НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних систем, мереж та кібербезпеки

Касаткін Д.Ю., к. пед.н., доц.

Підпис ПІБ, вчене звання і ступінь

«___»_____2025 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

На тему: Розробка системи контролю доступу для торгової організації з використанням обладнання виробника ZKTeco

Спеціальність F7 «Комп'ютерна інженерія»

Гарант освітньої програми

к.фіз.-мат.н., доцент _____ Євгеній НІКІТЕНКО

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник випускної бакалаврської роботи

старший викладач _____ Володимир МАТІЄВСЬКИЙ

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав

(підпис) (ПІБ студента)

Київ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАНН УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки

Касаткін Д.Ю., к.пед.н., доц. /
підпис ПІБ, вчене звання і ступінь
р.

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ СТУДЕНТА

Денисюк Олег Павлович
(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки) Комп'ютерна інженерія
Тема випускної бакалаврської роботи Розробка системи контролю доступу для торгової організації з використанням обладнання виробника ZKTeco
керівник проекту (роботи) Матієвський В.В. , к.тех.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджено наказом ректора НУБіП України від «16» грудня 2024 р. № 2250 «С»
Термін подання завершеної роботи на кафедру 28.05.2025
(рік, місяць, число)

Вихідні дані до випускної бакалаврської роботи _____

Перелік питань, які потрібно розробити: Аналіз вимог до системи, аналіз предметної області, проєктування, реалізація, тестування системи

Перелік графічних документів (за потреби)_____

Дата видачі завдання "17" грудня 2024 р

Керівник випускної бакалаврської роботи _____ / Матієвський В.В.

(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ / Денисюк О.П.

(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз вимог до системи	07.03.2025 р.	Виконано
2	Проєктування системи	21.03.2025 р.	Виконано
3	Реалізація системи	12.04.2025 р.	Виконано
4	Тестування розробленої системи	27.04.2025 р.	Виконано
5	Оформлення пояснювальної записки	07.05.2025 р.	Виконано
6	Оформлення графічного матеріалу	15.05.2025 р.	Виконано

Студент _____ / Олег ДЕНИСЮК /

(підпис) (ініціали та прізвище)

Керівник проекту (роботи) _____ /Володимир МАТІЄВСЬКИЙ/

(підпис) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 56 сторінок, 20 рисунків, 24 додатки, 15 джерел.

СИСТЕМА КОНТРОЛЮ ДОСТУПУ ZKTECO

Об'єкт дослідження - розробка системи контролю доступу з використанням обладнання виробника ZKTeco. Метою роботи є створення ефективної, зручної та надійної системи, яка забезпечує автоматизований контроль доступу за допомогою реле, які можуть використовуватись у будь-яких цілях. Робота складається із чотирех розділів.

У першому розділі розглядаються основні поняття та технології, пов'язані з розпізнаванням карток. Особливу увагу було приділеного перевагам даного методу автентифікації порівнянно з іншими способами, а також обгрунтовується вибір обладнання і технологій, для реалізації цієї задачі.

Другий розділ зосереджений на структурі та функціоналі системи контролю доступу. Детально описуються ключові компоненти системи, та пояснено процес налаштування та конфігурації, для забезпечення стабільності та ефективності роботи.

Третій розділ присвячений етапам проєктування та реалізації систем. В ньому буде описано, як відбувається спосіб передачі даних, їх варіації, переваги та недоліки такого методу.

Четвертий розділ присвячено аналізуванню результатів, та буде запропоновано шляхи вдосконалення системи. Буде звернуто увагу на основні проблеми, що виникали під час розпізнавання карток, та передачі даних, та буде продемостровано можливі напрями для покращення роботи системи.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 8
ВСТУП 9
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1. Поставлення мети роботи 11
1.2. Дослідження предметної області 12
Висновок до розділу 1 18
РОЗДІЛ 2 СТРУКТУРА ТА ФУНКЦІОНАЛ СИСТЕМИ 19
2.1. Етапи розробки системи 19
2.2. Алгоритм розпізнавання 20
2.3. Опис обладнання 23
2.3.1. Технічна характеристика C2-260 23
2.3.1. Технічна характеристика KR503E-RS 25
2.3.2. Технічна характеристика YM-60N 26
2.3.3. Технічна характеристика BGP-125Pro 27
Висновок до розділу 2 28
РОЗДІЛ 3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ 29
3.1. Налаштування середовища розробки VSCode IDE 29
3.2 Огляд програмної частини системи 32
3.2.1 Огляд серверної частини 32
3.2.2 Огляд клієнтської частини 33
3.2.3 Взаємодія з пристроєм 34
3.3. Підключення та живлення системи 38
3.4. Аналіз схеми роботи 44
Висновок до розділу 3 46
РОЗДІЛ 4 ТЕСТУВАННЯ 47
4.1. Демонстрація роботи системи 47
4.2. Аналіз проблематики розпізнавання у ході дослідження 50
4.3. Внесення можливих покращень системи 51
Висновок до розділу 4 54
ВИСНОВКИ 55
СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ 57
ДОДАТОК А Server/app.py 59
ДОДАТОК Б Server/const.py 60
ДОДАТОК В Server/models.py 61
ДОДАТОК Г Server/routes/__init__.py 64
ДОДАТОК Д Server/routes/cdata.py 65
ДОДАТОК Е Server/routes/devicecmd.py 71
ДОДАТОК Є Server/routes/getrequest.py 73

ДОДАТОК Ж Server/routes/ping.py 75
ДОДАТОК И Server/routes/push.py 76
ДОДАТОК І Server/routes/registry.py 78
ДОДАТОК Ї Server/commands/control.py 80
ДОДАТОК Й Server/commands/door.py 81
ДОДАТОК К Server/data/data.py 82
ДОДАТОК Л Server/data/mulcarduser.py 83
ДОДАТОК М Server/data/user.py 85
ДОДАТОК Н Server/data/userauthorize.py 87
ДОДАТОК Н Server/set/set.py 89
ДОДАТОК О Client/app.py 90
ДОДАТОК П Client/templates/index.html 97
ДОДАТОК Р Client/templates/layout.html 98
ДОДАТОК С Client/templates/device.html 100
ДОДАТОК Т Client/templates/parameters.html 102
ДОДАТОК У Client/templates/realtime.html 103

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СКД	Система Контролю Доступу
ADMS	Server for controller
API	Application Program Interface
PIP	Python Package Index (PyPi)
IDE	Integrated Development Environment

ВСТУП

В сучасному світі, питання безпеки та контролю доступу, стає все більш важливим для сфер, такиї як бізнес, освіта, державні та приватні структури. Звичайні методи захисту, які були раніше, типу паролі або ключі, поступово відходять на задній план, через проблему втрат або зламів. На зміну їм, приходять нові технології розпізнавання карток, які поєднуються у собі зручність використання, а також, високий рівень безпеки.

Актуальність моєї роботи пояснюється крайньою необхідністю впровадження нового, надійного та простого рішення для автентифікації користувача, та просто контролю доступу в умовах зростання вимог до безпеки. Якщо розглянути варіант систем розпізнавання карток, то це перспективний напрям, яким може дозволити значно знизити ризик небажаного доступу, спростити процедуру входу в будівлю чии кімнату, а також налаштування доступу користувача відповідно до можливостей контроллеру СКД.

Метою роботи було обрано систему, що автоматично зчитує та розпізнає дані карток користувачів, для надання обмеженого чи необмеженого доступу.

Об’єктом дослідження моєї роботи стане система контролю та управління доступом (СКУД), що базується на технології розпізнавання карток користувачів(згідно [1], [3]). Така система охоплює фізичні модулі – зчитувачі, контроллери, електронні замки, а також програмні модулі, що інтегрують все в одну комплексну систему обробки та аналізу даних, управління правами доступу і збереження інформації про події(згідно [1], [4]).

Предметом дослідження є програмні та апаратні засоби створення автоматизованої системи автентифікації користувачів з допомогою фізичного носія даних, а також способи забезпечення безпеки, точності і надійності при їх використанні. Особлива увага приділяється алгоритмам обробки даних, також способам забезпечення програмної інтеграції з апаратними частинами системи, цілісності та конфіденційності даних, і також можливостям масштабування та

адаптації системи до різних змінних потреб.

Наукова новизна розробки полягає в комплексному підході до створення системи безпеки, що дозволить поєднати програмне рішення та обладнання, для досягнення мети проекту. Використання сучасної технології обробки даних картки, та механізмів фізичного контролю доступу дозволить мені досягти високої точності, надійності, та достовірності даних, при роботі моєї системи.

Практичне значення полягає у створенні прототипу СКД на основі технології розпізнавання карток, що може бути використанна у таких галузях як підприємство, навчальний заклад, держ. установа, тощо.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поставлення мети роботи

Метою розробки цієї системи, було як і власний інтерес до такого продукту, так і створення механізму, який би дозволив використати ідентифікацію карток, для забезпечення достатньо високого рівня безпеки. На відміну від паролів, який до речі, згадуючи особисті випадки, можна доволі легко забути – картка є фізичним носієм даних про власника, яка може легко використовуватись для швидкої та надійної ідентифікації власника. Такий підхід, дозволив використати картки як зручний та простий спосіб ідентифікації користувача, та легко контролювати доступ до будь-якої точки.

Основною вимогою до такої системи, звісно, є можливість зчитувати картку користувача, та порівнювати їх із заздалегідь сформованою базою даних дозволених осіб. У разі збігу в такій базі – система надає доступ, відповідно перемикаючи положення реле.

В системі, центральним контроллером виступає контроллер ZKTeco C2-260, з 2 реле на платі, а для зчитування карток, використовується зчитувач ZKTeco KR503E-RS. З допомогою зчитувача, пристрій отримує дані картки, передає на контроллер, який порівнює із своєю базою даних, та в разі позитивного збігу, подає команду на реле контроллера для перемикання, відповідно до налаштувань пристрою(відповідно до [2], [6], [13]).

1.2. Дослідження предметної області

В цьому розділі буде розглянуто основні аспекти та технології, що будуть стосуватись розробки нашої СКД на базі розпізнавання карток.

Аналіз використовуваних технологій.

Для реалізації цієї СКД ми використаємо кілька основних технологій та засобів:

- Згідно з джерелом [12], HTTP(HyperText Transfer Protocol) вважається протоколом прикладного рівня, який використовується для передачі даних в мережі Інтернет. Він є основою обміну інформацією між клієнтами, у нашому випадку пристроями – контроллерами, та серверами, зокрема для передачі даних з використанням API.

  Основними перевагами HTTP, я вважаю:

  - Простота використання та впровадження, завдяки простій та зрозумілій структурі запитів та відповідей.
  - Підтримка кількох типів даних, наприклад текст, зображення та бінарні дані.
  - Широке розповсюдження та підтримка сучасними пристроями та серверними технологіями, що забезпечать надійність і сумісність при розробці будь-якого проекту.

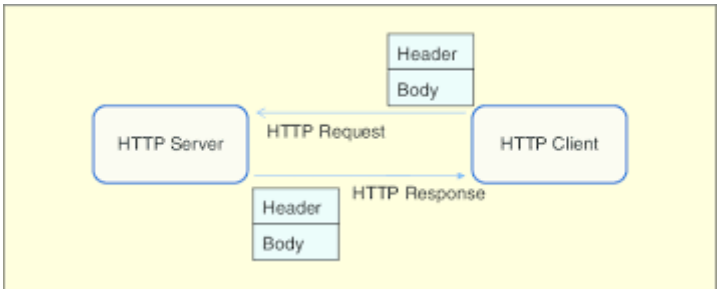

Рис. 1.1 – Схема обміну даними в протоколі HTTP

- RS485 – стандарт комунікаційного протоколу, який використовується для обміну даними між пристроями на великі відстані та при складних умовах навколишнього середовища. Базовими особливостями такого протоколу я вважаю такі можливості:
    - Робота при великих відстаннях – близько 1200 метрів без втрати якості сигналу.
    - Підтримка багатоабонентської топології, що дозволяє підключати декілька пристроїв до однієї шини(Приблизно 32-64 в залежності від пристрою та особливостей виконання).
    - Стійкість до електромагнітних завад, завдяки використанню диференційної передачі сигналу .

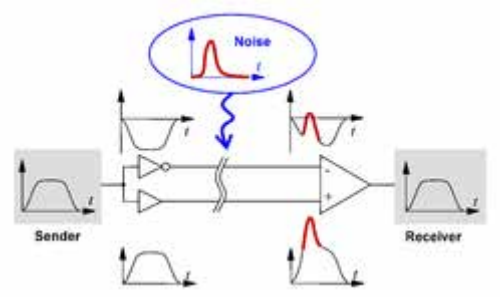

Рис. 1.2 – Зниження шуму за допомогою диференційної передачі сигналів

- Wiegand – популярний протокол обмыну даними, який часто використовуэться у системах контролю доступу для передачы інформації між зчитувачами карток та контролерами:
    - Простота та надійність передачі даних за допомогою спеціальних сигнальних ліній D0 і D1.
    - Обмежена довжина передачі(Орієнтовно 80-150 метрів, в залежності від виробника), але висока стабільність і простота роботи.
    - Стандартна передача фіксованого обсягу інформації(26, 34 або більше бітів), що робить Wiegand сумісним з більшістю різноманітних систем контролю доступу

Опис мови програмування

Python - це високорівнева інтерпретована мова програмування яка призначенна для загального використання. Вона була створена Гвідо ван Россумом у 1991 році. Завдяки простому синтаксису та читабельності коду, разом з гнучкістю, Python став одним з найпоширеніших інструментів розробки сучасних програм. Її активно використовують не тільки дляо аналізу даних або машинного навчання, але і також для розробки веб-додатків, API, сервісів та автоматизації процессів.

Одним з головних, та переважно, корисних моментів, я вважаю це велика к-сть фреймворків для розробки серверних додатків та API. Якщо розглянути, до прикладу, такі фреймворки, як Flask та Fast Api, то вони дозволяють швидко створювати прості веб-сервіси для обробки запитів HTTP, а також REST API. Це важливо для інтеграції різноманітних компонентів системи, організації взаємодії сервісів між собою та керування у розподілених мережах.

Python чудово підходить для роботи з серверною логікою: підтримує одночасну обробку запитів, дозволяє легко працювати з базами даних, керувати підключеннями пристроїв через сокети та реалізовувати асинхронну обробку даних, що важливо для розробки стабільних і масштабованих систем.

Python також має велику стандартну бібліотеку для роботи з протоколами передачі даних, файловими системами, криптографією, мережами та веб-технологіями. Це може дозволити розробникам створювати складні сервісні рішення без необхідності постійно залучати сторонні інструменти.

В контексті моєї роботи, вибір мови Python був обгрунтований необхідністю створення ефективного серверного середовища для організації обробки запитів, взаємодії з пристроями через API, та забезпечення надійної роботи системи включно з простотою проектування.

Опис та принцип роботи RS-485

Стандарт RS-485 розроблявся Інститутом інженерів з електротехніки та електроніки(IEEE), у 1983 році. Незважаючи на те, що з моменту його появи минуло декілька десятиліть, даний інтерфейс залишається одним з найбільш популярних рішень для надійної та, відносно, далекої передачі даних у промисловості, системах контролю доступу та автоматизаціях. Його перевагами я міг би назвати:

- Висока стійкість до перешкод.
- Підтримка великих відстаней.
- Можливість підключення кількох пристроїв до однієї шини.
- Забезпечення стабільної роботи у складних умовах.

Загалом, резюмуючи та поглиблюючись в особливості роботи цього протоколу, можна виділити такі основні пункти:

- Диференціальна передача сигналу.
- Підтримка багатоточкових з’єднань.
- Висока стійкість до електромагнітних завад.

Підтримка багатоточкових з’єднань дозволяє підключати до однієї шини одночасно до 32 (або більше у розширених версіях) передавачів і приймачів. Це робить RS485 ідеальним вибором для створення розподілених мереж контролю доступу або систем збору даних, де важливо передавати інформацію між багатьма пристроями.

