Título: “Desarrollo de software a partir de la integración de sus módulos componentes”
Evidencia: GA8-220501096-AA1-EV01
Proyecto: Bot de trading automatizado (EMA + Fibonacci)
Nombre: Carlos Alvarez
Centro de formación / Programa: [lo que corresponda]
Fecha: 01 de diciembre de 2025

Introducción

“Este documento presenta el desarrollo de una aplicación web para un bot de trading automatizado, cuyo objetivo es analizar precios de mercado y generar señales de operación basadas en el cruce de la media móvil exponencial (EMA) de 30 periodos y niveles de retroceso de Fibonacci. La solución se implementa con arquitectura por capas utilizando Java, Spring Boot, Spring Data JPA y MariaDB/MySQL como base de datos, cumpliendo con los lineamientos de la evidencia GA8-220501096-AA1-EV01.”

Requerimientos e historias de usuario

En el documento crea una sección llamada: “Requerimientos del sistema / Historias de usuario” y añade algo así (puedes ajustar palabras):

Requerimiento 1: Registrar activos
El sistema debe permitir registrar y gestionar los activos financieros sobre los cuales operará el bot (por ejemplo: XAUUSD, EURUSD).

Requerimiento 2: Registrar precios históricos
El sistema debe almacenar precios de cierre asociados a cada activo y a una fecha y hora específicas para poder calcular indicadores técnicos.

Requerimiento 3: Calcular EMA de 30 periodos
El sistema debe calcular la media móvil exponencial (EMA) de 30 periodos a partir de los precios almacenados para un activo.

Requerimiento 4: Calcular niveles de Fibonacci
El sistema debe calcular niveles de retroceso de Fibonacci entre un mínimo y un máximo de precio definidos para un activo.

Requerimiento 5: Generar señales de operación
El sistema debe generar señales de compra o venta cuando el precio se encuentre en la zona del 70% de Fibonacci y la EMA 30 cruce por esa zona.

Requerimiento 6: Exponer servicios web (API REST)
El sistema debe exponer servicios web para registrar precios, consultar datos y disparar el cálculo de indicadores y señales, de forma que puedan ser consumidos por herramientas como Postman u otras aplicaciones.

Historias de usuario

- HU-01: Como usuario, quiero registrar activos para que el bot pueda analizarlos.
- HU-02: Como usuario, quiero guardar precios de cierre de un activo para que el sistema pueda calcular indicadores.
- HU-03: Como usuario, quiero que el sistema calcule la EMA de 30 periodos para un activo para identificar tendencias.
- HU-04: Como usuario, quiero que el sistema calcule los niveles de Fibonacci para detectar zonas relevantes de precio.
- HU-05: Como usuario, quiero que el sistema genere una señal de trading cuando el precio esté cerca del 70% de Fibonacci y la EMA 30 pase por esa zona.
- HU-06: Como usuario, quiero consumir la API desde Postman para probar y automatizar las operaciones del bot.

Diseño y Arquitectura del Sistema

Arquitectura por capas

“El sistema sigue una arquitectura por capas, separando la lógica de negocio, el acceso a datos, la exposición de servicios web y el modelo de dominio:
Capa de modelo (model): contiene las entidades de negocio como Asset y Price, que representan los activos y sus precios.
Capa de acceso a datos (repository): contiene las interfaces JPA (AssetRepository, PriceRepository) responsables de interactuar con la base de datos MariaDB/MySQL.
Capa de servicios (service): implementa la lógica de negocio para registrar activos, almacenar precios y, posteriormente, calcular indicadores como la EMA y los niveles de Fibonacci.
Capa de presentación / API (controller): expone endpoints REST (por ejemplo, /api/prices) que permiten a clientes externos (Postman u otras aplicaciones) registrar y consultar información del bot de trading.”

