Documentación Técnica: Stock Scanner Pro v1.2.0

1. Resumen Ejecutivo

Stock Scanner Pro es una plataforma analítica de alta performance diseñada para el monitoreo cuantitativo de activos financieros. El sistema integra algoritmos de análisis técnico multi-temporal (Daily/Weekly) para identificar patrones de capitulación y momentum, permitiendo a los usuarios filtrar universos de activos mediante reglas estricta de jerarquización operativa.

---

2. Pila Tecnológica (Tech Stack)

2.1 Backend Core

- **Lenguaje**: Python 3.10+
- **Framework Web**: Flask (WSGI compliant).
- **ORM**: SQLAlchemy para la gestión de la persistencia de datos relacionales.
- **Engine Técnico**:
  - `Pandas`: Estructuras de datos matriciales (DataFrames).
  - `Pandas-TA`: Biblioteca de análisis técnico para cálculos vectorizados de indicadores.

2.2 Persistencia de Datos

- **Motor**: SQLite 3.
- **Modelo Relacional**:
  - `Ticker`: Entidad maestra de activos. Implementa normalización automática de símbolos (ej. conversión de `BCBA:TICKER` a `TICKER.BA`).
  - `Price`: Serie temporal histórica. Almacena OHLCV (Open, High, Low, Close, Volume).

2.3 Frontend & Visualización

- **Arquitectura**: Single Page Application (SPA) basada en componentes nativos (Vanilla JS).
- **UI/UX**: Estética Dark-Premium con efectos de Glassmorphism (CSS Moderno).
- **Export Engine**: Integración con `SheetJS` (XLSX) para procesamiento de reportes del lado del cliente.

---

3. Lógica Cuantitativa y Estrategias

3.1 Estrategia 1: RSI + MACD Momentum Rebound

Diseñada para la detección de reversiones en zonas de agotamiento de venta.

- **Indicador RSI (p=14)**:
  - **Trigger de Captura**: Localiza el evento `RSI < 30` en una ventana retrospectiva de 365 días.
  - **Señal de Rebote**: Identifica el primer punto de inflexión donde `RSI > SMA(RSI, 14)` posterior al trigger de captura.
- **Indicador MACD (12, 26, 9)**:
  - **Filtro de Oportunidad**: Cruce alcista (`MACD > Signal`) condicionado a que el valor absoluto del MACD se encuentre en terreno negativo o neutro (`MACD <= 0`).
  - **Invalidación (Exit Strategy)**: La señal conmuta a estado `Inactive` (Visual: Red) de forma inmediata si el cruce se vuelve bajista o si el indicador supera el umbral de `0` .

3.2 Estrategia 2: 3-EMA Multi-Timeframe Alignment (4, 9, 18)

Estrategia de seguimiento de tendencia que busca la alineación de momentum en diferentes horizontes temporales.

- **Parámetros**: Medias Móviles Exponenciales (EMA) de corto (p=4), medio (p=9) y largo (p=18) recorrido.
- **Alineación Diaria (D)**: Validada mediante `Close > EMA4, EMA9, EMA18` .
- **Alineación Semanal (W)**:
  - **Algoritmo de Resampling**: Transforma la serie diaria en semanal utilizando el alias `W-FRI` (Weekly ending on Fridays).
  - **Seguridad de Datos**: El índice temporal se limita estrictamente a la fecha actual (`Capped Logic`) para prevenir la generación de etiquetas futuras y asegurar la integridad de los días transcurridos.

---

4. Gestión de Trazabilidad y Ordenamiento

El sistema implementa un algoritmo de **Ordenamiento Jerárquico por Desempate (Hierarchical Sorting)** para la Estrategia 2, asegurando que los activos con mayor fuerza relativa encabecen el listado:

1. **Prioridad de Alineación (Score)**: `(D_Active + W_Active)` . Puntuación máxima = 2.
2. **Jerarquía Temporal Mayor**: Ante igualdad de score, se prioriza el activo con el cruce **Semanal** más reciente (ASC).
3. **Jerarquía de Micro-Momentum**: Ante igualdad en la señal semanal, se desempata por el cruce **Diario** más reciente (ASC).

