Architecture & Reqirement Document

for

BlockPe

Version 1.0

Prepared by

Group: 1

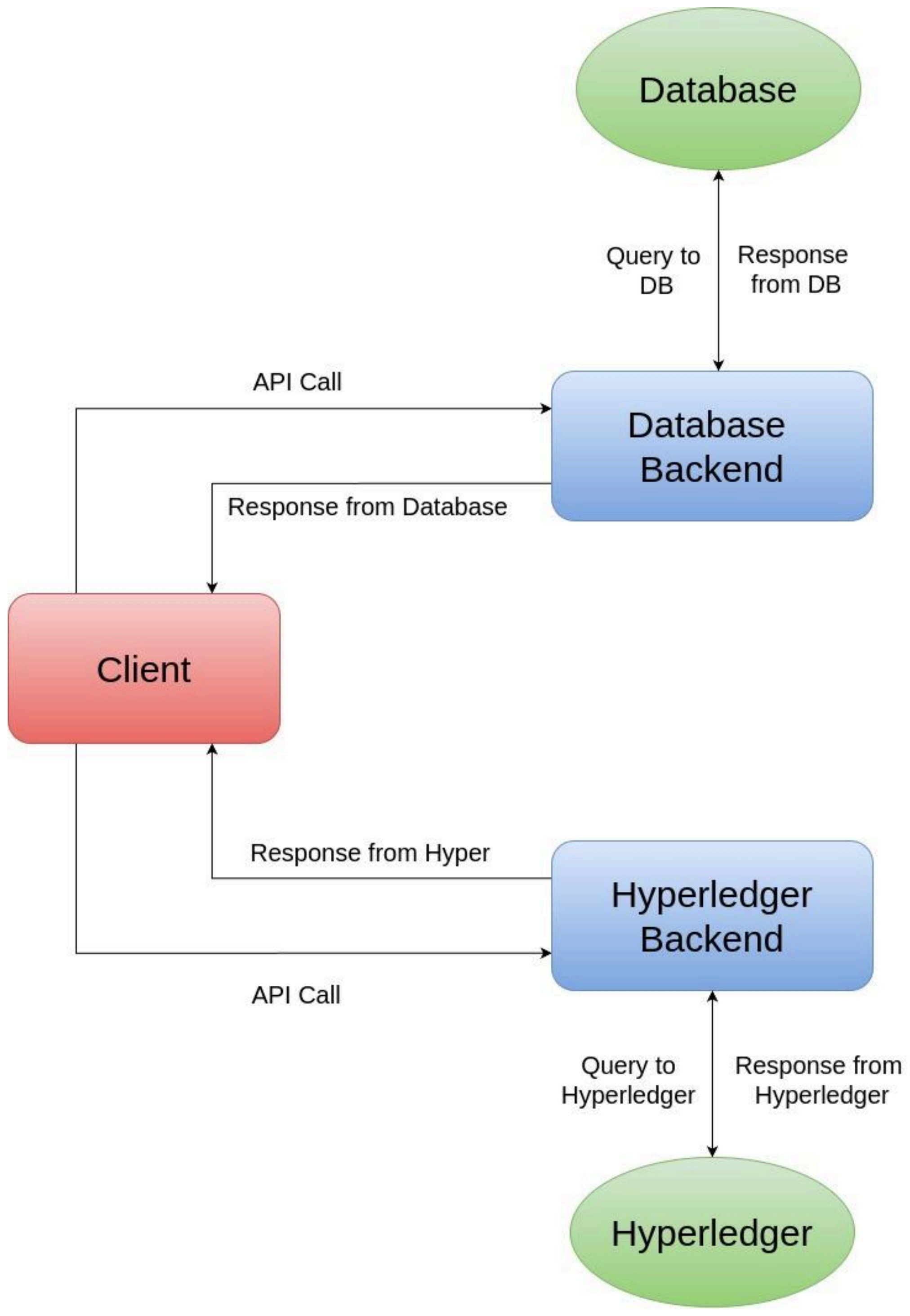
Chinmay Hiran Pillai	200298	chinmay20@iitk.ac.in
Raj Vardhan Singh	200760	rajvs20@iitk.ac.in

