Requirements 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. 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.

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 to 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.

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.

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.

2. 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 low latency and high throughput.

2.2. Storage:

- A scalable database system shall be implemented to securely store contract details, financial information, and transaction records.
- Sufficient storage capacity shall be ensured to accommodate increasing data volumes over time.

2.3. Latency:

- The system shall minimise latency in contract agreement processing, payroll calculations, and payment settlements.

2.4. Scalability:

- The system shall be designed to scale horizontally and vertically to accommodate a growing user base and transaction volumes.

3. Non-Functional Requirements Achievement using Technologies:

Languages: Go, TypeScript, NodeJS

- **Go**: Known for its efficiency and concurrency features, Go ensures high performance by providing a lightweight, compiled language that can efficiently handle concurrent tasks. Its built-in garbage collection and goroutines facilitate low latency by managing memory efficiently and enabling concurrent operations. Go's static typing also helps in catching errors at compile time, reducing runtime issues.
- **TypeScript**: TypeScript is a statically typed superset of JavaScript that compiles down to plain JavaScript. Its static typing helps catch errors early in the development process, reducing runtime errors and enhancing performance. TypeScript's compiler optimises the code, resulting in efficient execution. TypeScript also allows developers to use modern JavaScript features while ensuring compatibility with older browsers, thus contributing to moderate storage overhead.
- **NodeJS**: Node.js is known for its event-driven architecture and non-blocking I/O, which contribute to low latency by allowing applications to handle multiple concurrent requests efficiently. Its package manager, npm, offers a vast ecosystem of libraries and modules, allowing developers to choose lightweight and optimised solutions. However, Node.js might require careful resource management to maintain high performance and low latency under heavy loads.

Frontend: ReactJS, MUI, Bootstrap, Material Design Bootstrap, React-router-dom, Axios

- **ReactJS**: React's virtual DOM and reconciliation algorithm optimise rendering performance by minimising actual DOM manipulations. This results in faster updates and a more responsive user interface, contributing to low latency. React also encourages component-based architecture, promoting code reusability and maintainability, which can indirectly improve performance.
- **MUI** (Material-UI): MUI is a React UI framework based on Google's Material Design principles. It provides pre-designed components and theming capabilities, allowing developers to create consistent and visually appealing interfaces efficiently. MUI's components are optimised for performance, ensuring smooth interactions and low latency.
- **Bootstrap and Material Design Bootstrap:** Both Bootstrap and Material Design Bootstrap offer pre-styled UI components and layouts, reducing the need for custom CSS and speeding up frontend development. These frameworks are designed with performance in mind and often include optimizations such as minimal CSS footprint and lazy loading, contributing to low latency.
- **React-router-dom**: React Router enables client-side routing in React applications, allowing for fast navigation between different views without full-page reloads. This helps maintain a smooth user experience and contributes to low latency by reducing the time required to fetch and render new pages.

- **Axios**: Axios is a popular HTTP client for making asynchronous requests in JavaScript applications. It offers features like request and response interception, automatic JSON data transformation, and support for promises, making it efficient and easy to use. Axios also provides options for request caching and concurrency control, which can further improve performance and reduce latency.

Hyperledger Backend: grpc-js, fabric-gateway, Express, TypeScript, Cors

- **grpc-js**: gRPC is a high-performance RPC (Remote Procedure Call) framework developed by Google. It uses Protocol Buffers for efficient serialisation and supports streaming and bi-directional communication, making it suitable for building low-latency, high-throughput services. grpc-js is the Node.js implementation of gRPC, providing asynchronous, non-blocking communication, which helps maintain low latency.
- **fabric-gateway**: Hyperledger Fabric Gateway SDK provides a higher-level interface for interacting with Hyperledger Fabric networks. It abstracts away the complexities of the underlying protocol and provides optimised methods for querying and updating the ledger. This abstraction layer helps ensure high performance and low latency by handling protocol-level optimizations internally.
- **Express**: Express is a minimal and flexible Node.js web application framework that provides a robust set of features for building APIs and web applications. It is known for its simplicity and performance, allowing developers to create lightweight and efficient backend services. Express middleware, such as compression and caching, can further improve performance and reduce latency.
- **TypeScript**: As mentioned earlier, TypeScript's static typing and compilation to JavaScript help catch errors early and optimise code for performance. Its support for modern JavaScript features and ES6+ syntax ensures efficient execution while maintaining code readability and maintainability.
- **Cors** (Cross-Origin Resource Sharing): Cors is a mechanism that allows web servers to specify which origins are permitted to access the resources on a server. Enabling Cors appropriately can prevent unnecessary latency by allowing cross-origin requests when needed, while still maintaining security. By configuring Cors policies correctly, developers can strike a balance between performance and security.

Backend: Cors, Express

- **Cors**: Cross-Origin Resource Sharing (CORS) is a mechanism that allows resources on a web page to be requested from another domain outside the domain from which the resource originated. Properly configuring CORS policies helps prevent unnecessary latency by allowing cross-origin requests when necessary, while still maintaining security.
- **Express**: As discussed earlier, Express is a minimalist web framework for Node.js. It's known for its simplicity and performance, making it an excellent choice for building backend services. Express provides features like routing, middleware support, and HTTP utilities, allowing developers to create efficient and scalable APIs.

By leveraging middleware like compression and caching, developers can further optimise performance and reduce latency.

Hyperledger

- Hyperledger is a blockchain platform that aims to provide enterprise-grade, permissioned blockchain solutions. Hyperledger Fabric, one of the frameworks under the Hyperledger umbrella, is designed for developing distributed ledger applications. It ensures high performance and low latency through various features:
- **Permissioned network**: Hyperledger Fabric uses a permissioned network model, where only known participants can join the network and transact. This reduces the overhead of reaching consensus among a large number of nodes, improving performance and latency.
- **Modular architecture**: Hyperledger Fabric's modular architecture allows for customization of consensus algorithms, membership services, and smart contract execution environments. This flexibility enables developers to optimise the network for specific use cases, ensuring high performance and low latency.
- **Efficient consensus mechanisms**: Hyperledger Fabric supports pluggable consensus mechanisms, such as Practical Byzantine Fault Tolerance (PBFT) and Raft, which are designed for high throughput and low latency. These consensus algorithms ensure that transactions are processed quickly and efficiently, contributing to overall network performance.

MongoDB

- MongoDB is a NoSQL database known for its scalability and performance. It ensures high performance, low latency, and moderate storage overhead through the following mechanisms:
- **Indexes**: MongoDB allows the creation of indexes on fields to improve query performance. Indexes help in quickly locating and retrieving documents, reducing query latency.
- **Sharding**: MongoDB supports horizontal scaling through sharding, which distributes data across multiple servers. Sharding improves read and write performance by distributing the load evenly across servers, reducing latency.
- **Document-oriented storage**: MongoDB stores data in flexible, JSON-like documents, which makes it efficient for querying and retrieval. This document-oriented storage model reduces the need for complex joins and improves query performance, contributing to low latency.
- **Compression and WiredTiger storage engine**: MongoDB's WiredTiger storage engine supports compression, which reduces storage overhead by compressing data on disk. Compression also improves read and write performance by reducing the amount of data transferred between disk and memory.