Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

перший (бакалаврський)
(рівень вищої освіти)

Розробка автоматичного контролю робочого часу співробітників на основі QR кодів
(тема)

Виконав:
здобувач 4 року навчання,
групи АКТСІ -21-3
Тимофій ЧЕРЕДНІЧЕНКО
(власне ім'я, прізвище)
Спеціальність 51 Автоматизація та комп`ютерно-інтегровані технології
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Системна інженерія
(повна назва освітньої програми)
Керівник Олександр ЦИМБАЛ
(посада, власне ім'я, прізвище)

Допускається до захисту
Зав. кафедри КІТАР _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Я, Чередніченко Тимофій Олександрович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на віповідне джерело.

24.06.2025  Тимофій ЧЕРЕДНІЧЕНКО

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет АКТ
Кафедра КІТАР
Рівень вищої освіти перший (бакалаврський)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва)
Тип програми Освітньо-професійна
Освітня програма Системна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
« ___»_____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Чередніченко Тимофію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка автоматичного контролю робочого часу співробітників на основі QR кодів
Затверджена наказом по університету від 19.05.2025 р. No 391 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 13.06 2025 р.
3. Вихідні дані до роботи
3.1 Веб-технології: HTML5, CSS3, JavaScript, Vue.js, Vuetify
3.2 Серверна частина: Node.js
3.3 База даних: SQLite
3.4 Сторонні бібліотеки: SweetAlert2, Vue-router, QR Code generator libraries
4. Перелік питань, що потрібно опрацювати в роботі
4.1 Аналіз існуючих рішень для обліку робочого часу та визначення їхніх переваг і недоліків.
4.2 Обґрунтування вибору технологій та інструментів для реалізації системи
4.3 Розробка архітектури системи з використанням QR-кодів
4.4 Реалізація веб-інтерфейсу на базі Vue.js і Vuetify

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

Демонстраційний матеріал, представлений у форматі презентації PowerPoint (*.ppt) – 12 с. формату А4

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Приміт-ка
1	Актуальність роботи, постановка задачі	21.09.24 – 30.09.24	виконано
2	Визначення мети, предмету та об'єкту розробки	30.09.24 – 30.10.24	виконано
3	Аналіз літературних джерел. Аналіз існуючих систем для контролю робочого часу	30.10.24 – 15.11.24	виконано
4	Вибір технології для розробки системи	15.11.24 – 25.11.24	виконано
5	Розробка вимог до системи	25.11.24 – 30.11.24	виконано
6	Проектування архітектури системи	01.12.24 – 20.12.24	виконано
7	Моделювання системи контролю робочого часу	20.12.24 – 10.05.25	виконано
8	Розробка програмного забезпечення для системи	11.05.25 – 01.05.25	виконано
9	Тестування та налагодження системи	01.05.25 – 08.05.25	виконано
10	Оформлення пояснювальної записки	08.05.25 – 20.05.25	виконано
11	Подання роботи на нормоконтроль	10.06.25	виконано

Дата видачі завдання 04.20.2025 р.

Здобувач _____ Тимофій ЧЕРЕДНЧЕНКО
(підпис) (посада, власне ім'я, прізвище)

Керівник роботи _____ професор Олександр ЦИМБАЛ
(підпис) (власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 62 с., 1 табл., 53 рис., 1 дод., 15 джерел.

КОМП’ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ, ЗОБРАЖЕННЯ, СИСТЕМА АВТОМАТИЗАЦІЇ, QR КОДИ, АВТОМАТИЗАЦІЯ ОБРОБКИ.

Мета роботи – розробка системи для автоматизованого контролю робочого часу співробітників за допомогою QR кодів, з реалізацією адмін-панелі для керування користувачами та моніторингу даних.

Об’єкт розробки – процес автоматизованого обліку робочого часу співробітників через сканування QR кодів із подальшим зберіганням даних у базі даних.

Предмет розробки – комп’ютерна система для контролю робочого часу співробітників.

В роботі проведено аналіз сучасних підходів до автоматизації обліку робочого часу. Реалізована система, яка включає такі функції, як реєстрація, авторизація користувачів, контроль входу/виходу на робоче місце, та зберігання даних про робочі години. Також було реалізовано адмін-панель для адміністрування та моніторингу інформації про робочий час співробітників.

Зокрема, в роботі використовуються фреймфорки Vue + Vuetify для розробки інтерфейсу користувача, Node.js для серверної логіки та SQLite як база даних. Окрім цього, розроблено механізм інтеграції QR кодів для автоматизації процесу реєстрації та контролю робочих годин.

Завдяки створеній системі забезпечується точність та автоматизація обліку робочого часу співробітників, що сприяє підвищенню ефективності управління персоналом і зменшенню можливості помилок або шахрайства.

ABSTRACT

Explanatory note: 62 p., 1 tabl., 53 pic., 1 applications, 15 sources.

COMPUTER-INTEGRATED TECHNOLOGY, IMAGING, AUTOMATION SYSTEM, QR CODES, PROCESSING AUTOMATION.

The purpose of the work is to develop a system for automated control of employee working hours using QR codes, with the implementation of an admin panel for user management and data monitoring.

The object of development is the process of automated recording of working hours of employees through scanning QR codes and further storing data in the database.

The subject of the development is a computer system for monitoring the working hours of employees.

The work carried out an analysis of current approaches to automation of work hours. The development of the system is described, which includes such functions as registration, authorization of employees, control of entry/exit at the workplace, and saving data about workdays in the database. An admin panel was also implemented for administering and monitoring information about the working hours of employees.

The project uses Vue + Vuetify frameworks for developing the user interface, Node.js for server logic and SQLite as a database. In addition, a mechanism for integrating QR codes has been developed to automate the registration process and control of working hours.

Once the system is created, the accuracy and automation of the work hours of the employees will be ensured, which will result in increased efficiency of personnel management and a reduced possibility of pardons or fraud.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних;

API – Application Programming Interface;

DB – DataBase;

NFC – Near Field Communication;

RFID – Radio Frequency Identification;

JS – мова програмування JavaScript;

QR – Quick Response;

SQL – Structured Query Language.

ВСТУП

У сучасному світі, де технології стрімко розвиваються, автоматизація процесів стає невід'ємною частиною ефективного управління організаціями. Одним із ключових аспектів, що впливають на продуктивність компаній, є облік робочого часу співробітників. Традиційні методи контролю, такі як паперові журнали чи ручне введення даних, є застарілими, схильними до помилок і потребують значних ресурсів для обробки. Саме тому розробка системи автоматичного контролю робочого часу співробітників на основі QR-кодів є актуальною та перспективною темою в умовах цифрової трансформації.

Сьогодні облік робочого часу є невід'ємною частиною управління персоналом у будь-якій організації, незалежно від її розміру чи сфери діяльності. Ефективний контроль дозволяє не лише фіксувати час приходу та уходу співробітників, але й аналізувати продуктивність, оптимізувати розподіл ресурсів, виявляти порушення трудової дисципліни та підвищувати загальну ефективність роботи.

Крім того, актуальність теми зумовлена зростанням популярності мобільних технологій. Сучасні смартфони, які є у більшості співробітників, дозволяють швидко та зручно сканувати QR-коди, що робить цей метод універсальним і простим у використанні. Поєднання QR-кодів із веб-технологіями, такими як Vue.js та Vuetify, забезпечує створення адаптивних, користувацьки орієнтованих інтерфейсів, які відповідають сучасним стандартам розробки програмного забезпечення. Використання легкої бази даних SQLite додатково підвищує доступність системи, оскільки вона не вимагає складного серверного обладнання та може бути легко впроваджена в малих і середніх підприємствах.

В Україні, де економічні умови часто обмежують можливості компаній інвестувати у дорогі рішення, такі системи є особливо актуальними. Вони

дозволяють оптимізувати процеси управління персоналом, зменшити витрати на адміністрування та забезпечити прозорість у фіксації робочого часу.

Наукова цінність роботи полягає у дослідженні можливостей інтеграції QR-кодів із сучасними веб-технологіями для створення ефективних систем обліку робочого часу. Робота демонструє, як доступні та прості інструменти можуть бути використані для вирішення складних управлінських завдань, а також пропонує нові підходи до організації подвійної авторизації та обробки даних. Практична цінність полягає у створенні готового до впровадження програмного продукту, який може бути використаний у реальних організаціях для підвищення ефективності управління персоналом, зниження адміністративних витрат і забезпечення прозорості обліку робочого часу.

Мета роботи – розробка системи для автоматизованого контролю робочого часу співробітників за допомогою QR кодів, з реалізацією адмін-панелі для керування користувачами та моніторингу даних.

Об’єкт розробки – процес автоматизованого обліку робочого часу співробітників через сканування QR кодів із подальшим зберіганням даних у базі даних.

Предмет розробки – комп’ютерна система для контролю робочого часу співробітників.

Кваліфікаційна робота виконана згідно ДСТУ 3008 – 15 [1] та керуючись методичними вказівками з написання кваліфікаційної роботи бакалавра [2].

Отримані результати роботи можна віднести до Цілі сталого розвитку 9 “Промисловість, інновації та інфраструктура”, а саме п. 9.4 “Сприяти 12 прискореному розвитку високо- та середньо-високотехнологічних секторів переробної промисловості, які формуються на основі використання ланцюгів «освіта – наука – виробництво» та кластерного підходу за напрямами: розвиток інноваційної екосистеми”, індикатор 9.4.1.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У сучасних умовах цифровізації бізнес-процесів автоматизований контроль робочого часу співробітників стає одним з ключових елементів ефективного управління персоналом. За даними [6], точний облік часу дозволяє зменшити втрати на «мертвий час», підвищити дисципліну працівників і забезпечити прозорість в управлінні трудовими ресурсами. Проте, більшість існуючих рішень вимагають значних фінансових витрат на інфраструктуру, зокрема біометричні системи, RFID-картки чи спеціалізовані термінали.

Унікальність цієї розробки полягає у створенні комплексної системи обліку робочого часу, що поєднує в собі інноваційність, доступність та адаптованість до локальних умов. На відміну від традиційних систем, де переважає орієнтація на великі корпорації (наприклад, Kronos Workforce Ready [7]), розробка, представлена в цій роботі, передбачає використання QR-кодів, що дозволяє значно знизити витрати на впровадження та експлуатацію системи.. Основні аспекти унікальності системи:

– подвійна авторизація через QR-коди: на відміну від рішень, що використовують одноразове сканування (Kimai, OpenTimeClock), система передбачає двоетапний процес – сканування на вході та повторне підтвердження після проходження турнікету чи дверей. Це дозволяє мінімізувати ймовірність шахрайства, такого як передача коду іншій особі, і підвищити достовірність обліку фізичної присутності. Подібна функціональність відсутня у більшості популярних SaaS-рішень, що підтверджується дослідженнями [8].

– мобільна авторизація: Завдяки використанню вже наявної інфраструктури – особистих смартфонів працівників – система усуває необхідність у додатковому обладнанні. Це є значною перевагою порівняно з системами, де використовується спеціалізоване обладнання або біометричні

сканери, що потребують складного обслуговування [11];

– сучасний веб-інтерфейс на основі Vue.js та Vuetify: використання популярного JavaScript-фреймворку Vue.js у поєднанні з бібліотекою компонентів Vuetify забезпечує створення адаптивного, інтуїтивно зрозумілого інтерфейсу, який працює на різних пристроях – від десктопів до смартфонів. Це підвищує зручність використання як для співробітників, так і для адміністраторів [3-5];

– легка база даних SQLite: це дозволяє створювати компактні та швидкі рішення, які не потребують складного серверного обладнання. Це особливо важливо для малих і середніх підприємств, де обмежені ресурси для адміністрування баз даних. SQLite забезпечує достатню продуктивність для обробки даних про робочий час і легко масштабується при необхідності;

– функціональна адміністративна панель: система включає зручну адмін-панель, яка дозволяє керівникам і HR-менеджерам керувати даними, генерувати звіти, аналізувати статистику та налаштовувати параметри системи. Це робить рішення гнучким і придатним для організацій різного масштабу;

– безпека та захист даних: система передбачає авторизацію через мобільний додаток або веб-інтерфейс, що забезпечує захист від несанкціонованого доступу. Використання сучасних методів шифрування та аутентифікації дозволяє відповідати вимогам до безпеки даних, що є критично важливим у сучасних умовах;

– орієнтація на локальний контекст: система розроблена з урахуванням потреб українських компаній, які часто стикаються з обмеженим бюджетом і необхідністю швидкого впровадження ефективних рішень. Вона є економічно вигідною альтернативою дорогим системам і може бути адаптована до різних галузей – від офісів до промислових підприємств;

– інтеграція та масштабованість: завдяки модульній архітектурі та використанню сучасних технологій система може бути легко інтегрована з іншими інформаційними системами, такими як ERP чи CRM, а також

модифікована для додавання нових функцій, наприклад, аналітики продуктивності чи інтеграції з платіжними системами;

– унікальність роботи також полягає в її практичній спрямованості. Система не лише відповідає теоретичним вимогам до автоматизації обліку робочого часу, але й пропонує готове рішення, яке може бути впроваджене в реальних організаціях. Це робить її цінною як для наукових досліджень, так і для практичного застосування.

– практична спрямованість: у той час як більшість досліджень зосереджуються на теоретичному аналізі систем контролю (наприклад, у [8]), ця робота пропонує реальну, функціональну реалізацію, придатну для впровадження в організаціях.

Загалом, системи обліку робочого часу можна умовно класифікувати на декілька типів: біометричні, карткові (RFID/NFC), веб-орієнтовані SaaS-рішення та мобільні додатки з геолокацією або QR-ідентифікацією. Як відзначає Miller [15], жоден із підходів не є універсальним: вибір залежить від цілей організації, бюджету та організаційної структури. QR-коди, у цьому контексті, представляють компроміс між простотою реалізації, низькими витратами та відносно високим рівнем надійності при правильній реалізації.

Завдяки своїй відкритості та доступності, QR-коди стали ефективним інструментом верифікації не лише в роздрібній торгівлі чи логістиці, а й у HR-системах. Як зазначено у дослідженні Smith і Jones [8], впровадження QR-систем може зменшити витрати на апаратну частину в 5–7 разів порівняно з біометричними пристроями, а також знизити бар’єр входу для малого та середнього бізнесу.

Додатковою перевагою є можливість інтеграції з іншими засобами цифрового управління, зокрема ERP-системами. У контексті цифрової трансформації підприємств, поєднання систем контролю часу із загальною інформаційною архітектурою організації дозволяє забезпечити наскрізну аналітику та автоматизувати обробку даних для розрахунку заробітної плати, виявлення неефективностей або оцінювання продуктивності персоналу [7].

Крім того, важливою характеристикою ефективної системи обліку часу є її масштабованість. Як зазначає Hipp [13], використання легковагих реляційних баз даних, таких як SQLite, дає змогу швидко запускати MVP-рішення та легко переносити дані у більш потужні системи за потреби. Подібний підхід особливо актуальний для стартапів та компаній, що швидко зростають і не можуть дозволити собі складну IT-інфраструктуру на старті.

Особливу роль у сучасних системах відіграє юзабіліті. Як відзначає Choi у своїй праці [12], навіть найбільш функціонально повноцінне рішення буде неефективним без зручного інтерфейсу. Тому важливим трендом є адаптація систем під мобільні пристрої та використання сучасних фронтенд-фреймворків, таких як Vue.js. Завдяки цьому забезпечується інтуїтивна взаємодія користувача з системою навіть без попереднього навчання.

Ще одним важливим аспектом є кібербезпека. Згідно з OWASP Top Ten [14], до основних загроз для веб-застосунків належать слабка автентифікація, неправильне керування сеансами, ін’єкції та витоки конфіденційної інформації. У зв’язку з цим, надійні рішення повинні реалізовувати сучасні механізми захисту: шифрування даних, токенізацію, контроль доступу та журнали активності. У випадку із QR-ідентифікацією критично важливо обмежити можливість багаторазового використання одного й того ж коду, що і вирішується через реалізацію двоетапної авторизації.

Узагальнюючи, можна сказати, що впровадження автоматизованих систем контролю робочого часу на базі QR-кодів є логічним кроком в умовах цифрової еволюції підприємств. Такий підхід поєднує в собі:

– мінімальні витрати на обладнання та обслуговування;
– високу адаптивність до змінних умов праці;
– відповідність сучасним вимогам до безпеки та конфіденційності;
– зручність використання для кінцевих користувачів;
– можливість швидкої інтеграції з іншими системами управління підприємством.

