МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут автоматики і електротехніки

Кафедра комп'ютерних технологій та інформаційної безпеки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних технологій та інформаційної безпеки, к.т.н., доцент
_____ С.М. Нужний
«____» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

УДОСКОНАЛЕННЯ ЗАХИЩЕНОЇ СИСТЕМИ ВІДДАЛЕНОГО КОНТРОЛЮ КЛІМАТИЧНИХ ПАРАМЕТРІВ СЕРВЕРНИХ ПРИМІЩЕНЬ

Ступінь вищої освіти: магістр
Галузь знань: 14 Електрична інженерія
Спеціальність: 141 «Електроенергетика, електротехніка та електромеханіка»
Освітньо-професійна програма: Системи технічного захисту інформації, автоматизація її обробки

Виконав: студент 6 курсу групи 6366м
Стефанюк Юрій Олександрович _____

Керівник:
к.т.н., доцент кафедри комп'ютерних технологій та інформаційної безпеки

Сірівчук Андрій Сергійович _____

Кваліфікаційна робота містить результати самостійного виконання навчального завдання. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Миколаїв – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут автоматики і електротехніки

ЗАТВЕРДЖУЮ
Гарант освітньої програми, д.т.н., професор,
професор кафедри КТІБ
_____ Передерій В.І.
«14» 10 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу

Студент: Стефанюк Юрій Олександровмч

Курс:6

Група: 6366м

Ступінь вищої освіти: магістр

Галузь знань: 14 Електрична інженерія

Спеціальність: 141 «Електроенергетика, електротехніка та електромеханіка»

Освітньо-професійна програма: Системи технічного захисту інформації, автоматизація її обробки

1. Тема роботи: Удосконалення захищеної системи віддаленого контролю кліматичних параметрів серверних приміщень
затверджена наказом НУК від «14» 10 2025 р. № 1217-ц

2. Вихідні дані до роботи: платформа одноплатний комп'ютер, мережевий інтерфейс Ethernet, криптографічний протокол TLS 1.3, контроль фізичного доступу, датчик клімату, управління кондиціонером через ІЧ-діод, протоколи обміну даними HTTP/HTTPS, TCP/IP, вебпортал з розмежуванням прав доступу (оператор/адміністратор).

3. Перелік питань, які необхідно розробити: провести аналіз сучасних платформ та обґрунтувати перехід з *ESP/Arduino* на *Raspberry Pi* 4, розробити архітектуру удосконаленої захищеної системи, розробити апаратну частину та програмне забезпечення (вебпортал, шифрування, двофакторна аутентифікація, логування), провести експериментальні дослідження та оцінку захищеності системи.

4. Терміни виконання завдання:
термін видачі завдання: «01- 05» вересня 2025 р.
термін подання роботи: «18» грудня 2025 р.

6. План-графік виконання кваліфікаційної роботи

п/п	Заходи	Дата		Готовність
		Навч. тиждень	Контрольна дата	
1.	Наукове стажування	1 - 8	27.10.2025	Звіт з наукового стажування
2.	Організаційні збори	9	29.10.2025	Попередній варіант змісту КР
3.	Рубіжний контроль	10	05.11.2025	Попередній варіант оглядових розділів, звіт з практика
4.	Рубіжний контроль	13	27-28.11.2025	Доповідь на 13-ій Всеукраїнській науково-технічній конференції з міжнародною участю «СПІБТ–2025»
5.	Рубіжний контроль	14	3.12.2025	1. Попередній варіант повної КР 2. Попередній варіант презентації
6.	КЕ на рівень «магістра»	15	8,9.12.2025	Складання державного екзамену зі спеціальності на ступінь магістра
7.	Перевірка на плагіат	15	10.12.2025	Електронний варіант кваліфікаційної роботи, попередній захист (робота передається студентом на кафедру для внутрішньої рецензії).
8.	Оформлення КР та супутньої документації	16	15-17.12.2025	1. Оформлена КР (в форматі pdf) У разі отримання позитивної внутрішньої рецензії, КР подається на зовнішню рецензію. 2. Оформлена презентація (в форматі pdf).
9.	Подання документів на захист	16	18.12.2025	1. Подання щодо захисту КР: тема, успішність, висновок керівника та висновок кафедри про КР. 2. Зовнішня рецензія (окремо від КР). Резюме на КР. 3. Електронний варіант презентації. 4. Електронний варіант КР на диску
10	Захист ДР	17	22,23,24 12.2025	1. Приєднання студента до засідання кваліфікаційної комісії (конференції) у встановлений час для проведення процедури дистанційного захисту 2. Процедура захисту КР.

Керівник
к.т.н., доцент кафедри комп'ютерних технологій
та інформаційної безпеки, _____ А.С. Сірівчук

Студент _____ Ю.О. Стефанюк

АНОТАЦІЯ

Стефанюк Ю. О. Удосконалення захищеної системи віддаленого контролю кліматичних параметрів серверних приміщень.

Кваліфікаційна робота на здобуття ступеня вищої освіти «магістр» за спеціальністю 141 «Електроенергетика, електротехніка та електромеханіка» галузі знань 14 «Електрична інженерія» освітньо-професійної програми «Системи технічного захисту інформації, автоматизація її обробки». – Національний університет кораблебудування імені адмірала Макарова, Миколаїв, 2025.

Пояснювальна записка: 62 с., 30 посилань та 1 додаток.

У кваліфікаційній роботі здійснено удосконалення захищеної автономної системи віддаленого контролю кліматичних параметрів серверних приміщень шляхом переходу з архітектури *ESP32/Arduino* на платформу *Raspberry Pi 4 Model B,* мінімізації ланцюгів передачі даних та реалізації комплексного захисту на всіх рівнях. Розроблено апаратно-програмний комплекс, що включає: контроль фізичного доступу за допомогою, моніторинг температури та вологості, керування кондиціонером через ІЧ-діод, локальний вебпортал з двофакторною аутентифікацією.

Об’єктом дослідження є процеси віддаленого моніторингу та керування кліматичними параметрами в серверних приміщеннях з обмеженим доступом. Предметом дослідження – методи та засоби побудови захищеної автономної системи на базі ш з інтеграцією криптографічного, фізичного та адміністративного захисту.

Результати роботи впроваджено в навчальному процесі кафедри комп’ютерних технологій та інформаційної безпеки НУК ім. адмірала Макарова та можуть бути застосовані на об’єктах критичної інфраструктури воднотранспортного комплексу й енергетичного сектору.

Ключові слова: віддалене керування, , *Raspberry Pi 4, RFID*-аутентифікація, двофакторна автентифікація, кліматичний контроль, серверне приміщення.

ANNOTATION

Stefaniuk Yu. O. Improvement of server rooms climatic parameters remote control protected system.

Qualification work for obtaining the degree of higher education "Master" in specialty 141 "Power Engineering, Electrical Engineering and Electromechanics" field of knowledge 14 "Electrical Engineering" educational and professional program "Technical Information Protection Systems, Automation of its Processing". – Admiral Makarov National University of Shipbuilding, Mykolaiv, 2025.

Explanatory note: 62 p., 35 references.

In the qualification work, a protected autonomous system for remote control of climate parameters of server rooms was improved by switching from the ESP32/Arduino architecture to the Raspberry Pi 4 Model B platform, minimizing data transmission chains and implementing comprehensive protection at all levels. A hardware and software complex was developed, including: physical access control using, temperature and humidity monitoring, air conditioner control via an IR diode, and a local web portal with two-factor authentication.

The object of the research is the processes of remote monitoring and control of climatic parameters in restricted-access server rooms. The subject of the research is methods and means of building a secure autonomous system based on Raspberry Pi 4 with integration of cryptographic, physical and administrative protection.

The results of the work have been implemented in the educational process of the Department of Computer Technologies and Information Security of Admiral Makarov National University of Shipbuilding and can be applied at critical infrastructure facilities of the water transport complex and energy sector.

Keywords: remote control, Raspberry Pi 4, RFID authentication, two-factor authentication, autonomous system, climate control, server room.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface;

CA – Certificate Authority;

CLI – Command Line Interface;

DNS – Domain Name System;

GPIO – General-Purpose Input/Output;

HTTP(S) – HyperText Transfer Protocol (Secure);

I2C – Inter-Integrated Circuit;

IDE – Integrated Development Environment;

IoT – Internet of Things;

IP – Internet Protocol;

LoRaWAN – Long Range Wide Area Network;

MAC – Message Authentication Code;

OTA – Over-The-Air;

PWM – Pulse-Width Modulation;

REST – Representational State Transfer;

SSL – Secure Sockets Layer;

TCP – Transmission Control Protocol;

UDP – User Datagram Protocol;

USB – Universal Serial Bus;

VPN – Virtual Private Network;

ДБЖ – джерело безперебійного живлення;

ВСТУП

Зосередження значних обсягів інформації з обмеженим доступом в інформаційних ресурсах серверних приміщень об’єктів критичної інфраструктури зумовлює актуальність захисту цих ресурсів від цілеспрямованих і випадкових загроз кібербезпеки. Стабільність кліматичних параметрів (температури та вологості) є ключовим фактором надійності серверного обладнання, порушення якого може призвести до перегріву, збоїв або повної відмови апаратних компонентів, спричиняючи значні фінансові та операційні втрати. Згідно з даними Державного центру кіберзахисту України, у 2024 році кількість кіберінцидентів, пов’язаних з порушеннями доступу до систем управління інфраструктурою, зросла на 38 % порівняно з попереднім роком [1].

Розробці захищених систем віддаленого моніторингу та керування кліматичними параметрами присвячено значну кількість наукових робіт вітчизняних [2] та зарубіжних науковців [3–5]. Зокрема, досліджено застосування протоколу *HTTP* з *SSL/TLS (HTTPS)* для безпечного моніторингу критичної інфраструктури на базі *Raspberry Pi* [3], а також загальні рекомендації щодо безпеки *IoT*-пристроїв у критичних секторах [4, 5]. Проте більшість існуючих рішень залежать від хмарних сервісів (наприклад, *AWS IoT* чи *Azure IoT Hub*), що створює додаткові вразливості, або не забезпечують комплексного захисту фізичного доступу до апаратного забезпечення. Відсутність автономних рішень з мінімальними ланцюгами передачі даних та інтеграцією двофакторної аутентифікації (*RFID* + пароль) робить тему удосконалення таких систем особливо актуальною для об’єктів критичної інформаційної інфраструктури воднотранспортного комплексу та енергетичного сектору.

Метою даної кваліфікаційної роботи є удосконалення захищеної системи віддаленого контролю кліматичних параметрів серверних приміщень шляхом

переходу на платформу *Raspberry Pi 4 Model B*, реалізації комплексного захисту (криптографічного, фізичного та адміністративного) та мінімізації ланцюгів передачі даних для усунення можливих вразливостей на різних рівнях системи.

Для досягнення поставленої мети необхідно вирішити такі задачі:

− провести аналіз сучасних платформ для створення захищених систем віддаленого керування кліматичними параметрами, обґрунтувати вибір *Raspberry Pi 4* як базової платформи та перехід від архітектури *ESP/Arduino*;

− розробити архітектуру удосконаленої системи з урахуванням вимог захисту інформації, включаючи механізми аутентифікації, авторизації, шифрування трафіку (*SSL/TLS*), контроль фізичного доступу за допомогою *RFID* та соленоїдного замка;

− розробити програмне забезпечення у вигляді локального вебпорталу з розмежуванням прав доступу (користувач / адміністратор), інтеграцією датчика *DHT11*, керуванням кондиціонером через ІЧ-діод, веденням логів та можливістю віддаленого керування соленоїдом;

− провести експериментальні дослідження розробленої системи, оцінити її стійкість до типових загроз, енергоспоживання та працездатність в умовах реального серверного приміщення.

Об’єктом дослідження є процеси віддаленого моніторингу та керування кліматичними параметрами в серверних приміщеннях з обмеженим доступом.

Предметом дослідження є методи та засоби побудови захищеної автономної системи віддаленого контролю кліматичних параметрів на базі *Raspberry Pi 4* з інтеграцією *RFID*-автентифікації, соленоїдного замка та вебінтерфейсу з криптографічним захистом.

Методами дослідження є системний аналіз, методи криптографічного захисту інформації, методи проектування вбудованих систем, методи веброзробки (*HTML, CSS, JavaScript, Python/Flask*), тестування захищеності (*OWASP Top-10* для *IoT*), експериментальні дослідження.

Практична значимість даної кваліфікаційної роботи полягає у можливості впровадження розробленої системи в серверних приміщеннях об’єктів критичної інформаційної інфраструктури, навчальних лабораторіях НУК ім. адм. Макарова та інших установах, де потрібен автономний контроль клімату. Результати можуть бути використані як типове рішення для освітньо-професійної програми «Системи технічного захисту інформації, автоматизація її обробки» зі спеціальності 125 «Кібербезпека».

Кваліфікаційна робота відповідає наступним фаховим компетентностям:

КФ-1 – Здатність обґрунтовано застосовувати, інтегрувати, розробляти та удосконалювати сучасні інформаційні технології, фізичні та математичні моделі, а також технології створення та використання прикладного і спеціалізованого програмного забезпечення для вирішення професійних задач у сфері інформаційної безпеки та/або кібербезпеки;

КФ-3 – Здатність досліджувати, розробляти і супроводжувати методи та засоби інформаційної безпеки та/або кібербезпеки на об’єктах інформаційної діяльності та критичної інфраструктури;

КФ-6 – Здатність аналізувати, контролювати та забезпечувати систему управління доступом до інформаційних ресурсів згідно встановленої стратегії і політики інформаційної безпеки та/або кібербезпеки організації;

КФ-9 – Здатність аналізувати, розробляти і супроводжувати систему аудиту та моніторингу ефективності функціонування інформаційних систем і технологій, бізнес/операційних процесів в галузі інформаційної безпеки та/або кібербезпеки організації в цілому.

1 АНАЛІЗ СУЧАСНИХ ПЛАТФОРМ ТА ОБҐРУНТУВАННЯ ПЕРЕХОДУ НА *RASPBERRY PI* 4

Необхідно розуміти важливість систем віддаленого керування, особливо в умовах війни та воєнного стану. Такі системи можуть використовуватися для моніторингу різноманітних даних, від заряду акумуляторів джерел безперебійного живлення на вузлових станціях інтернет-/мобільних мереж, керування відключеннями світла/газу для оптимізації генерації на віддалених точках, або у точках, що знаходяться у зоні бойових дій – до керування пошуковими системами, дронами, отримання та передача даних радіолокаційної розвідки, тощо.

Системи віддаленого керування кліматичними параметрами серверних приміщень належать до класу інформаційно-телекомунікаційних систем, що обробляють дані в реальному часі та часто інтегруються з елементами Інтернету речей (*IoT*). Такі системи потенційно обробляють конфіденційну інформацію (наприклад, дані про стан обладнання, логи доступу), тому підпадають під дію нормативно-правових актів України у сфері захисту інформації та кібербезпеки. В контексті серверних приміщень, де стабільність кліматичних параметрів впливає на надійність обладнання, системи віддаленого моніторингу класифікуються як елементи критичної інфраструктури, що вимагає комплексного захисту на апаратному, програмному та фізичному рівнях.

1.1 Аналіз нормативно-правової бази та вимог до захисту інформації в системах віддаленого керування

Розглядаючи віддалені системи контролю мікрокліматом як інформаційно-телекомунікаційні системи то і у питаннях заходів захисту слід регулюватися нормативно-правовими актами, що регулюють захист у таких системах. нижче наведено аналіз ключових документів національного законодавства,

нормативних документів системи технічного захисту інформації (НД ТЗІ), а також міжнародних стандартів, що застосовуються в Україні.

Закон України «Про основні засади забезпечення кібербезпеки України» від 05.102017 № 2163-*VIII* [2]. Цей закон визначає правові та організаційні основи захисту національних інтересів у кіберпросторі, включаючи критичну інформаційну інфраструктуру. Системи віддаленого моніторингу серверних приміщень можуть бути частиною об'єктів критичної інфраструктури (енергетика, IT-сектор), тому вимагають впровадження заходів кіберзахисту, аудиту та реагування на інциденти.

Закон України «Про захист інформації в інформаційно-телекомунікаційних системах» від 05.07.1994 № 80/94-ВР (в редакції) [6]. Цей закон регулює правові відносини щодо захисту інформації в автоматизованих системах, визначає права доступу та обмеження на доступ до такої інформації. Дія цього Закону поширюється на будь-яку інформацію, що обробляється в автоматизованих системах.

Постанова Кабінету Міністрів України від 29.03.2006 № 373 «Про затвердження Правил забезпечення захисту інформації в інформаційних, телекомунікаційних та інформаційно-телекомунікаційних системах» [7]. Визначає загальні вимоги до захисту інформації, що є власністю держави або з обмеженим доступом, включаючи криптографічний та технічний захист, контроль доступу, логування подій та захист від несанкціонованих дій.

НД ТЗІ 2.5-004-99 «Критерії оцінки захищеності інформації в комп'ютерних системах від несанкціонованого доступу» [8]. Встановлює класи захищеності та вимоги до контролю доступу в автоматизованих системах.

НД ТЗІ 2.5-005-99 «Класифікація автоматизованих систем і стандартні функціональні профілі захищеності» [9]. Визначає профілі захищеності для систем, подібних до віддаленого моніторингу.

НД ТЗІ 3.7-003-05 «Порядок проведення робіт із створення комплексної системи захисту інформації» [10]. Регулює розробку КСЗІ, включаючи віддалений доступ.