Висока стійкість до електромагнітних завад є ще однією перевагою RS485. Завдяки диференціальній передачі навіть у середовищах з сильними промисловими шумами сигнал залишається чітким і надійним. Для забезпечення максимальної ефективності часто використовують екрановані кабелі та правильне завершення ліній спеціальними резисторами.

Таким чином, завдяки поєднанню довгої дистанції, підтримки великої кількості пристроїв і стійкості до завад, інтерфейс RS485 залишається основою для побудови стабільних та ефективних комунікаційних систем у багатьох технічних проєктах.

1.2.4. Опис та принцип роботи Wiegand

Протокол Wiegand був розроблений в 1970-х роках компанією Sensor Engineering (пізніше HID Global) і швидко став стандартом для передачі даних в системах контролю доступу. Незважаючи на свою простоту, протокол Wiegand залишається одним з найпоширеніших рішень у секторі безпеки завдяки своїй надійності, низькій вартості впровадження та сумісності з широким спектром пристроїв.

Основними особливостями протоколу Wiegand я відношу такі моменти як:

- Передача даних по трипровідній схемі(Data0, Data1, Ground)
- Використання методу передачі бітів з допомогою імпульсів
- Фіксована довжина даних для передачі(До прикладу 26, 34, 37 бітів і більше)

Трипровідна передача даних, в нашому випадку, означає, що інформація від зчитувача передається до контроллеру використовуючи окремі сигнальні лінії для нулів та одиниць. Відповідно при наявності імпульсу на лінії D0, вважаємо цей сигнал нулем, при D1, це одиниця. Відсутність сигналу синхронізації робить цей метод надзвичайно простим, але тим часом, чутливим до довжини проводу та якості з’єднання.

Імпульсний метод передачі гарантує стабільний обмін даними на невеликих відстанях, зазвичай від 80 до 150 метрів. Завдяки цьому методу кодування передача даних є простою і не вимагає складної логіки на стороні зчитувача і контролера. Це зробило даний протокол ідеальним рішенням для простих і компактних систем СКД, де є важливим надійність та мінімальні витрати на обладнання.

Фіксована або змінна довжина даних може дозволити адаптувати протокол до конкретних вимог будь-якої системи. Найпопулярнішим форматом такого протоколу є W26, в ньому передається 24 біти корисної інформації та 2 біти парності. Іншим, не менш популярним є W34, в якому вже 32 біти корисної інформації, і 2 біти парності використовуються для передачі даних. Звісно існуються інші формати , які дозволять передавати більше інформації про користувача, але їх ми не розглянули, через їх непопулярність.

Беручи до уваги все вищезазначене, можемо зробити висновок, що завдяки простоті реалізації та інтеграції пристроїв з цим протоколів, поширеність його у майже всіх СКД, цей протокол можна вважати деякою класикою сучасних СКД, а також ефективним рішенням для побудови систем такого класу.

Висновок до розділу 1

У першому розділі було проведено детальний опис предметної області, що дозволив чітко окреслити основні технічні аспекти, пов’язані з розробкою СКД на основі розпізнавання карток.

Було визначено мету створення системи, яка полягала в реалізації надійного, зручного та безпечного способу ідентифікації користувачів за допомогою карток доступу. У порівнянні з стандартними методами, такими як паролі, або фізичні ключі. Цей підхід забезпечує вищий рівень зручності, та знизить ймовірність зловживань або втрату доступу, внаслідок втрати картки.

Розглянуто ключові технології, що лежать в базі функціонуваня мережевого СКД:

- HTTP, для взаємодії з сервером.
- RS-485, для надійної передачі даних на великі відстані.

Ці технології мають свої переваги, які обумовлюють її вибір для мого проекту.

Також, було описано вибір мови програмування Python, яка завдяки своїй простоті, а також великій кількості бібліотек, є оптимальним рішенням для розробки серверної частини системи. Її використання дозволяє швидко створити API, реалізувати обробку запитів, керувати базами даних і здійснювати взаємодію з апаратними компонентами.

Таким чином, отримані результати дослідження предметної області, створюють надійну теоретичну основу для подальшої реалізації програмно-апаратної СКД, що відповідатиме сучасним вимогам безпеки, гнучкості, та масштабованості.

РОЗДІЛ 2 СТРУКТУРА ТА ФУНКЦІОНАЛ СИСТЕМИ

2.1. Етапи розробки системи

Розробку можна поділити на декілька основних етапів, що є достатньо важливими для подальшого проектування системи:

1) Підготовка середовища:
   - Встановлення Python, разом з необхідними бібліотеками та налаштування середовища розробки.
2) Розробка системи розпізнавання карток:
   - Захоплення та попередня даних карток.
   - Виявлення номерів з використанням API.
   - Виділення ключового коду картки та їх порівняння з базою даних.
3) Керування електромагнітним замком:
   - Програмування мікроконтролера для взаємодії з YM-60N.
   - Синхронізація результатів розпізнавання облич з групою доступу для здійснення фізичного контролю доступу.
4) Тестування та оцінка продуктивності:
   - Проведення тестів у різних умовах для оцінки точності та надійності системи.
   - Аналіз результатів та оптимізація системи для покращення її роботи.

2.2. Алгоритм розпізнавання

В контексті розробки системи контролю доступу для торгової організації, з огляду на вимоги до надійності, швидкодії та економічної ефективності, було прийнято рішення про реалізацію механізму розпізнавання на основі карткових технологій. Після аналізу доступних варіантів, зокрема високочастотних (HF RFID, наприклад, Mifare) та низькочастотних (LF RFID, наприклад, EM-Marin) систем, перевага була віддана останнім. Ключовим фактором вибору низькочастотних RFID-карток (125 кГц) стала їхня менша вартість як самих ідентифікаторів, так і зчитувального обладнання, що є суттєвим для бюджету торгових організацій, особливо при оснащенні значної кількості точок проходу. До того ж, для типових завдань контролю фізичного доступу, де не вимагається зберігання великого обсягу даних на картці чи складних криптографічних операцій, функціоналу LF RFID цілком достатньо.

Обраний підхід дозволив ефективно реалізувати алгоритм розпізнавання з використанням наявних контролерів виробництва ZKTeco, які штатно підтримують роботу з низькочастотними зчитувачами та картками. Процес розпізнавання є послідовністю чітко визначених етапів, кожен з яких відіграє критичну роль у забезпеченні коректного функціонування системи. Нижче наведено деталізований опис ключових етапів алгоритму:

Ініціалізація процесу та виявлення картки:

Активізація зчитувача: Контролер ZKTeco підтримує зчитувач RFID у постійному режимі очікування або періодично активує його для генерації електромагнітного поля низької частоти (зазвичай 125 кГц) навколо антени зчитувача.

Входження картки в поле зчитування: Коли RFID-картка потрапляє в зону дії цього поля (зазвичай на відстані кількох сантиметрів), енергія електромагнітного поля індукує струм в антені картки. Для пасивних LF-карток, які не мають власного джерела живлення, ця енергія є достатньою для активації мікрочипа картки.

Відповідь картки: Активований мікрочип картки модулює сигнал, що передається назад на антену зчитувача, транслюючи свій унікальний ідентифікаційний номер (UID).

Зчитування та декодування ідентифікатора картки:

Прийом сигналу: Антена зчитувача приймає модульований сигнал від картки.

Демодуляція та декодування: Електронні компоненти зчитувача (інтегровані в пристрій ZKTeco або як окремий модуль, підключений до контролера) демодулюють отриманий сигнал для виділення цифрових даних. Потім відбувається декодування цих даних відповідно до протоколу, що використовується карткою (наприклад, Manchester кодування для EM-Marin).

Передача ідентифікатора контролеру: Відповідно до [5], [10], [11] Після успішного декодування, унікальний ідентифікатор картки (зазвичай у вигляді послідовності цифр або буквено-цифрового коду) передається від зчитувача до головного модуля контролера ZKTeco для подальшої обробки. Ця передача здійснюється по стандартизованому інтерфейсу, наприклад, Wiegand.

Верифікація ідентифікатора шляхом порівняння з базою даних:

Запит до бази даних: Отримавши UID картки, мікропроцесор контролера ZKTeco звертається до своєї внутрішньої енергонезалежної пам'яті, де зберігається база даних авторизованих користувачів. Ця база даних містить перелік валідних ідентифікаторів карток та пов'язані з ними права доступу (наприклад, дозволені зони, часові обмеження).

Процес порівняння: Алгоритм здійснює пошук отриманого UID у базі даних.

Якщо ідентифікатор знайдено, система перевіряє додаткові параметри: статус картки (чи не заблокована вона), відповідність часовим зонам доступу, рівень доступу для конкретної точки проходу.

Якщо ідентифікатор не знайдено в базі даних, картка вважається неавторизованою.

Прийняття рішення та управління виконавчим механізмом:

Авторизація доступу (Access Granted): Якщо UID картки знайдено в базі даних, картка активна, і користувач має відповідні права доступу для даної точки проходу в поточний час, контролер ZKTeco приймає рішення про надання доступу.

Контролер активує відповідне реле, яке управляє виконавчим пристроєм (наприклад, електромагнітним замком, електромеханічною засувкою, турнікетом), розблоковуючи його на встановлений проміжок часу.

Система фіксує подію успішного доступу у внутрішньому журналі подій (логі) із зазначенням часу, дати, ідентифікатора картки та точки проходу.

Може активуватися візуальна (зелений світлодіод) та/або звукова індикація на зчитувачі.

Відмова в доступі (Access Denied): Якщо UID картки не знайдено, картка заблокована, або права доступу не відповідають умовам, контролер приймає рішення про відмову в доступі.

Виконавчий пристрій залишається заблокованим.

Система фіксує подію відмови в доступі у журналі подій.

Може активуватися відповідна візуальна (червоний світлодіод) та/або звукова індикація.

Цей послідовний алгоритм, реалізований на базі обладнання ZKTeco, забезпечує надійне та оперативне розпізнавання користувачів, що є фундаментальною основою для побудови ефективної системи контролю доступу на об'єкті торгової організації. Кожен етап є важливим для гарантування безпеки та запобігання несанкціонованому проникненню.

2.3. Опис обладнання

У цьому розділі детально розглянемо апаратні компоненти, які використовуються для реалізації системи контролю доступу на базі розпізнавання карток. Основними елементами системи є ПК з серверним програмним забезпеченням, контроллер ZKTeco C2-260 та зчитувач карток ZKTeco KR503E-RS отож розглянемо кожен окремо.

2.3.1. Технічна характеристика C2-260

Представляє собою мініатюрний комп'ютер, який надає високу обчислювальну потужність у компактному форматі. Він є ідеальною платформою для реалізації проектів, що вимагають обробки зображень та контролю апаратури.

Рис. 2.1 – ZKTeco C2-260

Основні характеристики C2-260:

- Процесор: ARM Cortex-A7.
- Оперативна пам'ять: 256MB.
- Накопичувач: Вбудована пам’ять для збереження даних користувачів, карток, журналів подій та налаштувань

Порти та інтерфейси:

- RS485 для підключення зчитувачів та пк
- Wiegand, на платі розширення
- Релейні входи/виходи для підключення замків, кнопок
- Програмовані входи/виходи, для налаштуваннь нестандартних операцій
- Ethernet-порт, для підключення до мережі та віддаленого керування через наше пз.

Додаткові можливості:

- Підтримка локального зберігання даних, на випадок втрати зв’язку з сервером, для подальшої синхронізації.
- Захист даних користувачів за допомогою шифрування.
- Можливість роботи в режимі реального часу із моніторингом подій в стані реального часу.
- Гнучка система прав доступу, підтримка розкладів та зональних обмежень користувачів.

C2-260 підтримує інтеграцію із сучасним програмним забезпеченням для керування доступом (наприклад, ZKBioSecurity), що дозволяє масштабувати рішення відповідно до вимог об'єкта. Завдяки підтримці як мережевої, так і автономної роботи, цей контролер є ефективним рішенням як для малих офісів, так і для великих корпоративних систем безпеки.

2.3.1. Технічна характеристика KR503E-RS

ZKTeco KR503E-RS – це компактний зчитувач RFID-карток, призначений для використання у системах контролю доступу. Завдяки надійній роботі та підтримці комунікації через RS485, цей пристрій добре підходить для інтеграції в професійні мережеві рішення безпеки.

Рис. 2.2 – ZKTeco KR503E-RS

Основні характеристики KR503E-RS:

- Підтримка карток: RFID-картки стандарту EM 125кГц
- Інтерфейс зв’язку: RS485
- Індикація: Вбудовані світлодіоди(LED) для індикації статусу роботи та зчитування картки.
- Звукові сигнали: Наявність бузера(дзвінка) для підтвердження нормального зчитування картки.
- Дальність зчитування: 2-10 см залежно від типу та якості картки.
- Робоча напруга: 12В постійного струму.
- Споживання струму: до 100 мА

Додаткові можливості:

- Захист корпусу: Вологозахищене та пилозахищене виконання стандарту IP65, що дозволяэ використовувати зчитувач як усередині приміщення, так і на вулиці.

- Робоча температура: від -20°C до +65°C, що забезпечує надійну роботу в різних температурних та кліматичних умовах.
- Простота монтажу: Компактний розмір і легке встановлення на стіну або біля дверної шілини.

2.3.2. Технічна характеристика YM-60N

YM-60N – це звичайний базовий електромагнітний замок, який є рішенням від компанії Yli Electronics.

Рис. 2.3 – YM-60N

Основні характеристики:

- Сила утримання: 60кг
- Матеріал корпусу: Анодований алюміній
- Живлення: 12-24В
- Споживання: 12В 160мА, 24В 80мА

2.3.3. Технічна характеристика BGP-125Pro

BGP-125Pro – це імпульсний блок живлення бренду Full Energy.

Рис. 2.3 – YM-60N

Основні характеристики:

- Вхід: ~220В
- Вихід: +12В постійного струму / 5А

Висновок до розділу 2

В цьому розділі було представлено повну структуру та можливості СКД на розпізнаванні карток. Розглянуто ключові етапи реалізації проекту – від підготовки середовища розробки та встановлення ПЗ до програмування керування замку, та проведення тестування системи в різних умовах.

Була описана логіка роботи системи, яка охоплює процес зчитування даних з картки, перевірки її в базі даних, та прийняття рішення щодо надання або відмови в доступі. Такий підхід дозволяє досягти автоматизованого, безпечного та зручного процесу ідентифікації користувача.

Окрему увагу приділено апаратному забезпеченню, яке є основою функціонування системи. Детально описано технічні характеристики кожного елементу:

- Контроллера ZKTeco C2-260
- Зчитувача ZKTeco KR503E-RS
- Електромагнітного замку YM-60N
- Блоку живлення BGP-125Pro.

Перевагами даного обладнання, це підтримка сучасного інтерфейсу зв’язку RS-485, захист выд зовнішніх факторів та енергоефективність.

Таким чином, у розділі було сформовано повне уявлення про функціональні та технічні основи системи, що дозволяє перейти до етапу практичної реалізації, налаштування та подальшої експлуатації рішення. Отримані результати підтверджують доцільність обраної архітектури та обгрутнованість вибору використовуваних компонентів.

РОЗДІЛ 3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Налаштування середовища розробки VSCode IDE

Одним із перших етапів підготовки до запуску програмного забезпечення є налаштування середовища, в якому воно функціонуватиме. Оскільки розроблене ПЗ побудоване з використанням мови програмування Python, яка є міжплатформною, воно не залежить від конкретної операційної системи. Це означає, що система може ефективно працювати як у середовищі Windows, так і на базі Linux, чи macOS, за умови попередньо встановленого інтерпретатора Python відповідної версії.

Після вибору або встановлення необхідної ОС, наступним кроком є встановлення всіх додаткових бібліотек і залежностей, від яких безпосередньо залежить працездатність програмного коду. Для цього зазвичай використовується спеціальний файл requirements.txt, у якому міститься перелік усіх необхідних пакетів з вказанням їх точних версій. Це дозволяє автоматизувати процес встановлення та уникнути помилок, пов’язаних з відстуністю певного модуля, або з конкретними версіями. Процедура встановлення бібліотек виконується за допомогою стандартного пакетного менеджера “pip”, що входить до складу Python. Команда для встановлення пакетів виглядає ось так – pip install -r requirements.txt. Параметр -r, показує що перелік пакетів знаходиться в певному файлі, який ми вказали після аргументу.