Course: CS731
Date: 07/04/2024

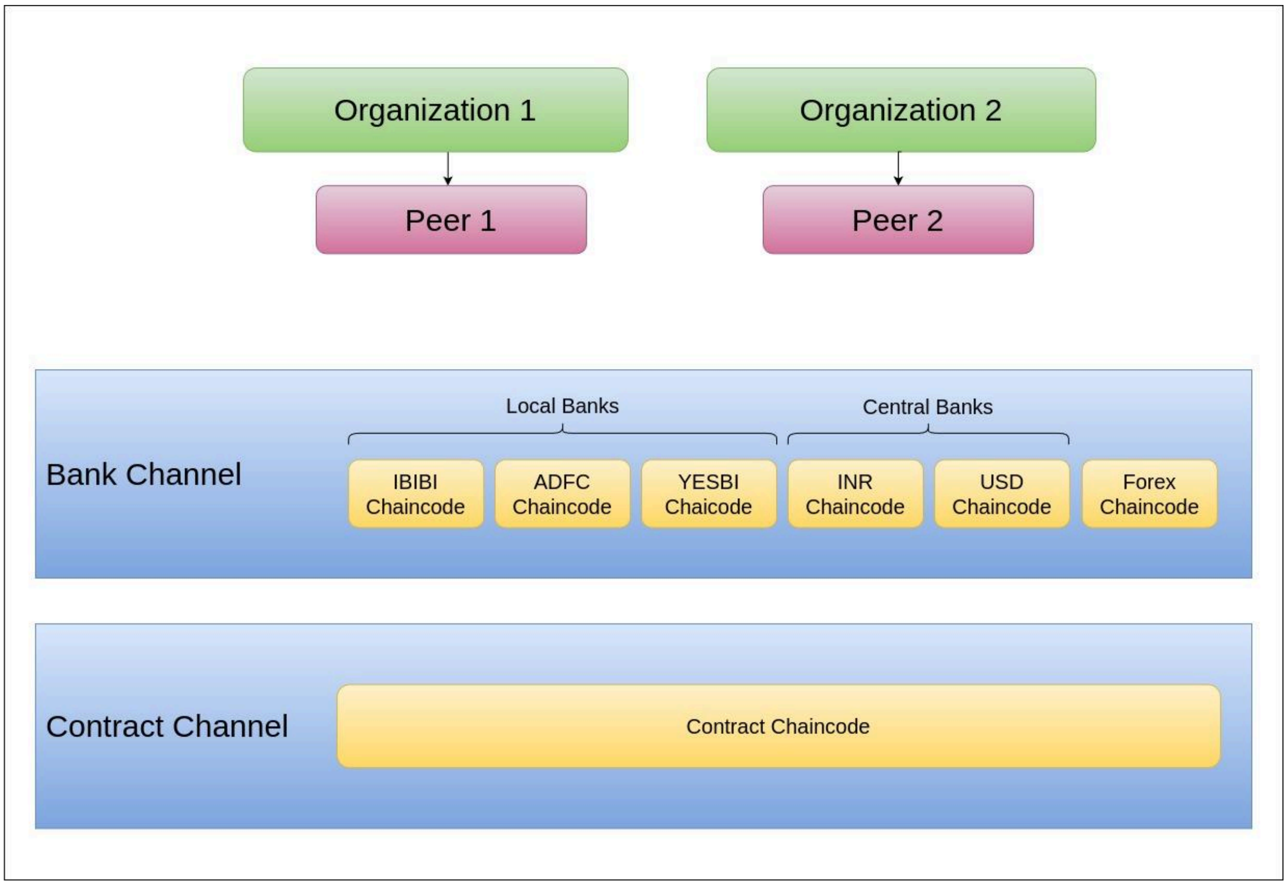
Index

1. Architecture diagrams:

System Diagram:



Hyperledger Diagram:



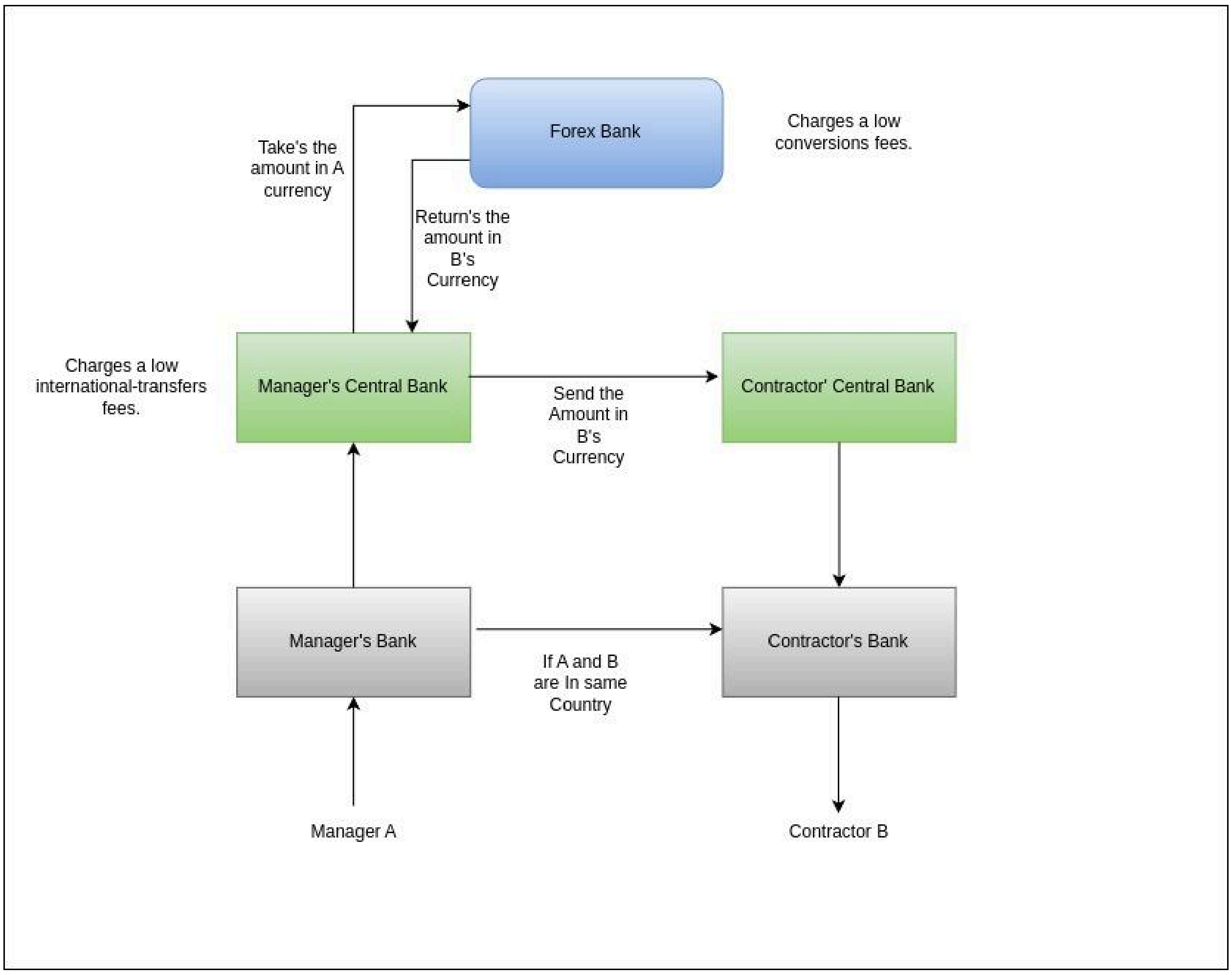
We have set up two channels within your Hyperledger Fabric network. Here's an overview of what each channel contains:

1. Contract Channel:
This channel contains the contract chaincode.
The contract chaincode manages contracts between managers and contractors, handling functionalities such as contract creation, acceptance, and management, as described in the previous explanation.

