

Stock Analyzer Bot

Una aplicación de análisis financiero impulsada por IA construida con **smolagents**, **FastAPI** y **MCP** (Model Context Protocol). Este bot utiliza Modelos de Lenguaje de Gran Escala para orquestar herramientas de análisis financiero y generar informes de inversión profesionales.

Tabla de Contenidos

- [Descripción General](#)
- [Arquitectura](#)
- [Cómo Funciona](#)
- [Instalación](#)
- [Configuración](#)
- [Referencia de Módulos](#)
 - [main.py](#) - Motor de Análisis Principal
 - [api.py](#) - Backend FastAPI
 - [tools.py](#) - Wrappers de Herramientas Smolagents
 - [mcp_client.py](#) - Conexión al Servidor MCP
- [Endpoints de API](#)
- [Tipos de Análisis](#)
- [Frontend Streamlit](#)
- [Variables de Entorno](#)
- [Ejemplos de Uso](#)

Descripción General

El Stock Analyzer Bot transforma datos financieros en bruto en información accionable de inversión usando IA. A diferencia de las herramientas de análisis tradicionales que muestran números crudos, este sistema usa LLMs para:

1. **Orquestar** - Decidir qué herramientas llamar y en qué orden
2. **Ejecutar** - Llamar herramientas financieras MCP para obtener datos de mercado
3. **Interpretar** - Entender qué significan los números
4. **Sintetizar** - Combinar múltiples fuentes de datos en análisis coherente
5. **Reportar** - Generar informes markdown profesionales con recomendaciones

