COMP5521 DISTRIBUTED LEDGER TECHNOLOGY, CRYPTOCURRENCY AND E-PAYMENT

2024-Fall Project: Student Attendance Application Based on NaiveCoin

Presenter: MA Zhiyuan, QIN Cailing

November 25, 2024

Contents

Introduction

Project Background

Problem Statement:

- To build a blockchain-powered student attendance system based on the **NaiveCoin** project.

Objective:

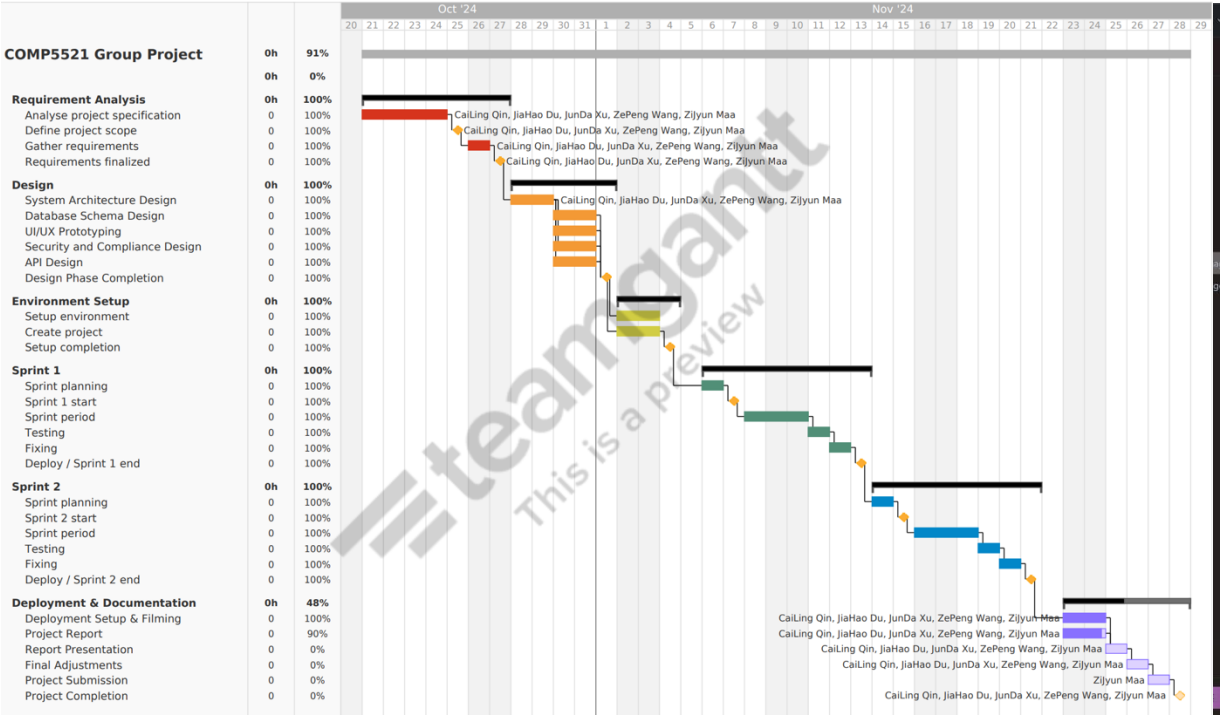
- Design and implement a blockchain-based attendance management system based on the **NaiveCoin**, supporting core functions such as student sign-in, attendance record and query.