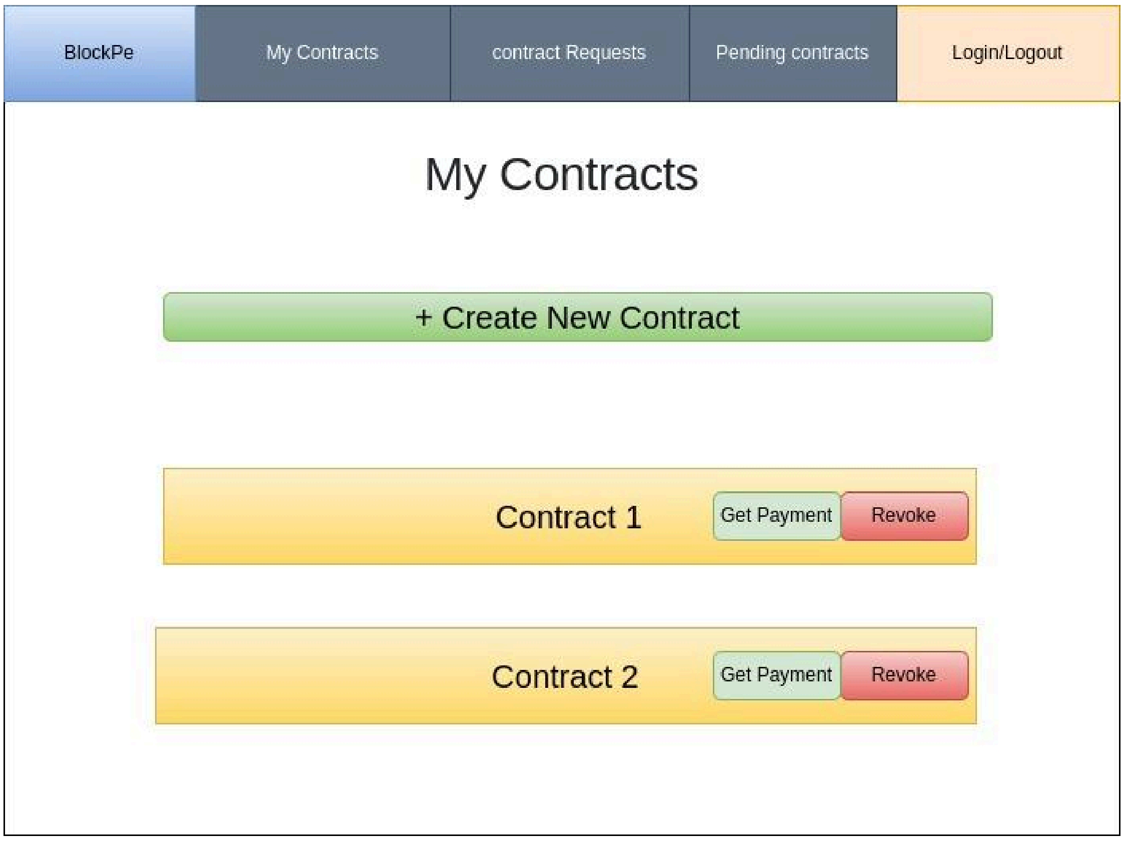
2. Bank Channel:
This channel contains the chaincode related to banking operations, including local and global banks, as well as the forex chaincode.
The banking chaincodes manage banking operations such as account management, fund transfers, and other financial transactions within the network.
The forex chaincode likely facilitates foreign exchange transactions, enabling conversion between different currencies.

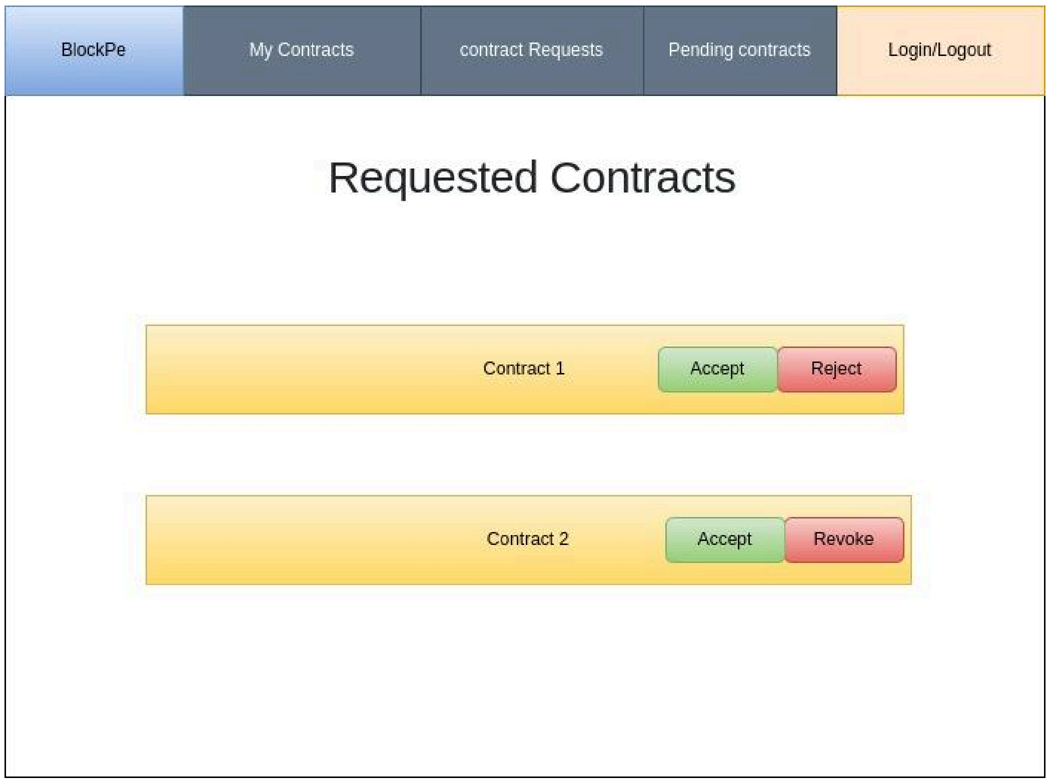
Each channel segregates the business logic and data, providing isolation and privacy for the participants involved. This architecture allows for different sets of chaincode to be deployed on separate channels, catering to specific business requirements while maintaining data confidentiality and integrity.

