**Requisitos Funcionales**

**Funciones Principales de la Base de Datos Financiera**

**Descripción:** La base de datos actuará como el repositorio central de datos históricos de precios y costos operacionales de activos financieros. Este sistema está diseñado para garantizar la integridad y consistencia de los datos, proporcionando una plataforma robusta para la gestión y análisis de información financiera.

Características Clave:

* Actualizaciones Diarias Automatizadas: La base de datos se actualizará automáticamente todos los días a una hora fija (por ejemplo, 7 PM, hora Colombia), asegurando que los datos estén siempre al día.
* Integridad y Consistencia: Se utilizarán constraints, normalización y triggers para garantizar la integridad y consistencia de los datos.
* Registro Detallado de Logs: Se mantendrá un registro detallado de logs que indique el estado de cada actualización (éxito o fallo), proporcionando transparencia y trazabilidad en el proceso de actualización de datos.

**Tipos de Usuarios y Permisos en el Sistema**

**Descripción:** El sistema de base de datos financiera cuenta con diferentes tipos de usuarios, cada uno con permisos específicos para garantizar la seguridad y la integridad de los datos. A continuación, se detallan los tipos de usuarios y sus respectivos permisos, así como el mecanismo de autenticación utilizado para operaciones críticas.

**Tipos de Usuarios:**

* **ADMIN:**
  + **Acceso Total:** El administrador tiene acceso completo para la administración, configuración y modificación del sistema.
  + **Almacenamiento Seguro de Credenciales:** Las credenciales del administrador se almacenan en un archivo seguro local, como un archivo .env que no se incluye en el repositorio1.
* **USUARIO:**
  + **Permiso de Consulta:** Los usuarios tienen permiso exclusivo para consultar o extraer datos históricos2.

**Mecanismo de Autenticación:**

* **Autenticación Básica con JWT:** Se implementará autenticación básica utilizando JSON Web Tokens (JWT) para operaciones críticas, como las modificaciones realizadas por el administrador3.

**Operaciones sobre Datos Financieros**

**Descripción:** Se desarrollará un proceso ETL documentado que extraiga, transforme y cargue datos desde las fuentes definidas, registrando en logs detallados cada operación. Estos logs incluirán indicadores de éxito o fallo y tiempos de ejecución, proporcionando una visión clara del rendimiento y la integridad del proceso.

**Notificaciones:** El sistema enviará alertas en consola si se detectan datos inconsistentes después de una actualización. Estas alertas permitirán una rápida identificación y corrección de problemas, asegurando la precisión y confiabilidad de los datos.

**Simulaciones y Backtesting**

**Descripción:** El sistema se centra en el almacenamiento y actualización de datos; la simulación y el backtesting se realizarán de forma externa usando los datos extraídos. Este enfoque permite una mayor flexibilidad y precisión en el análisis de datos financieros, ya que los usuarios pueden utilizar herramientas especializadas para sus simulaciones y pruebas retrospectivas.

**Implementación:**

* **Script de Instalación Único:** Se dispone de un script de instalación único (install.sh) que clona el repositorio, crea la base de datos y configura cron jobs para las actualizaciones. Este script simplifica el proceso de configuración inicial y asegura que todas las dependencias y configuraciones necesarias estén en su lugar.
* **Documentación en Markdown:** La documentación se proporcionará en formato Markdown, explicando el proceso de instalación y uso. Esto facilita la comprensión y el seguimiento de las instrucciones por parte de los usuarios, asegurando una implementación sin problemas.

**Trazabilidad de Datos**

**Descripción:** El sistema implementará un robusto mecanismo de trazabilidad de datos para garantizar la integridad y la transparencia en la gestión de la información. Este mecanismo incluirá versionado y auditoría, centralización de logs y un script para la recuperación de versiones anteriores.

