**PROYECTO PERSONAL**

**Exchange de Criptomonedas – API REST**



**Fecha de inicio:** 18 / diciembre / 2024

**Fecha de finalización:**

**Estado del proyecto:** En desarrollo.

**Demostración del proyecto – Video YouTube:**

**Tecnologías utilizadas:** Java 17 LTS, Spring Boot 3, MySQL 8.2 / Flyway, Spring Data JPA, Maven, Lombook

**Objetivo:** Plasmar los conocimientos adquiridos en un proyecto de la vida real que me permita seguir practicando y aprendiendo sobre programación; demostrar por medio de un proyecto personal mis habilidades como programadora.

**Características del proyecto:** Sistema que permite gestionar la compra y venta de criptomonedas en un Exchange ficticio.

**Autora:** María Raquel Martínez Urrutia

Estudiante de Ing. en Desarrollo de Software en UnADM

**LinkedIn:** www.linkedin.com/in/raqmartinez

**Email:** [meztinea.dev@gmail.com](mailto:meztinea.dev@gmail.com)

# Requerimientos del Sistema

El Exchange permite a los usuarios registrarse en el sistema para tener una cuenta propia en la que pueden realizar compras y ventas de las criptomonedas disponibles. El usuario deberá tener la posibilidad de consultar las transacciones que ha realizado sobre sus criptomonedas y moneda fiduciaria.

## Requerimientos funcionales y no funcionales

Con base en las características del sistema se han identificado los requerimientos funcionales y no funcionales que guiarán la realización del proyecto.

**Requerimientos funcionales**

1. Es necesario que el sistema permita gestionar cuentas de usuario, incluyendo todas las operaciones de un CRUD, crear usuarios, consultar usuarios, modificar usuarios y eliminar usuarios. La operación de eliminación de usuario solo debe permitirse en el caso de que la cuenta del usuario se encuentre en cero, que no tenga criptomonedas ni dinero en moneda local.
2. El sistema debe permitir a los usuarios realizar depósitos o retiros en su moneda local, por ejemplo, pesos mexicanos.
3. El sistema debe mostrar una lista de las criptomonedas disponibles para comprar, indicando su precio actual.
4. El sistema debe permitir a los usuarios consultar las criptomonedas que tiene en su billetera digital.
5. El sistema debe permitir a los usuarios comprar y vender criptomonedas, utilizando como moneda de cambio su moneda local, por ejemplo, pesos mexicanos.
6. Las compras o ventas deben ejecutarse a precio de mercado, el usuario no puede indicar un precio de compra o venta.
7. Si el usuario quiere comprar o vender una criptomoneda deberá indicarse la cantidad de monedas que desea intercambiar.
8. Después de una compra o venta, el sistema debe actualizar el total de saldo de la billetera del usuario, actualizando las monedas compradas o vendidas y el total en su moneda local.
9. Cuando el usuario quiera comprar o vender criptomonedas, el sistema debe mostrar a los usuarios el precio de las criptomonedas en tiempo real.
10. El monto mínimo para realizar una compra o venta de criptomonedas será de 50 pesos mexicanos; no debe permitirse comprar o vender por debajo de este monto.
11. Se debe contar con un historial de transacciones para cada usuario, debiendo almacenar las transacciones como depósitos, compras, ventas y retiros.

**Requerimientos no funcionales**

1. Es necesario que el sistema se conecte con una base de datos para almacenar la información de manera persistente y permita consultar los datos en el futuro.
2. El sistema debe contar con una arquitectura que permita realizar mantenimiento de manera fácil y eficaz.
3. Se debe implementar protección contra ataques por medio de la encriptación de datos sensibles.
4. Es necesario garantizar que la disponibilidad del software sea mayor al 99%.
5. El sistema debe contar con una documentación ampliamente detallada para evitar futuros problemas en actualizaciones o modificaciones.

## Arquitectura en Capas (Layered Architecture)

Después de realizar el análisis el sistema que se espera como resultado, se identifica como una buena opción de modelo arquitectónico para el desarrollo del sistema el modelo de Capas (Layered Architecture). El proyecto se dividirá en las siguientes carpetas para mantener una separación de responsabilidades clara: controller, domain, service, repository, dto y configurations.

**Controller:** Se encargará de ser el punto de entrada para las solicitudes HTTP. Almacenará las clases RestController para gestionar las respuestas a las solicitudes del cliente. Se comunicará con la capa Service; solicitando datos y devolviendo los datos al cliente.

**Service:** Se encargará de la lógica de negocio del sistema, validar datos y coordinar la interacción entre varias entidades. Se comunica con la capa Controller; solicita operaciones de tipo CRUD a la capa Repository y devuelve resultados procesados al Controller para que sean entregados al cliente.

**Repository:** Interactúa directamente con la Base de Datos utilizando JPA/Hibernate. Contiene las interfaces de repositorios que definen los métodos para interactuar con las entidades de la Base de Datos. Recibe solicitudes desde la capa Service, devolviendo las entidades o resultados a esta capa.