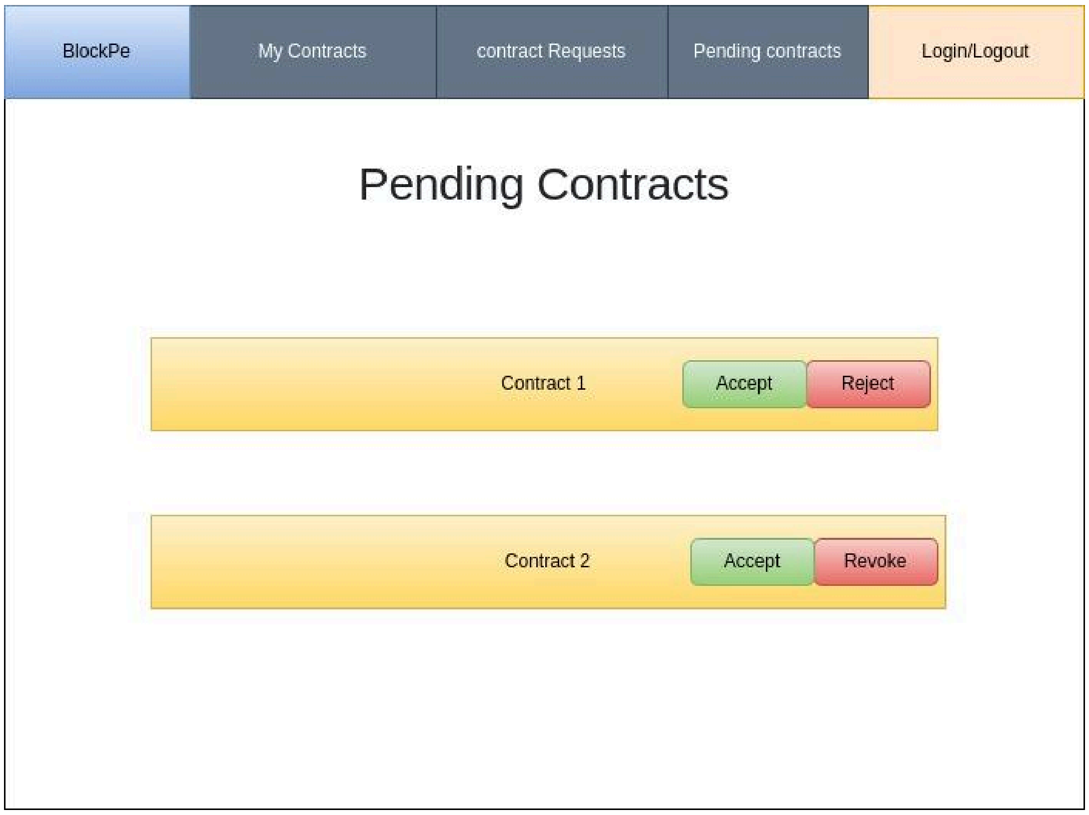
Payment Process Diagram:

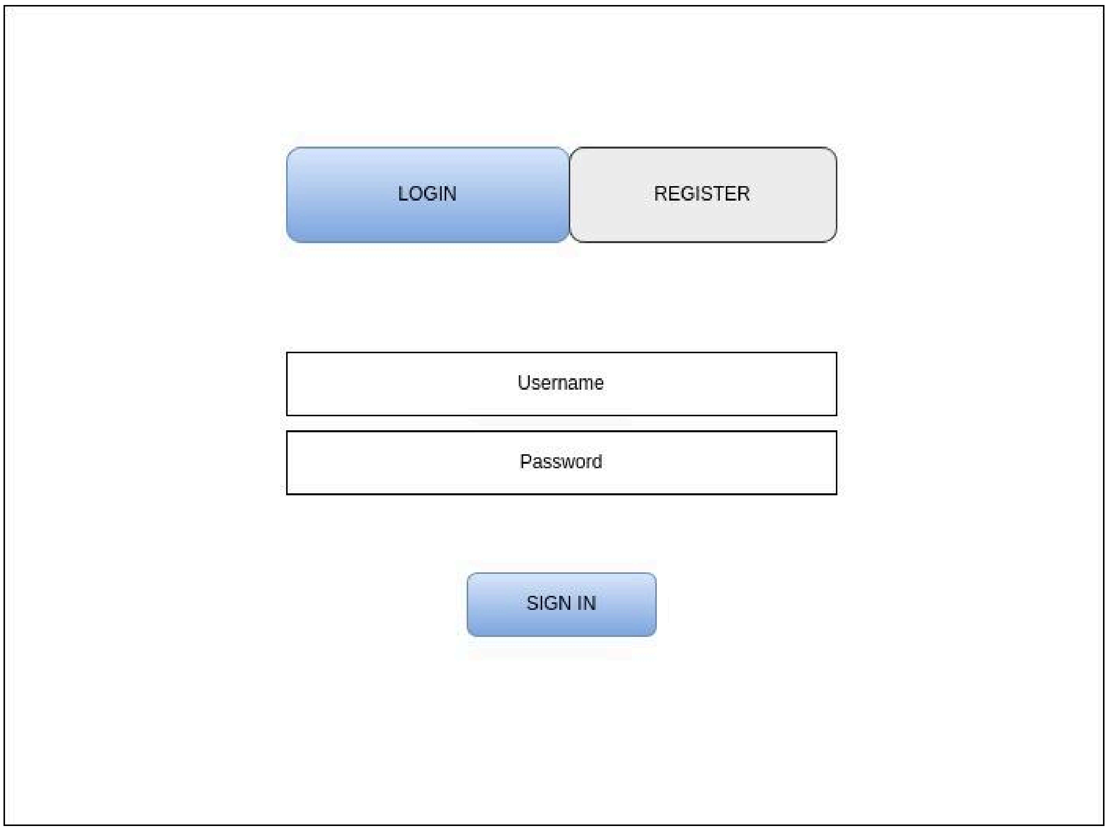

Forex Bank
Charges a low conversions fees.
Take's the amount in A currency
Return's the amount in B's Currency
Charges a low international-transfers fees.
Manager's Central Bank
Send the Amount in B's Currency
Contractor' Central Bank
Manager's Bank
If A and B are In same Country
Contractor's Bank
Manager A
Contractor B

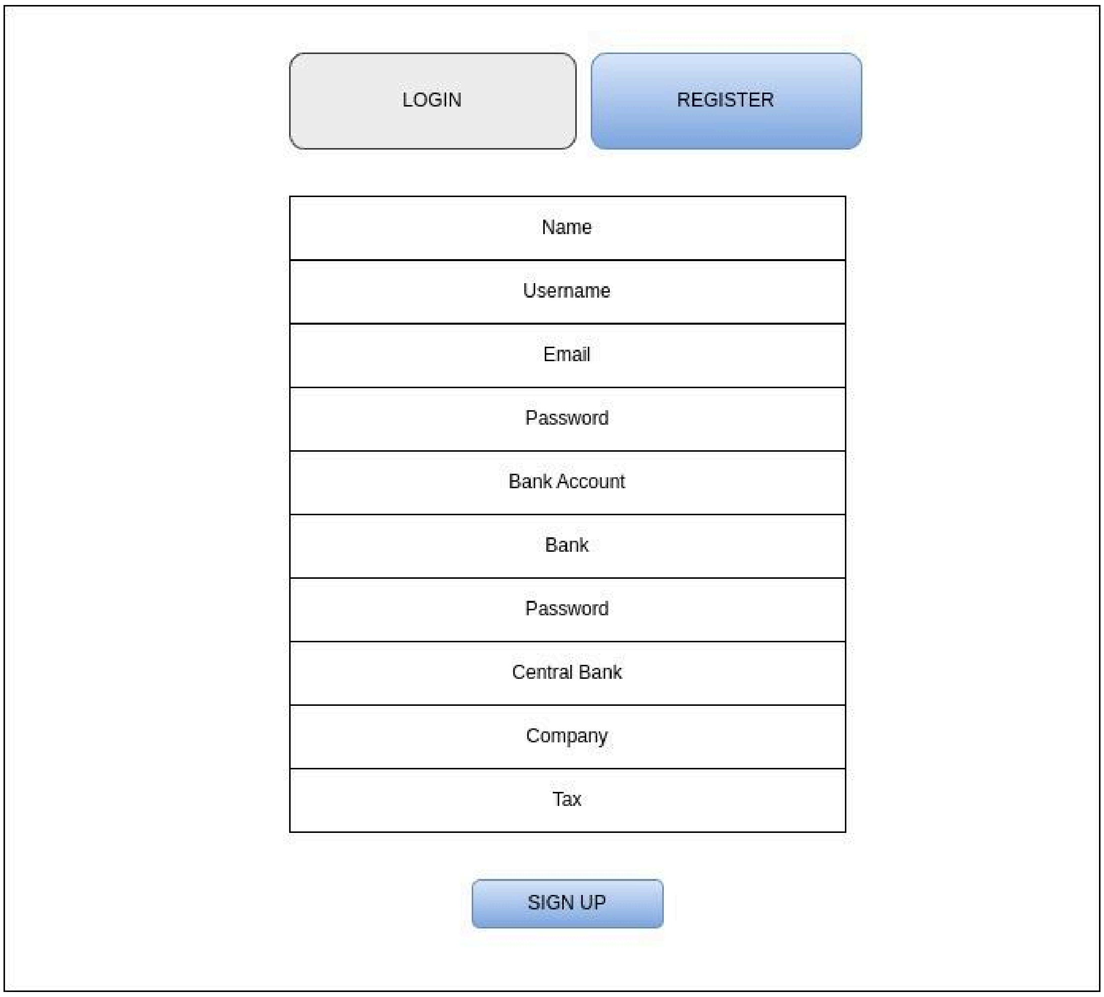
2. Wireframe diagram:




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
Name
Username
Email
Password
Bank Account
Bank
Password
Central Bank
Company
Tax
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. These contracts may include details such as contract ID, manager, contractor, duration, payment details, etc.
- **Requests**: An array of ContractAsset representing contract requests made to the user (contractor) by a potential manager. These requests contain details of the proposed contracts, which the user can review and either accept or reject.
- **Pending**: An array of ContractAsset representing contracts pending activation. These are contracts between the manager (user) and a contractor who has accepted the contract proposed by the manager. They are awaiting final confirmation or activation.
- **Username**: A unique identifier for the user, serving as the primary key. It uniquely identifies each user within the system.
- **Name**: The name of the user. It represents the user's full name or any other name associated with the account.
- **Password**: A hashed representation of the user's password. This field ensures security by storing passwords securely.
- **Bank**: The name or identifier of the bank associated with the user. It indicates the bank where the user holds their account.
- **BankAccountNo**: The bank account number associated with the user. It uniquely identifies the user's bank account within the specified bank.
- **CentralBankID**: The ID of the central bank associated with the user. It indicates the central bank that oversees or manages the user's banking activities, especially in the context of international transactions or regulatory compliance.
- **Company**: The company affiliation of the user, if applicable. It represents the organisation or company with which the user is associated.