Для систем віддаленого керування ключовими в поданих документах є вимоги до організації захищених каналів зв’язку та віддаленого доступу (використання *HTTPS, SSL/TLS*), відповідність вимогам щодо ідентифікації, аутентифікації та авторизації користувачів, ведення журналів подій та взаємодії користувачів з інформацією всередині системи.

Міжнародні стандарти відіграють ключову роль у формуванні найкращих практик захисту інформації в системах віддаленого керування, особливо тих, що належать до класу *IoT*-пристроїв або промислових систем моніторингу (наприклад, *BMS — Building Management Systems*). В Україні ці стандарти часто адаптуються як національні (ДСТУ) або використовуються як рекомендації для впровадження комплексної системи захисту інформації (КСЗІ) та систем управління інформаційною безпекою (СУІБ). Вони доповнюють національну нормативну базу, забезпечуючи відповідність глобальним вимогам до захищеного віддаленого доступу, шифрування даних та управління ризиками.

ДСТУ *ISO/IEC* 27001:2023 «Інформаційна безпека, кібербезпека та захист конфіденційності. Системи керування інформаційною безпекою. Вимоги» (ідентичний *ISO/IEC* 27001:2022) [11]. Цей стандарт є основою для побудови СУІБ і безпосередньо стосується систем віддаленого керування. Контроль *A.6.7* (*Remote working*) вимагає впровадження заходів для безпечної віддаленої роботи, включаючи захищені канали зв'язку (*VPN, HTTPS*), сильну аутентифікацію (*MFA*), контроль доступу та моніторинг сесій. Для *IoT*-подібних систем віддаленого моніторингу клімату рекомендується інтеграція цих вимог з контролями *A.8* (*Asset management*) та *A.*14 (*System and application security*), що забезпечують захист від несанкціонованого доступу через віддалені інтерфейси (*WiFi, HTTP/HTTPS*).

Рекомендації *NIST* (США) для кібербезпеки *IoT*-пристроїв. Серія публікацій *NIST SP 800-213 (2021)* та *NIST SP 800-213A* надає керівництво щодо вимог до кібербезпеки *IoT*-пристроїв, включаючи базові можливості (*device capabilities*) для підтримки контролю доступу, шифрування та оновлення ПЗ [12, 15]. *NISTIR 8228* (2019) акцентує увагу на управлінні ризиками кібербезпеки та конфіденційності в *IoT* протягом усього життєвого циклу пристроїв [13]. Ці рекомендації застосовні до систем віддаленого моніторингу серверних приміщень як найкращі практики: обов'язкове використання *SSL/TLS* для передачі даних, *MFA* для аутентифікації та моніторинг аномалій у трафіку.

Стандарт *IEC* 62443 (серія) для промислових систем автоматизації та керування (*IACS*) [14]. Цей стандарт визначає вимоги до кібербезпеки промислових систем, включаючи віддалений доступ та моніторинг. Частина *IEC* 62443-3-3 фокусується на системних вимогах (*SR*), таких як ідентифікація та аутентифікація (*SR* 1.1–1.3), рольовий доступ (*RBAC*) та зонування мереж. Для систем віддаленого керування кліматом (аналогів *SCADA*/*BMS*) рекомендується сегментація зон та кондуїтів, захищений віддалений доступ (*VPN, TLS*) та аудит сесій. В Україні *IEC* 62443 використовується як основа для захисту об'єктів критичної інфраструктури.

ETSI EN 303 645 «*Cyber Security for Consumer Internet of Things*: *Baseline Requirements*» (2020, з оновленнями) [16]. Хоча орієнтований на споживчі *IoT*-пристрої, цей стандарт встановлює базові вимоги до безпеки, включаючи відсутність універсальних паролів, захищене оновлення ПЗ та мінімізацію поверхні атаки. Рекомендації застосовні до систем віддаленого моніторингу, де пристрої підключені до мережі (*WiFi, HTTPS*).

Вимоги *GDPR* (*Regulation* (*EU*) 2016/679) щодо захисту даних у віддалених системах [17]. Якщо система обробляє персональні дані (наприклад, логи доступу з ідентифікацією користувачів), необхідно забезпечити шифрування

передачі (*end-to-end*), контроль доступу та можливість видалення даних. Стаття 32 вимагає технічних заходів для захисту даних під час віддаленого доступу.

Застосування цих стандартів у проекті захищеної системи віддаленого керування кліматичними параметрами дозволяє досягти високого рівня безпеки, сумісного з національними вимогами, та забезпечити стійкість до сучасних загроз (*DDoS*, *MITM*-атаки). Подальша реалізація включає інтеграцію *SSL*/*TLS*, цифрових сертифікатів та *MFA*.

1.2 Огляд існуючих рішень для контролю кліматичних параметрів серверних приміщень

Контроль кліматичних параметрів у серверних приміщеннях є критичним завданням для забезпечення стабільної роботи обладнання, запобігання перегріву та мінімізації ризиків виходу з ладу апаратного забезпечення. Згідно з рекомендаціями *ASHRAE*, оптимальний діапазон температури становить 18–27 °*C*, відносної вологості – 40-60 % [18]. Сучасні рішення включають апаратні датчики, програмне забезпечення для моніторингу та віддаленого доступу, інтеграцію з *IoT*-технологіями. Вони дозволяють реалізувати реальний час моніторингу, сповіщення про аномалії та автоматизоване реагування.

На сучасному ринку представлено широкий спектр комерційних рішень для моніторингу кліматичних параметрів серверних приміщень та дата-центрів. Ці системи забезпечують збір даних з датчиків температури, вологості, точки роси, витоку води, доступу до приміщень, диму та інших параметрів, з можливістю віддаленого доступу через веб-інтерфейс, мобільні додатки, інтеграцію з *SNMP*, сповіщеннями *via email*/*SMS*/*SNMP traps*. Більшість рішень підтримують *PoE* (*Power over Ethernet*), розширення кількості датчиків та інтеграцію з системами *DCIM* (*Data Center Infrastructure Management*) (рис. 1.1).

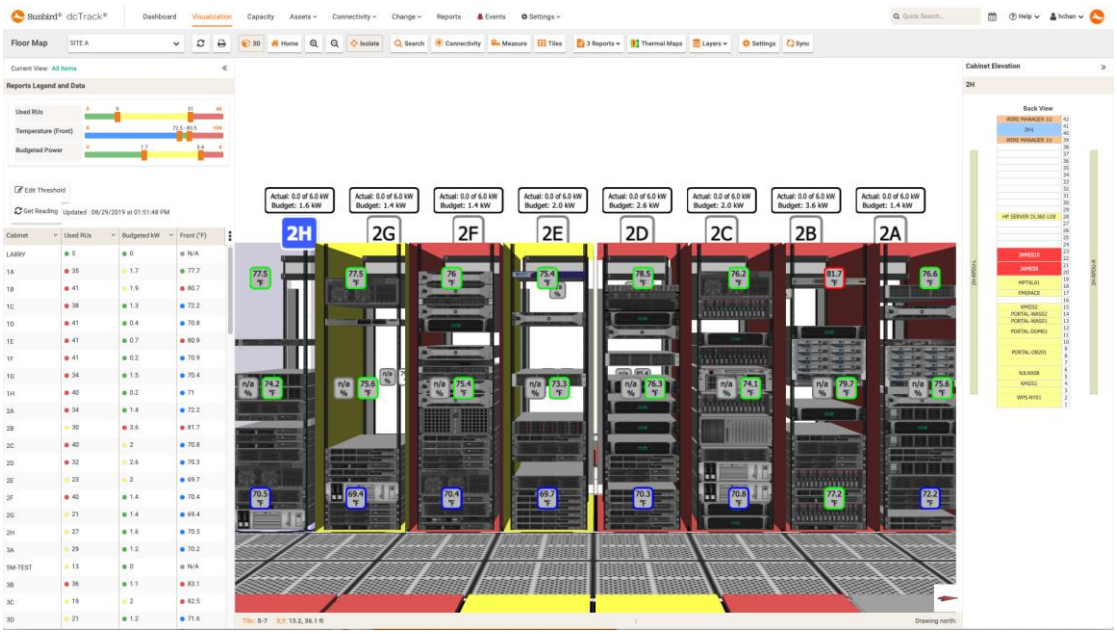


Рисунок 1.1 – Зовнішній вигляд *DCIM* на прикладі *Sunbird dc*:*Track*

APC by Schneider Electric NetBotz Rack Monitor серії (наприклад, *NetBotz Rack Monitor* 750/755/250). Це комплексна система моніторингу та управління фізичною безпекою (рис. 1.2), що включає інтегровані датчики температури, вологості, точки роси, камери спостереження (до 4 *HD*-камер), контроль доступу (до 2 дверей стійок з *RFID*) та підтримку до 16 дротових/47 бездротових датчиків [19, 20]. Інтеграція з платформою *EcoStruxure IT* забезпечує централізоване управління, сповіщення та аналіз даних. Система призначена для захисту від фізичних загроз у дата-центрах та *edge*-обчисленнях.

Рисунок 1.2 – Зовнішній вигляд *NetBotz Rack Monitor* 250

Vertiv Geist Watchdog серії (*Watchdog* 15, *Watchdog* 15-*P*, *Watchdog* 100). Пристрої з вбудованими датчиками температури, вологості та точки роси, підтримкою *PoE*, до 4–8 зовнішніх датчиків. Забезпечують веб-інтерфейс з логуванням даних, графіками та сповіщеннями про перевищення порогів. Моделі *Watchdog* 15/15-*P* (рис. 1.3) – компактні з *PoE*, *Watchdog* 100 – розширена версія з реле для керування зовнішніми пристроями [21, 22].

Рисунок 1.3 – *Geist WATCHDOG* 15

Rittal Computer Multi Control III (*CMC III*) *Processing Unit* та *CMC III Processing Unit Compact*. Центральний блок моніторингу з підтримкою до 32 датчиків через *CAN-bus* (температура, вологість, доступ, дим, витік води). Модульна архітектура, інтеграція з *SNMP*, веб-доступ, *PoE*. Призначена для моніторингу стійок та приміщень у промислових та *IT*-середовищах *[*23, 24*]*.

Ці системи забезпечують високу надійність та функціональність, але часто залежать від пропрієтарного ПЗ або хмарних сервісів, що може створювати ризики безпеки при віддаленому доступі.

1.3 Порівняльний аналіз одноплатних комп’ютерів та обґрунтування вибору *Raspberry Pi 4 Model B*

Попередня реалізація захищеної системи віддаленого контролю кліматичних параметрів серверних приміщень базувалася на мікроконтролері *NodeMCU ESP8266 V3*. Така система забезпечувала базовий збір даних з датчиків температури, вологості, керування ІЧ-діодом та передачу даних/керування через *WiFi/HTTPS*, але виявило обмеження: недостатня обчислювальна потужність для одночасної передачі даних у реальному часі та кодів керування, неможливість аутентифікації з *2FA/MFA*, маршрутизації та контролю доступу (наприклад, *RFID/Z-Wave*). Обмежений обсяг *RAM* (80 КБ у *NodeMCU ESP8266 V3*) ускладнювало впровадження повноцінного веб-сервера з *SSL/TLS*, логуванням та керуванням системою фізичного доступу (*RFID* + реле з соленоїдом).

Для удосконалення системи з елементами контролю фізичного доступу (інтеграція *RFID*-рідерів, камер, реле) необхідний перехід до потужнішого одноплатного комп'ютера (*SBC*), здатного запускати повноцінну *ОС* (*Linux*), обробляти криптографію та забезпечувати стабільний віддалений доступ.

Порівняно популярні *SBC: Raspberry Pi 4 Model B, Orange Pi 5, BeagleBone Black, ODROID N2+* та *NVIDIA Jetson Nano [*25–29*]* (табл. 1.1).

Таблиця 1.1 – Порівняльні характеристики одноплатних комп’ютерів

Одноплатний ПК / Параметр	*Raspberry Pi* 4 *Model B*	*Orange Pi*	*,BeagleBone Black*	*ODROID N*2+	*NVIDIA Jetson Nano*
Процесор	"*Broadcom BCM*2711, 4× *Cortex-A*72 1.8 *GHz*"	"*Rockchip RK*3588*S*, 4× *A*76 + 4× *A*55 до 2.4 *GHz*"	*AM*3358 *Cortex-A*8 1 *GHz*	"*Amlogic S*922*X*, 4× *A*73 + 2× *A*53 до 2.2 *GHz*"	*Quad Cortex-A*57 1.43 *GHz*
RAM	1/2/4/8 *GB LPDDR*4	4/8/16/32 *GB LPDDR*4/5	512 *MB DDR*3	2/4 *GB DDR*4	4 *GB LPDDR*4

Продовження таблиці 1.1

Одноплатний ПК / Параметр	*Raspberry Pi* 4 *Model B*	*Orange Pi*	,*BeagleBone Black*	*ODROID N*2+	*NVIDIA Jetson Nano*
GPIO	40 *pin*	40 *pin* (сумісний)	65 *pin*	40 *pin*	40 *pin*
Мережа	"*Gigabit Ethernet*, *WiFi* 5, *BT* 5.0"	"*Gigabit Ethernet*, *WiFi* 6, *BT* 5.0"	*Ethernet* 100 *Mbps*	*Gigabit Ethernet*	*Gigabit Ethernet*
USB	2× *USB* 3.0 + 2× *USB* 2.0 + 1× *USB Type C*	2× *USB* 3.0 + *USB* 2.0	*USB* 2.0	4× *USB* 3.0	4× *USB* 3.0
Відео	2× *micro HDMI* (4*K*@60)	*HDMI* 2.1 (8*K*@60)	*micro HDMI*	*HDMI* 2.0 (4*K*@60)	*HDMI*/*DP* (4*K*)
Споживання енергії	5–8 Вт	5–10 Вт	2–3 Вт	5–8 Вт	5–10 Вт
"Ціна (орієнтовно, 2025)"	45–75 *USD*	60–120 *USD*	50–60 *USD*	70–90 *USD*	100–130 *USD*
Спільнота/підтримка	Найбільша	Середня	Хороша (промислова)	Хороша	Хороша

Виходячи з наведеної вище таблиці 1 можна дійти висновку, що *Raspberry Pi* 4 *Model B* (рис. 1.4) забезпечує оптимальний баланс потужності, сумісності *GPIO* з проєктами на базі *ESP*/*Arduino* та підтримки *Raspberry Pi OS* (на базі *Debian*), що розширює спектр використання даного одноплатного комп’ютера не тільки у вигляді керуючого пристрою системи, а й дає можливість розгортати керуюче програмне забезпечення для кількох систем одночасно. Велика спільнота та довготривала підтримка полегшує інтеграцію бібліотек та драйверів, зокрема адаптації бібліотек *Adafruit* з *Arduino*/*ESP*, бібліотек для *SSL*/*TLS*, ПЗ для *Linux* і т. д.

Рисунок 1.4 – Зовнішній вигляд *Raspberry Pi* 4

Orange Pi 5 пропонує порівняно вищу продуктивність (*RK*35882 з *NPU*), але меншу спільноту та гіршу стабільність драйверів для *GPIO*.

BeagleBone Black підходить для реального часу (*PRU*), але обмежена потужність та відсутність *WiFi* роблять його менш зручним для використання, оскільки в такому разі необхідна окрема лінія для під'єднання наведеного одноплатного комп'ютера до мережі.

*ODROID N*2+ та *Jetson Nano* потужніші, але дорожчі та менш сумісні з екосистемою *ESP, Arduino, Raspberry Pi.*

Відомі та виявлені у ході створення попередньої реалізації системи віддаленого контролю мікрокліматом на базі *NodeMCU V*3 *ESP*8266 оголили питання недостатньої обчислювальної потужності та гнучкості мікроконтролерів для створення подібних систем. Перехід саме на *Raspberry Pi* 4 *Model B* (1 *Gb*) обґрунтований гарною екосистемою та підтримкою спільноти, потужним процесором та об'ємом пам'яті, варіативністю використання, можливістю запуску різних варіантів систем з одного одноплатного комп'ютера (модульність), можливістю використання одночасно декількох мережевих інтерфейсів та великою кількістю *GPIO* виходів, що дозволяє змінювати та доповнювати системи у подальшому.

2 АРХІТЕКТУРА ТА ФУНКЦІОНУВАННЯ УДОСКОНАЛЕНОЇ ЗАХИЩЕНОЇ СИСТЕМ

2.1 Загальна структурна схема системи

На рисунку 2.1 наведено детальну схему архітектури системи віддаленого контролю мікрокліматом. Подана схема відображає можливість у двох режимах, але не відображає додаткової можливості використання *Gigabit Ethernet* на *Raspberry Pi* 4.