---

5. Capacidades de Interoperabilidad

5.1 API RESTful

- `GET /api/scan?strategy=[id]` : Motor de escaneo bajo demanda. Devuelve estructura JSON enriquecida con metadatos técnicos.
- `POST /api/refresh` : Sincronización asíncrona incremental con Yahoo Finance API.

- `POST /api/tickers` : Punto de entrada para nuevos activos con normalización inteligente BCBA/Standard.

5.2 Exportación de Datos (Data Portability)

El sistema permite la exportación íntegra del estado actual del dashboard a formato XLSX. Los archivos generados incluyen una nomenclatura estricta: `[StrategyName]_[YYYY-MM-DD_HH-MM].xlsx` .

---

6. Mantenimiento y Extensibilidad (AI Ready)

- **Dependency Management**: El entorno de ejecución se rige por el archivo `requirements.txt` .
- **Contexto de Agente**: El archivo `.agent` define las invariantes del sistema, prohibiendo expresamente el uso de datos sintéticos (placeholders) y garantizando la coherencia estilística en futuras expansiones.
- **Versionado**: Gestión de ramas por fecha (ISO 8601) para asegurar la trazabilidad de features.

---

7. Propuestas de mejora (rama: ztmp-01)

Se documentan propuestas priorizadas, preparadas en la rama "ztmp-01" para implementación incremental.

- Configuración y despliegue
    - Usar variable de entorno DATABASE_URL y ruta relativa (ej. `sqlite:///instance/scanner.db`) en lugar de ruta absoluta.
    - Añadir Dockerfile y preparar configuración para gunicorn/uWSGI en producción.
- Modelos y base de datos
    - Añadir relación Ticker.prices con backref y cascade delete.
    - Crear índice compuesto en Price (ticker_id, date) y considerar Numeric para precisión de precios.
    - Reemplazar uso de create_all por migraciones con Flask-Migrate / Alembic.
- Robustez y manejo de errores
    - Validar request.json y campos (evitar usar .get si request.json es None).
    - Manejar y loggear excepciones en sync_ticker_data; evitar silenciar errores con continue.
    - Usar transacciones y rollback en operaciones de escritura.
- Rendimiento y escalabilidad
    - Usar inserciones por lotes (bulk_save_objects / bulk_insert_mappings) o upserts para evitar queries por fila.
    - Validar y normalizar datos de yfinance (gestión de MultiIndex, NaN) antes de persistir.
    - Implementar caching de cálculos de señales (TTL) para evitar recomputos innecesarios.

- Arquitectura de ejecución
    - Desacoplar sincronizaciones a background jobs (Celery, RQ o APScheduler).
    - Añadir locking por ticker o encolamiento para evitar sincronizaciones concurrentes del mismo activo.
    - Implementar retry/backoff y control de rate-limit para llamadas a APIs externas.
- API y seguridad
    - Añadir endpoints: `GET /api/tickers/<id>/prices` (paginado), `POST /api/tickers/<id>/resync`, `GET /api/health`.
    - Añadir autenticación/autorization para rutas administrativas (token, OAuth o Flask-Login).
    - Normalizar formato de respuestas JSON y códigos HTTP.
- Funcionalidades y UX
    - Interfaz: lista de tickers con estado, gráficos interactivos (Chart.js), botón de re-sync por ticker.
    - Export CSV/JSON de precios y señales; historial de señales y backtesting básico.
    - Alertas (email/Telegram) para notificaciones de señales relevantes.
- Calidad y CI
    - Tests unitarios y de integración (mockear yfinance/pandas).
    - Linter y formateo (black, isort, flake8) y hooks pre-commit.
    - GitHub Actions para tests y lint en PRs.
- Observabilidad y despliegue
    - Logging estructurado y métricas (Prometheus / /metrics).
    - Preparar imágenes Docker y documentación de despliegue.

Notas:

- Prioridad sugerida para implementación: 1) mover DB URI a env var + validaciones de API, 2) migraciones y relación cascade, 3) desacoplar sync a background jobs.
- Puedo aplicar los cambios iniciales en la rama ztmp-01 y crear commits con pruebas unitarias y ejemplos de uso si autoriza la implementación.