Diagrama simple de paquetes

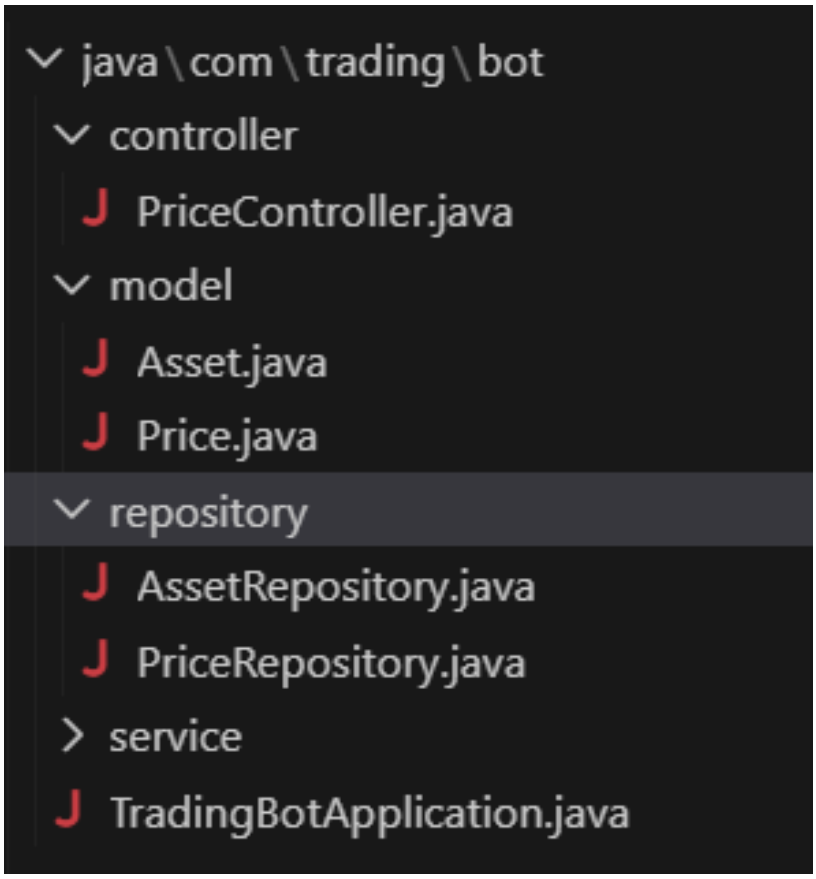
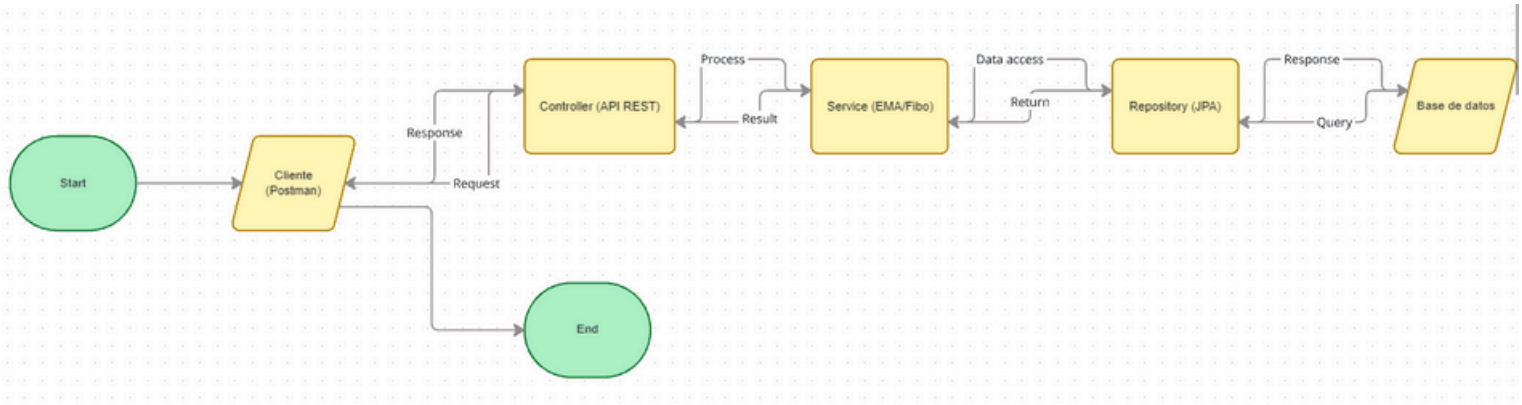


Diagrama de arquitectura



Mapa de navegación

“En esta versión inicial, la interacción se realiza mediante una API REST consumida desde Postman, por lo que el mapa de navegación se basa en la secuencia de llamadas a los endpoints: primero se registran precios con /api/prices, y posteriormente se consumirán endpoints para cálculo de EMA, Fibonacci y generación de señales.”

Desarrollo de módulos

Desarrollo de módulos e integración por capas

“Para el desarrollo de la solución se implementó una arquitectura por módulos y capas utilizando Java y Spring Boot. A continuación se describen los componentes principales desarrollados para el bot de trading.