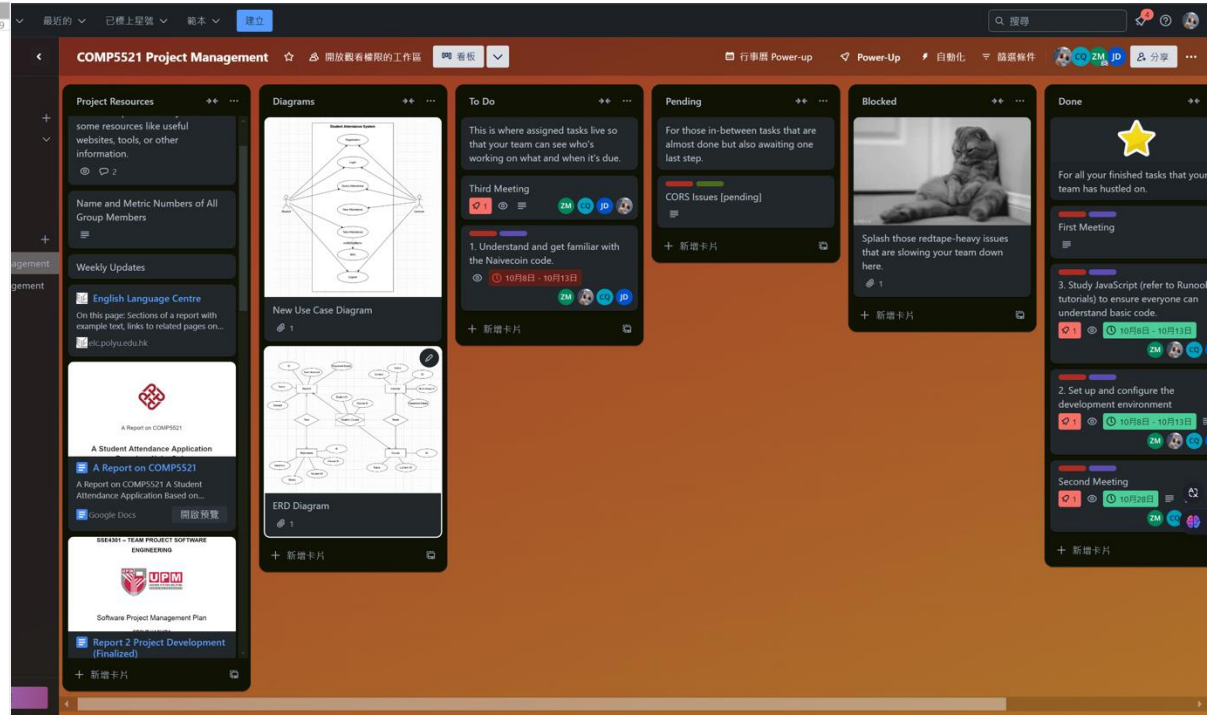
Methodology:

- We used **RAD** as our methodology of project development. Because of **SHORT deadlines** and **FREQUENT changes.**

Project Management

- We use **Gantt charts** to plan tasks and allocate work, and **Trello** to share information and track progress in real time.



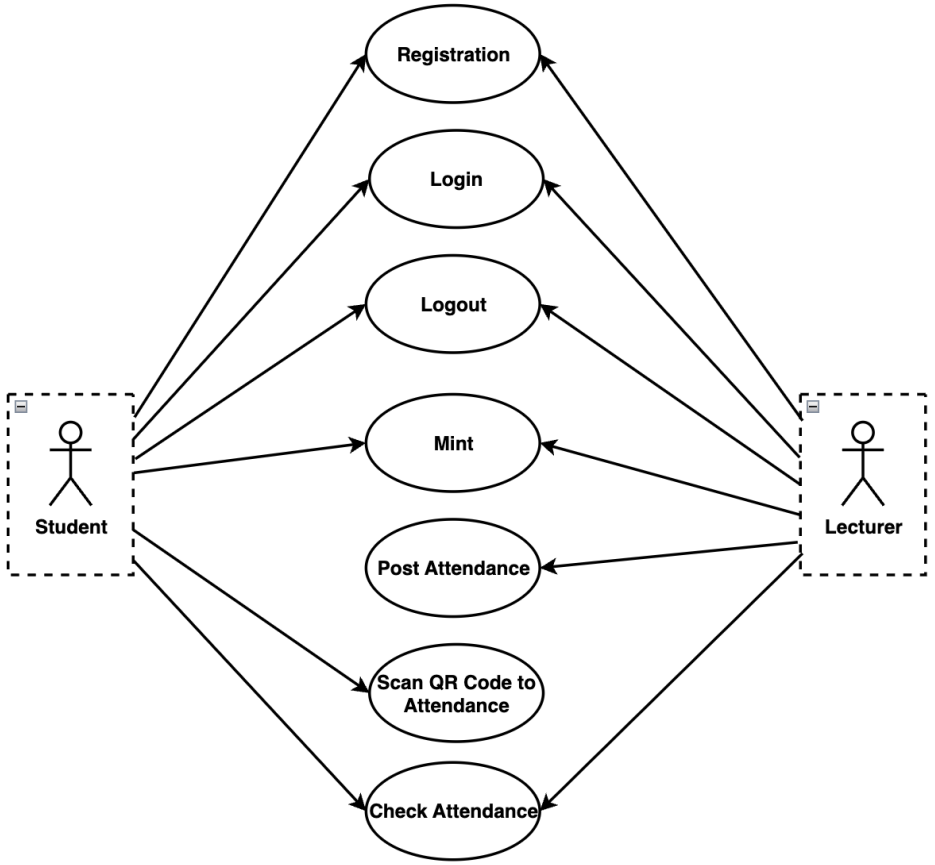


Analysis & Design - Use Cases

Requirement Analysis

Use Case Diagram

- Identify Actors and Interactions
- Define Functional Requirements
- Scope Validation



