UTH UNIVERSITY OF TRANSPORT HOCHIMINH CITY

University of Transport Ho Chi Minh City

Information Technology Department

Human Resource Management Software

Software Design Document

Author: Cao Nguyen Tri Ngoc

Version: 1.0
Last Modified: 3/6/2025

Document Number: <document's configuration item control number>

Contract Number: <current contract number of company maintaining document>

Preface

In today's era of rapid technological advancement, the application of software solutions in human resource management not only helps businesses optimize workflows but also enhances efficiency and transparency within the organization. The human resource management system we have designed aims to provide a comprehensive solution, supporting businesses in managing employee information, attendance tracking, performance evaluation, and various reward and allowance policies effectively. Furthermore, the system also offers a built-in chat feature and task assignment functionality, eliminating the need for third-party software and addressing part of the economic challenges for businesses.

The purpose of this document is to provide a detailed description of the system’s design, including its overall architecture, main components, and operational mechanisms. This document serves as a reference for the development team, ensuring that the software development process follows the intended direction while also facilitating system maintenance and future expansion.

This system is developed using blockchain technology to ensure transparency and data security in salary management and related transactions. With a distributed model, the system can maintain data integrity, minimize fraud risks, and enhance trust among stakeholders.

We hope that this document will be beneficial to all parties involved in the system's development and serve as a foundation for further improvements and expansions in the future.

Software Development Team

REVISION HISTORY

Version	Date	Organization/Point of Contact	Description of Changes
1.0	06/03/2025	caonguyentringoc@gmail.com	Baseline Version

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose

In today's era of rapid technological advancement, there is an increasing demand for simplifying and automating tasks, including human resource management.

For businesses, the adoption of software and technology for HR management has been in place for several years. However, with the rapid growth of the Internet and artificial intelligence, protecting sensitive information in an online environment has become more critical than ever, posing significant security challenges for enterprises.

To address these concerns, we provide a human resource management system with robust security capabilities powered by blockchain technology. Additionally, our system inherits and enhances various features from traditional HR management solutions, effectively tackling challenges that businesses currently face.

Along with the software, we also provide comprehensive documentation detailing the HRM (Human Resource Management) system. This includes system architecture, database structure, system components, and implementation plans.

1.2. Scope

1.2.1. System Overview

This Human Resource Management system is a personnel management platform that ensures robust security for sensitive employee information through blockchain technology. Additionally, the system provides all essential functionalities of a traditional HR management system while optimizing and enhancing various features to meet the specific needs of businesses.

1.2.2. Core Functionalities

Function code	Function name	Description
FUNC-001	Authentication	Authenticate user with JWT
FUNC-002	Authorization	Restrict user access through roles and permissions management
FUNC-003	Project Function	Allows operations and management of company projects (including progress, plans, tasks, etc.)
FUNC-004	Task Function	Enables leaders or managers to assign tasks to employees and oversee their progress.
FUNC-005	Relation Function	Employees can send feedback to the company on any issue, with complete anonymity.

FUNC-006	Performance Evaluation Function	Calculate KPIs Manage supervisor feedback.
FUNC-007	Personnel Function	Calculate salaries, bonuses, manage contracts, taxes, etc. Manage information of personnel, contracts, taxes
FUNC-008	Report Function	Report on human resources situation, personnel changes, KPI, etc. in the month/quarter/year.

1.2.3. Limitations

The blockchain does not store all data but only retains employees' private information and smart contracts with KPIs.

The software is still in the testing phase, so its performance has not been fully optimized. However, it is fully capable of serving a small to medium-sized business.

The software does not integrate with any third parties.

1.2.4. External System Dependencies

External system name	Integrated description
Blockchain Ethereum	Store employee data and transactions related to employee rewards.
….	….

In the future, software needs to integrate with company systems to achieve high efficiency, such as timekeeping systems, accounting systems, etc.

1.3. Definitions, Acronyms and Abbreviations

Delete: All deletion methods mentioned in the documentation are soft deletes (only change the value of the field in database, no actual record deletion in the database).
IU: Interface User – an interface defined in the User Module.
UI/UX: User Interface/User Experience.
Collection: Similar to a table in a relational database, this is a collection of records in MongoDB.
Report Module: The reports generated in the Report Module group employees by their department.
CRUD: Create, Read, Update, and Delete – the 4 basic operations on data, implemented in most modules.

DTO: Data Transfer Object – an intermediary object used to transfer data between client and server.
JSON: JavaScript Object Notation – a data format for exchanging data between client and server.
JWT: JSON Web Token – an encoded token string used for user authentication.
IV: Initialization Vector – a random value used in encryption algorithms.

1.4. References

1.4.1. Blockchain in HR: A Guide for HR Professionals: https://www.testgorilla.com/blog/blockchain-in-hr/

1.4.2. Blockchain Applications in HR: https://www.linkedin.com/pulse/understanding-hr-blockchain-technology-aditi-gupta-vg9dc/

1.4.3. Blockchain-based human resource management practices for mitigating skills and competencies gap in workforce: https://journals.sagepub.com/doi/full/10.1177/1847979020966400

1.4.4. How blockchain technology could impact HR and the world of work: https://www.pwc.ch/en/insights/hr/how-blockchain-can-impact-hr-and-the-world-of-work.html

1.4.5. 7 way blockchain revolutionizes HRM: https://www.pace.edu.vn/shrm/knowledge-center/7-ways-blockchain-revolutionizes-hr-management

1.4.6. Blockchain implementation toolkit for HR: a step-by-step guide: https://www.shrm.org/in/topics-tools/tools/toolkits/blockchain-implementation-toolkit-for-hr--a-step-by-step-guide

1.4.7. Details on how to implement blockchain into nestjs project: https://claude.ai/share/03c50a05-5354-4aa5-bf3e-bff3209e05d5

2. SYSTEM ARCHITECTURE DESIGN

2.1. Architecture Overview

The system is built based on a Modular Architecture, incorporating elements of Microservices Architecture, using Nest.JS—a modern Node.js framework with full TypeScript support. This architecture enables the system to be developed flexibly, making it easy to maintain and scale.

2.1.1. Design Principles

The system adheres to core design principles:

- **SOLID Principles**: Especially the **Single Responsibility Principle** (SRP).
- **Domain-Driven Design (DDD)**: Organizing code based on business domains.
- **Dependency Injection (DI)**: Reducing dependencies between components.
- **Layered Architecture**: Clearly separating **Controller, Service, and Repository** layers.

2.1.2. Module Structure

The system is divided into multiple specialized modules, with each module responsible for a specific function.

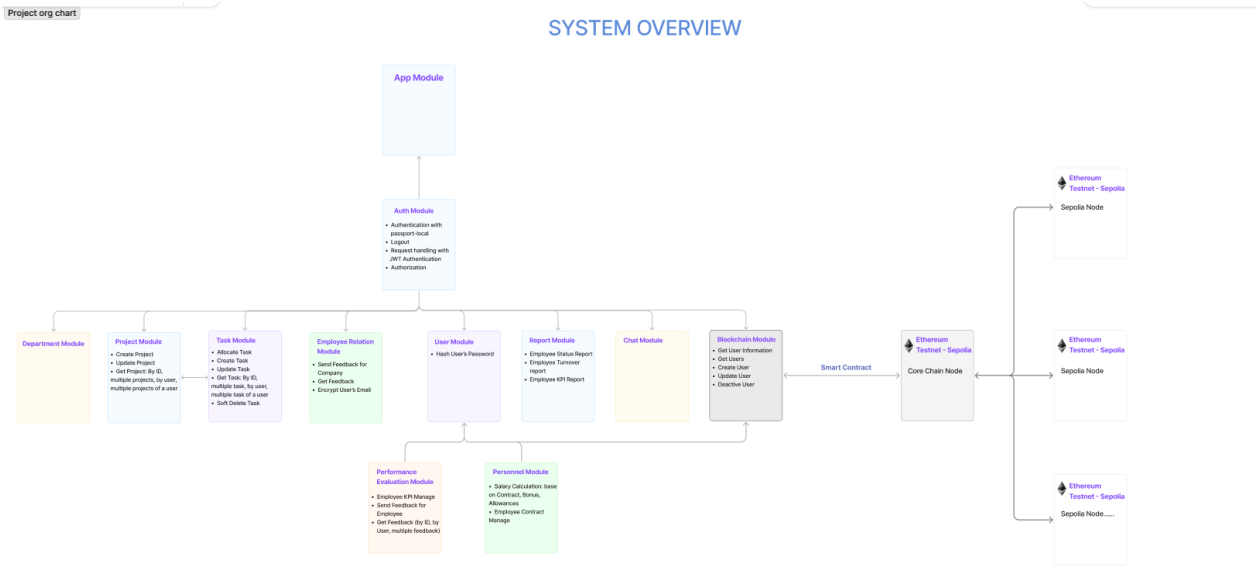


Figure 2.1. Overview diagram of the System

2.2. System Component

- Backend Framework: NestJS (Node.js + TypeScript)
- Blockchain: Truffle, Ganache, Ethereum Sepolia testnet, Web3.js/Ethers.js, Chainstack
- Smart Contract: Solidity
- Frontend Framework: NextJS
- Mobile Framework: React Native (NodeJS Expo + TypeScript)
- Real-time: Socket.io
- WebSocket: Socket.io
- Database: MongoDB
- API Documentation: Swagger/OpenAPI, Compodoc
- Testing: Jest, Supertest

