Portada e introducción

Título: “Módulos integrados del sistema de trading automatizado”
Evidencia: GA8-220501096-AA1-EV02 – Módulos integrados
Proyecto: Bot de trading automatizado (EMA + Fibonacci)
Nombre: Carlos Alvarez
Programa / Centro: ANALISIS Y DESARROLLO DE SOFTWARE. (2977356)
Fecha: 09 de diciembre de 2025

Introducción

“En este documento se presenta la integración de los módulos desarrollados para la aplicación web de un bot de trading automatizado, construida con Java, Spring Boot y MariaDB/MySQL. El sistema permite registrar activos y precios de mercado, exponiendo servicios web que serán la base para el cálculo posterior de indicadores técnicos como la EMA de 30 periodos y los niveles de Fibonacci.
La evidencia GA8-220501096-AA1-EV02 se centra en mostrar cómo se integran los módulos de modelo, acceso a datos, servicios y API REST, así como en documentar las interfaces de entrada y salida, las configuraciones de servidores y bases de datos, los ambientes de desarrollo y las pruebas realizadas sobre cada módulo.”

Módulos integrados del sistema

Los módulos integrados en esta versión del sistema de trading automatizado son:
Módulo 1: Gestión de activos (Asset). Administra los instrumentos financieros con los que puede operar el bot, permitiendo registrar y reutilizar símbolos como XAUUSD.
Módulo 2: Gestión de precios (Price). Registra y consulta los precios de cierre asociados a cada activo y fecha y hora determinadas.
Módulo 3: Servicios web (API REST). Expone endpoints HTTP que permiten a clientes externos, como Postman, interactuar con los módulos de activos y precios.

Documentación por módulo y componente

Descripción funcional
El módulo de activos permite administrar los instrumentos financieros sobre los que opera el bot de trading. Cuando se registra un precio para un símbolo que aún no existe, el sistema crea automáticamente el activo y lo almacena en la base de datos.

Componentes técnicos

- Entidad: Asset
- Campos principales:
- id (Long): identificador único del activo.
- symbol (String): símbolo del activo, por ejemplo XAUUSD.
- description (String): descripción opcional del activo.

Repositorio: AssetRepository

- Basado en Spring Data JPA.
- Operaciones principales:
- Búsqueda por id.
- Método findBySymbol(String symbol) para localizar un activo por su símbolo.

Servicio: AssetService

- Responsabilidades:
- Método getOrCreate(String symbol): busca un activo por su símbolo; si no existe, crea un nuevo registro con ese símbolo y una descripción por defecto, y lo guarda en la base de datos.
- Integra de forma transparente la lógica de negocio con el repositorio JPA.

Entradas y salidas principales

Aunque el módulo de activos no se expone directamente por un endpoint dedicado en esta versión, se utiliza de forma interna por el módulo de precios. Cada vez que se registra un nuevo precio para un símbolo, el sistema utiliza AssetService para asegurar que exista un registro en la tabla de activos.

Módulo 2: Gestión de precios (Price)

Descripción funcional

El módulo de precios se encarga de registrar y consultar los precios de cierre asociados a cada activo. Es la base para que, posteriormente, se puedan calcular indicadores técnicos como la EMA y los niveles de Fibonacci.

Componentes técnicos

- Entidad: Price
- Campos principales:
- id (Long): identificador del precio.
- asset (Asset): referencia al activo al que pertenece el precio (relación ManyToOne).
- closePrice (double): precio de cierre.
- timestamp (LocalDateTime): fecha y hora del precio registrado.

Repositorio: PriceRepository

- Basado en Spring Data JPA.
- Operaciones principales:
- findByAssetOrderByTimestampDesc(Asset asset): devuelve los precios de un activo ordenados por fecha y hora descendente.
- findByAssetAndTimestampBetweenOrderByTimestampAsc(Asset asset, LocalDateTime from, LocalDateTime to): permite consultar precios en un rango de tiempo

Servicio: PriceService

- Métodos principales:
- addPrice(String symbol, double closePrice, LocalDateTime timestamp):
- Obtiene o crea el activo mediante AssetService.
- Crea un nuevo objeto Price con el símbolo, el precio de cierre y la fecha y hora.
- Guarda el registro en la base de datos y devuelve el precio persistido.
- getLastPrices(String symbol, int periods):
- Consulta los precios más recientes para un activo y devuelve los últimos N registros, que servirán para cálculos de indicadores.

Controlador REST: PriceController

- Anotaciones: @RestController y @RequestMapping("/prices").
- Endpoint principal:

Endpoint: Registrar precio

- Método: POST
- URL: http://localhost:8080/api/prices
- Descripción: Registra un nuevo precio de cierre para un activo.

Datos de entrada (Query Params)

- asset (String): símbolo del activo.
- Ejemplo: XAUUSD.
- closePrice (double): precio de cierre.
- Ejemplo: 2100.5.
- timestamp (LocalDateTime, formato yyyy-MM-dd'T'HH:mm:ss): fecha y hora del precio.
- Ejemplo: 2025-11-28T13:15:00.