Після завершення попереднього етапу та успішного встановлення всіх компонентів, програму можна запускати у відповідному середовищі виконання, що дозволить перейти безпосередньо до запуску.

Такий підхід дозволить забезпечити гнучкість, кросфорплатформність і зручність розгортання системи незалежно від апаратного чи програмного середовища.

Для розробки у рамках проєктування нашого пз було обрано середовище

розробки Visual Studio Code (VS Code), фото інтерфейсу якого представлено на рис.3.1.

VS Code — це потужне, легке та гнучке інтегроване середовище розробки, яке підтримує широкий спектр мов програмування, зокрема Python, що використовується у цьому проєкті. Завдяки своїй адаптивності та великій кількості розширень, VS Code є популярним вибором серед розробників як для невеликих, так і для масштабних проєктів.

Основні можливості Visual Studio Code:

• Підсвічування синтаксису та інтелектуальне автодоповнення коду — забезпечують швидке написання коду та допомагають уникати синтаксичних помилок.

• Вбудований термінал — дозволяє виконувати команди, запускати та тестувати скрипти безпосередньо з середовища розробки, що спрощує робочий процес.

• Плагіни та розширення — VS Code підтримує велику кількість розширень, зокрема офіційне розширення Python, яке додає можливості для роботи з віртуальними середовищами, налагодженням коду, форматуванням, автотестами та запуском скриптів.

• Інтеграція з системами керування версіями (наприклад, Git) — дозволяє відслідковувати зміни у коді, працювати з репозиторіями та управляти версіями проєкту без виходу з середовища.

• Налаштовуваність — середовище легко адаптується під потреби розробника завдяки широким можливостям персоналізації інтерфейсу та налаштуванню процесів.

Завдяки таким можливостям, Visual Studio Code забезпечує зручність, високу продуктивність і гнучкість при розробці програмного забезпечення на Python у рамках даного проєкту.

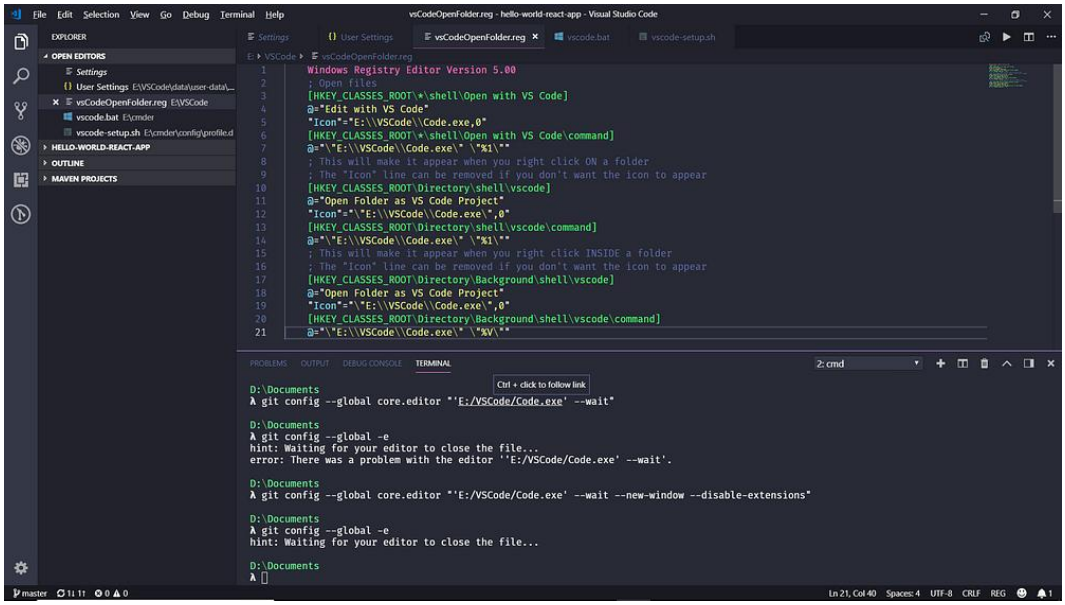

Рис. 3.1 – Інтерфейс VSCode IDE

3.2 Огляд програмної частини системи

Користуючись джерелом [5], програмне забезпечення СКД реалізоване за клієнт-серверною архітектурою. Серверна частина відповідає за прийом даних від пристроїв, їх обробку, та збереження інформації в базу даних, разом з генерацією команд для пристроїв. Клієнтська частина надає веб-інтерфейс адміністрування, з якого можна додавати користувачів, керувати пристроями, переглядати журнали подій тощо.

3.2.1 Огляд серверної частини

Сервер реалізований на основі Flask, з використанням SQLAlchemy як ORM для роботи з базою даних, до речі конфігурація якої виконувалась з допомогою джерела [15] . Конфігурація підключення до бд наводиться у файлі app.py.

Основна логіка взаємодії пристрою з сервером реалізована через маршрути, які описані у відповідному модулі Flask (див. Додаток Г), серед них:

– обробка подій пристрою та завантаження інформації (див. Додаток Д).

- /iclock/devicecmd – передача команд для виконання пристроєм (див. Додаток Е).
- /iclock/getrequest – Отримання черги команд (див. Додаток Є).
- /iclock/push – Ініціалізація налаштувань та параметрів пристрою (див. Додаток И).
- /iclock/registry – Реєстрація нового пристрою та збереження його параметрів у базі (див. Додаток І).
- /iclock/ping – Перевірка доступності сервера для пристрою (див. Додаток Ж).

Бізнес-логіка управління пристроєм реалізована у вигляді окремих класів для керування дверима (див. Додаток Й), оновлення або видалення користувачів (див. Додаток М), даних карток (див. Додаток Л), авторизації доступу (див. Додаток Н) та встановлення параметрів (див.

Додаток Н).

3.2.2 Огляд клієнтської частини

Клієнтська частина мого проекту також побудована на Flask, із шаблонізацією за Jinja2. Основний файл який відповідає за це все – Client/app.py (див. Додаток О).

Основні функції:

- Головна сторінка (/) – Відображає список усіх зареєстрованих пристроїв (див. Додаток П)
- Деталы пристрою (/device/<serial_number>) – Додавання користувача, автоматичне зчитування картки з логів, перегляд останніх подій (див. Додаток С).
- Журнал подій (/device/<serial_number>/logs) – Показує список усіх дій, зафіксованих пристроєм з деталізацією картки, PIN а також типом подій.
- Realtime-mode (/device/<serial_number>/realtime) – Дозволяє моніторити стан дверей у режимі реального часу та керувати замком вручну (див. Додаток У).
- Параметри пристрою (/device/<serial_number>/parameters) – Список внутрішніх параметрів пристрою, які були надіслані ним під час реєстрації (див. Додаток Т).
- Очищення даних пристрою – Повне видалення всіх користувачів, карток і авторизацій.

Також реалізовано обробку POST-запитів:

- Додавання користувача через /device/<serial_number> - включає виклики класів Data.update (з модулю data), для надсилання команд на пристрій.
- Видалення користувача – викликає відповідні методи Data.delete.

3.2.3 Взаємодія з пристроєм

Пристрій надсилає запити до сервера у форматі, описаному в документації виробника. До прикладу:

- Запит конфігурації: GET /iclock/cdata?SN=…&options=all
- Завантаження логів: POST /iclock/cdata?tablename=transaction
- Отримання команд: GET /iclock/getrequest?SN=…

Пристрій надсилає запити до сервера у форматі, описаному в технічній документації виробника(Яку за потреби можу презентувати). Комунікація здійснюється через стандартні HTTP-запити, що забезпечує гнучкість інтеграції з мережевим середовищем та спрощує налагодження. До основних типів запитів, які формує контроллер у процесі роботи, належать:

- GET /iclock/cdata?SN=…&options=all

  У відповідь на цей запит сервер надсилає набір параметрів конфігурації, необхідних для роботи пристрою. Серед них verVersion, TransTables, SessionID, TimeoutSec, тощо. Відповідь формується у файлі cdata.py (див. Додаток Д), а самі параметри можуть бути збережені або згенеровані на основі класу DeviceConfig (див. Додаток В).

  У випадку зчитування картки, відкриття дверей чи навіть виникнення події, пристрій надсилає серверу запит-лог у вигляді текстових рядків, наприклад: cardno=0412548745 event=200 pin=0001 timestamp=2024-12-10 14:22:10. Ці логи обробляються у тій же функції cdata (див. Додаток Д), де вони розбираються на ключі за допомогою регулярних виразів, а потім зберігаються у таблиці DeviceEventLog бази даних (див. Додаток В). Лог зберігається у вигляді транзакцій, що дозволяє його повторно оброблювати на клієнтській стороні.

- GET /iclock/getrequest?SN=…
  Контроллер періодично звертається за новими командами. Сервер перевіряє, чи існують у черзі недоставленні команди з флагом DeviceCommand.delivered=False, для заданого пристрою та повертає першу з них у вигляді рядка формату C:ID:CMD CMD_DETAIL. Команду позначають як доставлену. Цей механізм дозволяє передавати інструкції типу «відкрити двері», «додати користувача» тоцо. Відповідний обробник знаходиться у файлі getrequest.py (див. Додаток Є).

  Цей endpoint обробляє зворотний зв’язок від пристрою після виконання команди, яка була надіслана через getrequest. Контролер надсилає результат виконання команди (Return=0, ID=...), після чого сервер оновлює стан команди в базі, зазначаючи, що вона була доставлена та виконана. Код реалізовано у devicecmd.py (див. Додаток Е).

  Контролер періодично надсилає ping-запити для підтвердження доступності сервера. Сервер повертає відповідь без тіла, лише з технічними заголовками. Даний запит може використовуватися для перевірки зв’язку. Реалізовано у ping.py (див. Додаток Ж).

  Цей endpoint використовується для ініціалізації конфігурації пристрою з боку сервера. Якщо для пристрою ще не створено об’єкта DeviceConfig, він створюється із типовими значеннями.

Повертається конфігурація, аналогічна тій, що у відповіді на /cdata?options=all, але через POST. Реалізація — push.py (див. Додаток И).

При реєстрації пристрою (перший запуск), він надсилає POST-запит із набором параметрів у вигляді рядка, поділеного комами. Сервер зберігає ці параметри у таблицю DeviceParameters, а також створює новий об’єкт пристрою, якщо SN ще не існує. Повертається спеціальний RegistryCode. Реалізація — у registry.py (див. Додаток І).

Клієнтська частина веб-інтерфейсу використовує цей endpoint для зчитування останньої картки, яка не була ще додана до системи. Сервер перевіряє журнали подій за останні 5 секунд і повертає номер картки, якщо така знайдена. Реалізація — у client/app.py (див. Додаток О).

Цей запит надсилається з веб-інтерфейсу вручну, щоб відкрити двері. Сервер додає відповідну команду в чергу пристрою (DeviceCommand з типом CONTROL DEVICE). Команда буде оброблена при наступному запиті getrequest. Код — у client/app.py.

Цей endpoint відповідає за додавання користувача до пристрою через веб-інтерфейс. Користувач вводить ім’я, PIN і номер картки, після чого сервер генерує декілька команд для додавання

користувача, авторизації та карти. Дані фіксуються у базі. Логіка — у client/app.py.

Виводить журнал подій із логів пристрою. Сервер витягує останні DeviceEventLog для пристрою і парсить поля cardno, pin, event, щоб показати у таблиці. Реалізація — у client/app.py та шаблоні

Повертає значення всіх збережених параметрів пристрою з таблиці DeviceParameters. Дані використовуються для діагностики конфігурації. Код — у client/app.py.

Цей endpoint відкриває інтерфейс для моніторингу стану дверей у реальному часі. Дані отримуються періодичними запитами до /state. Код — у client/app.py і шаблоні realtime.html.

Повертає останній статус реле дверей (відкрито/зачинено), парсингом останнього запису в логах. Даний запит використовується для оновлення індикатора на сторінці Realtime. Код — у client/app.py.

Головна сторінка керування пристроєм — додавання користувача, перегляд останньої картки, доступ до кнопки "Зчитати з логу". Код реалізації — у client/app.py, шаблон device.html.

- GET /device/<serial_number>/clear
  Очищення всіх користувачів, авторизацій і карток на пристрої. Сервер додає відповідні DELETE-команди в чергу. Логіка — у

  Повне видалення всіх користувачів через кнопку "Очистити всіх" у веб-інтерфейсі. Аналогічно попередньому, але викликається іншим способом.

  Видаляє конкретного користувача з пристрою і бази, за його ID. Видаляються всі пов’язані записи. Код — client/app.py.

3.3. Підключення та живлення системи

Спершу варто зазначити, що розробка системи контролю доступу включає інтеграцію всіх зазначених компонентів у єдину систему. На цьому етапі важливо правильно підключити всі компоненти та забезпечити їм відповідне живлення.

Підключення магнітного замку до системи:

Беручи до уваги Рис3.2, можна приблизно побачити, яким чином необхідно підключати електромагнітний замок, для правильної роботи в парі з контроллером. Оскільки контроллер керує зовнішніми замками з допомогою реле, а нашому замку необхідні постійне живлення, то підключаємо наш замок з допомогою додаткових контактів контроллера Vout, а також використовуємо бортове реле(Нам необхідні контакти COM(Центральний) – NC(Normal Closed, нормально замкнений контакт))

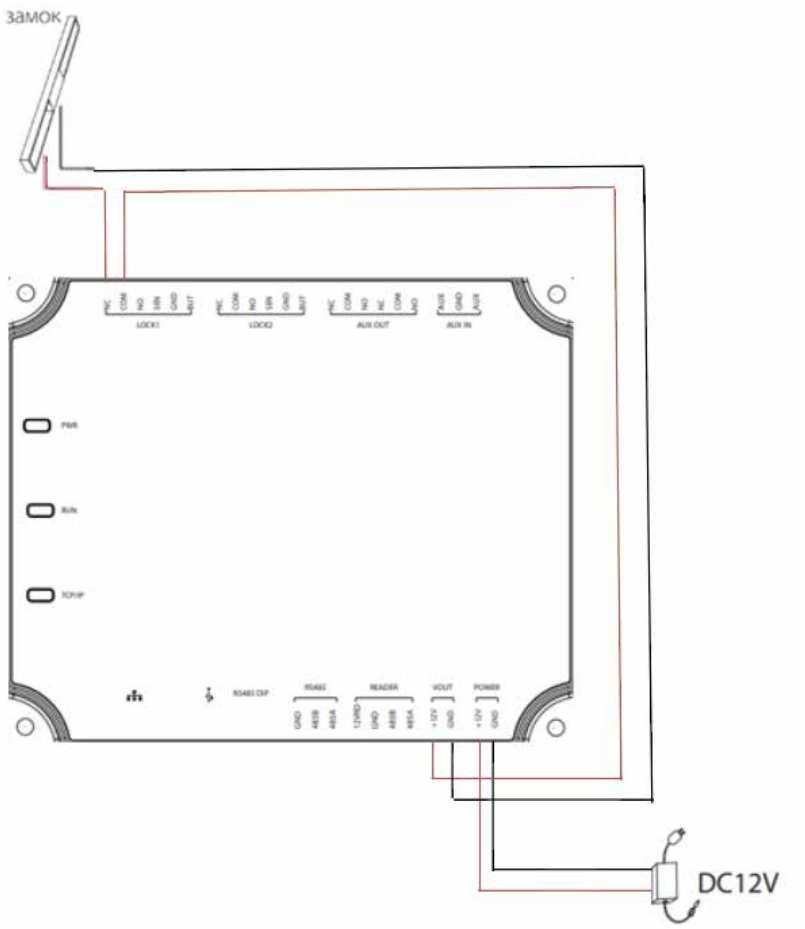

Рис. 3.2 – Схема підключення електромагнітного замку

Підключення зчитувачів до контроллера:

Для правильної роботи декількох зчитувачів, нам необхідно перед монтажем виставити адреси кожного зчитувача, це допоможе їм не конфліктувати в мережі, мати двухсторонній зв’язок, а також обмінюватись даними.(1 зчитувач 1000, порядок бітів інвертований, 2 - 0100)

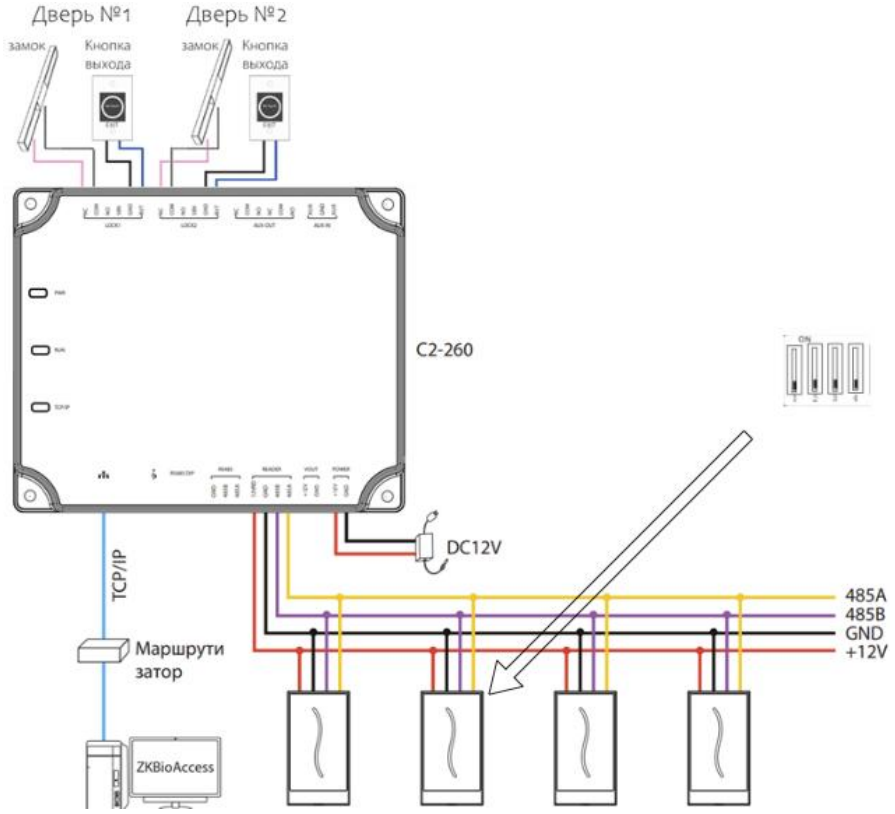

Рис. 3.3 – Схема підключення контроллера та зчитувача

Заживлення системи:

Оскільки контроллер та електромагнітний замок живляться від 12В постійного струму, тому просто підключаємо все паралельно до одного блоку живлення(Наше сумарне використання буде до 1А в простої, і до 1.5А в момент обміну даними/перемикання реле)

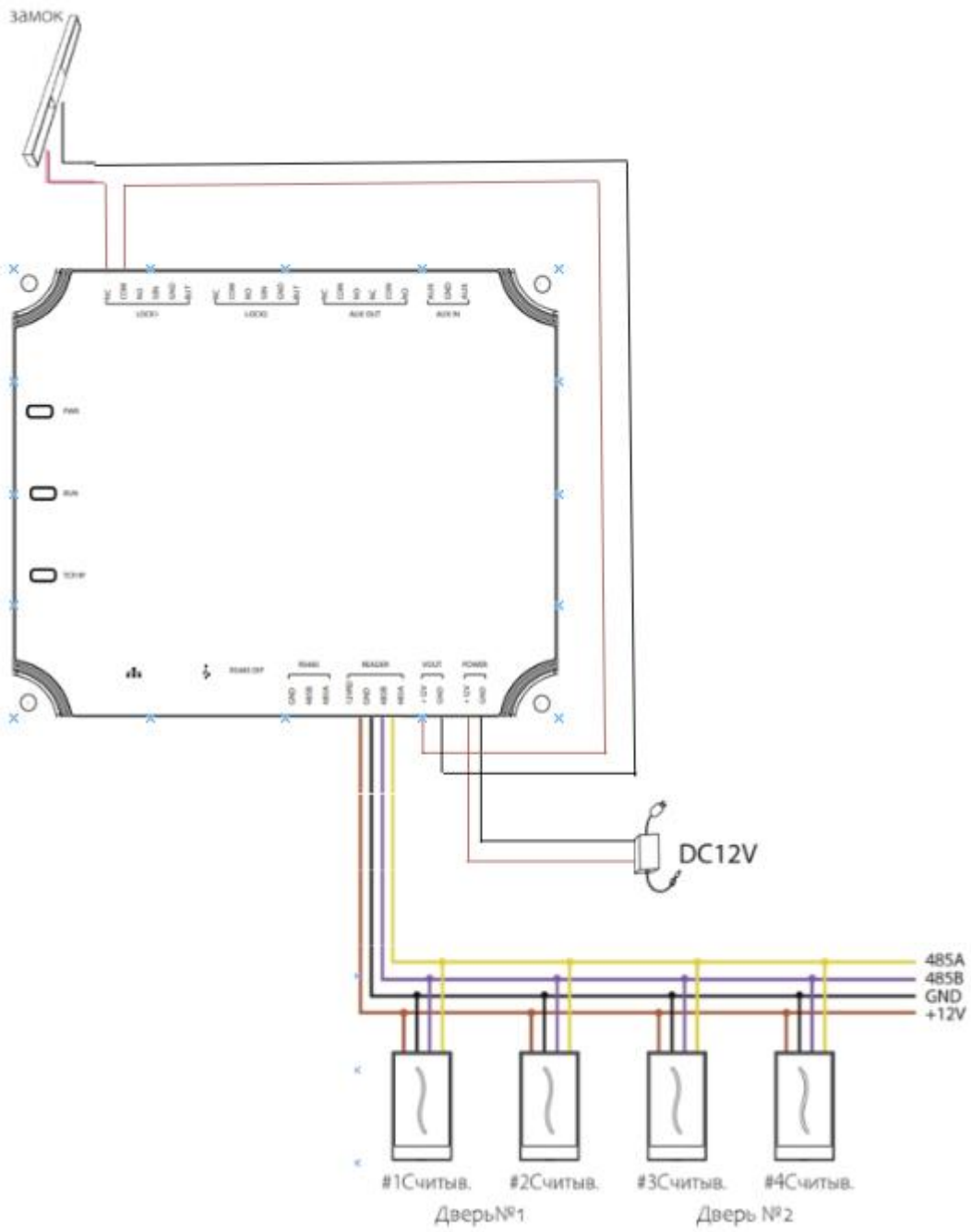

Рис. 3.4 – Фінальна схема комутації проекту

Монтаж системи:

Якщо додати трохи декоративних накладок, виходить більш красиво ніж просто монтаж типу “на руках”, це наглядно можна побачити на рисунку 3.5

Рис. 3.5 – Наглядний монтаж системи

З’єднання з сервером:

Після послідовного з’єднання всіх елементів, з допомогою ПЗ - кофнфігуратора(CommunicationSettingTool, компанії виробника), нам необхідно налаштувати адресу нашого сервера, який буде керувати даним контроллером. На рисунку 3.5 можемо побачити інтерфейс програми.

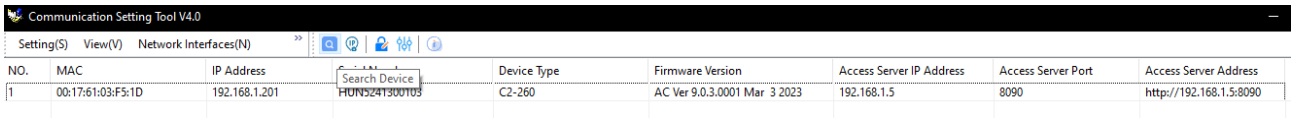

Рис. 3.5 – CommunicationSettingTool інтерфейс

З допомогою цього пз, ми можемо змінити статичну адресу контроллеру, а також адресу, порт, і шифрування серверу, на рисунку 3.6-3.7 ми можемо побачити як це робиться

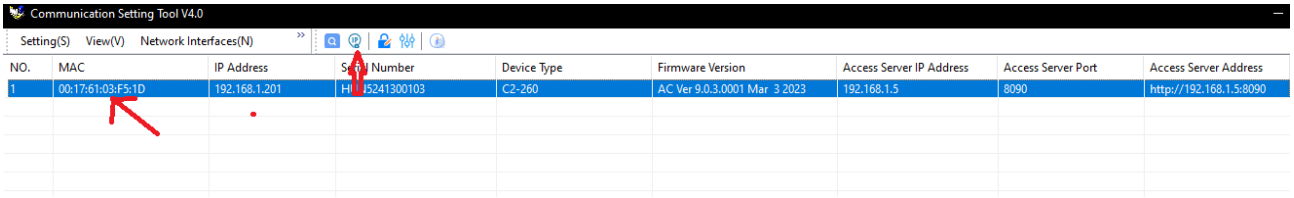

Рис. 3.6 – Послідовність дій для зміни мережевих налаштувань

З наданих програмою даних, які ми бачимо на Рис 3.6-3.7, ми розуміємо що контроллер має адресу – 192.168.1.201(яка є заводським значенням), значення нової IP адреси я задаю таке-ж, за відсутності потреби в зміні адреси пристрою. Gateway, це наша адреса роутера, який надає доступ до глобальної мережі(Так-так, контроллер може спілкуватись з віддаленим сервером, не обов'язково щоб він був поряд).

А тепер, найнеобхідніше для нас – Access Server Ip, Port, та перемикач HTTP/HTTPS. IP адреса, це адреса серверу, з яким контроллер буде обмінюватись даними, порт пишемо тей, що використовується на Серверній частині, це важливо(в мойому випадку, порт 8090). І перемикач HTTP/HTTPS. Контроллер підтримує шифрований обмін даними, що дозволяє приховати дані від атак типу "Man in the middle". Але це вимагає підвищити складність та відладку проекту, через що мии не змінюємо цей параметр(Для спрощення роботи з контроллером).

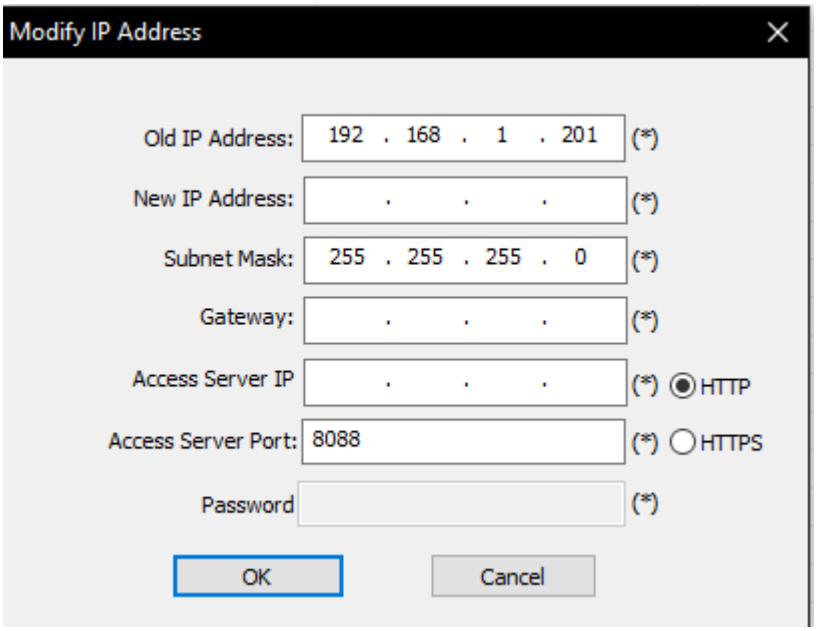

Рис. 3.7 – Параметри для зміни в програмі

3.4. Аналіз схеми роботи

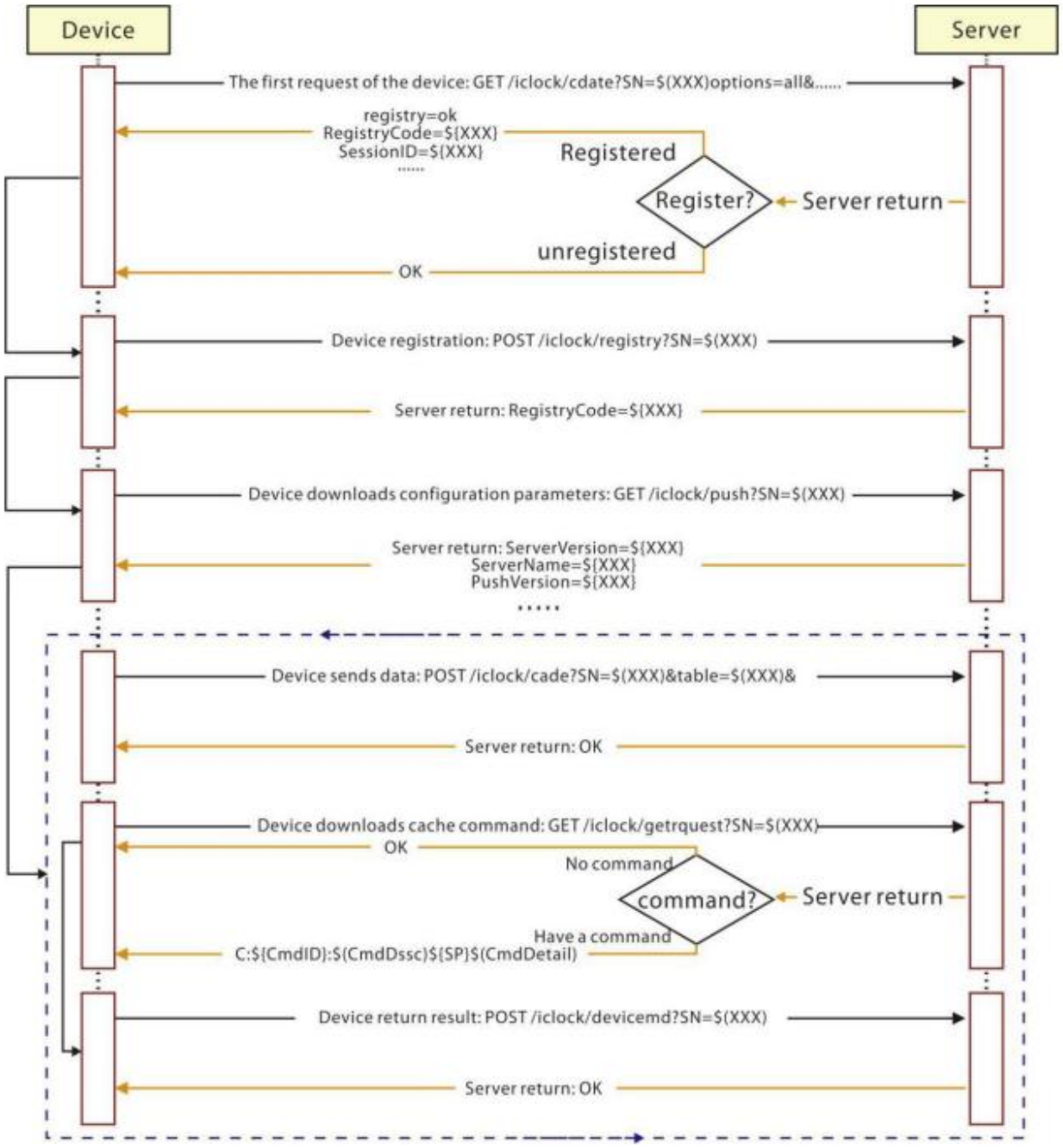

Рис. 3.8 – Схема обміну даними між пристроєм та сервером

Процедура базового обміну даними, представлена на зображенні, описує етапи взаємодії між пристроєм і сервером в системі контролю доступу.