У рамках аналізу подібних розробок було виявлено наступні рішення,

що використовують QR-коди для контролю відвідуваності або обліку робочого часу (таблиця 1.1).

Таблиця 1.1 – Порівняння з іншими системами

Система	QR	Мобільна верифікація	Інтерфейс	Ціна/Модель
OpenTimeClock	Так	Так, одноетапна	Web/Mobile	Saas + FOSS
QRCode-scan	Так	Ні	Web	Open-Source
Kimai	Так	Так, одноетапна	Web/Mobile	Open-Source
IceHrm	Так	Так, одноетапна	Web/Mobile	Open-Source SaaS
Цей проект	Так	Так, двоетапна	Vue.js (Web/Mobile)	Open-Source, локальна

Для більшого розуміння роботи системи, можна ознайомитись зі схемою на рисунку 1.1.

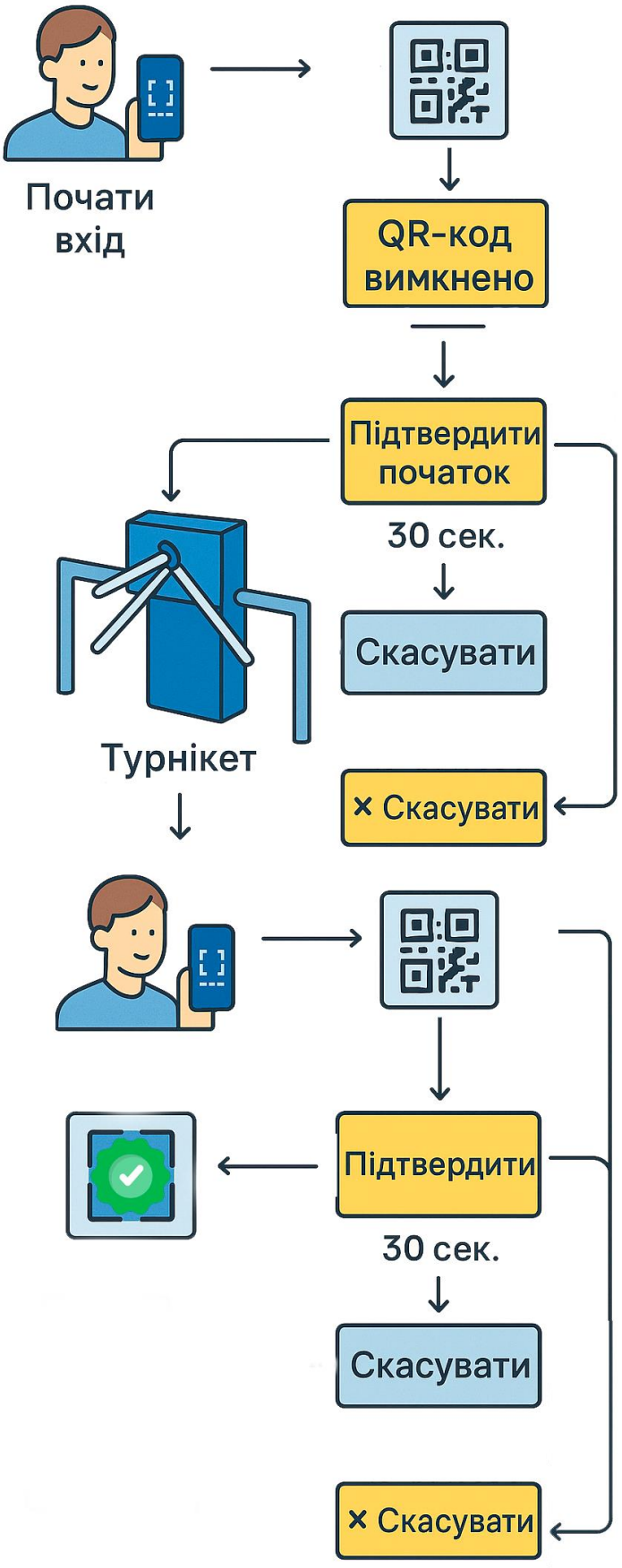


Рисунок 1.1 – Схема рооти системи контролю управління доступом

2 РОЗРОБКА ОСНОВНОГО ІНТЕРФЕЙСУ

Система, яка розробляється в роботі функціонує за чітко продуманим алгоритмом, який враховує специфіку роботи підприємства та потреби безпеки. На вході до підприємства, перед турнікетом, розташовано спеціальне табло, що генерує унікальний QR-код. Співробітник, який попередньо пройшов авторизацію в системі, сканує цей код за допомогою мобільного пристрою. Після сканування код деактивується, запобігаючи повторному використанню. Далі співробітнику надається 30 секунд для підтвердження початку робочого дня через спеціальний інтерфейс на його пристрої. Після цього відкривається доступ до другого табло, розташованого вже за турнікетом, де потрібно просканувати ще один QR-код протягом наступних 30 секунд. Це гарантує, що працівник реально пройшов через турнікет і знаходиться на території підприємства, унеможливлюючи ситуації, коли особа може позначити початок робочого дня, але не приступити до виконання обов’язків [6-7].

Для демонстраційних цілей у системі передбачено можливість емуляції сканування QR-кодів за допомогою спеціальних кнопок, які імітують процес сканування шляхом відкриття відповідних посилань. У реальних умовах такі кнопки відсутні, а сканування здійснюється виключно через QR-коди, що підвищує рівень безпеки та достовірності фіксації. Система авторизації користувачів також є невід’ємною частиною розробки. Вона побудована таким чином, щоб унеможливити доступ неавторизованих осіб. Наприклад, спроба сканування QR-коду без попередньої авторизації завершується помилкою, а повторне сканування одного й того самого коду неможливе завдяки механізму деактивації.

Особливу увагу приділено таймерам, які відіграють ключову роль у

забезпеченні коректної роботи системи. Якщо співробітник не підтверджує дію протягом відведеного часу або натискає кнопку скасування, система автоматично скидає QR-коди, дозволяючи розпочати процес заново. Це забезпечує гнучкість у використанні та запобігає накопиченню помилок у процесі фіксації робочого часу. Окрім того, система включає функціонал для обліку перерв, що дозволяє працівникам гнучко планувати свій робочий день, зберігаючи при цьому точність даних про їхню присутність [8-9].

Важливим елементом системи є механізм реєстрації користувачів, який також має високий рівень захисту. Для створення облікового запису співробітник повинен отримати спеціальний токен, згенерований адміністратором через адмін-панель. Без цього токена реєстрація неможлива, що виключає можливість несанкціонованого доступу до системи сторонніми особами. Адмін-панель, у свою чергу, надає розширені можливості для управління доступом: адміністратор може створювати, блокувати або змінювати статуси облікових записів працівників, а також контролювати їхню активність у системі.

Система враховує як потреби працівників, так і вимоги адміністрації підприємства, створюючи оптимальні умови для точного та безпечного обліку робочого часу. Завдяки використанню QR-кодів, сучасних методів авторизації та гнучких таймерів система є універсальним рішенням, яке може бути адаптоване до потреб різних організацій, незалежно від їхньої специфіки чи масштабів.

Система легко адаптується до різних підприємств, забезпечуючи контроль доступу, безпеку та точний облік часу [10].

Головне табло демонструє два QR-коди, розташовані поруч – перший з боку входу, а другий з боку виходу (за турнікетом). Це зроблено навмисно: система використовує двоступеневу авторизацію, яка гарантує, що працівник справді проходить фізично через прохідну. Після сканування першого коду (перед турнікетом) співробітник має авторизуватись на мобільному пристрої. Лише після цього система дозволяє відсканувати другий код (після турнікету),

завершуючи реєстрацію події (вхід, вихід або перерву). Такий підхід унеможливлює фіктивне сканування чи передачу коду іншій особі.

На екрані відображається інтерфейс із двома великими картками QR-кодів (рис. 2.1). Якщо код уже просканово або ще не згенерований – він стає розмитим (рис. 2.2), а поверх нього з’являється текст (наприклад: «Цей QR вже просканували» або «Чекайте на вибір співробітника»). Самі QR-коди генеруються динамічно та оновлюються кожні 15 хвилин для підвищення безпеки, щоб уникнути повторного використання старих токенів.

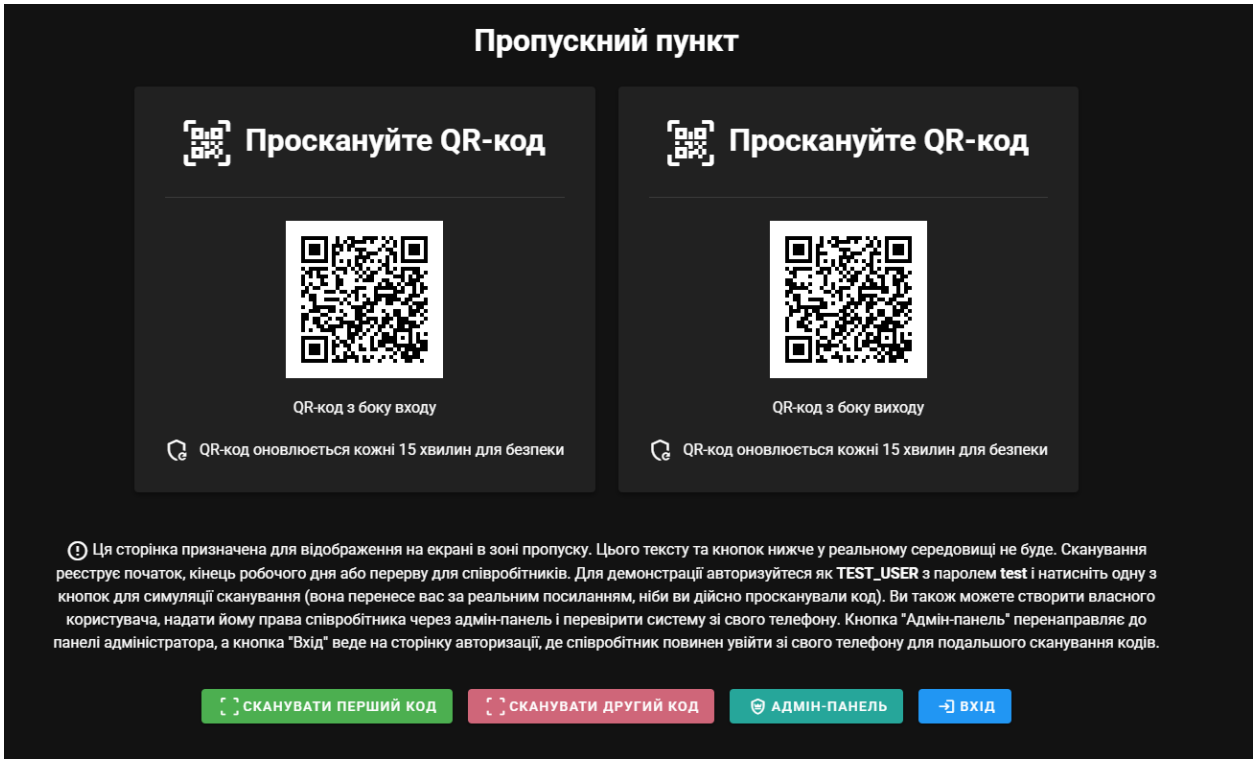


Рисунок 2.1 – Інтерфейс головного екрану

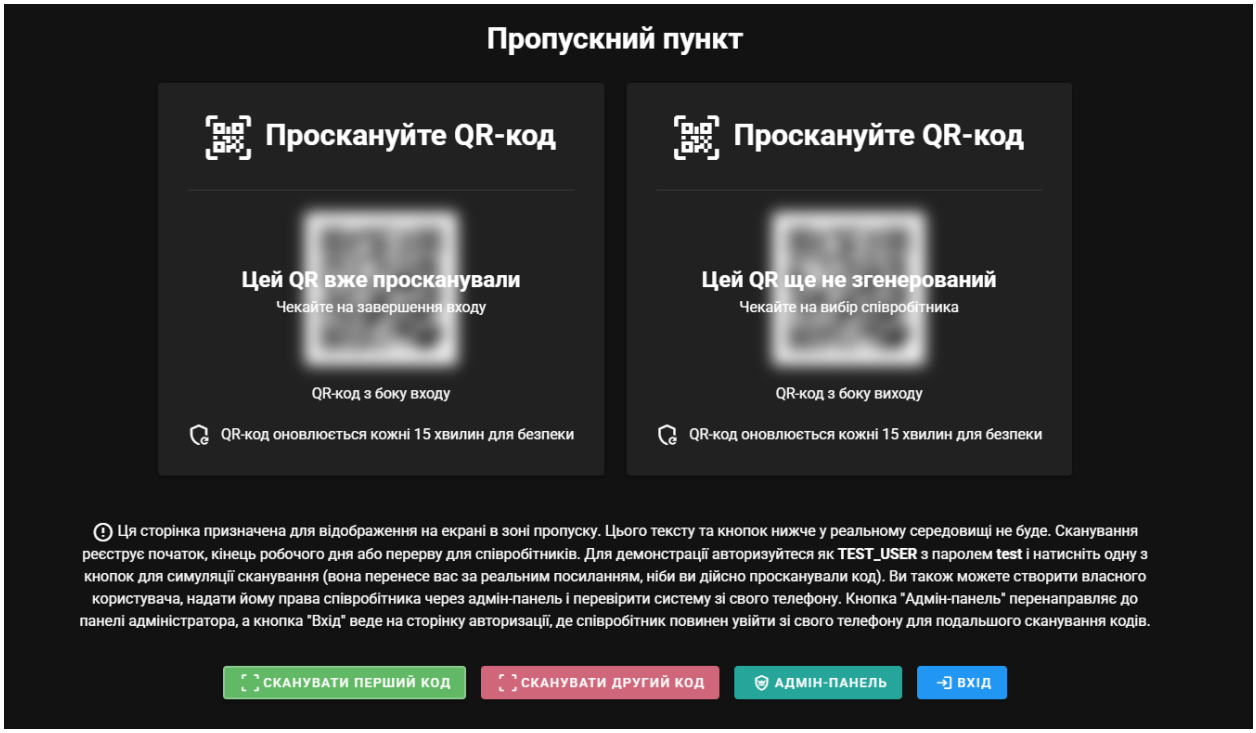


Рисунок 2.2 – Приклад розмиття

Після того як співробітник сканує перший QR-код зі свого мобільного пристрою, система переходить до етапу авторизації. На екрані смартфона з’являється привітальне повідомлення та дві інтерактивні кнопки: «Розпочати робочий день» та «Скасувати» (рис. 2.3). Одночасно запускається таймер зворотного відліку на 30 секунд, протягом яких працівник має підтвердити свій вибір.

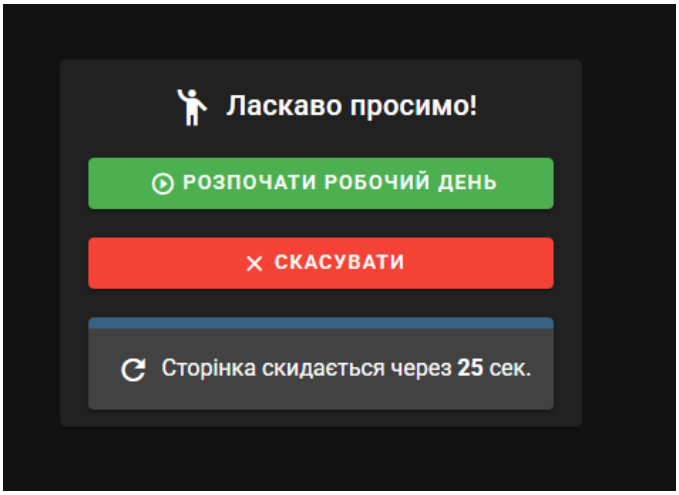


Рисунок 2.3 – Інтерфейс вибору дії

Якщо користувач натискає «Почати робочий день», другий QR-код, розташований за турнікетом, стає чітким і доступним для сканування (рис. 2.4). На телефоні також з’являється повідомлення про те, що працівник має 30 секунд, щоби завершити процедуру, просканувавши другий код (рис. 2.5).

