TP1 - Ledger - Reportes

Objetivo del trabajo:
Desarrollar un sistema de libro contables (ledger) que registre transacciones de diferentes monedas entre usuarios. El sistema se basará en archivos CSV para la simplicidad, funcionando como una base de datos accesible.

Componentes principales

El sistema constará de dos archivos CSV centrales:

monedas.csv:

Función: Actúa como el registro maestro de las monedas disponibles y su valor de referencia en USD.

Formato: Cada línea representa una moneda.

Estructura:

```
nombre_moneda;precio_usd
```

Ejemplo:

```
BTC;55000
ETH;3000
ARS;0.0012
USDT;1
EUR;1.18
```

transacciones.csv:

Función: Registra immutablemente toda actividad financiera en el sistema. Cada línea es una transacción.

Formato: Cada línea representa una transacción única.

Estructura:

```
id_transaccion;timestamp;moneda_origen;moneda_destino;
monto;cuenta_origen;cuenta_destino;tipo
```

- `id_transaccion` es un identificador unico por transaccion
- `fecha_hora` indica en que momento se realizo la transaccion
- `moneda_origen` y `moneda_destino` indican en que moneda esta el valor de la cuenta de origen y destino, respectivamente
- `monto` indica el monto de la transaccion
- `cuenta_origen` y `cuenta_destino` son los identificadores unicos de las cuentas de origen y destino, respectivamente
- `tipo` indica el tipo de transaccion, y este puede ser:
  - `transferencia` : una transaccion que transfiere un monto de una cuenta a otra
  - `swap` : una transaccion que convierte un monto en una moneda a otra
  - `alta_cuenta` : una transaccion que crea una cuenta y fija su valor inicial

Ejemplo:

```
1;1754937004;USDT;USDT;100.50;userA;userB;transferencia
2;1755541804;BTC;USDT;0.1;userB;;swap
3;1756751404;BTC;;50000;userC;;alta_cuenta
```

Funcionalidades requeridas

El programa se debe poder ejecutar desde un comando de linea de comando nombrado `ledger` .
El comando tendra dos subcomandos: `transacciones` y `balance` . Su funcionalidad se describira a continuacion.

Se debe crear un comando ejecutable `./ledger` . El mismo debe tener 2 subcomandos posibles:

- transacciones
- balance

Para todas las funcionalidades pedidas se espera que se puedan flexibilizar con el uso de los siguientes flags:

-c1: especifica la cuenta origen (si no se completa, toma todas las cuentas)
-c2: especifica la cuenta destino (si no se completa, toma todas las cuentas)
-t: archivo transacciones input (si no se completa toma por default `transacciones.csv`)
-m: moneda a utilizar para el cálculo de balances.
-o: archivo output (si no se completa se imprimen por terminal)

1. **Listar transacciones**: el subcomando `transacciones` listará por pantalla todas las transacciones que cumplen con los flags propuestos.

Ejemplos:

```
./ledger transacciones
Lista por terminal todas las transacciones del archivo transacciones.csv

./ledger transacciones –t=transac.csv –c1=345 –o=result.csv
Lista todas las transacciones del archivo transac.csv que fueron realizadas
desde la cuenta 345 y las almacena en el archivo result.csv
```

2. **Listar balance**: el subcomando `balance` calculará el balance de una cuenta. En este subcomando, el flag `–c1` es obligatorio. Si no se completa el flag `–m` deberán listarse los balances de todas las monedas de esa cuenta en el formato:

```
MONEDA=BALANCE(6 decimales siempre)
BTC=2.453445
ARS=234435345.000000
```

Ejemplos:

```
./ledger balance –c1=867
Lista por terminal el balance de todas las monedas de la cuenta 867.

./ledger balance –c1=867 –m=BTC
Lista por terminal el balance la cuenta 867 convertido a BTC.
```

Consideraciones

- En caso de encontrar una incosistencia, se debe mostrar por pantalla la tupla `{:error, <nro linea>}` . Por ejemplo si la línea 45 no cuenta con el formato indicado, de devolverá `{:error, 45}` , de la misma manera si una transacción se

realiza en una moneda no existente, si un monto es negativo, etc. (indicar en el README todos los errores manejados).

- El trabajo se entregará por Algotrón en un .zip (se les creará usuario a la brevedad).
- Se debe documentar como compilar y correr el proyecto en el README.
- El proyecto debe tener un test coverage de al menos el 90%.
- Para generar un ejecutable, se recomienda seguir la siguiente guía: https://elixirschool.com/en/lessons/intermediate/escripts (https://elixirschool.com/en/lessons/intermediate/escripts)
- Fecha límite de entrega 15/09/2025 a las 23.59.