1. Перший запит від пристрою (GET /iclock/cdate):

   Пристрій надсилає запит на сервер для перевірки статусу реєстрації.

   Сервер відповідає, зазначаючи, чи пристрій зареєстрований. Якщо він зареєстрований, сервер повертає код реєстрації та ідентифікатор сесії.

2. Реєстрація пристрою(POST /iclock/registry):

   Якщо пристрій не був зареєстрований, він пройде процес ініціації на сервері. В кінці сервер поверне код реєстрації.

3. Завантаження конфігураційних параметрів(GET /iclock/push):

   Після реєстрації, пристрій завантажує параметри конфігурації, для прикладу наведено версія сервера, ім’я сервера та версія протоколу push, який використовує пристрій.

4. Надсилання даних пристроєм(GET /iclock/cdate with options):

   Пристрій починає завантаження наявних даних, зокрема інформації про події та користувачів. Сервер підтверджує отримання даних.

5. Завантаження кешованих команд(GET /iclock/getrequest):

   Сервер на даному етапі надсилає пристрою команди, які пристрій може прийняти тільки у відповідь на цей запит.

6. Підтвердження виконання команди(GET /iclock/devicecmd):

   Пристрій надсилає інформацію про статус обробки команд(Якщо 0, команда виконана).

Ця процедура дозволяє запезпечити двосторонній обмін даними між пристроєм та сервером, що являється базовою складовою обміну даними цього проекту. Це дозволить мені налаштувати отримання конфігурації, реєстрацію, передачу та отримання команд на виконання.

Висновок до розділу 3

В цьому розділі було розглянуто практичні аспекти проектування та реалізації СКД. Було детально описано етапи налаштування середовища розробки, що є критично важливими для забезпечення корректної роботи програмного забезпечення.

Особливу увагу було приділено вибору середовища розробки VSCode, яке завдяки своїй гнучкості, інтелектуальним підказкам та інтеграції з системою керування версіями, значно полегшує процесс написання, тестування та налагоджування коду.

Важливою частиною розділу стало описання процесу підключення фізичних компонентів, серед яких контроллер, зчитувачі, замок та блок живлення. Було наведено схеми комутації, враховано параметри живлення та особливості інтерфейсів зв’язку, що дозволяє забезпечити стабільну й надійну роботу всієї системи.

Крім того, було докладно обгрунтовано процедуру налаштування мережевих параметрів контроллеру та його зв’язку з сервером за допомогою спеціалізованого ПЗ, а також механізм обміну даними між пристроєм і сервером. Це включає етапи ініціалізації, реєстрації, передачі даних, виконання команд та підтвердження виконання, що забезпечує повноцінну двосторонюю комунікацію.

Загалом, реалізована система має високу гнучкість у налаштуванні, просту у використанні архітектуру та забезпечує всі необхідні функції для ефективного управління доступом. Завдяки детально продуманій інтеграції програмної та апаратої частин, розробка демонструє стабільну роботу та готовність до розгортання у реальних умовах.

РОЗДІЛ 4 ТЕСТУВАННЯ

4.1. Демонстрація роботи системи

Перший запуск відбувається запуском серверного та клієнтського ПЗ.

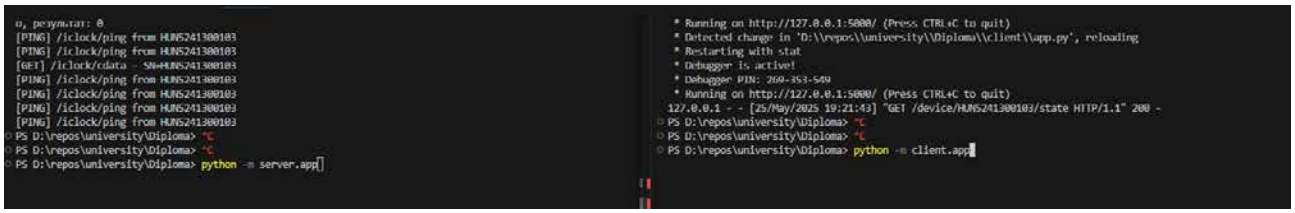

Рис. 4.1 – Запуск клієнт-серверного ПЗ

Для підтвердження працездатності розробленої системи було проведено демонстрацію функціональності веб-інтерфейсу управління системою контролю доступу на основі карток. Програмне забезпечення розгорнуто локально, з використанням Flask-сервера, бази даних SQLite та взаємодії з пристроєм через програмні команди.

Після запуску системи у браузері відображається головна сторінка, що містить перелік доступних пристроїв. Кожен пристрій має посилання на кілька вкладок:

1. Додавання користувача
    - Вибравши вкладку, користувач може ввести ім’я, PIN-код та номер картки.
    - Передбачено можливість автоматичного зчитування останньої активної картки з логів пристрою.
    - Після підтвердження, новий користувач додається в базу даних і авторизується для всіх дверей пристрою.
2. Realtime-моніторинг

- У цьому режимі відображається поточний стан дверей пристрою (відчинено / зачинено), який оновлюється кожні 2 секунди.
- Додано інтерактивну кнопку «Відкрити двері», яка надсилає відповідну команду пристрою.

3. Журнал подій
    - Виводиться таблиця останніх подій з логів пристрою.
    - Для кожного запису зазначено час, картку, PIN та тип події.
4. Параметри пристрою
    - Відображаються усі параметри пристрою, які зберігаються в базі. роботи тощо.
5. Управління користувачами
    - У розділі «Додавання користувача» також реалізовано таблицю з усіма поточними користувачами.
    - Для кожного доступна дія «Видалити», а також кнопка для масового видалення всіх користувачів.

Тестування

Було змодельовано ситуації:

- Додавання нового користувача з автоматичним зчитуванням картки.
- Перегляд змін у журналі подій у реальному часі.
- Надсилання команд на відкриття дверей.
- Видалення одного або всіх користувачів з пристрою.

Програма продемонструвала стабільну роботу, своєчасну реакцію на події пристрою та коректну взаємодію з базою даних. Перевірити це я зміг, з допомогою доступу виробника в прошивку, та вивантаженні даних контроллеру, для повної звірки даних. Ці дані ми можемо побачити на рисунках 4.2 – 4.5.

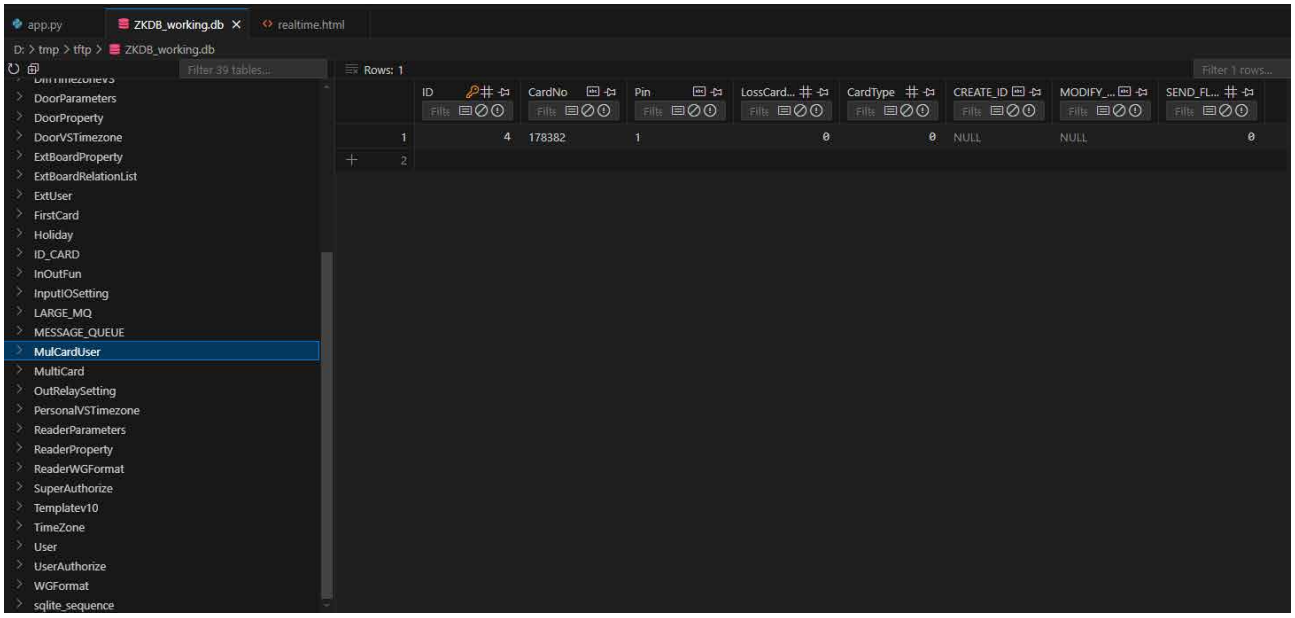

Рис. 4.2 – Налагоджувальна інформація прошивки контроллеру

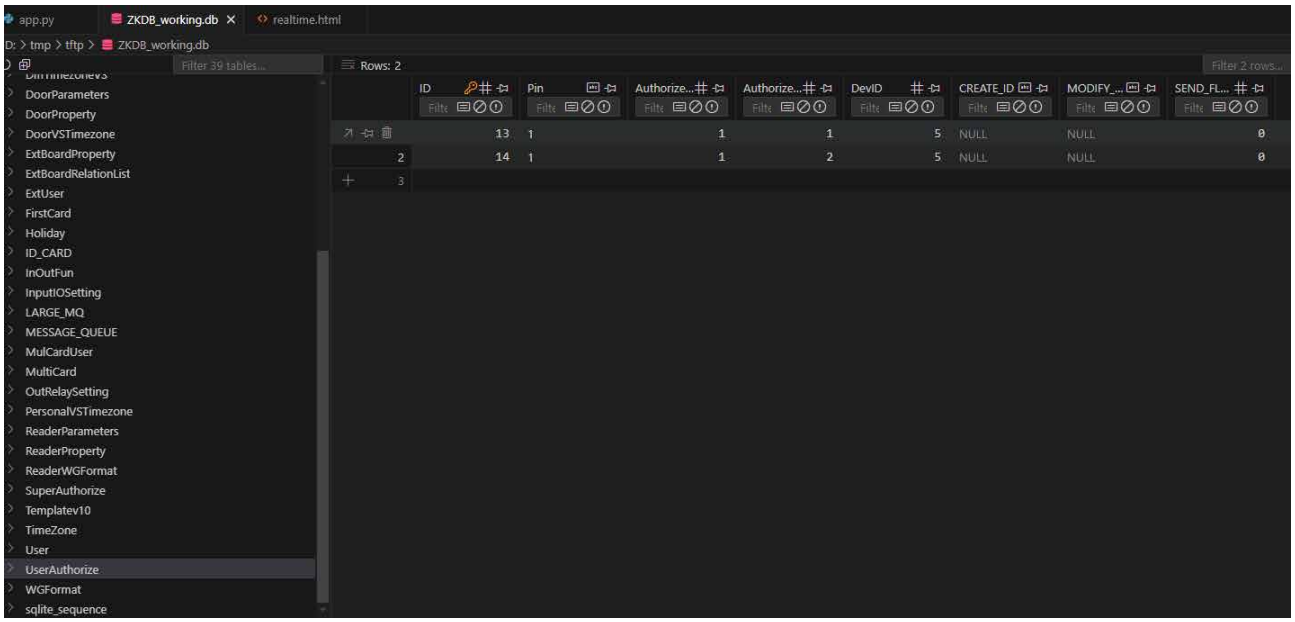

Рис. 4.3 – Налагоджувальна інформація прошивки контроллеру

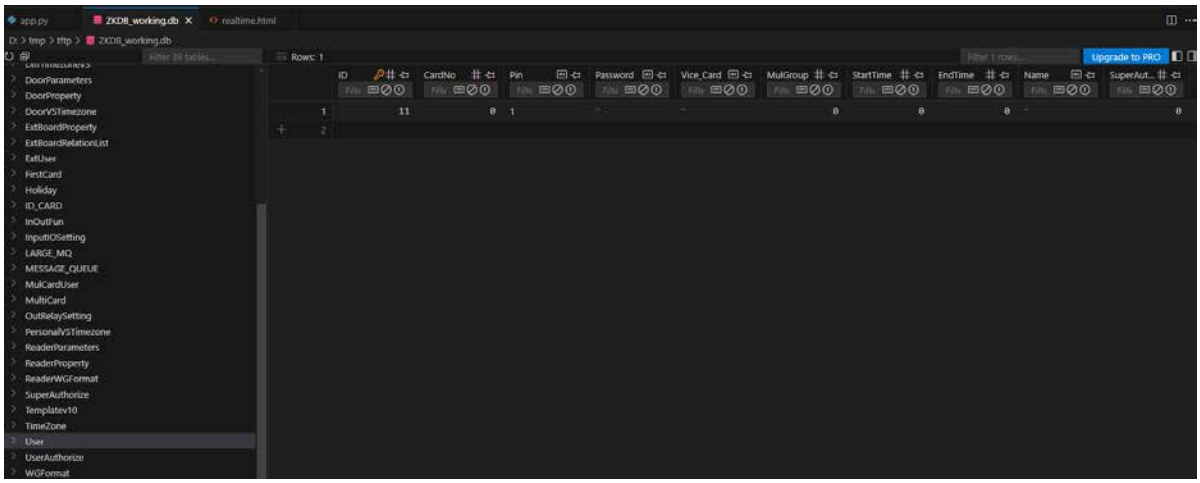

Рис. 4.4 – Налагоджувальна інформація прошивки контроллеру

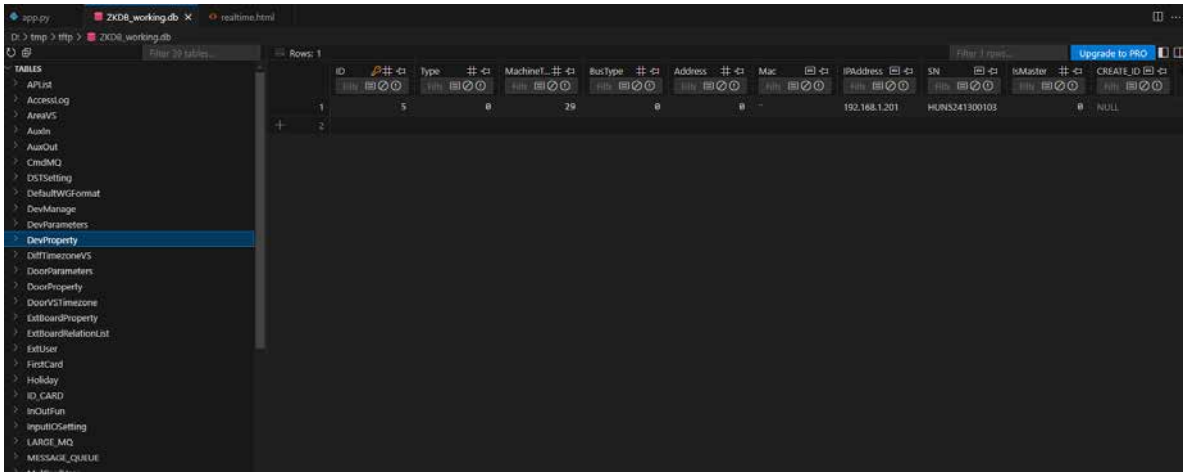

Рис. 4.5 – Налагоджувальна інформація прошивки контроллеру

4.2. Аналіз проблематики розпізнавання у ході дослідження

У ході дослідження та реалізації системи контролю та управління доступом(СКУД) з використанням безконтактних карток стандарту EM-Marine виявлено низку технічних та організаційних проблем, які істотно впливають на якість розпізнавання та обробки подій доступу.

Технічні аспекти проблеми

Однією з основних проблем є затримка або відсутність миттєвого оновлення логів подій при використанні апаратних зчитувачів. У реальних умовах пристрій не завжди синхронізується з базою даних миттєво, що ускладнює оперативне реагування системи, зокрема при додаванні нових користувачів на основі карток, не внесених до системи заздалегідь.

