Design Choices

1. There is a complete separation of the banks from the clients. The interaction has to happen via the payment gateway.
2. The clients are able to do four types of transactions - payments, deposits, withdraws and views. All of them are carried out with 2PC protocol to ensure atomicity.
3. To simulate authentication, the client is sending passwords along with their messages, and the passwords are stored in the SHA-256 format and an exact match is carried out at the gateway to authenticate the requests.
4. To ensure Idemopotency, every request is piggybacked with a UUID which is generated randomly and is unique. This bank server checks whether a request of the same UUID has been carried out before or not, and in case if it has, the request is not executed. The code is designed in such a way that such retries are not happening, however, it has been tested by making slight and temporary modifications to the code.
5. To simulate buffering of requests (only at the client end), the user flags to `fail` the client - a simulation of the client getting disconnected from the server. When this happens, the client will still keep accepting user responses, but these will be queued. Once the client returns (flag unset using `recover` user input), all the requests that were stored are carried out in a FIFO order.
6. Appropriate locking is required to ensure mutual exclusion - common variables should not be updated by multiple threads simultaneously.
7. Account creation is simulated by an Admin Service implemented at the client end which receives information such as balance and password from the clients and stores the details in `/account_details/<server_number>.csv`. Note that I am operating with the assumptions that a client will have only one username which it uses across accounts (I am taking username to be a primary key, but this is easily modifiable). Additionally, one client can have only one account at a given bank (assumption valid and verified by HackMD doubts document).
8. ETCD is used for bank server discovery. Requests can be carried out only with an alive bank server (no buffering here), and accounts can be created with a bank only when its server is alive.
9. Extensive error handling has been carried out, with error messages specifying concretely the reasons for failure - network, authentication, bank failure, etc.
10. 2PC has two phases - prepare and commit. In the prepare phase, a message with an appropriate timeout are sent to all the bank servers involves in the transaction. When both are prepared, commit phase proceeds wherein both the bank servers involbed will commit. This ensures atomicity. In case either one is not available, or the timeout is exceeded, the abortion of request will proceed.