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 комп'ютерної системи моніторингу та контролю доступу з використанням IoT пристроїв»

Спеціальність F7 «Комп’ютерна інженерія»

Гарант освітньої програми
к.фіз.-мат.н., доц. _________ / Нікітенко Є.В. /
(підпис) (ПІБ)

Керівник дипломного проекту: _________ / Місюра М.Д. /
(підпис) (ПІБ)

Виконав: _________ / Прус О.Б. /
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ 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)
«__» ___________ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ СТУДЕНТУ

Пруса Олександра Борисовича
(прізвище, ім’я, по батькові)

Спеціальність (напрям підготовки): комп’ютерна інженерія

Тема кваліфікаційної бакалаврської роботи: «Розробка комп’ютерної системи моніторингу та контролю доступу з використанням IoT пристроїв»

Термін подання завершеної роботи на кафедру ______________________

Вихідні дані до кваліфікаційної бакалаврської роботи література по використанню біометрії

Перелік питань, що підлягають розробці:

1. Аналіз існуючих систем контролю доступу та моніторингу подій
2. Вибір мікроконтролера та сенсору для розробки системи контролю доступу
3. Розробка програмного забезпечення
4. Розгортання серверної частини в хмарному середовищі

Перелік графічного матеріалу (за потреби) ______________________

Дата видачі завдання “ 16 ” 12 2024 р.

Керівник кваліфікаційної роботи _____________ Місюра М.Д., к.т.н.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання ___________ Прус О.Б.
(підпис) (прізвище та ініціали студента)

затверджено наказом ректора НУБіП У

РЕФЕРАТ

Пояснювальна записка: 86 сторінок, 63 рисунки, 32 лістингів, 21 джерел.

РОЗПІЗНАВАННЯ ВІДБИТКА ПАЛЬЦЯ, СИСТЕМА КОНТРОЛЮ ДОСТУПУ, МІКРОКОНТРОЛЕР, БЕЗПЕКА, АВТЕНТИФІКАЦІЯ, ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ, СЕРВЕР, ХМАРА

Об'єкт дослідження – комп’ютерна система контролю доступу, що поєднує мікроконтролер ESP32, біометричний сенсор відбитків пальців та серверна частина, розгорнута в хмарному середовищі.

Метою роботи є створення надійної, зручної та безпечної системи, яка виконує локальну біометричну ідентифікацію та централізоване логування подій доступу.

У першому розділі обґрунтовано актуальність теми, визначено мету, завдання, об’єкт і предмет дослідження, а також описано структуру роботи.

Другий розділ присвячено аналізу сучасних систем безпеки, використання IoT-технологій у СКУД та формуванню вимог до проєктованої системи.

У третьому розділі розглянуто теоретичні основи побудови систем контролю доступу, сенсорні технології, архітектуру IoT-систем та захищені протоколи обміну даними.

Четвертий розділ містить опис реалізації системи: вибір апаратної бази, розробку програмного забезпечення для ESP32, створення веб-сервера на Flask, тестування та розгортання на платформі Render.

П’ятий розділ узагальнює результати роботи та пропонує напрямки подальшого вдосконалення системи.

Explanatory Note: 86 pages, 63 figures, 32 listings, 21 sources.

FINGERPRINT RECOGNITION, ACCESS CONTROL SYSTEM, MICROCONTROLLER, SECURITY, AUTHENTICATION, INTELLIGENT SYSTEMS, SERVER, CLOUD

The object of this research is a computer-based access control system that integrates an ESP32 microcontroller, a fingerprint biometric sensor, and a cloud-hosted server backend.

The aim of the project is to develop a reliable, user-friendly, and secure system that performs local biometric identification and centralized logging of access events.

The first section outlines the relevance of the topic, defines the aim, objectives, object, and subject of the research, and describes the overall structure of the work.

The second section focuses on the analysis of modern security systems, the use of IoT technologies in access control, and the formulation of requirements for the designed system.

The third section covers the theoretical foundations of access control systems, sensor technologies, IoT system architecture, and secure communication protocols.

The fourth section describes the implementation of the system, including hardware selection, firmware development for the ESP32, Flask-based web server creation, testing, and deployment to the Render platform.

The fifth section summarizes the results of the work and suggests directions for further system improvement.

ЗМІСТ

РЕФЕРАТ 3
ABSTRACT 4
ЗМІСТ 5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 8
РОЗДІЛ 1 10
Вступ 10
1.1. Актуальність теми 10
1.2. Мета та завдання дослідження 11
1.3. Об'єкт і предмет дослідження 11
1.3.1. Об'єкт дослідження 12
1.3.2. Предмет дослідження 12
1.4. Методи дослідження 12
1.5. Структура роботи 14
РОЗДІЛ 2 15
Аналіз предметної області та постановка задачі 15
2.1. Огляд сучасних систем моніторингу та контролю доступу 15
2.2. Використання IoT у системах безпеки та контролю доступу 16
2.3. Аналіз популярних IoT-платформ та технологій 18
2.3.1. Wi-Fi 18
2.3.2. Bluetooth / BLE (Bluetooth Low Energy) 18
2.3.3. RFID 18
2.3.4. NFC 19
2.3.5. IoT-платформи 19
2.4. Визначення основних функціональних та нефункціональних вимог до системи 20
2.4.1. Функціональні вимоги 20
2.4.2. Нефункціональні вимоги 20
РОЗДІЛ 3 22
Теоретична частина 22
3.1. Принципи роботи систем контролю доступу 22
3.1.1. Визначення СКУД 22
3.1.2. Структура СКУД 22
3.1.3. Підходи до організації СКУД 23
3.1.4. Висновки оглянутої теми 23
3.1.5. Вибір підходів для реалізації нашого проекту 23
3.2. Огляд сенсорних та ідентифікаційних технологій 24
3.2.1. Біометричні методи ідентифікації 24
3.2.2. Методи ідентифікації за носієм 24
3.2.3. Методи ідентифікації за знанням 25

3.2.4. Порівняння ідентифікаційних технологій 25
3.2.5. Вибір сенсорних та ідентифікаційних технологій для моєї системи 25
3.3. Архітектура IoT-систем та підходи до обробки даних 26
3.3.1. Рівні типової IoT-системи 26
3.3.2. Моделі IoT-систем за місцем обробки даних 27
3.3.3. Важливі принципи в побудові IoT-системи 27
3.3.4. Опис обраної архітектури 27
3.4. Протоколи безпечної передачі даних в IoT-системах 28
3.4.1. Найпоширеніші протоколи 28
3.4.2. Додаткові механізми захисту 29
3.4.3. Організація аутентифікації та авторизації у системі 29
3.4.4. Вибір протоколів передачі інформації 29
РОЗДІЛ 4 31
Практична реалізація 31
4.1. Вибір апаратної бази 31
4.1.1. Вибір мікроконтролера 31
4.1.2. Вибір сканера FPM10A (AS608) 32
4.2. Проектування архітектури системи 34
4.2.1. Опис архітектури системи 34
4.2.2. Побудова апаратної частини системи 35
4.3. Розробка програмного забезпечення для IoT-пристроїв 39
4.3.1. Підготовка до програмування мікроконтролера 39
4.3.2. Перелік функцій, реалізованих у скетчі відповідно до функціональних вимог 42
4.3.3. Пояснення роботи скетчу 42
4.3.4. Підготовка до розробки веб-сервера 55
4.3.5. Перелік функцій, реалізованих у веб-сервері 57
4.3.6. Написання коду веб-сервера 58
4.3.7. Створення БД 72
4.3.8. HTML-сторінки веб-серверу 73
4.4. Перенесення в хмару та розгортання сервера 76
4.4.1. Перенесення БД 76
4.4.2. Перенесення серверної частини в Render 80
4.5. Тестування роботи системи в реальних умовах 82
4.5.1. Тестування базової функції зчитування пальця 82
4.5.2. Тестування серверної частини та веб-інтерфейсу 85
4.5.3. Тестування веб-інтерфейсу на мобільному пристрої 92
4.6. Аналіз ефективності, продуктивності та безпеки системи 95
4.6.1. Аналіз швидкодії системи 95
4.6.2. Аналіз можливих обмежень системи 96
4.6.3. Аналіз безпеки системи 97

ВИСНОВКИ...99
СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ...100
ДОДАТОК А...104
ДОДАТОК Б...109
ДОДАТОК В...113
ДОДАТОК Г...115
ДОДАТОК Д...119
ДОДАТОК Е...121

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

9

РОЗДІЛ 1

Вступ

1.1. Актуальність теми

У сучасному суспільстві значну увагу приділяють питанням безпеки й автоматизації процесів доступу до приміщень та об'єктів. Із розвитком цифрових технологій зростає потреба в таких системах, які не тільки забезпечують захист від несанкціонованого проникнення, а й надають зручні засоби керування, обліку та моніторингу. Традиційні методи — ключі, магнітні картки, кодові замки — вже не задовольняють вимог ефективності, зручності та надійності. Їм на зміну приходять сучасні системи, побудовані на основі технологій Інтернету речей (IoT) та біометричної ідентифікації.

Інтернет речей (IoT) — концепція, яка передбачає підключення фізичних об’єктів до мережі з можливістю обміну даними [1]. Що важливо, у наш час реалізація IoT-систем не є привілеєм лише великих компаній. Завдяки розвитку апаратної бази та доступності відкритого програмного забезпечення стало можливим створювати повнофункціональні рішення навіть у межах навчального або малого корпоративного середовища. Мікроконтролери з вбудованими модулями зв’язку, сенсори різного типу, хмарні й локальні сервери, мобільні застосунки — усе це вже стало масово доступним і сумісним між собою.

Особливу актуальність має застосування таких технологій саме у сфері контролю доступу. Існує безліч способів реалізації подібних систем: від локальних рішень, де вся логіка зберігається на одному пристрої, до розподілених варіантів з обробкою даних на віддалених серверах або в хмарі. Біометричні методи — як-от розпізнавання обличчя, сканування відбитка пальця або райдужної оболонки ока — значно підвищують рівень безпеки [2], адже прив’язка до унікальних фізіологічних характеристик особи унеможливлює

передачу доступу третім особам.

Такі системи також зручні для адміністрування. В залежності від архітектури, можна реалізувати централізований перегляд логів, дистанційне керування правами доступу, інтеграцію з іншими підсистемами (наприклад, відеоспостереженням, системою сигналізації чи обліку часу.

Усе це свідчить про те, що тема розробки комп'ютерної системи контролю доступу з використанням технологій Інтернету речей та біометричної ідентифікації є сучасною, прикладною та важливою для багатьох сфер — від освіти і побуту до промисловості та корпоративного середовища.

1.2. Мета та завдання дослідження

Метою даної кваліфікаційної роботи є створення програмно-апаратної системи контролю доступу на основі біометричної ідентифікації, яка забезпечує локальну перевірку відбитків пальців, централізоване логування подій та можливість керування правами доступу через серверну частину.

Для досягнення цієї мети необхідно вирішити наступні завдання:

- проаналізувати сучасні підходи до побудови систем контролю доступу з використанням біометричних сенсорів та IoT-пристроїв;
- обґрунтувати вибір архітектури, в якій перевірка відбитків здійснюється на рівні мікроконтролера, а сервер відповідає за логування та централізоване адміністрування доступу;
- реалізувати зчитування відбитків пальців за допомогою; мікроконтролера ESP32 та сенсора FM10A;
- розробити серверну частину на базі веб-фреймворку Flask з базою даних PostgreSQL для обробки логів і керування дозволами;
- розгорнути сервер на хмарній платформі Render
- забезпечити захист каналу передачі даних між мікроконтролером і сервером;
- протестувати систему у різних сценаріях використання, оцінити її стабільність, точність спрацьовувань та зручність адміністрування.

1.3. Об'єкт і предмет дослідження

1.3.1. Об’єкт дослідження

Програмно-технічні засоби комп’ютерної системи контролю та управління доступом (СКУД), реалізованої як система Інтернету речей (IoT) з хмарною архітектурою [3]. До об’єкта дослідження належать механізми централізованого керування доступом, серверна інфраструктура, що забезпечує зберігання, обробку та логування подій доступу, а також засоби взаємодії між фізичними пристроями на кшталт мікроконтролерів та віддаленим сервером, розгорнутим у хмарному середовищі Render.

1.3.2. Предмет дослідження

Хмарні технології та серверні рішення, що використовуються у складі систем Інтернету речей для реалізації централізованого контролю доступу, адміністрування та моніторингу подій. Особлива увага приділяється побудові веб-сервера на базі Python/Flask [4], роботі з базою даних PostgreSQL [5], організації взаємодії з клієнтськими пристроями через HTTP-протоколи [6], а також забезпеченню надійності, безперервності роботи та масштабованості системи у хмарному середовищі.

1.4. Методи дослідження

У ході виконання дослідження було використано комплекс загальнонаукових, інженерних та прикладних методів, що забезпечують цілісне розуміння принципів побудови та функціонування комп’ютерної системи контролю доступу з використанням IoT-технологій.

Аналіз літературних джерел та прикладних рішень у сфері систем контролю доступу дозволив ознайомитися з існуючими підходами до побудови біометричних та мережевих систем і виявити їхні переваги та недоліки. Це дало

змогу обґрунтувати вибір архітектури, в якій основні функції ідентифікації виконуються на рівні мікроконтролера, а централізоване керування і логування реалізуються на сервері.

Методи системного аналізу були застосовані для проектування загальної структури системи, визначення ролей і взаємодії між її компонентами, зокрема ESP32, сенсором відбитків пальців, сервером та базою даних.

Методи програмної інженерії використовувались під час розробки серверної частини системи, яка побудована з використанням веб-фреймворку Flask і реляційної бази даних PostgreSQL. Окрему увагу приділено організації логування подій та реалізації інтерфейсів для керування доступом у хмарному середовищі Render.

Моделювання та експериментальні дослідження застосовувалися для перевірки працездатності системи в умовах, наближених до реального використання. Тестування включало оцінку стабільності зв’язку між пристроями, швидкодії системи, коректності логування подій, а також точності обмеження доступу відповідно до встановлених правил.

Елементи мережевого аналізу та безпеки використовувалися під час оцінки надійності каналу передавання даних між мікроконтролером та сервером, включаючи базові заходи захисту та шифрування запитів.

Таким чином, дослідження поєднує як теоретичний, так і практичний підходи до побудови IoT-системи контролю доступу, приділяючи особливу увагу хмарній інфраструктурі, взаємодії її компонентів та зручності адміністрування.

1.5. Структура роботи

Перший розділ присвячений вступним аспектам дослідження: обґрунтовано актуальність теми, сформульовано мету, завдання, об’єкт, предмет і методи дослідження, а також описано загальну структуру роботи.

У другому розділі подано аналіз предметної області, представлено огляд сучасних систем моніторингу та контролю доступу, розглянуто роль технологій IoT у таких рішеннях, а також визначено основні функціональні й нефункціональні вимоги до проєктованої системи.

Третій розділ містить теоретичну базу дослідження. В ньому розглянуто принципи роботи СКУД, сенсорні технології, архітектуру IoT-систем, протоколи передачі даних та методи аутентифікації користувачів.

У четвертому розділі описано практичну реалізацію системи. Представлено вибір апаратної частини, реалізацію логіки на мікроконтролері ESP32, організацію взаємодії з біометричним сенсором, побудову серверної частини на Flask, розгортання її на платформі Render, а також приклади логування і тестування системи в реальних умовах.

П’ятий розділ містить загальні висновки, підбиття підсумків, оцінку відповідності розробленої системи поставленим цілям і вимогам, а також визначає напрями її подальшого вдосконалення.

Структура роботи забезпечує логічне і послідовне викладення матеріалу, починаючи з теоретичного аналізу й завершуючи практичним застосуванням розробленої системи.

РОЗДІЛ 2

Аналіз предметної області та постановка задачі

2.1. Огляд сучасних систем моніторингу та контролю доступу

Системи контролю доступу (СКУД) є невід'ємною складовою сучасної інфраструктури безпеки. Їхнє основне призначення полягає в забезпеченні авторизованого доступу до приміщень, обліку відвідувань, захисту територій та інформаційних активів. Сучасні СКУД не лише обмежують доступ, але й виконують функції моніторингу, інтеграції з іншими системами безпеки та логування дій користувачів[7].

Традиційно системи цього типу реалізовувалися на основі простих технологій, таких як магнітні або безконтактні карти, PIN-коди, ключі Touch Memory тощо. Ці засоби є досить поширеними завдяки своїй простоті, проте мають низку обмежень — зокрема, можливість передачі ключа іншій особі або його копіювання.

Із розвитком електроніки та мережевих технологій все більшого поширення набувають рішення, що базуються на біометричній ідентифікації: розпізнавання відбитків пальців, обличчя, райдужки ока тощо. Біометричні системи дозволяють підвищити рівень безпеки, адже використовують ознаки, притаманні лише конкретній особі. За прогнозами, ринок електронних СКУД, орієнтованих на такі технології, продовжуватиме стрімко зростати до 2032 року [8].

Паралельно з удосконаленням методів автентифікації активно розвивається й напрям Інтернету речей (IoT), що дозволяє створювати масштабовані, віддалено керовані та хмарно орієнтовані системи [9]. Дослідження підтверджують, що впровадження IoT-рішень у СКУД підвищує їхню ефективність і гнучкість, забезпечуючи безперервний моніторинг,

аналітику подій та глибоку інтеграцію з іншими платформами [10]. Зараз на ринку доступні як готові комерційні рішення, так і відкриті IoT-платформи для розробки індивідуальних СКУД.

Прикладами сучасних комерційних систем є:

– Ajax Systems — український виробник, що пропонує бездротові рішення для охоронної сигналізації та контролю доступу з можливістю централізованого моніторингу через хмару.

– ZKTeco — міжнародний бренд біометричних систем, що використовує розпізнавання облич, відбитків та RFID у пристроях доступу.

– U-Prox — гнучка система управління доступом на основі IP-рішень, з підтримкою хмарного адміністрування.

– Hikvision Access Control — продукти для корпоративного ринку, що інтегрують відеоспостереження, домофонію та біометрію.

Окрім готових рішень, значна частина СКУД розробляється індивідуально — на базі мікроконтролерів, відкритих API та мережевих протоколів, що дозволяє гнучко адаптувати систему до конкретних умов використання. Такі рішення часто використовуються в навчальних закладах, малому бізнесі або для дослідницьких цілей.

У цілому, сучасні СКУД еволюціонують від простих замкнених систем до інтегрованих рішень, що поєднують локальні пристрої, мережеву взаємодію, аналітику подій і зручне хмарне керування, що відповідає концепції Індустрії 4.0 та цифрової трансформації безпеки.

2.2. Використання IoT у системах безпеки та контролю доступу

Технології Інтернету речей (IoT) стали ключовим етапом еволюції в сфері автоматизації та безпеки. Вони дозволяють побудову розподілених, адаптивних та масштабованих систем, де фізичні пристрої взаємодіють між собою, з користувачами та з віддаленим сервером у режимі реального часу. У контексті

систем контролю доступу це означає, що традиційні замкнені рішення можуть бути перетворені на інтелектуальні та динамічні структури з підтримкою гнучкого адміністрування, обліку та інтеграції з іншими сервісами [11].

На відміну від класичних систем, у яких кожен пристрій жорстко підключений до центрального контролера і не має автономних функцій, IoT-рішення надають змогу створювати більш «розумну» логіку доступу. Наприклад, мікроконтролер може локально обробляти частину логіки (ідентифікацію, таймінг, сигнали тривоги), а сервер — виконувати централізоване управління дозволами, логування, зберігання історії подій або аналітику.

IoT-підхід забезпечує також значно кращу масштабованість: нові пристрої можна легко додати до системи без повного її переналаштування. Більше того, такі рішення підтримують віддалений доступ для адміністраторів, що дозволяє змінювати права користувачів, переглядати логи чи навіть оновлювати прошивку пристроїв без фізичного втручання на об’єкті.

У сфері безпеки використання IoT також відкриває нові можливості для інтеграції — наприклад, автоматичне реагування на події (відкриття дверей без авторизації, збої зв’язку), взаємодія з камерами спостереження, системами освітлення чи навіть клімат-контролем. Дані з усіх вузлів можуть передаватися до хмари, де здійснюється централізований аналіз або зберігання архівів.

Надійність IoT-систем значною мірою залежить від правильного вибору протоколів зв’язку, архітектури мережі та безпекових заходів [11] — про це детальніше йтиметься у наступних розділах. Проте вже зараз очевидно, що саме підхід IoT забезпечує гнучкість, адаптивність та розширюваність, які недоступні у класичних централізованих СКУД.

У результаті, використання IoT у системах контролю доступу є не просто черговим етапом модернізації, а логічним переходом до цифрових, автономних і аналітично насичених рішень, що повністю відповідають вимогам сучасного середовища та концепції Індустрії 4.0.

2.3. Аналіз популярних IoT-платформ та технологій

Ефективність побудови системи контролю доступу на основі IoT багато в чому залежить від обраних технологій передачі даних, методів ідентифікації, а також засобів комунікації між пристроями. У цьому підпункті розглянуто найбільш поширені платформи та протоколи, що застосовуються в IoT-рішеннях, з урахуванням специфіки їх використання у сфері безпеки.

2.3.1. Wi-Fi

Wi-Fi є однією з найпоширеніших технологій для з’єднання IoT-пристроїв у локальну мережу або Інтернет. Його перевагами є висока пропускна здатність, універсальність та сумісність із великою кількістю пристроїв і платформ. Для систем контролю доступу Wi-Fi забезпечує можливість передавання великих обсягів даних, оперативного зв’язку із сервером та віддаленого адміністрування. Однак слід враховувати потребу в стабільному покритті мережі та реалізації захищеного з’єднання.

2.3.2. Bluetooth / BLE (Bluetooth Low Energy)

Bluetooth, особливо в енергоощадному варіанті BLE, часто використовується у мобільних застосунках та автономних пристроях. Ця технологія підходить для тимчасового з’єднання з пристроєм або ідентифікації користувача зі смартфона. Хоча радіус дії обмежений (до 10–30 м), BLE чудово підходить для локального контролю, зокрема при використанні мобільного телефону як ключа доступу.

2.3.3. RFID

Радіочастотна ідентифікація (RFID) є одним з найстаріших і найпоширеніших методів безконтактного розпізнавання об’єктів. У системах

контролю доступу активно використовуються RFID-мітки та зчитувачі. Серед переваг — простота реалізації, невисока вартість і швидка ідентифікація. Проте основний недолік — відсутність механізмів шифрування або автентифікації в базових реалізаціях, що обмежує рівень безпеки. альтернативою RFID із вищим рівнем безпеки.

2.3.4. NFC

NFC — це варіація RFID із дуже малим радіусом дії (до 10 см), яка дозволяє реалізувати більш захищену автентифікацію, зокрема за допомогою смартфонів або карт з чіпами. NFC активно використовується в платіжних системах і цифрових ключах, оскільки підтримує шифрування та взаємну автентифікацію пристроїв. У системах контролю доступу може слугувати альтернативою RFID із вищим рівнем безпеки.2.3.2 Bluetooth / BLE (Bluetooth Low Energy).

2.3.5. IoT-платформи

Для розгортання серверної частини IoT-рішень використовуються як відкриті фреймворки (наприклад, Flask, Node-RED, ThingsBoard), так і хмарні платформи — Blynk, Firebase, AWS IoT Core, Azure IoT Hub. Вони забезпечують візуалізацію даних, логіку керування, зберігання журналів, оновлення конфігурацій пристроїв тощо. Для систем контролю доступу, де важливі надійність та контроль над даними, часто використовуються власні або частково автономні рішення із розгортанням на платформах типу Render, Heroku, Vercel.

Вибір технологій залежить від цілей системи. Для безпеки, зручності та гнучкості найчастіше поєднують кілька рішень — наприклад, Wi-Fi для зв’язку, RFID/NFC для ідентифікації та HTTP або MQTT для обміну даними [12].

2.4. Визначення основних функціональних та нефункціональних вимог до системи

2.4.1. Функціональні вимоги

– Система має виконувати локальне зчитування та порівняння відбитків пальців без передавання біометричних даних на сервер.

– При кожній спробі входу система має передавати результат (доступ дозволено/заборонено) та пов'язану інформацію (час, ідентифікатор) на сервер для логування.

– Мікроконтролер має візуально позначати процеси зчитування та успішного порівняння за допомогою вбудованого світлодіода

– Серверна частина має надавати адміністратору інтерфейс для перегляду журналу подій, дистанційного керування доступом до сканера (вмикання/вимикання зчитування) та встановлення режиму сну.

– Сервер має бути доступним через вебінтерфейс із будь-якого пристрою (ПК, смартфон, планшет).

– Передбачена можливість розширення функціоналу в майбутньому (наприклад, додавання нових методів ідентифікації або інтеграція з іншими системами).

2.4.2. Нефункціональні вимоги

– Забезпечення безпеки біометричних даних — відбитки пальців не передаються через мережу і обробляються виключно на пристрої ESP32.

– Зручність користування інтерфейсом — вебсторінка повинна мати адаптивний дизайн і відкриватися коректно як у браузері ПК, так і на мобільних пристроях.

– Можливість адміністрування з будь-якої точки світу через хмарне

розміщення серверної частини.

– Серверна частина має бути розгорнута на надійній хмарній платформі (Render), що забезпечує безперервну роботу, резервування даних і масштабованість.

– Обмін даними між мікроконтролером і сервером повинен здійснюватися захищеним протоколом, із мінімальною затримкою та гарантованою доставкою.

– Архітектура має передбачати простоту адміністрування: швидке оновлення, контроль доступу, ведення журналів, моніторинг активності.

РОЗДІЛ 3
Теоретична частина

3.1. Принципи роботи систем контролю доступу

3.1.1. Визначення СКУД

Системи контролю доступу (СКУД) — це сукупність технічних і програмних засобів, призначених для регулювання прав доступу до фізичних або інформаційних ресурсів. Їхнє головне завдання полягає у тому, щоб надати доступ лише авторизованим користувачам відповідно до заданих політик безпеки. Такі системи є важливою складовою фізичної безпеки об’єктів, а також частиною інформаційної інфраструктури організацій, де необхідне управління правами входу, обмеженням зон доступу та фіксацією подій [12].