**Versionado y Auditoría:**

* **Tablas de Auditoría:** Se implementarán tablas de auditoría con campos valid\_from y valid\_to para registrar los cambios críticos en los datos.
* **Centralización de Logs:** Los logs se centralizarán en la carpeta /logs con rotación automática.
* **Recuperación de Versiones Anteriores:** Existirá un script (restore\_backup.sh) para recuperar versiones anteriores basado en los backups diarios.

**Exportación de Datos**

**Descripción:** Se desarrollará un script (export\_data.py) con una interfaz en consola para que el usuario pueda seleccionar el activo, la temporalidad y el formato de exportación (CSV, JSON, XML). Este script permitirá a los usuarios extraer datos financieros de manera flexible y personalizada, facilitando su análisis y uso en diferentes aplicaciones.

**Ejemplo de Código:**

import argparse

import pandas as pd

def export\_data(asset, timeframe, file\_format):

# Simulación de datos para el ejemplo

data = {

'timestamp': ['2025-02-08 00:00', '2025-02-08 01:00', '2025-02-08 02:00'],

'price': [45000, 45200, 45150]

}

df = pd.DataFrame(data)

# Exportar datos en el formato seleccionado

if file\_format == 'csv':

df.to\_csv(f'{asset}\_{timeframe}.csv', index=False)

elif file\_format == 'json':

df.to\_json(f'{asset}\_{timeframe}.json', orient='records')

elif file\_format == 'xml':

df.to\_xml(f'{asset}\_{timeframe}.xml', index=False)

else:

print('Formato no soportado')

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser(description='Exportar datos financieros')

parser.add\_argument('--asset', type=str, required=True, help='Activo financiero (ej. BTCUSD)')

parser.add\_argument('--timeframe', type=str, required=True, help='Temporalidad (ej. 1h)')

parser.add\_argument('--format', type=str, required=True, help='Formato de exportación (csv, json, xml)')

args = parser.parse\_args()

export\_data(args.asset, args.timeframe, args.format)

**Volumen y Escalabilidad**

**Registros Diarios y Almacenamiento**

**Descripción:** El sistema manejará un gran volumen de datos financieros, con un cálculo estimado de registros diarios y almacenamiento necesario. Este cálculo se basa en la cantidad de activos, temporalidades y horas del día, proporcionando una visión clara de la capacidad de almacenamiento requerida para mantener la integridad y accesibilidad de los datos.

**Cálculo:**

* **Registros Diarios:** 1000 activos × 5 temporalidades × 24 horas = 120,000 registros/día.
* **Estimación de Almacenamiento:** 120,000 registros × 1 KB ≈ 120 MB/día, lo que equivale a aproximadamente 43 GB/año.

**Particionamiento y Compresión**

**Descripción:** Para optimizar el rendimiento y el espacio de almacenamiento, se implementará una estrategia de particionamiento y compresión en la base de datos. Esta estrategia permitirá una gestión más eficiente de los datos históricos, mejorando tanto la velocidad de acceso como la utilización del espacio.

**Estrategia:**

* **Particionamiento por Año:** La tabla histórica se particionará por año, lo que facilitará la gestión y consulta de grandes volúmenes de datos1.
* **Compresión en PostgreSQL:** Se utilizará compresión en PostgreSQL, por ejemplo, utilizando la herramienta pg\_repack, para optimizar el espacio de almacenamiento2.

**Expansión de Mercados**

**Descripción:** Para facilitar la integración de nuevas fuentes de datos y evitar conflictos con la estructura existente, se adoptará un diseño modular. Este enfoque permitirá una expansión ordenada y eficiente del sistema, asegurando que las nuevas fuentes de datos se manejen de manera independiente y sin interferencias.

**Diseño Modular:**

* **Nuevas Fuentes de Datos:** Las nuevas fuentes de datos, como las criptomonedas, se integrarán mediante tablas independientes (por ejemplo, Crypto\_Data). Esto evitará conflictos con la estructura existente y permitirá una gestión más eficiente de los datos1.