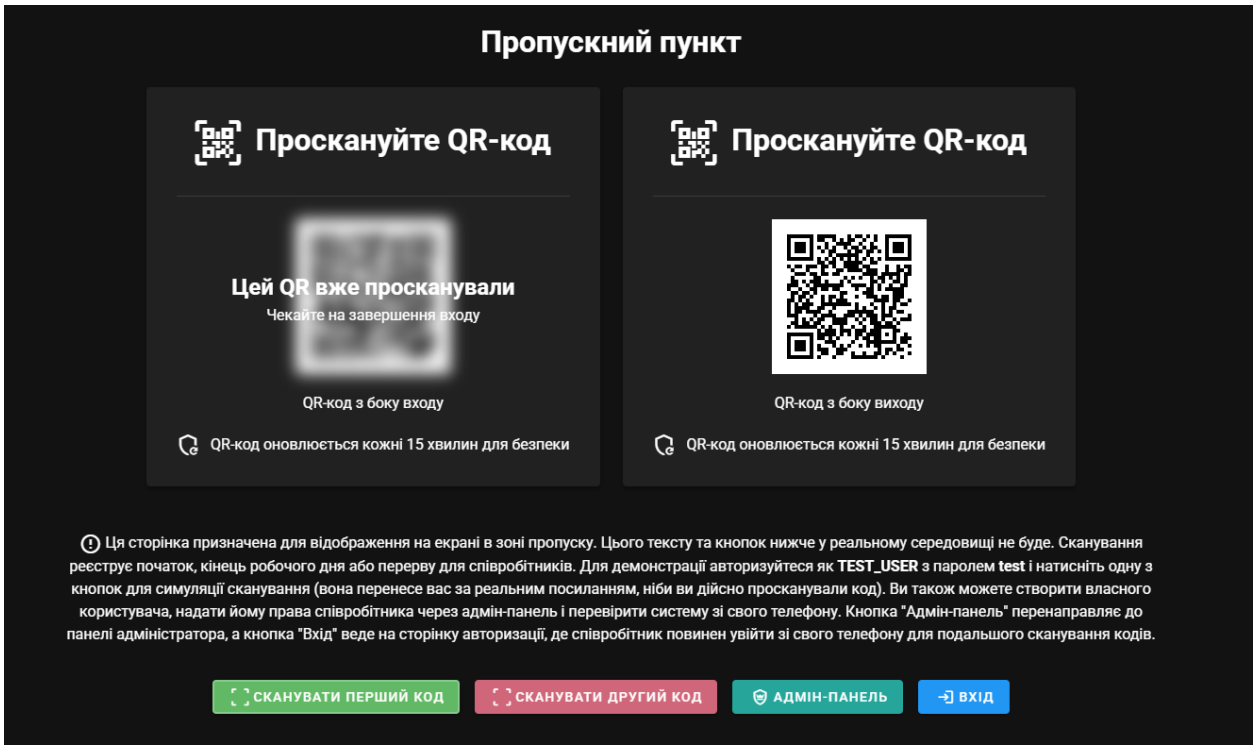


Рисунок 2.4 – Очікування сканування другого QR коду

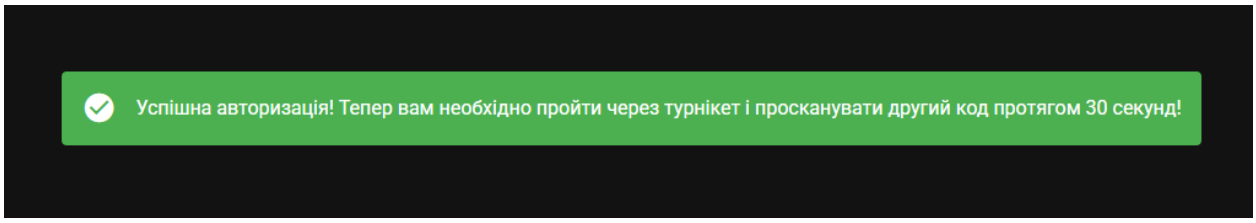


Рисунок 2.5 – Повідомлення для працівника

У випадку, якщо працівник не зробив вибору протягом 30 секунд або натиснув кнопку «Скасувати», система автоматично скидає авторизацію, і обидва QR-коди на табло деактивуються. Процес потрібно почати з початку.

Усі дії супроводжуються інтерактивними повідомленнями (рис. 2.6 та 2.7), які чітко інформують користувача про стан системи. Якщо, наприклад,

співробітник уже авторизований, не має прав доступу, або намагається просканувати QR-код після завершення відведеного часу – сайт повідомляє про це через повідомлення на екрані. Такий підхід забезпечує зручність і мінімізує помилки під час входу на робоче місце.

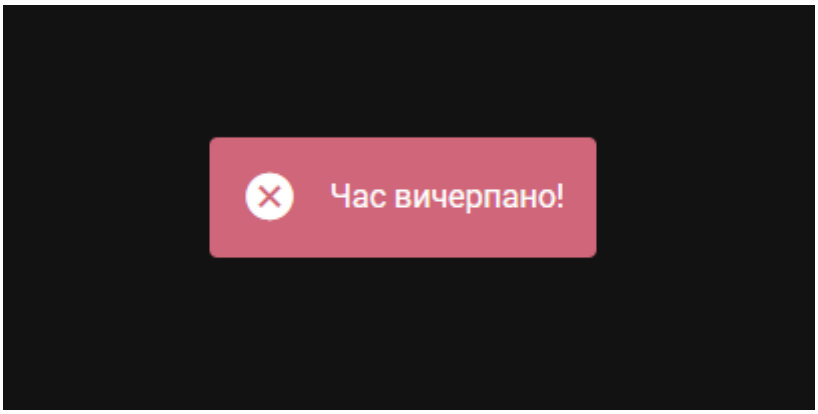


Рисунок 2.6 – Повідомлення про вичерпання часу

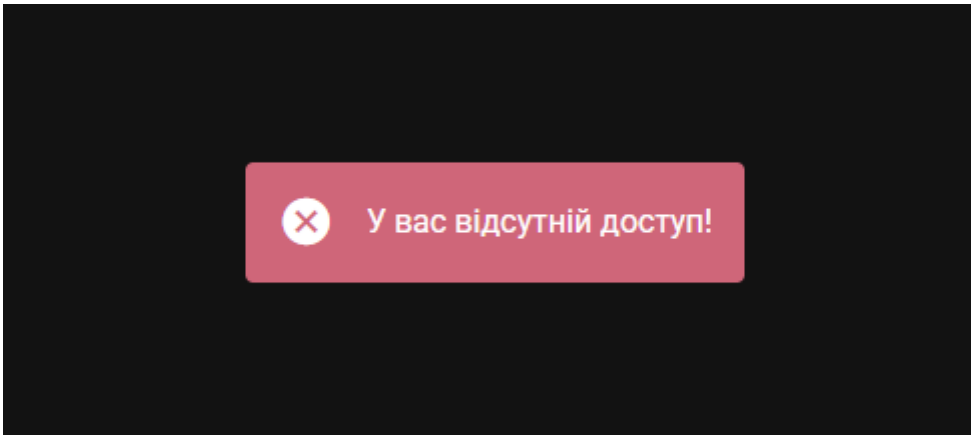


Рисунок 2.7 – Повідомлення про відсутність доступу

Алгоритм роботи системи такий:

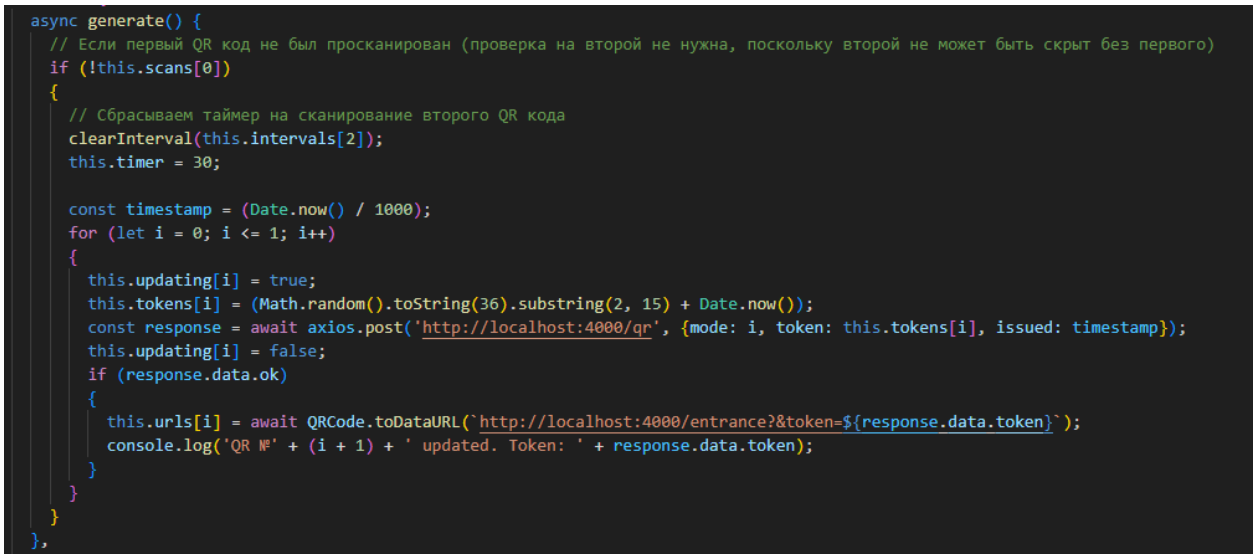
– ініціалізація: при завантаженні сторінки генеруються два QR-коди для табло перед і після турнікета (рис. 2.8). Коди оновлюються кожні 15 хвилин для безпеки;

– сканування першого QR-коду: співробітник сканує QR-код перед турнікетом за допомогою мобільного пристрою. Код деактивується (рис. 2.9), і запускається 30-секундний таймер для підтвердження входу;

– підтвердження входу: співробітник авторизується в системі через мобільний додаток і підтверджує початок робочого дня;

– сканування другого QR-коду: після проходження турнікета співробітник сканує другий QR-код протягом 30 секунд, що фіксує його присутність на території підприємства;

– оновлення та безпека: система кожну секунду перевіряє (рис. 2.10) стан QR-кодів через сервер, синхронізуючи дані та деактивуючи використані коди. Якщо 30-секундний таймер закінчується без сканування другого коду, система скидає стан і генерує нові QR-коди.



```
async generate() {
  // Если первый QR код не был просканирован (проверка на второй не нужна, поскольку второй не может быть скрыт без первого)
  if (!this.scans[0])
  {
    // Сбрасываем таймер на сканирование второго QR кода
    clearInterval(this.intervals[2]);
    this.timer = 30;

    const timestamp = (Date.now() / 1000);
    for (let i = 0; i <= 1; i++)
    {
      this.updating[i] = true;
      this.tokens[i] = (Math.random().toString(36).substring(2, 15) + Date.now());
      const response = await axios.post('http://localhost:4000/qr', {mode: i, token: this.tokens[i], issued: timestamp});
      this.updating[i] = false;
      if (response.data.ok)
      {
        this.urls[i] = await QRCode.toDataURL(`http://localhost:4000/entrance?&token=${response.data.token}`);
        console.log('QR №' + (i + 1) + ' updated. Token: ' + response.data.token);
      }
    }
  }
},
```

Рисунок 2.8 – Генерація QR кодів

```
async update() {
  if (!this.updating[0] && !this.updating[1])
  {
    const response = await axios.post('http://localhost:4000/update');

    // Если токен первого QR кода равен NULL
    if (response.data[0].token === null)
    {
      // Когда обработка пользователя полностью заверша, оба параметра user_id, совместно с токеном QR кода будут равны NULL
      if ((response.data[0].user_id === null) && (response.data[1].user_id === null))
      {
        console.log('[Update] Generating new QR codes...');

        this.scans = [false, false];
        this.generate();
      }
      // Если просканирован первый QR код и ожидается сканирование второго
      else if ((response.data[0].user_id === null) && (response.data[1].user_id !== null))
      {
        // На всякий случай скрываем оба кода
        if (!this.scans[0])
        {
          this.scans = [true, true];
        }
```

Рисунок 2.9 – Обробка сканування першого QR коду

```
    },
    start() {
      this.intervals[2] = setInterval(() => {
        this.timer--;
        if (this.timer <= 0) {
          clearInterval(this.intervals[2]);
          this.timer = 30;

          this.scans = [false, false];
          this.updating[0] = true;
          console.log('[Start] Generating new QR codes...');
          this.generate();
        }
      }, 1000);
    }
  },
  mounted() {
    this.generate();
    this.intervals = [setInterval(this.generate, (15 * 60 * 1000)), setInterval(this.update, 1000), null];
  },
  beforeUnmount() {
    for (let i = 0; i < this.intervals.length; i++)
    {
      if (this.intervals[i] !== null)
      {
        clearInterval(this.intervals[i]);
      }
    }
  }
```

Рисунок 2.10 – Реалізація таймеру перевірки

3 РОЗРОБКА СТОРІНОК ТА ЛОГІКИ СКАНУВАНЬ

Після того як користувач сканує перший QR-код, його мобільний пристрій автоматично перенаправляється на сторінку підтвердження входу. У параметрах URL-посилання передається токен QR-коду, а токен користувача (тобто сесійний токен) підтягується з локального сховища браузера (рис. 3.1). Це дозволяє системі ідентифікувати користувача і визначити, чи має він право доступу. Після завантаження сторінки викликається метод load(), який відправляє POST-запит на сервер (/scan) з параметрами токена користувача та токена коду. Якщо авторизація пройшла успішно – обробляється відповідь, з якої зчитується статус користувача, час видачі коду (issued) та активується таймер, що відраховує 30 секунд.

```
async load() {
  const response = await axios.post('http://localhost:4000/scan', {mode: 0, token: this.token, code: this.$route.query.token});
  console.log(response.data);

  if (response.data.ok) {
    this.authorized = true;
    this.issued = response.data.issued;
    this.status = response.data.status;
    this.start();
  }
  else
  {
    if (response.data.error) {
      this.authorized = response.data.error;
    }
    else {
      this.authorized = null;
    }
  }
```

Рисунок 3.1 – Перевірка сканування

Натискання кнопки «Розпочати» викликає метод day (рис. 3.2), який надсилає POST-запит на /entrance з параметром type:1. Якщо сервер підтверджує запит (ok: true), система встановлює статус авторизації як Success. – і на екрані з’являється зелена підказка, яка сповіщає користувача про те, що другий QR-код активований, і його потрібно просканувати протягом 30 секунд.

```
async day() {
  clearInterval(this.interval);
  this.timer = 0;

  const response = await axios.post('http://localhost:4000/entrance', {token: this.token, type: 1});
  console.log(response.data);
  if (response.data.ok) {
    this.authorized = 'Success.';
  }
},
async load() {
```

Рисунок 3.2 – Відправка запросу для початку робочого дня

У разі, якщо користувач не натисне кнопку або просрочить час, спрацює метод reset (рис. 3.3), який:

– обнуляє таймер;

– очищує інтервал оновлення;

– надсилає POST-запит на /entrance з параметром type: 0 (що сигналізує про скидання).

```
},
reset() { // Сброс после истечения 30-ти секунд
  clearInterval(this.interval);
  this.timer = 0;

  axios.post('http://localhost:4000/entrance', {token: this.token, type: 0});
},
start() {
```

Рисунок 3.3 – Сброс сканування

Крім того, якщо користувач натисне «Скасувати», виконується той самий метод reset, але статус авторизації встановлюється як 'Canceled.', після чого на екрані відображається повідомлення «Ви скасували заявку!».

Особливу увагу приділено різним варіантам обробки станів (рис. 3.4).

```
</div>
<div v-else-if="(authorized === null) || (authorized === 'Error retrieving data.')">
  <v-alert type="error" class="text-center">
    Цей код недійсний!
  </v-alert>
</div>
<div v-else-if="(authorized === 'Success.')">
  <v-alert type="success" class="text-center">
    Успішна авторизація! Тепер вам необхідно пройти через турнікет і просканувати другий код протягом 30 секунд!
  </v-alert>
</div>
<div v-else-if="(authorized === 'Access Denied.')">
  <v-alert type="error" class="text-center">
    У вас відсутній доступ!
  </v-alert>
</div>
<div v-else-if="(authorized === 'Canceled.')">
  <v-alert type="error" class="text-center">
    Ви скасували заявку!
  </v-alert>
</div>
<div v-else-if="(authorized === 'Need registration.')">
  <v-alert type="error" class="text-center">
    Вам потрібно зареєструватись у системі!
  </v-alert>
</div>
<div v-else-if="(status === 2)">
  <v-alert type="error" class="text-center">
    Ви вже розпочали робочий день!
  </v-alert>
</div>
<div v-else>
```

Рисунок 3.4 – Обробка станів

Система надійно контролює авторизацію, дозволяє уникнути помилок користувача, підтримує сценарії повторного доступу та обробку виключень. Завдяки зворотному зв’язку через динамічні повідомлення, користувач завжди розуміє, що відбувається на кожному етапі.