3.1.2. Структура СКУД

СКУД, як правило, складаються з трьох основних підсистем [12]:

– Підсистема ідентифікації користувача, яка виконує розпізнавання особи на основі одного або декількох факторів: картка, PIN-код, біометричний параметр, смартфон тощо.

– Підсистема прийняття рішень, яка перевіряє, чи має ідентифікований користувач дозвіл на доступ у певний час, до певного ресурсу чи зони.

– Виконавча підсистема, що відкриває або блокує доступ — наприклад, через електрозамок, реле, сигнал чи інший пристрій управління.

Додатково, сучасні СКУД часто включають систему моніторингу та журналювання подій, яка фіксує всі спроби доступу, незалежно від їх результату. Це дозволяє здійснювати аудит, аналітику та розслідування інцидентів.

3.1.3. Підходи до організації СКУД

Існує кілька підходів до організації логіки прийняття рішень [12]:

– Автономні системи — рішення приймається локально, на пристрої, без підключення до центрального сервера.

– Централізовані системи — всі дані передаються на сервер, де й відбувається перевірка та прийняття рішення.

– Гібридні системи — поєднують автономну роботу з можливістю централізованого адміністрування.

3.1.4. Висновки оглянутої теми

СКУД можуть мати різні рівні доступу, зонування, розклад дій (наприклад, доступ лише у робочі години), інтеграцію з іншими системами — відеоспостереженням, сигналізацією, контролем робочого часу тощо. Усі ці функції реалізуються через налаштування логіки, правил і прав доступу.

Сучасні підходи до побудови СКУД все частіше включають елементи Інтернету речей (IoT), що дозволяє забезпечити віддалене адміністрування, хмарне зберігання логів, масштабованість і інтеграцію з іншими цифровими сервісами.

Таким чином, основними принципами роботи систем контролю доступу є [12]: надійна ідентифікація особи, прийняття рішення на основі визначених правил, фізичне виконання команди доступу та облік усіх подій, пов'язаних із цією взаємодією.

3.1.5. Вибір підходів для реалізації нашого проекту

У межах проєкту реалізується система, що базується на логіці автономного прийняття рішень про доступ із наступною фіксацією подій на сервері. Такий підхід дозволяє досягти високої швидкодії та стабільності, що є критичним у системах контролю доступу.

3.2. Огляд сенсорних та ідентифікаційних технологій

У системах контролю доступу ключову роль відіграє спосіб розпізнавання користувача. Саме на основі цього етапу приймається рішення про дозвіл або заборону доступу. Від точності, стійкості до підробки та зручності обраного методу ідентифікації залежить загальний рівень безпеки всієї системи.

У цьому підпункті розглядаються класи методів ідентифікації, що застосовуються у СКУД, з фокусом на їх сенсорну та логічну природу — що саме розпізнає система, а не як вона передає ці дані.

3.2.1. Біометричні методи ідентифікації

Базуються на унікальних фізіологічних або поведінкових характеристиках людини:

- Відбитки пальців
- Розпізнавання обличчя
- Ідентифікація за райдужкою або сітківкою ока

Біометрія вважається однією з найбезпечніших технологій [13], адже прив'язана до конкретної особи. Водночас вимагає точного обладнання та контролю умов середовища.

3.2.2. Методи ідентифікації за носієм

Базуються на використанні фізичних предметів, які належать користувачу і є носіями унікальної інформації для допуску:

- Магнітні картки
- RFID/NFC-мітки
- Смарткартки

Такі засоби зручні та прості у використанні, однак підлягають копіюванню або передачі, що знижує рівень захисту.

3.2.3. Методи ідентифікації за знанням

Передбачають введення користувачем заздалегідь відомого коду або інформації, яку потрібно пам’ятати:

- Магнітні картки
- RFID/NFC-мітки
- Смарткартки

Такі засоби зручні та прості у використанні, однак підлягають копіюванню або передачі, що знижує рівень захисту.

3.2.4. Порівняння ідентифікаційних технологій

Порівняльні характеристики основних ідентифікаційних технологій наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння ідентифікаційних технологій у СКУД

Технологія	Рівень безпеки	Вартість реалізації	Зручність використання	Складність впровадження
RFID	Середній	Низька	Висока	Низька
NFC	Середній/високий	Середня	Висока	Середня
PIN-коди	Низький	Низька	Середня	Низька
QR-коди	Середній	Низька	Середня	Низька
Біометрія	Високий	Середня/висока	Висока	Середня

3.2.5. Вибір сенсорних та ідентифікаційних технологій для моєї системи

З огляду на потребу у високому рівні безпеки та унікальності користувача, в розробленій системі було обрано біометричний метод ідентифікації — зчитування відбитків пальців. Цей підхід не потребує носіїв і забезпечує надійність навіть за відсутності підключення до мережі.

3.3. Архітектура IoT-систем та підходи до обробки даних

Архітектура систем Інтернету речей (IoT) передбачає побудову багаторівневої структури, яка поєднує фізичні пристрої, мережеву інфраструктуру, програмну логіку, бази даних та користувацькі інтерфейси. У системах контролю доступу така архітектура повинна не лише забезпечувати обробку даних, а й реагувати на події в реальному часі, забезпечувати збереження історії, інтеграцію з іншими сервісами та віддалене керування.

3.3.1. Рівні типової IoT-системи

Архітектура систем Інтернету речей (IoT) зазвичай організована у вигляді багаторівневої структури, що включає:

– Пристрої збору даних (Edge Level) — сенсори, мікроконтролери, сканери, які безпосередньо взаємодіють з фізичним середовищем і формують первинну інформацію (наприклад, результат зчитування).

– Рівень обробки та логіки (Fog або Gateway Level) — пристрої або сервіси, які фільтрують, агрегують або попередньо обробляють дані перед передачею далі. У системах контролю доступу це може бути логіка прийняття рішення про надання доступу.

– Хмарний рівень (Cloud Level) — сервери або сервіси, які зберігають дані, забезпечують централізовану логіку, аналітику, адміністрування, доступ до інтерфейсів та API.

Такий поділ на рівні відповідає загальноприйнятій моделі обробки даних у IoT-системах, що дозволяє досягати гнучкості, відмовостійкості та ефективності при обробці подій у реальному часі. Кожен рівень виконує специфічну роль у маршрутизації, обробці та зберіганні інформації, і саме поєднання цих компонентів формує основу сучасної архітектури Інтернету речей [14].

3.3.2. Моделі IoT-систем за місцем обробки даних

– Локальна обробка (edge computing) — дані обробляються безпосередньо на пристрої, що забезпечує мінімальну затримку, автономність і підвищену безпеку.

– Централізована обробка (cloud computing) — усі дані передаються на сервер, де виконується перевірка, обробка, логіка та збереження.

– Гібридна модель — поєднує обробку даних на пристрої з передачею результатів на сервер для зберігання або адміністрування [14].

3.3.3. Важливі принципи в побудові IoT-системи

– Відмовостійкість: система має працювати навіть у разі тимчасової втрати зв'язку.

– Масштабованість: можливість додати нові пристрої без перебудови всієї системи.

– Мінімізація затримки: особливо важливо в задачах реального часу, таких як доступ.

– Безпека на кожному рівні: і при збиранні, і при передаванні, і при зберіганні даних.

– Оптимальна організація інформаційних потоків: дані повинні рухатися чітко визначеним шляхом — від пристрою до логіки, потім до збереження та відображення в інтерфейсі, без зайвих затримок або дублювання.

3.3.4. Опис обраної архітектури

Для забезпечення автономності й надійного функціонування в умовах нестабільного зв'язку було обрано гібридну архітектуру: обробка біометричних даних виконується локально на пристрої, а серверна частина відповідає за логування подій та керування доступом. Це дозволяє поєднати швидкість

локальної реакції з гнучкістю централізованого адміністрування.

3.4. Протоколи безпечної передачі даних в IoT-системах

У системах Інтернету речей передавання даних між пристроями, шлюзами та серверами відіграє ключову роль. Саме на цьому етапі дані найвразливіші до перехоплення, підміни чи несанкціонованого доступу. Тому забезпечення захищеної передачі інформації є обов'язковою вимогою до будь-якої IoT-архітектури, особливо у сферах, пов'язаних із контролем доступу та обробкою персональних або біометричних даних.

Основні ризики, які виникають під час передачі даних в IoT-середовищі: перехоплення незашифрованих повідомлень (sniffing), підміна даних (man-in-the-middle), несанкціоноване втручання в канал зв'язку, повторна передача старих даних (replay-атаки).

Для захисту каналів зв'язку використовуються спеціалізовані протоколи безпечної передачі, які забезпечують шифрування, автентифікацію та цілісність даних.

3.4.1. Найпоширеніші протоколи

Розберемо найпоширеніші протоколи безпечної передачі даних:

– HTTPS (HyperText Transfer Protocol Secure) — розширення HTTP із додатковим шифруванням за допомогою SSL/TLS. Забезпечує захист трафіку навіть у публічних мережах.

– TLS (Transport Layer Security) — криптографічний протокол для захищеного з'єднання між двома вузлами. Може застосовуватись у складі інших протоколів.

– MQTT over TLS — версія легкого брокерного протоколу MQTT з шифруванням трафіку. Застосовується для систем з великою кількістю пристроїв у режимі реального часу.

– DTLS (Datagram Transport Layer Security) — адаптований варіант TLS для протоколу UDP, з низькою затримкою, що актуально для часу-критичних застосунків.

Розглянуті протоколи широко використовуються для забезпечення безпечної передачі даних в IoT-системах. Їхні характеристики та застосування детально аналізуються в науковій літературі [15].

3.4.2. Додаткові механізми захисту

Також використовуються додаткові механізми захисту передачі:

– Гешування (SHA-256, SHA-3) — для перевірки цілісності даних;
– Цифрові підписи — для підтвердження джерела інформації;
– JWT (JSON Web Tokens) — для авторизованої передачі даних у веб-запитах.

Застосування вказаних технологій дозволяє ефективно захистити IoT-систему від основних загроз і забезпечити дотримання вимог до конфіденційності, цілісності та автентичності інформації.

3.4.3. Організація аутентифікації та авторизації у системі

У розробленій системі передбачено поділ рівнів доступу між користувачем та адміністратором. Аутентифікація користувача відбувається на основі біометрії, тоді як адміністратор проходить авторизацію через вебінтерфейс. Такий підхід дозволяє ефективно розмежовувати права та забезпечувати централізоване управління доступом.

3.4.4. Вибір протоколів передачі інформації

При налаштуванні обміну між пристроєм і сервером я одразу враховував питання безпеки. Зараз дані передаються не просто по HTTP, а через HTTPS — це означає, що весь трафік шифрується за допомогою TLS, тож його не так

просто перехопити чи змінити. Така схема не потребує якихось складних налаштувань, але дає базовий рівень захисту. Якщо в майбутньому виникне потреба перейти на інший протокол чи змінити формат взаємодії — вся логіка залишиться придатною без кардинальної переробки.

РОЗДІЛ 4

Практична реалізація

4.1. Вибір апаратної бази

Під час проєктування комп’ютерної системи контролю доступу було прийнято рішення використовувати мікроконтролер ESP32-WROOM-32 у якості центрального обчислювального елемента, а також оптичний сканер відбитків пальців FPM10A (на базі процесора Synochip AS608) як засіб біометричної ідентифікації.

4.1.1. Вибір мікроконтролера

ESP32-WROOM-32 (рис. 4.1) — це потужний мікроконтролер з вбудованими модулями Wi-Fi та Bluetooth, розроблений компанією Espressif [16]. Він має двоядерний 32-бітний процесор, що працює на частоті до 240 МГц, а також підтримує множину периферійних інтерфейсів (UART, I2C, SPI, PWM тощо). Більш детальні технічні характеристики мікроконтролера наведено в Таблиці 4.1

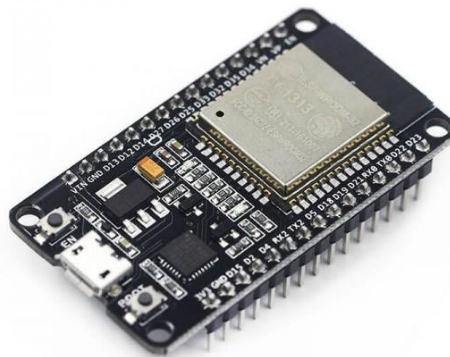


Рис. 4.1 – ESP32-WROOM-32

Таблиця 4.1 – Технічні характеристики мікроконтролера ESP32-WROOM-32 [16]

Параметр	Значення
Тип процесора	Tensilica Xtensa LX6, 32-біт, до 240 МГц
Кількість ядер	2 (двоядерний)
Оперативна пам’ять (SRAM)	520 КБ
Пам’ять Flash	від 4 МБ (залежить від модуля)
Інтерфейси	UART, SPI, I2C, PWM, ADC, DAC, GPIO
Кількість GPIO	до 34
Зв’язок	Wi-Fi 802.11 b/g/n, Bluetooth 4.2 (BLE + BR/EDR)
Живлення	3.3 В
Підтримка режимів енергозбереження	Так (Light Sleep, Deep Sleep)

Основні переваги ESP32-WROOM-32 для нашої системи:

- Вбудований Wi-Fi
- Висока продуктивність.
- Гнучкість інтерфейсів.
- Низьке енергоспоживання.
- Широка підтримка спільноти.
- Розумна ціна.

У порівнянні з Arduino Uno чи Nano, ESP32 має значно вищу продуктивність, більше пам’яті та вбудовані бездротові модулі, що дозволяє обійтись без додаткових плат. Raspberry Pi хоч і потужніший, проте вимагає складнішої ОС, споживає більше енергії й коштує дорожче. На цьому фоні ESP32-WROOM-32 вирізняється оптимальним балансом продуктивності, енергоефективності, простоти інтеграції та доступної ціни.

4.1.2. Вибір сканера FPM10A (AS608)

Для реалізації біометричної ідентифікації в системі контролю доступу

було обрано оптичний сканер відбитків пальців FPM10A (рис. 4.2) , побудований на основі процесора Synochip AS608. Пристрій забезпечує повний цикл роботи з відбитками — від зчитування до зберігання та порівняння — без потреби в зовнішньому обчислювальному модулі[17].

Детальні технічні характеристики наведені в Таблиці 4.2

Рис. 4.2 – FPM10A

Таблиця 4.2 – Технічні характеристики біометричного сенсора FPM10A (AS608) [17]

Параметр	Значення
Тип сенсора	Оптичний
Розмір зони сканування	≈ 15 × 18 мм
Процесор	Synochip AS608 (ARM Cortex-M 32-bit)
Ємність шаблонів	до 1000 відбитків
Функціонал	Зчитування, генерація, збереження, пошук, видалення
Інтерфейс підключення	UART (57600 біт/с)
Час порівняння	≈ 1 секунда
Живлення	3.3–6 В
Габарити	Компактний корпус для вбудованих рішень

Основні переваги FPM10A у порівнянні з аналогами:

– Автономність роботи
– Висока точність
– Сумісність з мікроконтроллером
– Наявність бібліотек і прикладів
– Баланс ціна/функціональність

У порівнянні з альтернативами, такими як GT511C3 (який потребує більше ресурсів для обробки) або R503 (дорожчий за аналогічну точність), FPM10A/AS608 є оптимальним вибором для проєктів середнього рівня складності, де важливі надійність, компактність та простота інтеграції.

4.2. Проектування архітектури системи

4.2.1. Опис архітектури системи

Архітектура розробленої системи контролю доступу базується на гібридному підході, що поєднує локальну обробку біометричних даних та централізоване логування та адміністрування через хмарний сервер. Такий підхід дозволяє досягти балансу між швидкодією, автономністю і можливістю віддаленого керування.

Основна ідея системи полягає в наступному: мікроконтролер із підключеним біометричним сенсором здійснює ідентифікацію користувача на місці, без потреби надсилання відбитків пальців або шаблонів на сервер. Після перевірки, незалежно від результату, система формує подію (успішна/невдала спроба доступу), яка надсилається на сервер для збереження в базі даних.

Крім того, система передбачає можливість керування доступом до сканера віддалено — наприклад, тимчасово його деактивувати або увімкнути певний режим роботи. Вся логіка адміністрування, перегляд журналу подій, встановлення режиму сну реалізовані на стороні сервера.

Серверна частина розгорнута у хмарному середовищі Render, що забезпечує безперервну доступність з будь-якого пристрою, незалежно від

фізичного розташування користувача. База даних, у якій зберігаються всі події, реалізована на платформі PostgreSQL, що дозволяє забезпечити надійне та масштабоване зберігання інформації.

Обрана архітектура дозволяє пристрою працювати автономно у випадку втрати зв'язку з сервером, при цьому всі критичні функції (ідентифікація, дозвіл/заборона доступу) виконуються локально. Як тільки зв'язок буде відновлено — події автоматично передаються на сервер.

Візуальне представлення архітектури системи (рис. 4.3):

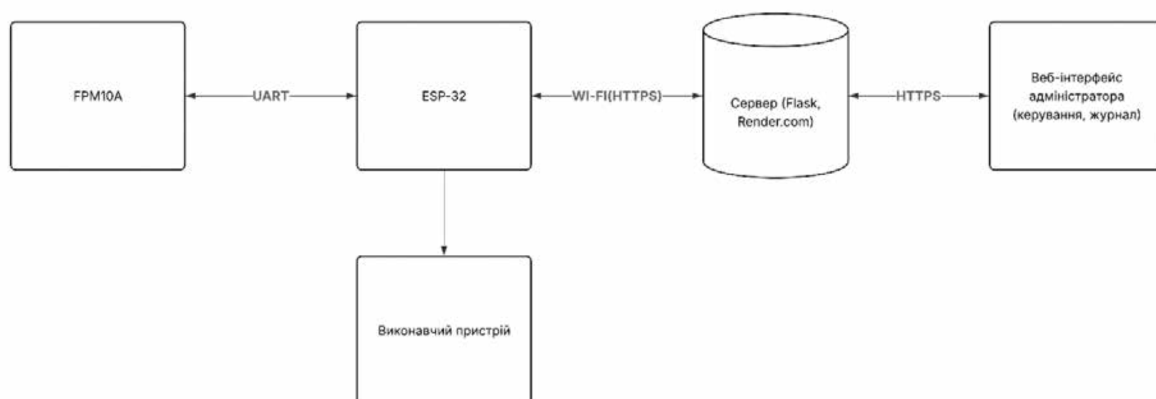


Рис. 4.3 – Структурна схема взаємодії компонентів IoT-системи контролю доступу

4.2.2. Побудова апаратної частини системи

При побудові нашої апаратної частини потрібно було лише під'єднати сенсор відбитків пальця до мікроконтролера. Для цього була використана макетна плата та комплектний з'єднувальний шлейф, що довелося доповнити однопіновими провідниками, припаяними вручну, для можливості з'єднання з вище згаданою макетною платою.

При з'єднанні з мікроконтролером необхідно було правильно під'єднати роз'єми сенсора (рис. 4.4) з роз'ємами ESP32 (рис. 4.5) .

Роз'єми з'єднання FPM10A:

– RX – приймач даних
– TX – передавач даних
– GND – заземлення
– 3v3 – живлення

Рис. 4.4 – Позначки роз'ємів в документації від дестрибютора сенсора [17]

Рис. 4.5 – Схема підключення в документації від дестрибютора сенсора [17]

Спочатку для перевірки правильного під’єднання та працездатності сенсору було завантаж Arduino IDE, в котрій і буде написано та скомпільовано скетч для нашого мікроконтролера, та встановив головну бібліотеку для роботи з сканера та опрацювання відбитків Adafruit_Fingerprin.h [17] . Для тестування роботи відбитка було запущено скетч для тестування, що лежить в папці прикладів цієї бібліотеки, а саме enroll, що виконує зчитування та збереження шаблонів відбитків на сенсорі. Проте цей скетч хоча і скомпілювався, працював не вірно (рис. 4.6), адже не зміг розпізнати сенсор, що вказало на неправильне підключення пінів.

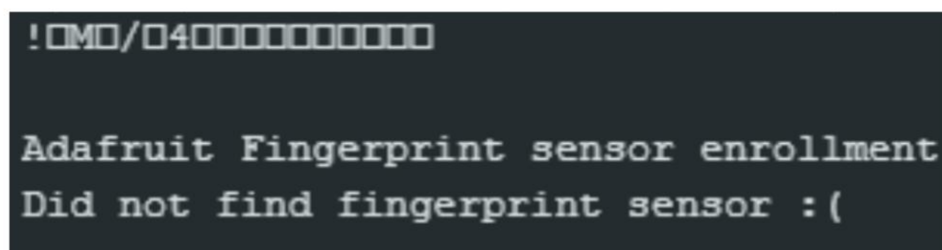


Рис. 4.6 – Повідомлення, що вказує на те, що сенсор не розпізнано

Згідно з результатами аналізу джерел[18], було виявлено, що хоча в більшості теоретичних матеріалів з підключення сканера рекомендується використовувати стандартні пінові роз’єми RX та TX мікроконтролера (перехресно), на практиці таке підключення призводить до конфлікту. Це пов’язано з тим, що GPIO1 (TX) та GPIO3 (RX) на ESP32 використовуються для USB-програмування та роботи Serial Monitor, і одночасне використання їх для зчитувача відбитків спричиняє порушення в роботі обох інтерфейсів.

Щоб уникнути цієї проблеми, було прийнято рішення використати UART1, другий апаратний послідовний інтерфейс ESP32, який дозволяє вільно перенаправляти сигнали на інші GPIO. Зокрема, було обрано піни: GPIO16 (RXD2) та GPIO16 (RXD2), до котрих сенсор був перепід’єднано (рис. 4.7) .

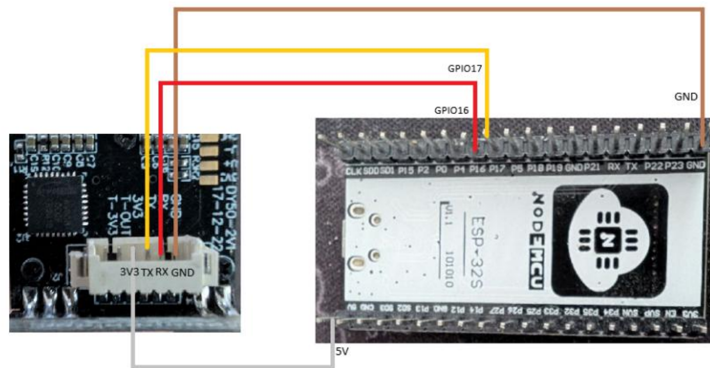


Рис. 4.7 – Візуальне представлення правильного варіанту підключення

Також програмно було встановлено зв’язок через апаратний UART1.

Лістинг 4.1 – Створення об’єктів для зв’язку з сенсором через відповідні піни

```
HardwareSerial mySerial(1);
Adafruit_Fingerprint finger(&mySerial);
```

Після цих змін в нашому макеті знову перевіряємо правильність роботи тестового скетчу(рис. 4.8).

```
Adafruit Fingerprint sensor enrollment
Found fingerprint sensor!
Reading sensor parameters
Status: 0x0
Sys ID: 0x0
Capacity: 150
Security level: 3
Device address: FFFFFFFF
Packet len: 128
Baud rate: 57600
Ready to enroll a fingerprint!
Please type in the ID # (from 1 to 127) you want to save this finger as...
```

Рис. 4.8 – Правильний вигляд виводу повідомлення при успішному підключенні сенсора

Таким чином сенсор та контролер (рис. 4.9) правильно під’єднанні та

готові для подальшої роботи.

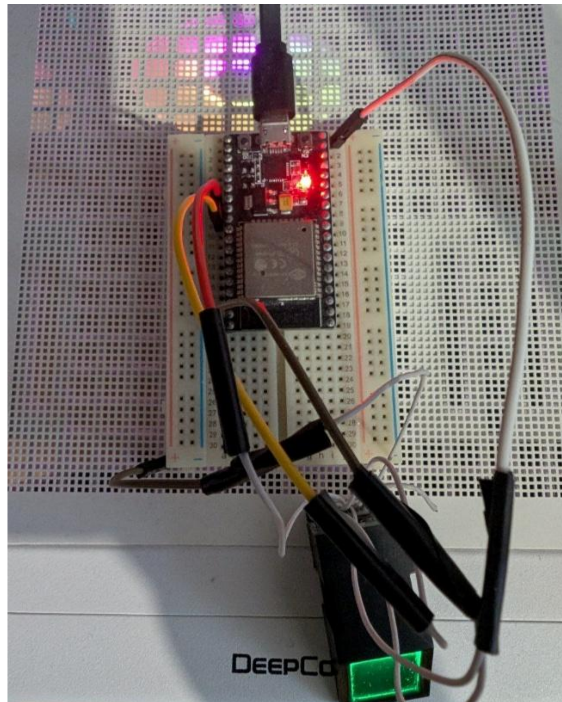


Рис. 4.9 – Готова апаратна частина системи

4.3. Розробка програмного забезпечення для IoT-пристроїв

4.3.1. Підготовка до програмування мікроконтролера

Як уже було вказано вище, для програмування нашого мікроконтролера було використано Arduino IDE (рис. 4.10) . Це середовище дає змогу зручно працювати з різними пристроями на базі платформи Arduino, використовуючи готові бібліотеки для необхідного обладнання.