**Tiempo de Respuesta**

**Descripción:** Para optimizar el tiempo de respuesta en las consultas a la base de datos, se implementarán estrategias específicas que mejorarán significativamente la eficiencia y la velocidad de acceso a los datos. Estas estrategias incluyen la creación de índices compuestos y el uso de índices BRIN para consultas por rangos temporales.

**Optimización:**

* **Índices Compuestos:** Se implementarán índices compuestos en (asset\_id, timestamp) para acelerar las consultas1.
* **Índice BRIN:** Se usará un índice BRIN en el campo timestamp para mejorar las consultas por rangos temporales2.

**Fuentes de Datos**

**Conexión a MetaTrader 5**

**Descripción:** Se desarrollará un script (fetch\_data.py) que se conecte a MetaTrader 5 mediante una librería de Python (por ejemplo, python-metatrader5). Este script incluirá manejo de errores y reintentos específicos para cada broker, como Pepperstone, Darwinex, Tickmill, IC Market, XM y Admiral Market.

La documentación proporcionará ejemplos de conexión para cada broker, asegurando que los usuarios puedan establecer conexiones de manera eficiente y sin problemas.

**Web Scraping de Costos Operacionales**

**Descripción:** Para realizar web scraping de costos operacionales de manera eficiente y evitar bloqueos, se implementará una estrategia que incluye la rotación de user-agents y la introducción de delays aleatorios entre requests. Esta estrategia permitirá obtener datos de manera continua y sin interrupciones, asegurando la integridad y disponibilidad de la información.

**Estrategia:**

* **Rotación de User-Agents:** Se aplicará rotación de user-agents para simular diferentes navegadores y dispositivos, evitando así ser detectados y bloqueados por los sitios web1.
* **Delays Aleatorios:** Se introducirán delays aleatorios entre requests para imitar el comportamiento humano y reducir la probabilidad de ser bloqueados2.
* **Documentación de Ejemplos de Código:** Se documentarán ejemplos de código para facilitar la implementación y asegurar que los usuarios puedan replicar la estrategia de manera efectiva3.

Validación de Integridad de los Datos

Descripción: Para garantizar la integridad de los datos, se desarrollará un proceso de validación que compare indicadores como medias móviles, volúmenes y otros parámetros entre las distintas fuentes. Este proceso ayudará a identificar y corregir discrepancias en los datos, asegurando que la información sea precisa y confiable.

Proceso de Validación: Se creará un script (validate\_data.py) que compare indicadores como medias móviles, volúmenes y otros parámetros entre las distintas fuentes1.

Ejemplo de Código:

import pandas as pd

import logging

# Configuración del logger

logging.basicConfig(level=logging.ERROR)

log = logging.getLogger(\_\_name\_\_)

# Simulación de datos para el ejemplo

df1 = pd.DataFrame({'close': [100, 101, 102, 103, 104]})

df2 = pd.DataFrame({'close': [100, 101, 102, 103, 105]})

# Umbral de discrepancia permitido

umbral = 0.5

# Validación de integridad de los datos

if abs(df1["close"].mean() - df2["close"].mean()) > umbral:

log.error("¡Discrepancia en datos!")

else:

print("Datos consistentes")

**Base de Datos y Optimización**

**Modelo de Datos y Diagrama ER**

**Descripción:** Se ha diseñado un modelo relacional que incluye las tablas: Mercados, Sectores, Activos, Brokers, Datos\_Historicos y Costos\_Operacionales. Este modelo relacional está estructurado para garantizar la integridad y consistencia de los datos, facilitando la gestión y el análisis de la información financiera. El diagrama ER se incluirá en la documentación, proporcionando una representación visual clara de las relaciones entre las tablas. El código actualmente está en el lenguaje admisible para diagram.io, lo que permite una fácil edición y actualización del diagrama.

**Índices y Triggers**

**Descripción:** Para optimizar el rendimiento y la integridad de los datos en la base de datos, se implementarán índices y triggers específicos. Estos mecanismos permitirán acelerar las consultas y garantizar que no se inserten datos duplicados, mejorando así la eficiencia y la consistencia del sistema.