2.3. Data Flow Diagram

2.3.1. Authentication

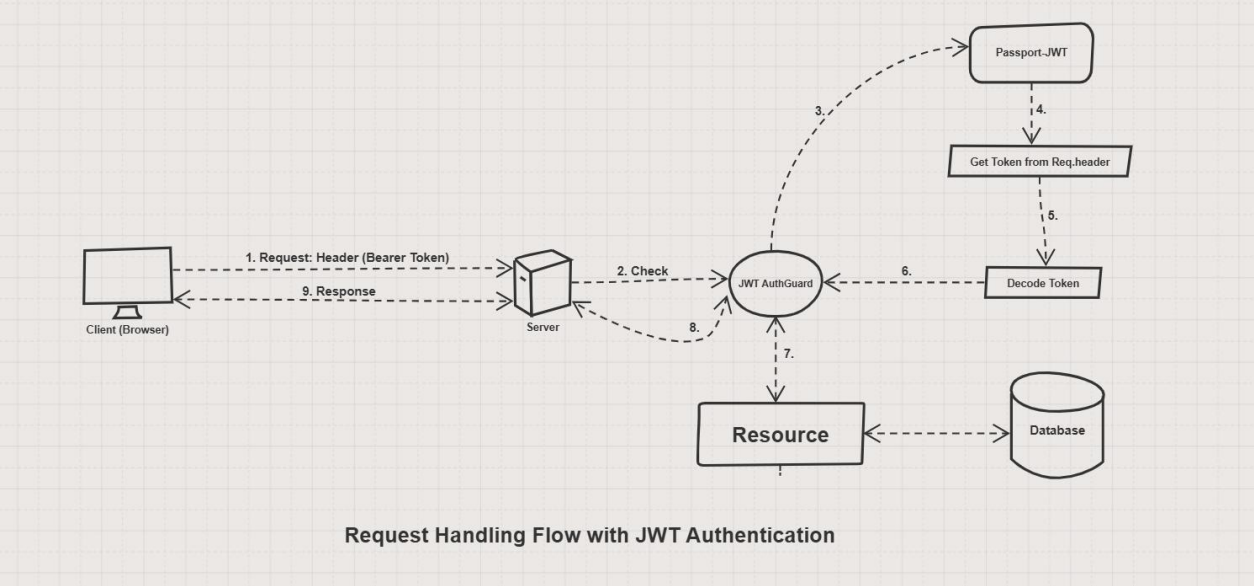


Figure 2.2. Request handling Flow with JWT Authentication

2.3.2. Login

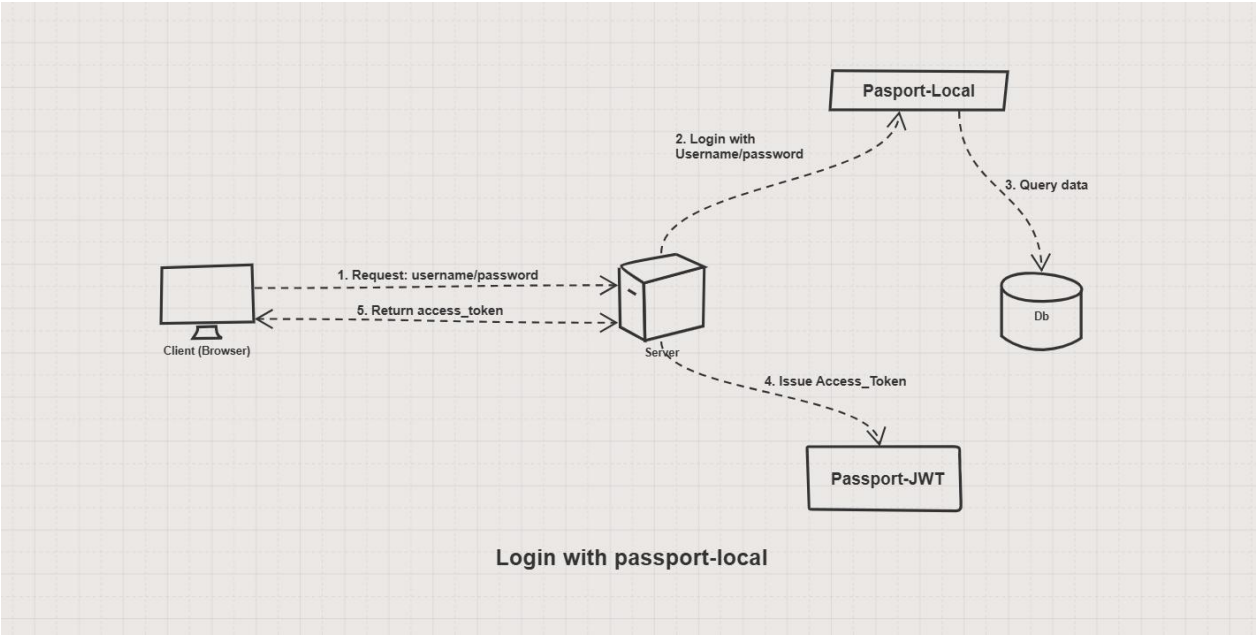


Figure 2.2. Login with passport-local

2.3.3. Authorization

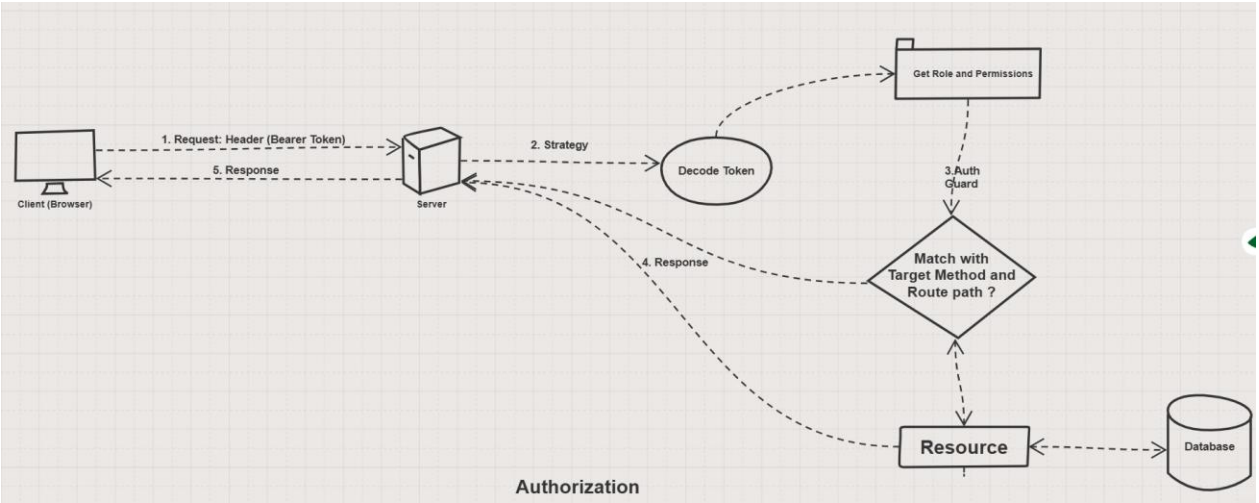


Figure 2.3. Authorization

2.3.4. Department Management

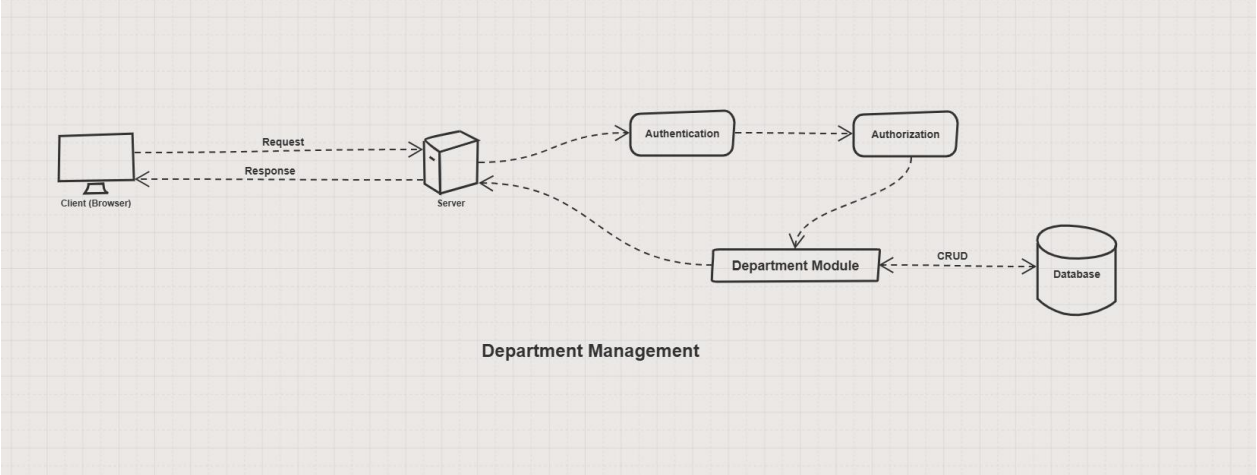


Figure 2.4. Department Management

2.3.5. Position Management

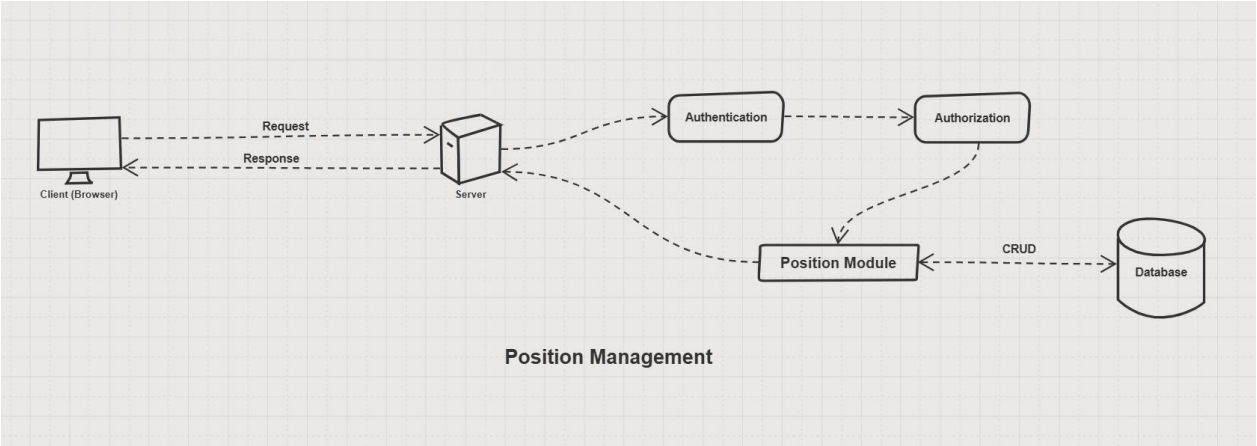


