Architecture Document

for

BlockPe

Version 1.0

Prepared by

Group: 1

Chinmay Hiran Pillai	**200298**	**chinmay20@iitk.ac.in**
Raj Vardhan Singh	**200760**	**rajvs20@iitk.ac.in**

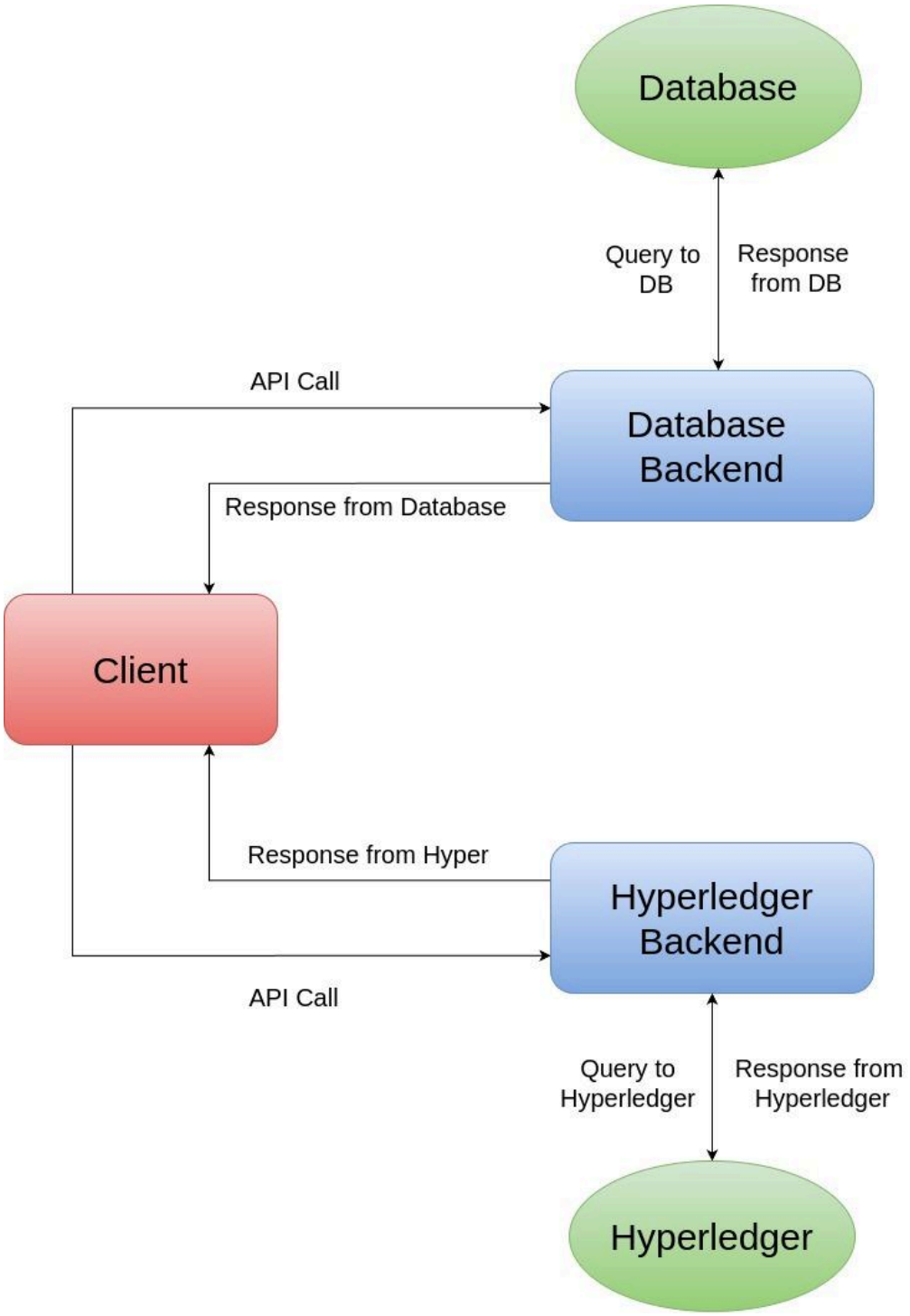
Course: CS731

Date: 07/04/2024

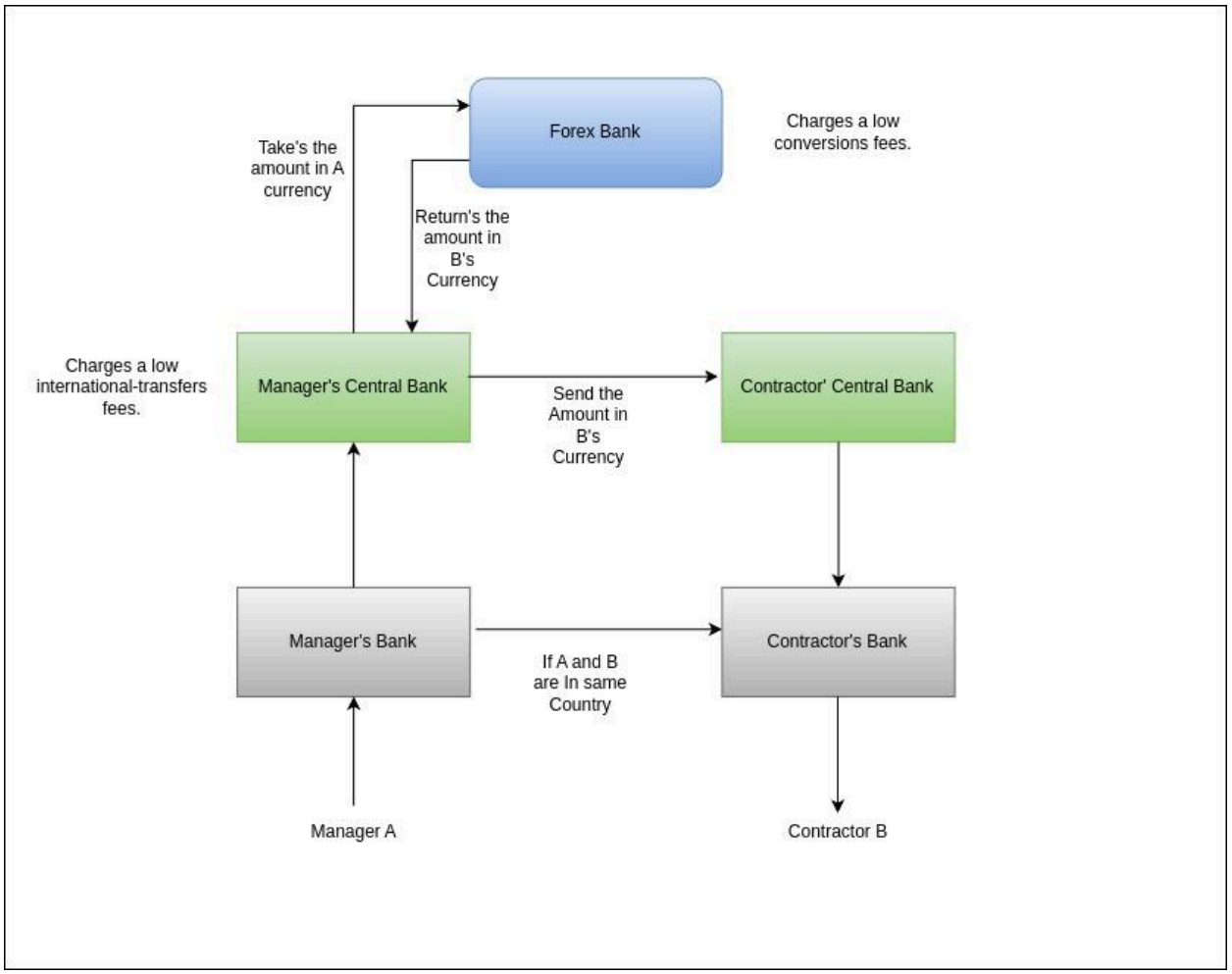
Index

1. Architecture diagrams:

System Diagram:



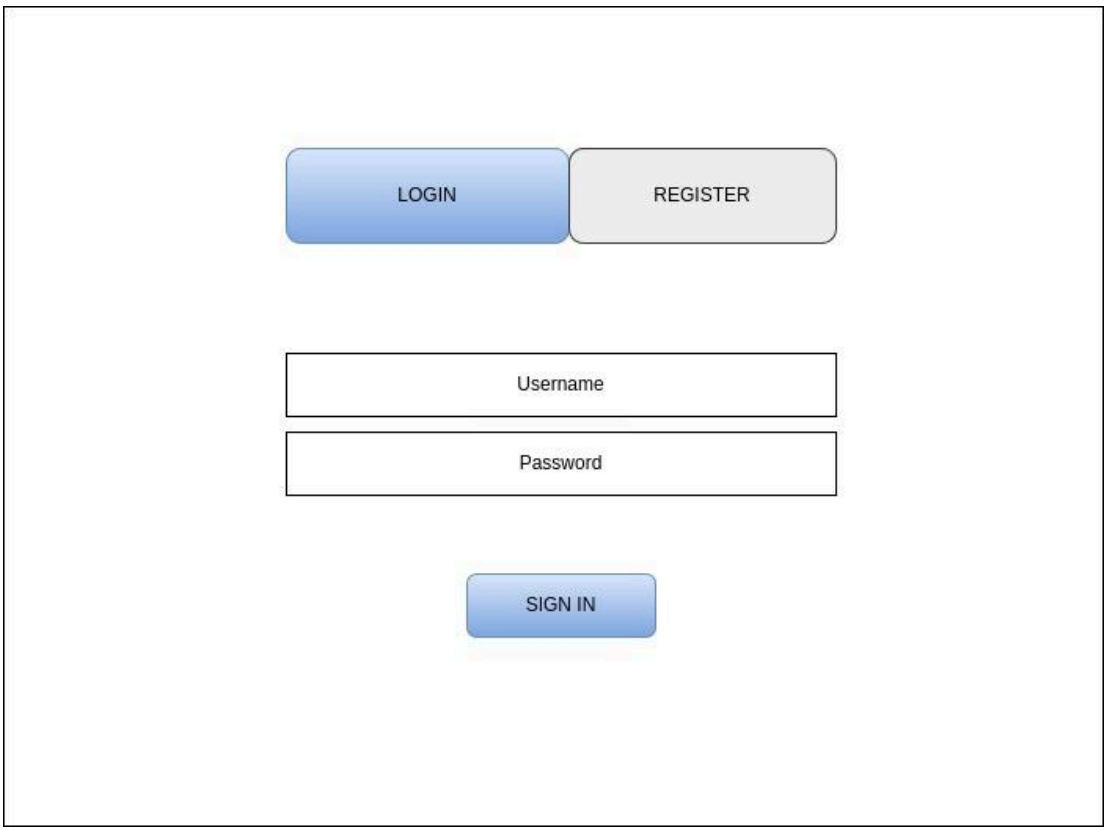
Payment Diagram:

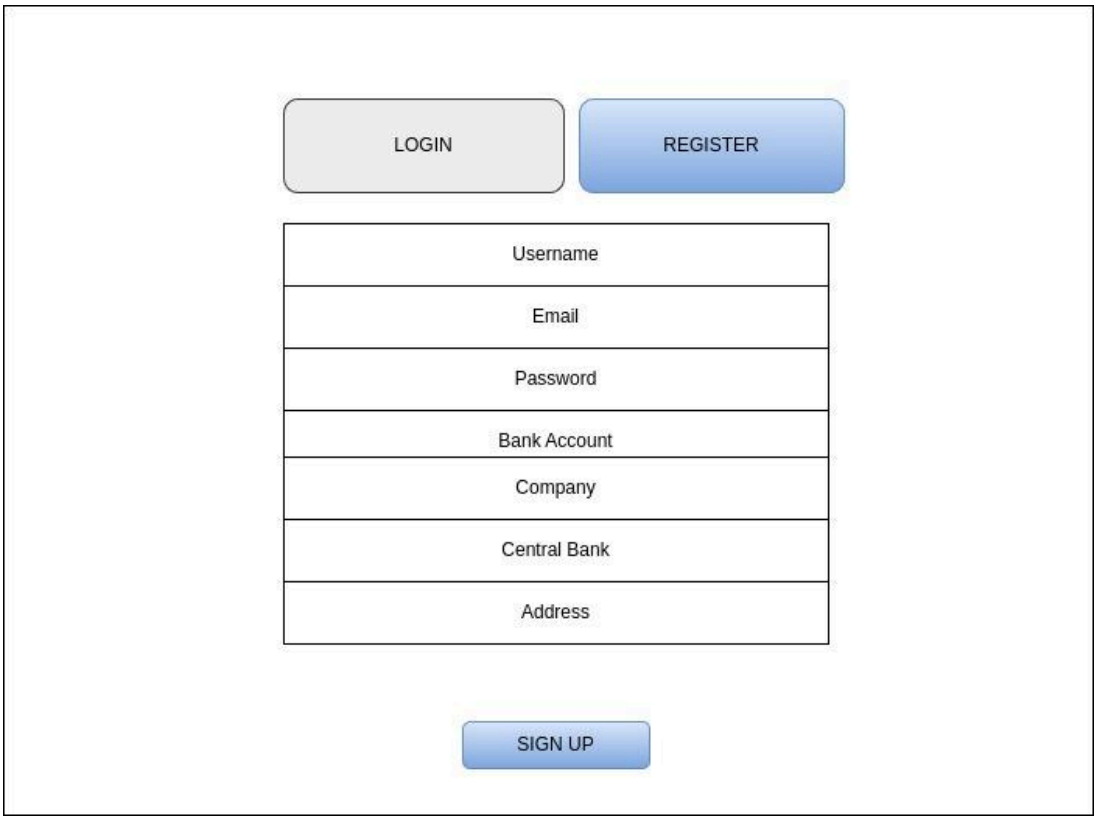


2. Wireframe diagram:


BlockPe
My Contracts
contract Requests
Pending contracts
Login/Logout
Requested Contracts
Contract 1
Accept
Reject
Contract 2
Accept
Revoke


BlockPe
My Contracts
contract Requests
Pending contracts
Login/Logout
Pending Contracts
Contract 1
Accept
Reject
Contract 2
Accept
Revoke


LOGIN
REGISTER
Username
Password
SIGN IN


LOGIN
REGISTER
Username
Email
Password
Bank Account
Company
Central Bank
Address
SIGN UP

3. Blockchain Assets:

The project involves managing different types of assets related to users, bank accounts, and contracts. Let's describe each asset:

UserAsset:

Represents a user's asset.
Contains the following fields:

- **Contracts**: An array of ContractAsset representing active contracts associated with the user.
- **Requests**: An array of ContractAsset representing contract requests made to the user (contractor) by a potential manager.
- **Pending**: An array of ContractAsset representing contracts pending to be activated, between the manager (user) and a contractor who has accepted the contract proposed by the user (manager)..
- **Username**: A unique identifier for the user (primary key).
- **BankAccountNo**: The bank account number associated with the user.
- **CentralBankID**: The ID of the central bank associated with the user.
- **Password**: Hash of the password of the user's account.

BankAccountAsset:

Represents a bank account asset.
Contains the following fields:

- **AccountNo**: A unique identifier for the bank account (primary key)..
- **CentralBank**: The central bank associated with the account.
- **Funds**: An integer representing the funds available in the account.
- **Owner**: The owner of the bank account.

ContractAsset:

Represents a contract asset.
Contains the following fields:

- **ContractID**: unique id of each contract
- **Manager**: The manager responsible for the contract.
- **Contractor**: The contractor involved in the contract.
- **Duration**: The duration of the contract.
- **Interval**: The interval at which payments are made for the contract.
- **RatePerInterval**: The rate of payment per interval.
- **RateCurrency**: The currency in which the manager states the rate to the contractor.
- **NatureOfWork**: Describes the nature of the work specified in the contract.
- **StartDate**: Start date of a contract
- **LastPaymentDate**: The payment date till which the funds have been redeemed.
- **ContractorAccount**: The bank account associated with the contractor for payment.
- **PaymentCurrency**: The currency the contractor would like to receive payment in.

4. Description of functionality:

The described functionality involves several steps in the interaction between users (managers and contractors), the backend database, and the Hyperledger backend for managing contracts. Here's a description of the functionality:

User Account Creation:
Users (either managers or contractors) can create their accounts at the backend.
During account creation, users provide necessary details such as username, bank account number, etc.

User Login:
After creating an account, users can log in to the system using their credentials.
Contract Creation by Manager:
A manager, upon logging in, can initiate the creation of a contract.
The manager enters basic details of the contract such as contractor, duration, interval, rate per interval, payment currency, and nature of work.
This contract creation process involves interaction with the Hyperledger backend.

Contract Proposal to Manager:
After the manager creates the contract, it is sent as a proposal to the specified contractor.
The proposal includes all the details entered by the manager.