**Índices Propuestos:**

* **Índice BRIN en timestamp:** Se utilizará un índice BRIN en el campo timestamp para optimizar las consultas de rango1.
* **Índice Compuesto en (broker\_id, asset\_id):** Se implementará un índice compuesto en (broker\_id, asset\_id) para mejorar los filtros cruzados1.

**Triggers:**

* **Verificación de Duplicados:** Se crearán triggers que verifiquen duplicados antes de insertar datos, validando la combinación (asset\_id, timestamp).

**Uso de TimescaleDB**

**Descripción:** Para mejorar el rendimiento de las consultas sobre series temporales, se evaluará la adopción de TimescaleDB. Esta evaluación se basará en pruebas reales que demuestren una mejora significativa en el rendimiento, asegurando que la implementación de TimescaleDB sea beneficiosa para el sistema.

**Criterio de Adopción:** Se considerará el uso de TimescaleDB solo si las pruebas reales demuestran una mejora superior al 20% en el rendimiento de las consultas sobre series temporales1.

**Versionado y Manejo de Datos Erróneos**

**Descripción:** Para garantizar la integridad y la transparencia en la gestión de datos, se implementará una estrategia de versionado y manejo de datos erróneos. Esta estrategia permitirá registrar cambios críticos y gestionar datos incorrectos de manera eficiente, asegurando que la información sea precisa y confiable.

**Estrategia:**

* **Versionado mediante Tablas de Auditoría:** Se manejarán versiones mediante tablas de auditoría con campos valid\_from y valid\_to1.
* **Manejo de Datos Erróneos:** Los datos erróneos se gestionarán mediante triggers y, en su caso, se almacenarán en una tabla quarantine\_table para revisión manual2.

**Arquitectura del Sistema**

**Comunicación API**

**Descripción:** Se desarrollará una REST API sencilla utilizando FastAPI. Esta API permitirá a los usuarios interactuar con el sistema de manera eficiente y segura, proporcionando acceso a datos financieros y funcionalidades clave a través de endpoints bien definidos.

**Implementación:**

* **Endpoints Principales**: Se implementarán los siguientes endpoints para facilitar el acceso a los datos:
  + GET /assets para listar activos.
  + GET /historical\_data para extraer datos históricos.
* **Documentación Automática**: La documentación Swagger se generará automáticamente, proporcionando una interfaz interactiva para explorar y probar los endpoints de la API.

**Sistema de Mensajería**

**Descripción:** En lugar de utilizar Kafka debido a su complejidad, se optará por colas asíncronas con Redis para gestionar tareas secundarias. Esta decisión tecnológica permitirá una gestión más eficiente y simplificada de notificaciones y procesamiento en segundo plano, asegurando que las tareas se manejen de manera oportuna y sin sobrecargar el sistema.

**Decisión Tecnológica:**

* **Colas Asíncronas con Redis:** Se utilizarán colas asíncronas con Redis para gestionar tareas secundarias, como notificaciones o procesamiento en segundo plano

**Actualizaciones sin Caídas**

**Descripción:** Para asegurar que el sistema se mantenga operativo durante las actualizaciones, se desarrollará un proceso que permita descargar cambios desde GitHub, validar la integridad de los archivos mediante checksums y reiniciar los servicios en segundo plano sin interrumpir el servicio activo. Además, se documentará un procedimiento de rollback en caso de fallos, garantizando que cualquier problema pueda ser revertido de manera rápida y eficiente.