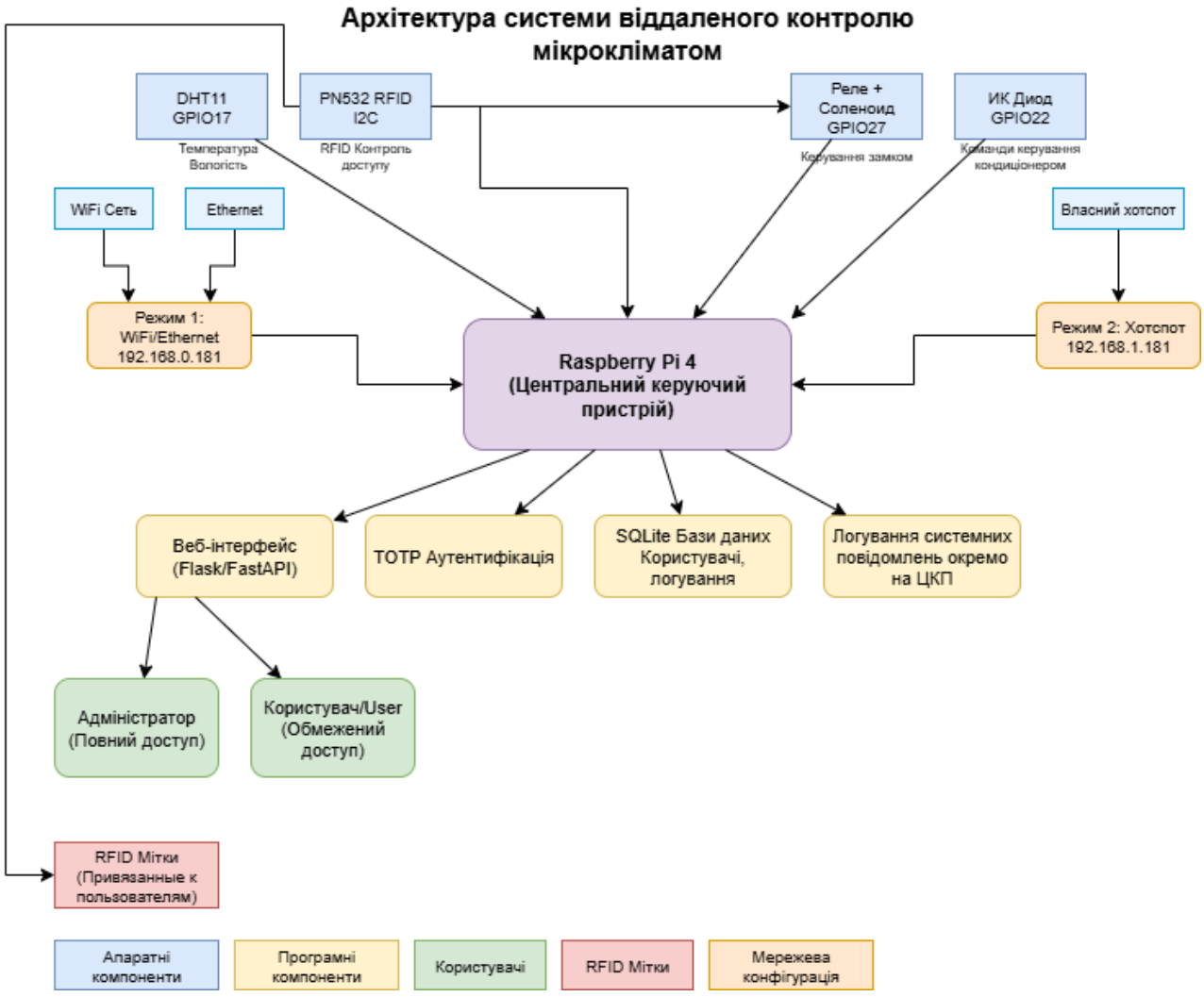


Рисунок 2.1 – Схематичне зображення архітектури системи віддаленого контролю мікрокліматом

Головним завданням системи є забезпечення безперебійного функціонування віддаленого моніторингу та керування не тільки кліматичними параметрами у серверному приміщені, а й можливістю керувати доступом – як фізичним доступом до приміщення так і фізичним доступом до самої системи. Інформація збирається та обробляється на центральному керуючому пристрою – *Raspberry Pi* 4, після чого надається у зручному для кінцевого користувача вигляді на веб-порталі.

Завдяки наявності двох мережевих інтерфейсів у *Raspberry Pi* 4, а саме – *Gigabit Ethernet* та *Wi-Fi* забезпечується безперебійна доступність системи навіть при відключенні одного з них.

У разі відключення обох інтерфейсів – центральний керуючий пристрій переходить у режим роботи хотспоту із заздалегід прописаними параметрами, що надає можливість отримати доступ до елементів керування навіть при відсутності Інтернет мережі.

На рисунку 2.2 показано структурну схему системи, від підключення до мережі електрозабезпечення – до підключення користувача до веб-порталу. Необхідно відмітити, що система є відносно енергонезалежною, може живитися від невеликого ДБЖ, при цьому буде забезпечено, як лінія живлення самого центрального керуючого пристрою, так і 12В лінія для керування соленоїдним замком.

На рисунку 2.2 кольорами виділено – лінії підключення електроживлення 220В з головної мережі червоним кольором, 12 В – чорним кольором, лінії зв’язку з датчиками та елементами керування – зеленим кольором, синім кольором – мережеве з’єднання.

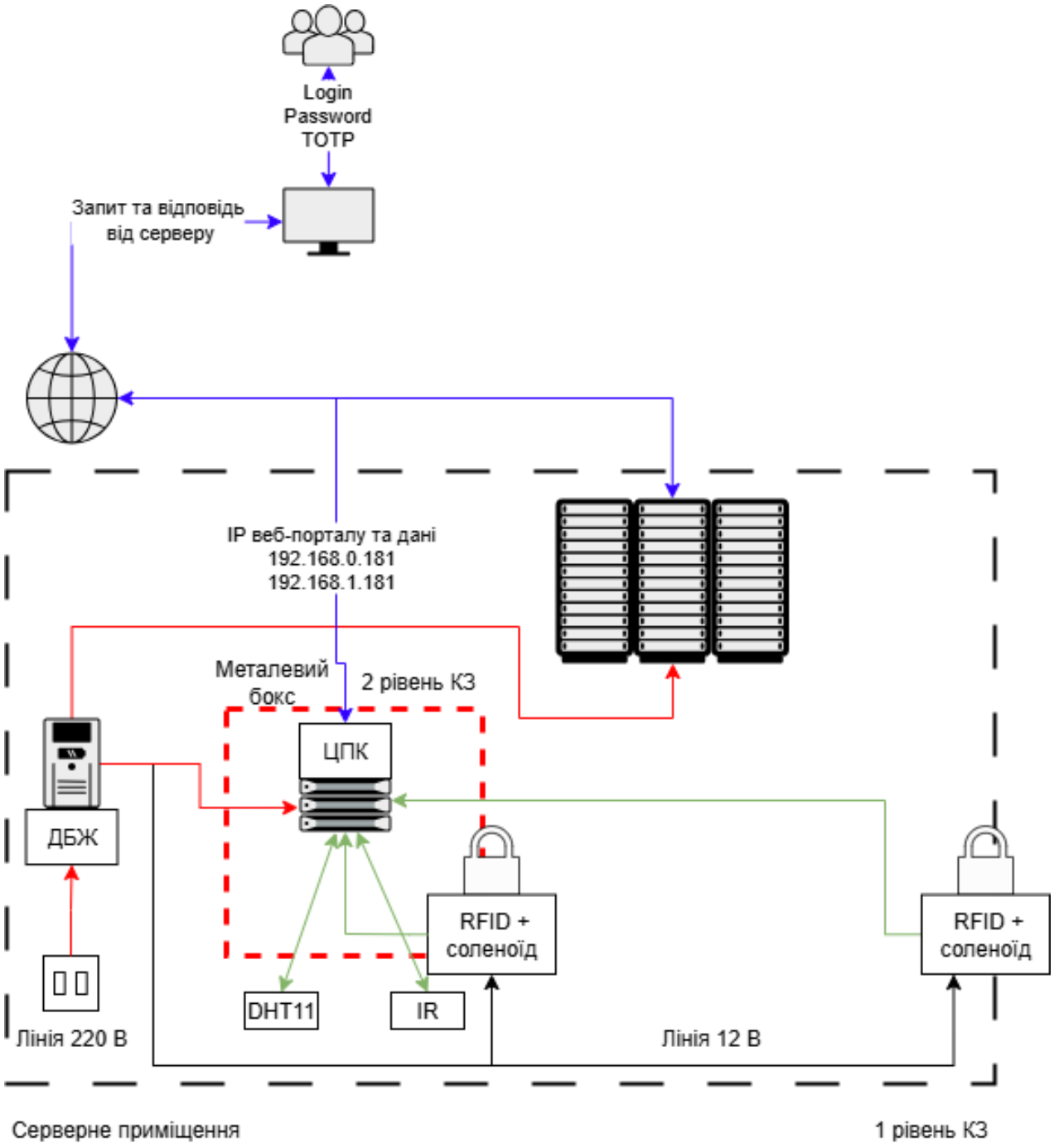


Рисунок 2.2 – Структурна схема системи

При роботі у режимі точки доступу, доступ до панелі керування можна отримати підключившись до мережі зсередини та за статичним адресом (який саме – задається при розробці керуючої програми).

2.2 Механізми захисту

Оскільки система має бути доступна з загальної мережі Інтернет – є необхідним забезпечити доступ до панелі керування за допомогою протоколу *HTTPS*. Етапи отримання домену, білого *IP*, сертифікатів *TLS*/*SSL* та впровадження сертифікатів від сервісу *Let`s Encrypt*, а також процес налаштування мережевого обладнання (маршрутизації та прокидання портів на роутері) детально описані у розділі 3 даного джерела *[*30*]*. Процес є цілком аналогічним за тієї відмінності, що запуск утиліти *certbot* можна здійснити напряму з центрального керуючого пристрою та не використовувати для цього віртуальні машини. В минулій реалізації використання віртуальної машини було необхідним, оскільки запуск *certbot* є *CLI*-утилітою, що унеможливлює її запуск з плати *ESP*.

Використання сертифікатів *TLS*/*SSL* забезпечує шифрування, це означає, що конфіденційна інформація, якою обмінюються через веб-портал, не може бути перехоплена і прочитана ніким, окрім передбачуваного одержувача. Система впроваджує сучасні механізми аутентифікації, авторизації, логування та фізичного контролю. Ці механізми забезпечують конфіденційність, цілісність та доступність даних відповідно до вимог ДСТУ *ISO*/*IEC* 27001:2023 та рекомендацій *NIST* щодо *IoT*-систем *[*11, 13, 15*]*.

2.3 Система двофакторної аутентифікації та авторизації (*TOTP*)

Time-based One-Time Password (*TOTP*) — це алгоритм генерації одноразових паролів, залежних від часу, визначений у стандарті *RFC* 6238 (2011) як розширення *HOTP* (*HMAC-based One-Time Password*) з *RFC* 4226. *TOTP* застосовується для реалізації двофакторної аутентифікації (2*FA*/*MFA*) і генерує динамічні коди (зазвичай 6–8 цифр), дійсні протягом короткого інтервалу

(стандартно 30 секунд). Це забезпечує додатковий рівень захисту від перехоплення паролів, фішингу та атак повторного використання (*replay attacks*).

Принцип роботи *TOTP*

Алгоритм базується на симетричному ключі (*shared secret*), який генерується випадково (наприклад, 160-бітний у *base*32-коді) та зберігається на сервері й у додатку аутентифікатора користувача (*Google Authenticator*, *Microsoft Authenticator* тощо).

Генерація коду виконується за формулою (1):

$$TOTP = HOTP(K,T), \quad (1)$$

де:

K – спільний секретний ключ;

T – лічильник часу, обчислений як $T = floor((unix\_time - T0) / X)$, де *unix_time* — поточний *Unix*-час, *T0* = 0 (за замовчуванням), *X* = 30 секунд (крок часу).

$HOTP(K,C) = Truncate(HMAC - SHA - 1(K,C))$, де *Truncate* скорочує 160-бітний *HMAC* до 6–8 цифр [3].

Сервер і клієнт синхронізуються за часом (допуск ±1–3 кроки для компенсації розбіжностей). Якщо код дійсний у поточному або сусідніх інтервалах, аутентифікація успішна.

На рисунку 2.3 показано роботу *TOTP* механізму. користувач вводить логін і пароль, і якщо вони правильні, система перевіряє, чи налаштований у нього автентифікатор. Якщо автентифікатор ще не налаштований, користувача просять зробити це через QR-код або ручний код, після чого додаток генерує одноразовий код. Якщо автентифікатор уже є, система одразу запитує цей код. У разі правильного введення користувач успішно входить у систему, а якщо логін або пароль були неправильними, спрацьовує захист від підбору — механізм Brute Force Lockout.

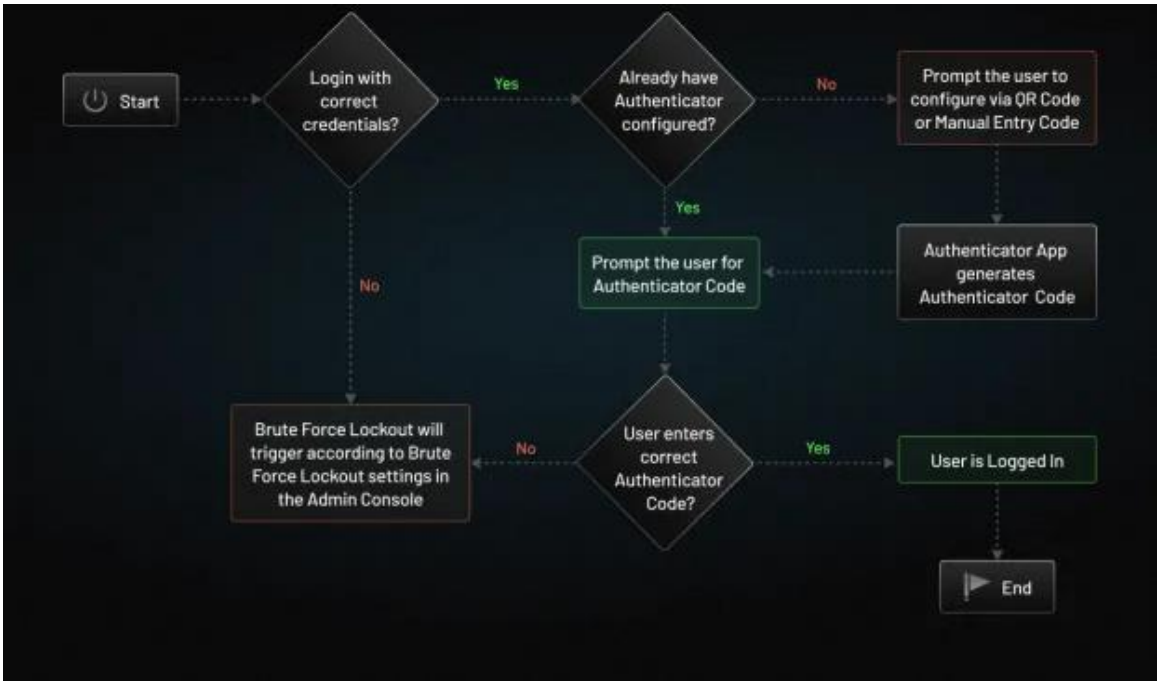


Рисунок 2.3 – Алгоритм авторизації з використанням *TOTP*

На рисунку 2.4 показано приклад вікна введення коду *TOTP* та вікно скидання/налаштування *TOTP*.

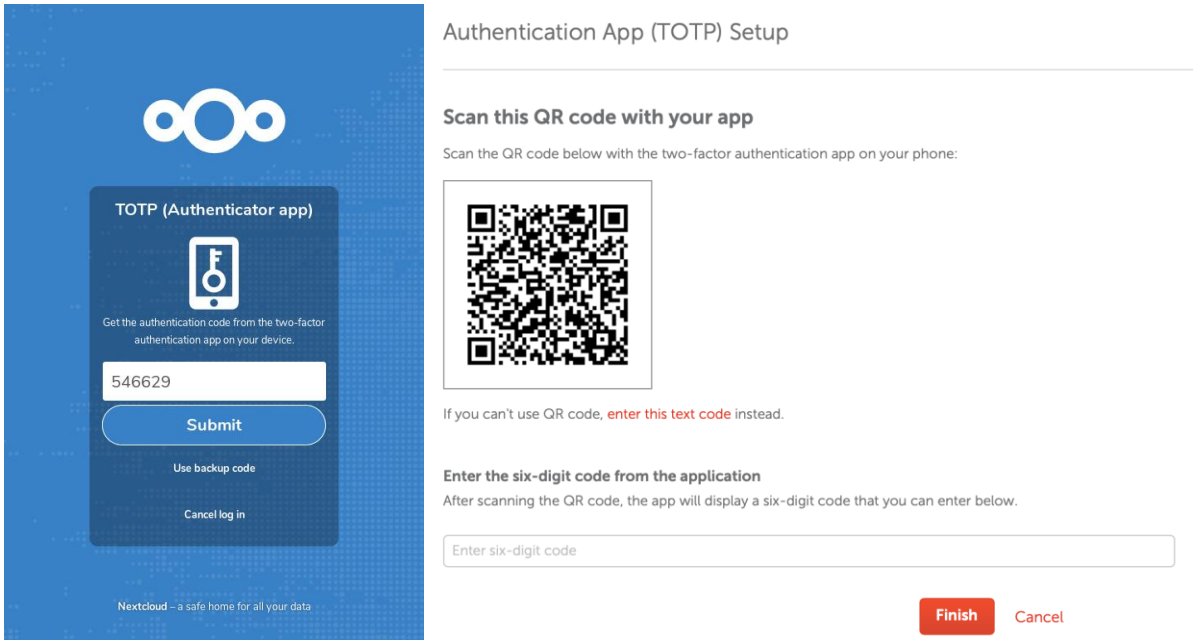


Рисунок 2.4 – Приклад роботи з кодом *TOTP*

Використання двофакторної аутентифікації забезпечить захист доступу до веб-порталу у разі компрометації даних користувача для входу, оскільки для отримання *TOTP* коду, зловмиснику необхідно буде мати фізичний або віддалений безперебійний доступ до мобільного пристрою користувача (можливе під'єднання аутентифікатора, як розширення в браузері, але такий варіант використання не рекомендується).