Figure 2.5. Position Management

2.3.6. Relation Employee

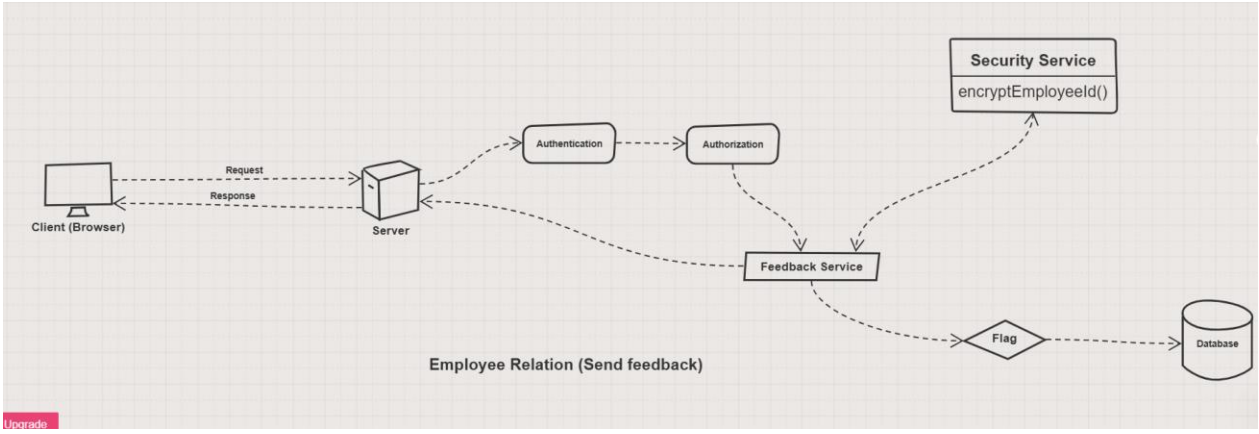


Figure 2.6. Send Feedback to Company

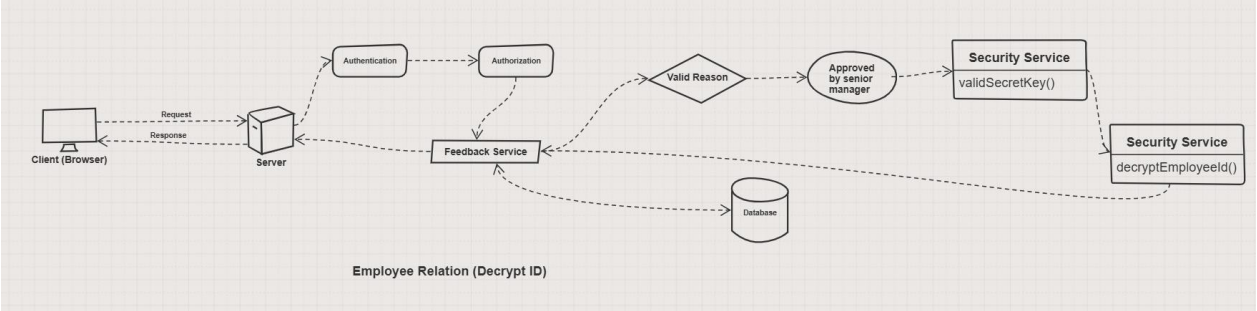


Figure 2.7. Decrypt Employee ID

2.3.7. Blockchain Module

Data flow diagram for creating a new user:

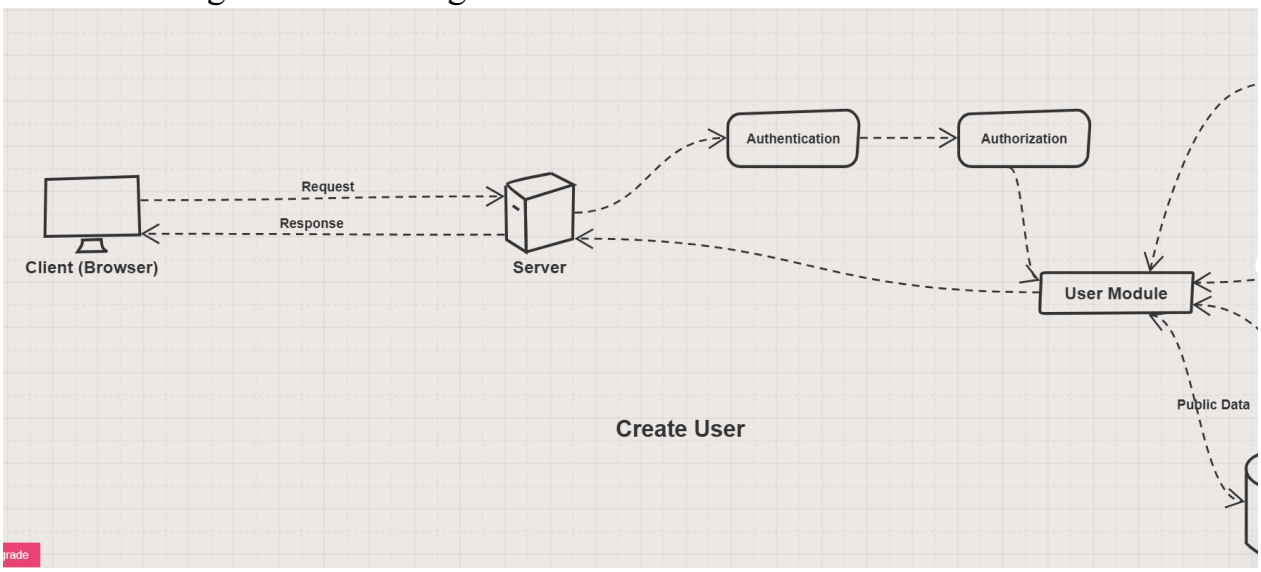


Figure 2.8. a) Create a new User

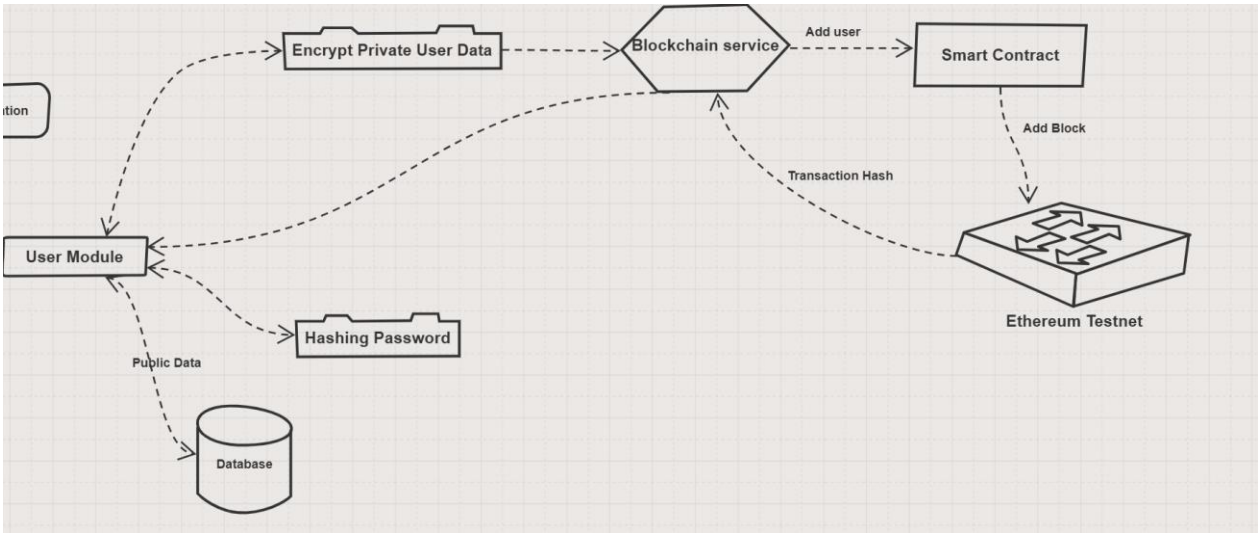


Figure 2.8. b) Create a new User

For updating and deactivating users, we have a similar data flow diagram:

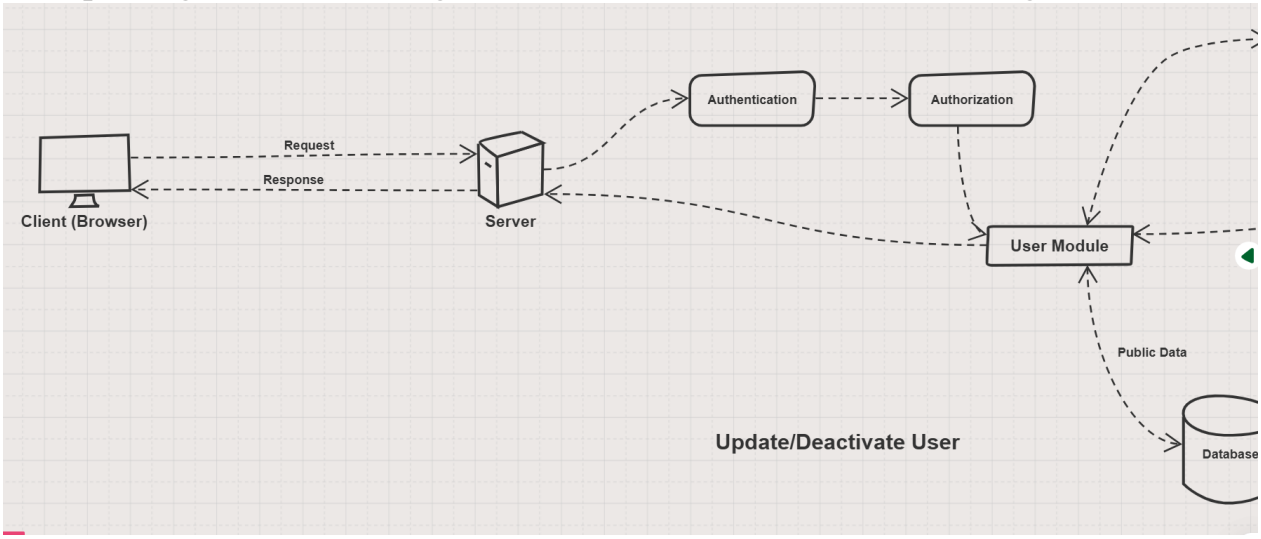


Figure 2.9. a) Update or Deactivate a User

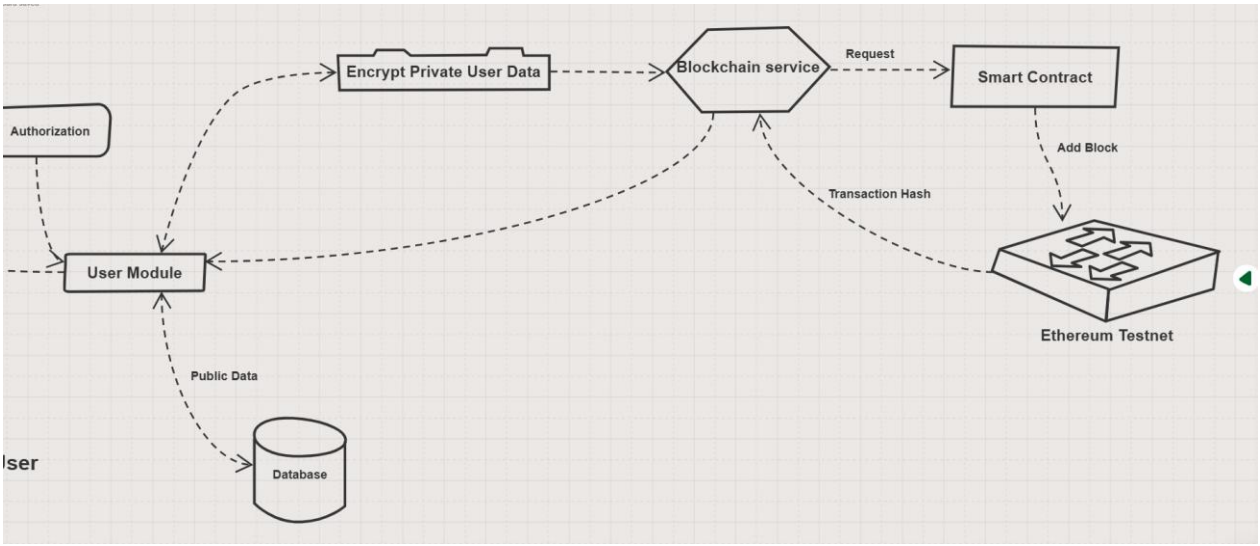


Figure 2.9. b) Update or Deactivate a User

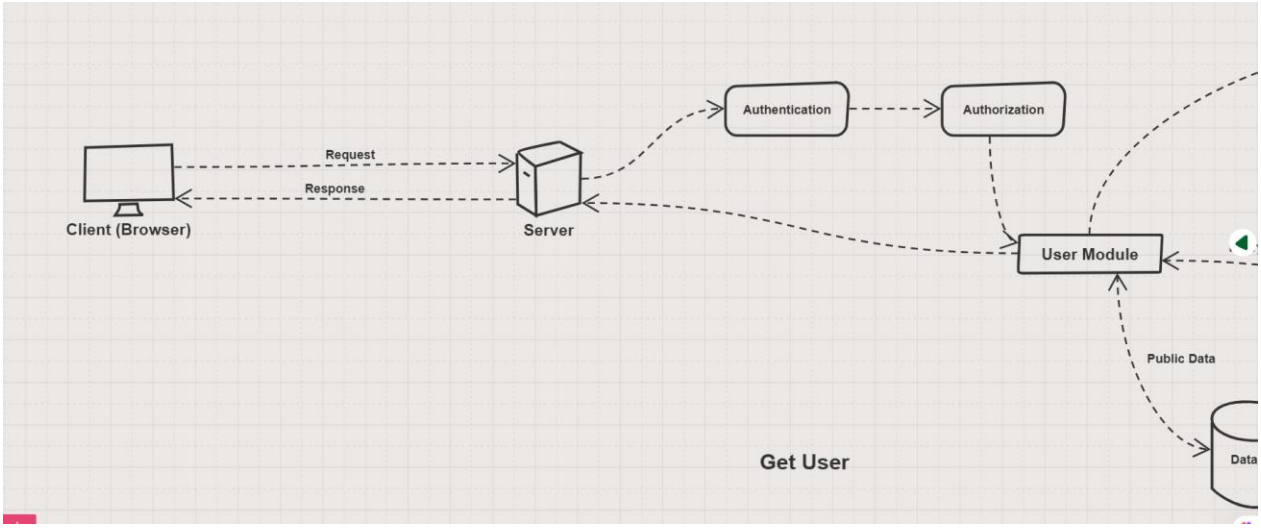


Figure 2.10. a) Get one or more Users

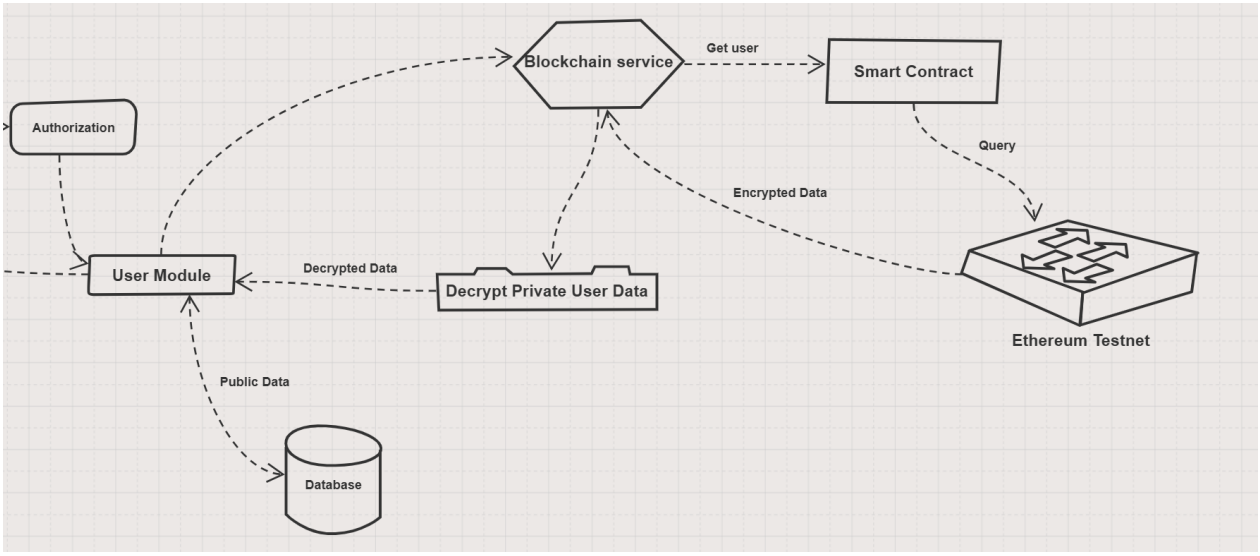


Figure 2.10. b) Get one or more Users

2.3.8. Upload File

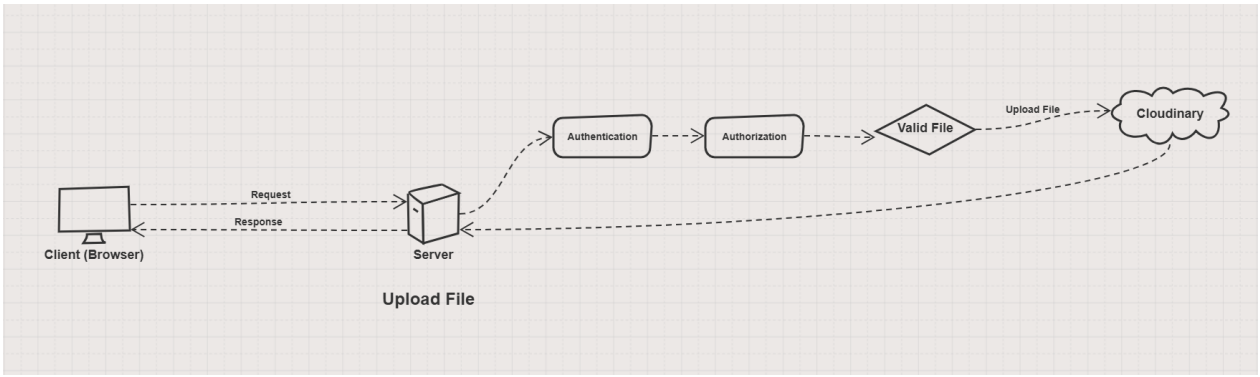


Figure 2.11. Upload a File

2.3.9. Contract Management

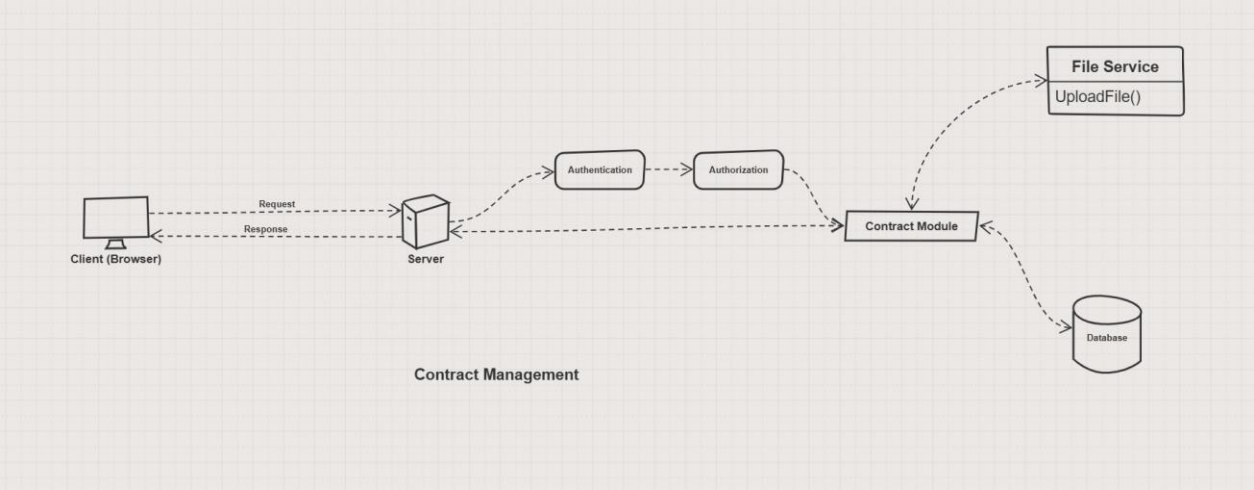


Figure 2.12. Contract Management

3. DATABASE DESIGN

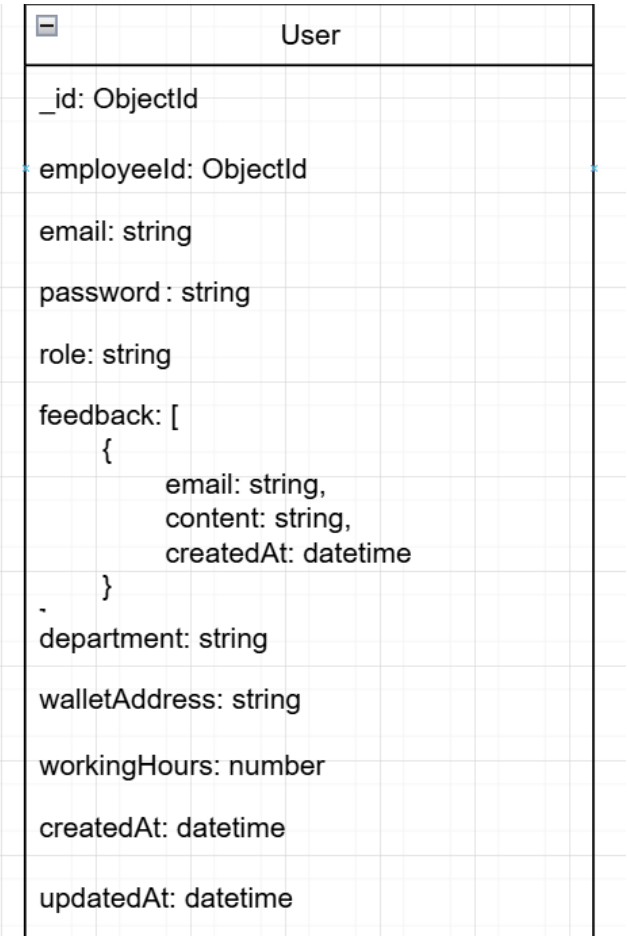
3.1. Data Model

3.2. Database Schema

Except for blockchain data, all other schemas are labeled with a timestamp and indicate who performed the create, update, and delete actions, as well as whether they have been deleted:

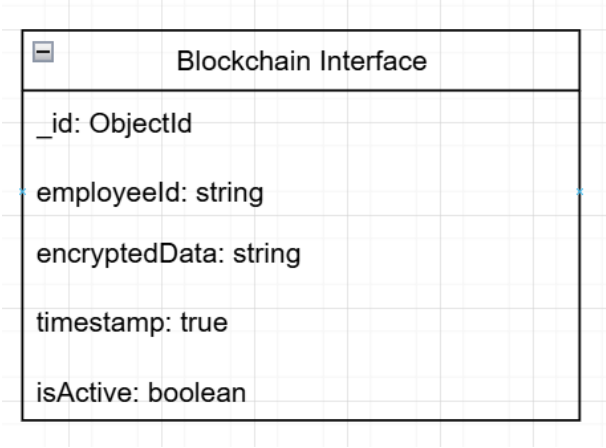
```
deletedBy {
        _id:ObjectId,
        email: string
}

updatedBy: {
        _id:ObjectId,
        email: string
}

createdBy: {
        _id:ObjectId,