**Proceso:**

* **Script de Actualización:** Se desarrollará un script (update\_system.sh) que descargue cambios desde GitHub, valide la integridad mediante checksums y reinicie los servicios en segundo plano sin interrumpir el servicio activo1.
* **Procedimiento de Rollback:** Se documentará el procedimiento de rollback en caso de fallos

**Seguridad y Control de Acceso**

**Protección de Datos**

**Medidas Implementadas:**

* **Encriptación Avanzada:** Utilizamos encriptación AES-256 para asegurar la base de datos, empleando, por ejemplo, la extensión pgcrypto de PostgreSQL.
* **Almacenamiento Seguro de Credenciales:** Las credenciales del ADMIN se guardan en un archivo .env, el cual está excluido del repositorio para garantizar su seguridad.

**Auditoría y Logs**

**Implementación:**

* **Registro de Operaciones Críticas:** Se creará una tabla audit\_logs que registrará todas las operaciones críticas junto con la IP de origen y otros metadatos relevantes.
* **Almacenamiento y Consulta de Logs:** Los logs de auditoría se almacenarán y podrán ser consultados a través de la interfaz o mediante scripts específicos.

**Procesos ETL**

**Orquestación**

**Implementación:**

* **Configuración de Cron Jobs:** Se utilizarán cron jobs configurados en el archivo /etc/cron.daily para ejecutar el proceso ETL (Extracción, Transformación y Carga) de datos una vez al día a las 7 PM (hora Colombia). Este proceso automatizado garantiza que los datos se actualicen de manera oportuna y precisa, manteniendo la integridad y consistencia de la información almacenada en la base de datos.

**Manejo de Errores y Reintentos**

**Estrategia Implementada:**

* **Reintentos Automáticos:** Si una fuente falla, el script intentará reconectar hasta 3 veces, asegurando la continuidad del proceso.
* **Gestión de Datos Corruptos:** Los datos corruptos o inconsistentes se moverán a una quarantine\_table para su revisión manual, garantizando la integridad de la información.
* **Registro de Incidencias:** Todas las incidencias se registrarán en los logs y se notificarán en consola, proporcionando transparencia y trazabilidad en el manejo de errores.

**Interfaz de Usuario**

**Interfaz de Consola**

**Diseño:**

* Se desarrollará un menú interactivo en la consola con opciones numeradas. Por ejemplo:

$ python3 main.py

>> Seleccione opción: 1

>> Ingrese activo: BTCUSD

El menú permitirá acciones como:

* Exportar datos.
* Ver logs.
* Consultar estado del sistema.

**Documentación:**

* Se incluirán instrucciones claras de uso y ejemplos para que el usuario final pueda navegar fácilmente por las opciones. Por ejemplo, para exportar datos:

$ python3 main.py

>> Seleccione opción: 2

>> Ingrese activo: BTCUSD

>> Ingrese temporalidad: 1h

>> Ingrese formato de exportación: csv

9. Infraestructura

Backups

* Estrategia:
  + Se desarrollará un script daily\_backup.sh que ejecute pg\_dump para generar un backup diario.
  + El backup se cifrará y se podrá subir opcionalmente a Google Drive (se incluirán instrucciones para la configuración de la API de Google).

Monitoreo

* Implementación:
  + Se incluye la opción de instalar Grafana, con un dashboard preconfigurado (monitoring\_dashboard.json) para visualizar métricas de rendimiento y estado del sistema.
  + Se documentará el proceso de instalación y configuración de Grafana.

**Pruebas y Validación**

**Pruebas de Carga y Estrés**

**Herramienta Utilizada:**

* **Script de Pruebas:** Se desarrollará un script stress\_test.py utilizando Locust para simular 100 usuarios concurrentes y medir el rendimiento del sistema. Este script permitirá identificar cuellos de botella y evaluar la capacidad del sistema para manejar cargas elevadas.

**Ejemplo de Implementación:**

from locust import HttpUser, TaskSet, task, between

class UserBehavior(TaskSet):

@task

def index(self):

self.client.get("/")

class WebsiteUser(HttpUser):

tasks = [UserBehavior]

wait\_time = between(1, 5)

if \_\_name\_\_ == "\_\_main\_\_":

import os

os.system("locust -f stress\_test.py --host=http://example.com")

**Validación de Precisión de Datos**

**Proceso:**

* **Pruebas Unitarias:** Se implementarán pruebas unitarias que comparen muestras de datos con fuentes externas para asegurar la exactitud. Por ejemplo, se utilizarán datos de Yahoo Finance para validar los precios históricos de activos financieros.

