You have to build a micro service called “Ledger” using the latest java and spring boot version.

The Requirement of the microservice is as follow:

1. Ledger is the source of truth for balances of assets and liabilities of an entity. So, the main responsibility of the ledger not only is to provide the latest but also all historical balances of any assets of an entity.
2. The ledger should support multi asset accounts owned by any entity. Examples of assets are fiat currency, crypto, stock, bond etc. Each asset is represented as a wallet. One entity can have multiple accounts. One account can have multiple wallets of different assets.
3. The client of the ledger should be able to move assets from one wallet to another.
4. The client of the ledger should be able to make multiple movements of assets in a single request.
5. All the requests to the ledger have to be executed as “all or nothing”.
6. The ledger should be able to manage the lifecycle of an account. For example OPEN state of an account means postings can happen to/from any wallets of the account. CLOSED means postings cannot happen to/from any wallets of the account. There could be more lifecycle state of an account. You have to design and document them for the client of the ledger.
7. The client of the ledger should be able to change the state of the account from one to another.
8. Similar to the account lifecycle there could be multiple life cycles of postings(movement). Such as PENDING, CLEARED, FAILED. You have to design and document them for the client of the ledger.
9. The client of the ledger should be able to change to postings it has done before.
10. The ledger should broadcast any balance change of any wallet for its client to listen to.
11. The ledger should broadcast any movement happening in the ledger for its client to listen to.
12. The client of the ledger should be able to make movement requests in asynchronous mode.
13. The client may request any historical balance of a wallet. The ledger should be able to provide that. Such as the client might request for the balance of a wallet at a given timestamp. The ledger should be able to entertain the client with the response.
14. Communication between the client can be established using both synchronous and asynchronous modes. Even though we prefer asynchronous communication where possible.
15. Ledger database will be very write heavy so design the database structure and usage keeping that in mind. Hints: “CQRS”.
16. The ledger should have a manual for its clients so that the client can use it efficiently.

If in some cases you feel requirements are not specific enough, you can make reasonable assumptions and document them. We will check your ability in implementing a scalable solution breaking down high level requirements into smaller technical parts and implementing them in code.