        email: string
}

deletedAt: datetime

isDeleted: boolean

updatedAt: datetime

createdAt: datetime
```

3.2.1. User Schema

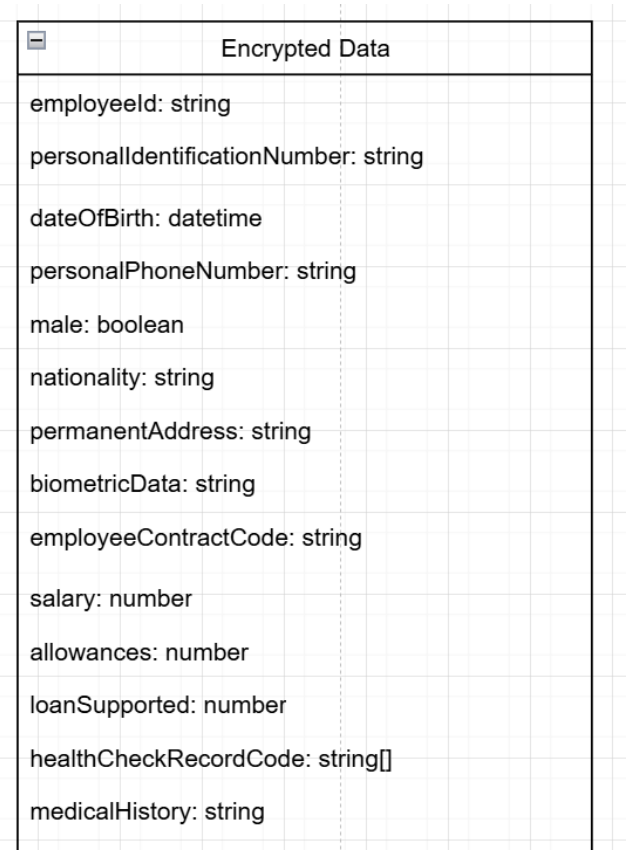


3.2.2. Blockchain

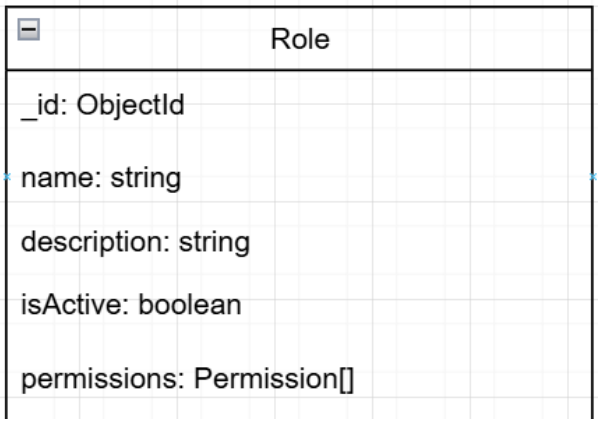
Interface Data Blockchain:



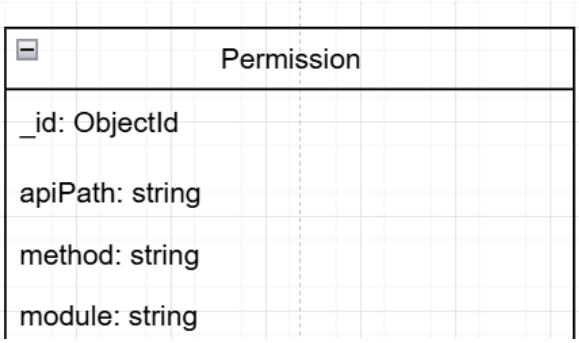
Blockchain Data:



3.2.3. Role Schema

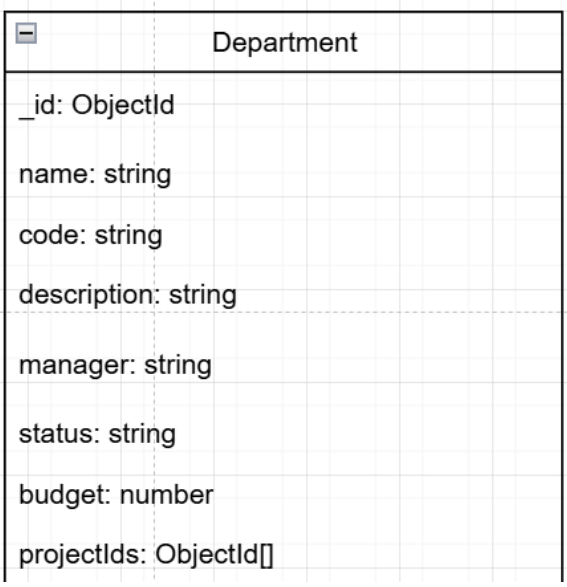


3.2.4. Permission Schema

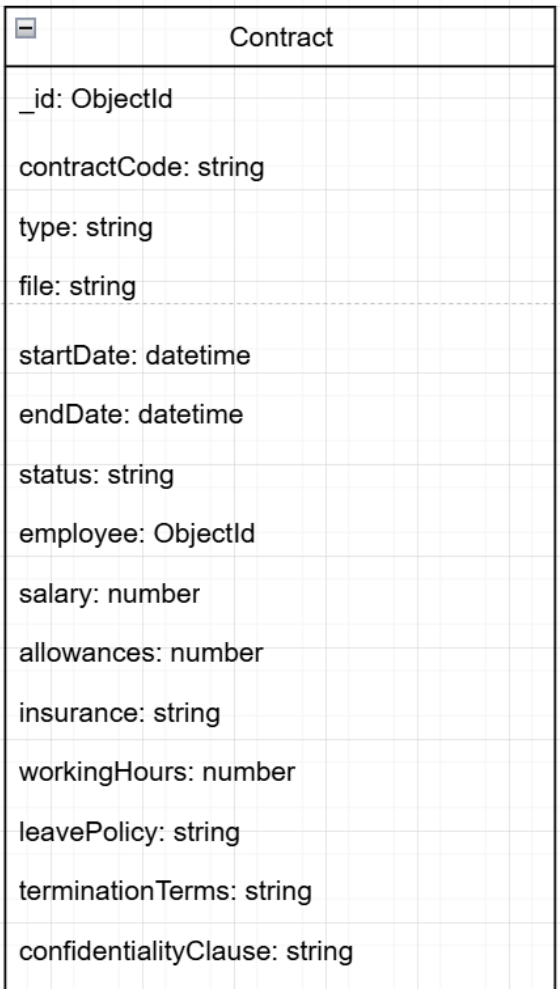


3.2.5. Feedback Schema

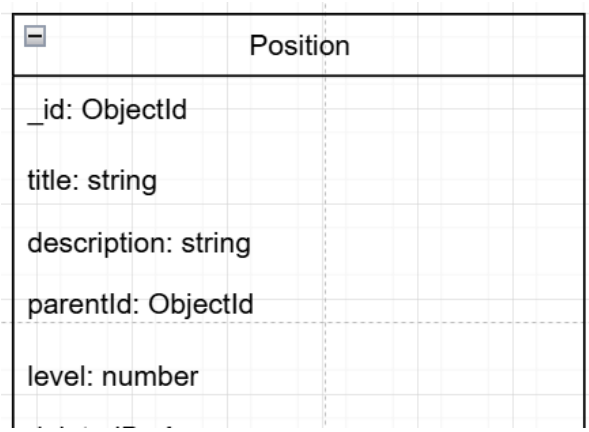
3.2.6. Department Schema



3.2.7. Contract Schema



3.2.8. Position Schema



3.3. Optimize database performance

In progress

4. COMPONENT DESIGN

4.1. List of Module

4.1.1. Auth Module

This module is responsible for user authentication and authorization, including the following functions:

- *Login authentication via passport-local:*

The system verifies the username and password provided by the user. If valid, it signs and returns an **access token** while storing the **refresh token** in a cookie.