Для вирішення цієї проблеми було запропоновано програмну реалізацію механізму відкладеного опитування логів (polling) протягом фіксованого часового інтервалу (наприклад, 5 секунд). Такий підхід дозволяє "впіймати" запис про нову картку, навіть якщо лог з’явився із запізненням.

Крім того, структура логів подій може бути неуніфікованою. Наприклад, не всі рядки містять однакові ключі (cardno=, event=, pin=), а деякі можуть містити неповні або пошкоджені записи. У дипломному проєкті це вирішувалось шляхом реалізації алгоритму парсингу із розбиттям рядків за табуляцією та

побудовою словника параметрів на основі наявних ключів.

Ще однією критичною вимогою є перевірка унікальності картки перед її додаванням до системи. При кожному запиті система перевіряє, чи не використовується ця картка вже іншим користувачем, що дозволяє уникнути дублювання облікових записів.

4.3. Внесення можливих покращень системи

У процесі розробки було реалізовано базовий функціонал для керування пристроєм контролю доступу. Однак із точки зору практичного використання та масштабованості система може бути значно розширена. Нижче описано перелік найбільш доцільних і реалістичних покращень, які можна впровадити в подальшому розвитку проєкту.

1. Підтримка декількох пристроїв у паралельному режимі

Наразі інтерфейс системи дозволяє обирати один пристрій та взаємодіяти з ним у рамках окремих сторінок. Для підприємств із кількома точками контролю бажано реалізувати:

- одночасний моніторинг кількох пристроїв;
- глобальне адміністрування користувачів, які мають доступ до кількох дверей на різних пристроях;
- групування пристроїв за зонами (наприклад, “Склад”, “Офіс”, “Серверна”). Це дозволить створити централізовану багатозонну СКУД.

2. Покращення зовнішнього вигляду та UX/UI інтерфейсу

Поточний інтерфейс є функціональним, проте з точки зору користувача може бути покращений за допомогою:

- адаптивної верстки (краща підтримка мобільних пристроїв);
- додавання піктограм, індикаторів стану дверей, кольорової індикації тривог;
- реалізації "живих" віджетів (наприклад, зелений індикатор при

авторизації, червоний — при відмові);

- інтеграції темного режиму або кастомних тем інтерфейсу;
- панелі швидких дій: відкриття дверей, блокування, створення нового користувача з одного екрану.

3. Налаштування програмованих подій (сценаріїв)

Система може бути доповнена модулем програмованих подій — умов, при яких відбувається певна дія. Наприклад:

- при авторизації певного користувача — відкривати двері + вмикати світло;
- при тривозі (невдала спроба входу) — надсилати email або Telegram-сповіщення;
- розклад подій: відкриття дверей автоматично в певний час (для техперсоналу, прибиральників тощо);
- реалізація "anti-passback" (повторна спроба входу без виходу).

Це може бути реалізовано через простий конструктор умов/дій у вебінтерфейсі.

4. Підключення до програмованих входів/виходів пристрою

Більшість сучасних контролерів доступу мають дискретні входи/виходи, які можна використовувати для:

- підключення кнопки виходу;
- використання датчика стану дверей (двері відкриті або замкнені);
- активації зовнішньої сигналізації або реле;
- обробки подій з зовнішніх пристроїв (наприклад, охоронної системи або датчика диму). У системі варто реалізувати логіку реакції на вхідні сигнали та інтерфейс для їх конфігурації.

5. Додавання журналу системних подій та аудиту

Для підвищення безпеки та контролю варто реалізувати логування:

- усіх змін в базі користувачів;
- подій авторизації;
- системних помилок та втручань адміністратора;
- дій з відкриттям дверей вручну.

Це дозволить не лише покращити безпеку, а й створить підстави для аналізу подій у разі інцидентів.

6. Інтеграція з месенджерами / email для сповіщення

Корисною є реалізація сповіщень через:

- Telegram-бота для повідомлень про спробу доступу;
- email-розсилку адміністратору при спробі доступу заблокованим користувачем;
- Web Push або SMS.

Це зробить систему більш оперативною у випадку позаштатної ситуації.

7. Підтримка резервного копіювання та експорту даних

Користувачі, логі, налаштування пристроїв мають зберігатися з можливістю:

- автоматичного резервного копіювання;
- експорту в CSV, JSON або PDF;
- переносу бази між серверами.

Висновок до розділу 4

В цьому розділі було проведено тестування реалізованої системи контролю доступу, що дозволило на практиці перевірити її функціональність, стабільність роботи, а також визначити її сильні сторони й потенційні напрямки для вдосконалення.

В результаті демонстрації роботи, клієнт-серверного ПЗ підтверджено працездатність основних модулів – від зчитування карток та додавання користувачів до журналювання подій та інтерактивного керування пристроєм з допомогою веб-інтерфейсу. Система успішно реагувала на запити, обробляла події доступу в реальному часі, та взаємодіяла з базою даних без збоїв.

Під час тестування було виявлено низку технічних нюансів, зокрема затримку оновлення логів та варіативність структури подій, що потребувало реалізації додаткової логіки обробки, включаючи відкладене опитування та валідацію даних. Завдяки додатковим доробкам, така система набула більшої гнучкості та стабільності в роботі.

Крім того, було сформульовано ряд можливих покращень, спрямованих на підвищення зручності, масштабованості, та функціональності системи. Серед них – підтримка декілької пристроїв, розширення інтерфейсу, створення механізму програмованих сценарціїв, інтеграція з зовнішніми системами та месенджерами, а також забезпечення резервного копіювання та аудиту подій.

Таким чином, проведене тестування підтвердило доцільність обраного технічного рішення, виявило шляхи оптимізації, і створило передумови для подальшого розвитку системи відповідно до потреб користувача та безпеки.

ВИСНОВКИ

Результатом виконання даної бакалаврської роботи став детальний аналіз та загальна концепція розробки комп'ютерної системи доступу.

У рамках дипломного проєкту було розроблено, реалізовано та протестовано інформаційно-керуючу систему для контролю доступу на основі мережевого контролера ZKTeco C2-260 із підключеними зчитувачами та периферійними пристроями. Система забезпечує повноцінний веб-інтерфейс адміністратора, який дозволяє в режимі реального часу управляти пристроєм, переглядати події, керувати користувачами та параметрами контролера.

У ході реалізації було створено серверну частину на базі фреймворку Flask із використанням SQLAlchemy для роботи з базою даних. Використовуючи [14], було реалізовано REST-інтерфейси для обміну з контролером, що дає змогу керувати пристроями, зчитувачами та обробляти події доступу. Особливу увагу було приділено роботі з логами, які надходять від пристрою. Зокрема, впроваджено механізм асинхронного опитування, що дозволяє зчитувати нові події із затримкою до 5 секунд, а також алгоритми для розпізнавання карток, що ще не зареєстровані в системі.

Веб-інтерфейс передбачає окремі модулі для додавання користувачів, перегляду логів подій, моніторингу стану дверей, управління параметрами пристрою, а також ручного керування відкриттям дверей. Крім того, реалізовано можливість видалення всіх або окремих користувачів з бази, що є важливою функцією для адміністратора системи.

Система також враховує специфіку роботи контролера ZKTeco C2-260, зокрема його підтримку багатьох дверей, можливість підключення до зовнішніх пристроїв через дискретні входи/виходи, та гнучке управління через локальну мережу. На відміну від комерційних систем, запропоноване рішення не вимагає встановлення складного або платного ПЗ, і є повністю відкритим для модифікацій.

Розроблений програмний комплекс можна вважати одним з найпростіших

для впровадження варіантів СКУД, який водночас забезпечує високу функціональність. Він підходить для використання як у корпоративному середовищі, так і в умовах невеликих офісів, навчальних закладів або приватних об'єктів. Завдяки відкритості коду та модульній структурі систему легко адаптувати до нових потреб, підключити зовнішні датчики, реалізувати реакції на програмовані події або інтегрувати з іншими сервісами.

Результати дипломного проєкту підтвердили ефективність поєднання доступного апаратного рішення з гнучким веб-сервером, що дає змогу побудувати масштабовану систему контролю доступу з мінімальними витратами.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

мперія Безпеки. Система контролю доступу (СКУД): принцип дії, склад,

adzor.ua. Контроль доступу: основні принципи роботи та особливості

mark.ua. Навіщо потрібна система контролю та управління доступом

vel.com.ua. СКУД: види та відмінності систем контролю доступу.

ID Global. Wiegand Interface Technical Specification. [Електронний ресурс].

KTeco PUSH SDK 3.1.1 Documentation. [Електронний ресурс]. – Режим доступу: електронне посилання відсутнє, тому буде прикладений файл виробника окремо.

tional Instruments. What is RS-485? [Електронний ресурс]. – Режим

exas Instruments. RS-485 Design Guide. [Електронний ресурс]. – Режим доступу: https://www.ti.com/lit/an/slla272c/slla272c.pdf

IA/TIA Standard RS-485 – Electrical Characteristics of Generators and

YPERLINK "https://www.hidglobal.com/sites/default/files/resource_files/9000-

ccess Control Systems – Design and Implementation, Second Edition. ISBN

l

q

s

k

QLite Documentation. [Електронний ресурс]. – Режим доступу:

D

o

c

u

m

e

n

t

a

t

i

o

n

.

[

E

л

e

к

т

p

o

н

н

ДОДАТОК А Server/app.py

```
# імпорти
from flask import Flask
from server.routes import register_routes
from server.models import db

app = Flask(__name__)

# налаштування підключення до бд
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///database.db"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

# ініціалізація бд
db.init_app(app)

# прибрати логування кожного запиту по апі
import logging
log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR)

register_routes(app)

# старт серверу
if __name__ == "__main__":
    with app.app_context():
        db.create_all()
    app.run(host="0.0.0.0", port=8090)
```

ДОДАТОК Б Server/const.py

```
server_version="3.1.1"
server_name="ADMS"

# Тут описано базові змінні які використовуються в різних місцях коду.
Зроблено для того, щоб змінити в одному місці, але змінило всюди по коду
```

ДОДАТОК В Server/models.py

```
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime, timezone

db = SQLAlchemy()

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    pin = db.Column(db.String)
    name = db.Column(db.String)
    card_number = db.Column(db.String)
    device_sn = db.Column(db.String, db.ForeignKey('devices.serial_number'))
    device = db.relationship("Device", back_populates="users")

class Device(db.Model):
    __tablename__ = 'devices'

    id = db.Column(db.Integer, primary_key=True)
    serial_number = db.Column(db.String, unique=True, nullable=False)
    name = db.Column(db.String)
    registry_code = db.Column(db.String)
    session_id = db.Column(db.String)

    logs = db.relationship("DeviceEventLog", back_populates="device", lazy=True)
    users = db.relationship("User", back_populates="device", lazy=True)
    configs = db.relationship("DeviceConfig", back_populates="device", lazy=True)
class DeviceParameters(db.Model):
    __tablename__ = 'device_parameters'
```

```
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    serial_number = db.Column(db.String(50), nullable=False)
    name = db.Column(db.String(100), nullable=False)
    value = db.Column(db.String(255), nullable=False)

class DeviceConfig(db.Model):
    __tablename__ = 'device_configs'

    id = db.Column(db.Integer, primary_key=True)
    device_id = db.Column(db.Integer, db.ForeignKey('devices.id'), nullable=False,
unique=True)

    push_version = db.Column(db.String)
    error_delay = db.Column(db.Integer)
    request_delay = db.Column(db.Integer)
    trans_times = db.Column(db.String)
    trans_interval = db.Column(db.Integer)
    trans_tables = db.Column(db.String)
    realtime = db.Column(db.Integer)
    timeout_sec = db.Column(db.Integer)

    device = db.relationship("Device", backref="config", uselist=False)
```

```
class DeviceEventLog(db.Model):
    __tablename__ = "device_event_logs"

    id = db.Column(db.Integer, primary_key=True)
    device_id = db.Column(db.Integer, db.ForeignKey("devices.id"), nullable=False)
    timestamp = db.Column(db.DateTime, default=datetime.now(timezone.utc))
    raw_data = db.Column(db.Text)

    device = db.relationship("Device", back_populates="logs")

class DeviceCommand(db.Model):
    __tablename__ = "commands"

    cmd_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    device_sn = db.Column(db.String, nullable=False)  # без ForeignKey
    cmd_desc = db.Column(db.String, nullable=False)
    cmd_detail = db.Column(db.String, nullable=False)
    delivered = db.Column(db.Boolean, default=False)
    delivered_at = db.Column(db.DateTime, default=None)
```

ДОДАТОК Г Server/routes/__init__.py

```
from .cdata import cdata_bp
from .registry import registry_bp
from .getrequest import getrequest_bp
from .push import push_bp
from .realtime import realtime_bp
from .devicecmd import devicecmd_bp
from .ping import ping_bp

def register_routes(app):
    app.register_blueprint(cdata_bp)
    app.register_blueprint(registry_bp)
    app.register_blueprint(getrequest_bp)
    app.register_blueprint(push_bp)
    app.register_blueprint(realtime_bp)
    app.register_blueprint(devicecmd_bp)
    app.register_blueprint(ping_bp)
```

ДОДАТОК Д Server/routes/cdata.py

```
from flask import Blueprint, request, Response
from server.models import db, Device, DeviceEventLog, User
from server.const import const

from datetime import datetime, timezone
import re
import urllib.parse

cdata_bp = Blueprint("cdata", __name__, url_prefix="/iclock")

# Розшифрування подій за кодом
EVENT_CODES = {
    0: "Door opened normally after verification",
    1: "Verify in normal open time period",
    2: "First user opened the door",
    3: "Multiple users opened the door",
    4: "Emergency password used",
    5: "Open the door in the normal open period",
    6: "Trigger linkage",
    7: "Cancel alarm",
    8: "Remote door open",
    9: "Remote door close",
    10: "Disable normal open time of the day",
    14: "Door opened normally after verification",
    15: "Multiple users opened the door",
    16: "Verify in normal open time period",
    17: "Door opened normally after verification",
    18: "First user opened the door",
    19: "First user opened the door",
```