Другий QR код, який розміщується вже всередині приміщення, після проходження турнікета. Цей етап необхідний для підтвердження авторизації, розпочатої скануванням першого QR-коду, а також надає співробітнику можливість взяти перерву або завершити робочий день. Якщо людина просто завершує робочий день, то для цього не потрібно повторно сканувати перший код – це зроблено для зручності, адже така дія не несе ризику шахрайства: ніхто не буде працювати понаднормово «безкоштовно», просто щоб закінчити день раніше в системі.

Після завантаження сторінки відбувається перевірка токена користувача, збереженого у localStorage (рис. 3.4), як і на першій сторінці.

```
},
mounted() {
  this.token = localStorage.getItem('token');

  if (this.token)
  {
    setTimeout(() => {
      this.load();
    }, 3000);
  }
},
beforeUnmount() {
  if (this.interval)
  {
    clearInterval(this.interval);
  }
}
```

Рисунок 3.4. – Зчитування даних сесії

Якщо токен відсутній, відображається повідомлення про те, що користувач не авторизован. Якщо токен є, протягом трьох секунд ініціюється запит на сервер на предмет дійсності переданого у параметрах запиту QR-коду. Цей запит містить режим mode: 1, що вказує на те, що обробляється другий QR-код (рис. 3.5).

```
async load() {
  const response = await axios.post('http://localhost:4000/scan', {code: this.$route.query.token, mode: 1, token: this.token});
  console.log(response.data);
  if (response.data.ok)
  {
    this.authorized = true;
    this.issued = response.data.issued;
    this.status = response.data.status;
    this.start();
  }
  else
  {
    if (response.data.error)
    {
      this.authorized = response.data.error;
    }
    else
    {
      this.authorized = null;
    }
  }
},
```

Рисунок 3.5 – Запит на сервер з обробкою другого QR коду

Якщо код дійсний і відповідає активній сесії, отримуються дані з сервера

й відображаються відповідні варіанти взаємодії: «Завершити робочий день» або «Взяти перерву» (рис. 3.6), якщо у користувача активний робочий день. На кожну з цих дій викликається метод action, який ініціює запит на /exit, передаючи дію (end_day або start_break), токен та код. У разі успіху на екрані з'являється повідомлення про завершення робочого дня або початок перерви.

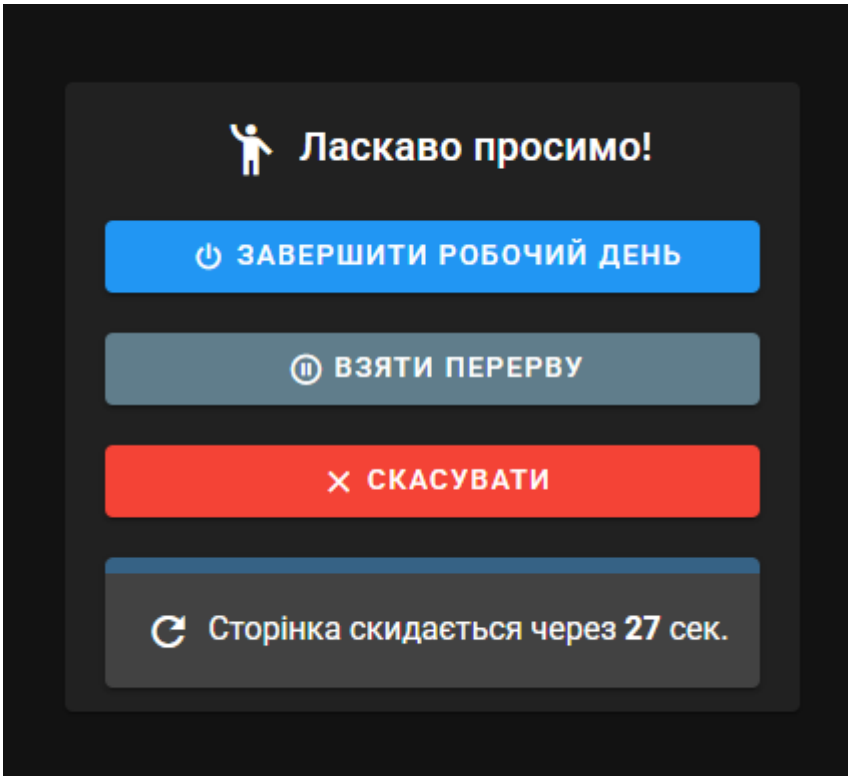


Рисунок 3.6 – Кнопки для користувача з активним робочим днем

На цій сторінці також є таймер, згідно першому QR коду, якщо час вичерпано, сторінка «скидається», а кнопки для взаємодії зникають. Це дозволяє уникнути повторного використання QR-коду, забезпечуючи цілісність авторизації.

Крім того, враховані й інші стани, такі як недійсний код, спроба взаємодії без початку авторизації, відмова у доступі, потреба у реєстрації або скасування запиту (рис. 3.7). Усі ці стани відображаються користувачу через компонент v-alert, завдяки чому інтерфейс залишаєсь зрозумілим та адаптивним до будь-якого сценарію.

```
<div v-else>
  <div v-if="(authorized === false)" class="d-flex flex-column align-center">
    <v-progress-circular indeterminate color="primary" size="70"></v-progress-circular>
    <p class="mt-3">Завантаження...</p>
  </div>
  <div v-else-if="(authorized === null) || (authorized === 'Error retrieving data.')">
    <v-alert type="error" class="text-center">
      Цей код недійсний!
    </v-alert>
  </div>
  <div v-else-if="((authorized === true) && !end_day && !start_break && (status !== 2))">
    <v-alert type="success" class="text-center">
      Ви успішно розпочали робочий день!
    </v-alert>
  </div>
  <div v-else-if="(authorized === 'Access Denied.')">
    <v-alert type="error" class="text-center">
      У вас відсутній доступ!
    </v-alert>
  </div>
  <div v-else-if="(authorized === 'Authorization was not started.')">
    <v-alert type="error" class="text-center">
      Ви не встигли просканувати код або не розпочинали авторизацію!
    </v-alert>
  </div>
  <div v-else-if="(authorized === 'Canceled.')">
    <v-alert type="error" class="text-center">
      Ви скасували заявку!
    </v-alert>
  </div>
  <div v-else-if="(authorized === 'Need registration.')">
    <v-alert type="error" class="text-center">
      Вам потрібно зареєструватись у системі!
    </v-alert>
  </div>
```

Рисунок 3.7 – Обробка станів другого QR коду

4 СИСТЕМА РЕЄСТРАЦІЇ ТА АВТОРИЗАЦІЇ

Система реєстрації та авторизації побудована з урахуванням високого рівня безпеки та контролю з боку адміністратора. Уся логіка побудована навколо токену – унікального коду, який може згенерувати лише адміністратор через панель керування. Саме токен є пропуском до реєстрації: без нього новий користувач не зможе створити акаунт. Під час створення токена адміністратор також призначає посаду користувача, що дозволяє системі розрізняти рівні доступу до різних приміщень або функцій. Після отримання токена користувач має можливість зареєструватися (рис 4.1), вказавши цей токен та свій пароль, який буде збережений у системі у вигляді хешу, а не у відкритому вигляді, що відповідає сучасним стандартам безпеки.

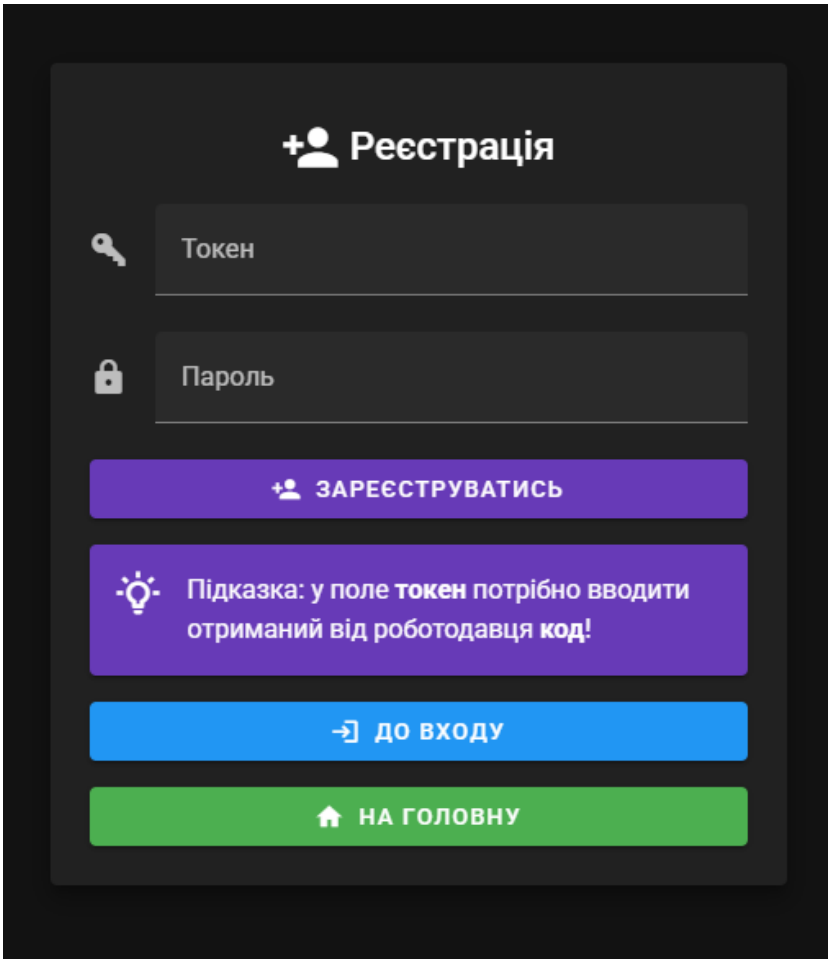


Рисунок 4.1 – Форма реєстрації

Після реєстрації (рис. 4.2), при наступних авторизаціях, система автоматично підтягує ім’я користувача з бази за допомогою API-запиту та зберігає виданий токен у localStorage (рис. 4.3) для подальших перевірок.

```
async register() {
  if (!this.code) {
    return this.error('Потрібно ввести токен!');
  }
  if (!this.password) {
    return this.error('Потрібно ввести пароль!');
  }

  const response = await axios.post('http://localhost:4000/register', {token: this.code, password: this.password});
  if (response.data.ok) {
    this.registration = false;

    this.name = response.data.name;
    this.password = '';
    this.token = response.data.token;
    this.username = '';
  }
  else {
    let error_text = 'Невідома помилка!';

    if (response.data.error == 'already_registered') {
      error_text = 'Користувач вже зареєстрований!'
    }
    else if (response.data.error == 'unknown_token') {
      error_text = 'Такого токена не існує!'
    }

    Swal.fire({
      title: 'Помилка!',
      text: error_text,
      icon: 'error',
      confirmButtonText: 'Закрити'
    });
  }
}
```

Рисунок 4.2 – Відправка запросу реєстрації

```
},
async login() {
  if (!this.username) {
    return this.error('Потрібно ввести ім\'я користувача!');
  }
  if (!this.password) {
    return this.error('Потрібно ввести пароль!');
  }

  const response = await axios.post('http://localhost:4000/auth', {username: this.username, password: this.password});
  if (response.data.ok) {
    this.username = '';
    this.password = '';

    this.name = response.data.name;
    this.token = response.data.token;
    this.access = response.data.access;

    localStorage.setItem('token', response.data.token);
  }
  else {
    let error_text = 'Невідома помилка!';

    if (response.data.error == 'username') {
      error_text = 'Такого користувача немає!'
    }
    else if (response.data.error == 'password') {
      error_text = 'Неправильний пароль!'
    }
    else if (response.data.error == 'need_registration') {
      error_text = 'Вам необхідно зареєструвати ваш токен!'
    }

    Swal.fire({
      title: 'Помилка!',
```

Рисунок 4.3 – Відправка запросу авторизації

Однак реєстрації недостатньо – адміністратор повинен вручну надати доступ кожному користувачу, адже без цього система не дозволить йому сканувати коди, проходити через турнікети чи взагалі користуватися сервісом. При цьому в інтерфейсі чітко виводиться повідомлення про статус сторінки користувача – активна вона (рис. 4.4) чи відключена (рис. 4.5).