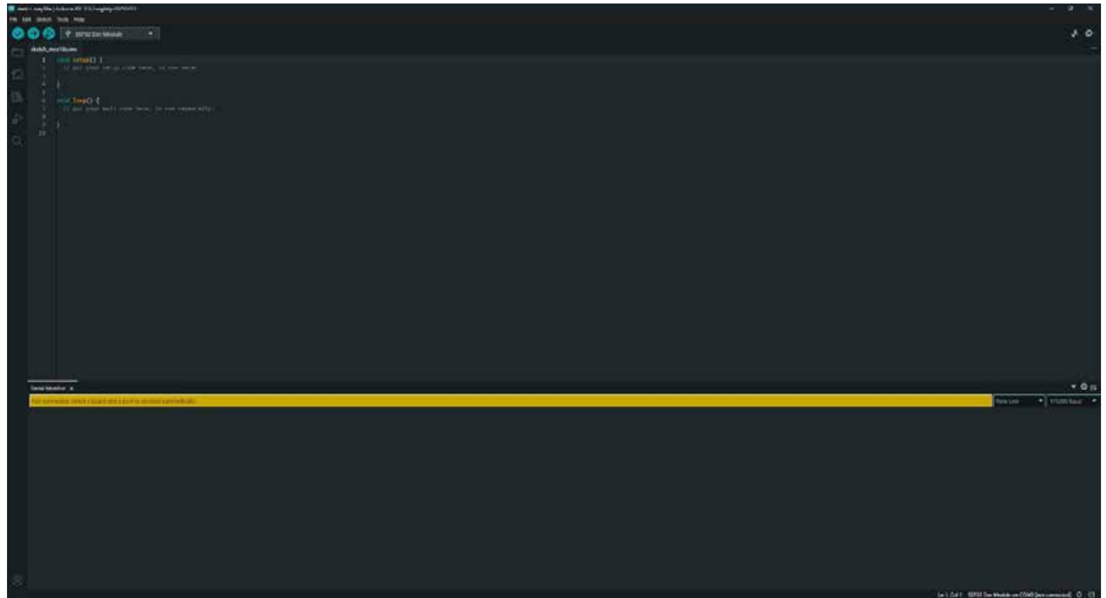


Рис. 4.10 – Інтерфейс Arduino IDE

Для роботи з ESP32 через менеджер плат було встановлено відповідний сторонній пакет від виробника плати esp32 by Espressif Systems .

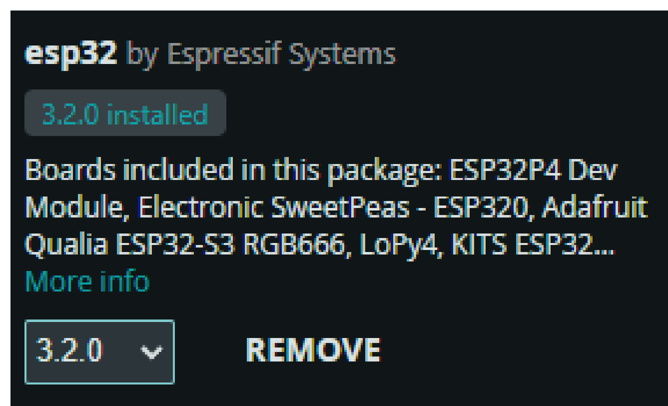


Рис. 4.11 – Встановлений пакет плат для ESP32

Також, для підключення сенсора відбитків пальців FPM10A, було завантажено додаткову, вже згадану, бібліотеку Adafruit_Fingerprint (рис. 4.12) .

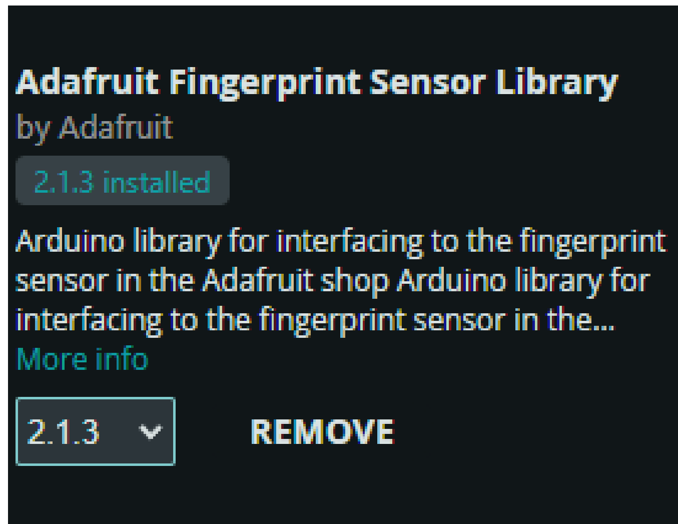


Рис. 4.12 – Встановлена бібліотека для роботи сенсора [17]

Для реалізації обміну даними через інтернет було використано вбудовані бібліотеки WiFi.h та HTTPClient.h, які використовуються для підключення до Wi-Fi і надсилання HTTP-запитів, їх додатково завантажувати не потрібно.

Перед початком написання коду було обрано відповідну плату та COM-порт (рис. 4.13) , необхідні для завантаження скетчів на мікроконтролер.

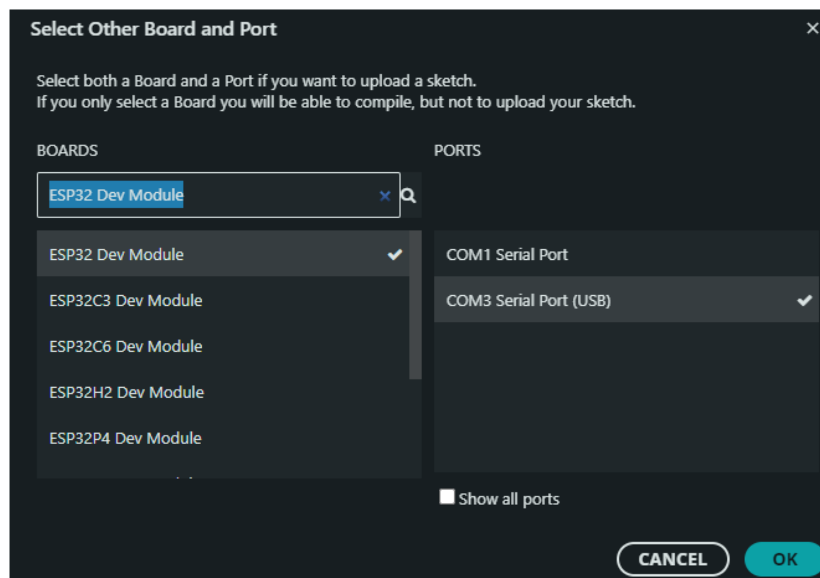


Рис. 4.13 – Обрані плата та серійний порт для завантаження скетчу

Тепер середовище готове для реалізації нашого проекту (додаток А) .

4.3.2. Перелік функцій, реалізованих у скетчі відповідно до функціональних вимог

– Встановлення Wi-Fi з’єднання — підключення мікроконтролера до заданої мережі для подальшої взаємодії з сервером.

– Перевірка статусу сканера — запит до серверного маршруту /scanner_status, який визначає, чи дозволено в поточний момент виконувати зчитування.

– Зчитування відбитка пальця — ініціалізація сканера та зчитування біометричних даних у разі дозволу з боку сервера.

– Порівняння збережених шаблонів — локальний пошук збігу з раніше збереженими відбитками за допомогою функції fingerSearch().

– Надсилання результату на сервер — відправлення ID користувача (або 0 при невдачі) на сервер через захищений POST-запит до маршруту /verify.

– Обробка команд із серійного монітора — розпізнавання команд користувача, наприклад r5, для запуску реєстрації нового шаблону відбитка.

– Реєстрація нового користувача — двоетапне зчитування пальця, створення шаблону та збереження його у пам’яті сенсора з подальшим переданням імені користувача на сервер.

– Світлодіодна індикація — візуальні сигнали стану системи: очікування, успішне зчитування або підтвердження подій.

– Обробка переривань під час сканування — можливість достроково припинити процес зчитування або реагувати на нові команди, навіть якщо сканування вже розпочалось.

4.3.3. Пояснення роботи скетчу

На початку імпортуємо необхідні бібліотеки.

Лістинг 4.2 – Імпорт необхідних бібліотек

```
#include <WiFi.h> // Бібліотека для підключення ESP32 до Wi-Fi-мережі
#include <HTTPClient.h> // Дозволяє здійснювати HTTP(S)-запити до серверу
#include <Adafruit_Fingerprint.h> // Бібліотека для роботи із сенсором відбитків пальців
#include <HardwareSerial.h> // Дозволяє використовувати апаратні UART-порти ESP32
```

На цьому етапі визначаємо пінні підключення, які будемо використовувати у скетчі.

Лістинг 4.3 – Оголошення пінів, що використовуються у програмі

```
#define RXD2 16 // Пін ESP32, до якого підключено TX сенсора (використовується як RX для UART1)
#define TXD2 17 // Пін ESP32, що відповідає за передачу даних до RX сенсора (TX для UART1)
#define LED_BUILTIN 2 // Вбудований світлодіод на платі ESP32 (GPIO2), використовується для індикації стану
```

На цьому етапі оголошуємо об’єкти та глобальні змінні, які будемо використовувати у скетчі.

Лістинг 4.4 – Оголошення об’єктів та глобальних змінних

```
HardwareSerial mySerial(1); // Створення об'єкта для апаратного UART1
Adafruit_Fingerprint finger(&mySerial); // Створення об'єкта для роботи з сенсором FPM10A через UART1

// Wi-Fi дані
const char* ssid = "****"; // Назва мережі
const char* password = "****"; // Пароль до мережі

// URL-и сервера
const char* verifyUrl = "https://fingerprint-access-server.onrender.com/verify"; // Адреса для перевірки ID
const char* enrollUrl = "https://fingerprint-access-server.onrender.com/enroll"; // Адреса для реєстрації користувача
const char* statusUrl = "https://fingerprint-access-
```

```
server.onrender.com/scanner_status"; // Адреса для перевірки
статусу сканера

     bool isVerifying = true; // Активність режиму ідентифікації
(true — сканування ввімкнене)
```

У наведеному лістингу значення SSID та пароля замінені на умовні позначення з міркувань безпеки.

В функції setup() виконуємо ініціалізацію компонентів.

Лістинг 4.5 – Інаціалізація компонентів у фінкції setup()

```
void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
  delay(1000);

  Serial.println("Connecting to WiFi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");

  mySerial.begin(57600, SERIAL_8N1, RXD2, TXD2);
  finger.begin(57600); // Ініціалізація UART-зв'язку з
сенсором через UART1

  if (finger.verifyPassword()) {
    Serial.println("Fingerprint sensor ready.");
  } else {
    Serial.println("Sensor not detected.");
    while (true) delay(1); // зупинка програми, якщо сенсор
не знайдено
  }
}
```

Уся логіка взаємодії з користувачем, обробки команд та зчитування відбитків зосереджена у функції loop().

Лістинг 4.6 – Функція loop()

```
void loop() {
  // Обробка команди на реєстрацію нового користувача
  if (pendingCommand.startsWith("r")) {
    int id = pendingCommand.substring(1).toInt();
    if (id > 0) {
      isVerifying = false;
      Serial.print("Starting enrollment for ID: ");
      Serial.println(id);

      if (getFingerprintEnroll(id)) {
        Serial.println("Enter name: ");
        while (!Serial.available());
        String name = Serial.readStringUntil('\n');
        name.trim();
        sendEnrollToServer(id, name); // Надсилання даних на
сервер
      } else {
        Serial.println("Enrollment failed.");
      }
      isVerifying = true;
    }
    pendingCommand = "";
    return;
  }

  // Обробка будь-якої команди із серійного монітора
  if (Serial.available()) {
    pendingCommand = Serial.readStringUntil('\n');
    pendingCommand.trim();
    return;
  }

  checkScannerStatus(); // Запит до сервера, чи дозволено
сканування
  if (!scannerEnabled) {
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
    return;
  }

  // Основна логіка ідентифікації
  if (isVerifying) {
    blinkWaiting(); // Миготіння LED у режимі очікування
```

```
        Serial.println("Place your finger...");
        while (finger.getImage() != FINGERPRINT_OK) {
          if (Serial.available()) {
            pendingCommand = Serial.readStringUntil('\n');
            pendingCommand.trim();
            Serial.println("[INTERRUPT] Detected command: " +
pendingCommand);
            return;
          }
          checkScannerStatus();
          if (!scannerEnabled) return;
          blinkWaiting();
          delay(100);
        }

        Serial.println("Image taken");
        digitalWrite(LED_BUILTIN, LOW);
        delay(200);

        if (finger.image2Tz(1) != FINGERPRINT_OK) return; //
Перетворення зображення у шаблон
        int result = finger.fingerSearch(); // Пошук збігу
        int idToSend = 0;

        if (result == FINGERPRINT_OK) {
          idToSend = finger.fingerID;
          Serial.print("Match found! ID: ");
          Serial.println(idToSend);
          flashLED(3); // Триразове блимання при збігу
          digitalWrite(LED_BUILTIN, HIGH);
          delay(10000);
          digitalWrite(LED_BUILTIN, LOW);
        } else {
          Serial.println("No match found.");
        }

        sendIdToServer(idToSend); // Надсилання результату на
сервер
        delay(5000);
      }
    }
```

Пояснення логіки функції loop() наведено у відповідній блок-схемі (рис. 4.14) .

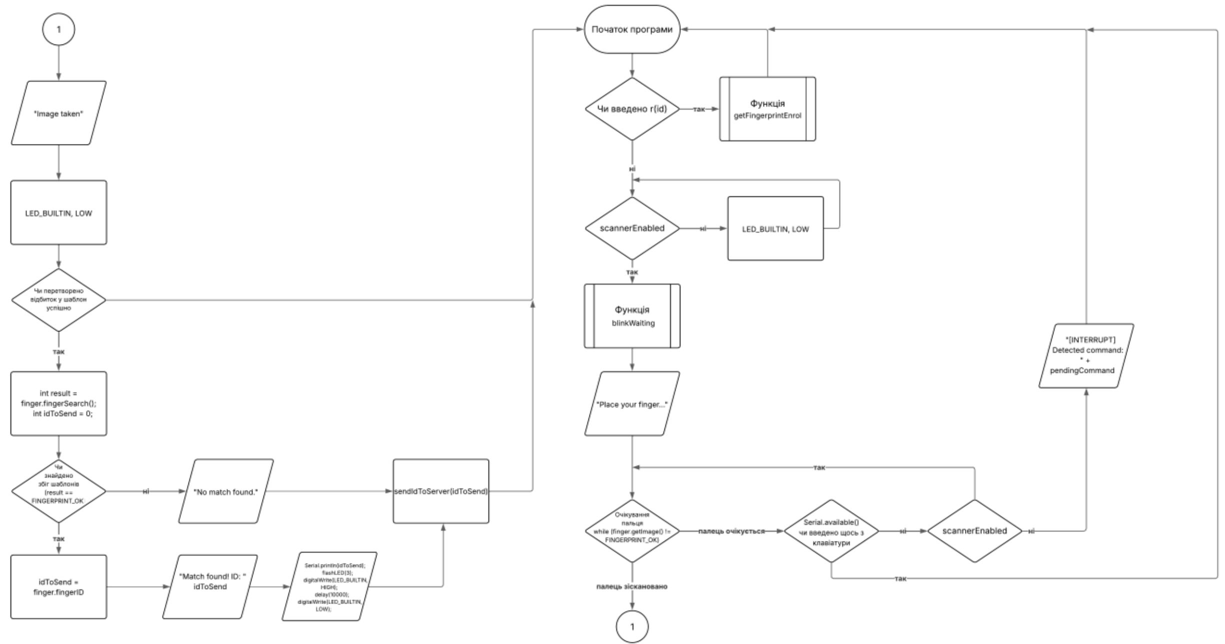


Рисунок 4.14 – Блок-схема алгоритму функції loop()

В наступній функції getFingerprintEnroll() реалізовано запис або ж перезапис нових шаблонів відбитків для порівняння.

Лістинг 4.7 – Функція getFingerprintEnroll()

```
bool getFingerprintEnroll(int id) {
  Serial.println("Place your finger...");
  while (finger.getImage() != FINGERPRINT_OK) {
    if (Serial.available()) return false; // Дозволити перервати команду
    delay(100);
  }
  if (finger.image2Tz(1) != FINGERPRINT_OK) return false; // Конвертація першого зображення

  Serial.println("Remove finger...");
  delay(2000);
  while (finger.getImage() != FINGERPRINT_NOFINGER) {
    if (Serial.available()) return false;
    delay(100);
  }

  Serial.println("Place same finger again...");
  while (finger.getImage() != FINGERPRINT_OK) {
    if (Serial.available()) return false;
```

```
          delay(100);
        }
        if (finger.image2Tz(2) != FINGERPRINT_OK) return false; //
Конвертація другого зображення
        if (finger.createModel() != FINGERPRINT_OK) return false;
// Створення моделі
        if (finger.storeModel(id) != FINGERPRINT_OK) return false;
// Збереження моделі

        return true; // Успішна реєстрація
      }
```

Пояснення логіки функції getFingerprintEnroll() наведено у відповідній блок-схемі (рис. 4.15) .

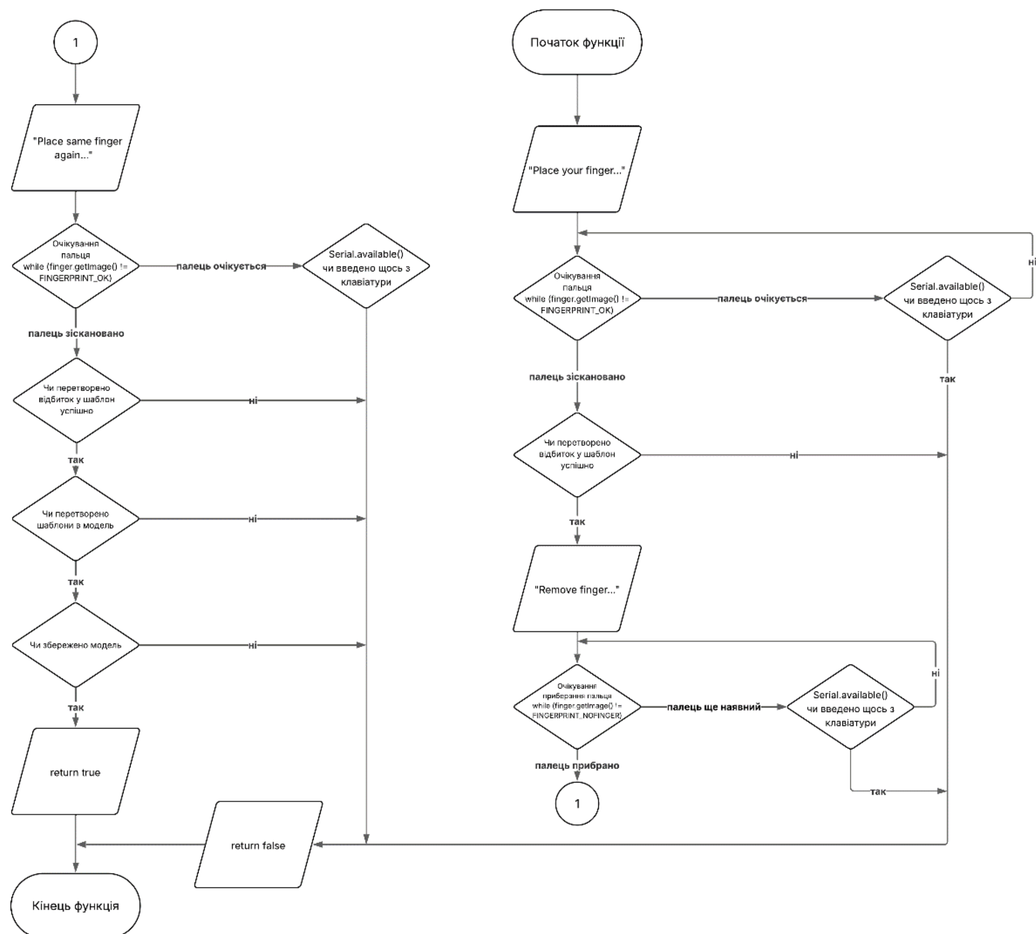


Рисунок 4.15 – Блок-схема алгоритму функції getFingerprintEnroll ()

В наступній функції checkScannerStatus() реалізовано перевірку у сервера на дозвіл роботи сканера.

Лістинг 4.8 – Функція checkScannerStatus()

```
void checkScannerStatus() {
  // Обмеження частоти запитів: максимум 1 раз на 3 секунди
  if (millis() - lastStatusCheck < 3000) return;
  lastStatusCheck = millis();

  HTTPClient http;
  http.begin(statusUrl); // ініціалізація GET-запиту
  int code = http.GET(); // Надсилання запиту

  if (code > 0) {
    String res = http.getString(); // Отримання відповіді
    scannerEnabled = (res == "1"); // Оновлення статусу
    Serial.print("Scanner status: ");
    Serial.println(scannerEnabled ? "ENABLED" : "DISABLED");
  } else {
    Serial.println("Failed to check scanner status.");
  }
  http.end(); // Закриття з'єднання
}
```

Пояснення логіки функції checkScannerStatus() наведено у відповідній блок-схемі (рис. 4.16) .

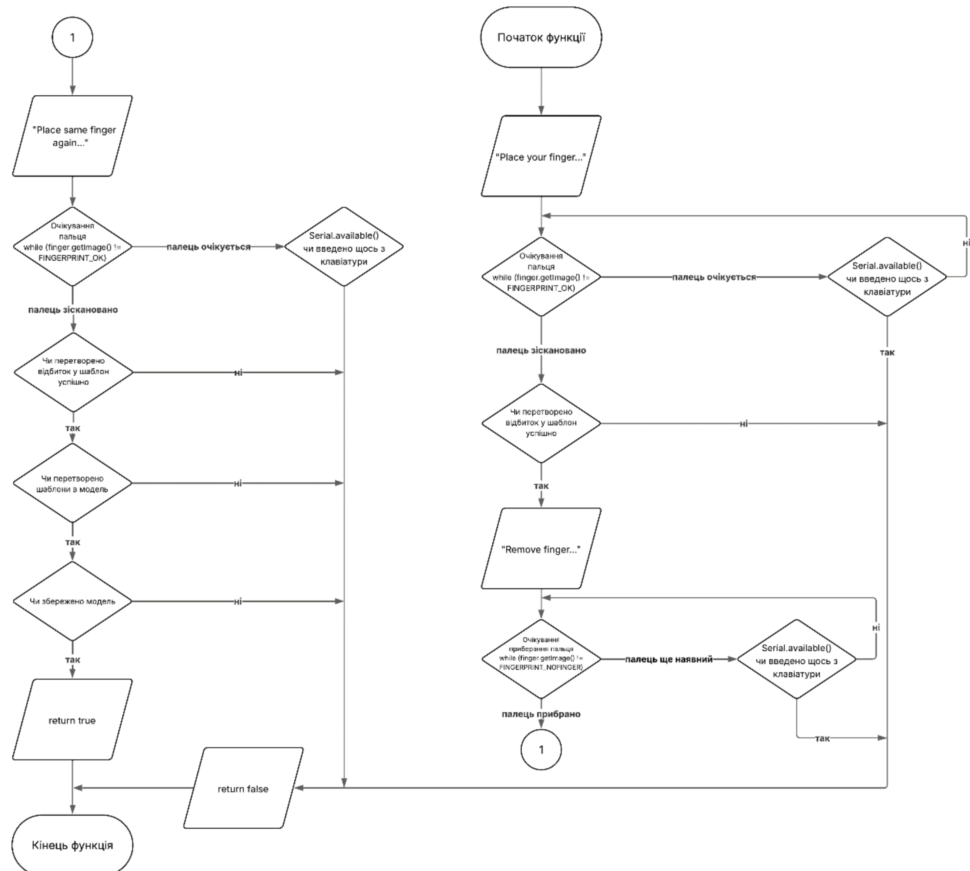


Рисунок 4.16 – Блок-схема алгоритму функції checkScannerStatus()

Наступна функція sendIdToServer() надсилає id відсканованого відбитка на сервер, що несе ключову роль в подальшій роботі логування подій.

Лістинг 4.9 – Функція sendIdToServer()

```
void sendIdToServer(int id) {
  HTTPClient http;
  http.begin(verifyUrl); // Встановлення з'єднання з сервером
  http.addHeader("Content-Type", "application/x-www-form-
urlencoded"); // Формат даних

  String postData = "id=" + String(id);
  Serial.println("Sending verification data to server: " +
postData);

  int httpResponseCode = http.POST(postData); // Надсилання
POST-запиту
  if (httpResponseCode > 0) {
    Serial.println("Server response: " + http.getString());
  } else {
```

```
        Serial.print("HTTP Error: ");
        Serial.println(http.errorToString(httpResponseCode));
      }
      http.end(); // Завершення HTTP-з'єднання
    }
```

Пояснення логіки функції sendIdToServer() наведено у відповідній блок-схемі (рис. 4.17) .

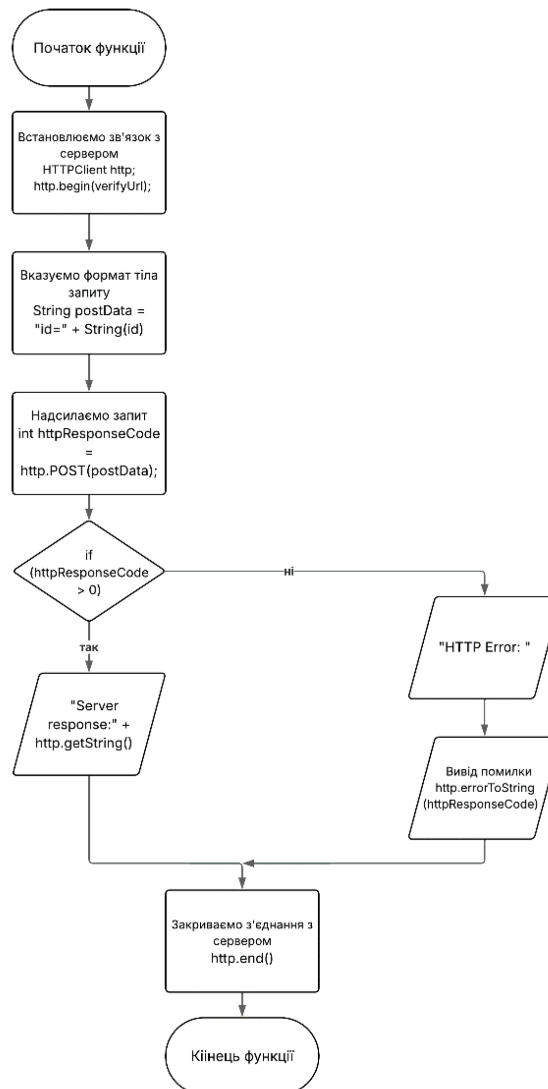


Рисунок 4.17 – Блок-схема алгоритму функції sendIdToServer()

Наступна функція sendEnrollToServer() надсилає дані нового користувача на сервер.