20: "Operation interval too short",

21: "Verify during non-valid time period",

22: "Invalid time period",

23: "Unauthorized access",

24: "APB",

25: "Interlock",

26: "Multiple users waiting for verification",

27: "Card/User not registered",

28: "Door opening timeout",

29: "Privilege expired",

30: "Wrong password",

31: "Operation interval too short",

32: "Multiple users waiting for verification",

33: "Privilege expired",

34: "Fingerprint not registered",

35: "Verify during non-valid time period",

36: "Exit button used during non-valid time",

37: "Door can’t close in normal open period",

38: "Card reported lost",

39: "Blacklist",

40: "Multi-user verification failed",

41: "Incorrect verification mode",

48: "Multi-user verification failed",

49: "Verify during non-valid time period",

50: "Operation interval too short",

51: "Multiple users waiting for verification",

52: "Multi-user verification failed",

53: "Privilege expired",

200: "Вхід дозволено",

201: "Вихід дозволено",

```
    202: "Вхід заборонено",
    203: "Кілька користувачів відкрили двері",
    206: "Двері відкриті вручну",
    214: "Виявлено дублікат",
    215: "Користувача не знайдено",
    217: "Невірний пароль",
    218: "Тривога",
    221: "Після обіду",
    255: "Інше"
}

@cdata_bp.route("/cdata", methods=["GET", "POST"])
def cdata():
    sn = request.args.get("SN")

    if not sn:
        return Response("Missing SN", status=400)

    device = Device.query.filter_by(serial_number=sn).first()
    if not device:
        device = Device(serial_number=sn, registry_code="default", session_id="auto")
        db.session.add(device)
        db.session.commit()

    if request.method == "GET":
        options = request.args.get("options", "")
        print(f"[GET] /iclock/cdata - SN={sn}")

        parameters = urllib.parse.parse_qs(options)
```

```
        if options == "all":
            response_text = (
                f"registry=ok\n"
                f"RegistryCode={device.registry_code}\n"
                f"ServerVersion=3.1.1\n"
                f"ServerName={const.server_name}\n"
                f"PushProtVer=3.1.1\n"
                f"ErrorDelay=30\n"
                f"RequestDelay=2\n"
                f"TransTimes=00:00;23:59\n"
                f"TransInterval=1\n"
                f"TransTables=User Transaction\n"
                f"Realtime=1\n"
                f"SessionID={device.session_id}\n"
                f"TimeoutSec=10\n"
            )

            gmt_date = datetime.now(timezone.utc).strftime('%a, %d %b %Y
%H:%M:%S GMT')
            headers = {
                "Server": {const.server_name},
                "Date": gmt_date,
                "Content-Length": str(len(response_text.encode("utf-8")))
            }

            return Response(response_text, mimetype="text/plain", status=200,
headers=headers)

        return Response("Unknown GET request", status=400)
```

```
else:
    table = request.args.get("table")
    tablename = request.args.get("tablename")
    data = request.data.decode("utf-8", errors="ignore")
    print(f"[POST] /iclock/cdata from {sn}\n{data}")

    if table == "tabledata" and tablename == "user":
        added = 0
        for line in data.splitlines():
            if not line.lower().startswith("user"):
                continue
            parts = dict(field.split("=", 1) for field in line.split("\t") if "=" in field)
            pin = parts.get("pin", "")
            cardno = parts.get("cardno", "")
            name = parts.get("name", "")

            if pin:
                user = User(pin=pin, name=name, card_number=cardno, device_sn=sn)
                db.session.add(user)
                added += 1
        db.session.commit()
        return Response(f"user={added}", status=200)

    # Обробка подій
    for line in data.strip().split("\n"):
        if "event=" in line:
            match = re.search(r"event=(\d+)", line)
            if match:
                code = int(match.group(1))
                desc = EVENT_CODES.get(code, f"Невідома подія ({code})")
```

```
                print(f"[EVENT] {sn}: {desc}")

        # Збереження raw логів
        log = DeviceEventLog(device_id=device.id, raw_data=data,
timestamp=datetime.now(timezone.utc))
        db.session.add(log)
        db.session.commit()

        return Response("OK", status=200)
```

ДОДАТОК Е Server/routes/devicecmd.py

```
from flask import Blueprint, request, Response
from server.models import db, DeviceCommand
from datetime import datetime, timezone
import server.const as const

devicecmd_bp = Blueprint("devicecmd", __name__, url_prefix="/iclock")

@devicecmd_bp.route("/devicecmd", methods=["GET", "POST"])
def devicecmd():
    sn = request.args.get("SN")
    if not sn:
        return Response("Missing SN", status=400)

    if request.method == "GET":
        # Віддаємо першу недоставлену команду
        cmd = DeviceCommand.query.filter_by(device_sn=sn, delivered=False).first()
        if not cmd:
            return Response("", status=200)

        cmd.delivered = True
        db.session.commit()

        response_body = f"ID={cmd.cmd_id}&Return=0&CMD={cmd.cmd_desc}
{cmd.cmd_detail}"
        headers = {
            "Server": const.server_name,
            "Content-Type": "text/plain;charset=UTF-8",
            "Content-Length": str(len(response_body.encode())),
```

```
            "Date": datetime.now(timezone.utc).strftime('%a, %d %b %Y %H:%M:%S
GMT')
        }
        return Response(response_body, status=200, headers=headers)

    elif request.method == "POST":
        body = request.get_data(as_text=True)
        print(f"[POST] /iclock/devicecmd from {sn}:\n{body}")

        # Обробка відповіді від пристрою
        parts = dict(x.split("=", 1) for x in body.strip().split("&") if "=" in x)
        cmd_id = parts.get("ID")
        result_code = parts.get("Return")

        if cmd_id is not None:
            cmd = DeviceCommand.query.filter_by(cmd_id=int(cmd_id)).first()
            if cmd:
                cmd.delivered = True  # вже відмічено при GET, але для надійності
                db.session.commit()
                print(f"[CMD : Secondary] Command ID {cmd_id} виконано, результат:
{result_code}")

        return Response("OK", status=200)
```

ДОДАТОК Є Server/routes/getrequest.py

```
from flask import Blueprint, request, Response
from server.models import db, DeviceCommand
from datetime import datetime, timezone

import server.const as const

getrequest_bp = Blueprint("getrequest", __name__, url_prefix="/iclock")

@getrequest_bp.route("/getrequest", methods=["GET"])
def getrequest():
    sn = request.args.get("SN")
    if not sn:
        return Response("Missing SN", status=400)

    # Отримати всі недоставлені команди для пристрою
    commands = DeviceCommand.query.filter_by(device_sn=sn,
delivered=False).all()

    if not commands:
        return Response("", status=200)

    lines = []
    for cmd in commands:
        lines.append(f"C:{cmd.cmd_id}:{cmd.cmd_desc} {cmd.cmd_detail}\n")
        # Позначити як доставлену
        cmd.delivered = True
        cmd.delivered_at = datetime.now(timezone.utc)
```

```
    db.session.commit()

    body = "\n".join(lines)
    headers = {
        "Server": {const.server_name},
        "Content-Type": "text/plain;charset=UTF-8",
        "Content-Length": str(len(body.encode("utf-8"))),
        "Date": datetime.now(timezone.utc).strftime('%a, %d %b %Y %H:%M:%S
GMT')
    }

    return Response(body, status=200, headers=headers)
```

ДОДАТОК Ж Server/routes/ping.py

```
from flask import Blueprint, request, Response
from datetime import datetime, timezone

import server.const as const

ping_bp = Blueprint("ping", __name__, url_prefix="/iclock")

@ping_bp.route("/ping", methods=["GET"])
def ping():
    sn = request.args.get("SN", "UNKNOWN")
    print(f"[PING] /iclock/ping from {sn}")

    headers = {
        "Server": {const.server_name},
        "Date": datetime.now(timezone.utc).strftime('%a, %d %b %Y %H:%M:%S
GMT'),
        "Content-Length": "0",
    }

    return Response(status=200, headers=headers)
```

ДОДАТОК И Server/routes/push.py

```
from flask import Blueprint, request, Response
from server.models import db, Device, DeviceConfig
from datetime import datetime, timezone

import server.const as const

push_bp = Blueprint("push", __name__, url_prefix="/iclock")

@push_bp.route("/push", methods=["POST"])
def push_config():
    sn = request.args.get("SN")
    if not sn:
        return Response("Missing SN", status=400)

    device = Device.query.filter_by(serial_number=sn).first()
    if not device:
        return Response("Device not registered", status=403)

    # Створення конфігурації, якщо її ще немає
    config = DeviceConfig.query.filter_by(id=device.id).first()
    if not config:
        config = DeviceConfig(
            device_id=device.id,
            push_version="3.1.1",
            error_delay=60,
            request_delay=2,
            trans_times="00:00;23:59",
            trans_interval=1,
```

```
            trans_tables="User Transaction",
            realtime=1,
            timeout_sec=10
        )
        db.session.add(config)
        db.session.commit()
    response_body = (
        f"ServerVersion={const.server_version}\n"
        f"ServerName={const.server_name}\n"
        f"PushVersion={config.push_version}\n"
        f"ErrorDelay={config.error_delay}\n"
        f"RequestDelay={config.request_delay}\n"
        f"TransTimes={config.trans_times}\n"
        f"TransInterval={config.trans_interval}\n"
        f"TransTables={config.trans_tables}\n"
        f"Realtime={config.realtime}\n"
        f"SessionID={device.session_id}\n"
        f"TimeoutSec={config.timeout_sec}\n"
    )
    headers = {
        "Server": const.server_name,
        "Content-Type": "application/push;charset=UTF-8",
        "Content-Length": str(len(response_body.encode("utf-8"))),
        "Date": datetime.now(timezone.utc).strftime('%a, %d %b %Y %H:%M:%S
GMT')
    }

    return Response(response_body, status=200, headers=headers)
```

ДОДАТОК І Server/routes/registry.py

```
from flask import Blueprint, request, make_response
from datetime import datetime, timezone

import random

from server.models import db, Device, DeviceParameters

registry_bp = Blueprint("registry", __name__, url_prefix="/iclock")

def generate_registry_code():
    return ''.join([str(random.randint(0, 9)) for _ in range(10)])

@registry_bp.route("/registry", methods=["POST"])
def registry():
    sn = request.args.get("SN")
    data_string = request.data.decode("utf-8")
    pairs = data_string.split(',')
    device = Device.query.filter(Device.serial_number == sn).first()
    if device is None:
        registry_code = generate_registry_code()
        session_id = ''.join([str(random.randint(0, 9)) for _ in range(32)])
        db.add(Device(
            serial_number=sn,
            registry_code=registry_code,
            session_id=session_id,
            registered=True
        ))
        db.commit()
```

```
    else:
        registry_code = device.registry_code
        session_id = device.session_id
    for pair in pairs:
        key, value = pair.split('=')
        existing_param =
db.session.query(DeviceParameters).filter_by(serial_number=sn, name=key).first()

        if existing_param:
            existing_param.value = value
        else:
            db.session.add(DeviceParameters(
                serial_number=sn,
                name=key,
                value=value
            ))
    db.session.commit()
    # Формуємо відповідь з необхідними заголовками
    response = make_response(f"RegistryCode={registry_code}", 200)
    response.headers["Set-Cookie"] =
f"token=lmao,timestamp={datetime.now(timezone.utc)}"
    response.headers["Server"] = {const.server_name}
    response.headers["Date"] = datetime.now(timezone.utc).strftime('%a, %d %b %Y
%H:%M:%S GMT')
    response.headers["Content-Type"] = "application/push; charset=UTF-8"

    return response
```

ДОДАТОК Ї Server/commands/control.py

```
from server.models import Device, DeviceCommand

from .door import ControlDoor

class Control:
    def __init__(self, database, device_sn):
        self.db = database
        self.device = Device.query.filter_by(serial_number=device_sn).first()

        self.door = self.ControlCommands(self.db, self.device)

    class ControlCommands:
        def __init__(self, db, device):
            self.db = db
            self.device = device

            self.open = ControlDoor(self.db, self.device)
```

ДОДАТОК Й Server/commands/door.py

```
from server.models import DeviceCommand

class ControlDoor:
    def __init__(self, db, device):
        self.db = db
        self.device = device

    def __call__(self, door, time=00,):
        """Відкрити двері"""
        cmd_desc = "CONTROL DEVICE"

        cmd_detail = (f'01{door}01{time}')

        self.db.session.add(DeviceCommand(
            device_sn=self.device.serial_number,
            cmd_desc=cmd_desc,
            cmd_detail=cmd_detail,

        ))

        self.db.session.commit()
        print(f"Команда для пристрою {self.device.serial_number} додана в чергу керування пристроєм")
```

ДОДАТОК К Server/data/data.py

```
from server.models import Device
from .user import UpdateUser, DeleteUser
from .userauthorize import UpdateUserAuthorize, DeleteUserAuthorize
from .mulcarduser import UpdateMulCardUser, DeleteMulCardUser
class Data:
    def __init__(self, database, device_sn):
        self.db = database  # Збереження підключення до бази даних
        self.device = Device.query.filter_by(serial_number=device_sn).first()  # Пошук пристрою
        # Ініціалізація команд для оновлення та видалення
        self.update = self.UpdateCommands(self.db, self.device)
        self.delete = self.DeleteCommands(self.db, self.device)
class UpdateCommands:
        def __init__(self, db, device):
            self.db = db
            self.device = device
            self.user = UpdateUser(self.db, self.device)
            self.userauthorize = UpdateUserAuthorize(self.db, self.device)
            self.mulcarduser = UpdateMulCardUser(self.db, self.device)
class DeleteCommands:
        def __init__(self, db, device):
            self.db = db
            self.device = device
            self.user = DeleteUser(self.db, self.device)
            self.userauthorize = DeleteUserAuthorize(self.db, self.device)
            self.mulcarduser = DeleteMulCardUser(self.db, self.device)
```

ДОДАТОК Л Server/data/mulcarduser.py

```
from server.models import DeviceCommand

class UpdateMulCardUser:
    def __init__(self, db, device):
        self.db = db
        self.device = device

    def __call__(self, card_number=None, pin=None, losscardflag=0, cardtype=0, cardbit=26, sitecode=0):
        """Оновлення даних карти користувача"""
        cmd_desc = "DATA UPDATE mulcarduser"

        cmd_detail = f'Pin={pin}\tCardNo={card_number}\tLossCardFlag={losscardflag}\tCardType={cardtype}\tCardBit={cardbit}\tSiteCode={sitecode}'

        self.db.session.add(DeviceCommand(
            device_sn=self.device.serial_number,
            cmd_desc=cmd_desc,
            cmd_detail=cmd_detail,

        ))
        self.db.session.commit()
        print(f"Команда для пристрою {self.device.serial_number} додана в чергу на оновлення даних карти користувача.")

class DeleteMulCardUser:
    def __init__(self, db, device):
        self.db = db
```

```
        self.device = device

    def __call__(self, pin=None):
        """Видалення даних користувача"""
        cmd_desc = "DATA DELETE mulcarduser"

        cmd_detail = (lambda: f'Pin={pin}' if pin else 'Pin=*')()
        self.db.session.add(DeviceCommand(
            device_sn=self.device.serial_number,
            cmd_desc=cmd_desc,
            cmd_detail=cmd_detail,

        ))
        self.db.session.commit()

        if pin:
            print(f"Команда для пристрою {self.device.serial_number} додана в чергу на видалення даних карти користувача з Pin={pin}.")
        else:
            print(f"Команда для пристрою {self.device.serial_number} додана в чергу на видалення всіх карт користувачів.")
```

ДОДАТОК М Server/data/user.py

```
from server.models import DeviceCommand

class UpdateUser:
    def __init__(self, db, device):
        self.db = db
        self.device = device

    def __call__(self, card_number=None, pin=None, name=None):
        """Оновлення даних користувача"""
        cmd_desc = "DATA UPDATE user"

        cmd_detail =
f'CardNo={card_number}\tPin={pin}\tGroup=1\tStartTime=0\tEndTime=0\tName={
name}\tPrivilege=0\tDisable=0\tVerify=0'

        self.db.session.add(DeviceCommand(
            device_sn=self.device.serial_number,
            cmd_desc=cmd_desc,
            cmd_detail=cmd_detail,

        ))
        self.db.session.commit()
        print(f"Команда для пристрою {self.device.serial_number} додана в чергу на
оновлення даних користувача.")

class DeleteUser:
    def __init__(self, db, device):
        self.db = db
        self.device = device
```

```
    def __call__(self, pin=None):
        """Видалення даних користувача"""
        cmd_desc = "DATA DELETE user"

        cmd_detail = (lambda: f'Pin={pin}' if pin else 'Pin=*')()
        self.db.session.add(DeviceCommand(
            device_sn=self.device.serial_number,
            cmd_desc=cmd_desc,
            cmd_detail=cmd_detail,

        ))
        self.db.session.commit()

        if pin:
            print(f"Команда для пристрою {self.device.serial_number} додана в чергу на видалення даних користувача з Pin={pin}.")
        else:
            print(f"Команда для пристрою {self.device.serial_number} додана в чергу на видалення всіх користувачів.")
```