Contract Details Entry by Contractor & Contract Acceptance/Rejection by Contractor:
Upon receiving the contract proposal, the contractor has the option to accept or reject it.
If the contractor rejects the contract, the process is terminated, and all related data is reverted.
If the contractor accepts the contract, the process proceeds to the next step.
If the contractor accepts the contract, they must provide additional details such as their contractor account and payment currency.
This information is necessary for processing payments during the contract period.

Final Acceptance by Manager:
After the contractor enters the required details and accepts the contract, it returns to the manager for final approval.
The manager reviews the contract details and can choose to accept or reject it.

Contract Finalisation:
If the manager accepts the contract, it is considered finalised.
The finalised contract is added to both the manager's and contractor's profiles, indicating their agreement to the terms and conditions outlined in the contract.
This functionality ensures a structured and secure process for creating, proposing, and finalising contracts between managers and contractors, leveraging both the backend database and the Hyperledger backend for efficient contract management and execution.

Payment details:
The system shall facilitate payment settlement between parties A (employer/manager) and B (contractor/employee) based on their geographical location.

If both A and B are located in the same country:
The system shall enable direct transfer of funds from A to B without involving additional intermediary banks.
Upon A's instruction, the system shall initiate the transfer of the agreed payment amount to B's designated bank account within the same country.
If A and B are located in different countries (cross-border payment):
The system shall route the payment through A's central bank for international transactions.
A's central bank shall charge a low international-transfer fee for facilitating the cross-border transaction.
The system shall then engage a forex bank to convert the payment currency from A's currency to B's currency at current market rates.
The forex bank shall apply low conversion fees for the currency exchange process.
Once the currency conversion is completed, A's central bank shall transfer the converted amount to B's central bank.
B's central bank shall receive the funds from A's central bank and credit the amount to B's designated bank account in the respective country.

Database:
The system shall store non-critical user details, such as email and address, in a separate MongoDB database for quicker retrieval of these details. This Database will be accessed through a separate server. Servers of backend and Hyperledger backend are different**.**

5. Tech stack:

1. **Languages**: Go, TypeScript, NodeJS
2. **Frontend**:
    - ReactJS
    - MUI
    - Bootstrap
    - Material design Bootstrap
    - React-router-dom
    - Axios
3. **Hyperledger Backend**:
    - grpc-js
    - fabric-gateway
    - express
    - typescript
    - cors
4. **Backend:**
    - Cors
    - Express
5. **Hyperledger**
6. **MongoDB**

6. Hyperledger Interactions:

Hyperledger Fabric smart contract (chaincode) is implemented using Go programming language. Hyperledger Fabric is a permissioned blockchain framework for developing enterprise-grade blockchain applications.
The smart contract defines various functionalities for managing assets and contract interactions between users (managers and contractors). Here's a detailed explanation of how Hyperledger Fabric interacts with the described functionality:

Smart Contract Definition:
The SmartContract struct defines the chaincode and embeds contractapi.Contract from the Fabric Contract API Go package. This struct provides functions for managing assets and contract interactions.

User Asset Management:
Functions like CreateUserAsset and GetUserAsset are provided to create and retrieve user assets respectively. These functions interact with the world state to store and retrieve user assets (such as username, name, bank account details, etc.).

Bank Account Asset Management:
Similar to user assets, functions like CreateBankAccountAsset and GetBankAccountAsset are provided to manage bank account assets. These functions store and retrieve bank account details in the world state.
Contract Asset Management:
The CreateContractAsset function is used to create a new contract asset and add it to the user's asset. It interacts with the world state to store contract details and updates the user's asset accordingly.

Contract Interaction:
The AcceptByContractor and AcceptByManager functions handle the interaction between contractors and managers during the contract lifecycle.
AcceptByContractor function allows the contractor to accept a contract proposal by filling in additional details such as contractor account and payment currency. It then moves the contract from the contractor's request array to the manager's pending array.
AcceptByManager function allows the manager to accept the finalised contract. It moves the contract from the manager's pending array to the contracts array for both the manager and contractor.

State Management:
Functions like UserAssetExists and BankAccountAssetExists are provided to check the existence of user and bank account assets in the world state.

Overall, the Hyperledger Fabric smart contract facilitates the management of assets (users, bank accounts, contracts) and contract interactions between users, ensuring secure and transparent execution of business logic on the blockchain platform.