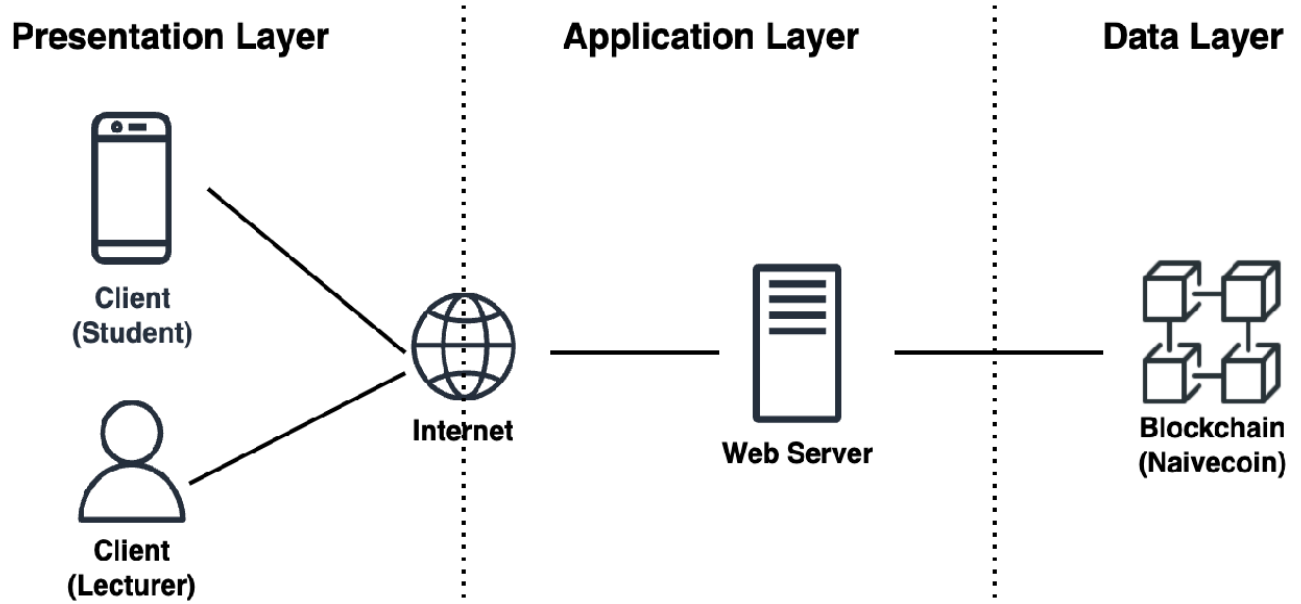
Analysis & Design - Requirements

Source	Type	Priority	Requirement Description	Status
Stakeholder	Non-Func.	High	*Dynamic difficulty	Finished
Stakeholder	Non-Func.	High	*Fork resolution mechanism	Finished
User	Func.	Medium	Registration	Finished
User	Func.	Medium	Login	Finished
User	Func.	Medium	Logout	Finished
User	Func.	High	New Attendance/ Event	Finished
User	Func.	High	Query Attendance	Finished
User	Func.	High	*Take Attendance/ Sign Certificate	Finished
User	Func.	High	*Mint (Mining) with Certificate Verification	Finished
User	Non-Func.	Medium	*User-friendly Operating Interfaces*	Finished

