/docs
   - Interactive Playground: http://localhost:8000/chat/playground
   - Health Check: http://localhost:8000/health

📡 API Endpoints

Main Endpoints

Endpoint	Method	Description
`/`	GET	API information and examples
`/health`	GET	Health check status
`/chat/invoke`	POST	Process trading strategy queries
`/chat/stream`	POST	Stream responses (for real-time output)
`/chat/batch`	POST	Process multiple queries
`/chat/playground`	GET	Interactive testing interface

Request/Response Format

Request Structure:

json

```
{
  "input": {
    "query": "Your trading strategy question",
    "previous_summary": "Optional: Summary of previous conversation"
  }
}
```

Response Structure:

json

```
{
  "output": {
    "answer": "Comprehensive strategy explanation and analysis",
    "code": "Generated PineScript code (if applicable)",
    "chatsummary": "Summary of current conversation"
  },
  "metadata": {
    "run_id": "unique-execution-id",
    "feedback_tokens": []
  }
}
```

📚 Usage Examples

1. Basic Strategy Request

bash

```
curl -X POST http://localhost:8000/chat/invoke \
  -H "Content-Type: application/json" \
  -d '{
    "input": {
      "query": "Create a simple moving average crossover strategy"
    }
  }'
```

2. Contextual Follow-up

bash

```
curl -X POST http://localhost:8000/chat/invoke \
  -H "Content-Type: application/json" \
  -d '{
    "input": {
      "query": "Add volume confirmation to this strategy",
      "previous_summary": "User created SMA crossover strategy with 20/50 periods"
    }
  }'
```

3. Complex Strategy Request

bash

```
curl -X POST http://localhost:8000/chat/invoke \
  -H "Content-Type: application/json" \
  -d '{
    "input": {
      "query": "Build a multi-timeframe strategy using RSI divergence and MACD confirma
    }
  }'
```

4. Pretty-printed Response (with jq)

bash

```
curl -X POST http://localhost:8000/chat/invoke \
  -H "Content-Type: application/json" \
  -d '{
    "input": {
      "query": "Create a Bollinger Bands squeeze strategy"
    }
  }' | jq .output
```

🏗️ Project Structure

```
api_backend/
├── main.py                    # FastAPI application and endpoints
├── config.py                  # Configuration management
├── requirements.txt           # Python dependencies
├── .env                       # Environment variables (create this)
└── llm_agent/
    ├── agent_multi.py         # Main agent logic
    ├── prompts_multi.py       # System prompts
    ├── tools_multi.py         # PineScript generation tool
    └── pinescript_knowledge.txt  # PineScript v5 knowledge base
```

🎯 Common Use Cases

1. Strategy Development

- Moving Average strategies (SMA, EMA, WMA)
- Momentum strategies (RSI, MACD, Stochastic)
- Volatility strategies (Bollinger Bands, ATR)
- Pattern recognition strategies
- Multi-indicator confirmation strategies

2. Risk Management

- Stop-loss and take-profit implementation
- Position sizing algorithms
- Drawdown management
- Risk-reward ratio optimization

3. Educational Queries

- "Explain how RSI divergence works"
- "What are the best market conditions for trend following?"
- "How do I implement trailing stops in PineScript?"

🔧 Configuration

Environment Variables

Variable	Description	Example
`AZURE_OPENAI_API_KEY`	Your Azure OpenAI API key	`sk-...`
`AZURE_OPENAI_ENDPOINT`	Azure OpenAI endpoint URL	`https://myresource.openai.azure.com/`
`AZURE_OPENAI_DEPLOYMENT`	Deployment name	`gpt-4`
`AZURE_OPENAI_API_VERSION`	API version	`2024-02-15-preview`

LLM Settings

- Temperature: 0 (for consistent outputs)
- Response format: JSON object
- Max iterations: 3 (for agent execution)

📝 Development Tips

1. **Testing Strategies**
   - Always test generated PineScript code in TradingView's Pine Editor
   - Start with simple strategies and gradually add complexity
   - Verify backtesting results before live trading
2. **API Usage**
   - Use the `/chat/playground` endpoint for interactive testing
   - Include `previous_summary` for multi-turn conversations
   - Check the response `code` field for generated PineScript
3. **Error Handling**
   - The API returns structured JSON even for errors
   - Check HTTP status codes for request validation
   - Review logs for detailed error information

🐛 Troubleshooting

Common Issues

1. **Connection Refused**
   - Ensure the server is running on the correct port
   - Check firewall settings
2. **Authentication Errors**
   - Verify Azure OpenAI credentials in `.env`
   - Ensure API key has proper permissions

3. **Invalid JSON Response**
   - Check request format matches the schema
   - Verify Content-Type header is set
4. **No PineScript Generated**
   - Ensure your query explicitly asks for code
   - Check if the strategy description is clear

Debug Mode

Run with debug logging:

bash

```
uvicorn main:app --reload --log-level debug
```

📈 Example Strategies

1. RSI Strategy

bash

```
curl -X POST http://localhost:8000/chat/invoke \
  -H "Content-Type: application/json" \
  -d '{"input": {"query": "Create an RSI strategy that buys oversold and sells overbou
```

2. MACD + Signal Line

bash

```
curl -X POST http://localhost:8000/chat/invoke \
  -H "Content-Type: application/json" \
  -d '{"input": {"query": "Build a MACD strategy with signal line crossovers and histo
```

3. Multi-Timeframe Analysis

bash

```
curl -X POST http://localhost:8000/chat/invoke \
  -H "Content-Type: application/json" \
  -d '{"input": {"query": "Create a strategy that uses daily trend and 4-hour entries"
```

🤝 Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/amazing-feature`)

3. Commit your changes (`git commit -m 'Add amazing feature'`)
4. Push to the branch (`git push origin feature/amazing-feature`)
5. Open a Pull Request

📜 License

This project is licensed under the MIT License - see the LICENSE file for details.

🙏 Acknowledgments

- Built with LangChain and LangServe
- Powered by Azure OpenAI
- PineScript strategies for TradingView

⚠️ Disclaimer

This API provides educational content about trading strategies. Always do your own research and consider consulting with financial professionals before making trading decisions. Past performance does not guarantee future results.