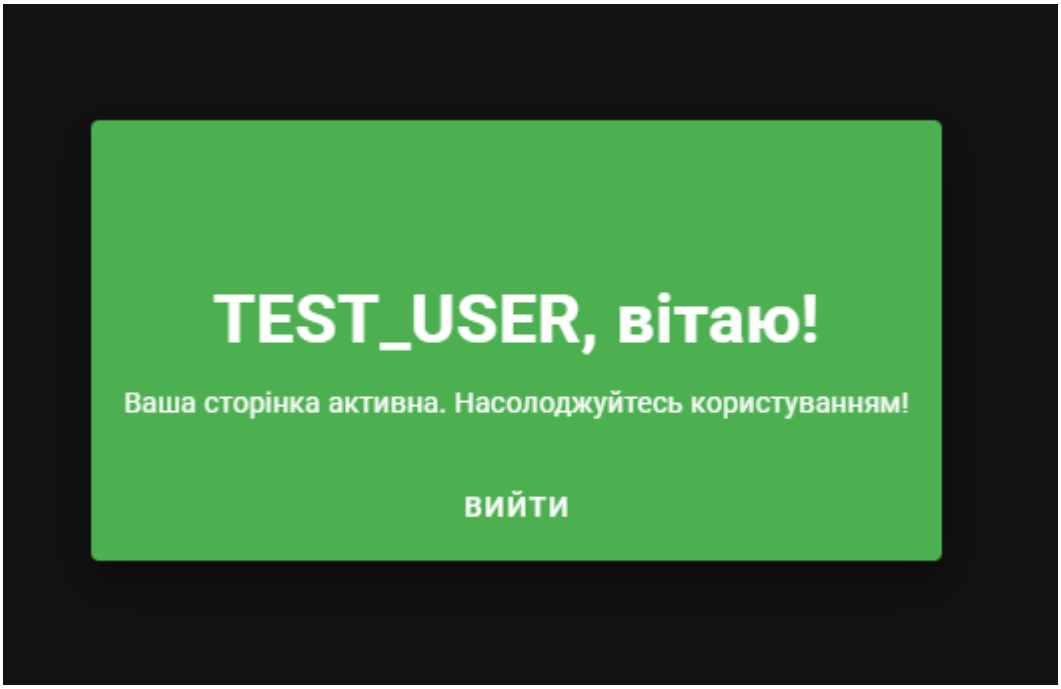


Рисунок 4.4 – Активна сторінка

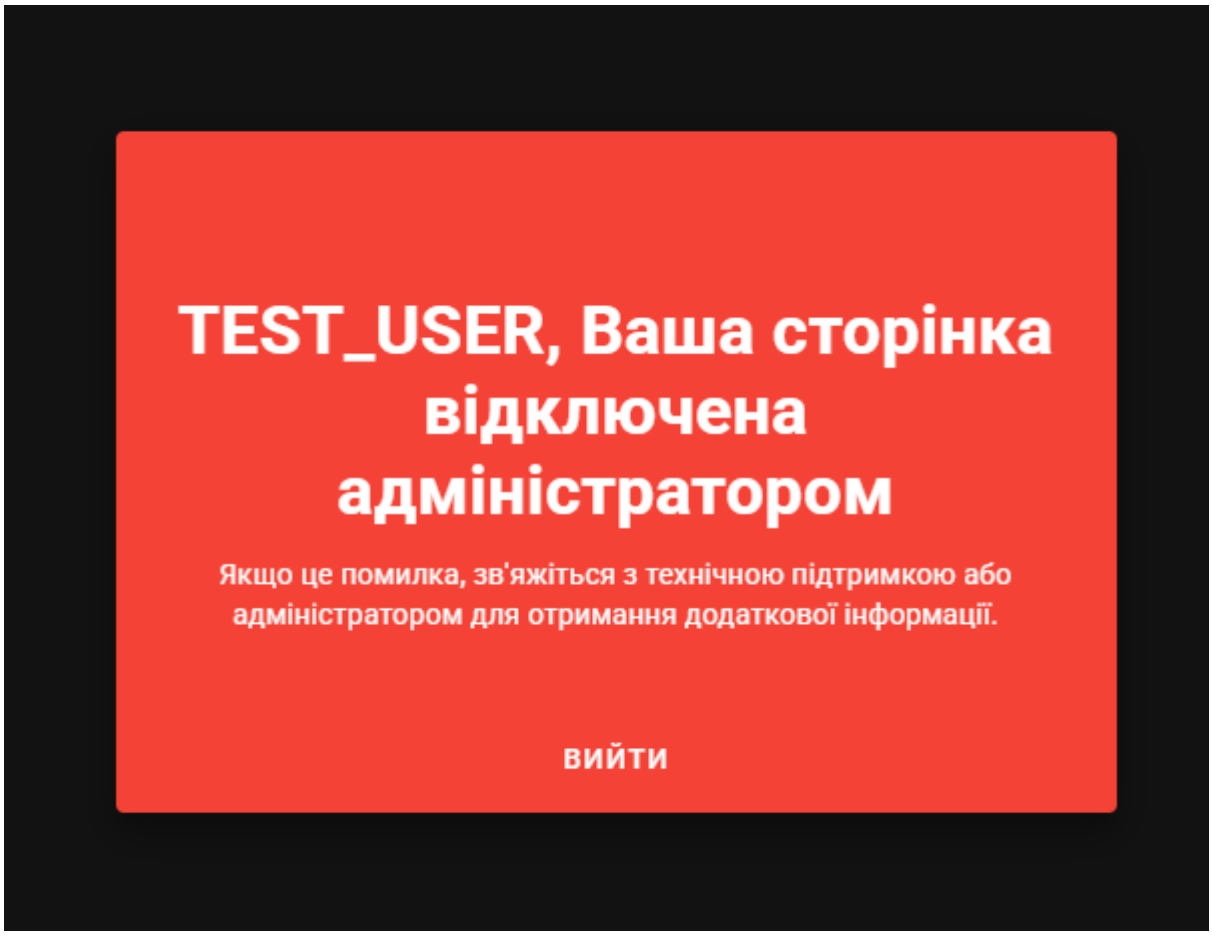


Рисунок 4.5 – Відключена сторінка

Якщо доступ не наданий адміністратором, на екрані з’являється червона картка з повідомленням про обмеження, і користувач не зможе взаємодіяти з системою. В іншому випадку він бачить вітальне повідомлення і може користуватися функціоналом, зокрема сканувати коди, відкривати турнікети тощо. Авторизація обробляється через API, де перевіряється валідність логіна і пароля, а також статус доступу. Всі помилки (наприклад, неправильний пароль, відсутність токену чи незареєстрований користувач) обробляються через вікна сповіщень SweetAlert2 (рис. 4.6), що забезпечує дружній інтерфейс. Таким чином, код ефективно поєднує зручний інтерфейс, гнучку логіку доступу і безпечну авторизацію.

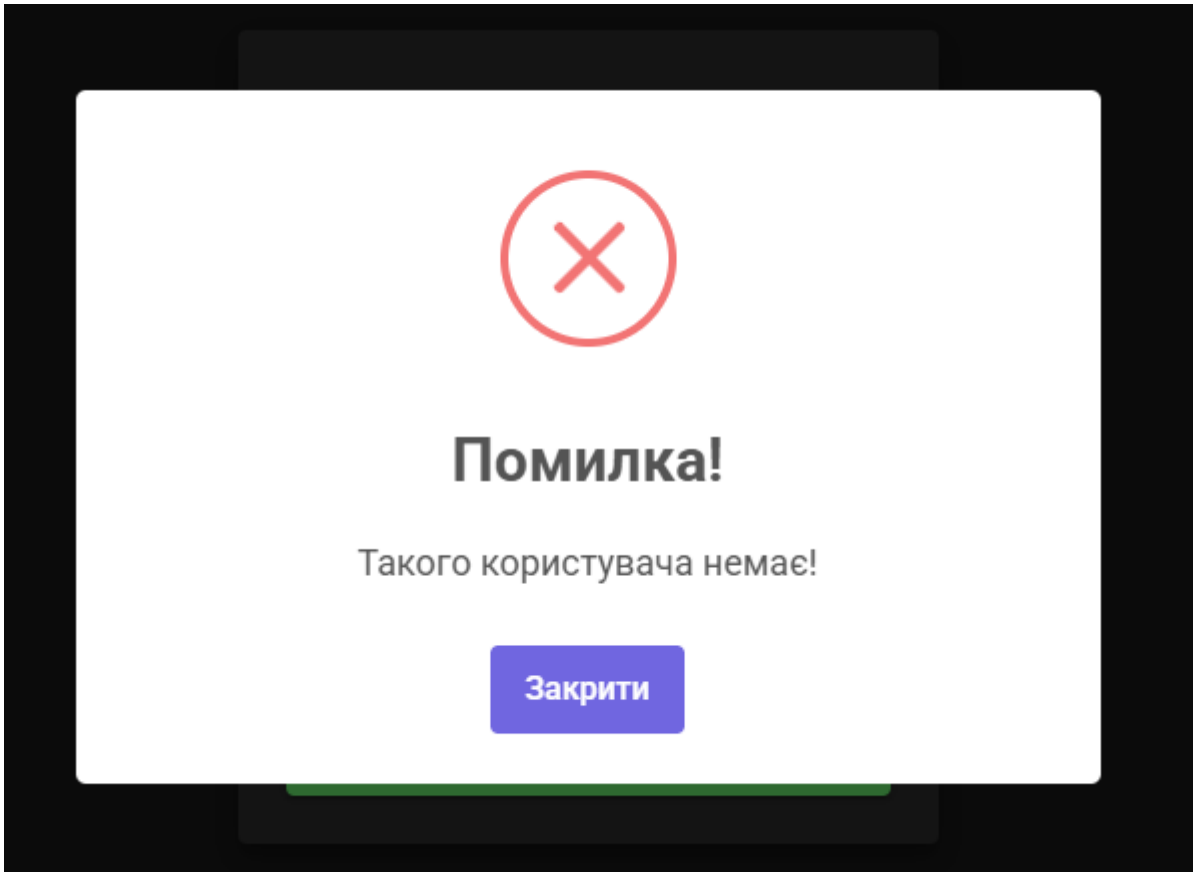


Рисунок 4.6 – Приклад сповіщення

5 РОЗРОБКА ПАНЕЛІ АДМІНІСТРАТОРА

Адмін-панель – це серце всієї системи контролю доступу та обліку робочого часу. Саме через неї адміністратор може повністю керувати користувачами: створювати нові облікові записи, генеруючи для них унікальний токен для подальшої реєстрації, надавати або відкликати доступ до системи, примусово завершувати робочий день чи перерву конкретного співробітника. Адмін також може видаляти користувачів, переглядати їхній статус, посаду та історію дій. Уся активність – від авторизації через QR-код до виходу на перерву – логуються. На основі цих подій обчислюється час, проведений на робочому місці, який можна зручно переглянути за останній тиждень або місяць.

У неавторизованому режимі користувач бачить форму входу з підказкою (рис. 5.1): логін і пароль за замовчуванням – admin/admin.

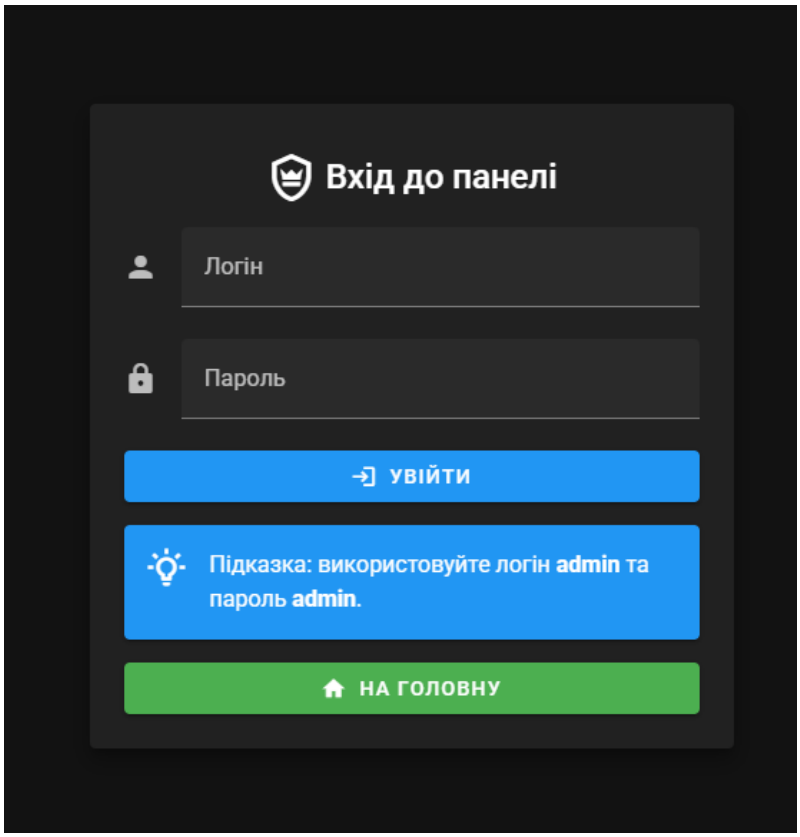


Рисунок 5.1 – Форма входу до адмін панелі

Звісно, це демонстраційний варіант, в реальному середовищі навіть посилання на адмін панель не буде. При успішному вході зберігається токен доступу в localStorage (рис. 5.2), який потім використовується для всіх запитів до бекенду. Успішна авторизація перемикає стан на auth на true, після чого показується основна частина інтерфейсу.

```
},
async login() {
  if (this.username === 'admin' && this.password === 'admin') {
    localStorage.setItem('auth', 'true');
    localStorage.setItem('access_token', 'Gc7iuGrDpWEcOLW7');
    this.auth = true;
  }
  else
  {
    Swal.fire({
      title: 'Помилка!',
      text: 'Невірний логін або пароль!',
      icon: 'error',
      confirmButtonText: 'Закрити'
    });
  }
},
async logout() {
  localStorage.removeItem('auth');
  localStorage.removeItem('access_token');
  this.auth = false;
},
```

Рисунок 5.2 – Обробка авторизації до адмін панелі

У авторизованому режимі є дві головні секції. Перша – це форма для додавання нового користувача (рис. 5.3): вводиться ім’я та посада, після чого запит надсилається на /add (рис. 5.4). У відповідь повертається унікальний токен і дата створення, які одразу додаються до таблиці користувачів.

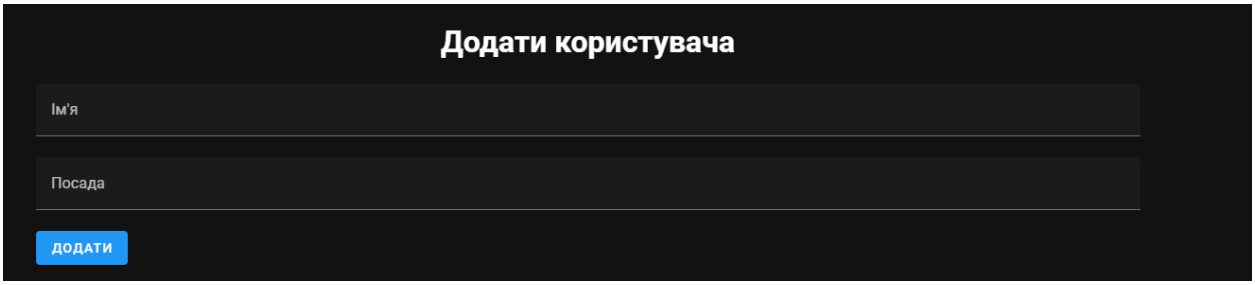


Рисунок 5.3 – Форма додавання нових користувачів

```
},
async add() {
  try
  {
    const response = await axios.post('http://localhost:4000/add', {access_token: this.access_token, name: this.new_user.name, position: this.new_user.position});
    if (response.data.ok)
    {
      this.users.push({id: response.data.id, issued: response.data.issued, status: 0, token: response.data.token, ...this.new_user});
      this.new_user.name = '';
      this.new_user.position = '';
    }
    else
    {
      Swal.fire({
        title: 'Помилка!',
        text: (response.data.error || 'Не вдалося створити нового користувача!'),
        icon: 'error',
        confirmButtonText: 'Закрити'
      });
    }
  }
  catch (error)
  {
    console.error('Error adding user:', error);
  }
},
```

Рисунок 5.4 – Відправка запросу на додавання користувача

Друга секція – таблиця з усіма зареєстрованими користувачами (рис. 5.5). Кожен рядок відображає керування (іконки перегляду, видалення, ID), ім’я, токен, посаду, статус доступу (зелений – активний, червоний – відключений), статус сесії (не авторизовано, авторизується, авторизовано, перерва) та дату створення токену. Кожен статус є клікабельним, і зміна доступу чи статусу відправляє відповідний запит до серверу.

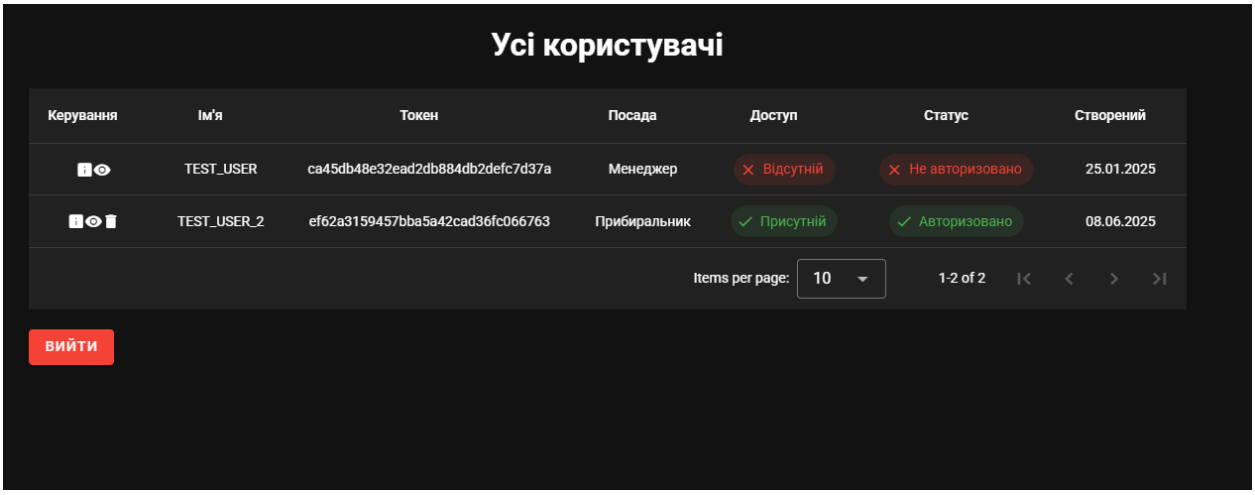


Рисунок 5.5 – Таблиця з користувачами

Для зручності усі дати форматуються у вигляді ДД.ММ.РРРР, а помилки відображаються через модальні вікна SweetAlert.

Також присутній компонент UserView який відкриває діалогове вікно з детальною інформацією про користувача, що дозволяє аналізувати його присутність, перерви та авторизації по днях.

Це інтерфейсне вікно, яке відкривається з адмін-панелі для перегляду детальної статистики конкретного користувача (рис. 5.6). Його основне призначення – надати адміністратору повну картину робочої активності працівника, включаючи історію подій входу/виходу та сумарний час присутності на робочому місці.