ДОДАТОК Н Server/data/userauthorize.py

```
from server.models import DeviceCommand

class UpdateUserAuthorize:
    def __init__(self, db, device):
        self.db = db
        self.device = device

    def __call__(self, pin=None, door_id=None):
        """Оновлення авторизації користувача"""
        cmd_desc = "DATA UPDATE userauthorize"

        cmd_detail =
f'Pin={pin}\tAuthorizeTimezoneId=1\tAuthorizeDoorId={door_id}\tDevID=5\tStart
Time=0\tEndTime=0'

        self.db.session.add(DeviceCommand(
            device_sn=self.device.serial_number,
            cmd_desc=cmd_desc,
            cmd_detail=cmd_detail,

        ))
        self.db.session.commit()
        print(f"Команда для пристрою {self.device.serial_number} додана в чергу на
оновлення авторизації користувача.")

class DeleteUserAuthorize:
    def __init__(self, db, device):
        self.db = db
        self.device = device
```

```
    def __call__(self, pin=None):
        """Видалення авторизації користувача"""
        cmd_desc = "DATA DELETE userauthorize"

        cmd_detail = (lambda: f'Pin={pin}' if pin else 'Pin=*')()

        self.db.session.add(DeviceCommand(
            device_sn=self.device.serial_number,
            cmd_desc=cmd_desc,
            cmd_detail=cmd_detail,

        ))
        self.db.session.commit()

        if pin:
            print(f"Команда для пристрою {self.device.serial_number} додана в чергу на видалення авторизації користувача з Pin={pin}.")
        else:
            print(f"Команда для пристрою {self.device.serial_number} додана в чергу на видалення всіх авторизацій користувачів.")
```

ДОДАТОК Н Server/set/set.py

```
from server.models import Device, DeviceCommand
class Set:
    def __init__(self, database, device_sn):
        self.db = database  # Збереження підключення до бази даних
        self.device = Device.query.filter_by(serial_number=device_sn).first()  # Пошук пристрою
        # Ініціалізація команд для оновлення та видалення
        self.options = self.OptionCommands(self.db, self.device)
    class OptionCommands:
        def __init__(self, db, device):
            self.db = db
            self.device = device
        def __call__(self, timezone=None):
            """Оновлення параметру"""
            cmd_desc = "Set OPTIONS"
            cmd_detail = (timezone)
            self.db.session.add(DeviceCommand(
                device_sn=self.device.serial_number,
                cmd_desc=cmd_desc,
                cmd_detail=cmd_detail,
                delivered=False,
                delivered_at=None
            ))
            self.db.session.commit()
            print(f"Команда для пристрою {self.device.serial_number} додана в чергу на оновлення параметру {cmd_detail}")
```

ДОДАТОК О Client/app.py

```
import os
import sys
import time
from datetime import datetime

from flask import Flask, render_template, request, redirect, url_for, flash, jsonify

from server.models import db, Device, DeviceParameters, User, DeviceEventLog
from server.commands.data.data import Data
from server.commands.control.control import Control

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///../server/database.db"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
app.secret_key = 'complete-ui'

db.init_app(app)

@app.route("/")
def index():
    devices = Device.query.all()
    return render_template("index.html", devices=devices)

@app.route("/device/<serial_number>")
def device_detail(serial_number):

    devices = Device.query.all()
```

```
    device = Device.query.filter_by(serial_number=serial_number).first_or_404()
    card_from_log = None
    for log in
reversed(DeviceEventLog.query.filter_by(device_id=device.id).order_by(DeviceEve
ntLog.id.desc()).all()):
        for line in reversed(log.raw_data.splitlines()):
            if "cardno=" in line:
                parts = dict(field.split("=", 1) for field in line.split("\t") if "=" in field)
                cardno = parts.get("cardno", "")
                if cardno and not User.query.filter_by(card_number=cardno).first():
                    card_from_log = cardno
                    break
        if card_from_log:
            break
    return render_template("device.html", device=device, devices=devices,
card_from_log=card_from_log)

@app.route("/device/<serial_number>", methods=["POST"])
def add_user(serial_number):
    name = request.form.get("name")
    pin = request.form.get("pin") or "1"
    card_number = request.form.get("card_number")
    data = Data(database=db, device_sn=serial_number)
    data.update.user(pin=pin, name=name, card_number=card_number)
    data.update.mulcarduser(pin=pin, card_number=card_number)
    door_count = int(DeviceParameters.query.filter_by(serial_number=serial_number,
name="LockCount").first().value or 1)
    for door in range(door_count):
        data.update.userauthorize(pin=pin, door_id=door + 1)
```

```
    flash("Користувача додано!", "success")
    return redirect(url_for("device_detail", serial_number=serial_number))

@app.route("/latest_card/<serial_number>")
def latest_card(serial_number):
    known_cards = {u.card_number for u in
User.query.filter_by(device_sn=serial_number).all()}
    timeout = 5
    interval = 0.5
    start_time = datetime.utcnow()

    device = Device.query.filter_by(serial_number=serial_number).first()

    while (datetime.utcnow() - start_time).total_seconds() < timeout:
        logs =
DeviceEventLog.query.filter_by(device_id=device.id).filter(DeviceEventLog.timesta
mp > start_time).order_by(DeviceEventLog.timestamp.desc()).all()
        for log in logs:
            for line in reversed(log.raw_data.splitlines()):
                if "cardno=" in line:
                    parts = dict(field.split("=", 1) for field in line.split("\t") if "=" in field)
                    cardno = parts.get("cardno", "").strip()
                    if cardno and cardno != "0" and cardno not in known_cards:
                        return jsonify({"card": cardno})
        time.sleep(interval)
    return jsonify({"card": None})

@app.route("/device/<serial_number>/clear")
def clear_device(serial_number):
    data = Data(database=db, device_sn=serial_number)
```

```
    data.delete.user()
    data.delete.userauthorize()
    data.delete.mulcarduser()
    flash("Дані пристрою очищено.", "info")
    return redirect(url_for("device_detail", serial_number=serial_number))

@app.route("/device/<serial_number>/realtime")
def device_realtime(serial_number):
    devices = Device.query.all()
    device = Device.query.filter_by(serial_number=serial_number).first_or_404()

    params = DeviceParameters.query.filter_by(serial_number=serial_number).all()
    parameters = {p.name: p.value for p in params}

    return render_template("realtime.html", device=device, devices=devices)

@app.route("/device/<serial_number>/state")
def device_state(serial_number):
    device = Device.query.filter_by(serial_number=serial_number).first()
    if not device:
        return jsonify({"state": "Пристрій не знайдено"})
    log =
DeviceEventLog.query.filter_by(device_id=device.id).order_by(DeviceEventLog.tim
estamp.desc()).first()
    state = "Зачинено"
    if log:
        for line in reversed(log.raw_data.splitlines()):
            if "relay=" in line:
                parts = dict(field.split("=", 1) for field in line.split("\t") if "=" in field)
                state = parts.get("relay", "Невідомо")
```

```
                break
    return jsonify({"state": state})

@app.route("/device/<serial_number>/open", methods=["POST"])
def device_open(serial_number):
    try:
        ctrl = Control(database=db, device_sn=serial_number)
        ctrl.door.open(door="01", time="05")
        return jsonify({"success": True, "message": "Команду відкриття надіслано"})
    except Exception as e:
        return jsonify({"success": False, "message": str(e)})

@app.route("/device/<serial_number>/parameters")
def device_parameters(serial_number):
    devices = Device.query.all()
    device = Device.query.filter_by(serial_number=serial_number).first_or_404()
    params = DeviceParameters.query.filter_by(serial_number=serial_number).all()
    parameters = {p.name: p.value for p in params}
    return render_template("parameters.html", device=device, devices=devices,
parameters=parameters)

@app.route("/device/<serial_number>/logs")
def device_logs(serial_number):

    devices = Device.query.all()

    device = Device.query.filter_by(serial_number=serial_number).first_or_404()
    logs = []
    all_logs =
DeviceEventLog.query.filter_by(device_id=device.id).order_by(DeviceEventLog.tim
```

```
estamp.desc()).all()
    for log in all_logs:
        for line in log.raw_data.splitlines():
            if "cardno=" in line:
                parts = dict(field.split("=", 1) for field in line.split("\t") if "=" in field)
                logs.append({
                    'cardno': parts.get("cardno", ""),
                    'pin': parts.get("pin", ""),
                    'event': parts.get("event", ""),
                    'time': log.timestamp
                })
    return render_template("logs.html", device=device, devices=devices, logs=logs)

@app.route("/device/<serial_number>/users/delete", methods=["POST"])
def delete_all_users(serial_number):
    try:
        data = Data(database=db, device_sn=serial_number)
        data.delete.user()
        data.delete.userauthorize()
        data.delete.mulcarduser()
        flash("Усіх користувачів пристрою видалено.", "info")
    except Exception as e:
        flash(f"Помилка видалення користувачів: {str(e)}", "danger")
    return redirect(url_for("device_detail", serial_number=serial_number))

@app.route("/device/<serial_number>/users/<int:user_id>/delete",
methods=["POST"])
def delete_user(serial_number, user_id):
    try:
        user = User.query.filter_by(id=user_id,
```

```
device_sn=serial_number).first_or_404()
        data = Data(database=db, device_sn=serial_number)
        data.delete.user(pin=user.pin)
        data.delete.userauthorize(pin=user.pin)
        data.delete.mulcarduser(pin=user.pin)
        db.session.delete(user)
        db.session.commit()
        flash(f"Користувач {user.name} видалений.", "info")
    except Exception as e:
        flash(f"Помилка видалення користувача: {str(e)}", "danger")
    return redirect(url_for("device_detail", serial_number=serial_number))

if __name__ == "__main__":
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

ДОДАТОК П Client/templates/index.html

```
{% extends "layout.html" %}
{% block title %}Список пристроїв{% endblock %}
{% block content %}
 <h1 class="text-2xl font-bold mb-4">Оберіть пристрій для управління</h1>
{% endblock %}
```

ДОДАТОК Р Client/templates/layout.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>{% block title %}Панель управління{% endblock %}</title>
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gray-100 text-gray-900">
  <div class="min-h-screen flex">
    <!-- Sidebar -->
    <div class="w-64 bg-white shadow-md p-4">
      <h2 class="text-xl font-bold mb-6">🔐 Пристрої</h2>
      <ul>
        {% for device in devices %}
          <li class="mb-3">
            <div class="font-semibold">{{ device.serial_number or 'No Name' }}</div>
            <ul class="ml-2 text-sm text-blue-600 space-y-1">
              <li><a href="{{ url_for('device_detail',
serial_number=device.serial_number) }}" class="hover:underline">➕ Додати
користувача</a></li>
              <li><a href="{{ url_for('device_realtime',
serial_number=device.serial_number) }}" class="hover:underline">📡
Realtime</a></li>
              <li><a href="{{ url_for('device_logs', serial_number=device.serial_number)
}}" class="hover:underline">📄 Логи</a></li>
              <li><a href="{{ url_for('device_parameters',
serial_number=device.serial_number) }}" class="hover:underline">⚙️
Параметри</a></li>
```

```
            </ul>
          </li>
        {% endfor %}
      </ul>
    </div>
    <!-- Content -->
    <div class="flex-1 p-6">
      {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
          {% for category, message in messages %}
            <div class="mb-4 p-3 rounded bg-green-100 border border-green-400 text-
green-800">
              {{ message }}
            </div>
          {% endfor %}
        {% endif %}
      {% endwith %}
      {% block content %}{% endblock %}
    </div>
  </div>
</body>
</html>
```

ДОДАТОК С Client/templates/device.html

```
{% extends "layout.html" %}
{% block title %}Пристрій: {{ device.serial_number or 'No Name' }}{% endblock %}
{% block content %}
 <h1 class="text-2xl font-bold mb-4">⌨ {{ device.serial_number or '[No Name]' }}</h1>
 <p class="text-sm text-gray-600 mb-6">SN: {{ device.serial_number }}</p>
 <h2 class="text-xl font-semibold mb-2">Додати користувача</h2>
 <form method="post" class="bg-white p-4 rounded shadow space-y-4">
  <div>
   <label class="block font-medium">Ім’я користувача</label>
   <input name="name" class="w-full p-2 border border-gray-300 rounded" required>
  </div>
  <div>
   <label class="block font-medium">PIN</label>
   <input name="pin" class="w-full p-2 border border-gray-300 rounded" placeholder="за замовчуванням 1">
  </div>
  <div>
   <label class="block font-medium flex justify-between items-center">
    <span>Номер картки</span>
    <button type="button" onclick="getLatestCard()" class="text-sm bg-blue-500 text-white px-2 py-1 rounded hover:bg-blue-600">Зчитати з логу</button>
   </label>
   <input id="card_input" name="card_number" class="w-full p-2 border border-gray-300 rounded" value="{{ card_from_log or '' }}" required>
  </div>
  <button class="bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-
```

```
700">Додати</button>
  </form>
  <div id="toast" class="fixed bottom-4 right-4 hidden px-4 py-2 rounded shadow
text-white"></div>
  <script>
    function showToast(message, isSuccess = true) {
      const toast = document.getElementById("toast");
      toast.innerText = message;
      toast.className = "fixed bottom-4 right-4 px-4 py-2 rounded shadow " +
        (isSuccess ? "bg-green-600" : "bg-red-600") + " text-white";
      toast.classList.remove("hidden");
      setTimeout(() => toast.classList.add("hidden"), 3000);}
    function getLatestCard() {
      fetch("/latest_card/" + encodeURIComponent("{{ device.serial_number }}"))
        .then(response => response.json())
        .then(data => {
          if (data.card) {
            document.getElementById("card_input").value = data.card;
            showToast("Картку зчитано: " + data.card, true);
          } else {
            showToast("Картку не знайдено в логах", false);
          }
        }).catch(err => {
          showToast("Помилка запиту до сервера", false);
        });
    }
  </script>
{% endblock %}
```

ДОДАТОК Т Client/templates/parameters.html

```
{% extends "layout.html" %}
{% block title %}Параметри: {{ device.serial_number or 'No Name' }}{% endblock
%}
{% block content %}
 <h1 class="text-2xl font-bold mb-4">⚙ Параметри пристрою</h1>
 <p class="mb-4 text-gray-600">SN: {{ device.serial_number }}</p>

 <div class="grid grid-cols-1 md:grid-cols-2 gap-4">
  {% for key, value in parameters.items() %}
   <div class="bg-white p-4 rounded shadow">
    <h3 class="font-semibold text-gray-700 truncate">{{ key }}</h3>
    <p class="text-lg break-words">{{ value }}</p>
   </div>
  {% endfor %}
 </div>
{% endblock %}
```

ДОДАТОК У Client/templates/realtime.html

```
{% extends "layout.html" %}
{% block title %}Realtime: {{ device.serial_number or 'No Name' }}{% endblock
%}
{% block content %}
 <h1 class="text-2xl font-bold mb-4">🟢 Realtime моніторинг дверей</h1>
 <p class="mb-4 text-gray-600">SN: {{ device.serial_number }}</p>

 <div class="mb-4 space-x-2">
  <button onclick="openDoor()" class="bg-blue-600 text-white px-4 py-2 rounded
hover:bg-blue-700">Відкрити двері</button>
 </div>

 <div class="bg-white p-4 rounded shadow mb-4 flex items-center space-x-4">
  <h3 class="text-lg font-semibold">Стан дверей:</h3>
  <div id="doorIcon" class="text-3xl">🔒</div>
  <p id="doorState" class="text-xl font-mono text-gray-700"></p>
 </div>

 <div id="realtimeToast" class="fixed bottom-4 right-4 hidden px-4 py-2 rounded
shadow text-white"></div>

 <script>
  function showRealtimeToast(msg, success = true) {
   const el = document.getElementById("realtimeToast");
   el.innerText = msg;
   el.className = "fixed bottom-4 right-4 px-4 py-2 rounded shadow text-white " +
    (success ? "bg-green-600" : "bg-red-600");
   el.classList.remove("hidden");
```

```
  setTimeout(() => el.classList.add("hidden"), 3000);
}

function updateState() {
  fetch("/device/{{ device.serial_number }}/state")
    .then(res => res.json())
    .then(data => {
      const state = data.state || "Невідомо";
      const icon = document.getElementById("doorIcon");
      const text = document.getElementById("doorState");
      text.innerText = state == "000001" ? "Відчинено" : "Зачинено";

      console.log(state)

      if (state == "000001") {
        icon.innerText = " 🔓 ";
      } else {
        icon.innerText = " 🔒 ";
      }
    });
}

function openDoor() {
  fetch("/device/{{ device.serial_number }}/open", { method: "POST" })
    .then(res => res.json())
    .then(data => {
      showRealtimeToast(data.message, data.success);
    });
}
```

```
    setInterval(updateState, 2000);
    updateState();
  </script>
{% endblock %}
```