Лістинг 4.10 – Функція sendEnrollToServer()

```
void sendEnrollToServer(int id, String name) {
  HTTPClient http;
  http.begin(enrollUrl); // З'єднання з URL реєстрації
  http.addHeader("Content-Type", "application/x-www-form-
urlencoded");

  String postData = "id=" + String(id) + "&name=" + name;
  Serial.println("Sending enrollment data to server: " +
postData);

  int httpResponseCode = http.POST(postData); // POST-запит з
ID та іменем
  if (httpResponseCode > 0) {
    Serial.println("Server response: " + http.getString());
  } else {
    Serial.print("HTTP Error: ");
    Serial.println(http.errorToString(httpResponseCode));
  }
  http.end(); // Завершення HTTP-з'єднання
}
```

Пояснення логіки функції sendEnrollToServer() наведено у відповідній блок-схемі (рис. 4.18) .

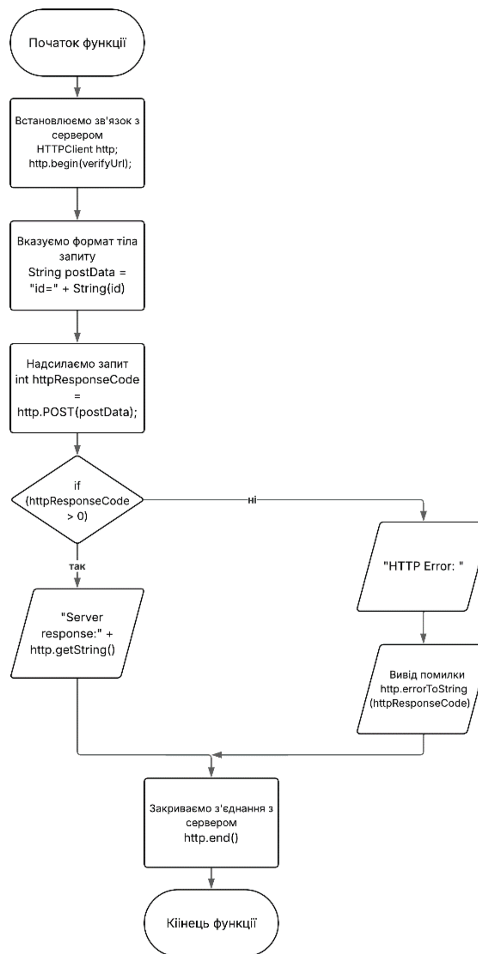


Рисунок 4.18 – Блок-схема алгоритму функції sendEnrollToServer()

Наступна функція blinkWaiting() виконує блимання світлодіода під час очікування пальця.

Лістинг 4.11 – Функція blinkWaiting()

```
void blinkWaiting() {
  static unsigned long lastBlink = 0;
  static bool ledState = false;

  // Перевіряємо, чи минуло 500 мс з останнього миготіння
  if (millis() - lastBlink >= 500) {
    ledState = !ledState;
    digitalWrite(LED_BUILTIN, ledState); // Перемикання стану світлодіода
    lastBlink = millis();
  }
}
```

Пояснення логіки функції blinkWaiting() наведено у відповідній блок-схемі

(рис. 4.19) .

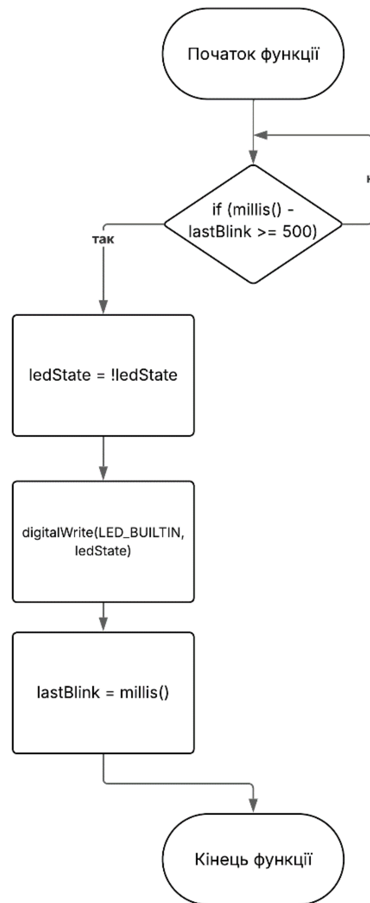


Рисунок 4.19 – Блок-схема алгоритму функції blinkWaiting()

Остання функція в скетчі b flashLED() виконує блимання світлодіода задану кількість разів.

Лістинг 4.12 – Функція flashLED()

```
void flashLED(int times) {
  // Цикл блимання світлодіода задану кількість разів
  for (int i = 0; i < times; i++) {
    digitalWrite(LED_BUILTIN, HIGH); // Увімкнення
світлодіода
    delay(300);
    digitalWrite(LED_BUILTIN, LOW); // Вимкнення світлодіода
    delay(300);
  }
}
```

Пояснення логіки функції flashLED() наведено у відповідній блок-схемі (рис. 4.20) .

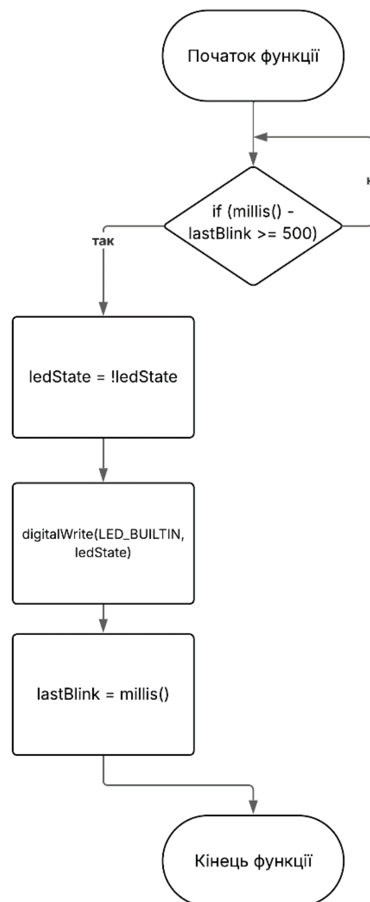


Рисунок 4.20 – Блок-схема алгоритму функції flashLED()

4.3.4. Підготовка до розробки веб-сервера

Для реалізації серверної частини нашого проекту було обрано мову програмування Python та фреймворк Flask [4], що створений для розробки подібних серверів.

Перш за все потрібно було завантажити та встановити (рис. 4.21) саму мову Python, що виконується швидко та легко через офіційний сайт .

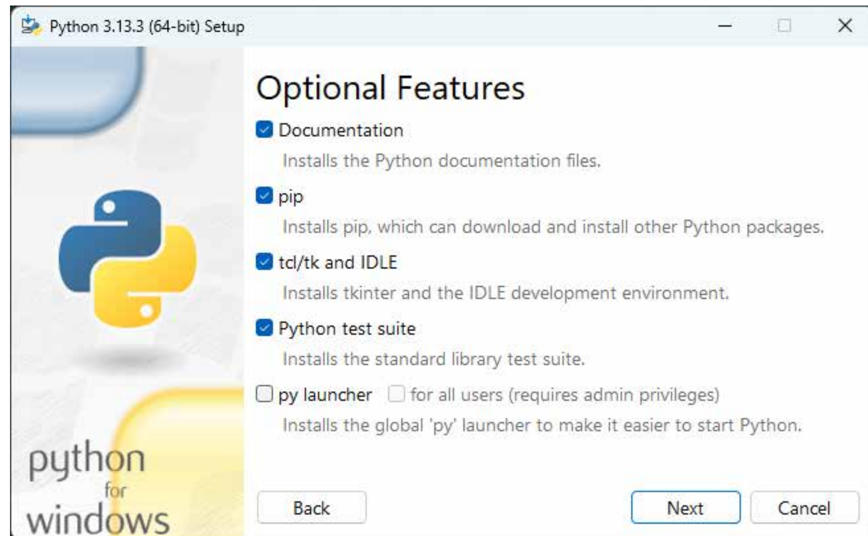


Рисунок 4.21 – Вікно встановлення Python

Тепер необхідно встановити середовище розробки, було обрано PyCharm (рис. 4.22), котрий розроблений спеціально для роботи з Python.

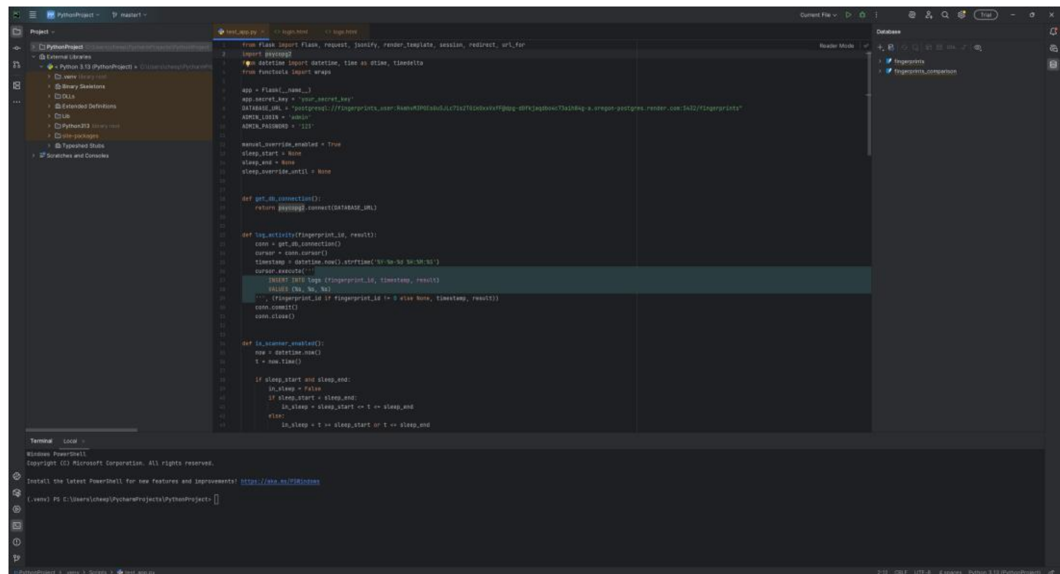


Рисунок 4.22 – Інтерфейс PyCharm

Після створення проекту середовище майже готове до роботи, проте потрібно встановити додаткові бібліотеки, що знадобляться для нашого коду, а найголовніше фреймворк Flask, що і дозволить нам написати наш веб-сервер.

У початковій реалізації проєкту серверна частина працювала з локальною базою даних SQLite через ORM-бібліотеку SQLAlchemy, інтегровану у Flask за

допомогою Flask-SQLAlchemy. Такий підхід дозволив швидко протестувати функціонал логування, збереження користувачів та роботи з шаблонами.

Завантаження бібліотек виконується за допомогою команди pip install <назва_бібліотеки> надісланої в термінал PyCharm .

Лістинг 4.13 – Команди для встановлення необхідних бібліотек

```
pip install flask
pip install psycopg2-binary
pip install pytz
pip install flask-sqlalchemy
pip install sqlalchemy
pip install greenlet
```

Надалі будемо розглядати вже кінцевий варіант сервера адаптований до хмарного розгортання (додаток Б), проте про етап локального тестування та його особливості варто було зазначити.

4.3.5. Перелік функцій, реалізованих у веб-сервері

– Обробка запитів на перевірку відбитка — прийом ID, логування спроби входу, повернення результату (дозволено / заборонено)

– Обробка запитів на статус сканера — перевірка активності сканера з урахуванням режиму сну та ручного керування.

– Запис подій у базу даних — логування спроб доступу із зазначенням часу, результату та ідентифікатора.

– Надання адміністративної панелі — перегляд журналу спроб входу через HTML-інтерфейс.

– Авторизація адміністратора — захист сторінок авторизацією через сесію.

– Можливість вмикання/вимикання сканера— дистанційне керування

дозволом на зчитування.

– Зміна режиму сну— встановлення часових меж, у які сканер автоматично вимикається.

– Робота з базою даних SQLite — в тестовому варіанті, PostgreSQL – в кінцевому.

– Обробка запитів на реєстрацію нового користувача — прийом ID та імені, збереження нового користувача у базі або оновлення імені, якщо ID вже існує.

4.3.6. Написання коду веб-сервера

Імпортуємо бібліотеки.

Лістинг 4.14 – Імпорт необхідних бібліотек

```
from flask import Flask, request, jsonify, render_template, session, redirect, url_for  # Flask — основа веб-сервера. request — обробка запитів. jsonify — JSON-відповіді. session — авторизація.
import psycopg2  # Бібліотека для з'єднання з PostgreSQL
from pytz import timezone  # Бібліотека для роботи з часовими поясами
from datetime import datetime, time as dtime, timedelta  # Модуль для роботи з часом і датами
from functools import wraps  # Декоратор для обгортки функцій авторизації
```

Оголошення глобальних змінних і констант

Лістинг 4.15 – Оголошення глобальних змінних і констант

```
app = Flask(__name__)  # Створення веб-додатку на Flask
app.secret_key = '***'  # Секретний ключ для сесій користувача
DATABASE_URL = " postgresql://..."  # URL з'єднання з PostgreSQL-базою
ADMIN_LOGIN = '***'  # Облікові дані адміністратора
ADMIN_PASSWORD = '***'

kyiv = timezone('Europe/Kyiv')  # Змінна, що відповідає за
```

```
місцевий час

    manual_override_enabled = True  # Ручний дозвіл
увімкнення/вимкнення сканера
    sleep_start = None
    sleep_end = None
    sleep_override_until = None  # Змінні для задання режиму
"сну"
```

У наведеному лістингу значення логіна, пароля, секретного ключа та посилання на базу даних замінені на умовні позначення з міркувань безпеки.

Функція get_db_connection() відповідає за встановлення з’єднання з онлайн-базою даних PostgreSQL.

Лістинг 4.16 – Встановлення з’єднання з БД

```
    def get_db_connection():  # Повертає нове з’єднання з базою
PostgreSQL
        return psycopg2.connect(DATABASE_URL)
```

Функція log_activity() реалізує процес логування подій доступу до системи в базу даних.

Лістинг 4.17 – Функція log_activity()

```
    def log_activity(fingerprint_id, result):
        conn = get_db_connection()
        cursor = conn.cursor()
        timestamp = datetime.now(kyiv).strftime('%Y-%m-%d
%H:%M:%S')  # Отримання поточного часу для логування
        cursor.execute('''
            INSERT INTO logs (fingerprint_id, timestamp, result)
            VALUES (%s, %s, %s)
        ''', (fingerprint_id if fingerprint_id != 0 else None,
timestamp, result))  # Формування запиту до БД, з перевіркою id =
0 → None
        conn.commit()  # Збереження змін
        conn.close()  # Закриття з’єднання з базою
```

Пояснення логіки функції log_activity() наведено у відповідній блок-схемі (рис. 4.23) .

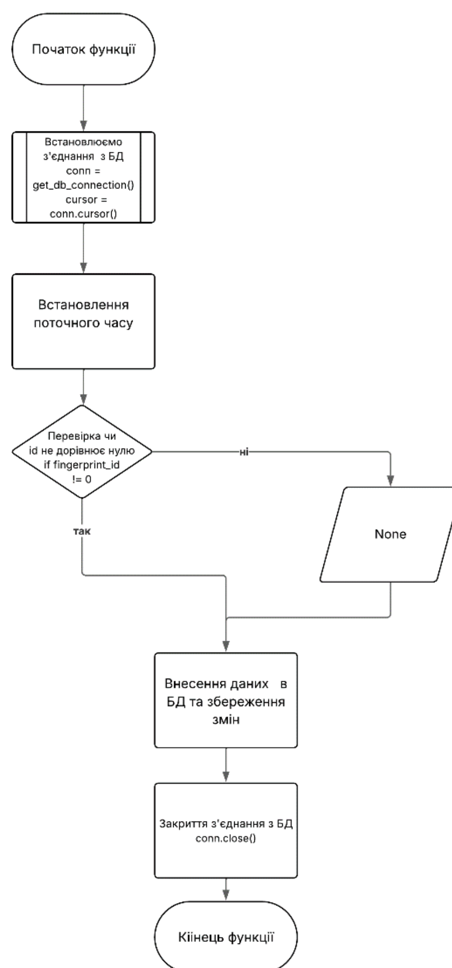


Рисунок 4.23 – Блок-схема алгоритму функції log_activity()

Функція is_scanner_enabled() визначає, чи дозволено в даний момент використання сканера відбитків пальців. Вона реалізує логіку дистанційного контролю доступу до сканера, враховуючи як встановлений режим сну , так і можливе ручне перевизначення.

Лістинг 4.18 – Функція is_scanner_enabled()

```
    def is_scanner_enabled():  # Функція для реалізації
дистанційного контролю доступу до сканера
        global manual_override_enabled, sleep_override_until
        now = datetime.now(kyiv)  # Поточний час у часовій зоні
Європа/Київ
        t = now.time()  # Отримуємо лише час (без дати)
```

```
        if sleep_start and sleep_end:
            in_sleep = False
            if sleep_start < sleep_end:
                in_sleep = sleep_start <= t <= sleep_end  # 
Перевірка в межах однієї доби
            else:
                in_sleep = t >= sleep_start or t <= sleep_end  # 
Перевірка через північ

            if in_sleep:
                if sleep_override_until and now < 
sleep_override_until:
                    return True  # Якщо є ручне перевизначення – 
сканер доступний
                manual_override_enabled = False
                return False  # Інакше сканер вимкнено через 
режим сну

        return manual_override_enabled  # Якщо немає режиму сну – 
орієнтуємось на ручне керування
```

Пояснення логіки функції is_scanner_enabled() наведено у відповідній блок-схемі (рис. 4.24) .

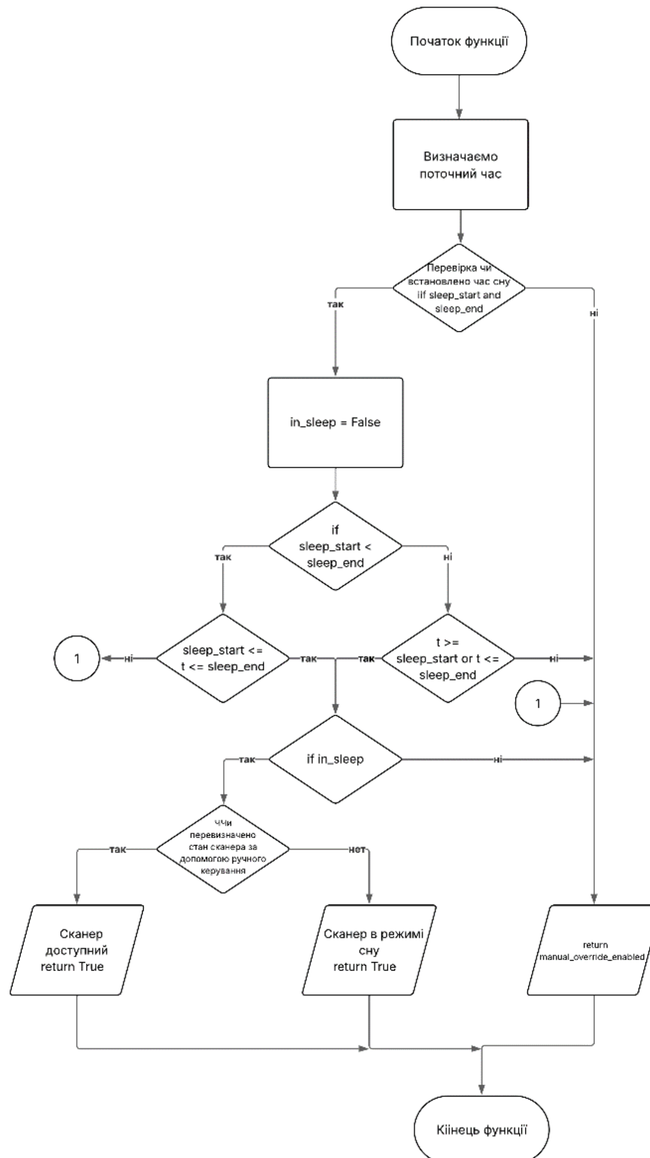


Рисунок 4.24 – Блок-схема алгоритму функції is_scanner_enabled()

Декоратор для контролю доступу до наступних функцій, що контролюватиме, щоб певні функції працювали лише для авторизованих користувачів.

Лістинг 4.19 – Декоратор admin_required(f)

```
def admin_required(f):  # Приймає функцію f, яку потрібно
захистити
    @wraps(f)  # Зберігає ім’я, докстрінг та інші метадані
функції
```

```
    def decorated_function(*args, **kwargs):
        if session.get('logged_in'):
            return f(*args, **kwargs)  # Якщо користувач увійшов
 - виконуємо функцію
        return redirect(url_for('login'))  # Інакше перекидаємо
 на сторінку входу
    return decorated_function
```

Пояснення логіки декоратора admin_required(f) наведено у відповідній блок-схемі (рис. 4.25) .

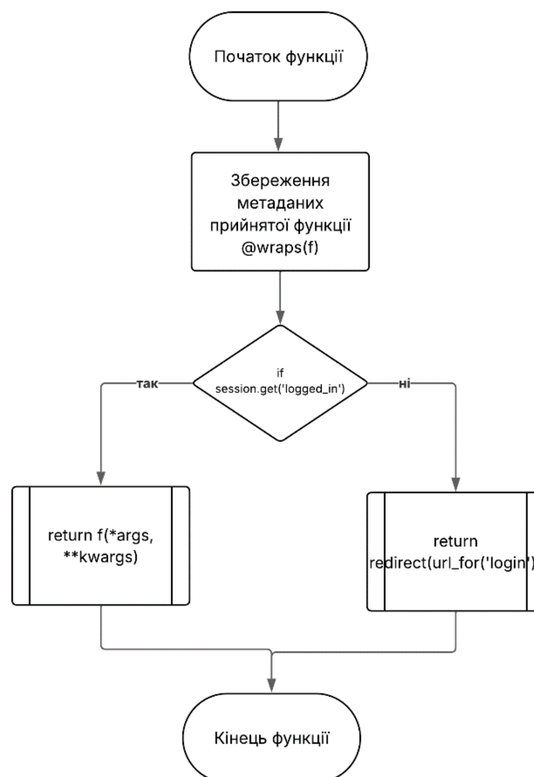


Рисунок 4.25 – Блок-схема алгоритму декоратора admin_required(f)

Функція verify() обробляє запити на перевірку доступу, які надходять від мікроконтролера. Після отримання POST-запиту з id користувача система перевіряє, чи активний сканер згідно з режимом роботи. У разі активності виконується логування спроби доступу до бази даних із відповідним статусом. Функція формує JSON-відповідь, що надсилається назад на пристрій для подальшої дії.

Лістинг 4.20 – Функція verify()

```
@app.route('/verify', methods=['POST'])
def verify():
    if not is_scanner_enabled():  # Якщо сканер неактивний — повертаємо відповідь про недоступність
        return jsonify({"match": "false", "message": "Scanner disabled"}), 403

    fid = int(request.form.get('id', 0))  # Отримуємо ID з POST-запиту
    if fid > 0:
        log_activity(fid, "Access granted")  # Логування спроби з доступом
        return jsonify({"match": "true", "id": fid}), 200
    else:
        log_activity(0, "Access denied")  # Логування відмови у доступі
        return jsonify({"match": "false"}), 200
```

Пояснення логіки функції verify() наведено у відповідній блок-схемі (рис. 4.26) .

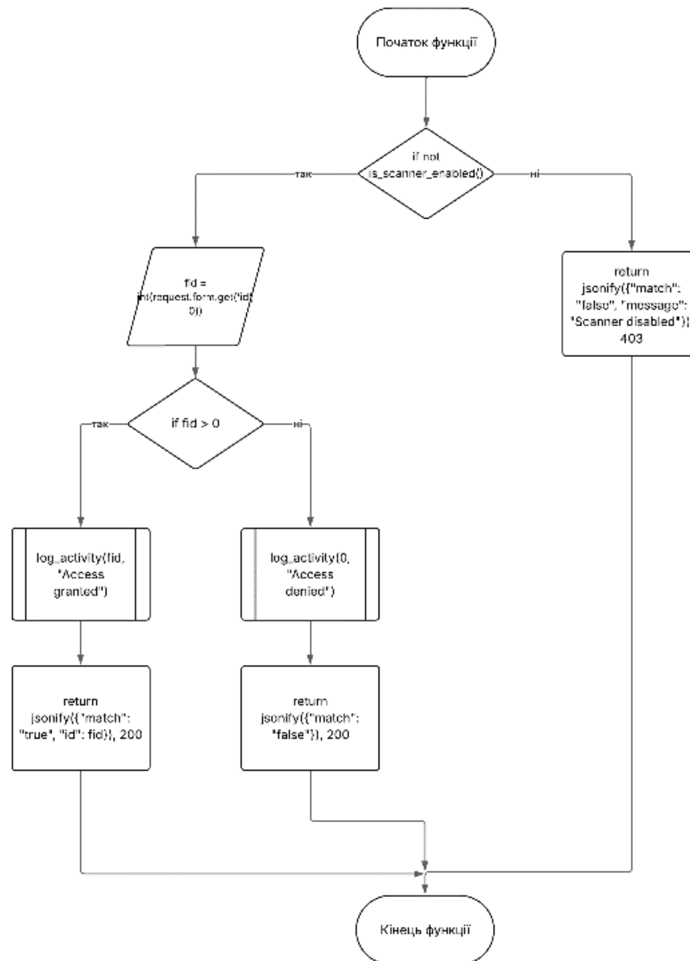


Рисунок 4.26 – Блок-схема алгоритму декоратора функції verify()

Функція enroll() виконує додавання нового користувача або оновлення його імені у базі даних системи. Після отримання POST-запиту з ідентифікатором та іменем, відбувається вставка запису у відповідну таблицю або оновлення імені у разі конфлікту за ідентифікатором (при повторній реєстрації).