Analysis & Design - Architecture

Presentation Layer

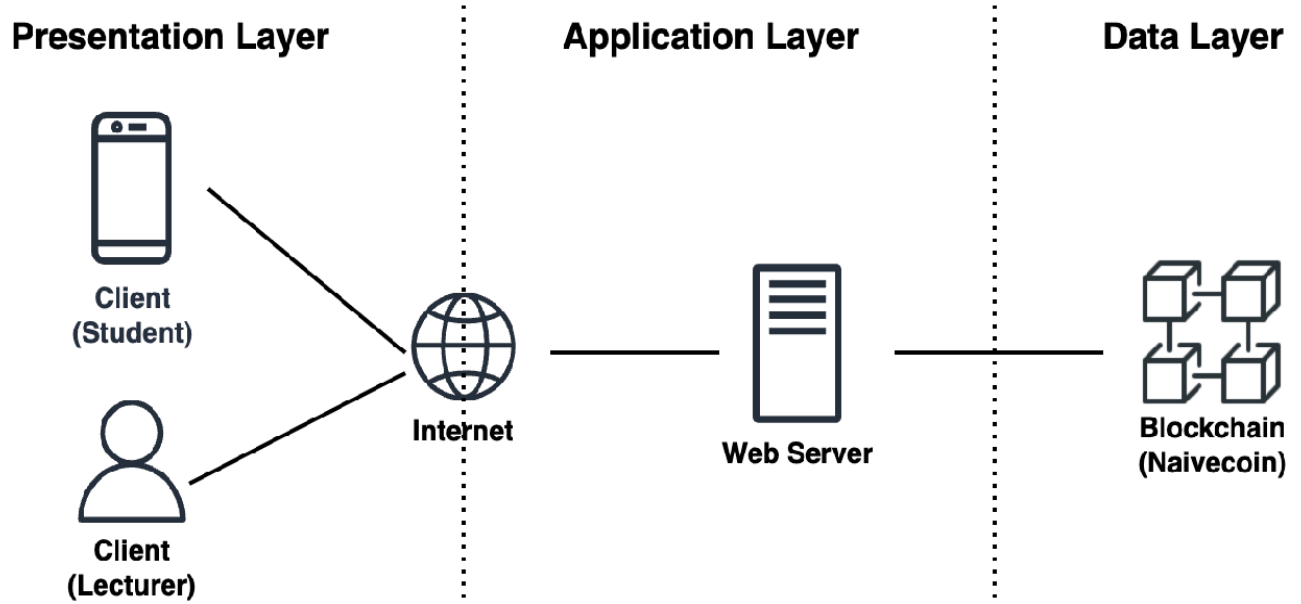
- Frontend: **Next.js**
- Handles user interaction, provides UI, and forwards requests to the backend.



Analysis & Design - Architecture

Business Logic Layer

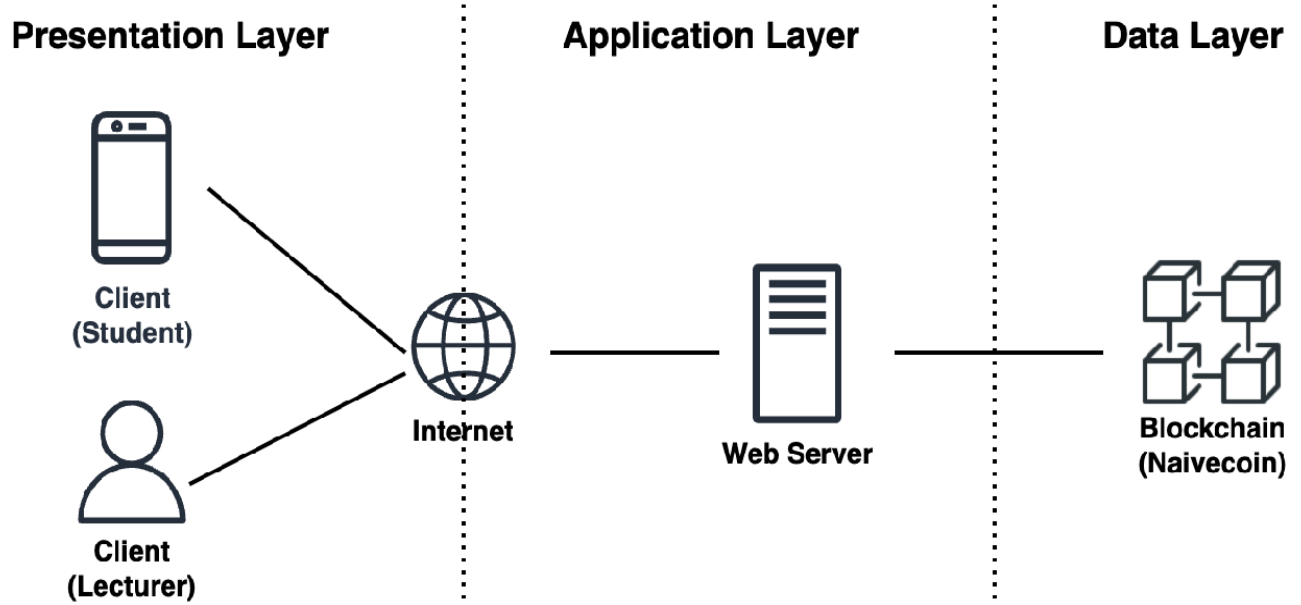
- Backend: **Node.js**
- Responsible for implementing the core blockchain logic and processing requests from the frontend.