2.4 Фізичний захист апаратного блоку та захист сесій

Фізичний захист апаратного блоку системи на базі *Raspberry Pi* 4 *Model B* та надійне управління сесіями віддаленого доступу становлять важливі компоненти комплексної системи захисту інформації. Ці заходи запобігають несанкціонованому фізичному доступу до пристрою та зменшують ризики компрометації сесій (*session hijacking*, *fixation*). Реалізація відповідає рекомендаціям ДСТУ *ISO*/*IEC* 27001:2023 (контролі *A*.11 – фізичний захист) та *NIST SP* 800-53 (контроль *PE – Physical and Environmental Protection*).

Апаратний блок розміщується в серверному приміщенні, де можливі загрози включають крадіжку, маніпуляції з *GPIO*, підключення зовнішніх пристроїв (*USB*/*JTAG*) або *tamper*-атаки. Для захисту застосовуються захищені корпуси з *tamper-evident* елементами. Використовуються металеві або алюмінієві корпуси з фіксацією гвинтами та пломбами (*tamper-evident seals*), що виявляють спроби відкриття. Приклади: *KKSB Aluminium Case* (рис. 14) або *Zymbit Zymkey* (рис. 2.5) з периметровим детектуванням.

Zymbit Zymkey — це просунутий апаратний модуль безпеки (*HSM*), спеціально розроблений для посилення захисту одноплатних комп'ютерів *Raspberry Pi* та подібних вбудованих систем, забезпечуючи багаторівневий захист як від кібер-, так і від фізичних атак. В основі *Zymkey* лежить архітектура з двома захищеними процесорами, яка ізолює критичні функції безпеки від потенційно скомпрометованої основної системи *Raspberry Pi*.

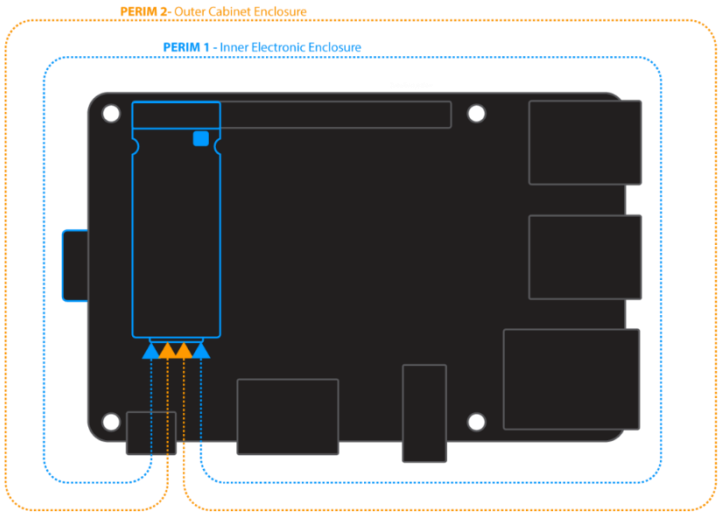


Рисунок 2.5 - *Zymbit Zymkey*

Zymkey надає комплексний набір функцій для захисту пристроїв в автономних або віддалених середовищах:

– шифрування кореневої файлової системи (*RFS*): Дозволяє повністю зашифрувати *SD*-карту за допомогою відкритих стандартів *LUKS* та *dm-crypt*. Ключ шифрування зберігається на *Zymkey* і "розблоковується" лише після успішної автентифікації хост-системи, що унеможливлює читання даних шляхом простого вилучення *SD*-карти.

– фізичне виявлення несанкціонованого доступу (*Tamper Detection*):

а) датчики периметра: Два незалежні електричні контури можуть бути інтегровані в корпус пристрою. Порушення будь-якого з них (наприклад, відкриття корпусу) ініціює заздалегідь визначену політику безпеки;

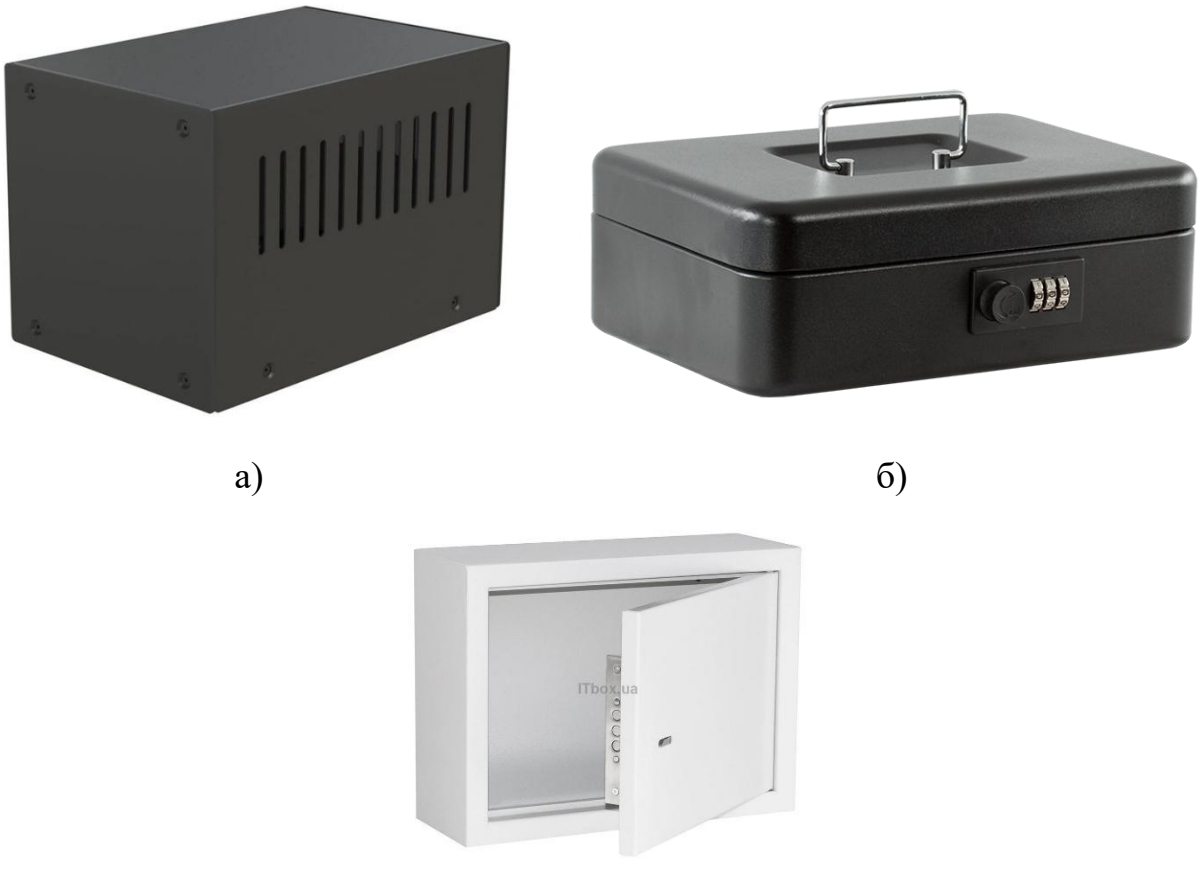
б) акселерометр: Вбудований датчик руху може виявляти переміщення або вібрацію пристрою. Чутливість можна налаштувати, а подію руху використати як тригер для захисних дій;

– політики реагування: У разі виявлення несанкціонованого доступу *Zymkey* може сповістити користувача або, в крайньому випадку (функція "*Last Gasp*"), безповоротно знищити секретні ключі (*zeroization*);

– унікальна багатофакторна ідентифікація пристрою: *Zymkey* прив'язує свою унікальну ідентичність до конкретного хост-комп'ютера та *SD*-карти, створюючи "відбиток" системи. Це гарантує, що ключі працюватимуть лише на цьому конкретному обладнанні;

– інтеграція з хмарними сервісами: Модуль спрощує безпечну автентифікацію з хмарними платформами, такими як *AWS IoT*, використовуючи сертифікати клієнта *TLS*, згенеровані та захищені апаратним забезпеченням.

– комплектація всієї системи у металеві ящики з замком на ключу, або переробка такого металевого ящику (рис. 2.6) для використання разом з *RFID* зчитувачем та соленоїдним замком.



а) б)

в)

Рисунок 2.6 – Бокси для зберігання системи: а) збірний металевий бокс без замку; б) бокс з кодовим замком; в) антивандальний ящик

На наведених вище рисунках показано різні варіації боксів та ящиків, що можна використати для розміщення системи задля забезпечення ще одного периметру захисту. Необхідно відмітити, що на рис. 2.6 в – показано бокс для грошей, але, його фізичні розміри надають можливість для розміщення всередині всіх чутливих елементів системи. Зовнішній вигляд такого ящику може збити з пантелику зловмисника.

3 РОЗРОБКА СИСТЕМИ

3.1 Опис апаратної частини

Апаратна частина розробленої захищеної системи віддаленого контролю кліматичних параметрів серверних приміщень базується на одноплатному комп'ютері *Raspberry Pi* 4 *Model B* з обсягом оперативної пам'яті 1 ГБ. До нього підключено периферійні модулі для моніторингу, контролю доступу та керування виконавчими пристроями. На рисунку 3.1 показано зовнішній вигляд поданого одноплатного комп’ютера.

Рисунок 3.1 – Зовнішній вигляд *Raspberry Pi* 4 *Model B*

Raspberry Pi 4 *Model B* — це компактний комп'ютер з процесором *Broadcom BCM*2711 (4-ядерний *Cortex-A*72, 1.8 ГГц), 1 ГБ *LPDDR*4 *RAM*. Він забезпечує обчислювальну потужність для запуску веб-сервера *FastAPI*, обробки даних з датчиків та керування периферією. Інтерфейси: *Gigabit Ethernet*, *WiFi* 802.11*ac*, *Bluetooth* 5.0, 40-*pin GPIO*, 2×*USB* 3.0, 2×*USB* 2.0, 2×*micro-HDMI* (4*K*). Живлення: 5 В / 3 А через *USB-C*. Споживання: 5-8 Вт.

*PN*532 (рис. 3.2) – це модуль для безконтактного зв'язку на частоті 13.56 МГц (*NFC*/*RFID*). Підтримує протоколи *ISO*/*IEC* 14443*A*/*B*, *MIFARE*, *FeliCa*. Інтерфейси: *I2C*, *SPI*, *UART*. Дальність зчитування: до 10 см. Живлення: 3.3–5 В. Використовується для ідентифікації *RFID*-міток та контролю фізичного доступу.

Рисунок 3.2 – Зовнішній вигляд модулю сканера на *PN*532

Технічні характеристики модулю:

- мікросхема контролера: *PN*532;
- робоча частота: 13.56*MHz;*
- відстань спрацьовування: до 5 см;
- напруга живлення: 3,3-5,4 В;
- струм: 100-150 мА;
- інтерфейс модуля: *UART*, *I2C*, *SPI;*
- габарити модуля: 51*x*25.5 мм;
- габарити антени: 51*x*51 мм.

RFID-мітки на базі чіпа *MIFARE Classic* 1*K* (13.56 МГц, *ISO* 14443*A*). Пам'ять: 1 КБ, *UID*: 4–7 байт. Форм-фактори: картка та брелок. Використовуються для аутентифікації доступу

*DHT*11 (рис. 3.3) — цифровий датчик з діапазоном температури 0–50 °*C* (±2 °*C*), вологості 20–90 % *RH* (±5 %). Інтерфейс: однопровідний цифровий. Живлення: 3.3–5.5 В. Час відгуку: <5 с. Використовуєтсья для моніторингу кліматичних параметрів.

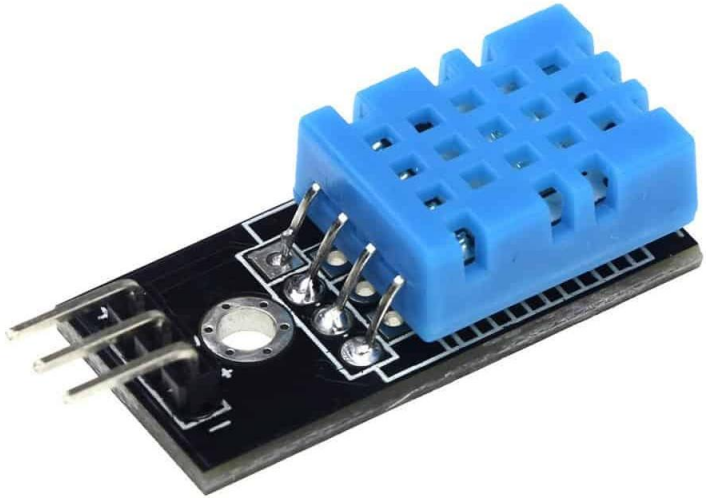


Рисунок 3.3 – Зовнішній вигляд *DHT* 11

JF-0826*B* (рис. 21) — лінійний соленоїд *push-pull* типу. Напруга: 12 В *DC*, струм: 2 А, хід: 10 мм, сила: 20 Н. Розміри: 26×25×22 мм. Може використовуватися для керування замком дверей до серверного приміщення або дверцят металевого боксу.

Технічні характеристики:

- напруга живлення: *DC* 12*V*;
- номінальний струм: 2*A*;
- сила утримання: 20Н;
- хід руху штока: 10 мм;
- довжина кабелю: 190 мм;
- розмір котушки: 26x22x25 мм;
- довжина штока (робоча / загальна): 22/55 мм;
- діаметр штока: 7 мм;

- матеріал: метал;
- вага: 70 г.

KY-019 – одноканальне реле 5 В. Комутація: до 10 А (250 В *AC* / 30 В *DC*). Керування: *TTL*-сигнал 5 В. Оптоізоляція. Використовується для безпечного підключення соленоїда до *GPIO Raspberry Pi*.

Інфрачервоний світлодіод (940 нм) для передачі команд керування (наприклад, кондиціонером). Живлення: 5 В через обмежувальний резистор та транзистор.

У якості джерела живлення системи та 12 В лінії для соленоїдів використано ДБЖ *DC*1018*P* (рис. 3.4) (міні-*UPS*, 10400 мА·год, вихід: 5/9/12 В, до 18 Вт, *POE*) з кабелем *DC to DC* 5.5×2.5 мм. ДБЖ забезпечує автономну роботу при відключенні мережі (до кількох годин залежно від навантаження). ДБЖ використовує збірку з акумуляторів 18650 та має можливість заміни заводської збірки на більш потужну або з більшою ємністю. Але слід враховувати, що при збільшенні ємності – збільшиться не тільки час заряджання, а й навантаження на модуль заряду. Кабель *DC to DC* 5.5×2.5 мм використовує для макетного моделювання.

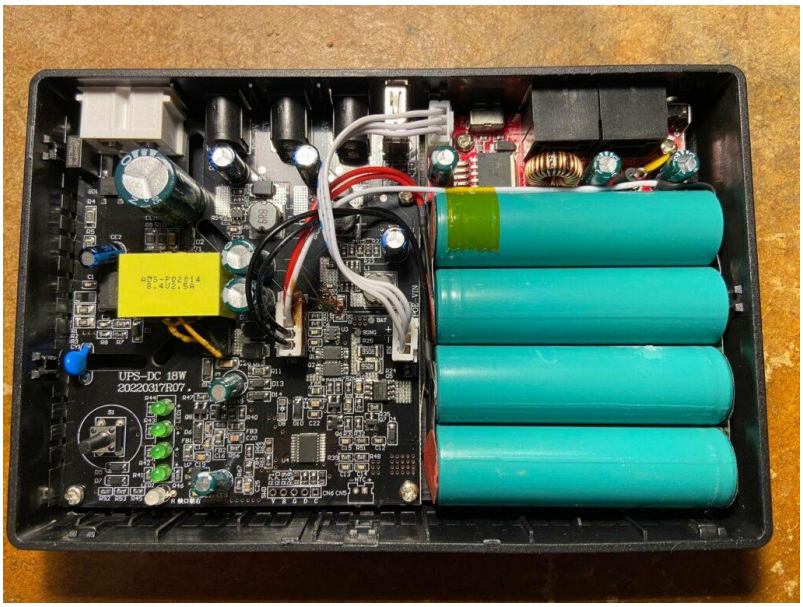


Рисунок 3.4 – ДБЖ DC1018P Вигляд всередині

Перелічені компоненти забезпечують надійну роботу системи моніторингу та контролю доступу. Використання ДБЖ *DC*1018*P* з кабелем 5.5×2.5 мм гарантує стабільне живлення *Raspberry Pi* (5 В) та соленоїда (12 В) навіть при відключенні мережі, не зважаючи на те, що за даними наданими виробником соленоїда даними, соленоїд має номінальний струм 2 А, він чудово спрацьовує від джерела живлення з характеристиками 12 В/0.4 А. Таким чином, сумарна питома потужність системи становитиме приблизно 10 Ватт, що дозволяє мати 8 Ватт запасу від ДБЖ. На рисунку 3.5 наведено фотографії з працюючої макетної моделі системи. Вже в цьому стані систему можна розміщувати у захисному боксі та використовувати.