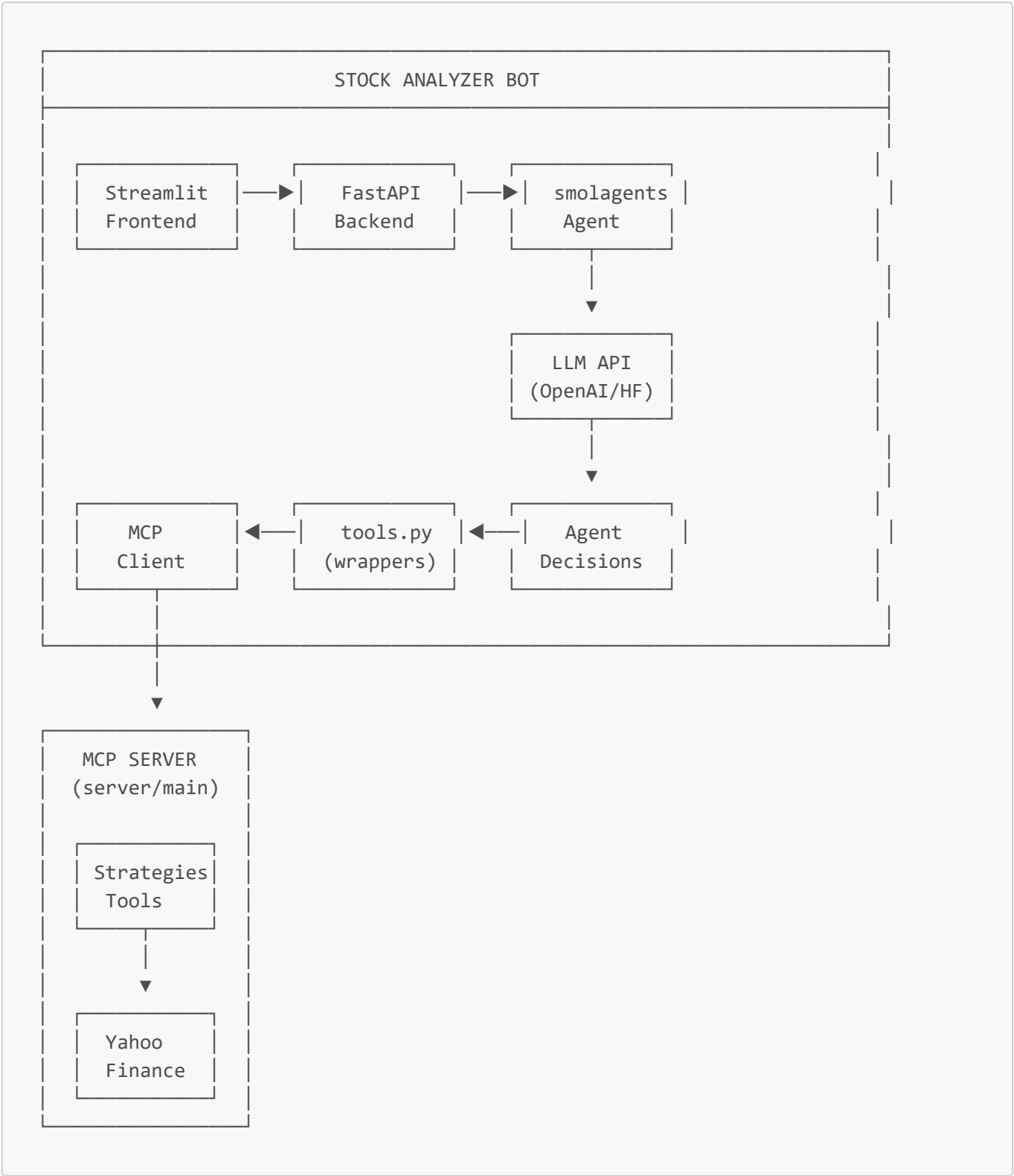
Características Clave

Característica	Descripción
5 Tipos de Análisis	Técnico, Escáner, Fundamental, Multi-Sector, Combinado
Informes Impulsados por IA	El LLM interpreta datos, no solo los muestra
Soporte Múltiples LLM	OpenAI, HuggingFace y otros modelos compatibles con LiteLLM
REST API	Backend FastAPI para integración

Característica	Descripción
Interfaz Web	Frontend Streamlit para análisis interactivo
Integración MCP	Se conecta al servidor MCP para herramientas financieras