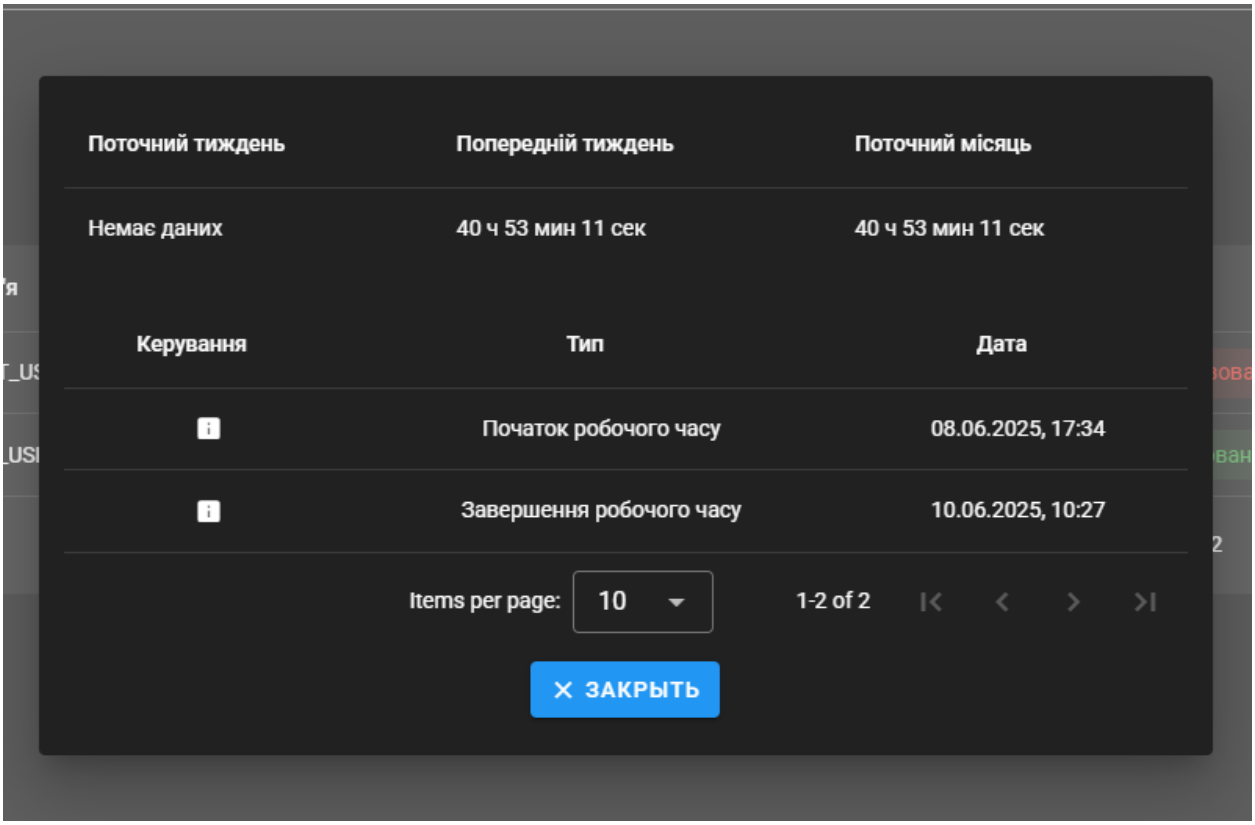


Рисунок 5.6 – Перегляд статистики користувача

Коли адміністратор натискає на кнопку перегляду біля конкретного працівника, відкривається модальне вікно (<v-dialog>), де відображається загальний час роботи та журнал активностей.

Коли адміністратор відкриває це вікно, відбувається запит до бекенду (рис. 5.7) на два різні маршрути – /user та /worktime.

```
methods: {
  async update() {
    const response = await axios.post('http://localhost:4000/user', {access_token: this.access_token, id: this.client_id});
    if (response.data.ok) {
      this.data = response.data.data;
      console.log(response.data);
    }

    const response2 = await axios.post('http://localhost:4000/worktime', {access_token: this.access_token, id: this.client_id});
    if (response2.data.ok) {
      this.time = response2.data.data;
      console.log(response2.data);
    }
  },
```

Рисунок 5.7 – Запит для отримання статистики

Перший повертає історію дій користувача, таких як початок чи завершення роботи, або початок перерви. Ці події відображаються у вигляді

таблиці з колонками: тип події, дата та ідентифікатор. Тип події конвертується у зручний для сприйняття текст за допомогою функції type_format (рис. 5.8), яка перетворює числові значення у назви: «Початок робочого часу», «Завершення робочого часу» тощо.

```
format(seconds) {
  if (seconds === 0) {
    return 'Немає даних';
  }
  if (seconds < 60) {
    return `${seconds} сек`;
  }

  let minutes = Math.floor(seconds / 60);
  let sec = seconds % 60;

  if (minutes < 60) {
    return `${minutes} мин ${sec} сек`;
  }

  let hours = Math.floor(minutes / 60);
  minutes = minutes % 60;

  return `${hours} ч ${minutes} мин ${sec} сек`;
},
type_format(id) {
  return ['Завершення робочого часу', 'Початок робочого часу', '', 'Початок перерви'][id];
}
```

Рисунок 5.8 – Конвертація часу

Дата події форматована у вигляді ДД.ММ.РРРР, ГГ:ХХ за допомогою функції date_format (рис. 5.9).

```
},
date_format(text) {
  let date = new Date(text * 1000);

  return (date.getDate().toString().padStart(2, '0') + '.' + (date.getMonth() + 1).toString().padStart(2, '0') + '.' + date.getFullYear() + ', ' + date.getHours().toString().pads
},
format(seconds) {
```

Рисунок 5.9 – Конвертація дати

Другий запит до /worktime повертає агреговані значення часу – скільки користувач провів у системі протягом поточного тижня, попереднього тижня та поточного місяця. Ці значення обробляються функцією format, яка перетворює кількість секунд у вигляд 1 ч 24 мин 35 сек, або вказує «Немає даних», якщо значення дорівнює нулю. Всі запити супроводжуються токеном

авторизації access_token, який зчитується з localStorage при монтуванні компонент. Компонент працює як модальне вікно v-dialog, яке відкривається або закривається за допомогою двостороннього зв’язку через v-model. Таблиця має заголовки, визначені у headers, і підтримує сортування. Також кожен рядок має іконку з тултіпом, що відображає ID події.

6 СЕРВЕРНА ЧАСТИНА СИСТЕМИ КОНТРОЛЮ УПРАВЛІННЯ ДОСТУПОМ

В проєкті була реалізована серверна частина вебзастосунку на основі платформи Node.js. Ця частина відповідає за управління користувачами, реєстрацію подій, облік робочого часу, а також забезпечення доступу до даних через API.

Запит на /access (рис. 6.1) змінює право доступу конкретного користувача. Спочатку перевіряється токен (access_token). Якщо він не збігається, повертається помилка. Далі виконується запит до БД, щоб отримати поточне значення поля access для користувача за його id. Якщо користувач знайдений, його access перемикається. Потім оновлене значення записується назад до БД. Успішний запрос повертає {ok: true}.

```
// ------------------ [[ admin.vue ]] ------------------ \\
// Обновление доступа пользователя
app.post('/access', (req, res) => {
  if (req.body.access_token !== access_token) {
    return res.status(201).json({ok: false, error: 'Access Denied.'});
  }
  db.get('SELECT access FROM users WHERE id = ?', [req.body.id], (err, row) => {
    if (err) {
      return res.status(201).json({ok: false});
    }

    if (row) {
      const new_access = (row.access === 1 ? 0 : 1);

      db.run('UPDATE users SET access = ? WHERE id = ?', [new_access, req.body.id], function(err) {
        if (err) {
          return res.status(201).json({ok: false});
        }

        return res.status(201).json({ok: true});
      });
    }
  });
});
```

Рисунок 6.1 – Запит зміни доступа користувача

Запит на /status (рис. 6.2) дозволяє змінити status користувача (наприклад, 0 – неактивний, 1 – працює, 2 – завершив день). Токен

адміністратора перевіряється аналогічно до /access. Потім виконується вибірка поточного status, і якщо користувач знайдений, його status оновлюється до переданого у тілі запиту. У випадку успіху повертається {ok: true}.

```
// Обновление статуса пользователя
app.post('/status', (req, res) => {
  if (req.body.access_token !== access_token) {
    return res.status(201).json({ok: false, error: 'Access Denied.'});
  }
  db.get('SELECT status FROM users WHERE id = ?', [req.body.id], (err, row) => {
    if (err) {
      return res.status(201).json({ok: false});
    }

    if (row) {
      db.run('UPDATE users SET status = ? WHERE id = ?', [req.body.status, req.body.id], function(err) {
        if (err) {
          return res.status(201).json({ok: false});
        }

        return res.status(201).json({ok: true});
      });
    }
  });
});
```

Рисунок 6.2 – Запит на зміну статуса користувача

Маршрут /add призначений для додавання нового користувача до бази даних (рис. 6.3). Він реалізує базову валідацію авторизації за допомогою токена (access_token), переданого в тілі запиту. Якщо токен недійсний, сервер повертає відповідь з помилкою доступу.

```
// Добавление нового пользователя
app.post('/add', (req, res) => {
  if (req.body.access_token !== access_token) {
    return res.status(201).json({ok: false, error: 'Access Denied.'});
  }
  const {name, position} = req.body;

  if (!name || !position)
  {
    return res.status(201).json({ok: false, error: 'Необхідно вказати ім\'я та посаду!'});
  }

  const token = crypto.randomBytes(16).toString('hex'); // Генерация токена
  const issued = Math.floor(Date.now() / 1000); // Время в формате UNIX

  db.run('INSERT INTO users (name, position, token, issued) VALUES (?, ?, ?, ?)', [name, position, token, issued], function (err) {
    if (err) {
      console.error('[/add] Error adding user:', err);
      return res.status(201).json({ok: false, error: 'Помилка додавання користувача!'});
    }

    return res.status(201).json({ok: true, id: this.lastID, token, issued});
  });
});
```

Рисунок 6.3 – Запит на додавання користувача

Маршрут /delete (рис. 6.4) видаляє користувача та всі пов’язані з ним записи в таблиці scans. Спочатку перевіряється access_token. Потім з таблиці users видаляється рядок за id, і далі – з scans всі записи з відповідним user_id. Успішне виконання повертає {ok: true}.

```
// Удаление пользователя
app.post('/delete', (req, res) => {
  if (req.body.access_token !== access_token) {
    return res.status(201).json({ok: false, error: 'Access Denied.'});
  }
  db.run('DELETE FROM users WHERE id = ?', [req.body.id], function(err) {
    if (err) {
      res.status(201).json({ok: false});
    }

    db.run('DELETE FROM scans WHERE user_id = ?', [req.body.id], function(err) {
      if (err) {
        res.status(201).json({ok: false});
      }
    });

    return res.status(201).json({ok: true});
  });
});
// Получение списка пользователей
```

Рисунок 6.4 – Запит на видалення користувача

Запит на /get повертає всі записи з таблиці users. Перевіряється access_token, і якщо він коректний – зчитуються всі користувачі (db.all). Успішна відповідь містить масив усіх користувачів у полі data.

```
// Получение списка пользователей
app.post('/get', (req, res) => {
  if (req.body.access_token !== access_token) {
    return res.status(201).json({ok: false, error: 'Access Denied.'});
  }
  db.all('SELECT * FROM users', [], (err, rows) => {
    if (err) {
      return res.status(201).json({ok: false});
    }

    return res.status(201).json({ok: true, data: rows});
  });
});
// Получение сканирований клиента
```

Рисунок 6.5 – Запит для отримання усіх користувачів

Маршрут /users (рис. 6.6) перевіряє access_token, а потім виводить з таблиці scans всі записи, де user_id відповідає ID з тіла запиту. У відповідь повертаються всі події, пов’язані з цим користувачем: початок дня, завершення, перерви тощо.

```
});
// Получение сканирований клиента
app.post('/user', (req, res) => {
  if (req.body.access_token !== access_token) {
    return res.status(201).json({ok: false, error: 'Access Denied.'});
  }
  db.all('SELECT * FROM scans WHERE user_id = ?', [req.body.id], (err, rows) => {
    if (err) {
      return res.status(201).json({ok: false});
    }

    return res.status(201).json({ok: true, data: rows});
  });
});
// Получение времени клиента
```

Рисунок 6.6 – Запит усіх сканувань співробітника

Маршрут /worktime (рис. 6.7) є розрахунком робочого часу користувача, що:

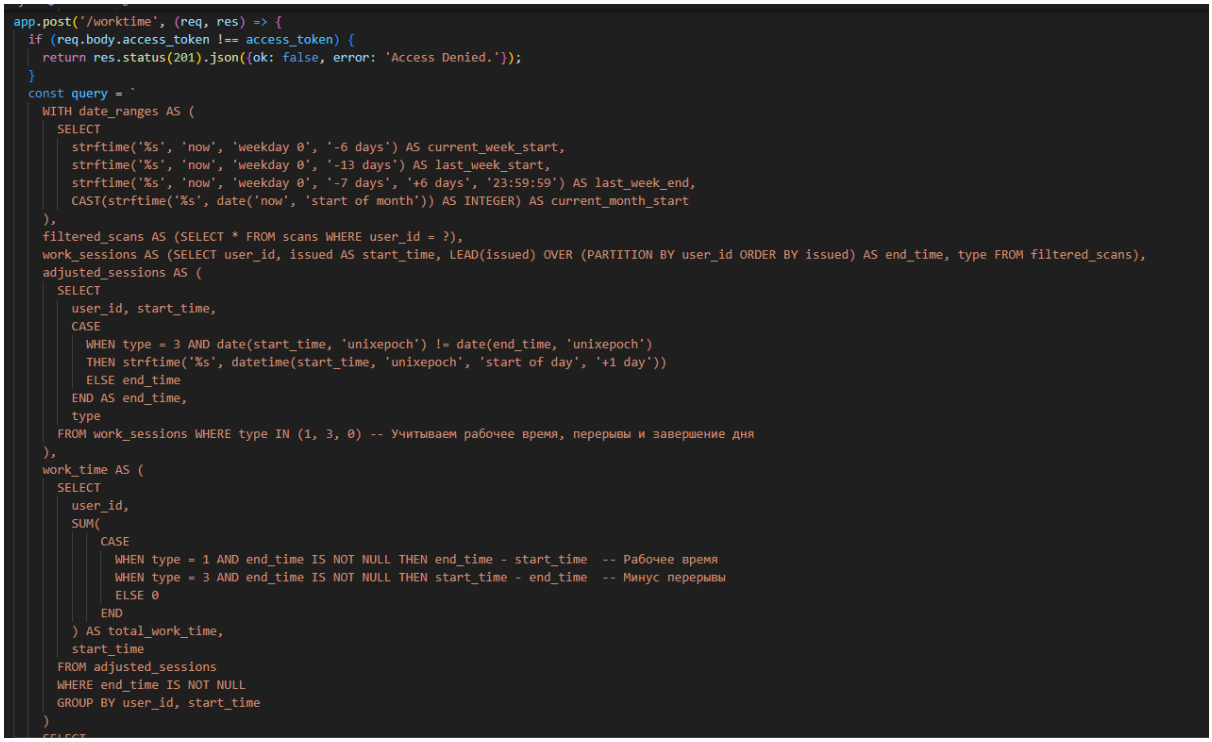
– вираховує дати початку поточного та минулого тижня/місяця;

– витягує пари подій (наприклад, початок і кінець дня);

– враховує перерви, щоб відняти їх з загального часу;

– обчислює сумарний час, відпрацьований користувачем за поточний тиждень, минулий тиждень і поточний місяць.