Analysis & Design - Architecture

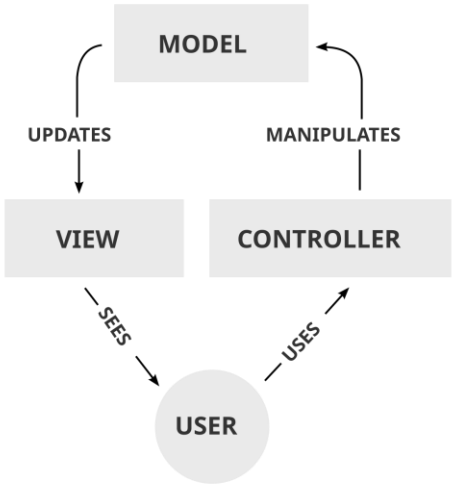
Data Layer

- Ledgers (**NaiveCoin**)
- Where data about blocks, transactions, and blockchain history is stored



Analysis & Design – Web Application

- Front-End Development
  - Development Framework:
    - Next.js - *Fast routing, High performance, Easy to use*
    - MVC Pattern – *Model*, *View*, *Controller*
  - CSS Style:
    - Tailwind



Implementation

- Authentication
  - Functions:
    - ***Registration***
    - ***Login***
    - ***Logout***

  - Use ***Localstorage*** to store user (lecturer & student) information, you can only use this application if the user information exists in ***Localstorage***.

```
const setAttribute = async (
  name: string,
  value: string
): Promise<boolean> => {
  try {
    localStorage.setItem(name, value);
    console.log(`Set ${name}: ${value} in localStorage`);
    return true;
  } catch (error) {
    console.error(`Failed to set ${name} in localStorage:`, error);
    return false;
  }
};
```

```
useEffect(() => {
  const user = LocalStorage().getAttribute('user');
  if (!user) {
    router.push('/login'); // 用户未登录时跳转到登录页
  }
}, [router]);
```

Implementation

Attendance – Event (Lecturer & Student):

Lecturer:

- *Lecturers* records the Information of the created Event on the blockchain by **sending transaction to themselves**.

Student:

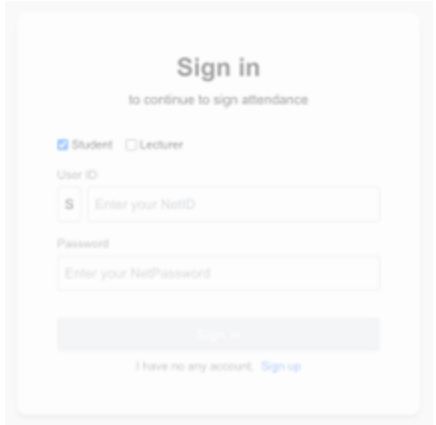
- *Students* sign in existing events by **sending transactions to** the event owners (**Lecturers**).

Tips: Transactions created by *Lectures* are different from those created by *Students*. “<eventId>-created” & “stuId” is void.

Implementation

Data Structure

- Class Diagram
- To show what kind of attributes we used for contain the Event and User information to communicate between Frontend and Backend.



UserClass
walletId: string userId: string address: string
constructor(walletId: string, userID: string, address: string): UserClass toJSON(): object fromJSON(json: any): UserClass

EventClass
fromAddress: string toAddress: string; amount: number; changeAddress: string; stuId: string; teacherId: string deadline: string remark: string timestamp?: number
constructor(fromAddress: string, toAddress: string, amount: number, changeAddress: string, stuId: string, teacherId: string, eventId: string, deadline: string, remark: string, timestamp?: number): EventClass toJSON(): object fromJSON(json: any): EventClass

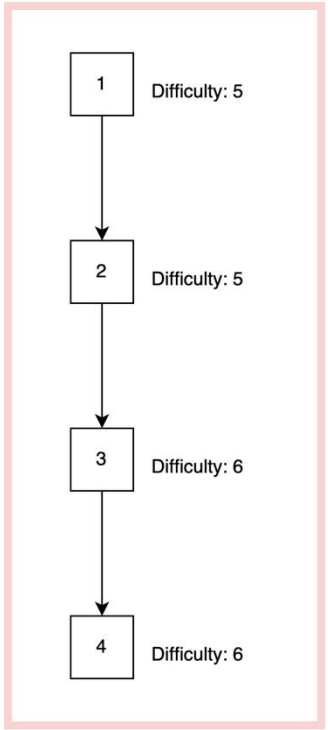
{"fromAddress":"050174cbfb1b5670ad7e5ee3de22cd9bf544ddd5cf8b76ac2053283889189f12","toAddress":"","amount":0,"changeAddress":"050174cbfb1b5670ad7e5ee3de22cd9bf544ddd5cf8b76ac2053283889189f12","stuId":"","teacherId":"L200801","eventId":"COMP1234-create","deadline":"2024-11-24T11:50","remark":"Lecture1"}