Arquitectura

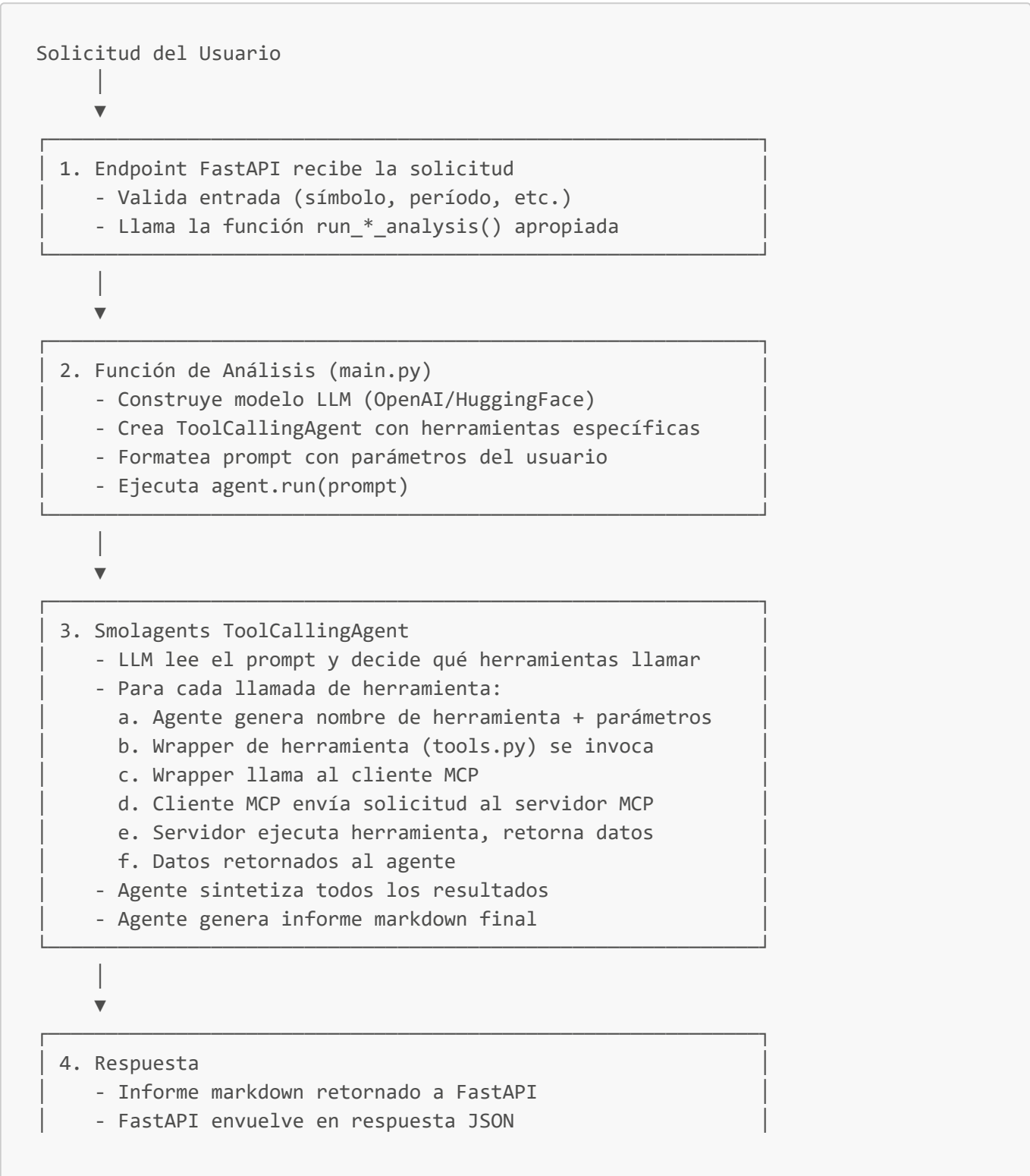
Descripción General del Sistema



Estructura de Carpetas

```
stock_analyzer_bot/
├── __init__.py           # Inicialización del paquete
├── main.py               # Motor de análisis principal con prompts LLM
├── api.py                # Endpoints REST de FastAPI
├── tools.py              # Wrappers de herramientas Smolagents para MCP
└── mcp_client.py         # Gestor de conexión al servidor MCP
```

Flujo de Datos



```
- Streamlit muestra informe formateado
```

Cómo Funciona

El Patrón Smolagents

Smolagents es un framework para construir agentes de IA que pueden usar herramientas. Así es como lo usamos:

```
from smolagents import ToolCallingAgent, LiteLLMModel

# 1. Construir el modelo (conexión al LLM)
model = LiteLLMModel(model_id="gpt-4o", api_key="...")

# 2. Crear agente con herramientas
agent = ToolCallingAgent(
    tools=[tool1, tool2, tool3], # Funciones que el LLM puede llamar
    model=model,
    max_steps=25, # Limitar iteraciones de razonamiento
)

# 3. Ejecutar con un prompt
result = agent.run("Analiza la acción AAPL usando todas las herramientas disponibles")
```

Flujo de Ejecución de Herramientas

Cuando el agente decide llamar una herramienta:

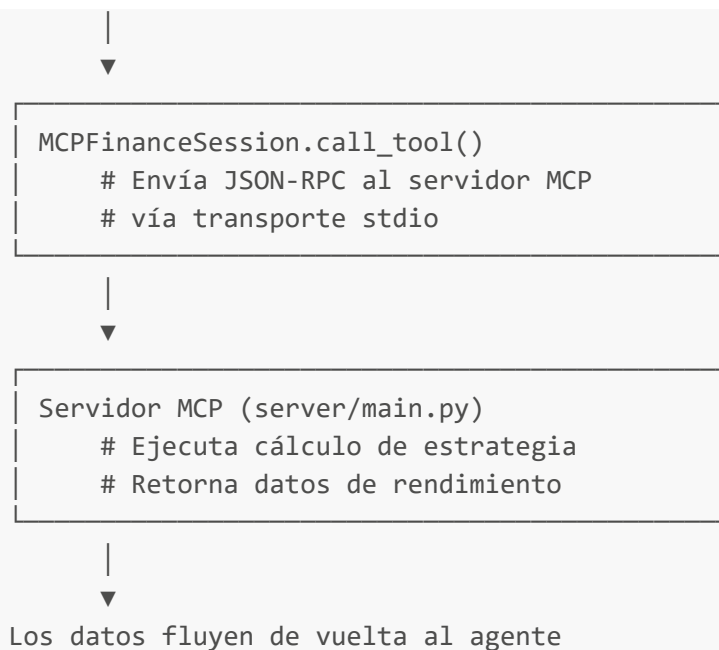
Decisión del Agente: "Necesito llamar bollinger_fibonacci_analysis para AAPL"



```
@tool
def bollinger_fibonacci_analysis(symbol):
    return _call_finance_tool(
        "analyze_bollinger_fibonacci...",
        {"symbol": symbol, "period": "1y"}
    )
```



```
_call_finance_tool()
    session = get_session()
    return session.call_tool(name, params)
```