Módulo de gestión de activos (Asset).
Se creó la entidad Asset, que representa cada instrumento financiero con el que puede operar el bot (por ejemplo, XAUUSD). Esta entidad incluye los campos id, symbol y description. Sobre esta entidad se implementó la interfaz AssetRepository, basada en Spring Data JPA, que permite realizar operaciones CRUD y una búsqueda específica por símbolo de activo. Encima del repositorio se desarrolló la clase AssetService, anotada con @Service, que contiene el método getOrCreate. Este método consulta si existe un activo por su símbolo y, si no existe, lo crea y lo persiste automáticamente, facilitando la gestión de activos desde la lógica de negocio.

Módulo de precios (Price).
Se implementó la entidad Price, que almacena los precios de cierre asociados a un activo y a un instante de tiempo determinado. La entidad incluye los campos id, asset (relación ManyToOne con Asset), closePrice y timestamp. Para el acceso a datos se definió la interfaz PriceRepository, que extiende JpaRepository y agrega consultas específicas para obtener los precios de un activo ordenados por fecha y para consultar precios en un rango de tiempo. Sobre este repositorio se creó la clase PriceService, también anotada con @Service, que ofrece dos operaciones principales: addPrice, que recibe el símbolo del activo, el precio de cierre y la fecha/hora, obtiene o crea el activo y guarda el nuevo registro de precio; y getLastPrices, que devuelve los últimos N precios registrados para un activo, lo cual servirá de base para el cálculo de indicadores como la EMA.

Exposición de servicios web (API REST).
En la capa de presentación se desarrolló la clase PriceController, anotada con @RestController y @RequestMapping("/prices"). Este controlador expone el endpoint POST /api/prices, que permite registrar precios desde clientes externos como Postman. El controlador recibe los parámetros asset, closePrice y timestamp, invoca al servicio PriceService y retorna el objeto Price persistido en la base de datos en formato JSON. De esta forma, se integra la capa de presentación con la lógica de negocio y el acceso a datos, cumpliendo con el enfoque de desarrollo de servicios web orientados a la reutilización de módulos.

Configuración del entorno y persistencia.
La aplicación se ejecuta sobre Spring Boot en el puerto 8080, utilizando el contexto /api. La configuración de la base de datos se realiza mediante el archivo application.properties, donde se define la URL de conexión a la base trading_bot_db, el usuario javauser y la contraseña configurada en MariaDB/MySQL. Se utiliza el driver oficial de MySQL y el proveedor de conexiones HikariCP, mientras que Spring Data JPA se encarga de generar y actualizar automáticamente las tablas a partir de las entidades. Esta configuración permite integrar de manera transparente los módulos de modelo, repositorio, servicio y controlador, garantizando una base sólida para la posterior incorporación de los cálculos de EMA, Fibonacci y la generación de señales de trading.”

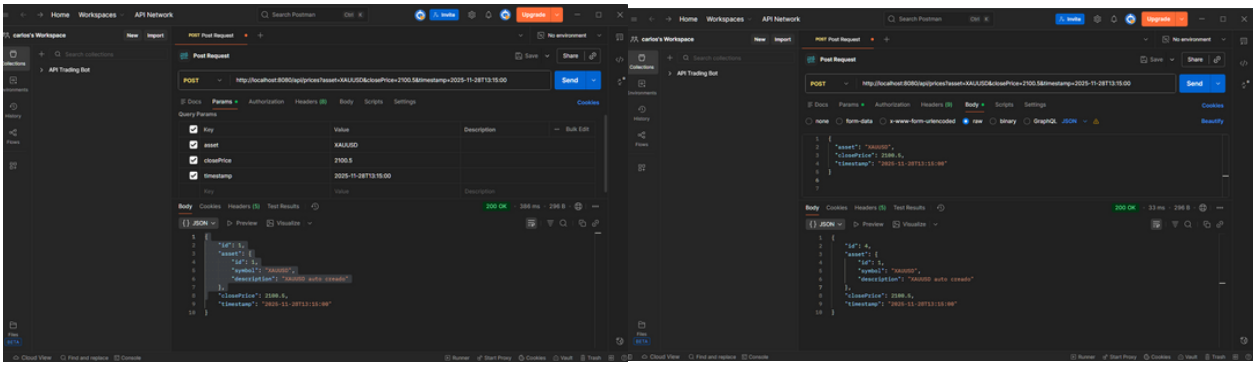
Pruebas

Pruebas y validación de la aplicación

“Para validar el funcionamiento de los módulos desarrollados se realizaron pruebas unitarias básicas y pruebas funcionales sobre la API REST utilizando Postman.