Implementation

Basic Fork Resolution

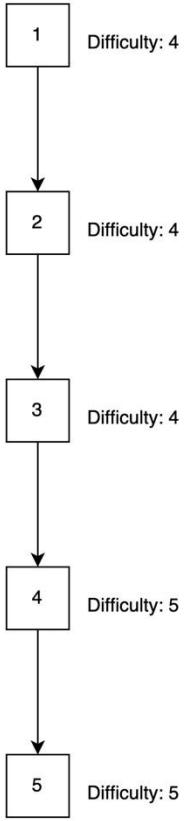
Chain A

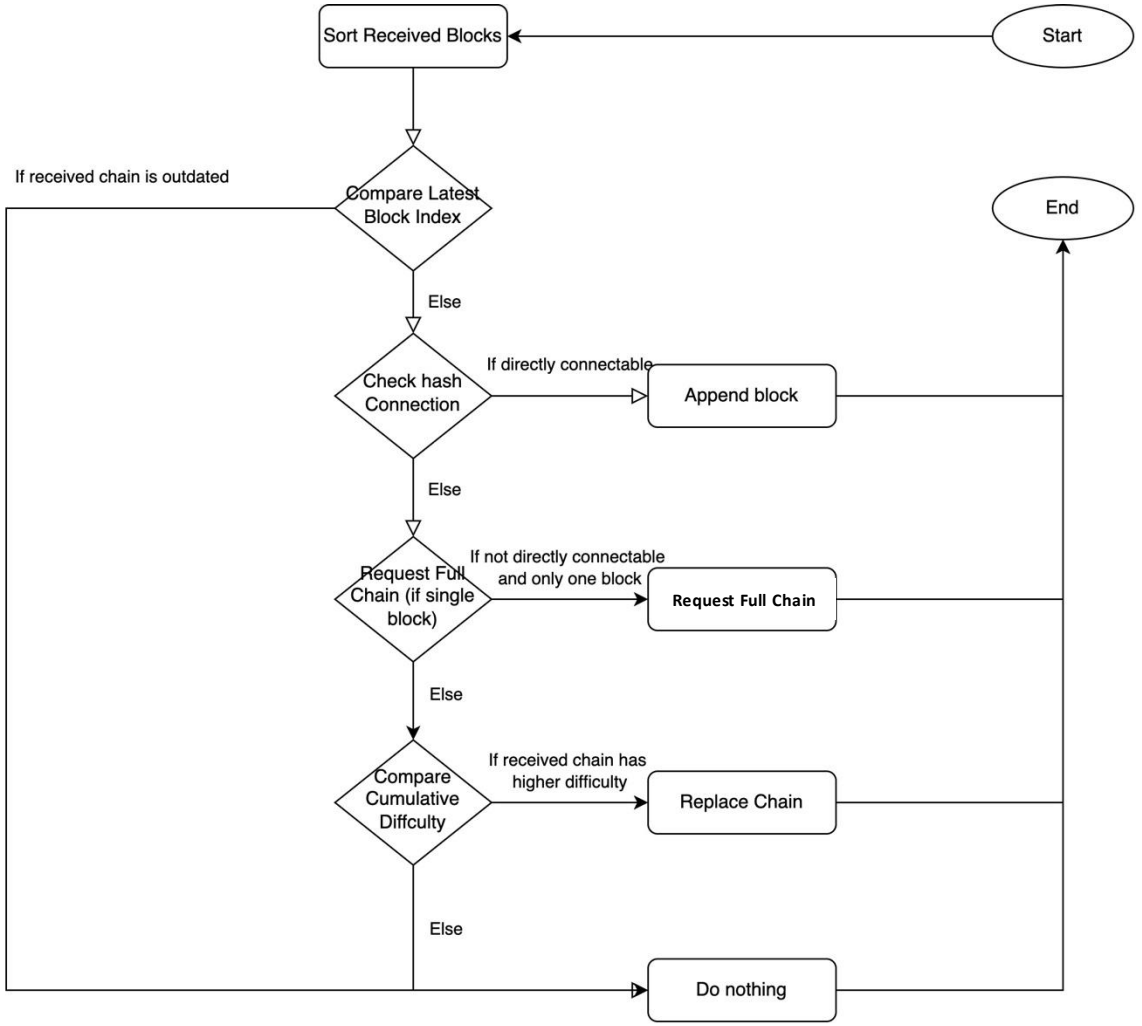
$2^5 + 2^5 + 2^6 + 2^6 = 192$



Chain B

$2^4 + 2^4 + 2^4 + 2^5 + 2^5 = 112$





Implementation

Dynamic Difficulty

Purpose:

- Maintain the time stability of block generation.
- Avoid excessive consumption of network resources.