Після обчислень повертаються значення часу у секундах для кожного з періодів.



```
app.post('/worktime', (req, res) => {
  if (req.body.access_token !== access_token) {
    return res.status(201).json({ok: false, error: 'Access Denied.'});
  }
  const query = `
    WITH date_ranges AS (
      SELECT
        strftime('%s', 'now', 'weekday 0', '-6 days') AS current_week_start,
        strftime('%s', 'now', 'weekday 0', '-13 days') AS last_week_start,
        strftime('%s', 'now', 'weekday 0', '-7 days', '+6 days', '23:59:59') AS last_week_end,
        CAST(strftime('%s', date('now', 'start of month')) AS INTEGER) AS current_month_start
    ),
    filtered_scans AS (SELECT * FROM scans WHERE user_id = ?),
    work_sessions AS (SELECT user_id, issued AS start_time, LEAD(issued) OVER (PARTITION BY user_id ORDER BY issued) AS end_time, type FROM filtered_scans),
    adjusted_sessions AS (
      SELECT
        user_id, start_time,
        CASE
          WHEN type = 3 AND date(start_time, 'unixepoch') != date(end_time, 'unixepoch')
          THEN strftime('%s', datetime(start_time, 'unixepoch', 'start of day', '+1 day'))
          ELSE end_time
        END AS end_time,
        type
      FROM work_sessions WHERE type IN (1, 3, 0) -- Учитываем рабочее время, перерывы и завершение дня
    ),
    work_time AS (
      SELECT
        user_id,
        SUM(
            CASE
              WHEN type = 1 AND end_time IS NOT NULL THEN end_time - start_time  -- Рабочее время
              WHEN type = 3 AND end_time IS NOT NULL THEN start_time - end_time  -- Минус перерывы
              ELSE 0
            END
        ) AS total_work_time,
        start_time
      FROM adjusted_sessions
      WHERE end_time IS NOT NULL
      GROUP BY user_id, start_time
    )
    SELECT
```

Рисунок 6.7 – Запит на розрахунок робочого часу

Запит на /qr (рис. 6.8) використовується для створення або оновлення QR-коду в системі. Коли запит надходить, сервер перевіряє, чи вже існує запис із відповідним режимом (mode) у таблиці codes. Якщо запис є і він прив’язаний до користувача, то система додатково перевіряє його статус, і, якщо користувач ще не почав робочий день, статус обнуляється. Потім або оновлюється існуючий запис із новим токеном, або створюється новий запис із вказаними параметрами – режимом, токеном і міткою часу видачі.

```
// Создание QR
app.post('/qr', (req, res) => {
  db.get('SELECT id, user_id FROM codes WHERE mode = ?', [req.body.mode], (err, row) => {
    if (err) {
      console.error('[/qr] Error getting information from table "codes":', err);
      return res.status(201).json({ok: false});
    }

    const {mode, token, issued} = req.body;
    // Проверка на наличие параметров запроса
    if (((mode !== 0) && (mode !== 1)) || !token || !issued) {
      return res.status(201).json({ok: false, error: 'Error retrieving data.'});
    }

    // Проверка на существование полей в таблице с QR кодами
    if (row) {
      // Если генерируется новый QR код, а старый был привязан к пользователю - очищается статус авторизации у пользователя
      if (row.user_id) {
        // Проверка статуса пользовател, чтобы случайно не сбить статус тому, кто уже начал рабочий день
        db.get('SELECT status FROM users WHERE id = ?', [row.user_id], (err, row) => {
          if (err) {
            console.error('[/qr] Error getting information from table "users":', err);
            return res.status(201).json({ok: false});
          }

          if (row && (row.status === 1)) {
            db.run('UPDATE users SET status = 0 WHERE id = ?', [row.user_id]);
          }
        });
      }

      // Обновление информации о QR коде
      db.run('UPDATE codes SET token = ?, issued = ?, user_id = NULL WHERE mode = ?', [token, issued, mode], function(err) {
        if (err) {
          console.error('[/qr] Error updating information in the table "codes":', err);
          return res.status(201).json({ok: false});
        }

        res.status(201).json({ok: true, token: token});
      });
    }
```

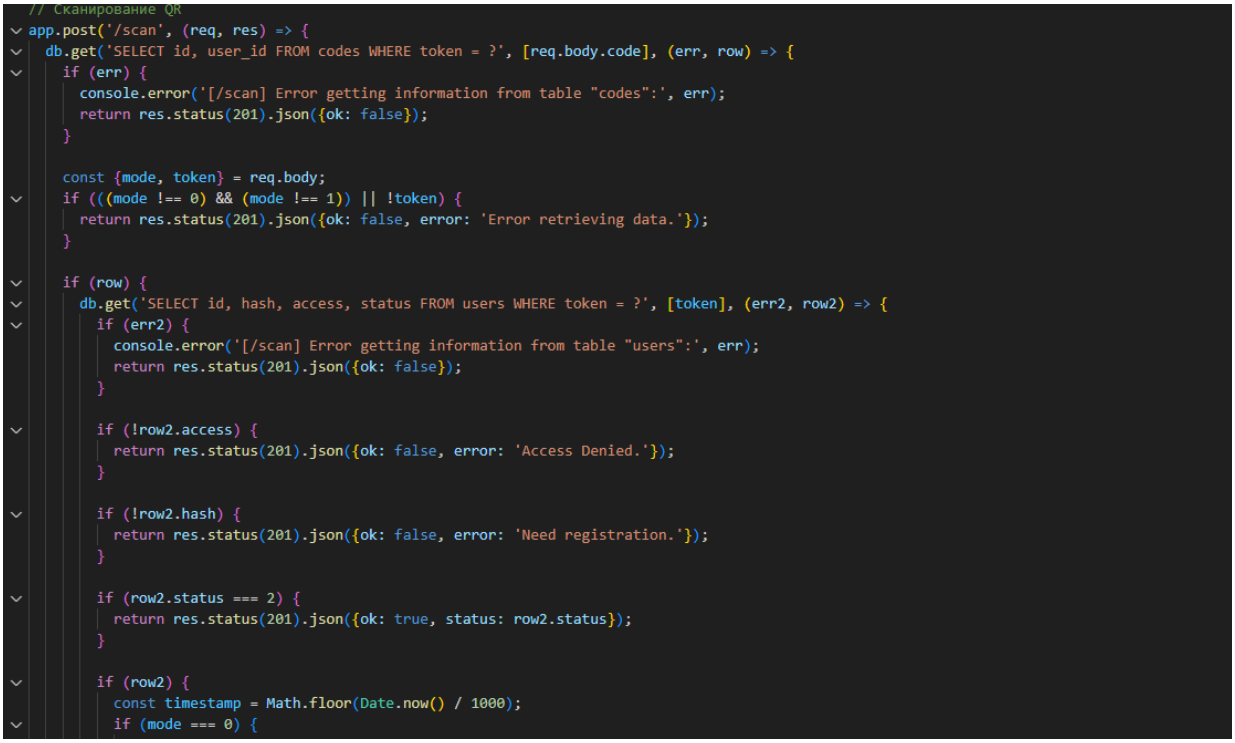
Рисунок 6.8 – Запит для отримання робого часу співробітників

Запит на /update призначений для отримання всієї актуальної інформації про наявні QR-коди у системі (рис. 6.9). Сервер просто читає всі записи з таблиці codes та повертає їх клієнту. Це може бути корисно для адміністративного інтерфейсу або для періодичного оновлення стану QR-кодів на клієнтському боці.

```
// Получение информации о QR
app.post('/update', (req, res) => {
  db.all('SELECT * FROM codes', [], (err, rows) => {
    if (err) {
      console.error('[/update] Error getting information from table "codes":', err);
      res.status(201).json({ok: false});
    }
    else {
      res.status(201).json(rows);
    }
  });
});
```

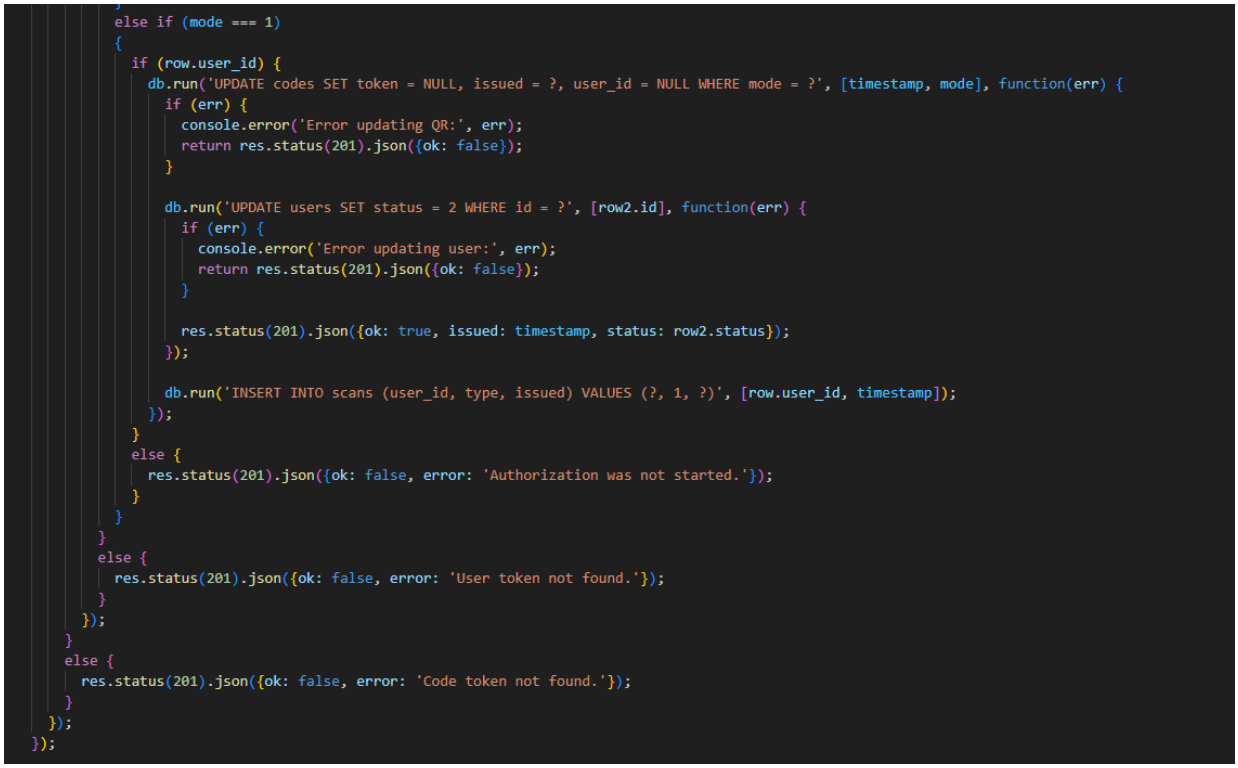
Рисунок 6.9 – Запит для отримання наявних QR кодів

Запит на /scan обробляє сканування QR-коду користувачем (рис. 6.10 та рис. 6.11). Він перевіряє, чи дійсний токен коду, та чи співпадає він з токеном зареєстрованого користувача. Якщо користувач має доступ, зареєстрований і не завершив робочий день, тоді залежно від режиму (вхід або вихід), система або закріплює користувача за QR-кодом (режим 0), або навпаки – скидає токен і змінює статус користувача на «робочий день завершено» (режим 1), а також фіксує подію в таблиці scans.



```
// Сканирование QR
app.post('/scan', (req, res) => {
  db.get('SELECT id, user_id FROM codes WHERE token = ?', [req.body.code], (err, row) => {
    if (err) {
      console.error('[/scan] Error getting information from table "codes":', err);
      return res.status(201).json({ok: false});
    }

    const {mode, token} = req.body;
    if (((mode !== 0) && (mode !== 1)) || !token) {
      return res.status(201).json({ok: false, error: 'Error retrieving data.'});
    }

    if (row) {
      db.get('SELECT id, hash, access, status FROM users WHERE token = ?', [token], (err2, row2) => {
        if (err2) {
          console.error('[/scan] Error getting information from table "users":', err);
          return res.status(201).json({ok: false});
        }

        if (!row2.access) {
          return res.status(201).json({ok: false, error: 'Access Denied.'});
        }

        if (!row2.hash) {
          return res.status(201).json({ok: false, error: 'Need registration.'});
        }

        if (row2.status === 2) {
          return res.status(201).json({ok: true, status: row2.status});
        }

        if (row2) {
          const timestamp = Math.floor(Date.now() / 1000);
          if (mode === 0) {
```

Рисунок 6.10 – Обробка сканування QR кода (перша частина)



```
        else if (mode === 1)
        {
          if (row.user_id) {
            db.run('UPDATE codes SET token = NULL, issued = ?, user_id = NULL WHERE mode = ?', [timestamp, mode], function(err) {
              if (err) {
                console.error('Error updating QR:', err);
                return res.status(201).json({ok: false});
              }

              db.run('UPDATE users SET status = 2 WHERE id = ?', [row2.id], function(err) {
                if (err) {
                  console.error('Error updating user:', err);
                  return res.status(201).json({ok: false});
                }

                res.status(201).json({ok: true, issued: timestamp, status: row2.status});
              });

              db.run('INSERT INTO scans (user_id, type, issued) VALUES (?, 1, ?)', [row.user_id, timestamp]);
            });
          }
          else {
            res.status(201).json({ok: false, error: 'Authorization was not started.'});
          }
        }
      }
      else {
        res.status(201).json({ok: false, error: 'User token not found.'});
      }
    });
  }
  else {
    res.status(201).json({ok: false, error: 'Code token not found.'});
  }
 });
});
```

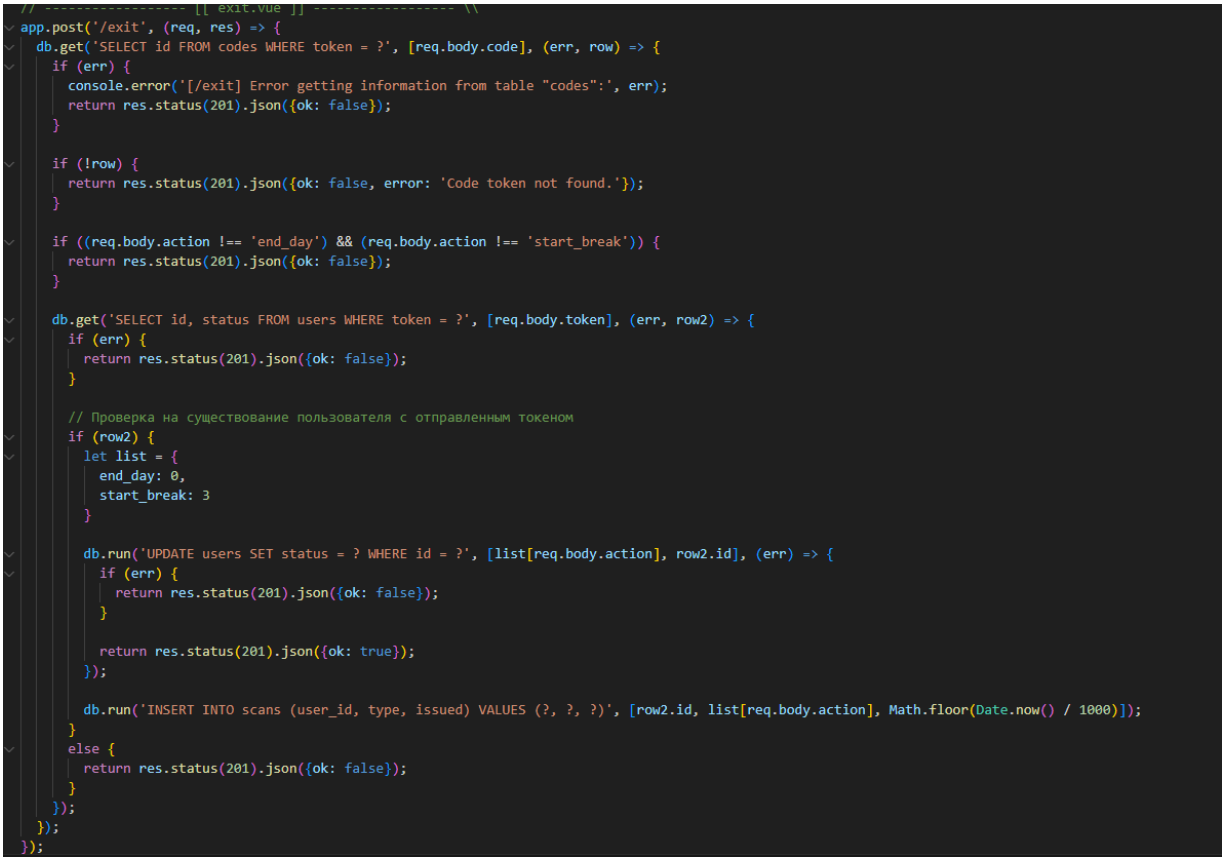
Рисунок 6.11 – Обробка сканування QR кода (друга частина)