Instalación

Prerrequisitos

- Python 3.10+
- Servidor MCP (carpeta `server/` en la raíz del proyecto)
- Clave API de OpenAI (o token de HuggingFace)

Dependencias

```
pip install smolagents fastapi uvicorn streamlit requests python-dotenv mcp
```

Inicio Rápido

```
# 1. Iniciar el backend FastAPI
uvicorn stock_analyzer_bot.api:app --reload --port 8000

# 2. (Opcional) Iniciar frontend Streamlit
streamlit run streamlit_app.py

# 3. (Opcional) Ejecutar análisis CLI
python -m stock_analyzer_bot.main AAPL --period 1y
```

Configuración

Variables de Entorno

Crear un archivo `.env` en la raíz del proyecto:

```
# Configuración LLM
OPENAI_API_KEY=sk-tu-clave-openai-aqui
OPENAI_BASE_URL=                        # Opcional: para endpoints personalizados
HF_TOKEN=hf_tu-token-huggingface       # Para modelos HuggingFace

# Valores Predeterminados del Modelo
SMOLAGENT_MODEL_ID=gpt-4.1              # Modelo predeterminado
SMOLAGENT_MODEL_PROVIDER=litellm        # litellm o inference
SMOLAGENT_MAX_STEPS=25                  # Máx iteraciones del agente

# Valores Predeterminados de Análisis
DEFAULT_ANALYSIS_PERIOD=1y
DEFAULT_SCANNER_SYMBOLS=AAPL,MSFT,GOOGL,AMZN

# Servidor API
STOCK_ANALYZER_API_URL=http://localhost:8000
```

Referencia de Módulos

main.py - Motor de Análisis Principal

El corazón de la aplicación. Contiene prompts LLM y funciones de orquestación de análisis.

Componentes Clave

Templates de Prompts:

```
TECHNICAL_ANALYSIS_PROMPT    # Acción única, 4 estrategias
MARKET_SCANNER_PROMPT        # Comparación de múltiples acciones
FUNDAMENTAL_ANALYSIS_PROMPT  # Estados financieros
MULTI_SECTOR_ANALYSIS_PROMPT # Comparación entre sectores
COMBINED_ANALYSIS_PROMPT     # Técnico + Fundamental
```

Funciones Constructoras:

```
def build_model(model_id, provider, ...) -> Model:
    """Crea LiteLLMModel o InferenceClientModel basado en el proveedor."""

def build_agent(model, tools, max_steps) -> ToolCallingAgent:
    """Crea smolagents ToolCallingAgent con herramientas especificadas."""
```

Funciones de Análisis:

Función	Propósito	Herramientas Usadas	Max Steps
<code>run_technical_analysis()</code>	Acción única, 4 estrategias	STRATEGY_TOOLS (4)	25
<code>run_market_scanner()</code>	Comparar múltiples acciones	STRATEGY_TOOLS (4)	50+
<code>run_fundamental_analysis()</code>	Estados financieros	<code>fundamental_analysis_report</code>	25
<code>run_multi_sector_analysis()</code>	Comparación entre sectores	STRATEGY_TOOLS (4)	100+
<code>run_combined_analysis()</code>	Téc + Fundamental	STRATEGY_TOOLS + fundamental	35