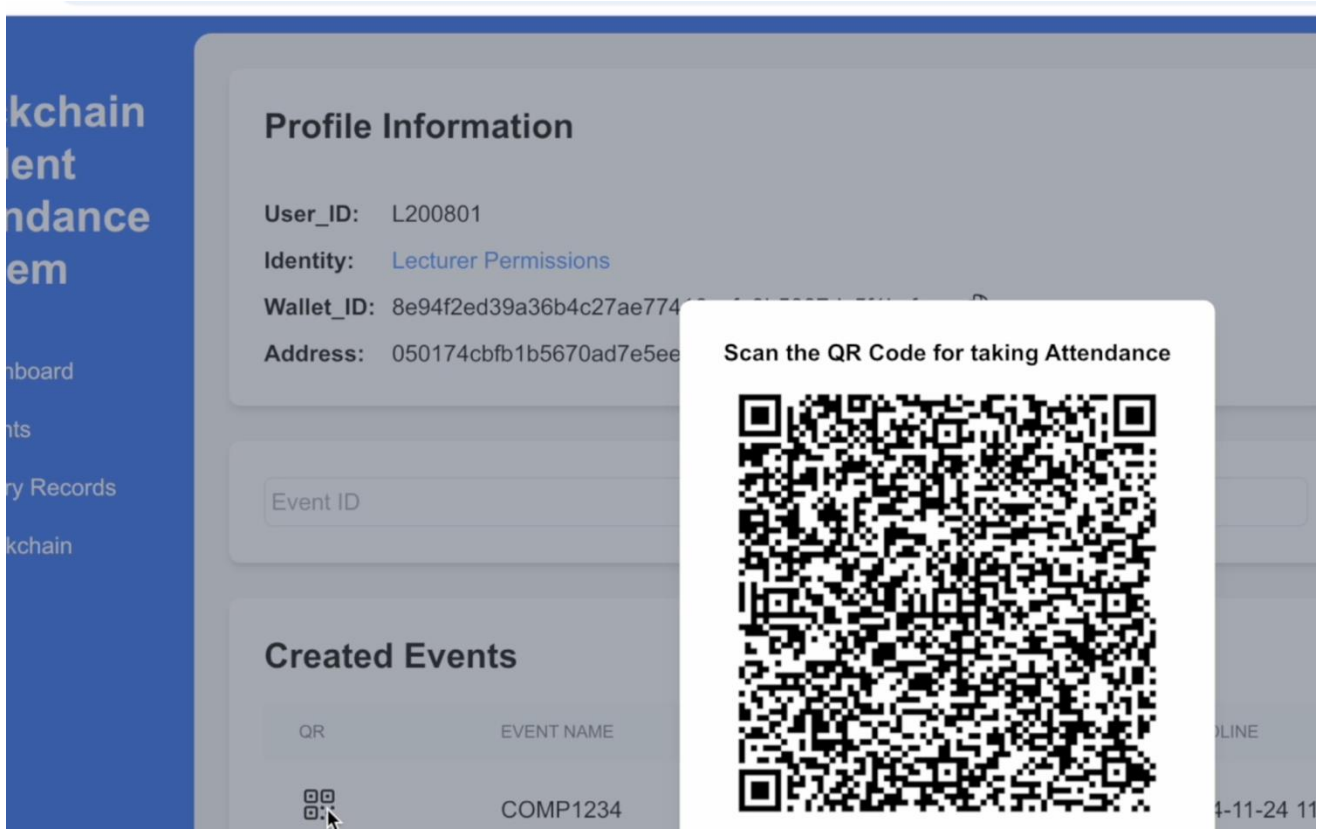
Design logic :

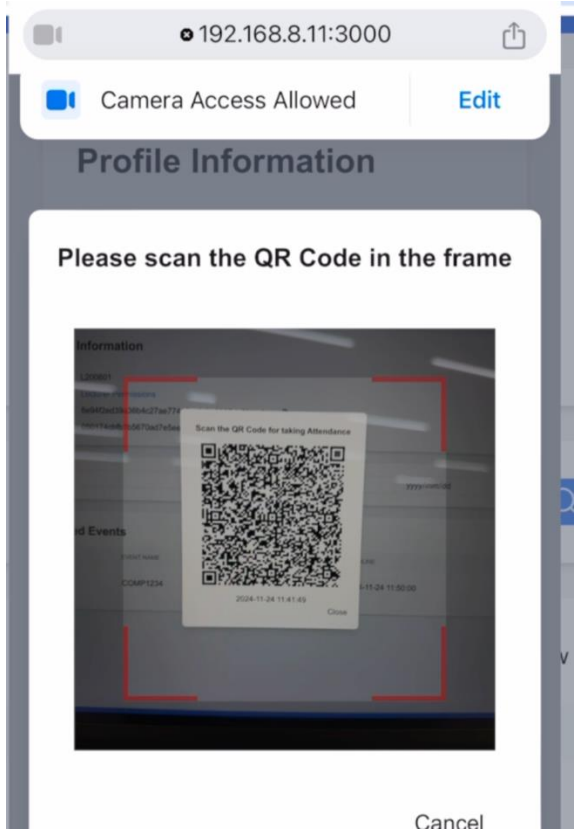
- Compare the **actual generation time** of the block with the **expected time**:

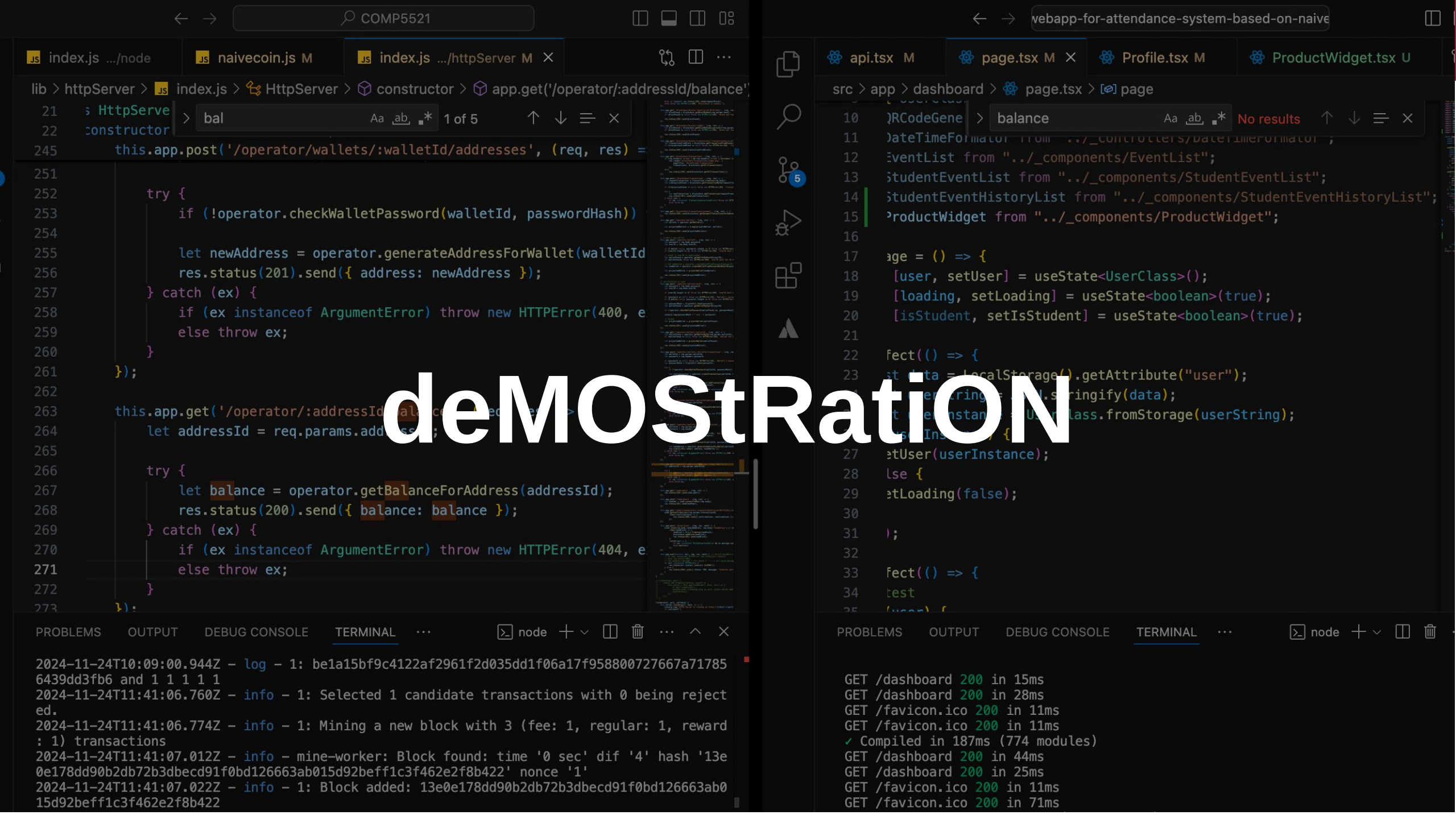
  - If the actual time is **less than** the expected time (block generation is too fast), **increase** the mining difficulty.
  - If the actual time is **greater than** the expected time (block generation is too slow), **reduce** the mining difficulty.

Implementation – Additional Feature

- Design Principles
  - User interface is simple and clean
  - Convenient check-in (Support students to sign in by scanning **QR code**)







deMOStRatiON

ThAnk yOU vEry mUCh