Лістинг 4.21 – Функція enroll()

```
def enroll():
    fid = int(request.form.get('id', 0))  # Отримуємо ID користувача
    name = request.form.get('name', '').strip()  # Отримуємо ім’я користувача

    if fid > 0 and name:
        conn = get_db_connection()
        cursor = conn.cursor()
```

```
            cursor.execute('''
                INSERT INTO fingerprints (id, name)
                VALUES (%s, %s)
                ON CONFLICT (id) DO UPDATE SET name =
EXCLUDED.name
            ''', (fid, name))  # Додаємо нового користувача або
оновлюємо ім’я при співпадінні ID
            conn.commit()
            conn.close()
            return jsonify({"status": "success", "id": fid,
"name": name})
        else:
            return jsonify({"status": "error", "message":
"Invalid data"}), 400
```

Пояснення логіки функції enroll() наведено у відповідній блок-схемі (рис. 4.27) .

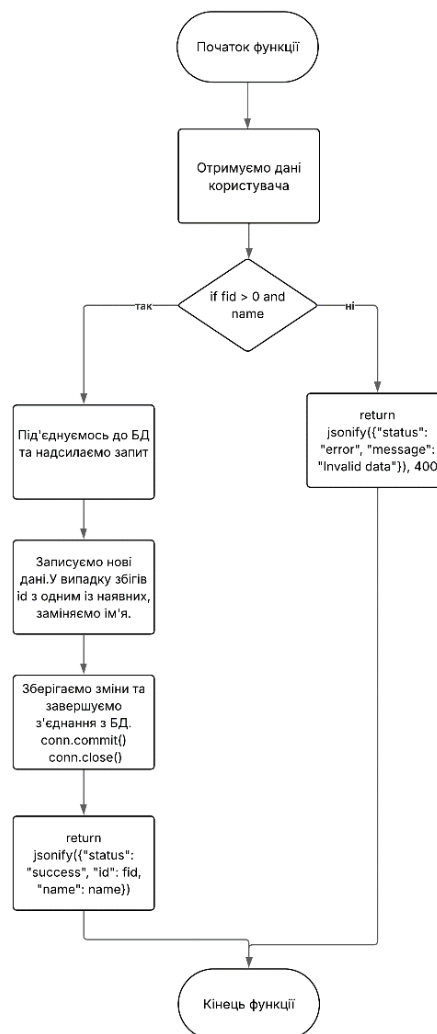


Рисунок 4.27 – Блок-схема алгоритму декоратора функції enroll()

Функція toggle_scanner() реалізує можливість ручного увімкнення або вимкнення сканера відбитків пальців адміністратором. Вона враховує заданий графік режиму сну пристрою та, за потреби, дозволяє тимчасове переривання цього режиму шляхом активації сканера до завершення поточного періоду сну. У разі повторного виклику функції здійснюється скасування ручного керування, що повертає систему до автоматичного розкладу роботи.

Лістинг 4.22 – Функція toggle_scanner()

```
@admin_required
def toggle_scanner():
    global manual_override_enabled, sleep_override_until
    now = datetime.now(kyiv)  # Поточний час у часовій зоні

    if not is_scanner_enabled():  # Якщо сканер зараз
вимкнений
        manual_override_enabled = True  # Вмикаємо вручну
        if sleep_end:
            sleep_end_today = datetime.combine(now.date(),
sleep_end)
            if sleep_start and sleep_start > sleep_end and
now.time() < sleep_end:
                sleep_end_today += timedelta(days=1)
            sleep_override_until =
kyiv.localize(sleep_end_today)  # Сканер активний до кінця графіку
    else:
        manual_override_enabled = False  # Вимикаємо вручну
        sleep_override_until = None  # Скасовуємо ручне
перевизначення

    return redirect(url_for('logs'))  # Переадресація до
журналу подій
```

Пояснення логіки функції toggle_scanner() наведено у відповідній блок-схемі (рис. 4.28) .

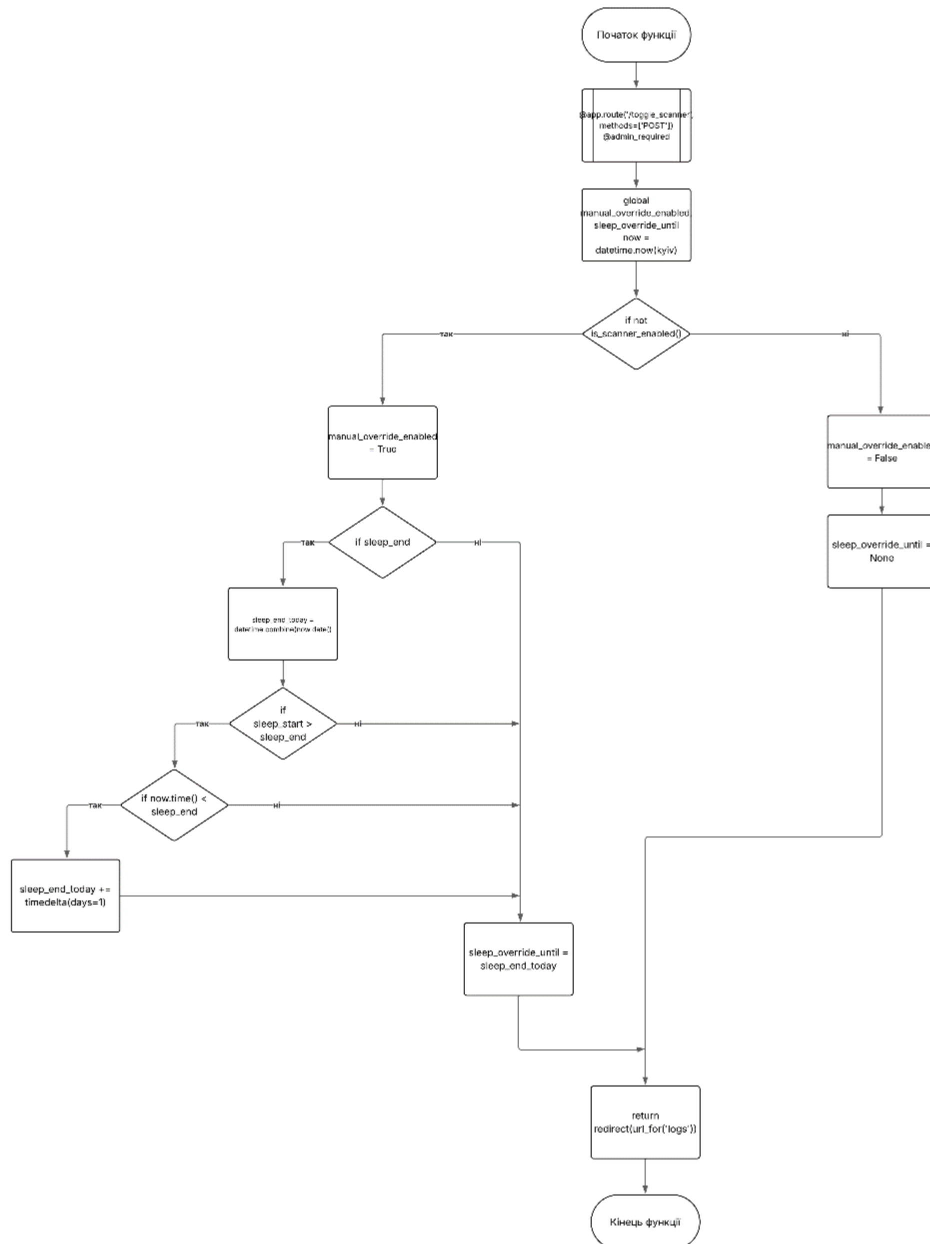


Рисунок 4.28 – Блок-схема алгоритму декоратора функції toggle_scanner()

Функція logs() відповідає за формування зведеної таблиці результатів сканування, здійснюючи з'єднання з онлайн-базою даних PostgreSQL. Вона виконує SQL-запит, що об'єднує таблиці відбитків та журналу перевірок, і впорядковує записи за часом у спадному порядку. Отримані дані передаються у HTML-шаблон для відображення у вигляді зручної для перегляду таблиці, яка містить інформацію про дату, час, результат і користувача.

Лістинг 4.23 – Функція set_sleep_schedule()

```
@app.route('/set_sleep_schedule', methods=['POST'])
@admin_required
def set_sleep_schedule():
    global sleep_start, sleep_end, sleep_override_until
    try:
        h1, m1 = map(int, 
request.form['from_time'].split(':'))
        h2, m2 = map(int, request.form['to_time'].split(':'))
        new_start = dtime(hour=h1, minute=m1)
        new_end = dtime(hour=h2, minute=m2)

        # Якщо змінився розклад - скидаємо ручне керування
        if sleep_start != new_start or sleep_end != new_end:
            sleep_override_until = None

        sleep_start = new_start
        sleep_end = new_end
    except Exception as e:
        print(f" Помилка встановлення режиму сну: {e}")

    return redirect(url_for('logs'))  # Переадресація на 
сторінку логів
```

Наступна функція scanner_status() визначає стан сканера.

Лістинг 4.24 – Функція scanner_status()

```
@app.route('/scanner_status')
def scanner_status():
    return '1' if is_scanner_enabled() else '0'  # Повертає 
1, якщо сканер активний, інакше 0
```

Далі йде функція logs() , що структурує інформацію про сканування в зручну таблицю взаємодіючи з онлайн БД.

Лістинг 4.25 – Функція logs()

```
@app.route('/logs')
@admin_required
def logs():
```

```
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('''
            SELECT l.verification_id, f.id, f.name, l.timestamp,
l.result
            FROM logs l LEFT JOIN fingerprints f ON
l.fingerprint_id = f.id
            ORDER BY l.timestamp DESC
        ''')  # Об'єднання таблиці логів і відбитків
        logs_data = cursor.fetchall()
        conn.close()
        return render_template('logs.html', logs=logs_data,

scanner_enabled=is_scanner_enabled(),

manual_override=manual_override_enabled,
                           sleep_start=sleep_start,
sleep_end=sleep_end)
```

Функція login() реалізує механізм авторизації адміністратора системи. У випадку надсилання POST-запиту з правильними обліковими даними, користувач отримує відповідний сесійний маркер, що підтверджує його авторизований статус. Успішна авторизація призводить до перенаправлення на сторінку перегляду журналу подій. У разі невірних даних формується повідомлення про помилку авторизації.

Лістинг 4.26 – Функція login()

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    login_error = None
    if request.method == 'POST':
        login = request.form.get('login')
        password = request.form.get('password')
        if login == ADMIN_LOGIN and password ==
ADMIN_PASSWORD:
```

```
                session['logged_in'] = True
                return redirect(url_for('logs'))
            else:
                login_error = "Невірний логін або пароль"
        return render_template('login.html',
login_error=login_error)
```

Пояснення логіки функції login() наведено у відповідній блок-схемі (рис. 4.29) .

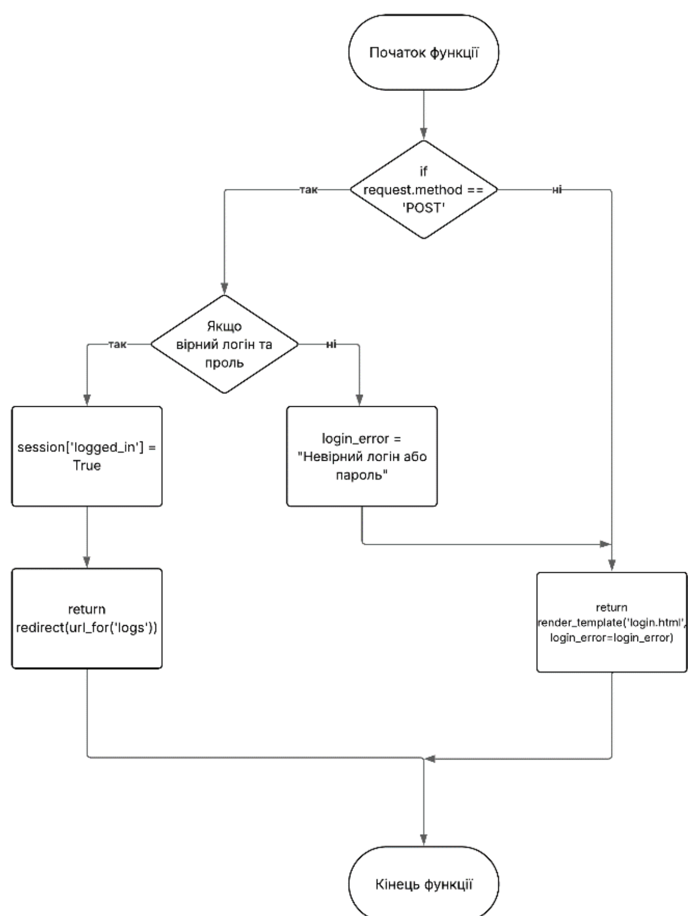


Рисунок 4.29 – Блок-схема алгоритму декоратора функції login()

Функція logout() відповідає за завершення поточної адміністративної сесії шляхом видалення маркера авторизації з об’єкта сесії. Після цього користувача автоматично перенаправляє на сторінку входу, що забезпечує контроль доступу та запобігає несанкціонованому перегляду журналу подій після виходу.

Лістинг 4.27 – Функція logout()

```
@app.route('/logout')
def logout():
    session.pop('logged_in', None)  # Видаляємо ознаку
авторизації з сесії
    return redirect(url_for('login'))  # Повертаємо на
сторінку входу
```

Прописуємо команду для запуску нашого серверу. Цей фрагмент виконується лише у випадку, якщо даний файл запускається напряму (а не імпортується в інший модуль), що дозволяє безпечно інкапсулювати запуск програми.

Лістинг 4.28 – Команда для запуску серверу

```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)  # Команда
для запуску серверу
```

4.3.7. Створення БД

На початковому етапі розробки система використовувала локальну базу даних на основі SQLite, що забезпечувало просту й швидку взаємодію в умовах локального середовища. Однак згодом, у процесі перенесення всієї системи до хмарного середовища Render, було прийнято рішення про міграцію на PostgreSQL — більш масштабовану та надійну СУБД, підтримувану платформою Render. Нижче наведено фрагмент коду ініціалізації локальної бази даних у форматі SQLite, що використовувалася на ранньому етапі розробки [18]. У наступному розділі буде розглянуто процес перенесення бази даних до хмари.

Лістинг 4.29 – Функція init_db

```
import sqlite3

def init_db(db_path='fingerprints.db'):
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
```

```
    # Створення таблиці користувачів за відбитками пальців
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS fingerprints (
            id INTEGER PRIMARY KEY UNIQUE,
            name TEXT NOT NULL
        );
    ''')
    # Створення таблиці логів верифікацій
    cursor.execute('''
CREATE TABLE IF NOT EXISTS verification_log (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp DATETIME NOT NULL,
            success BOOLEAN NOT NULL,
            matched_id INTEGER NOT NULL
        );
    ''')

    conn.commit()
    conn.close()
    print(" База даних та таблиці успішно створено.")
```

4.3.8. HTML-сторінки веб-серверу

Нижче наведено опис структури HTML-файл, який відповідає за відображення сторінки входу до системи адміністрування. Для зручності розглядається лише функціональна частина (форма авторизації), без стилів CSS, оскільки вони не впливають на логіку взаємодії з користувачем. Повна версія HTML-документа із CSS оформленням надається у додатку (додаток В) до пояснювальної записки.

Лістинг 4.30 – Функціональна частина login.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Вхід до системи</title>
```

```
</head>
<body>
<div class="login-box">  <!-- Основний контейнер форми входу
-->
        <h2>Вхід до системи</h2>  <!-- Заголовок сторінки -->

        <form method="post">  <!-- Форма авторизації з
методом POST -->
            <label for="login">Логін</label>
            <input type="text" id="login" name="login"
required>  <!-- Поле введення логіну -->

            <label for="password">Пароль</label>
            <input type="password" id="password"
name="password" required>  <!-- Поле введення паролю -->

            <button type="submit">Увійти</button>  <!--
Кнопка відправки форми -->
        </form>
    </div>
</body>
</html>
```

Нижче подано опис структури HTML-файлу, що відповідає за відображення сторінки журналу верифікацій у веб-інтерфейсі адміністратора. Повну версію файлу наведено в додатку (додаток Г).

Лістинг 4.31 – Функціональна частина logs.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Журнал верифікацій</title>
</head>
<body>
<div class="container">

    <a href="{{ url_for('logout') }}"
class="logout">Вийти</a>  <!-- Посилання на вихід із системи -->
```

```
        <h1>Журнал верифікацій</h1>  <!-- Заголовок сторінки -->

        <!-- Блок стану сканера і кнопка перемикання -->
        <div class="status-section">
            <div class="scanner-status">
                Сканер:
                <span>{{ '🟢 Увімкнено' if scanner_enabled else
'🔴 Вимкнено' }}</span>
            </div>
            <form method="post" action="{{
url_for('toggle_scanner') }}">
                <button type="submit">
                    {{ '🔴 Вимкнути сканер' if scanner_enabled
else '🟢 Увімкнути сканер' }}
                </button>
            </form>
        </div>

        <!-- Блок встановлення режиму сну -->
        <h2>Режим сну</h2>
        <form method="post" action="{{
url_for('set_sleep_schedule') }}">
            <label for="from_time">З:</label>
    <input type="time" id="from_time" name="from_time" required>

            <label for="to_time">До:</label>
            <input type="time" id="to_time" name="to_time"
required>

            <button type="submit">Встановити</button>
        </form>

        <!-- Вивід активного графіку сну -->
        {% if sleep_start and sleep_end %}
            <p>Час сну: з {{ sleep_start.strftime('%H:%M') }} до
{{ sleep_end.strftime('%H:%M') }}</p>
        {% endif %}

        <!-- Таблиця логів верифікацій -->
        <table>
            <thead>
                <tr>
                    <th>ID верифікації</th>
                    <th>ID відбитка</th>
```

```
                    <th>ПІБ</th>
                    <th>Час</th>
                    <th>✅</th>
                </tr>
            </thead>
            <tbody>
                {% for log in logs %}
                <tr>
                    <td>{{ log[0] }}</td>
                    <td>{{ log[1] if log[1] is not none else '—'
}}</td>
                    <td>{{ log[2] if log[2] is not none else '—'
}}</td>
                    <td>{{ log[3] }}</td>
                    <td>{{ '🟢' if log[4] == 'Access granted'
else '🔴' }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>

    </div>
    </body>
    </html>
```

4.4. Перенесення в хмару та розгортання сервера

4.4.1. Перенесення БД

Як вже було зазначено для переходу від системи локальної до хмарного рішення потрібно було для початку перенести базу даних з SQLite на PostgreSQL розгорнувши її на платформі Render. Для цього створюємо нову БД в Render [19] (рис. 4.30-4.32) .

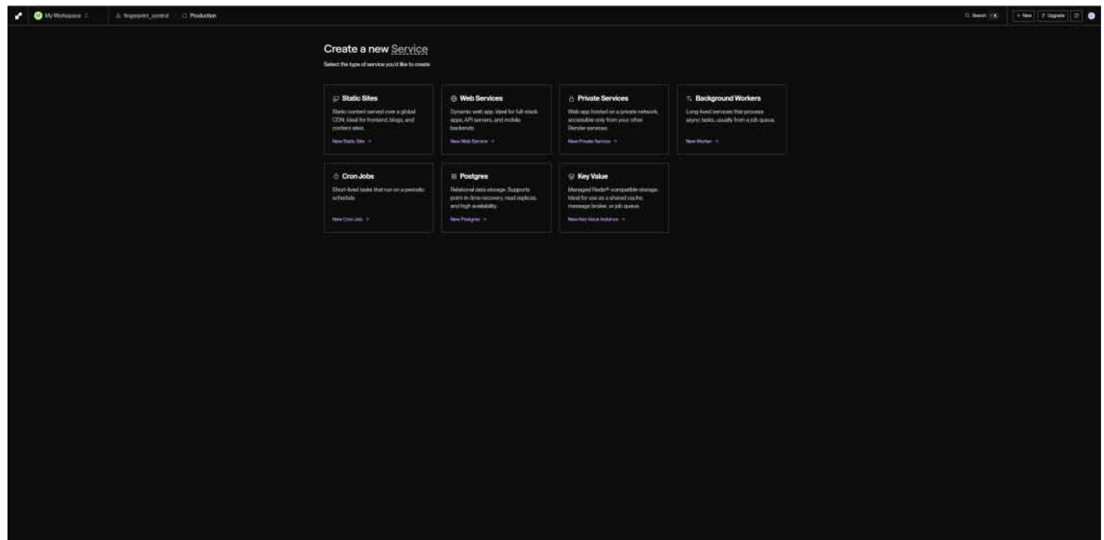


Рисунок 4.30 – Головне вікно створення нових сервісів Render

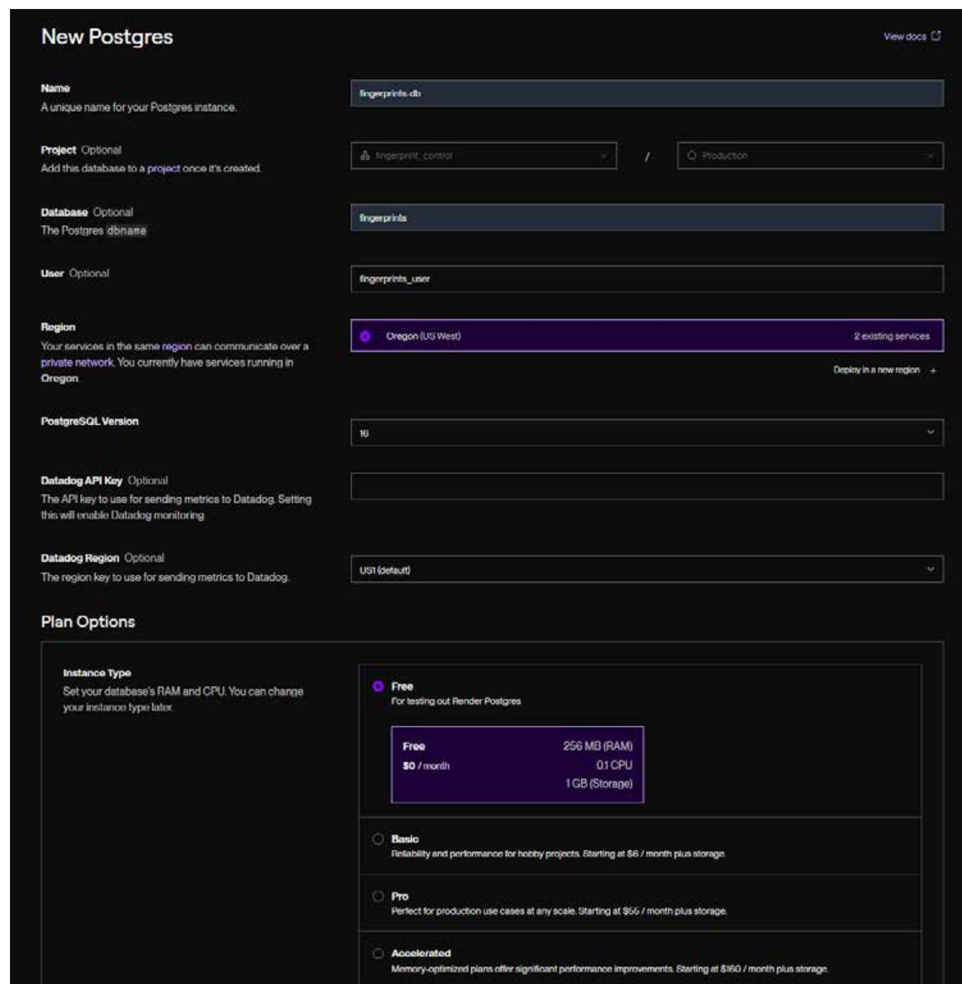


Рисунок 4.31 – Вікно створення нової БД в Render

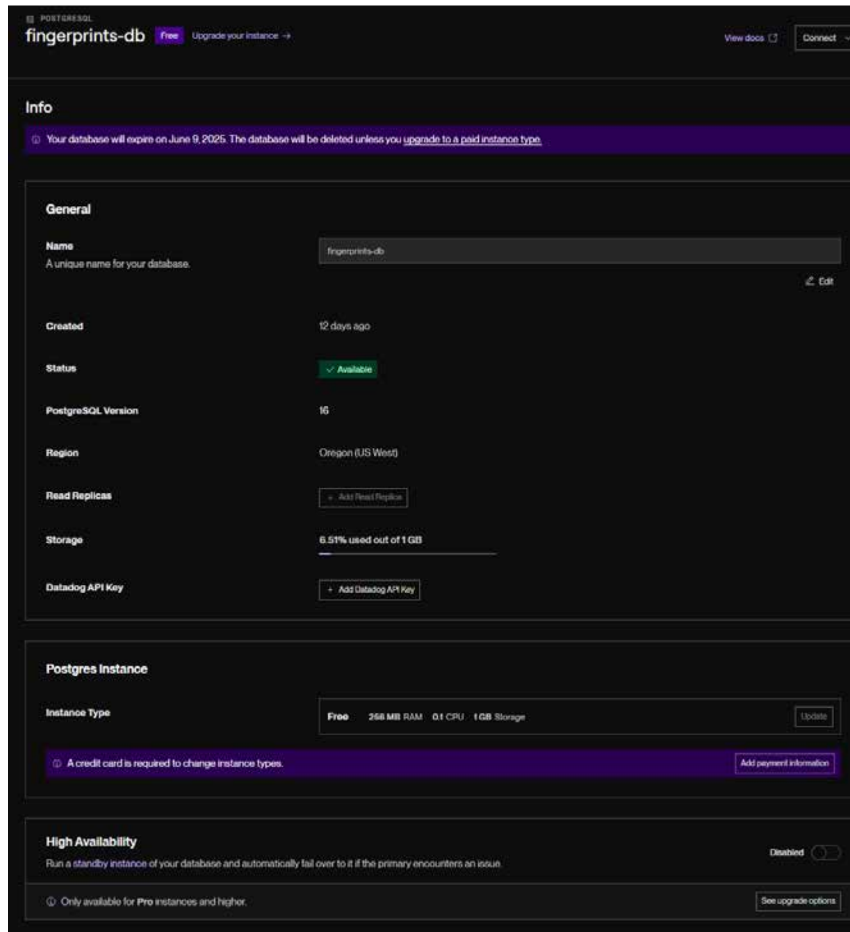


Рисунок 4.32 – Вікно створеної БД

Далі важливо в параметрах з’єднання (рис. 4.33) копіювати засекречене посилання для встановлення зі’язку між сервером та БД.

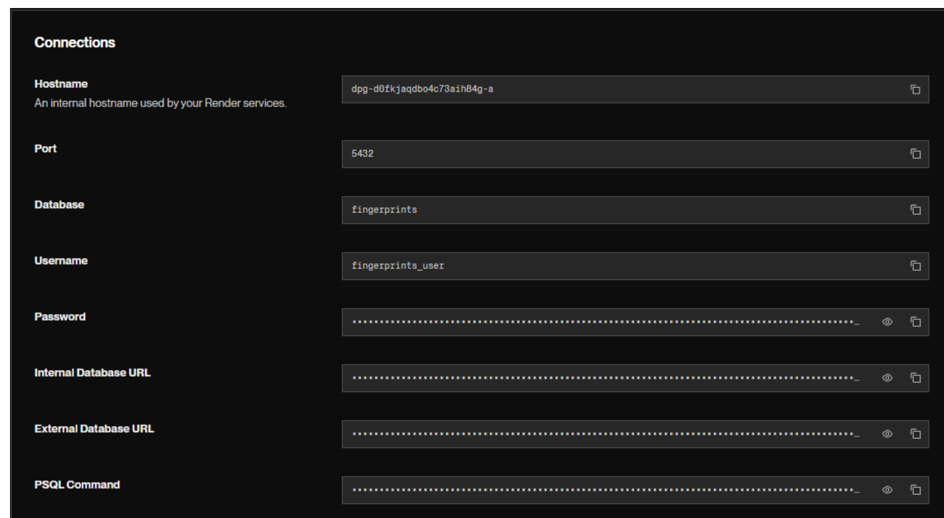


Рисунок 4.33 – Параметри з’єднання з таблицею

Наступним кроком було написано скрипт (додаток Д), що встановлює з’єднання з базою даних, створює відповідні таблиці та переносить записи з локальної БД.

Лістинг 4.32 – Скрипт migrate_to_render.py