BankAccountAsset:

Represents a bank account asset.
Contains the following fields:

- **AccountNo**: A string representing a unique identifier for the bank account. It serves as the primary key to uniquely identify each bank account within the system. The account number uniquely identifies the bank account.
- **CentralBank**: A string indicating the name or identifier of the central bank associated with the account. It specifies the central bank that oversees or manages the bank account.
- **Funds**: An integer representing the funds available in the bank account. It indicates the amount of money currently held in the bank account, measured in the currency of the central bank.
- **Owner**: A string representing the owner of the bank account. It specifies the individual or entity that owns or is authorised to use the bank account.
- **Tax**: An integer representing the tax amount applicable to the bank account. It specifies any taxes or fees associated with maintaining the bank account or conducting

transactions.

ContractAsset:

Represents a contract asset.
Contains the following fields:

- **ContractId**: An integer representing the unique identifier for the contract. It serves as the primary key to uniquely identify each contract within the system.
- **Manager**: A string indicating the username or identifier of the manager associated with the contract. The manager is typically the party responsible for initiating and overseeing the contract.
- **Contractor**: A string indicating the username or identifier of the contractor involved in the contract. The contractor is the party responsible for fulfilling the terms of the contract.
- **Duration**: An integer representing the duration of the contract, typically measured in days, weeks, months, or years. It specifies the length of time for which the contract is valid or active.
- **Interval**: An integer representing the interval at which payments are made for the contract. It defines the frequency of payment, such as weekly, monthly, or annually.
- **RatePerInterval**: An integer representing the rate of payment per interval. It specifies the amount of payment agreed upon for each payment period defined by the interval.
- **RateCurrency**: A string indicating the currency in which the payment rate is specified. It defines the currency denomination for the payment rate, such as USD, EUR, GBP, etc.
- **NatureOfWork**: A string describing the nature of the work specified in the contract. It provides details about the tasks, services, or activities to be performed under the contract.
- **StartDate**: A string representing the start date of the contract. It indicates the date on which the contract becomes effective or starts being enforced.
- **LastPaymentDate**: A string representing the date of the last payment made for the contract. It indicates the most recent date on which payment was made to fulfill the terms of the contract.
- **ManagerBank**: A string indicating the name or identifier of the bank associated with the manager. It specifies the bank where the manager holds their account.
- **ManagerBankAccountNo**: A string representing the bank account number associated with the manager. It uniquely identifies the manager's bank account within the specified bank.
- **ContractorAccount**: A string representing the bank account associated with the contractor for payment. It specifies the bank account to which payments are to be made for fulfilling the contract.
- **PaymentCurrency**: A string indicating the currency in which the contractor would like to receive payment. It defines the currency denomination for the payment amount.
- **ContractorBank**: A string indicating the name or identifier of the bank associated with the contractor. It specifies the bank where the contractor holds their account.

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.

AddFunds:

- Adds funds to a bank account asset.
- Retrieves the asset by its account number, adds the specified amount of funds (minus tax if applicable), and updates the asset in the world state.

RemoveFunds:

- Removes funds from a bank account asset.
- Retrieves the asset by its account number, checks if the funds are sufficient, subtracts the specified amount of funds, and updates the asset in the world state.

ForeignTransfer:

- Facilitates a foreign exchange transfer.
- Invokes the appropriate foreign exchange chaincode to perform the transfer based on the specified currencies, amount, destination bank, and account.

Pay:

- Handles payment transactions between accounts.
- If the currencies are the same, it either adds funds directly to the recipient's account (if it's within the same bank) or initiates a transfer to the recipient's account in another bank.
- If the currencies are different, it initiates a foreign exchange transfer using the ForeignTransfer function and then credits the recipient's account.

InvokeForex:

- Invokes the foreign exchange (Forex) chaincode to convert currency from one type to another.
- Constructs the arguments needed for the Forex chaincode invocation, including the source currency, target currency, and amount.

- Upon successful invocation, it parses the response payload, which contains the converted amount, and returns it.

Receive:
- Used to receive funds into a bank account from a central bank.
- Constructs the arguments needed for the central bank chaincode invocation to add funds to the specified bank account.
- Upon successful invocation, it verifies the status of the response and returns nil if successful. Otherwise, it returns an error.

PayCentralBnk:
- Facilitates a payment to a central bank in a different currency.
- Invokes the InvokeForex function to convert the currency to the desired type and calculates the amount to send after deducting international transfer fees.
- Constructs arguments for the central bank chaincode invocation to receive the converted funds.
- Upon successful invocation, it verifies the status of the response and returns nil if successful. Otherwise, it returns an error.

Forex Function:
- The main function provided by the Forex chaincode is called "Forex."
- It accepts parameters such as the source currency ("currencyFrom"), the target currency ("currencyTo"), and the amount to be exchanged ("amount").
- Within the function, a foreign exchange fee of 1% ("forexFees") is applied to the transaction.
- The fees are subtracted from the transaction amount to calculate the net amount to be exchanged.
- The function then calculates the exchanged amount based on the exchange rate between the specified currencies.
- If the currency pair is "USD" to "INR," the transaction amount is multiplied by the exchange rate (83) and the fees multiplier to convert USD to INR.
- Conversely, if the currency pair is "INR" to "USD," the transaction amount is divided by the exchange rate (83) and the fees multiplier to convert INR to USD.
- If the specified currency pair is not supported or invalid, the function returns an error.