---

8. Ejecución y despliegue

Se describen las formas recomendadas para ejecutar y desplegar la aplicación.

8.1 Ejecución local (desarrollo)

- Crear y activar entorno virtual, instalar deps:

```
python -m venv venv
venv\Scripts\activate      # Windows
```

```
source venv/bin/activate   # macOS / Linux
pip install -r requirements.txt
```

- Ejecutar servidor de desarrollo:

```
export FLASK_DEBUG=1
python app.py
```

o en Windows (PowerShell):

```
$env:FLASK_DEBUG = "1"
python app.py
```

8.2 Ejecución local (producción - gunicorn)

- Recomendado para pruebas de producción local:

```
pip install gunicorn
gunicorn app:app --bind 0.0.0.0:8000 --workers 3 --timeout 120
```

8.3 Despliegue en Render

- Campo "Start Command" (Start Command / Deploy):

```
gunicorn app:app --bind 0.0.0.0:$PORT
```

- Recomendado con workers:

```
gunicorn app:app --workers 3 --bind 0.0.0.0:$PORT --timeout 120
```

- Notas:
  - Render expone el puerto en la variable de entorno `$PORT`; no usar puerto fijo.
  - Asegúrese que `gunicorn` y dependencias estén en `requirements.txt`.
  - Si usa Swagger/Flasgger, incluya `flasgger` en `requirements.txt` (opcional: la app tolera su ausencia).

8.4 Despliegue con Docker (ejemplo mínimo)

- Dockerfile mínimo:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
ENV PORT=8000
CMD ["gunicorn", "app:app", "--workers", "3", "--bind", "0.0.0.0:8000", "--
timeout", "120"]
```

- Build & run:

```
docker build -t scanner-pro .
docker run -p 8000:8000 -e PORT=8000 scanner-pro
```

8.5 Variables de entorno importantes

- DATABASE_URL — URI SQLAlchemy (ej.: `sqlite:///instance/scanner.db` o `sqlite:///C:/full/path/instance/scanner.db` o una URL PostgreSQL).
- HOST / PORT — opcionales para ejecución local; Render provee $PORT.
- FLASK_DEBUG — "1" para activar modo debug local.

8.6 Troubleshooting común

- Error "unable to open database file":
  - Verificar que `instance/scanner.db` existe y permisos de escritura.
  - Para rutas relativas, la app ahora genera ruta absoluta; si persiste, pruebe con `DATABASE_URL=sqlite:///C:/ruta/completa/instance/scanner.db`.
- Error de módulo faltante en deploy (ej. `ModuleNotFoundError: No module named 'flasgger'`):
  - Añadir la dependencia a `requirements.txt` o dejar la importación opcional (la app soporta ambos modos).

---

9. Convenciones de Git y flujo de trabajo (estilo del proyecto)

Estas reglas deben registrarse y respetarse en el proyecto para mantener consistencia operativa.

- Publicación de ramas
  - Al crear una rama local se debe publicar en GitHub inmediatamente y configurar upstream:

```
git checkout -b nombre-rama
git push -u origin nombre-rama
```

- Mensajes de commit
  - Mensajes en español.
  - Asunto en presente indicativo (≤ 72 caracteres). Cuerpo opcional separado por línea en blanco.

```
Corregir validación en POST /api/tickers

Se añade comprobación de request.json y mensaje de error claro cuando
falta 'symbol'.
```

- Push obligatorio después de commit
  - Tras cada commit relevante, hacer push:

```
git add .
git commit -m "Mensaje en español"
git push
```

- Pull Requests
  - Crear PRs con título y descripción en español; referenciar issue/objetivo.
- Nomenclatura de ramas
  - feature/<breve-descripción>, fix/<breve-descripción>, chore/.
- Hooks y verificación automática (recomendado)
  - Configurar pre-commit hooks (black, isort, flake8).
- Buenas prácticas
  - Commits atómicos y pequeños.
  - Incluir tests al añadir funcionalidad.
  - Mantener rama principal protegida y usar PRs revisados.