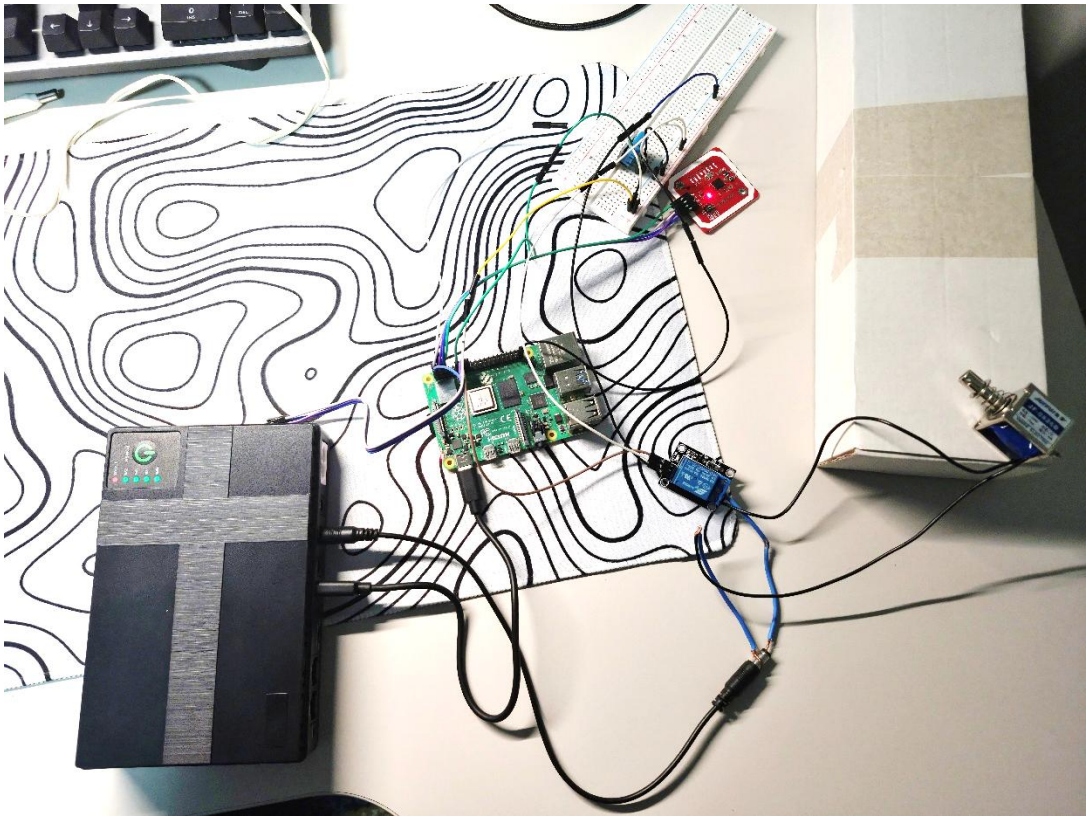


Рисунок 3.5 – Зовнішній вигляд макетної моделі системи (повний)

Необхідно відмітити, що кабель не витримує пусковий струм соленоїда, для реальної системи рекомендується обрати соленоїд з меншою потужністю.

3.2 Налаштування мережевого середовища

Налаштування мережевого середовища є критичним етапом реалізації захищеної системи віддаленого контролю, оскільки забезпечує стабільний і безпечний доступ до веб-інтерфейсу на базі *FastAPI* без залежності від сторонніх хмарних сервісів. *Raspberry Pi* 4 *Model B* оснащений *Gigabit Ethernet* та *WiFi* 5 (802.11*ac*), що дозволяє гнучко інтегрувати пристрій у локальну мережу.

Для стабільної роботи системи у загальній мережі Інтернет – слід закріпити *IP* адресу *Raspberry Pi* 4 *Model B* на *DHCP* сервері, або налаштувати статичну адресу, що не входить до пулу адресів *DHCP*-серверу. Встановлення *IP* адреси з пулу адресів *DHCP*-серверу може призвести не лише до неможливості доступу до веб-порталу, а, й, у найгіршому випадку – до виникнення мережевих аномалій, зокрема зависання мережевого обладнання (комутаторів, маршрутизаторів, тощо).

У випадку роботи у режимі хотспоту – у системі буде підніматися власний *DHCP* сервер та призначатися заздалегідь відома статична адреса. Така реалізація надає змогу налаштовувати та використовувати систему навіть у випадку відсутності мережі.

Оскільки *Raspberry Pi* 4 *Model B* є одноплатним комп’ютером, він вразливий до тих самих загроз, що й звичайний ПК. Тому для захисту мережі у системі налаштовано *Uncomplicated Firewall* з правилами, що відкривають лише 2 порти – 22 (*SSH*) та 443 (*HTTPS*).

SSH (*Secure Shell*) — це безпечний мережевий протокол для віддаленого керування комп'ютерами, що шифрує весь трафік (включно з паролями) для захисту від несанкціонованого доступу, дозволяючи безпечно виконувати

команди, передавати файли та тунелювати інші протоколи, наче ви сидите безпосередньо перед сервером. Це стандарт для адміністрування *Linux* та інших *Unix*-систем, що працює за принципом «клієнт-сервер». Саме цьому мережевому протоколу відповідає порт 22.

HTTPS (*HyperText Transfer Protocol Secure*) – це безпечна версія стандартного протоколу *HTTP*, яка шифрує дані, що передаються між вашим браузером та веб-сайтом, захищаючи конфіденційну інформацію, як паролі чи дані карток, від перехоплення зловмисниками. Це робиться за допомогою криптографічних протоколів *TLS*/*SSL*, візуальна відмінність від *HTTP* для кінцевого користувача в адресному рядку за префіксом *https*:// та значком замка (🔒). Цей мережевий протокол використовує порт 443, незахищена версія *HTTP* використовує порт 80.

Нижче наведено лістинг команд необхідних для налаштування *UFW* на *Raspberry Pi* 4 *Model B*:

– оновлення пакетів та встановлення *ufw*:

```
sudo apt update
sudo apt install ufw -y
```

– скидання правил, необхідно для налаштування конфігурації з нуля:

```
sudo ufw reset
```

– застосовуємо політики за замовчуванням, блокуємо усі вхідні підключення та дозволяє всі вихідні:

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
```

– дозволяємо необхідні порти:

sudo ufw allow 22*/tcp*
sudo ufw allow 443*/tcp*

– запускаємо ufw:

sudo ufw enable

Для покращення захисту можна скористатися утилітою *Fail2Ban*. *Fail2Ban* – це програмний засіб захисту серверів і мережевих сервісів від атак типу *brute-force* та інших несанкціонованих спроб доступу. Інструмент широко використовується в *Linux*-системах (*Debian*, *Ubuntu*, *Raspberry Pi OS*, *CentOS* тощо) і працює як фоновий сервіс.

Fail2Ban аналізує лог-файли системи та сервісів (наприклад, *SSH*, вебсервери, поштові служби) і відстежує підозрілу активність,:

– багаторазові невдалі спроби входу;

– помилки автентифікації;

– запити, характерні для автоматизованих атак.

При перевищенні встановленого порогу (кількість спроб за певний час) *Fail2Ban* автоматично застосовує санкції до джерела атаки:

– тимчасово або постійно блокує *IP*-адресу через *iptables* або *UFW*;

– припиняє доступ до відповідного сервісу.

Але в даній реалізації *Fail2Ban* не використовується, замість даної утиліти - обмежено доступ до *SSH* з одного *IP* у внутрішній мережі (для роботи у режимі хотспот – окремий *IP*, для під’єднання достатньо налаштувати цей *IP* власноруч у налаштуваннях мережевого адаптера ПК).

– для роботи з загальною мережею:

sudo ufw allow from 192.168.0.200 *to any port* 22 *proto tcp*

– для роботи у режимі хотспот:

sudo ufw allow from 192.168.1.200 *to any port* 22 *proto tcp*

Така конфігурація забезпечить досить високий рівень захисту доступу до веб-порталу системи.

3.3 Попередні налаштування *Raspberry Pi* 4 *Model B*

Для забезпечення безперервного доступу до захищеної системи віддаленого контролю кліматичних параметрів у разі втрати основного Інтернет-з'єднання розроблено спеціальний скрипт на мові *Python* (*network_manager_script.py*). Скрипт реалізує механізм автоматичного перемикання між основною *Wi-Fi*-мережею та режимом точки доступу (*Hotspot*), а також забезпечує постійний запуск веб-сервера на базі *Uvicorn*. Це дозволяє підтримувати доступ до системи навіть при тимчасовій відсутності зовнішнього Інтернету. Повний скрипт наведено у Додатку. У цьому розділі будуть наведені лише основні блоки з коментарями.

Наведений нижче код (лістинг 3.1) відповідає за встановлення змінних параметрів для коректної роботи системи, саме в цьому блоці задаються параметри існуючої мережі, хотспоту, пул адресів, що буде використовуватися у режимі хотспоту та шляхи до виконуваного файлу з веб-порталом та керуванням платою.

Лістинг 3.1 –

```
# --- Конфігурація МЕРЕЖІ ---
TARGET_SSID = «TP-Link_C2D4»
TARGET_PASSWORD = «45404580»
HOTSPOT_SSID = «PiHotspot»
HOTSPOT_PASSWORD = «HotspotPassword123»
HOTSPOT_IP = «192.168.1.181/24»
ETH_INTERFACE = «eth0»
WIFI_INTERFACE = «wlan0»
PING_HOST = «8.8.8.8»

# --- Конфігурація ДОДАТКА ---
# Розширення шляху до домашньої папки (~/project/env)
#VENV_PATH = os.path.expanduser(«/home/user/project/env»)
PROJECT_PATH = «/home/user/project»
VENV_PATH = f»{PROJECT_PATH}/env»
UVICORN_EXE = f»{VENV_PATH}/bin/uvicorn» # Повний шлях до виконуваного файлу uvicorn
UVICORN_CMD_ARGS = f»app:app –host 0.0.0.0 –port 443»
SCAN_INTERVAL = 30 # Інтервал сканування цільової мережі, коли Hotspot активний (сек)

# --- Глобальні змінні для процесів ---
uvicorn_process = None
# ---------------------
```

Скрипт (лістинг 3.2) використовує інструмент *nmcli* (*NetworkManager command-line interface*) для керування мережевими підключеннями та виконується з правами *root*. Головний цикл (*main_loop*) періодично перевіряє наявність Інтернету (*ping* до 8.8.8.8), намагається підключитися до цільової *Wi-Fi*-мережі та, у разі невдачі, активує *Hotspot*. При активації *Hotspot* скрипт сканує наявність цільової мережі кожні 30 секунд і автоматично перемикається назад при її появі. Незалежно від стану мережі скрипт забезпечує запуск *Uvicorn*-процесу для веб-додатка. Нижче наведено лістинг головного циклу:

Лістинг 3.2 – Скрипт керування мережею

```
def main_loop():
    «»»Головний цикл перевірки та управління мережею.»»»

    Run_command(«sudo pigpiod»)
    hotspot_active = False
    last_scan_time = 0

    print(«Запуск мережевого менеджера.»)

    # Спробувати підключитися до цільової мережі при запуску
    if connect_wifi(TARGET_SSID, TARGET_PASSWORD):
        print(f»Стартове підключення до {TARGET_SSID} успішне.»)

    try:
        while True:
            # Перевіряємо, чи є Інтернет
            is_internet_ok = check_internet_connection(ETH_INTERFACE) or
check_internet_connection(WIFI_INTERFACE)

            # --- СЦЕНАРІЙ 1: Інтернет працює ---
            if is_internet_ok:
                if hotspot_active:
                    print(«Інтернет відновлено. Вимикаємо Hotspot.»)
                    stop_hotspot()
                    hotspot_active = False
                else:
                    print(«Інтернет-з’єднання активне.»)

                # Якщо ми не підключені до цільового Wi-Fi, намагаємося підключитися
                is_target_active = run_command(f»nmcli con show –active | grep -w ‘{TARGET_SSID}’»)[2]
== 0
                if not is_target_active:
                    connect_wifi(TARGET_SSID, TARGET_PASSWORD)

            # --- СЦЕНАРІЙ 2: Інтернет відсутній ---
            else:
                print( Інтернет-з’єднання відсутнє.»)
```

```
                if hotspot_active:
                    current_time = time.time()
                    if current_time – last_scan_time >= SCAN_INTERVAL:
                        last_scan_time = current_time

                        if scan_for_target_wifi(TARGET_SSID):
                            # Цільова мережа знайдена! Перемикаємося.
                            Print(«Цільова мережа з’явилася! Вимкнення Hotspot та підключення.»)
                            stop_hotspot()
                            hotspot_active = False
                            # Логіка повернеться до «Hotspot неактивний» і спробує підключитися

                        else:
                            print(f»Hotspot активний. Цільова мережа не знайдена. Сканування через {SCAN_INTERVAL} сек.»)

                # Якщо Hotspot неактивний
                if not hotspot_active:
                    if connect_wifi(TARGET_SSID, TARGET_PASSWORD):
                        print(«Успішно підключено до цільового Wi-Fi.»)
                        continue

                    # Якщо підключення не вдалося – запускаємо Hotspot
                    if start_hotspot():
                        hotspot_active = True

            # --- ОСНОВНА ВИМОГА: Запуск додатка Uvicorn у будь-якому сценарії ---
            start_uvicorn_app()

            time.sleep(5) # Пауза перед наступною перевіркою

        except KeyboardInterrupt:
            print(«\nОтримано сигнал зупинки (Ctrl+C). Завершення роботи…»)
        finally:
            # Очищення при виході
            stop_uvicorn_app()
            if hotspot_active:
                stop_hotspot()
            print(«Скрипт завершено.»)
```

```
if __name__ == «__main__»:
    if sys.platform.startswith('linux'):
        print(«!!! Для коректної роботи необхідні права root (sudo) та встановлений NetworkManager (nmcli).»)
        main_loop()
    else:
        print(«Цей скрипт призначений для виконання на Linux-системах.»)
```

Поданий скрипт має бути під'єднано як сервіс до *systemd* для того, щоб він запускався після запуску *Raspberry Pi* 4. Необхідно створити файл з поданим скриптом за шляхом */usr/local/bin/network_manager_script.py* та надати йому права на виконання командою:

sudo chmod +x /usr/local/bin/network_manager_script.py

Наступним кроком створюється файл служби *systemd* за шляхом - */etc/systemd/system/network_manager_script.service*, з наступним вмістом :

```
[Unit]
Description=Raspberry Pi Network Manager Script
After=network-online.target wpa_supplicant.service
Wants=network-online.target

[Service]
# Тип служби. «simple» означає, що процес запускається одразу.
Type=simple

# Шлях до інтерпретатора Python та скрипта.
# Використовуйте повний шлях до python3 (зазвичай /usr/bin/python3).
ExecStart=/usr/bin/python3 /usr/local/bin/network_manager_script.py

# Робочий каталог
WorkingDirectory=/usr/local/bin/

# Виконання з правами root (необхідно для nmcli)
```

```
User=root
Group=root

# Перезапуск у разі збою
Restart=always
RestartSec=10

[Install]
# Запуск служби під час завантаження
WantedBy=multi-user.target
```

Після виконаних маніпуляція необхідно перезавантажити плату та перевірити стан служби. Якщо все виконано правильно – після перезапуску система під ’єднається до мережі самостійно або створить власну мережу.

3.4 Реалізація програмної частини

Розробка програмного забезпечення захищеної системи віддаленого контролю кліматичних параметрів серверних приміщень виконана на мові *Python* з використанням фреймворку *FastAPI* для створення веб-інтерфейсу та *API*. На рисунку 3.6 показано алгоритм за яким запускається веб-портал.

Основний файл додатка (app.py) реалізує аутентифікацію, авторизацію, моніторинг датчиків, керування виконавчими пристроями та логування подій. Код забезпечує graceful fallback для апаратних бібліотек, що дозволяє тестування на некритичних машинах.

Залежності можна встановити однією командою:

pip install fastapi uvicorn jinja2 passlib pyotp qrcode[pil] piir adafruit-circuitpython-dht Rpi.GPIO adafruit_pn532 pigpiod

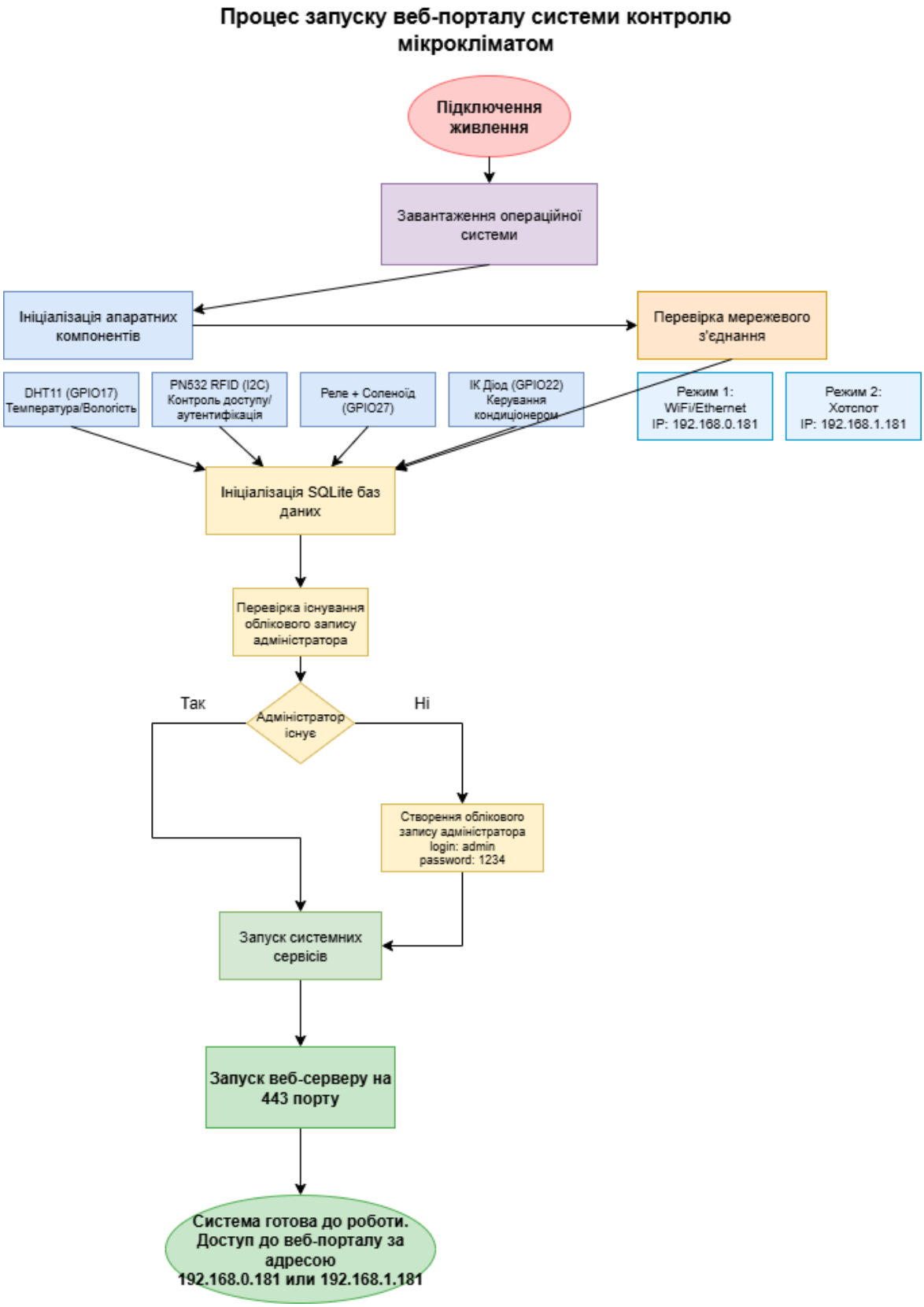


Рисунок 30 – Процес запуску веб-порталу

Проєкт програмної частини має наступну структуру:

project /
    env / #Віртуальне середовище
    static /
        chart.js #Копія біблотеки для побудоваи графіків у разі роботи у режимі хотспоту
        style.css #Файл з основними стилями для всіх шаблонів
    templates / #Папка з шаблонами сторінок веб-порталу
        admin_users.html
admin_user_create.html
show_qr.html
403.html
admin_user_edit.html
index.html
dashboard.html
totp.html
admin_logs.html
login.html
base.html
404.html
    app.py #
    logs.db
    app.log
    light.json
    users.db

В даному розділі будуть розглядатися лише визначні блоки з точки зору безпеки. Повний код файлу буде наведено у Додатку. Додаток базується на *FastAPI* для асинхронної обробки запитів, *Jinja2* для шаблонів, *SQLite* для зберігання даних користувачів та логів. Використовуються бібліотеки: *pyotp* (*TOTP*), *passlib* (хешування паролів), *qrcode* (генерація *QR*-кодів), *piir* (ІЧ-керування), *adafruit_dht* та *Rpi.GPIO* (апаратна інтеграція з *fallback*). Конфігурація включає *SECRET_KEY* з середовища для безпеки сесій.

Програмну частину можна поділити на декілька блоків:

– імпорти та конфігурація: Визначаються шляхи до баз даних, пінів *GPIO* та секретного ключа. Логування налаштовано на файл та консоль;