- *Request authentication via JWT access token:*

The JWT token is sent in the request headers. When the server receives the request, it is immediately intercepted and processed at the strategy level (before reaching middleware). The system decodes the token and extracts important information from when the user signed the JWT (including user ID, name, email, and role). Then, based on the role, the system determines the permissions the user can access. The request is then passed to Auth Guard, which retrieves information from the strategy, verifies the token, and compares the user's permissions with the target API and target method. If valid, the request is allowed to proceed. This is a crucial step in securing APIs.

- *Handling expired access token:*

When the access token expires, the system retrieves the refresh token from the cookie to exchange it for a new access token. It then verifies the token against the database. If valid, a new access token and refresh token are generated, and the process follows the same steps as the login procedure. If the refresh token has also expired, the user will be required to log in again.

- *Logout:*

The system handles token deletion and invalidation.

4.1.2. Blockchain Module

The Blockchain Module is a critical component of the system, serving as an intermediary between the HRM system and the Ethereum network. It must be carefully managed and meticulously executed to ensure maximum security for users' private data, preventing any loss or errors. The module consists of the following key components:

- *Initialization Function:*

To connect to the Ethereum network, an initialization function is required. First, this function calls Web3 to initialize an instance that communicates with the blockchain through the blockchain endpoint provided in the environment variable. This endpoint is used to connect to a node in the blockchain network. To obtain a node, we create one using the *Chainstack* platform. Since the project is still in the development phase, most of the technologies used are free. However, once the project is deployed, we will transition to paid technologies or consider upgrading from a node to a validator to enhance security.

In case the endpoint fails (e.g., unable to retrieve the endpoint, node does not exist, etc.), Web3 will attempt to connect to a local blockchain network. This ensures that the system remains functional during debugging when the project is deployed. To provide a local blockchain network, we use *Ganache*. While its security is not entirely robust, this network runs directly on the local server even without an internet connection. This eliminates concerns about hacking and security issues.

Once the instance is created, Web3 will retrieve an account to perform transactions. This account exists within any wallet platform and is provided through an environment variable. In our project, we retrieve the account from a MetaMask wallet. If the private key is not found, Web3 will use an account from a predefined list (i.e., the accounts provided by Ganache).

At this stage, the connection to the blockchain is considered successful. However, to write transactions to the blockchain, an essential component called a *smart contract* is needed. Similarly, Web3 will fetch the smart contract address from an environment variable (this is the address provided by the blockchain network after deploying the smart contract) and create an instance to connect to the contract. Once connected to the contract, the initialization function is complete, and transactions can now be performed through the smart contract.

We develop our smart contracts using *Solidity*. During the development of the blockchain module, we also use the *Truffle framework*, which allows contract files to be compiled into *JSON (JavaScript Object Notation)* files. These files contain *ABI (Application Binary Interface), bytecode, metadata,* and other data that help Web3 connect to the smart contract. Additionally, Truffle supports smart contract deployment, smart contract management, blockchain network management, and more.

- *Get Balance, Call Contract and Execute Contract:*

After initialization, we define these three functions to enable more efficient interaction with the blockchain. Let's explore the functionality of each.

Get Balance: This function retrieves the balance of a wallet address (in Wei) and converts it to Ether. The main purpose is to check whether there are sufficient funds to perform transactions.

Call Contract: This function simply reads data from the smart contract. We implement this function for multiple purposes, the most important of which is retrieving employee information via the smart contract to support employee data queries. Additionally, it is used to verify permissions and validate data before performing actions.

Execute Contract: This function writes data to the smart contract to make changes in the blockchain. It is called whenever a modification to an employee's confidential data occurs, including adding, updating, and deactivating employees. Let's explore this in more detail below.

- *CRUD Employee Functions:*

We have developed a total of five functions, three of which perform data modifications (Add Employee, Update Employee, Deactivate Employee), and two that retrieve data (Get Employee, Get Employees).

These five functions primarily interact with the smart contract. The first three write transactions to the smart contract, while the last two query the smart contract for data.

Notably, both the *Add Employee* and *Update Employee* functions must encrypt user data before storing it on the blockchain. Since Ethereum operates as a public blockchain network, encryption is necessary to prevent the exposure of sensitive employee information.

Encryption algorithms and security technologies will be discussed in detail in a later section.

4.1.3. Chat Module

4.1.4. Contracts Module

This module is responsible for managing employee contracts with the company, including basic CRUD operations: creating, updating, deleting, retrieving a single contract, and retrieving multiple contracts. Several key points need clarification:

- This module manages most of the critical information related to contracts but **does not function as an actual contract**. The *file* field in the contract schema (as mentioned in the *Database Design* section) is the actual storage location for the official contract documents.
- The *salary* and *allowances* fields are used to support other modules, such as the Personnel Module and Reports Module. Additionally, while other fields have been designed, they currently do not contribute significantly to the system.

However, future system enhancements could introduce automation to reduce manual intervention by HR administrators. For example:

- The system could automatically detect expired contracts and notify employees to renew them.
- It could also track working hours and generate reports if an employee does not meet the required hours as per their contract.

4.1.5. Departments Module

This module is responsible for managing *departments*, grouping employees within the database to streamline data management. Each department consists of:

- A manager (i.e., the department head)
- Employees

Currently, due to the project's scale, all employees are simply listed in the department without distinct role classifications. However, if the project is deployed, we will need to develop a more detailed department structure to better accommodate medium and large enterprises.

Additionally, this module manages department budgets, enabling further development of salary and bonus-related functions. This helps ensure tighter financial management for the business.

This module plays *a crucial role* as it helps break down the organization's structure into smaller, manageable units, simplifying administration and reporting. It is also essential for generating *reports*, which will be discussed in the upcoming ***Reports Module*** section.

4.1.6. Feedback Module

This module is responsible for collecting feedback from employees. Feedback can be about any issue within the company, including colleagues, supervisors, company facilities, etc. The module also provides CRUD functionalities for feedback, meeting the basic needs of contributing opinions to the organization.

Submitting feedback ensures the anonymity of employees by *encrypting* the sender's ID and storing it along with the feedback. This is because, in cases of spam or malicious feedback intended to harm the company, we can decrypt the ID to identify the sender. This approach balances employee privacy and business protection.

To maximize employee rights, we have established a complex process for decrypting employee IDs:

- First, after a feedback submission, the system runs an automatic scanning function to detect specific keywords related to violence, suicide, abuse, sexual content, extremist language, and hate speech. If such keywords are found, a flag is automatically set when storing the feedback in the database.

- If an ID decryption request is made, the requester must have the necessary authorization to access this endpoint. This is a fundamental *authorization* mechanism of the system.
- Next, the reason for decryption must be verified. Currently, we only process requests where the reason is a *string longer than 10 characters*. Although this is a basic check, it is an important step to ensure that invalid or unjustified requests are reviewed properly.
- The system then examines who approves the request. ID decryption should be authorized by a *senior manager*. Currently, we only process this based on simple string verification. While this does not add much security, in real-world deployment, those with decryption privileges are typically high-ranking individuals such as *directors* or *system administrators*. However, we still log who approved the request for accountability.
- As the system evolves, an additional step will be implemented: logging all decryption requests into *an audit log*. This feature is planned for future development.
- The final layer of employee identity protection is the *secret key*. We use the ***AES-256-CBC*** encryption algorithm (which will be explained in detail later). This algorithm requires the same secret key for both encryption and decryption. Therefore, to decrypt an employee's ID, the requester must possess the correct secret key. If the provided key matches the one used during encryption, the decryption process will proceed.
- After decryption, feedback information is recorded in the database, including whether decryption was performed, the reason for decryption, who approved the request, and who carried out the decryption.
- The decrypted ID is returned to the client ***only once*** and is ***not stored in the database***. This ensures the employee's remaining rights and allows the company to handle the situation publicly or privately while preventing unauthorized individuals from viewing feedback senders.

4.1.7. Files Module

This module is responsible for uploading files (images, videos, documents, etc.) to the Cloudinary platform. We use Multer with the support of NestJS.

The module has only one endpoint upload, which includes a middleware configured to send files to the cloud. After that, the endpoint returns the file's *URL* and *Public ID*.

By default, uploaded files are stored in the *files* folder. However, if the client provides *a folder type*, the file will be saved in that specified folder.

Regarding allowed folders, we only accept image files (jpg, png, etc.) and documents (doc, docx, pdf, etc.), with a maximum file size of *10MB*.

4.1.8. Roles Module and Permissions Module

These are two very important modules as they play a crucial role in the authorization process of the system. In these modules, we also provide basic CRUD methods.

In the structure of a Permission, there are two attributes: name and module, which indicate what the permission is for and which module it belongs to. Additionally, there are two other important attributes: *API path* and *method*. The API path specifies which API endpoint the role is allowed to access, while the method indicates the type of HTTP request in the RESTful API (e.g., GET, POST, PUT, DELETE).

In the structure of a Role, the key aspect to focus on is an array of *Permissions*. This array defines the set of permissions that a user assigned to the role can access.