```
import sqlite3
import psycopg2

#  Підключення до локальної SQLite
sqlite_conn = sqlite3.connect('fingerprints.db')
sqlite_cursor = sqlite_conn.cursor()

#  Підключення до  PostgreSQL
pg_conn = psycopg2.connect(
    "postgresql://fingerprints_user:тут має бути валідне
посилання для з’єднання "
)
pg_cursor = pg_conn.cursor()

#  Створення таблиць у PostgreSQL
pg_cursor.execute('''
CREATE TABLE IF NOT EXISTS fingerprints (
        id INTEGER PRIMARY KEY,
        name TEXT
    )
''')

pg_cursor.execute('''
    CREATE TABLE IF NOT EXISTS logs (
        verification_id SERIAL PRIMARY KEY,
        fingerprint_id INTEGER,
        timestamp TIMESTAMP,
        result TEXT,
        FOREIGN KEY (fingerprint_id) REFERENCES
fingerprints(id)
    )
''')
pg_conn.commit()

#  Отримання даних з SQLite
sqlite_cursor.execute("SELECT id, name FROM fingerprints")
```

```
    fingerprints_data = sqlite_cursor.fetchall()

    sqlite_cursor.execute("SELECT fingerprint_id, timestamp,
result FROM logs")
    logs_data = sqlite_cursor.fetchall()

    #  Завантаження в PostgreSQL
    for row in fingerprints_data:
        pg_cursor.execute('''
            INSERT INTO fingerprints (id, name)
            VALUES (%s, %s)
            ON CONFLICT (id) DO UPDATE SET name = EXCLUDED.name
        ''', row)
    for row in logs_data:
        pg_cursor.execute('''
            INSERT INTO logs (fingerprint_id, timestamp, result)
            VALUES (%s, %s, %s)
        ''', row)

    pg_conn.commit()

    #  Закриття підключень
    sqlite_conn.close()
    pg_cursor.close()
    pg_conn.close()

    print(" Дані успішно перенесені ")
```

В разі успішного виконання в терміналі має з'явитися відповідне повідомлення (рис. 4.34) , а у вбудованому журналі подій для БД Render з'явитись нова активність.

Рисунок 4.34 – Підтвердження в терміналі PyCharm

4.4.2. Перенесення серверної частини в Render

Після перенесення бази даних у хмару та адаптації коду сервера під новий

формат БД система була готова до повної міграції на Render [20]. Для цього в інтерфейсі Render було створено вебсервіс, який запропонував перенести проєкт із репозиторію GitHub. Це є основним способом розгортання сервісів на цій платформі. Тому наступним кроком стало завантаження проєкту на GitHub. Це можна було зробити автоматично звичними способами або вручну завантажити лише потрібні файли — обрано було саме другий варіант, оскільки так було зручніше організувати проєкт (рис. 4.35) .

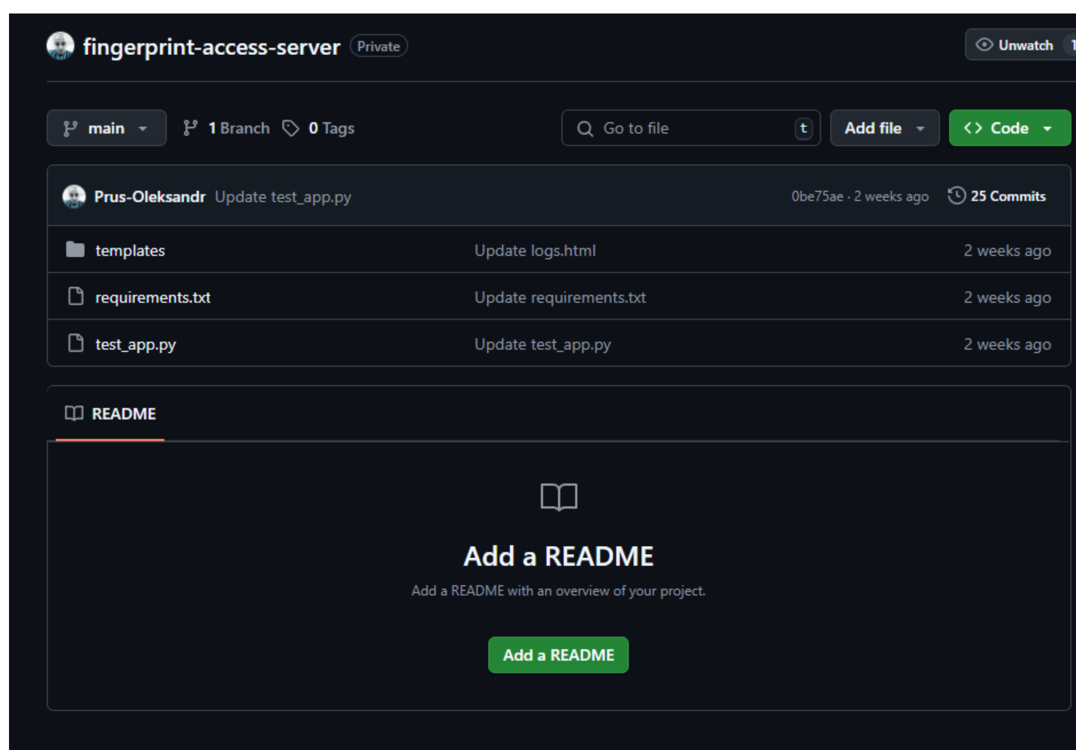


Рисунок 4.35 – Приватний репозиторій fingerprint-access-server

У нашому репозиторії розміщено: test_app.py — скрипт сервера(додаток Б); папку templates, у якій містяться файли login.html (додаток В) та logs.html (додаток Г); а також текстовий файл requirements.txt(додаток Д), у якому зазначено назви додаткових бібліотек, які Render автоматично інтегрує під час розгортання.

Перейдемо до налаштувань сервера (рис. 4.36) .

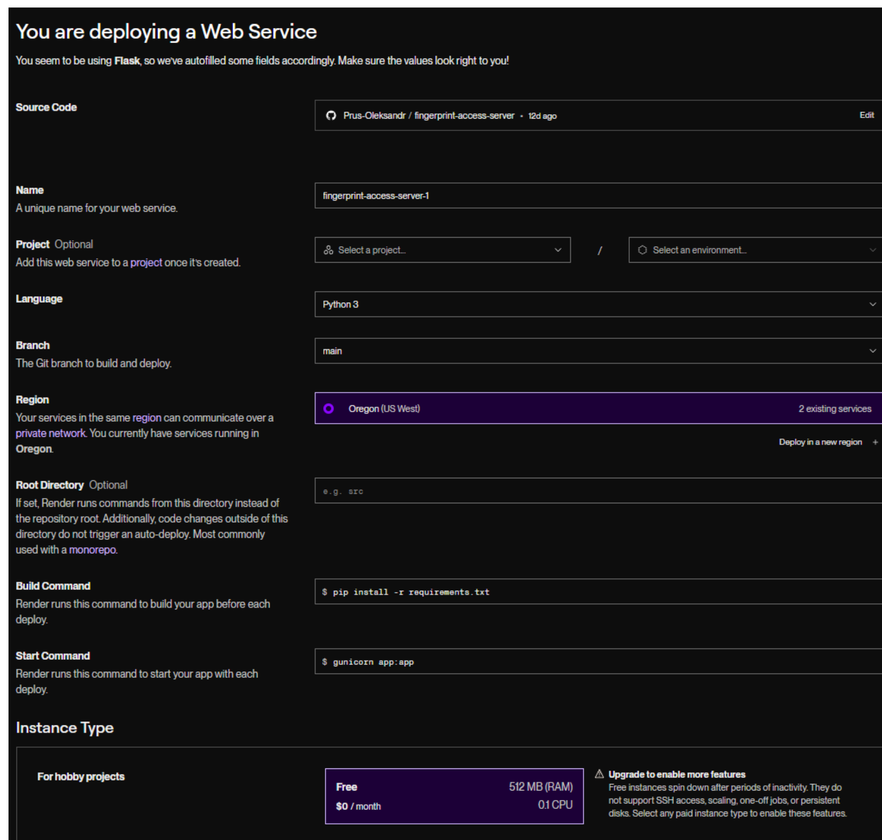


Рисунок 4.36 – Вікно налаштувань сервісу перед завантаженням

В нашому випадку налаштування можна лишити такими як Render визначає автоматично, головне обрати безкоштовний план. Тепер можна розгорнути сервер.

4.5. Тестування роботи системи в реальних умовах

4.5.1. Тестування базової функції зчитування пальця

Після запуску мікроконтролера одразу активується основна функція loop(), яка підключається до мережі через Wi-Fi, перевіряє наявність підключеного сенсора, запитує дозвіл на роботу в сервера, виводить відповідні повідомлення (рис. 4.37) в Serial Monitor та очікує на палець для зчитування з подальшим порівнянням із наявними шаблонами. Крім того, функція обробляє повідомлення формату r<id>, отримані через Serial Monitor, що використовуються для запису нового користувача. Стан сканера додатково сигналізується вбудованим синім

світлодіодом: блиманням (рис. 4.38) або його відсутністю залежно від дозволу на сканування.

```
Connecting to WiFi...
...
WiFi connected
Fingerprint sensor ready.
Scanner status: ENABLED
Place your finger...
Scanner status: ENABLED
Scanner status: ENABLED
```

Рисунок 4.37 – Повідомлення в Serial Monitor при завантаженні та дозволі на сканування

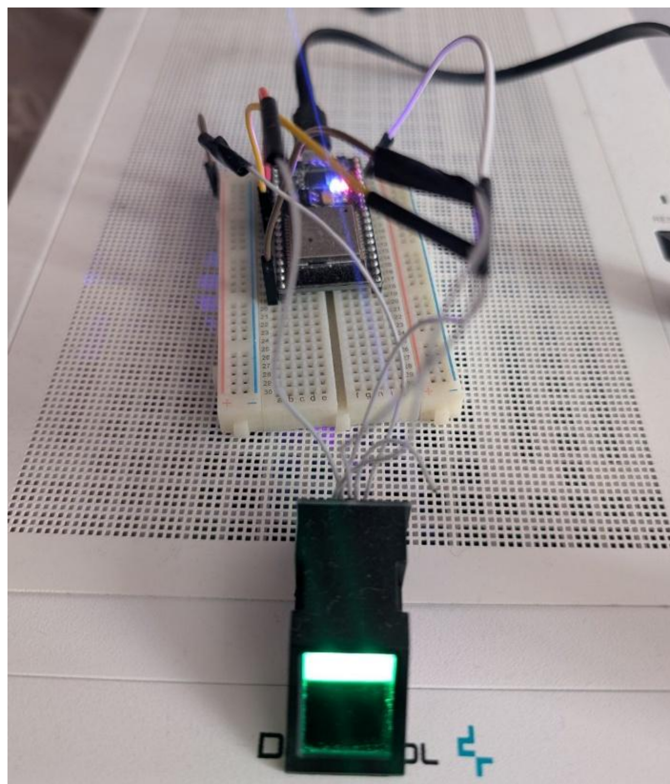


Рисунок 4.38 – Блимання світлодіода в очікуванні пальця

Перед тим як переходити до тестування зчитування для порівняння, спочатку потрібно внести відбитки, щоб система мала з чим здійснювати порівняння. Для цього по черзі виконаємо запис для ID 1, ID 2 та ID 3, надіславши команди r1, r2 та r3 відповідно, і завершимо процес реєстрації (рис. 4.39) .

```
[INTERRUPT] Detected command: r1
Starting enrollment for ID: 1
Place your finger...
Remove finger...
Place same finger again...
Enter name:
Sending enrollment data to server: id=1&name=Панченко
Server response: {"id":1,"name":"\u041f\u0430\u043d\u0447\u0435\u043d\u043a\u043e","status":"success"}
```

Рисунок 4.39 – Вивід повідомлень з інструкціями для зчитування та про надсилання нового запису на сервер та успішне збереження

Тепер перевіримо як працює функція порівняння відбитків просто приставивши палець коли сканер в режимі очікування. Спочатку підставимо 2 пальці, відбитки яких ми тільки що зберегли (рис. 4.40-4.41) і третім підставим незареєстрований палець (рис. 4.42) .

```
Match found! ID: 1
Sending verification data to server: id=1
Server response: {"id":1,"match":"true"}
```

```
Image taken
Match found! ID: 3
Sending verification data to server: id=3
Server response: {"id":3,"match":"true"}
```

Рисунки 4.40-4.41 – Звіти з підтвердженими відбитками

```
Image taken
No match found.
Sending verification data to server: id=0
Server response: {"match":"false"}
```

Рисунок 4.42 – Звіт з заблокованим відбитком

Можемо побачити, що порівняння відбитків працює вірно, а сервер отримує id, та правильно обробляє його в базі даних. Також світлодіод коректно виконує світіння протягом 3 секунд після підтвердження і затухання у разі відмови.

4.5.2. Тестування серверної частини та веб-інтерфейсу

Перевіримо, систему авторизації (рис. 4.43-4.45) , запис подій в сервері (рис. 4.46) і загальну роботу веб-сторінки.

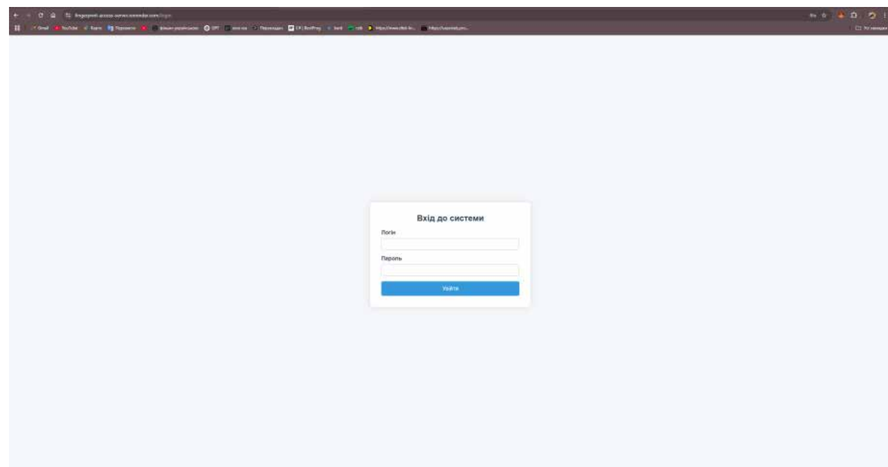


Рисунок 4.43 – Вікно авторизації у веб-застосунку

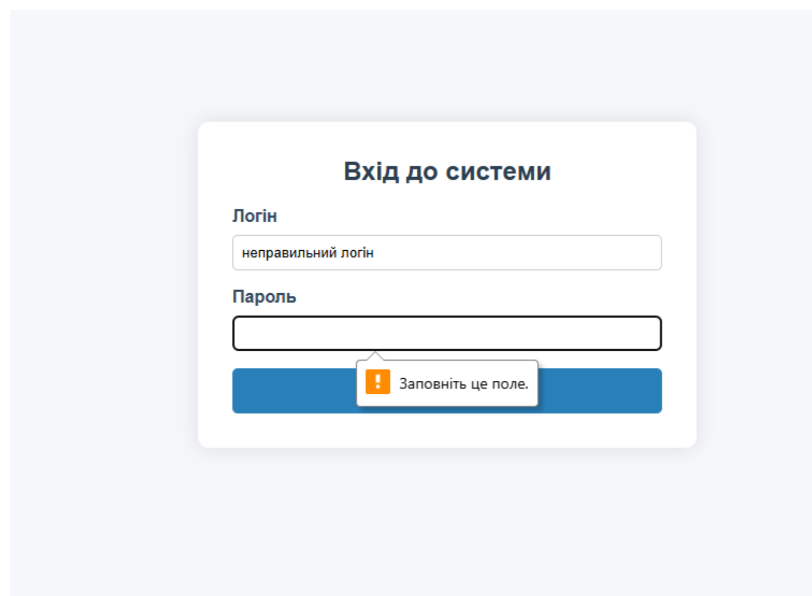


Рисунок 4.44 – Повідомлення про не заповнене поле

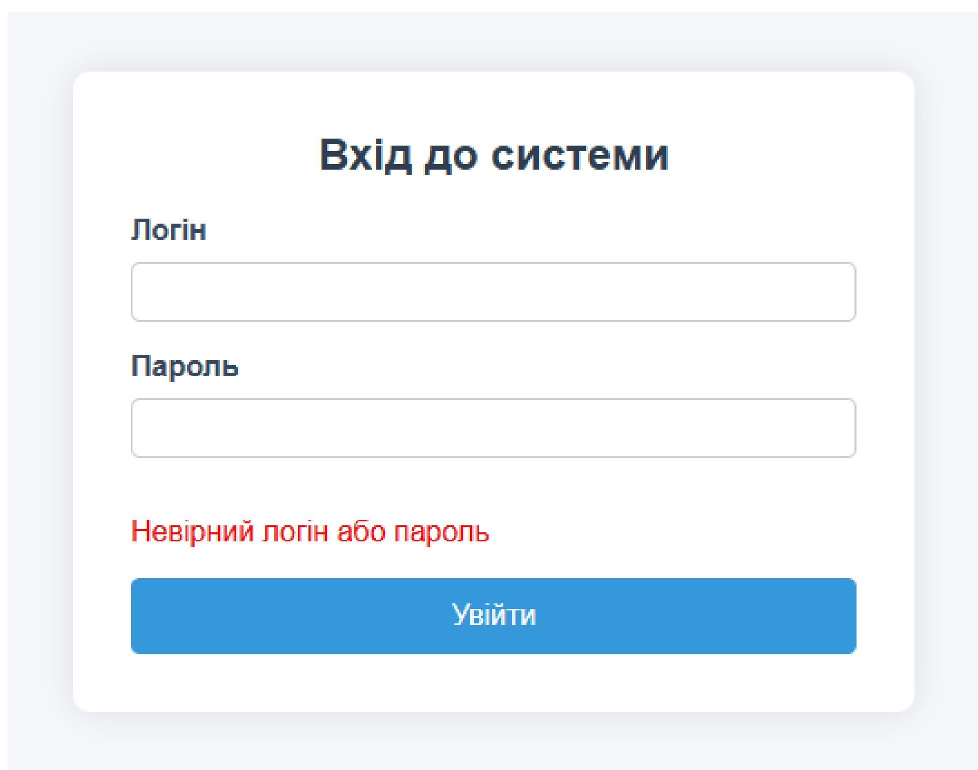


Рисунок 4.45 – Повідомлення про неправильний логін або пароль

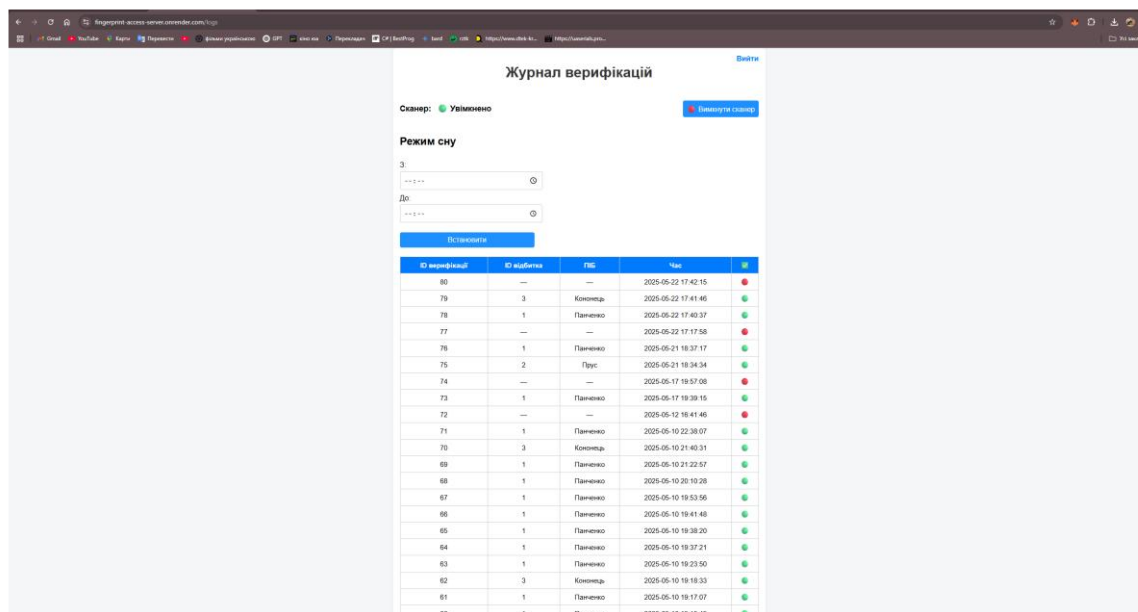


Рисунок 4.46 – Головна сторінка веб-додатку

Можна побачити, що сторінка побудована коректно: присутні всі елементи, які були створені, а останні три записи відображаються правильно.

Наступним кроком перевіримо перезапис прізвища для наявного ID. Для прикладу використаємо ID 1 — при повторній реєстрації відбитка (рис. 4.47) введемо прізвище Шевченко замість Кононець. У разі успішного виконання всі відповідні записи в таблиці мають оновитися (рис. 4.48) .