Firmas de Funciones

```
def run_technical_analysis(  
    symbol: str,                # ej., "AAPL"  
    period: str = "1y",        # Período histórico  
    model_id: str = "gpt-4.1",  # Modelo LLM  
    model_provider: str = "litellm", # Tipo de proveedor  
    hf_token: Optional[str] = None,  
    openai_api_key: Optional[str] = None,  
    openai_base_url: Optional[str] = None,  
    max_steps: int = 25,  
    ) -> str: # Retorna informe markdown
```

api.py - Backend FastAPI

API RESTful que expone todas las funciones de análisis.

Configuración de la Aplicación

```
app = FastAPI(  
    title="MCP Stock Analyzer API",  
    version="2.2.0",  
)  
  
# CORS habilitado para acceso del frontend  
app.add_middleware(CORSMiddleware, allow_origins=["*"], ...)  
  
# Eventos del ciclo de vida  
@app.on_event("startup")    # Inicializar conexión MCP  
@app.on_event("shutdown")  # Limpiar conexión MCP
```

Modelos de Solicitud

```
class TechnicalAnalysisRequest(BaseModel):
    symbol: str          # Requerido: símbolo ticker
    period: str = "1y"    # Período histórico
    model_id: Optional[str]
    model_provider: Optional[str]
    openai_api_key: Optional[str]
    hf_token: Optional[str]
    max_steps: Optional[int]

class MarketScannerRequest(BaseModel):
    symbols: str          # Separado por comas: "AAPL,MSFT,GOOGL"
    period: str = "1y"
    # ... mismos campos opcionales

class FundamentalAnalysisRequest(BaseModel):
    symbol: str
    period: str = "3y"    # Años de datos financieros
    # ... mismos campos opcionales

class MultiSectorAnalysisRequest(BaseModel):
    sectors: List[SectorConfig] # [{"name": "Banking", "symbols": "JPM,BAC"}]
    period: str = "1y"
    # ... mismos campos opcionales

class CombinedAnalysisRequest(BaseModel):
    symbol: str
    technical_period: str = "1y"
    fundamental_period: str = "3y"
    # ... mismos campos opcionales
```

Modelo de Respuesta

```
class AnalysisResponse(BaseModel):
    report: str           # Informe de análisis en markdown
    symbol: str            # Símbolo(s) analizados
    analysis_type: str     # "technical", "scanner", etc.
    duration_seconds: float # Tiempo de procesamiento
```

tools.py - Wrappers de Herramientas Smolagents

Conecta smolagents con el servidor MCP. Cada herramienta es una función decorada que el LLM puede llamar.

Categorías de Herramientas

STRATEGY_TOOLS (4 herramientas para análisis técnico):

```
STRATEGY_TOOLS = [
    bollinger_fibonacci_analysis,    # BB + Fibonacci
    macd_donchian_analysis,          # MACD + Donchian
    connors_zscore_analysis,         # Connors RSI + Z-Score
    dual_moving_average_analysis,     # Cruce 50/200 EMA
]
```

Herramientas Adicionales:

```
comprehensive_performance_report    # Informe multi-estrategia determinístico
unified_market_scanner              # Escáner multi-acción
fundamental_analysis_report         # Estados financieros
```

Patrón de Definición de Herramientas

```
from smolagents import tool

@tool
def bollinger_fibonacci_analysis(
    symbol: str,
    period: str = "1y",
    window: int = 20,
    num_std: float = 2.0,
    window_swing_points: int = 10,
) -> str:
    """Ejecuta el análisis de estrategia MCP combinado Bollinger-Fibonacci.

    Esta estrategia combina Bandas de Bollinger (reversión a la media) con
    niveles de retroceso de Fibonacci (soporte/resistencia) para análisis
    de precios integral.

    Args:
        symbol: Ticker a analizar (ej., 'AAPL', 'MSFT').
        period: Período histórico (predeterminado: '1y').
        window: Ventana retrospectiva de Bollinger (predeterminado: 20).
        num_std: Desviaciones estándar para bandas (predeterminado: 2.0).
        window_swing_points: Ventana de detección de puntos swing (predeterminado:
10).

    Returns:
        Informe de rendimiento detallado con señales y métricas.
    """
    params = {
        "symbol": _normalize_symbol(symbol),
        "period": period,
        "window": window,
```

```

        "num_std": num_std,
        "window_swing_points": window_swing_points,
    }
    return _call_finance_tool("analyze_bollinger_fibonacci_performance", params)