During login, an employee will sign a JWT (JSON Web Token), which is then returned to the client. Any subsequent request sent to the server will include this token. The system uses *Passport* to intercept and decode the token before the request reaches the middleware, extracting important information, including the user's *role*. Based on this role, the system queries the corresponding *permissions* and includes them along with the decoded token information.

The next stop for the request is the *Guard*, which validates the request by comparing the target API and method with the retrieved permissions. If there is a permission that matches the target API path and method, the request is allowed to proceed. Otherwise, the Guard denies access and returns a ***Forbidden Exception***.

Thus, these two modules play a critical role in securing data and protecting the API.

4.1.9. Position Module

This module is responsible for managing job positions within the company, playing a crucial role in structuring and operating the human resources system. Within this module, we provide full CRUD (Create, Read, Update, Delete) functionalities, allowing for flexible creation, updating, querying, and deletion of positions.

The structure of a *Position* includes basic information such as position name, job description, and responsibilities, along with a key attribute called *parent ID*, which specifies the higher-ranking position above the current one. This helps establish a clear hierarchical structure within the company, facilitating better management and task delegation.

Managing employee positions not only helps define each individual's role and responsibilities more precisely but also supports other processes such as permission control, performance evaluation, and optimizing the organizational structure. This ultimately enhances work efficiency and transparency within the company.

4.1.10. Projects Module

This feature is an extension compared to traditional HRM systems, allowing businesses to manage projects more efficiently. Even if a company does not take on external projects, this module can still be used to handle large and complex internal tasks.

The module provides full CRUD functionalities, enabling businesses to create, update, retrieve, and delete projects flexibly.

Projects are categorized by *departments*, ensuring optimized task management and resource allocation. Additionally, the system includes an *attachments* field, allowing users to store and manage project-related documents directly within the system, eliminating the need for third-party software or manual document handling.

Each project contains key information such as:

- *Manager (Product Owner):* The individual responsible for overseeing and managing the project.
- *Team Members*: A list of participating members, ensuring effective and transparent workforce management.
- *Task List:* A detailed list of tasks within the project, facilitating progress tracking and performance evaluation. The structure of a Task will be explored in the following section.

- *Progress*: This field represents the project's progress, calculated as the number of completed tasks over the total number of tasks. To ensure the most accurate progress calculation, we automatically compute the progress through three endpoints: get projects, get project by ID, and update project. When users call data query endpoints, the system automatically recalculates the project's progress before returning data to the client, ensuring the client receives the most accurate information. When the update project endpoint is called, the system automatically triggers the progress calculation function and updates this field in the database, eliminating the need for any additional intervention.

  ***Issue*:** This progress calculation method may not be fully optimized. When a task undergoes any changes (CRUD operations), there is *no intervention in the project's progress calculation*, which can lead to a significant discrepancy between the stored progress and the actual progress. The best solution would be to trigger the project progress calculation function whenever a task is modified.

  However, to avoid circular dependencies, we have decided not to take this approach. Instead, we will calculate the progress directly within the project query functions. This ensures that the project progress data retrieved by the client remains accurate at all times.

4.1.11. Tasks Module

During our business analysis, we realized that in order to improve efficiency within a department or unit, it is essential to clearly define and assign tasks to individual employees. However, most companies today do not have dedicated software for this purpose. Instead, they often rely on third-party applications such as Notion**,** Zalo, or similar tools.

Relying on external software presents several challenges:

- *Reduced security*, as internal data may be leaked or accessed without proper authorization.
- *Increased operational costs*, since businesses must pay for third-party services.
- *Difficulties in managing critical tasks*, especially when consistency and synchronization within the internal system are required.

To address these issues, we developed the *Task Module*—a built-in task management system that runs *entirely within the company's internal server*. This enhances *data security* while also helping businesses *reduce costs* by eliminating the need for external software.

This module allows leaders or managers to directly assign tasks to employees. Each task is designed with essential attributes, including important properties such as:

- *Priority*: Helps employees determine which tasks are more urgent.
- *Status*: Allows for tracking work progress with statuses such as Pending**,** Hold**,** Complete, etc.
- *Attachments*: Supports storing necessary documents directly within the system, eliminating the need for external file management.

Currently, the Task Module focuses on priority and status to help employees manage their work efficiently. In the future, we plan to expand its capabilities by introducing:

- *Notifications for Critical tasks* to ensure urgent work is not overlooked.
- *Automated reminders* to help employees stay on track with their assignments.

Additionally, the Task Module serves as the foundation for calculating ***employee KPIs****,* assessing individual and department performance, and optimizing company management strategies.

4.1.12. User Module

The *User Module* is the core component of our system, serving as the foundation for all other modules. Our development team has gone through multiple iterations to refine and enhance this module, ensuring its stability and efficiency.

Each *user* consists of two types of information:

- ***Public Data***: Stored in the database, accessible to anyone with permission to view user information.
- ***Private Data***: Stored on the blockchain, accessible only to the user and a limited number of authorized personnel (e.g., system administrators). The details of this data classification are outlined in the **User Schema** section above.

Key Methods:

To support internal operations and interactions with other modules, we have implemented the following essential methods:

- ***Hash Password***: Passwords are a crucial field for authentication and are stored in the database for performance optimization. To ensure a minimum level of security, we hash passwords before storing them. Since *hashing* is a one-way process (as opposed to encryption, which allows decryption), this guarantees that passwords remain protected. The specific hashing algorithm is discussed in the *Security* section.
- ***Is Valid Password & Get User by Username***: These two methods support the *Auth Module* during login and user authentication.
- ***Get User by Token & Update Token***: These methods handle user authentication when an access token expires, allowing the system to issue a new access token via a refresh token.
- ***Create User***: This method performs two main tasks:
    1. *Creates a user record in the database* with public data.
    2. *Registers a new user block on the blockchain.* After multiple iterations, we decided to allow users to update their own *private data*. However, for this to work smoothly, a block must be created without storing sensitive details (only the employee ID) on the blockchain. This requires the company to bear a small transaction fee (gas cost for block creation). Additionally, the method automatically updates the *user's department*, reducing manual work for HR.
- ***User Search Methods***: We have developed four search functions:
    1. *Find user by ID*: Retrieves a single user from the database.
    2. *Find multiple users*: Queries and returns multiple user records.
    3. *Find users by IDs*: Designed to support automated system reporting.
    4. *Find users including private data*: This endpoint is strictly restricted to the user and system administrators to ensure privacy protection.
- ***User Update Methods***: Over time, we have optimized user updates by dividing them into three separate functions:
    1. *Update Public User*: Updates *public data* and stores it in the database, functioning like standard update methods.

2. *Update Working Hours*: Since work hours tracking is a daily operation for attendance and performance monitoring, we created a dedicated function to optimize system performance.
3. *Update Private User Data*: Employees need to update their private data upon joining the company. HR can create user accounts but does not have access to private data. Separating this function prevents unnecessary blockchain updates, reducing potential errors.

- ***Remove User***: When an employee leaves the company, we deactivate their profile on the blockchain (since blockchain data cannot be deleted, only deactivated) to prevent errors during queries.

Additionally, the User Module provides an Interface User that logs account activities, playing a crucial role in system-wide interactions.

As the most essential module in the system, the User Module serves as the foundation for all other modules, which are developed to enhance and complement its functionality. Optimizing this module directly contributes to improving *HR management efficiency.*

4.1.13. Personnel Module

This module plays a critical role in *human resource management*, making it an indispensable component of the HRM system. It provides several essential functions, including:

- Payroll Calculation: This process is fully automated, requiring no external intervention. Since user data on the blockchain includes salary, bonuses, allowances, adjustments, and other compensation-related details (with potential future expansions), this endpoint retrieves private data from the blockchain, performs the necessary calculations, and returns the accurate payroll to the client.
- Salary Advance Requests: This feature is an enhancement compared to traditional HRM systems. Instead of requiring employees to submit manual requests and wait for approval, we have automated the process:
    1. If an employee requests an advance *of $400 or less* and has *no prior outstanding advances*, the system *automatically approves* the request.
    2. If the requested amount *exceeds $400* or the employee already has *an outstanding balance*, the system records the request in the database and *waits for manager approval*.

- Approving Salary Advance Requests: When a manager (or any authorized personnel) approves a pending request, the system records the approval status accordingly.
- Searching Salary Advance Requests: To ensure transparency and *financial control*, we have developed two search functions:
    1. View details of a single request.
    2. Retrieve multiple requests for better financial management.
- KPI Calculation: This is a *powerful tool* that helps businesses optimize human resources by automating KPI calculations. There are various types of KPIs, but since this software is designed for enterprises, we calculate them based on the following formula:

$$\frac{\text{Tasks Completed on Time}}{\text{Total Tasks}} \times 100$$

  Once calculated, KPIs are *stored in the database* for reporting and analytics.

- Managing Salary Adjustments: Throughout an employee's tenure, salary adjustments may be required due to *company policies* (e.g., exceeding KPIs, late arrivals, policy violations, etc.). Managing these adjustments ensures payroll accuracy. Since these records are *critical*, they are securely stored on the ***blockchain***.
- Updating Employee Working Hours: Although this function belongs to the Personnel Module, the endpoint is managed within this module due to its deep integration with the User Collection. The method *calls the User Service* to update employee working hours.

Currently, both automatically approved and manually approved salary advance requests only update the system status. However, once the system is fully deployed, the development team can *integrate with a bank's API* to enable *automatic salary transfers* upon approval.

This module *plays a crucial role* in HR operations, ensuring efficiency and accuracy in payroll management. To support a well-functioning business, it is essential to develop this module thoroughly and with precision.

4.1.14. Reports Module

The primary goal of this module is to ***automate the reporting and analytics process***, reducing manual effort and allowing HR teams to focus on more strategic tasks. The system automatically retrieves data from relevant modules, processes the statistics, and returns the final report to the client.

Types of Reports Provided:

- *Employee Report*: Lists all users categorized by department.
- *Employee Changes Report*: Tracks employees who have *resigned* or *recently joined*.
- *Working Hours Report*: Aggregates *working hours* by department, helping HR implement *reward and disciplinary measures* more effectively.
- *Leave Report*: Summarizes employee leave records.
- *KPI Report*: Compiles employee KPI statistics.
- *Salary Report*: Provides actual *salary data* for each employee, grouped by department, along with total payroll costs for each *department*. This ensures *financial transparency and optimization* for the business.