**Domain:** Contiene las clases POJO o que representan tablas en la base de datos. Estas clases serán utilizadas por la capa Repository para mapear las tablas a objetos en Java.

**DTO:** Almacena las clases de transferencia de datos entre capas, que permitan evitar que las entidades se expongan al cliente. Las clases contienen los métodos para convertir entre DTOs y entidades. Recibe datos desde la capa Service o transforma directamente las entidades de Domain.

Además de las capas mencionadas también participa la base de datos, que no se considera una capa en esta arquitectura, pero tiene una de las funciones más importantes del sistema, almacenar los datos de manera que persistan en el tiempo.

Diagrama

Descripción generada automáticamente

Ilustración . Arquitectura en Capas.

## Base de datos

Dado que el proyecto se ha de realizar con el lenguaje Java, el Sistema Gestor de Base de Datos (SGBD) MySQL se considera como la mejor opción para la creación y gestión de la base de datos.

Se han identificado las entidades que conforman la base de datos por medio de los requerimientos que se identificaron anteriormente. Las tablas que definirán la estructura de la base de datos son: usuario, billetera, transacción y criptomoneda.

El Diagrama Relacional muestra todas las entidades involucradas en el almacenamiento de los datos importantes.

Diagrama

Descripción generada automáticamente

Ilustración . Diagrama Relacional de la base de datos.

## Modelado a partir de UML

Se identificaron las entidades para el desarrollo del sistema y se plasmaron en el siguiente diagrama de clases. El diagrama de clases permitirá identificar cómo se relacionan los objetos entre sí, cuáles son sus atributos y métodos.

Diagrama

Descripción generada automáticamente

Ilustración . Diagrama de clases.

Con base en los requerimientos funcionales identificados, se elaboró el diagrama de casos de uso que permite tener una visión clara sobre la interacción del usuario con el sistema.

Diagrama

Descripción generada automáticamente

Ilustración . Diagrama de casos de uso.

## Gestión del Proyecto (SCRUM)

**Sprint 1 - Definición del proyecto Pt 1 (1 Semana) 18-24 diciembre 2024**

* Documentar las especificaciones del proyecto.
* Establecer los requisitos funcionales y no funcionales del proyecto.
* Desarrollar el diagrama de clases y de casos de uso.

**Sprint 2 - Definición del proyecto Pt 2 (1 Semana) 25-31 diciembre 2024**

* Definir la arquitectura que seguirá el proyecto.
* Definir la estructura de la base de datos.

**Sprint 3 - Creación de la arquitectura del proyecto y la Base de Datos. (1 Semana) 1-7 enero 2025**

* Crear la arquitectura del proyecto para comenzar a desarrollar el código.
* Crear la estructura de la base de datos en el SGBD.
* Configurar el control de versiones de la BD con Flyway.

**Sprint 4 – Gestionar cuentas de usuario (POST, GET, PUT, PATCH, DELETE). (1 Semana) 8-14 enero 2025**

* Crear los métodos que gestionen solicitudes HTTP respecto a los usuarios.
* Verificar el correcto funcionamiento de envío y solicitud de datos a la base de datos.

**Sprint 5 – Gestionar criptomonedas (POST, GET, PUT, DELETE). (1 Semana) 15-21 enero 2025**

* Crear los métodos que gestionen solicitudes HTTP respecto a criptomonedas.
* El precio de las criptomonedas debe actualizarse de automáticamente, en segundo plano, cada 15 minutos con los datos de la API REST de Bitso. (Ejecutar una tarea programada para un método asíncrono que actualice el precio).

**Sprint 6 – Gestionar depósitos en MXN. (2 Semanas) 22 enero – 04 febrero 2025**

* Crear el método para depositar MXN en la cuenta de un usuario y almacenarlo en la tabla de billetera como moneda MXN.

**Sprint 7 – Gestionar transacciones de depósitos.**

**(1 Semanas) 05 febrero – 11 febrero 2025**

* Cada vez que se registra un depósito debe ser incluido en el historial de transacciones, incluyendo el id de la transacción, id del usuario, id de la criptomoneda, el tipo (deposito, retiro, compra o venta), fecha de la transacción, monto en MXN, cantidad de la criptomoneda.

## Desafíos del proyecto

En esta sección se incorporan los desafíos con los que he tenido que lidiar al momento de realizar este proyecto.

**Crear método asíncrono programado**

Se creó un método asíncrono activando el Scheduling . El método consulta por medio de una solicitud HTTP a la API de Bitso el precio de cada criptomoneda almacenada en la base de datos y cada 15 minutos actualiza el precio de las criptomonedas.

**Crear una entidad con id compuesto**

Se creó una entidad que maneja un id compuesto por los ids de otras dos entidades. Esto se realizó para asegurar que dos objetos del mismo tipo no pueden contener las mismas claves de las entidades juntas. Es decir, solo un objeto puede tener esta combinación de claves identificadoras. De esta manera, las billeteras de criptomonedas tendrán una clave compuesta, combinando el id del usuario y el id de la criptomoneda, evitando que el mismo usuario pueda tener dos billeteras de la misma criptomoneda.