Database:
The system shall store non-critical user details, such as email and address, in a separate SQLite3 database for quicker retrieval of these details. This Database will be accessed through a separate server. Servers of backend and Hyperledger backend will be different**.**

5. Tech stack:

1. **Languages**:
    - Go
    - TypeScript
    - Javascript
    - NodeJS
2. **Frontend**:
    - ReactJS
    - MUI
    - Bootstrap
    - Material design Bootstrap
    - React-router-dom
    - Axios
    - bcryptjs
3. **Hyperledger Backend**:
    - grpc-js
    - fabric-gateway
    - express
    - typescript
    - cors
4. **Backend:**
    - Express
    - Sqlite3
    - nodemon
    - Cors
5. **Hyperledger**

6. **SQLite3 Database**

6. API Endpoints:

List of all APIs, reads (GET) and writes (PUT/POST) are seperated out as different microservices.

Hyperledger Backend APIs:

POST /acceptByContractor
POST /acceptByManager
POST /register
POST /login
POST /createContractAsset
POST /createBankAccountAsset
POST /pay

GET /userAsset/:username
GET /bankAccountAsset/:bank/:accountNo
GET /contracts/:username
GET /requestedContracts/:username
GET /pendingContracts/:username
GET /invokeForex/:currencyFrom/:currencyTo/:amount
GET /calculateRedemptionAmount/:contractId/:manager/:contractor/:currentDate

PUT /addFunds
PUT /removeFunds
PUT /revoke
PUT /removeFromPendingOfManager
PUT /removeFromRequestedOfContractor

SQLite3 APIs:

GET /api/users
GET /api/users/:username

POST /api/users

DELETE /api/users/:username

7. 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:

Chaincodes:

1. IBIBI, ADFC and YESBI Bank Chaincodes:

- These chaincode represent local banks.
- They facilitate the management of bank account assets within their respective local banking systems.
- Functions include initialising the ledger with sample assets, creating new bank account assets, checking asset existence, retrieving assets, adding funds, removing funds, conducting foreign transfers, and handling payments.
- Foreign transfers are facilitated through interactions with central bank chaincodes, such as "USD" and "INR."

2. INR and USD Central Bank Chaincodes:

- These chaincodes represent central banks.
- They facilitate currency exchange and international fund transfers within the blockchain network.
- Functions include initializing the ledger, invoking currency exchange (InvokeForex), receiving funds from local banks (Receive), and processing payments from local banks (PayCentralBnk).
- They interact with local bank chaincodes, such as "IBIBI" , "ADFC" and “YESBI” to handle fund transfers and ensure compliance with regulatory requirements.

3. Forex Chaincode:

- The main function provided by the Forex chaincode is called "Forex."
- It accepts parameters such as the source currency ("currencyFrom"), the target currency ("currencyTo"), and the amount to be exchanged ("amount").
- Within the function, a foreign exchange fee of 1% ("forexFees") is applied to the transaction.
- The fees are subtracted from the transaction amount to calculate the net amount to be exchanged.
- The function then calculates the exchanged amount based on the exchange rate between the specified currencies.
- If the currency pair is "USD" to "INR," the transaction amount is multiplied by the exchange rate (83) and the fees multiplier to convert USD to INR.
- Conversely, if the currency pair is "INR" to "USD," the transaction amount is divided by the exchange rate (83) and the fees multiplier to convert INR to USD.
- If the specified currency pair is not supported or invalid, the function returns an error.

The combination of these chaincodes and the Hyperledger Fabric smart contract framework provides a comprehensive solution for managing financial assets and facilitating transactions within a blockchain network. Here's a summary of the key points:

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.

1. CreateBankAccountAsset:
This function creates a new bank account asset.
It checks if a bank account asset with the provided account number already exists. If it exists, it returns an error. Otherwise, it creates a new asset with the provided details and stores it in the world state.

2. BankAccountAssetExists:
This function checks if a bank account asset with the given account number exists in the world state.
It returns true if the asset exists, otherwise false.

3. GetBankAccountAsset:
This function retrieves a bank account asset by its account number.
If the asset exists, it returns the asset details. If it doesn't exist, it creates a new one with zero funds and returns it.

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.

Payment Management

1. AddFunds:
This function adds funds to a bank account asset.
It retrieves the asset by its account number, adds the specified amount of funds (minus tax if applicable), and updates the asset in the world state.
2. RemoveFunds:
This function removes funds from a bank account asset.
It retrieves the asset by its account number, checks if the funds are sufficient, subtracts the specified amount of funds, and updates the asset in the world state.
3. ForeignTransfer:

This function facilitates a foreign exchange transfer.
It invokes the appropriate foreign exchange chaincode to perform the transfer based on the specified currencies, amount, destination bank, and account.
4. Pay:
This function handles payment transactions between accounts.
If the currencies are the same, it either adds funds directly to the recipient's account (if it's within the same bank) or initiates a transfer to the recipient's account in another bank.
If the currencies are different, it initiates a foreign exchange transfer using the ForeignTransfer function and then credits the recipient's account.
5. InvokeForex:
This function invokes the foreign exchange (Forex) chaincode to convert currency from one type to another.
It constructs the arguments needed for the Forex chaincode invocation, including the source currency, target currency, and amount.
Upon successful invocation, it parses the response payload, which contains the converted amount, and returns it.
6. Receive:
This function is used to receive funds into a bank account from a central bank.
It constructs the arguments needed for the central bank chaincode invocation to add funds to the specified bank account.
Upon successful invocation, it verifies the status of the response and returns nil if successful. Otherwise, it returns an error.
7. PayCentralBnk:
This function facilitates a payment to a central bank in a different currency.
It first invokes the InvokeForex function to convert the currency to the desired type and calculates the amount to send after deducting international transfer fees.
Then, it constructs arguments for the central bank chaincode invocation to receive the converted funds. Upon successful invocation, it verifies the status of the response and returns nil if successful. Otherwise, it returns an error.