```

Funciones Helper Internas

```

def _normalize_symbol(symbol: str) -> str:
    """Limpia y valida símbolo ticker."""
    cleaned = symbol.strip().upper()
    if not cleaned:
        raise ValueError("Symbol must be a non-empty string")
    return cleaned

def _call_finance_tool(tool_name: str, parameters: Dict) -> str:
    """Llama herramienta del servidor MCP vía sesión."""
    try:
        return get_session().call_tool(tool_name, parameters)
    except Exception as exc:
        logger.exception("Error calling %s", tool_name)
        return f"Error calling {tool_name}: {exc}"

```

mcp_client.py - Conexión al Servidor MCP

Gestiona la conexión de larga duración al servidor financiero MCP.

Clase MCPFinanceSession

```

class MCPFinanceSession:
    """Gestiona una conexión de larga duración al servidor financiero MCP."""

    def __init__(self, server_path: Path = None):
        self.server_path = server_path or _DEFAULT_SERVER_PATH
        self._session: Optional[ClientSession] = None
        self._loop: Optional[asyncio.AbstractEventLoop] = None
        self._thread: Optional[threading.Thread] = None

    def start(self):
        """Inicia conexión al servidor MCP en hilo de fondo."""
        # Crea event loop async en hilo separado
        # Establece conexión stdio a server/main.py

    def stop(self):
        """Detiene conexión al servidor MCP."""

    def call_tool(self, name: str, arguments: Dict) -> str:

```

```
"""Llama una herramienta en el servidor MCP sincrónicamente."""
# Conecta código síncrono a llamadas MCP asíncronas
```

Patrón de Conexión

```
# Ruta predeterminada del servidor (relativa a mcp_client.py)
_DEFAULT_SERVER_PATH = Path(__file__).resolve().parents[1] / "server" / "main.py"

# Parámetros del servidor para transporte stdio
server_params = StdioServerParameters(
    command="python",
    args=[str(self.server_path)]
)

# Conexión vía protocolo MCP
async with stdio_client(server_params) as (read, write):
    async with ClientSession(read, write) as session:
        await session.initialize()
        # Sesión lista para llamadas de herramientas
```

Funciones a Nivel de Módulo

```
# Gestión de sesión global
_session: Optional[MCPFinanceSession] = None

def configure_session(server_path: Path = None):
    """Inicializa sesión MCP (llamada al inicio)."""
    global _session
    _session = MCPFinanceSession(server_path)
    _session.start()

def get_session() -> MCPFinanceSession:
    """Obtiene la sesión MCP activa."""
    if _session is None:
        configure_session()
    return _session

def shutdown_session():
    """Detiene la sesión MCP (llamada al cerrar)."""
    global _session
    if _session:
        _session.stop()
        _session = None
```

Endpoints de API

GET /health

Verificación de salud y lista de características.

Respuesta:

```
{
  "status": "ok",
  "version": "2.2.0",
  "features": [
    "technical_analysis",
    "market_scanner",
    "fundamental_analysis",
    "multi_sector_analysis",
    "combined_analysis"
  ],
  "model": {
    "default_id": "gpt-4.1",
    "default_provider": "litellm"
  }
}
```

POST /technical

Análisis técnico de acción única con 4 estrategias.

Solicitud:

```
{
  "symbol": "AAPL",
  "period": "1y"
}
```

Qué Sucede:

1. El agente llama 4 herramientas de estrategia para AAPL
2. Cada herramienta retorna métricas de rendimiento
3. El agente sintetiza en informe

Respuesta:

```
{
  "report": "# Análisis Técnico Completo de AAPL\n...",
  "symbol": "AAPL",
  "analysis_type": "technical",
  "duration_seconds": 45.2
}
```

POST /scanner

Comparación y clasificación multi-acción.