```
# -------- CONFIGURATION --------
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
DB_PATH = os.path.join(BASE_DIR, «users.db»)
LOG_PATH = os.path.join(BASE_DIR, «logs.db»)
LOG_FILE = os.path.join(BASE_DIR, «app.log»)

DHT_PIN = 17      # GPIO17 for DHT11
RELAY_PIN = 27    # GPIO27 for relay(solenoid)
IR_PIN = 22       # GPIO22 for IR LED (carrier generation)
I2C_SCL = None
I2C_SDA = None

SECRET_KEY = os.environ.get(«APP_SECRET_KEY», «change_this_secret_in_prod»)
SESSION_KEY = SECRET_KEY  # for session middleware
```

– бази даних та ініціалізація: Функції *init_db* створюють таблиці *users* (з полями для ролі, *RFID*, *TOTP*) та *logs*. *Create_admin_if_missing* генерує дефолтного адміністратора.

```
# ---------- DB helpers ----------
def get_db_connection(db):
    if db == «main»:
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        return conn
    elif db==»logs»:
        conn = sqlite3.connect(LOG_PATH)
        conn.row_factory = sqlite3.Row
        return conn

def init_db(db):
    if db==»main»:
        conn = get_db_connection(«main»)
        cur = conn.cursor()
```

```
            cur.execute(«»»
            CREATE TABLE IF NOT EXISTS users(
                id INTEGER PRIMARY KEY,
                username TEXT UNIQUE NOT NULL,
                password_hash TEXT NOT NULL,
                role TEXT NOT NULL DEFAULT 'user', -- 'admin' or 'user'
                rfid_uid TEXT,
                totp_secret TEXT,
                totp_backup TEXT,
                created_at TEXT
            )
            «»»)
            conn.commit()
            conn.close()
        elif db==»logs»:
            conn = get_db_connection(«logs»)
            cur = conn.cursor()
            cur.execute(«»»
            CREATE TABLE IF NOT EXISTS logs(
                id INTEGER PRIMARY KEY,
                ts TEXT,
                level TEXT,
                message TEXT
            )
            «»»)
            conn.commit()
            conn.close()

    def create_admin_if_missing():
        conn = get_db_connection(«main»)
        cur = conn.cursor()
        cur.execute(«SELECT * FROM users WHERE username = ?», («admin»,))
        if not cur.fetchone():
            pwd = «1234»
            secret = pyotp.random_base32()
            backup = «BACKUP-« + pyotp.random_base32()[:10]
            cur.execute(
                «INSERT INTO users(username, password_hash, role, totp_secret, totp_backup, created_at)
VALUES(?,?,?,?,?,?)»,
                («admin», pwd_context.hash(pwd), «admin», secret, backup, datetime.utcnow().isoformat())
```

```
        )
        conn.commit()
        logger.info(«Created default admin (login: admin, password: 1234)»)
            print(«Created default admin (login: admin, password: 1234, secret {secret}, backup {backup})».format(secret=secret, backup=backup))
        conn.close()
```

– логування подій: *log_event* додає підпис користувача до повідомлень для аудиту:

```
Def log_event(level: str, message: str, user: Optional[Dict[str, Any]] = None):
    # Якщо передано користувача, додаємо підпис
    if user and ‘username’ in user:
        # User: Admin (пример)
        message = f»{message} User: {user[‘username’].capitalize()}»

    conn = get_db_connection(«logs»)
    cur = conn.cursor()
    cur.execute(«INSERT INTO logs(ts, level, message) VALUES(?,?,?)», (datetime.utcnow().isoformat(), level, message))
    conn.commit()
    conn.close()

    if level.lower() == «info»:
        logger.info(message)
    elif level.lower() == «warning»:
        logger.warning(message)
    else:
        logger.error(message)
```

– обгортки апаратної частини: Класи *DHTReader* (з ретраями), *RelayController* (з таймером авто-вимкнення та логуванням дій), *IRSender* (ІЧ-команди з *mock*-режимом), *RFIDMonitor* (фоновий потік для доступу за міткою);

– керування користувачами: Функції створення, оновлення, видалення з логуванням дій адміністратора;

– аутентифікація та сесії: Двофакторна (пароль + *TOTP*), обмеження сесії 300 с, рольовий доступ (*RBAC*);

```
Def require_login(request: Request):
    user = None
    if «user_id» in request.session:
        user = get_user_by_id(request.session[«user_id»])
    return user

def require_user_logged_in(request: Request):
    user = require_login(request)
    if not user:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail=»Not authenticated»)
    # Check session timeout (300 seconds)
    if «last_auth» in request.session:
        last_auth = datetime.fromisoformat(request.session[«last_auth»])
        if datetime.utcnow() – last_auth > timedelta(seconds=300):
            request.session.clear()
            raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail=»Session expired»)
    return dict(user) # Return as dict for easier usage