8. Functional Requirements:

1.1 User Account Creation:

- The system shall implement user account creation functionality for both employers/managers and employees/contractors.
- Employers/managers and employees/contractors shall be required to provide essential details such as company name, tax compliance information, and financial information during account creation.
- User Account Creation: Users (managers or contractors) shall be able to create their accounts by providing necessary details such as username, email, and bank account number. Upon creation, each user shall be assigned a unique identifier.
- User Authentication: The system shall support user authentication mechanisms to allow users to securely log in using their credentials.

1.2 Contract Agreement:

- The system shall allow employers/managers to create contracts specifying the contractor, duration of work, interval between payments, and money offered per interval of work.
- Employees/contractors shall have the ability to review and agree or reject the terms of the contract.
- In order to accept the contract, the contractors shall be required to provide their financial details, including preferred currency and banking information.
- Upon the contractor accepting the contract, the final contract shall again be sent back to the manager for approval.
- The manager should then be able to accept or reject the final contract.
- Contract Creation: Managers shall be able to initiate the creation of contracts by specifying details such as contractor, duration, payment terms, and nature of work. Contractors shall receive contract proposals and have the option to accept or reject them.
- Contract Finalisation: Once both the manager and contractor agree to the contract terms, the contract shall be finalised and added to both parties' profiles.

1.3 Payroll System:

- Payment calculations shall be performed by the system based on the terms outlined in the contract agreements.
- Payment disbursement shall be limited to once per interval for employees/contractors.
- Flexibility shall be provided for employees/contractors to withdraw payments on an as-needed basis.
- The system shall incorporate taxation functionality to calculate and deduct taxes from payments at source.
- Payment Transactions: The system shall facilitate secure payment transactions between users, including domestic and cross-border transfers. Payments shall be processed efficiently and accurately, with transaction details recorded for auditability.

1.4 Local Settlement:

- The system shall facilitate seamless transfer of funds between parties within the same country or currency zone.
- Transaction fees, if applicable, shall be implemented for inter-bank transfers.
- In cases where both parties are located within the same country, funds transfer shall be executed directly without currency conversion or cross-border transaction fees.

1.5 Cross-Border Settlement:

- The system shall support international transactions with currency conversion capabilities.
- Central banks and forex banks shall be utilised for accurate conversion support and low conversion fees.
- Secure and transparent transactions shall be recorded on the blockchain for non-repudiation.
- Different types of banks, including sponsor banks (central banks), member banks and forex banks shall be involved in the cross-border settlement process.
- Central banks shall handle cross-border payments, charging low international-transfer fees.
- Forex banks shall provide accurate currency conversion support and charge low conversion fees.

1.6 Off-Chain Database Storage:

- Non-critical user details, such as email and address, must be stored off-chain in a separate database.
- This Database must be accessed through a separate server.
- Servers of backend and Hyperledger should be different.
- Non-critical user details, such as email and address, shall be stored in a separate SQLite3 database for efficient retrieval. The database shall be accessed through a separate server to minimise latency and optimise performance.

9. Non-Functional Requirements:

2.1. Performance:

- The system shall efficiently handle a high volume of contract agreements, payroll calculations, and payment settlements.
- Database queries and transaction processing shall have high throughput.
- The system should optimise resource utilisation to ensure efficient use of computing resources and minimise costs.

2.2. Scalability:

- The system shall be designed to scale horizontally and vertically to accommodate a growing user base and transaction volumes.
- Scalability should be built into the system architecture to accommodate future growth in user base and transaction volume.

2.3. Usability Requirements:

- The user interface should be intuitive and user-friendly, requiring minimal training for users to perform tasks.
- Clear and concise documentation should be provided to guide users through system functionalities and processes.

2.4. Space Requirements:

- Sufficient storage capacity shall be ensured to accommodate increasing data volumes over time.
- Efficient data storage mechanisms should be implemented to optimise disk space utilisation and minimise storage costs.

2.5. Dependability Requirements:

- High availability should be maintained, with failover mechanisms to minimise downtime.
- System availability should exceed 99.99%, ensuring uninterrupted service for users.

2.6. Security Requirements:

- Data encryption should be implemented to protect sensitive information stored on the blockchain and in the database.
- A scalable hyperledger system shall be implemented to securely store contract details, financial information, and transaction records.
- Access controls should be enforced to prevent unauthorised access to system resources and data.
- The system shall ensure efficient and secure management of assets, users, bank accounts, and contracts.

2.7. Operational Requirements:

- The system must be built to run on the Linux operating system and all its different distributions.

2.8. Development Requirements:

- The development process should follow best practices for software engineering, including version control and code reviews.
- Documentation should be comprehensive and up-to-date, covering system architecture, APIs, and development guidelines.

2.9. Seamless Integration:

- The system shall seamlessly integrate local banking systems (IBIBI, ADFC and YESBI) with central banking systems (USD and INR) within the blockchain network.
- The integration shall ensure interoperability and smooth communication between different banking entities to facilitate efficient financial transactions.

2.10. Transparency and Auditability:

- The smart contract logic shall enforce business rules and logic transparently on the blockchain platform.
- All transactions and contract interactions shall be recorded on the blockchain ledger, providing a transparent and immutable audit trail for regulatory compliance and accountability.