Solicitud:

```
{
  "symbols": "AAPL,MSFT,GOOGL,META,NVDA",
  "period": "1y"
}
```

Qué Sucede:

1. El agente llama 4 herramientas de estrategia para CADA acción (20 llamadas totales)
2. Compara rendimiento entre todas las acciones
3. Clasifica e identifica las mejores oportunidades

Respuesta:

```
{
  "report": "# Informe de Análisis Multi-Acción del Mercado\n...",
  "symbol": "AAPL,MSFT,GOOGL,META,NVDA",
  "analysis_type": "scanner",
  "duration_seconds": 180.5
}
```

POST /fundamental

Análisis de estados financieros.

Solicitud:

```
{
  "symbol": "MSFT",
  "period": "3y"
}
```

Qué Sucede:

1. El agente llama la herramienta fundamental_analysis_report
2. Obtiene estado de resultados, balance general, flujo de caja
3. Interpreta salud financiera y crea tesis

Respuesta:

```
{
  "report": "# Informe de Análisis Fundamental de MSFT\n...",
  "symbol": "MSFT",
  "analysis_type": "fundamental",
  "duration_seconds": 35.0
}
```

POST /multisector

Análisis comparativo entre sectores.

Solicitud:

```
{
  "sectors": [
    {"name": "Banking", "symbols": "JPM,BAC,WFC,C,GS"},
    {"name": "Technology", "symbols": "AAPL,MSFT,GOOGL,META,NVDA"},
    {"name": "Clean Energy", "symbols": "TSLA,NIO,ENPH,PLUG,NEE"}
  ],
  "period": "1y"
}
```

Qué Sucede:

1. El agente procesa cada sector
2. Llama 4 herramientas por acción (60 llamadas totales para 15 acciones)
3. Compara rendimiento ENTRE sectores
4. Identifica mejores oportunidades de todo el universo

Respuesta:

```
{
  "report": "# Informe de Análisis Multi-Sector del Mercado\n...",
  "symbol": "Banking, Technology, Clean Energy",
  "analysis_type": "multi_sector",
  "duration_seconds": 450.0
}
```

POST /combined

Análisis Técnico + Fundamental combinado.

Solicitud:

```
{
  "symbol": "AAPL",
  "technical_period": "1y",
  "fundamental_period": "3y"
}
```

Qué Sucede:

- 1. El agente llama fundamental_analysis_report
- 2. El agente llama 4 herramientas de estrategia técnica
- 3. Sintetiza AMBAS perspectivas
- 4. Determina si las señales se alinean o divergen

Respuesta:

```
{
  "report": "# Análisis de Inversión Combinado de AAPL\n...",
  "symbol": "AAPL",
  "analysis_type": "combined",
  "duration_seconds": 75.0
}
```

Tipos de Análisis

Tabla Comparativa

Tipo	Endpoint	Acciones	Herramientas/Acción	Propósito	Tiempo
Técnico	/technical	1	4	Análisis profundo acción única	30-60s
Escáner	/scanner	N	4	Comparar oportunidades	2-5min
Fundamental	/fundamental	1	1	Salud financiera	30s
Multi-Sector	/multisector	N×M	4	Comparación entre sectores	5-15min
Combinado	/combined	1	5	Análisis completo	60-90s

Cuándo Usar Cada Uno

Caso de Uso	Análisis Recomendado
"¿Debería comprar AAPL?"	Análisis Combinado
"¿Cuál es la mejor acción tech?"	Escáner de Mercado

Caso de Uso	Análisis Recomendado
"¿Es MSFT financieramente saludable?"	Análisis Fundamental
"¿Dónde debería invertir entre sectores?"	Análisis Multi-Sector
"¿Qué dicen los gráficos sobre TSLA?"	Análisis Técnico

Frontend Streamlit

El `streamlit_app.py` proporciona una interfaz web con 5 pestañas:

Características

- **Pestaña Análisis Técnico:** Acción única, selector de período
- **Pestaña Escáner de Mercado:** Entrada multi-acción, comparación
- **Pestaña Análisis Fundamental:** Análisis de estados financieros
- **Pestaña Multi-Sector:** Sectores configurables con agregar/eliminar
- **Pestaña Análisis Combinado:** Téc + Fundamental juntos

Estado de Sesión

```
st.session_state.messages      # Historial de análisis
st.session_state.api_url       # URL del backend
st.session_state.model_id      # Modelo LLM
st.session_state.model_provider # Proveedor
st.session_state.openai_api_key # Sobrescritura de clave API
```

Comunicación API

```
def call_api(endpoint: str, payload: Dict) -> Dict:
    """Llama backend FastAPI con configuraciones de modelo."""
    # Agrega model_id, model_provider, openai_api_key al payload
    # Timeout: 600s normal, 1200s para multi-sector
    response = requests.post(url, json=payload, timeout=timeout)
    return response.json()
```

Ejemplos de Uso

Uso CLI

```
# Análisis técnico básico
python -m stock_analyzer_bot.main AAPL

# Con período personalizado
```

```
python -m stock_analyzer_bot.main TSLA --period 2y

# Con modelo personalizado
python -m stock_analyzer_bot.main MSFT --model-id gpt-4o --model-provider litellm

# Guardar salida a archivo
python -m stock_analyzer_bot.main GOOGL --output report.md
```

API Python

```
from stock_analyzer_bot.main import (
    run_technical_analysis,
    run_market_scanner,
    run_fundamental_analysis,
    run_combined_analysis,
)

# Análisis Técnico
report = run_technical_analysis("AAPL", period="1y")
print(report)

# Escáner de Mercado
report = run_market_scanner("AAPL,MSFT,GOOGL", period="1y")
print(report)

# Análisis Fundamental
report = run_fundamental_analysis("MSFT", period="3y")
print(report)

# Análisis Combinado
report = run_combined_analysis("AAPL", technical_period="1y",
fundamental_period="3y")
print(report)
```

REST API

```
# Análisis Técnico
curl -X POST "http://localhost:8000/technical" \
  -H "Content-Type: application/json" \
  -d '{"symbol": "AAPL", "period": "1y"}'

# Escáner de Mercado
curl -X POST "http://localhost:8000/scanner" \
  -H "Content-Type: application/json" \
  -d '{"symbols": "AAPL,MSFT,GOOGL", "period": "1y"}'

# Multi-Sector
curl -X POST "http://localhost:8000/multisector" \
  -H "Content-Type: application/json" \
```

```
-d '{
  "sectors": [
    {"name": "Tech", "symbols": "AAPL,MSFT"},
    {"name": "Finance", "symbols": "JPM,BAC"}
  ],
  "period": "1y"
}'
```

Solución de Problemas

Problemas Comunes

Problema	Causa	Solución
"MCP server not found"	Ruta del servidor incorrecta	Verifica que <code>server/main.py</code> existe
"Connection refused"	FastAPI no está corriendo	Iniciar con <code>uvicorn</code>
"Timeout"	Demasiadas acciones	Reducir cantidad de acciones o aumentar timeout
"Authentication error"	Clave API inválida	Verifica <code>OPENAI_API_KEY</code> en <code>.env</code>
"Agent stopped early"	Más steps alcanzado	Aumentar parámetro <code>max_steps</code>

Modo Debug

```
# Habilitar logging de debug
export LOG_LEVEL=DEBUG
uvicorn stock_analyzer_bot.api:app --reload --port 8000
```

Historial de Versiones

Versión	Cambios
1.0.0	Lanzamiento inicial con análisis técnico
2.0.0	Agregado Escáner de Mercado, Análisis Fundamental
2.1.0	Agregado Análisis Multi-Sector
2.2.0	Agregado Análisis Técnico + Fundamental Combinado

Licencia

Este software se proporciona para fines educativos e investigación. Siempre verifica los resultados de análisis y consulta profesionales financieros antes de tomar decisiones de inversión.

Construido con [smolagents](#), [FastAPI](#) y [MCP](#)