Запит на /entrance підтверджує сканування першого QR коду (рис. 6.12). Якщо користувач знайдений за токеном, його QR-записи обнуляються, щоб уникнути дублювання. Якщо тип події дорівнює 1, користувач прикріплюється до QR-коду з режимом 1 (вихід), і його статус оновлюється на «робочий день розпочато». В інших випадках просто очищується зв’язок без зміни статусу.

```
// ------------------ [[ entrance.vue ]] ------------------ \\
app.post('/entrance', (req, res) => {
  db.get('SELECT id, status FROM users WHERE token = ?', [req.body.token], (err, row) => {
    if (err) {
      return res.status(201).json({ok: false});
    }

    // Проверка на существование пользователя с отправленным токеном
    if (row) {
      db.run('UPDATE codes SET token = NULL, user_id = NULL WHERE user_id = ?', [row.id], function(err) {
        if (err) {
          return res.status(201).json({ok: false});
        }

        if (req.body.type === 1) {
          db.run('UPDATE codes SET user_id = ? WHERE mode = 1', [row.id], function(err) {
            if (err) {
              return res.status(201).json({ok: false});
            }

            db.run('UPDATE users SET status = 1 WHERE id = ?', [row.id], (err, row) => {
              if (err) {
                return res.status(201).json({ok: false});
              }

              return res.status(201).json({ok: true});
            });
          });
        }
        else {
          return res.status(201).json({ok: true});
        }
      });
    }
    else
    {
      return res.status(201).json({ok: false});
    }
  });
```

Рисунок 6.12 – Обробка запиту підтвердження початку робочого дня

Запит /exit використовується, коли працівник завершує робочий день або починає перерву (рис. 6.13). Сервер перевіряє токен QR-коду та користувача. Потім, залежно від дії (end_day або start_break), оновлює статус користувача відповідним чином (0 – день завершено, 3 – перерва) і створює запис у таблиці scans з позначкою часу.



```
// ------------------ [[ exit.vue ]] ------------------ \\
app.post('/exit', (req, res) => {
  db.get('SELECT id FROM codes WHERE token = ?', [req.body.code], (err, row) => {
    if (err) {
      console.error('[/exit] Error getting information from table "codes":', err);
      return res.status(201).json({ok: false});
    }

    if (!row) {
      return res.status(201).json({ok: false, error: 'Code token not found.'});
    }

    if ((req.body.action !== 'end_day') && (req.body.action !== 'start_break')) {
      return res.status(201).json({ok: false});
    }

    db.get('SELECT id, status FROM users WHERE token = ?', [req.body.token], (err, row2) => {
      if (err) {
        return res.status(201).json({ok: false});
      }

      // Проверка на существование пользователя с отправленным токеном
      if (row2) {
        let list = {
          end_day: 0,
          start_break: 3
        }

        db.run('UPDATE users SET status = ? WHERE id = ?', [list[req.body.action], row2.id], (err) => {
          if (err) {
            return res.status(201).json({ok: false});
          }

          return res.status(201).json({ok: true});
        });

        db.run('INSERT INTO scans (user_id, type, issued) VALUES (?, ?, ?)', [row2.id, list[req.body.action], Math.floor(Date.now() / 1000)]);
      }
      else {
        return res.status(201).json({ok: false});
      }
    });
  });
});
```

Рисунок 6.13 – Обробка запиту завершення робочого дня або взаємодії з перервою

Запит /data дозволяє отримати базову інформацію про користувача за його токеном, зокрема ім’я (рис. 6.14). Це потрібно, наприклад, для відображення даних про поточного авторизованого користувача в інтерфейсі без повторної перевірки логіна.

```
// ------------------ [[ user.vue ]] ------------------ \\
// Инфорамция о пользователе
app.post('/data', (req, res) => {
  db.get('SELECT name FROM users WHERE token = ?', [req.body.token], (err, row) => {
    if (err) {
      return res.status(201).json({ok: false});
    }

    if (row) {
      res.status(201).json({ok: true, name: row.name});
    }
    else {
      res.status(201).json({ok: false});
    }
  });
});
```

Рисунок 6.14 – Отримання інформації про користувача

Запит /auth відповідає за авторизацію користувача за логіном і паролем (рис. 6.15). Якщо користувач існує і має встановлений хеш пароля, то сервер порівнює його з хешем, створеним із надісланого пароля. Успішна перевірка повертає токен, ім’я користувача та його рівень доступу. У разі помилок (неіснуюче ім’я, неправильний пароль або відсутність реєстрації) повертається відповідна помилка.

```
function hash(password) {
  const hash = crypto.createHash('sha256');
  hash.update(password);
  return hash.digest('hex');
}

// Авторизация пользователя
app.post('/auth', (req, res) => {
  db.get('SELECT name, hash, token, access FROM users WHERE name = ?', [req.body.username], (err, row) => {
    if (err) {
      return res.status(201).json({ok: false});
    }

    let error;
    let username = (typeof(row) === 'object');
    let password = false;
    let result;
    if (username)
    {
      if (row && row.hash)
      {
        password = (hash(req.body.password) === row.hash);
        if (password)
        {
          result = row.hash;
        }
        else
        {
          error = 'password';
        }
      }
      else
      {
        error = 'need_registration';
      }
    }
    else
    {
      error = 'username';
    }

    if (error)
```

Рисунок 6.15 – Авторизація користувача

За реєстрацію нового користувача відповідає запит /register (рис. 6.16). Сервер перевіряє, чи існує користувач із надісланим токеном. Якщо існує і ще не має хешу пароля (тобто ще не зареєстрований), то сервер створює новий хеш пароля і додає його до запису. Якщо користувач уже зареєстрований або токен не знайдено – повертається відповідне повідомлення про помилку.

```
// Регистрация пользователя
app.post('/register', (req, res) => {
  db.get('SELECT name, hash FROM users WHERE token = ?', [req.body.token], (err, row) => {
    if (err) {
      return res.status(201).json({ok: false, error: 'Error fetching user.'});
    }

    if (row) {
      if (!row.hash) {
        db.run('UPDATE users SET hash = ? WHERE token = ?', [hash(req.body.password), req.body.token], function(err) {
          if (err) {
            return res.status(500).send('Error updating user.');
          } else {
            res.status(201).json({ok: true, name: row.name, token: row.token});
          }
        });
      }
      else
      {
        res.status(201).json({ok: false, error: 'already_registered'});
      }
    }
    else {
      res.status(201).json({ok: false, error: 'unknown_token'});
    }
  });
});
```

Рисунок 6.16 – Реєстрація нового користувача

7 ПЕРЕВІРКА ПРОДУКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

З метою перевірки продуктивності реалізованого серверного API було проведено серію нагрузочних тестів з використанням інструментів вимірювання часу відповіді на HTTP-запити. Основна мета полягала в оцінці стабільності роботи сервісу при послідовному виконанні великої кількості запитів до ключових маршрутів системи: /qr, /auth, /update, /data, /get, /access.

Усі тести проводилися на локальному сервері в контрольованих умовах, частина результатів тестування наведена нижче (рис. 7.1, рис. 7.2 та рис. 7.3).

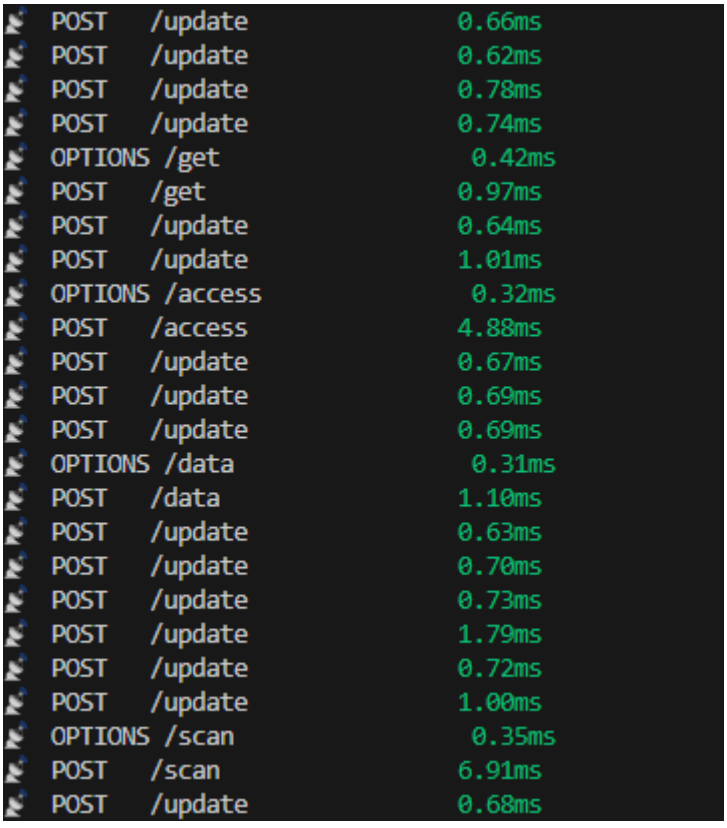


Рисунок 7.1 – Результати тестування (перша частина)

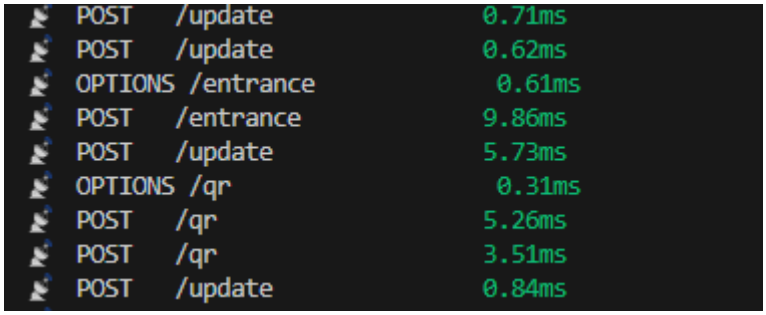


Рисунок 7.2 – Результати тестування (друга частина)

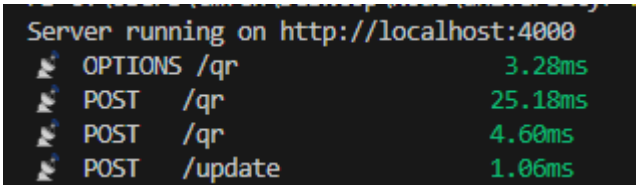


Рисунок 7.3 – Результати тестування (третя частина)

Аналіз результатів показує, що більшість запитів обробляються менш ніж за 2 мілісекунди, що є високим показником продуктивності. Найбільш ресурсозатратним виявився маршрут /qr, що пояснюється його логікою генерації або обробки даних. У середньому час відповіді не перевищував 25 мс, навіть у пікових значеннях.

Для порівняння: типовий API на базі Node.js або Python FastAPI з базою даних MongoDB чи PostgreSQL зазвичай демонструє час відповіді в межах 100 мс (рис. 7.4) при навантаженні середнього рівня. Отже, отримані результати демонструють високу швидкодію та ефективність реалізації навіть при використанні легкої СУБД SQLite.

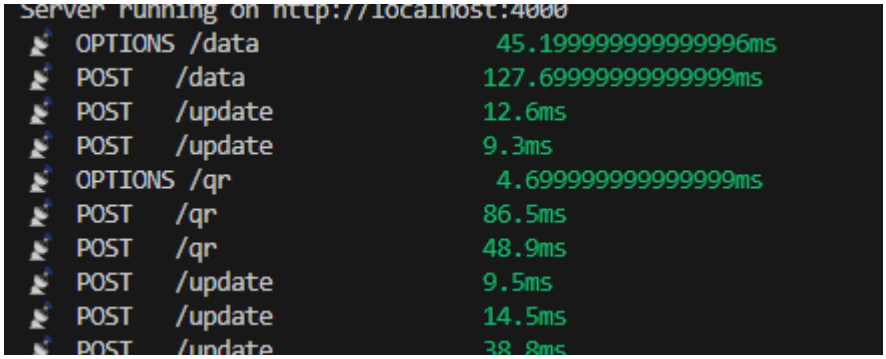


Рисунок 7.4 – Тестування з іншими базами даних

8 ОХОРОНА ПРАЦІ

У процесі розробки та впровадження веб-системи автоматичного обліку робочого часу співробітників необхідно враховувати низку питань, пов’язаних з охороною праці, інформаційною безпекою, ергономікою робочого місця та дотриманням нормативних вимог [16].

Оскільки основною діяльністю під час розробки було створення програмного забезпечення, основним видом робіт є робота з персональним комп’ютером. Така діяльність вимагає дотримання ергономічних норм:

– правильно організоване робоче місце (ергономічне крісло, стіл, належне освітлення);

– регламентовані перерви для зменшення навантаження на зір та опорно-руховий апарат.

У приміщенні, де проводилась розробка, наявні вогнегасники, евакуаційні виходи, інструкції на випадок пожежі. Розробка не передбачає використання легкозаймистих матеріалів, тому пожежна небезпека мінімальна.

Усі роботи виконувались з дотриманням чинних норм охорони праці та безпеки життєдіяльності. Впровадження системи не створює додаткових загроз, а навпаки – сприяє підвищенню дисципліни та безпеки на підприємстві завдяки прозорому обліку робочого часу та автоматизації доступу.

ВИСНОВКИ

У процесі виконання проєкту було успішно розроблено функціональну, безпечну та доступну систему автоматичного контролю робочого часу співробітників на основі QR-кодів. Результати дослідження та реалізації підтвердили актуальність тема. Отримані результати дозволяють зробити такі висновки:

– актуальність теми обумовлена потребою в автоматизації управлінських процесів у сучасних організаціях, зокрема в умовах цифрової трансформації, зростання популярності гнучких графіків та мобільних технологій;

– аналіз існуючих методів контролю показав, що традиційні підходи, як-от паперові журнали або Excel-таблиці, мають суттєві недоліки – високу ймовірність помилок, відсутність гнучкості та значні витрати на адміністрування;

– розроблена система включає двоетапну авторизацію із використанням QR-кодів, що підвищує точність обліку та рівень довіри до даних;

– адміністративна панель системи надає повний контроль над користувачами, формуванням звітів і загальним функціонуванням платформи, що робить систему придатною для впровадження у реальних умовах малих та середніх підприємств;

– безпека системи відповідає сучасним вимогам: реалізовано механізми авторизації, деактивації QR-кодів, захисту облікових записів та обробки персональних даних відповідно до міжнародних стандартів (зокрема, GDPR) та локального законодавства;

– охорона праці й техніка безпеки при розробці програмного забезпечення були належним чином враховані: забезпечено ергономічність середовища розробки, електробезпеку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. 29 с.

2. Методичні вказівки з підготовки кваліфікаційної роботи для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології освітньої програми «Системна інженерія» / Упоряд.: І.Ш. Невлюдов, О.М. Цимбал, О.В. Токарєва, А.І. Бронніков. Харків: ХНУРЕ, 2023. 65 с.

3. Vue.js Documentation. URL: https://vuejs.org/ (дата звернення: 12.06.2025).

4. Vuetify Documentation. URL: https://vuetifyjs.com/ (дата звернення: 12.06.2025).

5. SweetAlert2 Docs. URL: https://sweetalert2.github.io/ (дата звернення: 12.06.2025).

6. Koch, M. Time tracking in digital environments: methods and tools. – Springer, 2021. – 215 p.

7. Lee, J., Xie, Y. Workforce analytics: the intersection of HR and business intelligence // Journal of HR Technology. – 2020. – Vol. 5, No. 2. – P. 45–58.

8. Smith, A., Jones, P. QR Code Attendance Systems: A Modern Approach // Proceedings of the ACM Symposium on Applied Computing. – 2020. – P. 305–310.

9. Теорія автоматичного управління (збірник задач) [Текст]: навч. посіб. для студентів спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології / І.Ш. Невлюдов, О.В. Токарєва; Харків. нац. ун-т радіоелектроніки. – Харків. – 2020. – 240 с.

10. Невлюдов І.Ш. Комп'ютерно-інтегровані технології виробництва технічних засобів автоматизації. Частина 1: Підручник Харків. – 2021. – 604 с.

11. Hernandez, R. Mobile-enabled employee check-in systems // IEEE International Conference on Automation. – 2019. – P. 122–127.

12. Choi, H. Building Responsive Interfaces with Vue.js. – O’Reilly Media, 2022. – 312 p.

13. Hipp, D. R. The Architecture of SQLite [Електронний ресурс]. – Режим доступу: https://sqlite.org/arch.html (дата звернення: 12.06.2025).

14. OWASP Foundation. OWASP Top Ten Web Application Security Risks [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/ (дата звернення: 12.06.2025).

15. Miller, T. Comparative Analysis of Employee Time Tracking Solutions // International Journal of Computer Applications. – 2018. – Vol. 180, No. 41. – P. 11–18.

16. Методичні вказівки до виконання розділу "Охорона праці" у випускних роботах ОКР "бакалавр" усіх форм навчання / упоряд.: В. А. Айвазов. Т. Є. Стиценко., Н. Л. Березуцька ; М-во освіти і науки України, ХНУРЕ. – Харків : ХНУРЕ, 2018. – 28 с. – 1,81.