```
[INTERRUPT] Detected command: r3
Starting enrollment for ID: 3
Place your finger...
Remove finger...
Place same finger again...
Enter name:
Sending enrollment data to server: id=3&name=Шевченко Т.Г.
Server response: {"id":3,"name":"\u0428\u0435\u0432\u0447\u0435\u043d\u043a\u043e \u0422.\u0413.","status":"success"}

Scanner status: ENABLED
Place your finger...
```

Рисунок 4.47 – Процес перезапису з боку ESP32

Встановити

ID верифікації	ID відбитка	ПІБ	Час	
80	—	—	2025-05-22 17:42:15	
79	3	Шевченко Т.Г	2025-05-22 17:41:46	
78	1	Панченко	2025-05-22 17:40:37	
77	—	—	2025-05-22 17:17:58	
76	1	Панченко	2025-05-21 18:37:17	
75	2	Прус	2025-05-21 18:34:34	
74	—	—	2025-05-17 19:57:08	
73	1	Панченко	2025-05-17 19:39:15	
72	—	—	2025-05-12 16:41:46	
71	1	Панченко	2025-05-10 22:38:07	
70	3	Шевченко Т.Г	2025-05-10 21:40:31	
69	1	Панченко	2025-05-10 21:22:57	
68	1	Панченко	2025-05-10 20:10:28	
67	1	Панченко	2025-05-10 19:53:56	
66	1	Панченко	2025-05-10 19:41:48	
65	1	Панченко	2025-05-10 19:38:20	
64	1	Панченко	2025-05-10 19:37:21	
63	1	Панченко	2025-05-10 19:23:50	
62	3	Шевченко Т.Г	2025-05-10 19:18:33	
61	1	Панченко	2025-05-10 19:17:07	
60	1	Панченко	2025-05-10 19:15:49	
48	3	Шевченко Т.Г	2025-05-10 16:16:40	

Рисунок 4.48 – Результат перезапису на сервері

Тепер перевіримо ручне вимкнення та увімкнення сканера через сайт (рис. 4.49-4.51) .

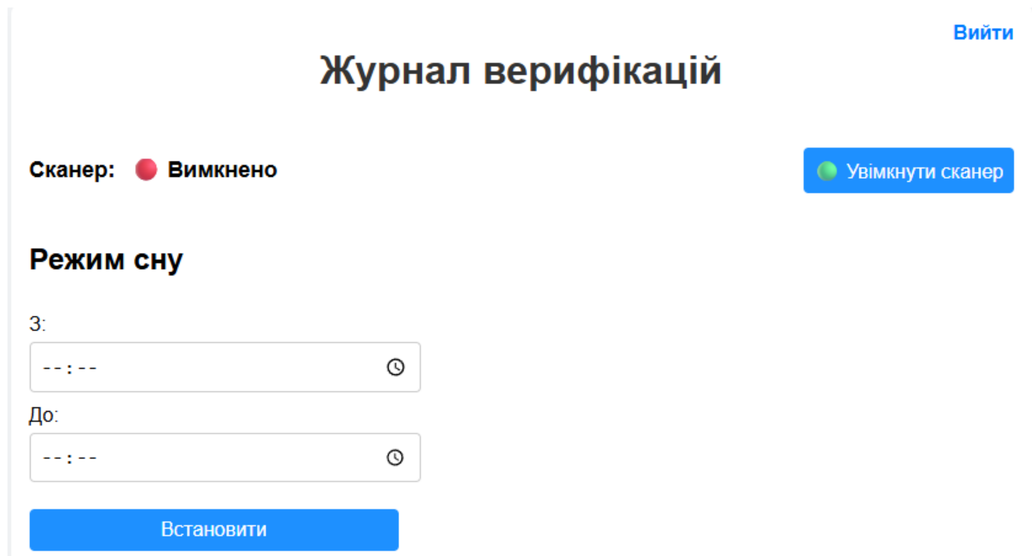


Рисунок 4.49 – Стан сканера на сайті після вимкнення натисканням на кнопку

```
Scanner status: ENABLED
Scanner status: ENABLED
Scanner status: ENABLED
Scanner status: ENABLED
Scanner status: DISABLED
Scanner status: DISABLED
Scanner status: DISABLED
```

Рисунок 4.50 – Результат в Serial Monitor

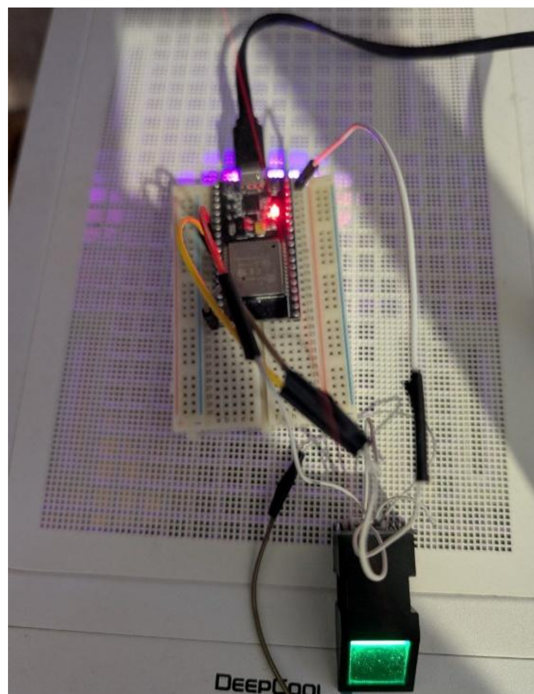


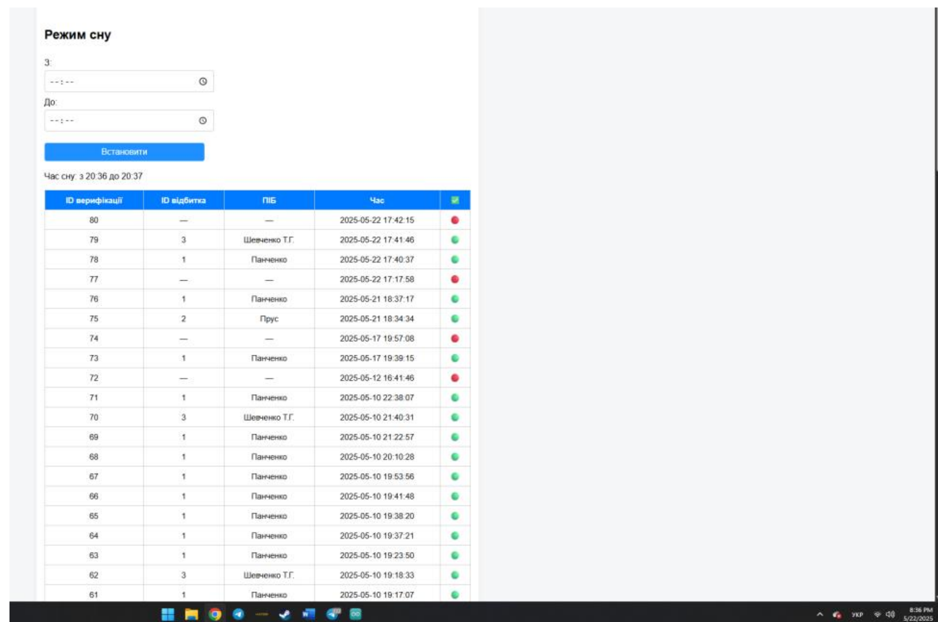
Рисунок 4.51 – Результат на платі – перестав блимати світлодіод

Як і планувалося, функція зчитування пальця не запускається поки немає дозволу з сервера. Тепер спробуємо аналогічно увімкнути сканер (рис. 4.52) .

```
Scanner status: DISABLED
Scanner status: DISABLED
Scanner status: ENABLED
Place your finger...
Scanner status: ENABLED
```

Рисунок 4.52 – Результат успішного увімкнення

Тепер протестуємо режим сну (рис. 4.53-4.54) , введемо невеликий проміжок, щоб зручно відслідкувати правильність виконання.



ID верифікації	ID відбитка	ПІБ	Час	✅
80	—	—	2025-05-22 17:42:15	🔴
79	3	Шевченко Т.Г.	2025-05-22 17:41:46	🟢
78	1	Панченко	2025-05-22 17:40:37	🟢
77	—	—	2025-05-22 17:17:58	🔴
76	1	Панченко	2025-05-21 18:37:17	🟢
75	2	Прус	2025-05-21 18:34:34	🟢
74	—	—	2025-05-17 19:57:08	🔴
73	1	Панченко	2025-05-17 19:39:15	🟢
72	—	—	2025-05-12 16:41:46	🔴
71	1	Панченко	2025-05-10 22:38:07	🟢
70	3	Шевченко Т.Г.	2025-05-10 21:40:31	🟢
69	1	Панченко	2025-05-10 21:22:57	🟢
68	1	Панченко	2025-05-10 20:10:28	🟢
67	1	Панченко	2025-05-10 19:53:56	🟢
66	1	Панченко	2025-05-10 19:41:48	🟢
65	1	Панченко	2025-05-10 19:38:20	🟢
64	1	Панченко	2025-05-10 19:37:21	🟢
63	1	Панченко	2025-05-10 19:23:50	🟢
62	3	Шевченко Т.Г.	2025-05-10 19:18:33	🟢
61	1	Панченко	2025-05-10 19:17:07	🟢

Рисунок 4.53 – Встановлений час близький до теперішнього

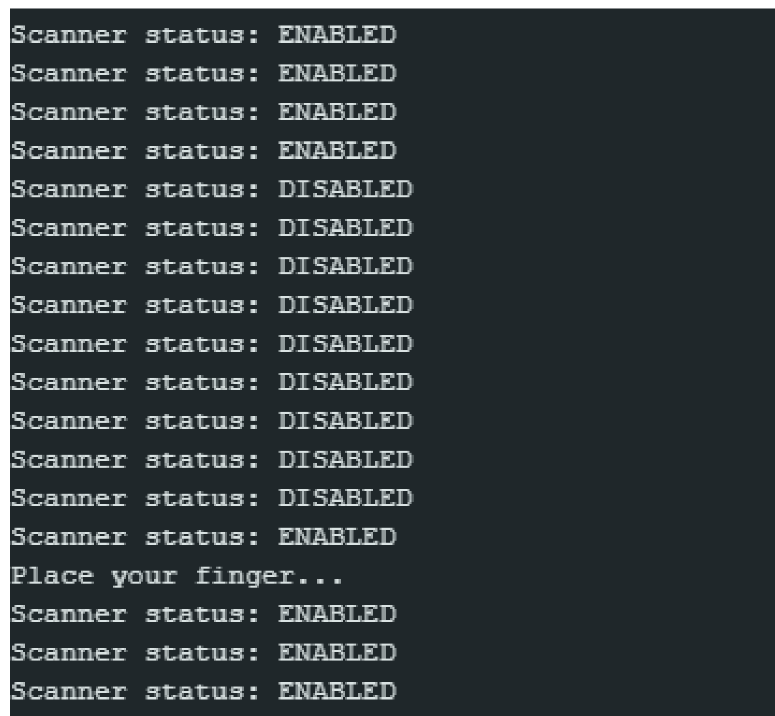


Рисунок 4.54 – Результати автоматичного вимкнення та увімкнення

Можна чітко побачити де почався режим сну і де скінчився.

Тепер перевіримо чи працює переривання режиму сну ручним увімкненням (рис. 4.55-4.56) .

Вийти

Журнал верифікацій

Сканер: Вимкнено

Увімкнути сканер

Режим сну

З:

--:--

До:

--:--

Встановити

Час сну: з 08:12 до 08:52

Рисунок 4.55 – Режим сну з великим проміжком для більш зручного відстежування збивання режиму

Вийти

Журнал верифікацій

Сканер: Увімкнено

Вимкнути сканер

Режим сну

З:

--:--

До:

--:--

Встановити

Час сну: з 08:12 до 08:52

Рисунок 4.56 – Перерваний режим сну

При натискання ручного увімкнення режим сну переривається. Крім того встановлення нового режиму сну відбувається коректно та перериває попередні ручні налаштування.

Перевіримо функції виходу до етапу авторизації (рис. 4.57-4.58) .

Вийти

Рисунок 4.57 – Кнопка виходу зі сторінки журналу подій назад до сторінки авторизації

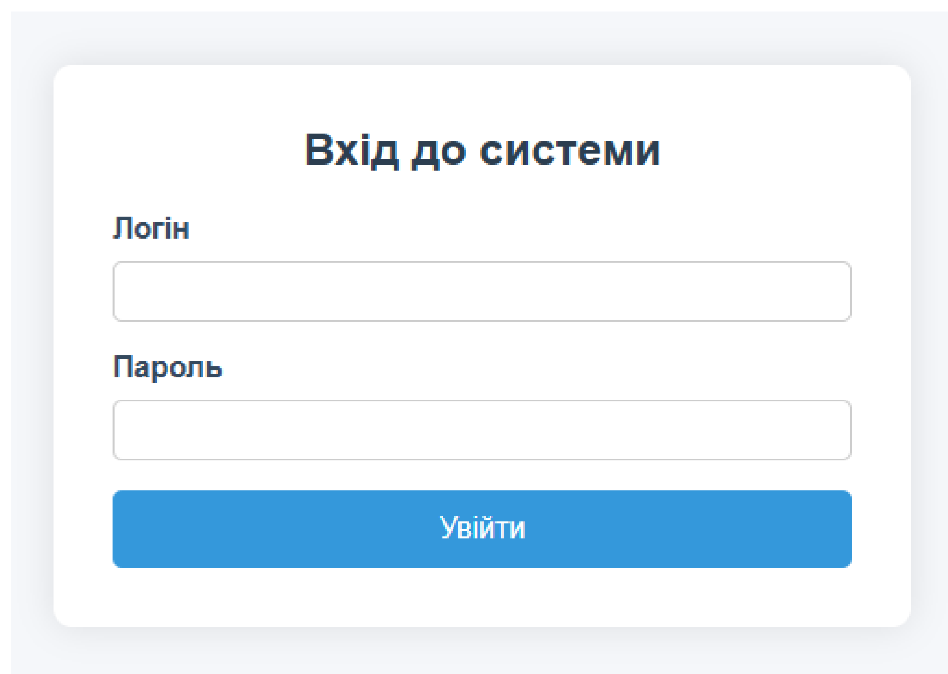


Рисунок 4.58 – Успішне повернення на сторінку авторизації

Ця та решта функцій спрацювали добре в різних сценаріях взаємодії.

4.5.3. Тестування веб-інтерфейсу на мобільному пристрої

Перевіримо чи правильно відображаються та функціонують сторінки авторизації (рис. 4.59) та журналу подій (рис. 4.60) .

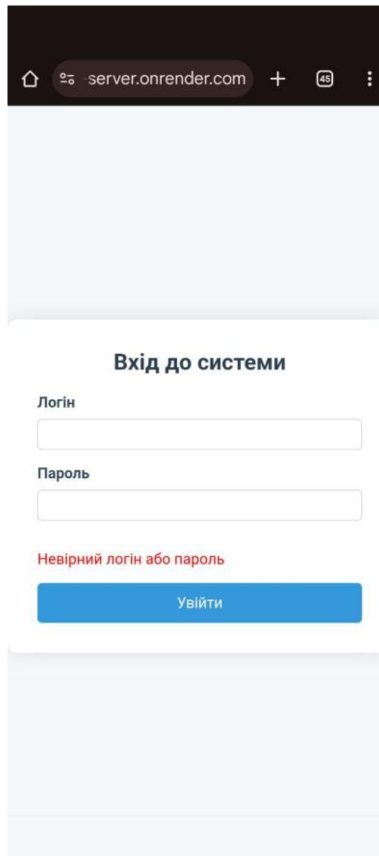


Рисунок 4.59 – Сторінка авторизації на мобільному пристрої

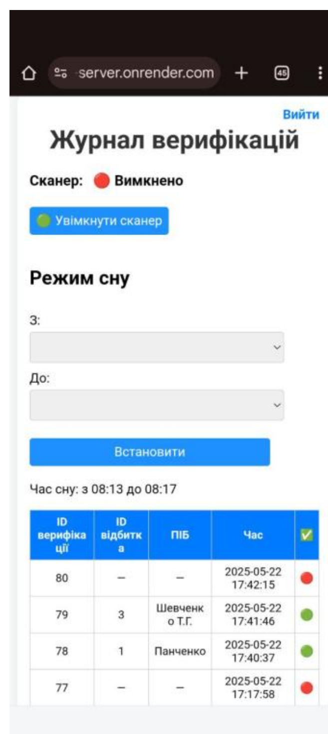
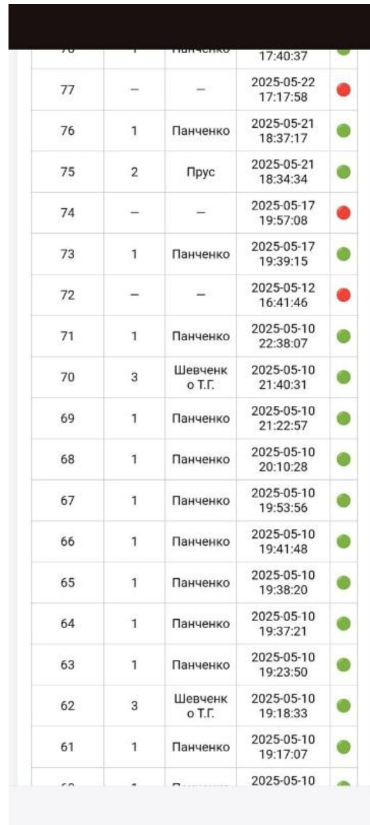


Рисунок 4.60 – Сторінка журналу подій на мобільному пристрої

Можна побачити, що наша система добре адаптована для перегляду на екранах мобільних пристроїв: усі функціональні елементи зручно розміщені на сторінці. Таблицю можна комфортно переглядати навіть у вертикальному форматі, а такі можливості, як прокручування (рис.4.61) та масштабування жестами, лише покращують зручність користування.



77	–	–	2025-05-22 17:17:58
76	1	Панченко	2025-05-21 18:37:17
75	2	Прус	2025-05-21 18:34:34
74	–	–	2025-05-17 19:57:08
73	1	Панченко	2025-05-17 19:39:15
72	–	–	2025-05-12 16:41:46
71	1	Панченко	2025-05-10 22:38:07
70	3	Шевченко Т.Г.	2025-05-10 21:40:31
69	1	Панченко	2025-05-10 21:22:57
68	1	Панченко	2025-05-10 20:10:28
67	1	Панченко	2025-05-10 19:53:56
66	1	Панченко	2025-05-10 19:41:48
65	1	Панченко	2025-05-10 19:38:20
64	1	Панченко	2025-05-10 19:37:21
63	1	Панченко	2025-05-10 19:23:50
62	3	Шевченко Т.Г.	2025-05-10 19:18:33
61	1	Панченко	2025-05-10 19:17:07

Рисунок 4.61 – Гортання таблиці на мобільному пристрої

Тепер перевіримо, як сторінка адаптується до горизонтального режиму перегляду на мобільному пристрої (рис. 4.62) , та чи зберігається зручність використання інтерфейсу.

Рисунок 4.62 – Відображення сторінки у горизонтальному режимі

В такому варіанті варіанті використання інтерфейс стає трохи менш зручним, адже не завжди усі необхідні елементи можна побачити на екрані, проте відображення самої таблиці (рис. 4.63) стало ще зручнішим.

80	–	–	2025-05-22 17:42:15
79	3	Шевченко Т.Г.	2025-05-22 17:41:46
78	1	Панченко	2025-05-22 17:40:37
77	–	–	2025-05-22 17:17:58
76	1	Панченко	2025-05-21 18:37:17
75	2	Прус	2025-05-21 18:34:34
74	–	–	2025-05-17 19:57:08
73	1	Панченко	2025-05-17 19:39:15
72	–	–	2025-05-12 16:41:46
71	1	Панченко	2025-05-10 22:38:07
70	3	Шевченко Т.Г.	2025-05-10 21:40:31

Рисунок 4.63 – Таблиця на мобільному девайсі у горизонтальному режимі роботи

Підсумовуючи, веб-додаток чудово виглядає та функціонує на мобільних пристроях.

4.6. Аналіз ефективності, продуктивності та безпеки системи

4.6.1. Аналіз швидкодії системи

Перевіримо швидкодію рзних процесів у системі:

– Зчитування та порівняння відбитка пальця – близько 1 секунди;
– Підтвердження на сервері та запис у таблицю – близько 7 секунд;
– Реакція сканера на ручне вимкнення – близько 1 секунди;
– Реакція сканера на ручне увімкнення – близько 3 секунд;
– Перезапис користувача на сервері – близько 2 секунд.

Система швидко порівнює відбитки та надає доступ, або в ньому відмовляє, отримуючи незначну затримку вже на етапі логування події.

Решта функцій теж мають задовільний результат по часу виконання.

4.6.2. Аналіз можливих обмежень системи

Пройдемося по основних етапах та розглянемо, які ключові обмеження має система.

На рівні сенсора маємо обмеження, пов'язані з обсягом його вбудованої пам'яті, у якій зберігаються шаблони відбитків. Обсяг пам'яті сенсора дозволяє зберігати приблизно 160 шаблонів, що для більшості випадків є більш ніж достатнім.

Також через особливості архітектури сенсора FPM10A виникає обмеження щодо опрацювання шаблонів відбитків за межами самого сенсора, наприклад, на контролері. Таке обмеження, хоча й не дає можливості додатково обробляти відбитки, позитивно впливає на безпеку системи.

З боку серверної частини обмеження пов'язані з особливостями тарифного плану, наданого хмарним сервісом Render. Було використано безкоштовний план, який передбачає мінімальні обчислювальні ресурси та обмежений обсяг сховища для бази даних.

Render надає 1 ГБ пам'яті, якого, зважаючи на просту структуру нашої системи, вистачає приблизно на мільйон записів — цього більш ніж достатньо для її стабільної роботи. Водночас Render обмежує тривалість активної сесії бази даних, тому з часом може виникати потреба її перезапуску або переходу на

платний тариф.

Наявних обчислювальних ресурсів достатньо для одночасного обслуговування кількох десятків підключених користувачів.

4.6.3. Аналіз безпеки системи

Розглянемо безпекову складову на різних етапах функціонування системи.

Як уже було зазначено, процес порівняння та збереження шаблонів відбитків є закритим у сенсорі FPM10A, тому отримати доступ до шаблонів ззовні неможливо. Таке обмеження апаратного рівня забезпечує надійний захист на одному з найважливіших етапів — збереженні біометричних даних.

Більш вразливою ланкою є ESP32, для якого не реалізовано додаткових механізмів захисту на рівні з’єднання. Проте передбачається, що в умовах реального використання мікроконтролер і сенсор будуть розміщені в захищеному корпусі, доступ до якого можливий лише фізично — наприклад, після відкриття корпусу або з іншого боку замкнених дверей. Конкретна реалізація залежить від сценарію використання.

Передача даних від мікроконтролера до сервера здійснюється через HTTPS, який базується на протоколі TLS, що забезпечує базовий, але достатній рівень захисту для передавання такого типу інформації. У даному випадку передаються лише ID користувача та ПІБ, які не є критичними з точки зору безпеки.

На стороні сервера реалізовано захист у вигляді простої авторизації з логіном і паролем. Облікові дані задаються безпосередньо в коді сервера і не можуть бути змінені через інтерфейс.

Сервіс Render, який використовується для розміщення серверної частини, гарантує базовий рівень безпеки, зокрема підтримку TLS 1.2/1.3, захист від DDoS-атак, ізоляцію середовищ, а також додаткові параметри захисту залежно від обраного тарифного плану.

Підсумовуючи, варто зазначити, що головним безпековим рішенням у побудові системи є вибір сенсора з повністю закритою системою зберігання шаблонів та розподіл обробки даних. Найважливіші дані зберігаються локально на сенсорі, до якого немає доступу, а менш критичні — наприклад, журнали подій — передаються й зберігаються у хмарному середовищі, яке, своєю чергою, також має багаторівневий захист, хоч і не є обов’язковим для цього типу інформації.

ВИСНОВКИ

У межах даної дипломної роботи було розроблено комп'ютерну систему моніторингу та контролю доступу з використанням IoT-пристроїв, що базується на мікроконтролері ESP32 та біометричному сенсорі відбитків пальців FPM10A. Система дозволяє локально виконувати верифікацію користувача, а також здійснювати централізоване керування доступом через веб-інтерфейс адміністратора, розгорнутий у хмарному середовищі. Проведене тестування підтвердило надійність функціонування системи, швидку реакцію на дії користувача, а також зручність її використання як з комп'ютера, так і з мобільних пристроїв. Інтерфейс спрощений, інтуїтивно зрозумілий, а основні функції працюють стабільно та без помітних затримок.

Усі поставлені на початку роботи цілі були досягнуті. Реалізовано повний цикл функціонування: зчитування біометричних даних, локальне порівняння шаблонів, логування результатів на сервері, а також дистанційне керування режимом сканера та часовим графіком його роботи. Результат повністю відповідає попередньо визначеним функціональним та нефункціональним вимогам, а якість і стабільність роботи системи перевищила очікування. Розроблена архітектура виявилась ефективною, гнучкою та придатною до масштабування, що свідчить про доцільність обраних підходів і технічних рішень.

Незважаючи на те, що поточний функціонал системи вже є повністю працездатним і задовольняє основні потреби, існує кілька напрямків її подальшого вдосконалення. На рівні серверної частини доцільно реалізувати додаткове логування — наприклад, фіксацію спроб повторного додавання або перезапису користувача.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Lynn T., Endo P. T., Ribeiro A. M. N. C., Barbosa G. B. N., Rosati P. (2020). The Internet of Things: Definitions, Key Concepts, and Reference Architectures. The Cloud-to-Thing Continuum, 1–22. Springer. [Електронний ресурс]. – Режим доступу: https://link.springer.com/chapter/10.1007/978-3-030-41110-7_1 (дата звернення: 12.03.2025).

2. Jain A. K., Kumar A. (2012). Biometric Recognition: An Overview. In: Mordini, E., Tzovaras, D. (Eds.). Second Generation Biometrics: The Ethical, Legal and Social Context. – Dordrecht: Springer, pp. 49–79. – (The International Library of Ethics, Law and Technology, vol. 11). – Режим доступу: https://doi.org/10.1007/978-94-007-3892-8_3 (дата звернення: 26.03.2025).

3. Salim A., Al-Khafajiy, M., Baker, T., et al. (2020). IoT-based Smart Access Control Systems: Architecture and Implementation. Sensors, 20(17), 4806. – Режим доступу: https://doi.org/10.3390/s20174806 (дата звернення: 29.03.2025).

4. Pallets Projects. (2025). Quickstart – Flask Documentation (3.1.x) [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/en/latest/quickstart/ (дата звернення: 29.03.2025).

5. PostgreSQL Global Development Group. (2025). What Is PostgreSQL? [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs/current/index.html (дата звернення: 29.03.2025).

6. MDN Web Docs. (2025). An overview of HTTP [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview (дата звернення: 01.04.2025).

7. Ahsan M. S., Pathan A.-S. K. (2025). A Comprehensive Survey on the Requirements, Applications, and Future Challenges for Access Control Models in IoT: The State of the Art. IoT, 6(1), 9. [Електронний ресурс]. – Режим доступу: https://doi.org/10.3390/iot6010009 (дата звернення: 01.04.2025).

8. SNS Insider. (2025). Electronic Access Control Systems Market to Grow

USD 106.68 Billion by 2032. [Електронний ресурс]. – Режим доступу: https://www.globenewswire.com/news-release/2025/03/26/3049750/0/en/Electronic-Access-Control-Systems-Market-to-Grow-USD-106-68-Billion-by-2032-Fueled-by-increasing-demand-for-advanced-security-solutions-and-innovation-SNS-Insider.html (дата звернення: 04.04.2025).

9. OptConnect. (2023). Elevating Access Control with IoT. [Електронний ресурс]. – Режим доступу: https://optconnect.com/elevating-access-control-with-iot/ (дата звернення: 04.04.2025).

10. Access Professional Systems. (2024). The Future of Access Technology. [Електронний ресурс]. – Режим доступу: https://accessprofessionals.com/the-future-of-access-technology/ (дата звернення: 05.04.2025).

11. Sharma A. (2024). A Comparison Between Various Wireless Technologies: UWB, NFC, Wi-Fi, Bluetooth, BLE, GPS. ENC Store Blog. [Електронний ресурс]. – Режим доступу: https://www.encstore.com/blog/5774-a-comparison-between-various-wireless-technologies-uwb-nfc-wi-fi-bluetooth-ble-gps (дата звернення: 10.04.2025).

12. Benantar M. Access Control Systems: Security, Identity Management and Trust Models. – Berlin: Springer, 2006. – 312 p.

13. Andriulo F. C., Fiore M., Mongiello M., Traversa E., Zizzo V. Edge Computing and Cloud Computing for Internet of Things: A Review // Informatics. – 2024. – Vol. 11, No. 4. – Article 71. – [Електронний ресурс]. – Режим доступу: https://doi.org/10.3390/informatics11040071 (дата звернення: 13.04.2025)

14. Nguyen K., Laurent-Maknavicius M., Laurent M., Oualha N. Survey on secure communication protocols for the Internet of Things // Ad Hoc Networks. – 2015. – Vol. 32. – P. 17–31. – [Електронний ресурс]. – Режим доступу: https://doi.org/10.1016/j.adhoc.2015.01.006 (дата звернення: 17.04.2025).

15. Espressif Systems. ESP32-WROOM-32 Datasheet. – [Електронний ресурс]. – Режим доступу: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-

32_datasheet_en.pdf (дата звернення: 21.04.2025).

16. Adafruit Industries. Adafruit Optical Fingerprint Sensor Guide. – [Електронний ресурс]. – Режим доступу: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-optical-fingerprint-sensor.pdf (дата звернення: 24.04.2025).

17. Espressif Systems. UART Driver — API Reference (ESP-IDF Documentation). – [Електронний ресурс]. – Режим доступу: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/uart.html (дата звернення: 28.04.2025).

18. SQLite. About SQLite [Електронний ресурс]. – Режим доступу: https://www.sqlite.org/about.html – (дата звернення: 30.04.2025).

19. Render. Create and Connect to Render Postgres [Електронний ресурс]. – Режим доступу: https://render.com/docs/postgresql-creating-connecting (дата звернення: 06.05.2025).

20. Render. Deploy a Flask Application on Render [Електронний ресурс]. – Режим доступу: https://render.com/docs/deploy-flask (дата звернення: 9.05.2025).

21. Прус О.Б., Місюра М.Д. Розробка комп'ютерної системи моніторингу та контролю доступу з використанням IoT-пристроїв [Електронний ресурс]. – Режим доступу: http://econference.nubip.edu.ua/index.php/taacsd/2025/paper/view/3685 (дата звернення: 28.05.2025).

Додатки

ДОДАТОК А