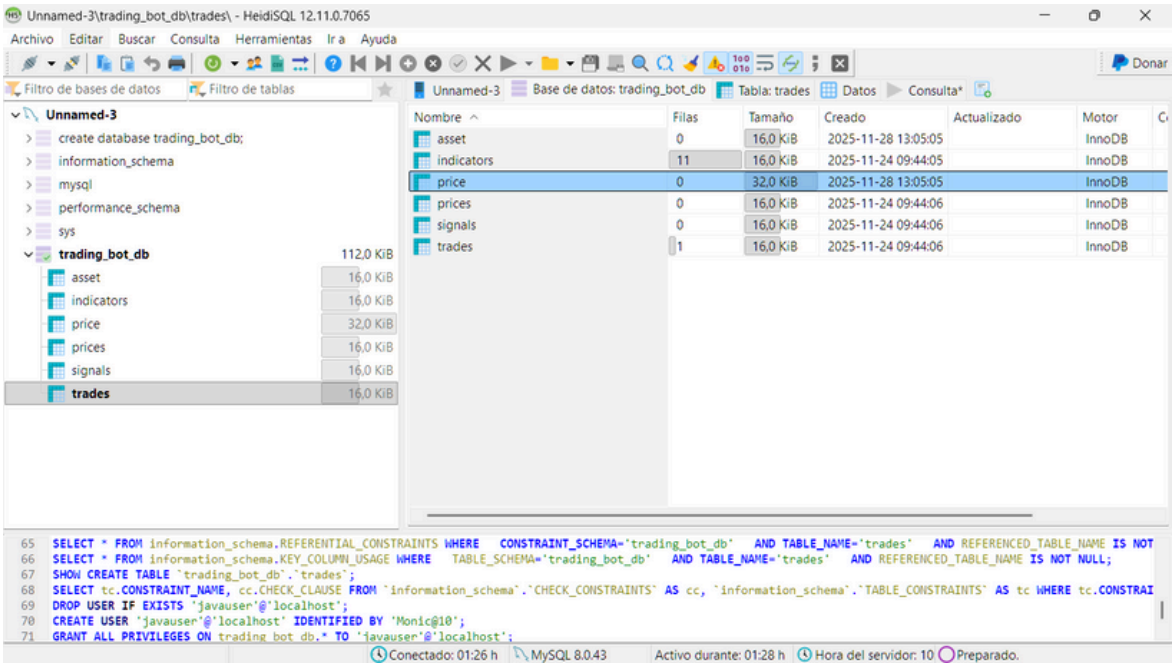
Pruebas funcionales con Postman.

Se ejecutó una prueba sobre el endpoint POST /api/prices, el cual permite registrar precios de cierre para un activo. Desde Postman se configuró una petición de tipo POST hacia la URL http://localhost:8080/api/prices, enviando los parámetros asset, closePrice y timestamp. Por ejemplo, se utilizó el activo XAUUSD, un precio de cierre de 2100.5 y un valor de fecha y hora en formato ISO 8601 (2025-11-28T13:15:00). La respuesta del servicio fue un objeto JSON que incluye el identificador generado, el símbolo del activo, el precio de cierre y la marca de tiempo registrada, lo que confirma que el módulo de precios funciona correctamente y persiste la información en la base de datos.