Datos de salida (JSON de ejemplo)


```
{
  "id": 1,
  "asset": {
    "id": 1,
    "symbol": "XAUUSD",
    "description": "XAUUSD auto creado"
  },
  "closePrice": 2100.5,
  "timestamp": "2025-11-28T13:15:00"
}
```


Integración de capas

El módulo de precios integra la capa de modelo (entidad Price), la capa de repositorio (PriceRepository), la capa de servicios (PriceService) y la capa de presentación (PriceController). De esta manera se garantiza una separación clara de responsabilidades y una fácil extensión futura para cálculos de EMA y Fibonacci.

Módulo 3: Servicios web (API REST)

Descripción funcional

El módulo de servicios web se encarga de exponer la funcionalidad del sistema a través de una API REST. En esta versión, el endpoint principal publicado es el de registro de precios, que se consume desde Postman.

Componentes técnicos

- Controladores anotados con @RestController.
- Configuración de contexto en application.properties:
- server.port=8080
- server.servlet.context-path=/api

URLs de los módulos integrados

- URL base de la API: http://localhost:8080/api
- Módulo de precios: http://localhost:8080/api/prices (POST)

Repositorio de control de versiones y ejecutables

Repositorio Git

- El código fuente del proyecto se gestiona mediante Git. El repositorio remoto se encuentra en GitHub (indicar la URL real del proyecto), por ejemplo:
- Repositorio: https://github.com/tuusuario/trading-bot-api
- Los cambios se registran mediante commits periódicos, lo que permite llevar el historial de evolución del proyecto y facilita la colaboración y el mantenimiento.

Archivos ejecutables

El sistema se empaqueta como un archivo JAR ejecutable usando Maven y Spring Boot. El proceso de compilación se realiza con:

mvn clean install

Este comando genera el archivo:

- target/trading-bot-api-0.0.1-SNAPSHOT.jar

El JAR contiene la aplicación completa y puede ejecutarse con:

java -jar target/trading-bot-api-0.0.1-SNAPSHOT.jar

Configuraciones de servidores, bases de datos y ambientes

Ambiente de desarrollo y pruebas

- Sistema operativo: Windows 11.
- Lenguaje: Java 17 / 21.
- Framework: Spring Boot 3.x.
- Gestor de dependencias: Maven.
- IDE: Visual Studio Code.
- Base de datos: MariaDB/MySQL.
- Configuración de servidor y contexto

En src/main/resources/application.properties se definió:

```
server.port=8080
server.servlet.context-path=/api

spring.datasource.url=jdbc:mysql://localhost:3306/trading_bot_db
spring.datasource.username=javauser
spring.datasource.password=Monic@10
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

Esta configuración indica el puerto, el contexto de la API, los datos de conexión a la base de datos y las propiedades de JPA/Hibernate utilizadas para la generación automática de tablas y el trazado de sentencias SQL.

Pruebas realizadas por módulo

Pruebas del módulo de precios

- Se realizaron llamadas al endpoint POST /api/prices desde Postman, enviando los parámetros asset, closePrice y timestamp con valores de prueba.
- En todos los casos válidos, el servicio respondió con código HTTP 200 y un cuerpo JSON que contiene el precio persistido junto con la información del activo.
- En la base de datos, al consultar la tabla price de la base trading_bot_db, se observaron los registros correspondientes a las pruebas, con referencias válidas a la tabla asset.

Pruebas de integración y compilación

- Se ejecutó el comando mvn clean install, obteniendo BUILD SUCCESS.
- Esto valida que el contexto de Spring Boot se inicializa correctamente, que los repositorios JPA están configurados y que la conexión con la base de datos es funcional.

Se recomienda acompañar esta sección con capturas de pantalla de Postman, de la consola de compilación y de la base de datos.

Manual técnico resumido

```
git clone https://github.com/tuusuario/trading-bot-api.git
cd trading-bot-api
```

Configuración de la base de datos

Crear la base de datos:

```
CREATE DATABASE trading_bot_db;
```

Crear usuario y otorgar permisos:

```
CREATE USER 'javauser'@'localhost' IDENTIFIED BY 'Monic@10';
GRANT ALL PRIVILEGES ON trading_bot_db.* TO 'javauser'@'localhost';
FLUSH PRIVILEGES;
```

Compilación y ejecución

```
mvn clean install
mvn spring-boot:run
```

La aplicación quedará disponible en:

- http://localhost:8080/api

Uso básico del endpoint de precios

- Método: POST
- URL: http://localhost:8080/api/prices
- Parámetros (query params):

1. **asset: símbolo del activo (ej. XAUUSD).**
2. **closePrice: precio de cierre (ej. 2100.5).**
3. **timestamp: fecha y hora en formato yyyy-MM-dd'T'HH:mm:ss (ej. 2025-11-28T13:15:00).**

Conclusiones

La integración de los módulos de activos, precios y servicios web demuestra el funcionamiento de una arquitectura por capas para un sistema de trading automatizado. El uso de Spring Boot y Spring Data JPA facilita la implementación y la conexión con la base de datos MariaDB/MySQL, permitiendo registrar y consultar datos de mercado a través de una API REST.

Las pruebas realizadas con Postman y las verificaciones en la base de datos confirman que los módulos se encuentran correctamente integrados y listos para ser extendidos con lógica de cálculo de indicadores técnicos como EMA y Fibonacci, así como con la generación de señales automáticas de operación.