```
#include <WiFi.h> // Бібліотека для підключення ESP32 до Wi-Fi-мережі
#include <HTTPClient.h> // Дозволяє здійснювати HTTP(S)-запити до серверу
#include <Adafruit_Fingerprint.h> // Бібліотека для роботи із сенсором відбитків пальців
#include <HardwareSerial.h> // Дозволяє використовувати апаратні UART-порти ESP32

#define RXD2 16 // Пін ESP32, до якого підключено TX сенсора (використовується як RX для UART1)
#define TXD2 17 // Пін ESP32, що відповідає за передачу даних до RX сенсора (TX для UART1)
#define LED_BUILTIN 2 // Вбудований світлодіод на платі ESP32 (GPIO2), використовується для індикації стану

HardwareSerial mySerial(1); // Створення об'єкта для апаратного UART1
Adafruit_Fingerprint finger(&mySerial); // Створення об'єкта для роботи з сенсором FPM10A через UART1

// Wi-Fi дані
const char* ssid = "****"; // Назва мережі
const char* password = "****"; // Пароль до мережі

// URL-и сервера
const char* verifyUrl = "https://fingerprint-access-server.onrender.com/verify"; // Адреса для перевірки ID
const char* enrollUrl = "https://fingerprint-access-server.onrender.com/enroll"; // Адреса для реєстрації користувача
const char* statusUrl = "https://fingerprint-access-server.onrender.com/scanner_status"; // Адреса для перевірки статусу сканера

bool isVerifying = true; // Активність режиму ідентифікації (true — сканування ввімкнене)
bool scannerEnabled = true; // Прапорець, який оновлюється з сервера (чи дозволено сканування)
unsigned long lastStatusCheck = 0; // Час останньої перевірки статусу сканера для уникнення зайвих запитів
String pendingCommand = ""; // Буфер для зберігання команди, отриманої через Serial Monitor

void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
  delay(1000);

  Serial.println("Connecting to WiFi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
```

```
        mySerial.begin(57600, SERIAL_8N1, RXD2, TXD2);
        finger.begin(57600); // Ініціалізація UART-зв'язку з сенсором
через UART1

        if (finger.verifyPassword()) {
          Serial.println("Fingerprint sensor ready.");
        } else {
          Serial.println("Sensor not detected.");
          while (true) delay(1); // зупинка програми, якщо сенсор не
знайдено
        }
      }

      void loop() {
        // Обробка команди на реєстрацію нового користувача
        if (pendingCommand.startsWith("r")) {
          int id = pendingCommand.substring(1).toInt();
          if (id > 0) {
            isVerifying = false;
            Serial.print("Starting enrollment for ID: ");
            Serial.println(id);

            if (getFingerprintEnroll(id)) {
              Serial.println("Enter name: ");
              while (!Serial.available());
              String name = Serial.readStringUntil('\n');
              name.trim();
              sendEnrollToServer(id, name); // Надсилання даних на сервер
            } else {
              Serial.println("Enrollment failed.");
            }
            isVerifying = true;
          }
          pendingCommand = "";
          return;
        }

        // Обробка будь-якої команди із серійного монітора
        if (Serial.available()) {
          pendingCommand = Serial.readStringUntil('\n');
          pendingCommand.trim();
          return;
        }

        checkScannerStatus(); // Запит до сервера, чи дозволено
сканування
        if (!scannerEnabled) {
          digitalWrite(LED_BUILTIN, LOW);
          delay(1000);
          return;
        }

        // Основна логіка ідентифікації
        if (isVerifying) {
          blinkWaiting(); // Миготіння LED у режимі очікування
```

```
        Serial.println("Place your finger...");
        while (finger.getImage() != FINGERPRINT_OK) {
          if (Serial.available()) {
            pendingCommand = Serial.readStringUntil('\n');
            pendingCommand.trim();
            Serial.println("[INTERRUPT] Detected command: " +
pendingCommand);
            return;
          }
          checkScannerStatus();
          if (!scannerEnabled) return;
          blinkWaiting();
          delay(100);
        }

        Serial.println("Image taken");
        digitalWrite(LED_BUILTIN, LOW);
        delay(200);

        if (finger.image2Tz(1) != FINGERPRINT_OK) return; //
Перетворення зображення у шаблон
        int result = finger.fingerSearch(); // Пошук збігу
        int idToSend = 0;

        if (result == FINGERPRINT_OK) {
          idToSend = finger.fingerID;
          Serial.print("Match found! ID: ");
          Serial.println(idToSend);
          flashLED(3); // Триразове блимання при збігу
          digitalWrite(LED_BUILTIN, HIGH);
          delay(10000);
          digitalWrite(LED_BUILTIN, LOW);
        } else {
          Serial.println("No match found.");
        }

        sendIdToServer(idToSend); // Надсилання результату на сервер
        delay(5000);
      }
    }

    bool getFingerprintEnroll(int id) {
      Serial.println("Place your finger...");
      while (finger.getImage() != FINGERPRINT_OK) {
        if (Serial.available()) return false; // Дозволити перервати
команду
        delay(100);
      }
      if (finger.image2Tz(1) != FINGERPRINT_OK) return false; //
Конвертація першого зображення

      Serial.println("Remove finger...");
      delay(2000);
      while (finger.getImage() != FINGERPRINT_NOFINGER) {
        if (Serial.available()) return false;
        delay(100);
```

```
    }

    Serial.println("Place same finger again...");
    while (finger.getImage() != FINGERPRINT_OK) {
      if (Serial.available()) return false;
      delay(100);
    }
    if (finger.image2Tz(2) != FINGERPRINT_OK) return false; //
Конвертація другого зображення
    if (finger.createModel() != FINGERPRINT_OK) return false; //
Створення моделі
    if (finger.storeModel(id) != FINGERPRINT_OK) return false; //
Збереження моделі

    return true; // Успішна реєстрація
  }

  void checkScannerStatus() {
    // Обмеження частоти запитів: максимум 1 раз на 3 секунди
    if (millis() - lastStatusCheck < 3000) return;
    lastStatusCheck = millis();

    HTTPClient http;
    http.begin(statusUrl); // ініціалізація GET-запиту
    int code = http.GET(); // Надсилання запиту

    if (code > 0) {
      String res = http.getString(); // Отримання відповіді
      scannerEnabled = (res == "1"); // Оновлення статусу
      Serial.print("Scanner status: ");
      Serial.println(scannerEnabled ? "ENABLED" : "DISABLED");
    } else {
      Serial.println("Failed to check scanner status.");
    }
    http.end(); // Закриття з'єднання
  }

  void sendIdToServer(int id) {
    HTTPClient http;
    http.begin(verifyUrl); // Встановлення з'єднання з сервером
    http.addHeader("Content-Type", "application/x-www-form-
urlencoded"); // Формат даних

    String postData = "id=" + String(id);
    Serial.println("Sending verification data to server: " +
postData);

    int httpResponseCode = http.POST(postData); // Надсилання POST-
запиту
    if (httpResponseCode > 0) {
      Serial.println("Server response: " + http.getString());
    } else {
      Serial.print("HTTP Error: ");
      Serial.println(http.errorToString(httpResponseCode));
    }
    http.end(); // Завершення HTTP-з'єднання
```

```
}

void sendEnrollToServer(int id, String name) {
  HTTPClient http;
  http.begin(enrollUrl); // З'єднання з URL реєстрації
  http.addHeader("Content-Type", "application/x-www-form-
urlencoded");

  String postData = "id=" + String(id) + "&name=" + name;
  Serial.println("Sending enrollment data to server: " + postData);

  int httpResponseCode = http.POST(postData); // POST-запит з ID та
іменем
  if (httpResponseCode > 0) {
    Serial.println("Server response: " + http.getString());
  } else {
    Serial.print("HTTP Error: ");
    Serial.println(http.errorToString(httpResponseCode));
  }
  http.end(); // Завершення HTTP-з'єднання
}

void blinkWaiting() {
  static unsigned long lastBlink = 0;
  static bool ledState = false;

  // Перевіряємо, чи минуло 500 мс з останнього миготіння
  if (millis() - lastBlink >= 500) {
    ledState = !ledState;
    digitalWrite(LED_BUILTIN, ledState); // Перемикання стану
світлодіода
    lastBlink = millis();
  }
}

void flashLED(int times) {
  // Цикл блимання світлодіода задану кількість разів
  for (int i = 0; i < times; i++) {
    digitalWrite(LED_BUILTIN, HIGH); // Увімкнення світлодіода
    delay(300);
    digitalWrite(LED_BUILTIN, LOW); // Вимкнення світлодіода
    delay(300);
  }
}
```

ДОДАТОК Б

```
from flask import Flask, request, jsonify, render_template, session, redirect, url_for  # Flask — основа веб-сервера. request — обробка запитів. jsonify — JSON-відповіді. session — авторизація.
import psycopg2  # Бібліотека для з’єднання з PostgreSQL
from pytz import timezone  # Бібліотека для роботи з часовими поясами
from datetime import datetime, time as dtime, timedelta  # Модуль для роботи з часом і датами
from functools import wraps  # Декоратор для обгортки функцій авторизації

app = Flask(__name__)  # Створення веб-додатку на Flask
app.secret_key = '****'  # Секретний ключ для сесій користувача
DATABASE_URL = "postgresql://..."  # URL з’єднання з PostgreSQL-базою
ADMIN_LOGIN = '***'  # Облікові дані адміністратора
ADMIN_PASSWORD = '***'

kyiv = timezone('Europe/Kyiv')  # Змінна, що відповідає за місцевий час

manual_override_enabled = True  # Ручний дозвіл увімкнення/вимкнення сканера
sleep_start = None
sleep_end = None
sleep_override_until = None  # Змінні для задання режиму "сну"

def get_db_connection():  # Повертає нове з’єднання з базою PostgreSQL
    return psycopg2.connect(DATABASE_URL)

def log_activity(fingerprint_id, result):
    conn = get_db_connection()
    cursor = conn.cursor()
    timestamp = datetime.now(kyiv).strftime('%Y-%m-%d %H:%M:%S')  # Отримання поточного часу для логування
    cursor.execute('''
        INSERT INTO logs (fingerprint_id, timestamp, result)
        VALUES (%s, %s, %s)
    ''', (fingerprint_id if fingerprint_id != 0 else None, timestamp, result))  # Формування запиту до БД, з перевіркою id = 0 → None
    conn.commit()  # Збереження змін
    conn.close()  # Закриття з’єднання з базою

def is_scanner_enabled():  # Функція для реалізації дистанційного контролю доступу до сканера
    global manual_override_enabled, sleep_override_until
    now = datetime.now(kyiv)  # Поточний час у часовій зоні Європа/Київ
    t = now.time()  # Отримуємо лише час (без дати)

    if sleep_start and sleep_end:
        in_sleep = False
        if sleep_start < sleep_end:
```

```
                in_sleep = sleep_start <= t <= sleep_end  # Перевірка в
межах однієї доби
            else:
                in_sleep = t >= sleep_start or t <= sleep_end  #
Перевірка через північ

            if in_sleep:
                if sleep_override_until and now < sleep_override_until:
                    return True  # Якщо є ручне перевизначення – сканер
доступний
                manual_override_enabled = False
                return False  # Інакше сканер вимкнено через режим сну

        return manual_override_enabled  # Якщо немає режиму сну –
орієнтуємось на ручне керування

    def admin_required(f):  # Приймає функцію f, яку потрібно захистити
        @wraps(f)  # Зберігає ім’я, докстрінг та інші метадані функції
        def decorated_function(*args, **kwargs):
            if session.get('logged_in'):
                return f(*args, **kwargs)  # Якщо користувач увійшов –
виконуємо функцію
            return redirect(url_for('login'))  # Інакше перекидаємо на
сторінку входу
        return decorated_function

    @app.route('/verify', methods=['POST'])
    def verify():
        if not is_scanner_enabled():  # Якщо сканер неактивний —
повертаємо відповідь про недоступність
            return jsonify({"match": "false", "message": "Scanner
disabled"}), 403

        fid = int(request.form.get('id', 0))  # Отримуємо ID з POST-
запиту
        if fid > 0:
            log_activity(fid, "Access granted")  # Логування спроби з
доступом
            return jsonify({"match": "true", "id": fid}), 200
        else:
            log_activity(0, "Access denied")  # Логування відмови у
доступі
            return jsonify({"match": "false"}), 200

    @app.route('/enroll', methods=['POST'])
    def enroll():
        fid = int(request.form.get('id', 0))  # Отримуємо ID
користувача
        name = request.form.get('name', '').strip()  # Отримуємо ім’я
користувача

        if fid > 0 and name:
            conn = get_db_connection()
            cursor = conn.cursor()
            cursor.execute('''
                INSERT INTO fingerprints (id, name)
```

```
                    VALUES (%s, %s)
                    ON CONFLICT (id) DO UPDATE SET name = EXCLUDED.name
                ''', (fid, name))  # Додаємо нового користувача або
оновлюємо ім’я при співпадінні ID
                conn.commit()
                conn.close()
                return jsonify({"status": "success", "id": fid, "name":
name})
            else:
                return jsonify({"status": "error", "message": "Invalid
data"}), 400

        @app.route('/toggle_scanner', methods=['POST'])
        @admin_required
        def toggle_scanner():
            global manual_override_enabled, sleep_override_until
            now = datetime.now(kyiv)  # Поточний час у часовій зоні

            if not is_scanner_enabled():  # Якщо сканер зараз вимкнений
                manual_override_enabled = True  # Вмикаємо вручну
                if sleep_end:
                    sleep_end_today = datetime.combine(now.date(),
sleep_end)
                    if sleep_start and sleep_start > sleep_end and
now.time() < sleep_end:
                        sleep_end_today += timedelta(days=1)
                    sleep_override_until = kyiv.localize(sleep_end_today)
# Сканер активний до кінця графіку
            else:
                manual_override_enabled = False  # Вимикаємо вручну
                sleep_override_until = None  # Скасовуємо ручне
перевизначення

            return redirect(url_for('logs'))  # Переадресація до журналу
подій

        @app.route('/set_sleep_schedule', methods=['POST'])
        @admin_required
        def set_sleep_schedule():
            global sleep_start, sleep_end, sleep_override_until
            try:
                h1, m1 = map(int, request.form['from_time'].split(':'))
                h2, m2 = map(int, request.form['to_time'].split(':'))
                new_start = dtime(hour=h1, minute=m1)
                new_end = dtime(hour=h2, minute=m2)

                # Якщо змінився розклад - скидаємо ручне керування
                if sleep_start != new_start or sleep_end != new_end:
                    sleep_override_until = None

                sleep_start = new_start
                sleep_end = new_end
            except Exception as e:
                print(f" Помилка встановлення режиму сну: {e}")

            return redirect(url_for('logs'))  # Переадресація на сторінку
```

```
логів

      @app.route('/scanner_status')
      def scanner_status():
          return '1' if is_scanner_enabled() else '0'  # Повертає 1, якщо
сканер активний, інакше 0

      @app.route('/logs')
      @admin_required
      def logs():
          conn = get_db_connection()
          cursor = conn.cursor()
          cursor.execute('''
              SELECT l.verification_id, f.id, f.name, l.timestamp,
l.result
              FROM logs l LEFT JOIN fingerprints f ON l.fingerprint_id =
f.id
              ORDER BY l.timestamp DESC
          ''')  # Об'єднання таблиці логів і відбитків
          logs_data = cursor.fetchall()
          conn.close()
          return render_template('logs.html', logs=logs_data,
                                 scanner_enabled=is_scanner_enabled(),
                                 manual_override=manual_override_enabled,
                                 sleep_start=sleep_start,
sleep_end=sleep_end)

      @app.route('/login', methods=['GET', 'POST'])
      def login():
          login_error = None
          if request.method == 'POST':
              login = request.form.get('login')
              password = request.form.get('password')
              if login == ADMIN_LOGIN and password == ADMIN_PASSWORD:
                  session['logged_in'] = True
                  return redirect(url_for('logs'))
              else:
                  login_error = "Невірний логін або пароль"
          return render_template('login.html', login_error=login_error)

      @app.route('/logout')
      def logout():
          session.pop('logged_in', None)  # Видаляємо ознаку авторизації
з сесії
          return redirect(url_for('login'))  # Повертаємо на сторінку
входу

      if __name__ == '__main__':
          app.run(debug=True, host='0.0.0.0', port=5000)  # Команда для
запуску серверу
```

ДОДАТОК В

```
<!DOCTYPE html>
     <html lang="uk">
     <head>
         <meta charset="UTF-8">
         <meta name="viewport" content="width=device-width, initial-
scale=1.0">
         <title>Вхід до системи</title>
         <style>
             body {
                 margin: 0;
                 font-family: Arial, sans-serif;
                 background-color: #f5f7fa;
                 display: flex;
                 align-items: center;
                 justify-content: center;
                 height: 100vh;
             }
             .login-box {
                 background-color: white;
                 padding: 2rem;
                 border-radius: 10px;
                 box-shadow: 0 0 20px rgba(0,0,0,0.1);
                 width: 100%;
                 max-width: 400px;
             }
             .login-box h2 {
                 margin-top: 0;
                 text-align: center;
                 color: #2c3e50;
             }
             label {
                 display: block;
                 margin-bottom: 0.5rem;
                 font-weight: bold;
                 color: #34495e;
             }
             input[type="text"], input[type="password"] {
                 width: 100%;
                 padding: 0.5rem;
                 margin-bottom: 1rem;
                 border: 1px solid #ccc;
                 border-radius: 5px;
                 box-sizing: border-box;
             }
             button {
                 width: 100%;
                 padding: 0.75rem;
                 background-color: #3498db;
                 color: white;
                 border: none;
                 border-radius: 5px;
                 font-size: 1rem;
                 cursor: pointer;
             }
```

```
        button:hover {
            background-color: #2980b9;
        }
        .error {
            color: red;
            text-align: center;
            margin-bottom: 1rem;
        }
    </style>
</head>
<body>
    <div class="login-box">
        <h2>Вхід до системи</h2>
        <form method="post">
            <label for="login">Логін</label>
            <input type="text" id="login" name="login" required>
            <label for="password">Пароль</label>
            <input type="password" id="password" name="password"
required>

            {% if login_error %}
            <p style="color: red;">{{ login_error }}</p>
            {% endif %}

            <button type="submit">Увійти</button>
        </form>
    </div>
</body>
</html>
```

ДОДАТОК Г

```
<!DOCTYPE html>
     <html lang="uk">
     <head>
         <meta charset="UTF-8">
         <meta name="viewport" content="width=device-width, initial-
scale=1.0">
         <title>Журнал верифікацій</title>
         <style>
             /* Основні стилі сторінки */
             body {
                 font-family: Arial, sans-serif;
                 background: #f4f6f8;
                 margin: 0;
                 padding: 0 10px;
             }

             .container {
                 max-width: 800px;
                 margin: auto;
                 padding: 15px;
                 background: #fff;
                 border-radius: 8px;
                 box-shadow: 0 0 10px rgba(0,0,0,0.05);
             }

             h1 {
                 text-align: center;
                 color: #333;
             }

             .status-section {
                 display: flex;
                 align-items: center;
                 justify-content: space-between;
                 flex-wrap: wrap;
                 margin-bottom: 20px;
             }

             .scanner-status {
                 font-size: 18px;
                 font-weight: bold;
             }

             .scanner-status span {
                 margin-left: 8px;
             }

             form {
                 display: flex;
                 flex-direction: column;
                 margin-bottom: 20px;
             }

             form label {
```

```
            margin-top: 10px;
            margin-bottom: 5px;
        }

        form input[type="time"], form button {
            padding: 8px;
            font-size: 16px;
            border-radius: 4px;
            border: 1px solid #ccc;
            width: 100%;
            max-width: 300px;
        }

        form button {
            background-color: #1e90ff;
            color: #fff;
            border: none;
            margin-top: 22px;
            cursor: pointer;
        }

        table {
            width: 100%;
            border-collapse: collapse;
            font-size: 15px;
        }

        table thead {
            background-color: #007BFF;
            color: white;
        }

        table th, table td {
            padding: 8px;
            border: 1px solid #ccc;
            text-align: center;
            word-break: break-word;
        }

        @media (max-width: 600px) {
            table th, table td {
                font-size: 14px;
                padding: 6px;
            }
        }

        .logout {
            float: right;
            text-decoration: none;
            color: #007BFF;
            font-weight: bold;
        }
    </style>
</head>
<body>
<div class="container">
```

```
        <!-- Посилання на вихід із системи -->
        <a href="{{ url_for('logout') }}" class="logout">Вийти</a>

        <!-- Заголовок сторінки -->
        <h1>Журнал верифікацій</h1>

        <!-- Блок стану сканера і кнопка перемикання -->
        <div class="status-section">
            <div class="scanner-status">
                Сканер:
                <span>{{ '🟢 Увімкнено' if scanner_enabled else '🔴
Вимкнено' }}</span>
            </div>
            <form method="post" action="{{ url_for('toggle_scanner')
}}">
                <button type="submit">
                    {{ '🔴 Вимкнути сканер' if scanner_enabled else '🟢
Увімкнути сканер' }}
                </button>
            </form>
        </div>

        <!-- Блок встановлення режиму сну -->
        <h2>Режим сну</h2>
        <form method="post" action="{{ url_for('set_sleep_schedule')
}}">
            <label for="from_time">З:</label>
            <input type="time" id="from_time" name="from_time"
required>

            <label for="to_time">До:</label>
            <input type="time" id="to_time" name="to_time" required>

            <button type="submit">Встановити</button>
        </form>

        <!-- Вивід активного графіку сну -->
        {% if sleep_start and sleep_end %}
            <p>Час сну: з {{ sleep_start.strftime('%H:%M') }} до {{
sleep_end.strftime('%H:%M') }}</p>
        {% endif %}

        <!-- Таблиця логів верифікацій -->
        <table>
            <thead>
                <tr>
                    <th>ID верифікації</th>
                    <th>ID відбитка</th>
                    <th>ПІБ</th>
                    <th>Час</th>
                    <th>✅</th>
                </tr>
            </thead>
            <tbody>
                {% for log in logs %}
```

```
                    <tr>
                        <td>{{ log[0] }}</td>
                        <td>{{ log[1] if log[1] is not none else '—'
}}</td>
                        <td>{{ log[2] if log[2] is not none else '—'
}}</td>
                        <td>{{ log[3] }}</td>
                        <td>{{ '🟢' if log[4] == 'Access granted' else '🔴'
}}</td>
                    </tr>
                    {% endfor %}
                </tbody>
            </table>

        </div>
        </body>
        </html>
```

ДОДАТОК Д

```
import sqlite3
      import psycopg2

      #  Підключення до локальної SQLite
      sqlite_conn = sqlite3.connect('fingerprints.db')
      sqlite_cursor = sqlite_conn.cursor()

      #  Підключення до  PostgreSQL
      pg_conn = psycopg2.connect(
          "postgresql://fingerprints_user:тут має бути валідне посилання
для з’єднання "
      )
      pg_cursor = pg_conn.cursor()

      #  Створення таблиць у PostgreSQL
      pg_cursor.execute('''
      CREATE TABLE IF NOT EXISTS fingerprints (
              id INTEGER PRIMARY KEY,
              name TEXT
          )
      ''')

      pg_cursor.execute('''
          CREATE TABLE IF NOT EXISTS logs (
              verification_id SERIAL PRIMARY KEY,
              fingerprint_id INTEGER,
              timestamp TIMESTAMP,
              result TEXT,
              FOREIGN KEY (fingerprint_id) REFERENCES fingerprints(id)
          )
      ''')
      pg_conn.commit()

      #  Отримання даних з SQLite
      sqlite_cursor.execute("SELECT id, name FROM fingerprints")
      fingerprints_data = sqlite_cursor.fetchall()

      sqlite_cursor.execute("SELECT fingerprint_id, timestamp, result
FROM logs")
      logs_data = sqlite_cursor.fetchall()

      #  Завантаження в PostgreSQL
      for row in fingerprints_data:
          pg_cursor.execute('''
              INSERT INTO fingerprints (id, name)
              VALUES (%s, %s)
              ON CONFLICT (id) DO UPDATE SET name = EXCLUDED.name
          ''', row)
      for row in logs_data:
          pg_cursor.execute('''
              INSERT INTO logs (fingerprint_id, timestamp, result)
              VALUES (%s, %s, %s)
          ''', row)
```

```
pg_conn.commit()

#  Закриття підключень
sqlite_conn.close()
pg_cursor.close()
pg_conn.close()

print(" Дані успішно перенесені ")}
```

ДОДАТОК Е

```
Flask
psycopg2-binary
gunicorn
pytz
```