def require_admin(request: Request):
    user = require_user_logged_in(request)
    if user[«role»] != «admin»:
        raise HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail=»Admin required»)
    return user
```

Розроблене ПЗ реалізує захищений віддалений доступ з *MFA*, реальним часом даних та інтеграцією апаратної частини, перевищуючи функціональність попередньої реалізації.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Тестування функціональності та продуктивності

Для доступу до веб-порталу достатньо ввести адресу та порт за яким він розміщений. *IP* адреса може змінюватися в залежності від режиму роботи, а порт завжди буде 443 (*HTTPS*). Процес входу до веб-порталу детально показано на рис. 4.1.

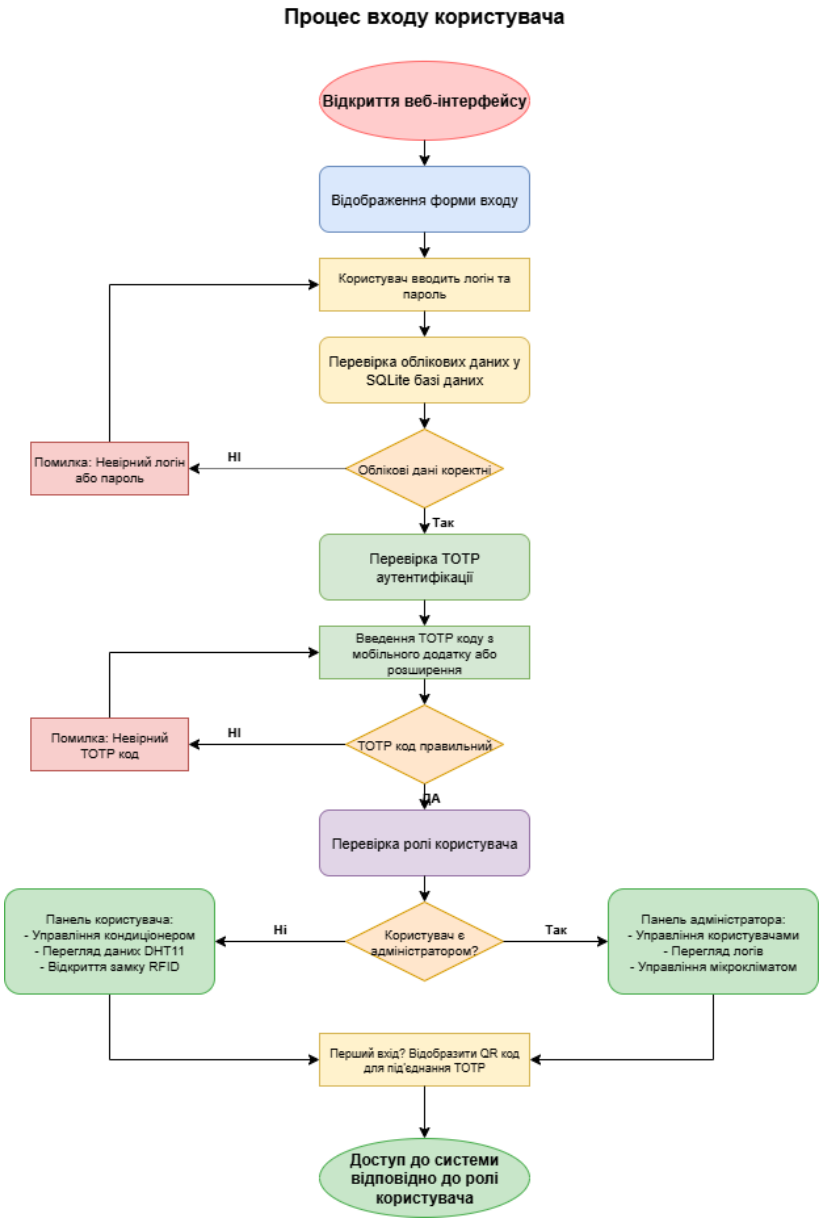


Рисунок 4.1 – Процес входу користувача до веб-порталу

При переході за адресом відкриється головна сторінка де нам запропонують пройти авторизацію (рис. 4.2, 4.3).

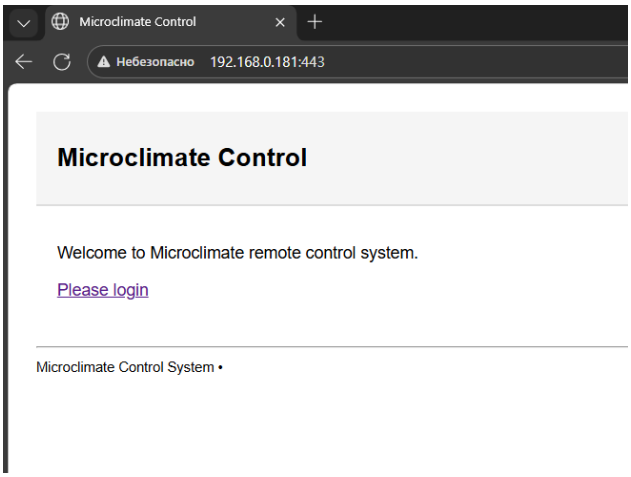


Рисунок 4.2 – Головна сторінка

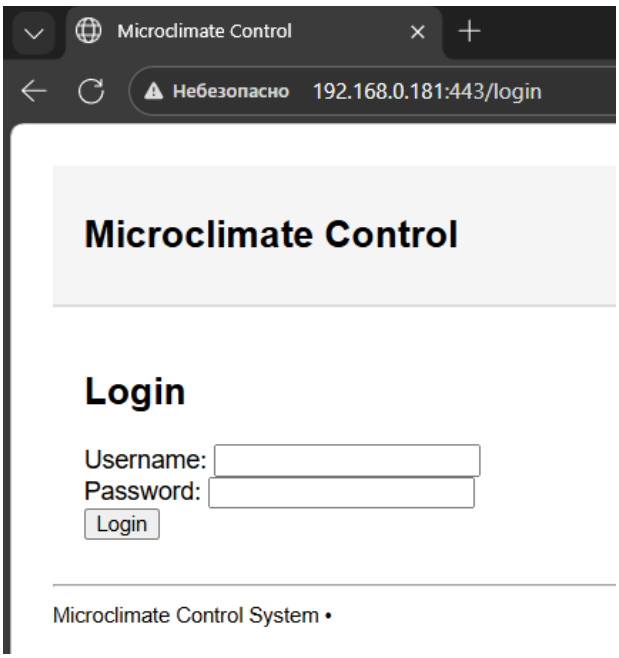


Рисунок 4.3 – Сторінка входу (логін/пароль)

Після вводу коректних даних для входу – відкриється сторінка двофакторної аутентифікації та необхідно буде ввести *TOTP* код (рис. 4.4).

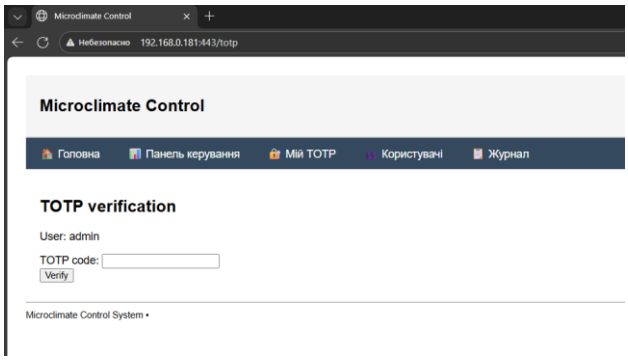


Рисунок 4.5 – Сторінка введення *TOTP* код

Після авторизації – відкриється сторінка з дашбордом де в реальному часі можна спостерігати та керувати кліматичними параметрами в приміщенні та системою контролю доступу (рис. 4.6). Звичайний користувач матиме змогу переглядати лише сторінку з власними даними для *TOTP* аутентифікації та керувати з дашборду модулями.

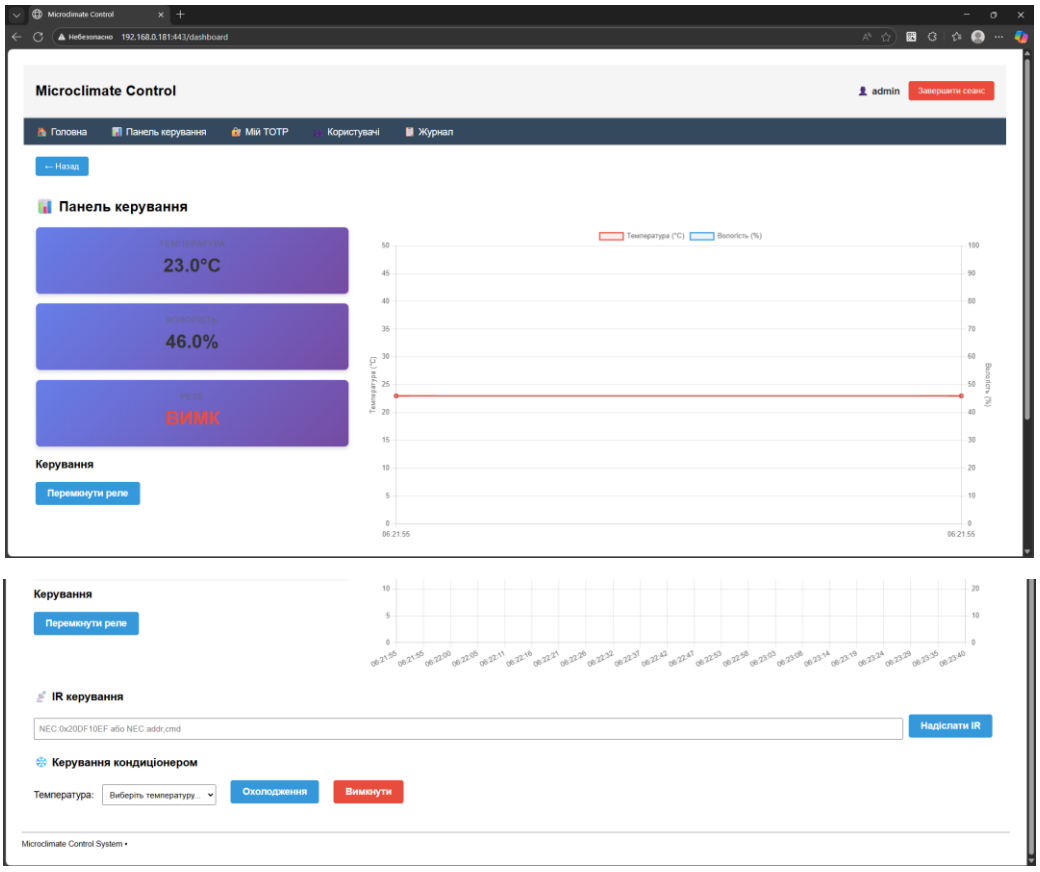


Рисунок 4.6 – Дашборд

Адміністратор має більші повноваження та доступ. На рисунках 4.7-4.10 показано сторінки перегляду логів та керування користувачами.

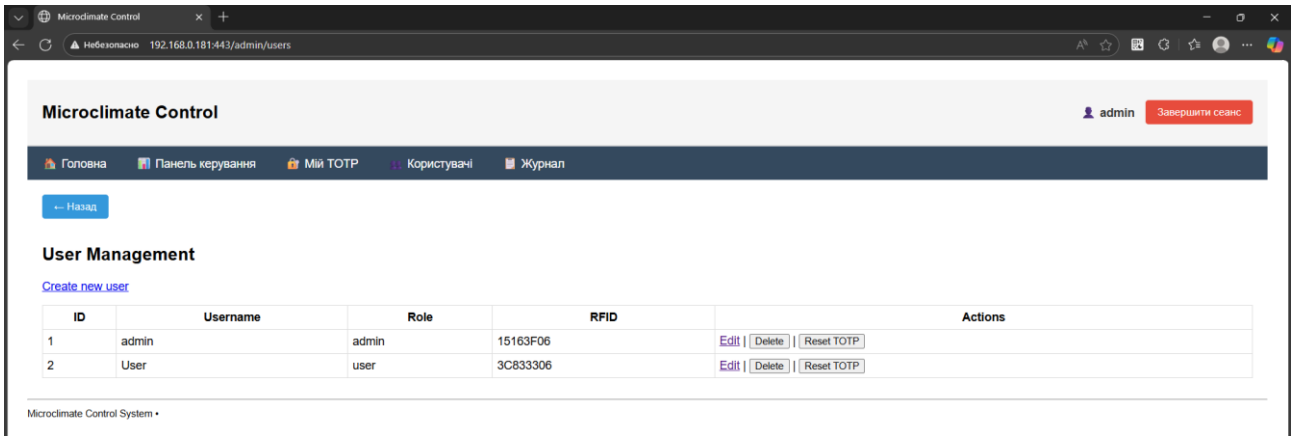


Рисунок 4.7 – Сторінка перегляду користувачів

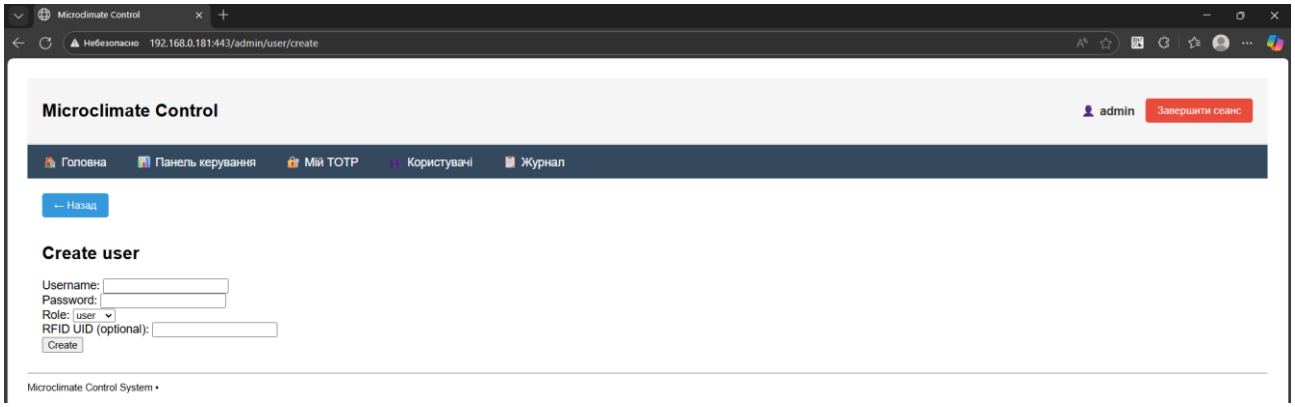


Рисунок 4.8 – Сторінка створення нового користувача

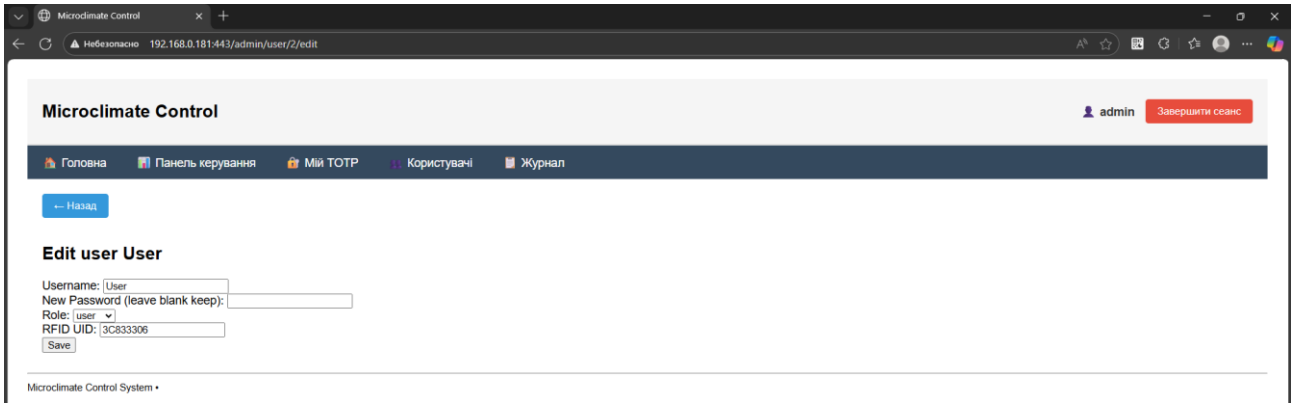


Рисунок 4.9 – Сторінка змінення даних існуючого користувача

Рисунок 4.10 – Сторінка скидання *TOTP* існуючого користувача

Керування користувачами реалізовано відповідно до *CRUD* (*Create*, *Read*, *Update*, *Delete*). Адміністратор має повноваження скидання *TOTP* аутентифікації для користувачів, приписувати *RFID* мітки та картки, створювати та змінювати логіни і паролі, змінювати ролі доступу.

Логування поділено на 2 блоки. Логування того, що виконується на порталі та цілком логування після запуску файлу. Логування того, що виконується на порталі ведеться у *SQLite* базу даних та доступне лише до перегляду з веб-порталу, що мінімізує можливість пошкодження цілісності логів. Кольором виділяються події, що мають різні критичності (рис. 4.11).

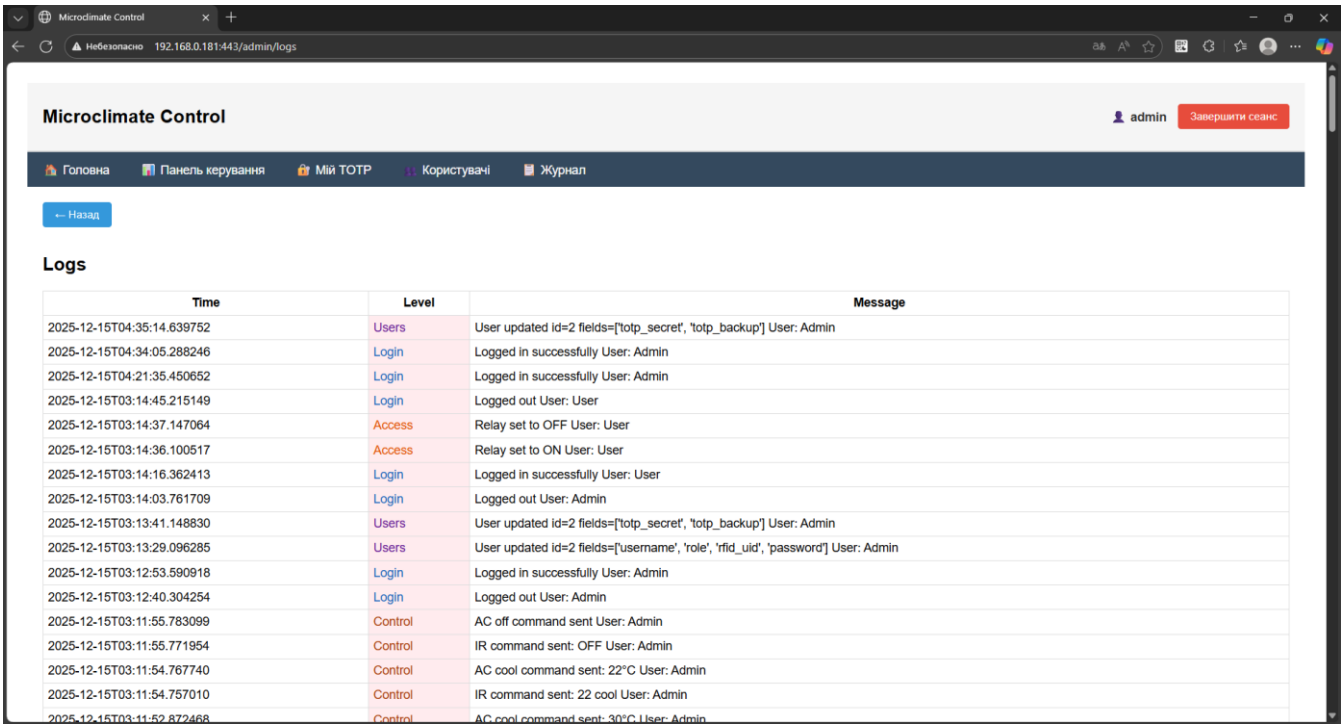


Рисунок 4.11 – Перегляд логів

4.2 Оцінка стійкості до типових кіберзагроз

Тестування безпеки програмного забезпечення захищеної системи віддаленого контролю кліматичних параметрів серверних приміщень проведено з метою перевірки стійкості до типових загроз веб-додатків. Тести включали аналіз вразливостей аутентифікації, ін’єкцій, керування сесіями та логування [31-35].

Перевірено механізм двофакторної аутентифкації (пароль + *TOTP*). Спроби *brute-force* на сторінці логіну (*/login*) та (*totp*) не вдалися через обмеження сесії (300 с) та відсутність *rate-limiting*.

Перевірка ролей (*RBAC*) підтвердила заборону доступу до адмін-панелі для звичайних користувачів, повертається сторінка з кодом 403 (рис. 4.12).

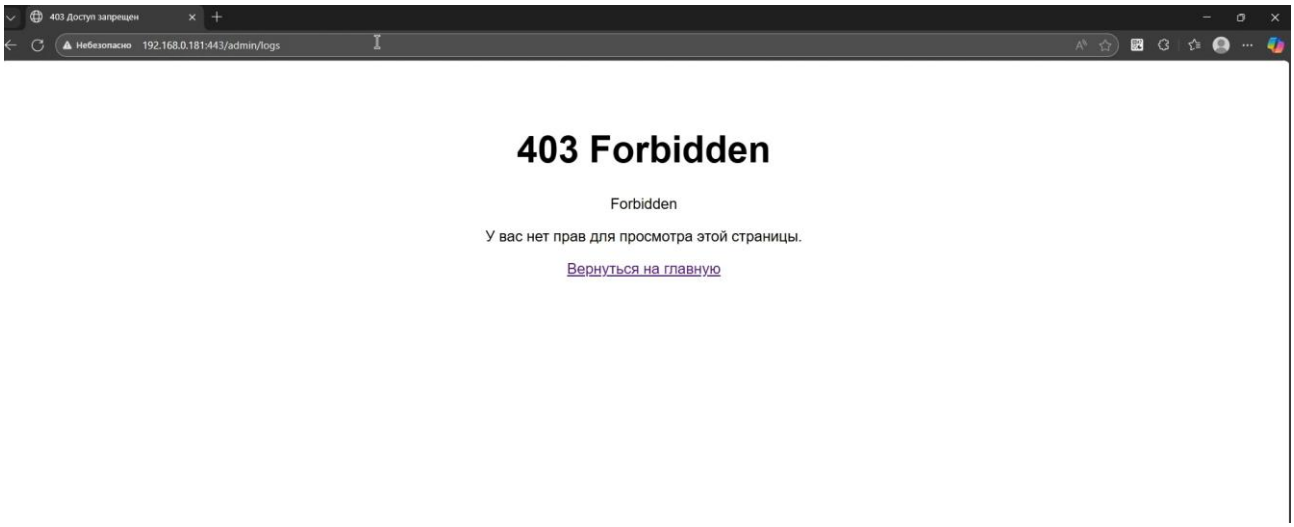


Рисунок 4.12 – Заборона доступу звичайному користувачу до адмін-панелі

Використано параметризовані запити *SQLite* (*sqlite*3 з ? *placeholders*) та утиліту *sqlmap*, що запобігає *SQL*-ін’єкціям. Тести з *payload*’ами (` *OR* 1=1 --) не призвели до несанкціонованого доступу (рис. 4.13)

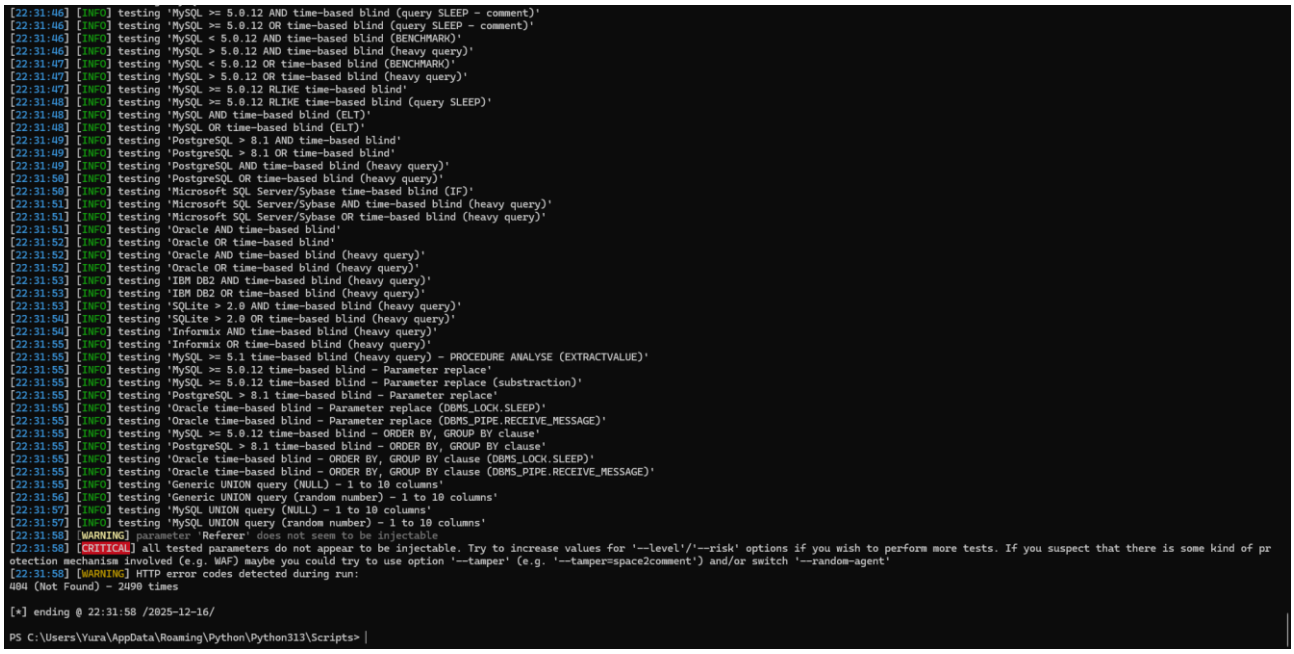


Рисунок 4.13 – Тестування за допомогою утиліти *sqlmap*

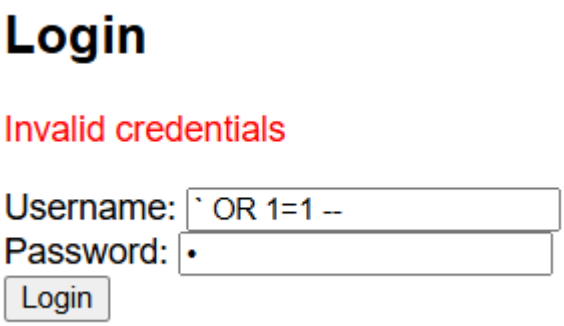


Рисунок 4.14 – Спроба несанкціонованого доступу

Сесії захищено секретним ключем та таймаутом. Логування всіх дій (*log_event* з підписом користувача) забезпечує аудит. Тести *session fixation* не виявили вразливостей.

Тестування не виявило критичних вразливостей, підтвердивши ефективність механізмів захисту.

ВИСНОВКИ

У кваліфікаційній роботі виконано удосконалення захищеної системи віддаленого контролю кліматичних параметрів серверних приміщень шляхом переходу з архітектури на базі *ESP*32/*Arduino* на платформу *Raspberry Pi* 4 *Model B*, мінімізації ланцюгів передачі даних та реалізації комплексного захисту на криптографічному, фізичному й адміністративному рівнях. Це дозволило усунути вразливості, притаманні попереднім рішенням, залежним від хмарних сервісів або з недостатнім захистом фізичного доступу.

Для досягнення мети вирішено такі задачі:

– проведено аналіз сучасних платформ для створення захищених систем віддаленого керування кліматичними параметрами та обґрунтовано перехід на *Raspberry Pi* 4 *Model B* завдяки вищій обчислювальній потужності, підтримці *Gigabit Ethernet*, вбудованого *Wi-Fi* та можливості локального розміщення вебпорталу без зовнішніх хмарних залежностей.

– розроблено архітектуру удосконаленої захищеної системи, що включає апаратну частину (датчики *DHT*22 для моніторингу температури та вологості, *RFID*/*NFC*-модуль *PN*532 для контролю фізичного доступу, соленоїдний замок *JF*-0826*B* 12 В, ІЧ-діод для керування кондиціонером) та програмне забезпечення (локальний вебпортал з двофакторною аутентифікацією на базі пароля та *TOTP*/*RFID*, шифруванням трафіку за протоколом *TLS* 1.3, розмежуванням прав доступу для ролей «користувач» і «адміністратор», логуванням подій).

– проведено експериментальні дослідження та оцінку захищеності системи, що підтвердило стабільність моніторингу кліматичних параметрів у діапазоні температур 10–40 °*C* та вологості 20–80 %, ефективність двофакторної аутентифікації, стійкість до спроб несанкціонованого доступу та відповідність вимогам нормативних документів з кібербезпеки.

Новизна роботи полягає в інтеграції комплексного захисту (криптографічного, фізичного та адміністративного) в автономну систему на базі *Raspberry Pi* 4 з мінімальними ланцюгами передачі даних, що зменшує поверхню атаки порівняно з хмарними рішеннями.

Практична значимість отриманих результатів полягає в можливості застосування розробленої системи на об’єктах критичної інфраструктури воднотранспортного комплексу та енергетичного сектору для забезпечення стабільності кліматичних параметрів серверних приміщень з обмеженим доступом. Результати роботи впроваджено в навчальному процесі кафедри комп’ютерних технологій та інформаційної безпеки Національного університету кораблебудування імені адмірала Макарова.

Подальші дослідження можуть бути спрямовані на інтеграцію механізмів автоматичного резервного живлення, розширення функціоналу за допомогою машинного навчання для прогнозування відхилень кліматичних параметрів та адаптацію системи до промислових протоколів типу *Modbus* для інтеграції з існуючими *SCADA*-системами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Річний звіт системи виявлення вразливостей і реагування на кіберінциденти та кібератаки 2024. Оперативний центр реагування на кіберінциденти Державного центру кіберзахисту Державної служби спеціального зв'язку та захисту інформації України. *URL*: https://cert.gov.ua/article/17696 (дата звернення: 01.12.2025).

2. Закон України «Про основні засади забезпечення кібербезпеки України» від 05.10.2017 № 2163-*VIII*. *URL*: https://zakon.rada.gov.ua/laws/show/2163-19 (дата звернення: 01.12.2025).

3. *Nannipieri P., Crocetti L., Di Matteo S., Fanucci L., Saponara S. Hardware design of an advanced-feature cryptographic tile within the european processor initiative*. *IEEE Transactions on Computers*. 2023. *Early access*. *DOI*: 10.1109/*TC*.2023.3278536. *URL*: https://ieeexplore.ieee.org/document/10130378 (дата звернення: 01.12.2025).

4. *ENISA. Good Practices for Security of IoT – Secure Software Development Lifecycle*. 2020. *URL*: https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot-1 (дата звернення: 01.12.2025).

5. *NIST Special Publication* 800-213. *IoT Device Cybersecurity Guidance for the Federal Government*: *Establishing IoT Device Cybersecurity Requirements*. 2021. *URL*: https://csrc.nist.gov/pubs/sp/800/213/final (дата звернення: 01.12.2025).

6. Закон України «Про захист інформації в інформаційно-телекомунікаційних системах» від 05.07.1994 № 80/94-ВР. *URL*: https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80 (дата звернення: 01.12.2025)

7. Постанова КМУ від 29.03.2006 № 373. *URL*: https://www.kmu.gov.ua/npas/32791826 (дата звернення: 01.12.2025)

8. Критерії оцінки захищеності інформації в комп'ютерних системах від несанкціонованого доступу : НД ТЗІ 2.5-004-99. Київ : ДСТСЗІ СБУ, 1999.

9. Класифікація автоматизованих систем і стандартні функціональні профілі захищеності оброблюваної інформації від несанкціонованого доступу : НД ТЗІ 2.5-005-99. Київ : ДСТСЗІ СБУ, 1999.

10. Порядок проведення робіт із створення комплексної системи захисту інформації в інформаційно-телекомунікаційній системі : НД ТЗІ 3.7-003-05. Київ : Держспецзв'язку, 2005.

11. Інформаційні технології. Техніки безпеки. Системи управління інформаційною безпекою. Вимоги : ДСТУ *ISO/IEC* 27001:2023. *URL*: *https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=104398* (дата звернення: 10.11.2025).

12. *IoT Device Cybersecurity Guidance for the Federal Government* : *NIST Special Publication* 800-213. *National Institute of Standards and Technology*, 2020. *URL*: *https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-213.pdf* (дата звернення: 09.11.2025).

13. *Considerations for Managing Internet of Things* (*IoT*) *Cybersecurity and Privacy Risks* : *NISTIR* 8228. *National Institute of Standards and Technology*, 2019. *URL*: *https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8228.pdf* (дата звернення: 05.11.2025).

14. *Security for industrial automation and control systems* : *IEC* 62443 *series*. *International Electrotechnical Commission*.

15. *IoT Non-Cellular Device Cybersecurity Requirement Catalog* : *NIST Special Publication* 800-213*A*. *National Institute of Standards and Technology*, 2021. *URL*: *https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-213A.pdf* (дата звернення: 16.12.2025).

16. *CYBER*; *Cyber Security for Consumer Internet of Things*: *Baseline Requirements* : *ETSI EN* 303 645. *European Telecommunications Standards Institute*, 2020.

17. *Regulation* (*EU*) 2016/679 *of the European Parliament and of the Council of* 27 *April* 2016 *on the protection of natural persons with regard to the processing of personal data and on the free movement of such data* (*General Data Protection Regulation*) : *GDPR*. *URL*: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679 (дата звернення: 07.11.2025).

18. *ASHRAE TC* 9.9. *Data Center Networking Equipment – Issues and Best Practices*. 2023.

19. *Schneider Electric*. *NetBotz Environmental Monitoring*. *URL*: https://www.se.com/us/en/product-range/7600-netbotz/ (дата звернення: 01.10.2025)

20. *APC by Schneider Electric NetBotz Monitoring System*. *URL*: https://www.apc.com/us/en/product-range/7600-netbotz/ (дата звернення: 01.10.2025)

21. *Vertiv Geist Environmental Monitors*. *URL*: https://www.vertiv.com/en-us/products-catalog/monitoring-control-and-management/monitoring/vertiv-geist-environmental-monitors/ (дата звернення: 01.10.2025)

22. *Vertiv Geist Watchdog Series*. *URL*: https://www.vertiv.com/en-us/products-catalog/monitoring-control-and-management/monitoring/watchdog-15/ (дата звернення: 01.10.2025)

23. *Rittal CMC III Monitoring System*. *URL*: https://www.rittal.com/us-en_US/products/PG0800ITINFRA1/PG1538ITINFRA1/PGR9560ITINFRA1 (дата звернення: 01.10.2025)

24. *Rittal CMC III Processing Unit*. *URL*: https://www.rittal.com/com-en/products/PG20231215ZUB101/PG20240405ZUB001/PG20240405ZUB002/PRO23677 (дата звернення: 01.10.2025)

25. *Raspberry Pi 4 Model B specifications*. *URL*: *https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/* (дата звернення: 01.10.2025)

26. *Orange Pi 5 Review and Comparison*. *URL*: *https://www.zdnet.com/article/best-raspberry-pi-alternative/* (дата звернення: 01.10.2025)

27. *BeagleBone Black vs Raspberry Pi*. *URL*: *https://mender.io/blog/beaglebone-vs-raspberry-pi* (дата звернення: 01.10.2025)

28. *ODROID N2+ Specifications*. *URL*: *https://www.hardkernel.com/shop/odroid-n2-with-4gbyte-ram-2/* (дата звернення: 01.10.2025)

29. *NVIDIA Jetson Nano Developer Kit*. *URL*: *https://developer.nvidia.com/embedded/jetson-nano* (дата звернення: 01.10.2025)

30. Стефанюк Ю. О. Захищена система віддаленого контролю кліматичних параметрів серверних приміщень : пояснювальна записка до кваліфікаційної роботи бакалавра. Миколаїв : НУК, 2024.

31. Тестування безпеки: SQL-ін'єкції. : веб-сайт. URL: https://training.qatestlab.com/blog/technical-articles/security-testing-sql-injection/

32. Захист веб-додатків від SQL-ін'єкцій: веб-сайт. URL: https://openarchive.nure.ua/server/api/core/bitstreams/4bc35e97-c6a3-4155-bc68-70591fe4fd27/content

33. Stuttard, D., & Pinto, M. The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws (2nd ed.). Wiley. 2011. 770 p.

34. Hack The Box: веб-сайт. URL: https://univd.edu.ua/uk/news/8668

35. Бурячок В. Л., Аносов А. О., Семко В. В., Соколов В. Ю., Складанний П. М. Технології забезпечення безпеки мережевої інфраструктури, К.: КУБГ, 2019. 218 с.

ДОДАТОК

Файл *app.py*