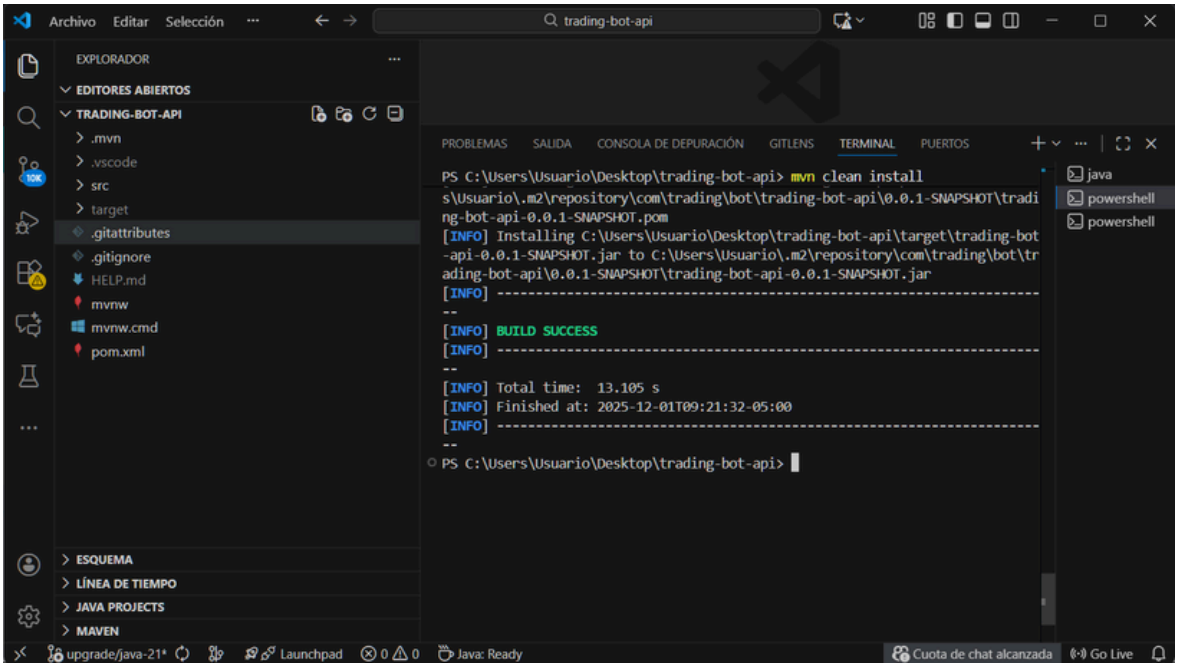


Verificación en la base de datos.

Posteriormente se verificó la inserción de los datos en la base de datos MariaDB/MySQL utilizando HeidiSQL. Al consultar la tabla price de la base trading_bot_db se observan los registros correspondientes a las pruebas realizadas desde Postman, con la relación correcta hacia la tabla asset. Esto evidencia la correcta integración entre la capa de servicios, el repositorio JPA y la base de datos.



Pruebas automatizadas.
Adicionalmente se ejecutó la suite de pruebas de Spring Boot mediante el comando mvn clean install, obteniendo un resultado BUILD SUCCESS. Esto confirma que el contexto de la aplicación se levanta correctamente, que la configuración de Spring Boot y de la base de datos es válida y que la estructura del proyecto cumple con los requisitos mínimos para su despliegue y ejecución.”



Conclusiones y cierre de la evidencia

“Como resultado del desarrollo de esta evidencia, se implementó una aplicación web básica para un bot de trading automatizado, construida sobre una arquitectura por capas con Java y Spring Boot. Se integraron los módulos de gestión de activos y registro de precios, conectados mediante servicios y repositorios JPA a una base de datos MariaDB/MySQL, cumpliendo con los lineamientos de diseño modular y reutilización de componentes.

Las pruebas realizadas con Postman y la verificación directa en la base de datos demostraron que el servicio para registrar precios funciona correctamente, que las entidades se mapean de forma adecuada y que la API responde con datos coherentes en formato JSON. Esto valida la integración de los módulos y deja la base lista para incorporar los cálculos de EMA, Fibonacci y la generación automática de señales de trading en futuras iteraciones.

Durante el desarrollo se aplicaron buenas prácticas como la separación de responsabilidades por paquetes, el uso de anotaciones de Spring para simplificar la configuración y la utilización de un repositorio de control de versiones, lo que facilita el mantenimiento, la evolución del proyecto y la colaboración en equipo.”

Checklist final y empaquetar

Haz una sección “Checklist de entrega” y marca:

- Código fuente completo del proyecto (carpetas src, pom.xml).
- Estructura por capas documentada (model, repository, service, controller).
- Configuración de servidor y base de datos documentada (application.properties).
- Pruebas funcionales documentadas con capturas de Postman y base de datos.
- Documento técnico en formato PDF con portada, requerimientos, diseño, desarrollo, pruebas y conclusiones.
- Archivo endpoints.txt con la lista de endpoints implementados.
- URL del repositorio en repositorio.txt (si lo tienes).
- Carpeta comprimida con todo el material lista para subir a la plataforma (nombre sugerido: TU_NOMBRE_GA8-220501096-AA1-EV01.zip).