**Ejemplo de Implementación:**

import pandas as pd

import requests

# Obtener datos de la fuente externa (Yahoo Finance)

response = requests.get('https://finance.yahoo.com/quote/BTC-USD/history')

external\_data = pd.read\_html(response.text)[0]

# Datos internos simulados

internal\_data = pd.DataFrame({

'Date': ['2025-02-08', '2025-02-09', '2025-02-10'],

'Close': [45000, 45200, 45150]

})

# Comparar datos internos con externos

for date in internal\_data['Date']:

internal\_price = internal\_data.loc[internal\_data['Date'] == date, 'Close'].values[0]

external\_price = external\_data.loc[external\_data['Date'] == date, 'Close\*'].values[0]

assert abs(internal\_price - external\_price) < 1, f'Discrepancia encontrada en la fecha {date}'

Este ejemplo muestra cómo se pueden comparar los datos internos con los datos obtenidos de Yahoo Finance para asegurar la precisión de los precios históricos.

**Pruebas de Seguridad**

**Estrategia:**

* **Pentesting y Fuzzing:** Se realizarán pruebas de pentesting y fuzzing antes del lanzamiento de cada versión para identificar y corregir vulnerabilidades. Se utilizarán herramientas como OWASP ZAP y Bandit para asegurar la robustez del sistema.
* **Ejemplo de Implementación:**
  + **OWASP ZAP:** Se configurará OWASP ZAP para realizar un escaneo completo de la aplicación web, identificando posibles vulnerabilidades como inyecciones SQL, XSS y CSRF. Por ejemplo:
* zap.sh -daemon -port 8080 -config api.key=12345
* zap-cli quick-scan http://example.com
  + **Bandit:** Se utilizará Bandit para analizar el código fuente en busca de vulnerabilidades de seguridad. Por ejemplo:

bandit -r my\_project/

.

**Documentación y Mantenimiento**

**Documentación Técnica**

**Formato:**

* Toda la documentación se estructurará en el GitHub Wiki del proyecto, proporcionando una plataforma centralizada y accesible para todos los miembros del equipo.
* **Ejemplos de Uso:** Se incluirán ejemplos detallados de cómo utilizar las diferentes funcionalidades del sistema. Por ejemplo, para ejecutar un script de exportación de datos:

$ python3 export\_data.py --asset BTCUSD --timeframe 1h --format csv

* **FAQs:** Se crearán secciones de preguntas frecuentes para abordar las dudas más comunes de los usuarios, facilitando la resolución rápida de problemas.
* **Instrucciones de Instalación:** Se proporcionarán guías paso a paso para la instalación y configuración del sistema, asegurando que los usuarios puedan poner en marcha el sistema sin dificultades. Por ejemplo:

$ git clone https://github.com/usuario/proyecto.git

$ cd proyecto

$ ./install.sh

* **Resolución de Errores Comunes:** Se documentarán soluciones para los errores más frecuentes que los usuarios puedan encontrar, mejorando la eficiencia en la resolución de problemas.

**Mantenimiento**

**Proceso:**

* **Etiquetado de Versiones en Git:** Se etiquetarán las versiones en Git para facilitar el seguimiento y la gestión de las versiones del proyecto. Por ejemplo, se utilizarán etiquetas como v1.0.0-stable para indicar versiones estables del software.
* **Clasificación de Issues:** Los issues se clasificarán en diferentes categorías, como bugs, mejoras y nuevas funcionalidades. Esto permitirá una gestión más eficiente del mantenimiento y futuras actualizaciones. Por ejemplo, un bug podría ser etiquetado como bug-crítico, mientras que una nueva funcionalidad podría ser etiquetada como feature-solicitada.