For new employees who have not yet received a *KPI score or salary record* (e.g., first-month employees), the system *automatically calculates KPI and salary*. However, in practice, HR should first call the salary and KPI calculation endpoints before generating the final report to ensure accuracy.

4.2. APIs & Interfaces

4.2.1. Overview

APIs are essential components that facilitate communication between the client and server. This section provides a detailed breakdown of the APIs used in our system.

Client-side Notice: When making API requests, ensure that all required information follows the specified Data Transfer Object (DTO) format to prevent errors.

Server-side Response Format: The API responses adhere to a standardized structure as follows:

Generate API Response Format
All APIs return data in JSON format with the following structure:

```
{
  "statusCode": (200-500),
```

```
  "message": "string",
  "data": (response data varies depending on the API)
}
```

Create APIs

For APIs that create new records (typically using the **POST** method), the data field contains only the _id of the newly created record, formatted as follows:

```
"data": {
  "_id": ObjectId
}
```

Update APIs

For APIs that update existing records (typically using the **PATCH** method), the response includes a MongoDB update operation result with the following structure:

```
"data": {
  "acknowledged": boolean,
  "modifiedCount": number,
  "upsertedId": ObjectId or null,
  "upsertedCount": number,
  "matchedCount": number
}
```

Get APIs

For APIs that retrieve data, we categorize them into two types:

- *Single Record Retrieval***:** The data field follows the predefined interface structure of the respective module.
- *Multiple Records Retrieval***:** The data field follows this format:

```
"data": {
  "meta": {
    "current": number,
    "pageSize": number,
    "pages": number,
    "total": number
  },
  "result": [
    // Each item follows the predefined module-specific interface.
  ]
```

}

Delete APIs

For APIs that delete records, the response format is as follows:

```
"data": {
  "deleted": number
}
```

4.2.2. Users Module

List APIs:

POST api/v1/users

PATCH api/v1/users/:id

PATCH api/v1/users/public/:id

DELETE api/v1/users/:id

GET api/v1/users/:id

GET api/v1/users/private/:id

GET api/v1/users

4.2.3. Chat Module

4.2.4. Contracts Module

List APIs:

POST api/v1/contracts

PATCH api/v1/contracts/:id

GET api/v1/contracts

GET api/v1/contracts/:id

DELETE api/v1/contracts/:id

4.2.5. Departments Module

List APIs:

POST api/v1/departments

PATCH api/v1/departments/:id

GET api/v1/departments

GET api/v1/departments/:id

DELETE api/v1/departments

4.2.6. Feedback Module

List APIs:

POST api/v1/feedback

GET api/v1/feedback

GET api/v1/feedback/:id

POST api/v1/feedback/decrypt/:id

4.2.7. Permissions Module

List APIs:

POST api/v1/permissions

PATCH api/v1/permissions/:id

GET api/v1/permissions

GET api/v1/permissions/:id

DELETE api/v1/permissions/:id

4.2.8. Personnel Module

List APIs:

POST api/v1/personnel/adjustments/:id

POST api/v1/personnel/salary

POST api/v1/personnel/salary/approve/:id

PATCH api/v1/personnel/working-hours/:id

GET api/v1/personnel/salary/calculate/:id

GET api/v1/personnel/salary/:id

GET api/v1/personnel/salary

GET api/v1/personnel/kpi/:id

4.2.9. Positions Module

List APIs:

POST api/v1/positions

PATCH api/v1/positions/:id

GET api/v1/positions

GET api/v1/positions/:id

DELETE api/v1/positions/:id

4.2.10. Projects Module

List APIs:

POST api/v1/projects

PATCH api/v1/projects/:id

GET api/v1/projects

GET api/v1/projects/:id

DELETE api/v1/projects/:id

4.2.11. Reports Module

List APIs:

GET api/v1/reports/employees

GET api/v1/reports/employees-turnover

GET api/v1/reports/working-hours

GET api/v1/reports/day-off

GET api/v1/reports/kpi

GET api/v1/reports/salary

4.2.12. Roles Module

List APIs:

POST api/v1/roles

PATCH api/v1/roles/:id

GET api/v1/roles

GET api/v1/roles/:id

DELETE api/v1/roles/:id

4.2.13. Tasks Module

List APIs:

POST api/v1/tasks

PATCH api/v1/tasks/:id

GET api/v1/tasks

GET api/v1/tasks/:id

DELETE api/v1/tasks/:id

4.3. Session Management & Security

4.3.1. Session Management

The user authentication process using JWT has been described previously. What we aim to clarify next is the concept of a *user session*.

By design, each access token has a lifespan of 10 minutes. After this period, the token expires and becomes invalid. To continue their session, the user must use the refresh token to obtain a new access token.

As for the refresh token, we define its lifetime as 1 day. After one day, the user is required to log in again to continue using the system. Therefore, in this setup, a user session is effectively limited to **1 day**.

Rotation Refresh Token Strategy

A 1-day session may be too short for regular employees, who would then need to log in every day. This can lead to inconvenience and a degraded user experience. To address this issue, we have implemented a *Refresh Token Rotation* strategy.

With this approach, each time a refresh token is used to obtain a new access token, a *new refresh token is also issued*. This strategy provides two key benefits:

- *Enhanced security*: If a refresh token is compromised, it becomes useless once it has been used, as a new token is immediately generated and the old one is invalidated.
- *Extended user session*: Since the refresh token is renewed every time the access token is refreshed, the session lifetime is effectively extended. As long as the user continues to request new access tokens before the current refresh token expires, their session can persist for an extended period.

This mechanism ensures both *security* and a *seamless user experience*, especially in scenarios where long-lived sessions are desirable.

4.3.2. Security

Security is a top priority in this project, and encryption algorithms play a crucial role in ensuring data protection. We employ multiple security techniques, which can be summarized as follows:

Authentication and Authorization

This is the most critical layer for securing APIs and managing user access. The implementation details are outlined in the *Auth Module* and the *Roles & Permissions Module*.

Encryption and Hashing Algorithms

Various types of sensitive data require encryption and hashing. Below is a summary of the key algorithms and tools we use:

Algorithm	*Purpose*	*Module*
Bcrypt	Hash user's password	Users
SHA-256	Encrypt secret key (32 byte string)	Security
MD5	Encrypt IV string (16 byte string)	Security
AES	Encrypt and decrypt employee data before writing to blockchain	Security, Blockchain
AES-256-CBC	Encrypt and decrypt employee ID in each feedback	Security. Feedback
RSA	Encrypt secret key	App

5. USER INTERFACE DESIGN

5.1. Wireframes & Mockups

In progress

5.2. User Flow Diagram

In progress

6. PERFORMANCE & SCALABILITY REQUIREMENTS

6.1. Performance Optimization

In progress ok

6.2. Scalability Considerations

The system is currently in the development phase, and future expansion is inevitable. With the addition of more features and increasingly complex processing flows, we have provided a fairly comprehensive description of the existing modules. However, those mainly cover minor features or specific sub-processes. Below is a summary of the major planned extensions for future development:

Recruitment Module

- In an HRM system, managing CV review, scheduling interviews, and overseeing the recruitment process are essential functionalities.
- However, our business analysis revealed that the recruitment process is relatively complex. Due to current limitations in time and resources, we have decided to defer this module's implementation to a future development phase.

Contracts Module

- Notify employees or the HR department when a contract is about to expire.
- Compare daily working hours stated in the contract with actual recorded working time to assess compliance.

Feedback Module

- Log audit records when decoding employee IDs to ensure transparency and traceability

Tasks Module

- Send alerts to employees when critical-level tasks are assigned.
- Remind employees of tasks that are approaching their deadlines.

Personnel Module

- Currently, financial data fields for each employee include salary, allowances, and adjustments.
- In the future, we plan to expand and detail these financial records further to support a more accurate and automated payroll system.

Other planned extensions

- Numerous other expansion plans, both minor and major, have been outlined and will continue to be updated and implemented in the future.

6.3. Fault Tolerance & Recovery Plan

6.3.1. Fault Tolerance

In terms of fault tolerance, we have made every effort to handle all potential error scenarios. For each specific error, we have implemented clear and detailed handling strategies. However, we understand that unforeseen issues may still arise during real-world deployment. To address this, we have *implemented a global exception filter* provided by the framework. This filter intercepts all exceptions that occur while the server is running, converts them into HTTP exceptions, and returns the corresponding status codes to the client. If the error is not an HTTP exception, it defaults to an internal server error (500).

By leveraging this global filter, we ensure that all runtime errors are captured effectively, allowing us to analyze and resolve issues more efficiently. This also guarantees the stability of the server, preventing it from crashing under unexpected circumstances.

6.3.2. Recovery Plan

In progress

7. TESTING PLAN

7.1. Types of Testing

In progress

7.2. Testing Tools

In progress

8. DEPLOYMENT & MANTAINANCE PLAN

8.1. Deployment Strategy

In progress

8.2. Monitoring & Maintenance

In progress

Acknowledgment

We would like to express our sincere gratitude to all individuals and organizations who have contributed to the development and completion of this software design document.

First and foremost, we extend our deepest appreciation to the members of the development team. Your dedication, relentless efforts in research, design, and system optimization have played a crucial role in the success of this document. Your creativity, effective teamwork, and strong commitment have been the solid foundation that enabled us to refine every detail of the design.

Additionally, we would like to express our gratitude to the lecturers, technical advisors, and mentors who have provided invaluable support throughout this process. Your profound expertise, constructive feedback, and insightful guidance have not only improved the content of this document but also enhanced our approach and systematic thinking in software design.

We also want to acknowledge the support from individuals and organizations who have directly or indirectly contributed by providing reference materials, sharing practical experiences, and offering valuable insights. These contributions have helped us enhance the quality of this document and ensure that our system designs are highly applicable.

Finally, we sincerely thank everyone who has accompanied us on this journey. Your support and encouragement have been a great motivation, enabling us to complete this document with the highest quality.

Thank you very much!

Best regards,

Tri Ngoc