```
# app.py
import os
import io
import base64
import threading
import logging
import sqlite3
import asyncio
import json
import time
from typing import Optional, List, Dict, Any
from datetime import datetime, timedelta

from fastapi import FastAPI, Request, Form, Depends, HTTPException, status, Response, WebSocket, WebSocketDisconnect
from fastapi.responses import HTMLResponse, RedirectResponse, StreamingResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from fastapi.security import OAuth2PasswordBearer
from pydantic import BaseModel
from starlette.middleware.sessions import SessionMiddleware

# Crypto / Auth
import pyotp
import qrcode
from passlib.context import CryptContext

# IR control
try:
    import piir
except Exception:
    piir = None

# Try import hardware libs, fallback to mocks for dev machine
try:
    import board
    import adafruit_dht
except Exception as e:
    board = None
    adafruit_dht = None

try:
    import busio
    from adafruit_pn532.i2c import PN532_I2C
except Exception:
    PN532_I2C = None
    busio = None

try:
    import RPi.GPIO as GPIO
except Exception:
    GPIO = None
```

```
# -------- CONFIGURATION --------
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
DB_PATH = os.path.join(BASE_DIR, "users.db")
LOG_PATH = os.path.join(BASE_DIR, "logs.db")
LOG_FILE = os.path.join(BASE_DIR, "app.log")

DHT_PIN = 17      # GPIO17 for DHT11
RELAY_PIN = 27    # GPIO27 for relay(solenoid)
IR_PIN = 22       # GPIO22 for IR LED (carrier generation)
I2C_SCL = None
I2C_SDA = None

SECRET_KEY = os.environ.get("APP_SECRET_KEY", "change_this_secret_in_prod")
SESSION_KEY = SECRET_KEY  # for session middleware

# ---------- logging ----------
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
    handlers=[
        logging.FileHandler(LOG_FILE),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger("microclimate")

# ---------- FastAPI app ----------
app = FastAPI()
app.add_middleware(SessionMiddleware, secret_key=SESSION_KEY)
templates = Jinja2Templates(directory=os.path.join(BASE_DIR, "templates"))
app.mount("/static", StaticFiles(directory=os.path.join(BASE_DIR, "static")), name="static")

# ---------- password hashing ----------
pwd_context = CryptContext(schemes=["pbkdf2_sha256"], deprecated="auto")

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

# ---------- DB helpers ----------
def get_db_connection(db):
    if db == "main":
        conn = sqlite3.connect(DB_PATH)
        conn.row_factory = sqlite3.Row
        return conn
    elif db=="logs":
        conn = sqlite3.connect(LOG_PATH)
        conn.row_factory = sqlite3.Row
        return conn

def init_db(db):
    if db=="main":
        conn = get_db_connection("main")
        cur = conn.cursor()
        cur.execute("""
        CREATE TABLE IF NOT EXISTS users(
            id INTEGER PRIMARY KEY,
            username TEXT UNIQUE NOT NULL,
            password_hash TEXT NOT NULL,
            role TEXT NOT NULL DEFAULT 'user', -- 'admin' or 'user'
            rfid_uid TEXT,
            totp_secret TEXT,
```

```
            totp_backup TEXT,
            created_at TEXT
        )
        """)
        conn.commit()
        conn.close()
    elif db=="logs":
        conn = get_db_connection("logs")
        cur = conn.cursor()
        cur.execute("""
        CREATE TABLE IF NOT EXISTS logs(
            id INTEGER PRIMARY KEY,
            ts TEXT,
            level TEXT,
            message TEXT
        )
        """)
        conn.commit()
        conn.close()

def create_admin_if_missing():
    conn = get_db_connection("main")
    cur = conn.cursor()
    cur.execute("SELECT * FROM users WHERE username = ?", ("admin",))
    if not cur.fetchone():
        pwd = "1234"
        secret = pyotp.random_base32()
        backup = "BACKUP-" + pyotp.random_base32()[:10]
        cur.execute(
            "INSERT INTO users(username, password_hash, role, totp_secret, totp_backup,
created_at) VALUES(?,?,?,?,?,?)",
            ("admin", pwd_context.hash(pwd), "admin", secret, backup,
datetime.utcnow().isoformat())
        )
        conn.commit()
        logger.info("Created default admin (login: admin, password: 1234)")
        print("Created default admin (login: admin, password: 1234, secret {secret}, backup
{backup})".format(secret=secret, backup=backup))
    conn.close()

# --- ЗМІНЕНО: Додано аргумент user для підпису дій ---
def log_event(level: str, message: str, user: Optional[Dict[str, Any]] = None):
    # Якщо передано користувача, додаємо підпис
    if user and 'username' in user:
        # User: Admin (пример)
        message = f"{message} User: {user['username'].capitalize()}"

    conn = get_db_connection("logs")
    cur = conn.cursor()
    cur.execute("INSERT INTO logs(ts, level, message) VALUES(?,?,?)",
(datetime.utcnow().isoformat(), level, message))
    conn.commit()
    conn.close()

    if level.lower() == "info":
        logger.info(message)
    elif level.lower() == "warning":
        logger.warning(message)
    else:
        logger.error(message)
```

```
# ---------- Hardware wrappers (with graceful fallback) ----------
class DHTReader:
    def __init__(self, pin=DHT_PIN):
        self.pin = pin
        self._dht = None
        self.last_reading = {"temperature": 25.0, "humidity": 45.0, "ts":
datetime.utcnow().isoformat()}
        if adafruit_dht and board:
            try:
                # adafruit_dht library expects board pin objects
                self._dht = adafruit_dht.DHT11(getattr(board, f"GPIO{pin}", None) or getattr(board,
"D17", None))
            except Exception as e:
                logger.warning(f"DHT init failed: {e}")
                self._dht = None

    def read(self):
        if self._dht:
            max_retries = 3
            for attempt in range(max_retries):
                try:
                    t = self._dht.temperature
                    h = self._dht.humidity
                    reading = {"temperature": t, "humidity": h, "ts": datetime.utcnow().isoformat()}
                    self.last_reading = reading
                    return reading
                except Exception as e:
                    if attempt < max_retries - 1:
                        logger.debug(f"DHT read attempt {attempt + 1} failed: {e}. Retrying...")
                        time.sleep(1)
                    else:
                        logger.warning(f"DHT read failed after {max_retries} attempts: {e}. Using
cached value.")
            return self.last_reading
        else:
            return {"temperature": 25.0, "humidity": 45.0, "ts": datetime.utcnow().isoformat()}

class RelayController:
    def __init__(self, pin=RELAY_PIN):
        self.pin = pin
        self.state = False
        self._timer = None
        if GPIO:
            try:
                GPIO.setmode(GPIO.BCM)
                GPIO.setup(self.pin, GPIO.OUT)
                GPIO.output(self.pin, GPIO.LOW)
            except Exception as e:
                logger.warning(f"Relay GPIO init failed: {e}")

    # --- ЗМІНЕНО: Додано аргумент user ---
    def set(self, on: bool, user: Optional[Dict] = None):
        if self._timer:
            self._timer.cancel()
            self._timer = None

        self.state = on
        if GPIO:
            try:
```

```
                GPIO.output(self.pin, GPIO.HIGH if on else GPIO.LOW)
            except Exception as e:
                logger.warning(f"Relay GPIO write failed: {e}")

        # Логируем действие с привязкой к пользователю
        action = 'ON' if on else 'OFF'
        log_event("Access", f"Relay set to {action}", user=user)

    def _auto_off(self):
        """Допоміжна функція, яку викличе таймер (системна подія)"""
        # Тут user не передаємо, тому що це дія системи (таймер)
        # Але щоб було зрозуміло, можна явно передати {'username': 'System'} або
залишити None
        log_event("Access", "Timer expired: Turning relay OFF automatically.")

        self.state = False
        if GPIO:
            try:
                GPIO.output(self.pin, GPIO.LOW)
            except Exception:
                pass

    # --- ЗМІНЕНО: Додано аргумент user ---
    def set_timed(self, seconds: float, user: Optional[Dict] = None):
        # 1. Включаємо реле (логуємо користувача)
        self.set(True, user=user)

        # 2. Запускаємо таймер
        self._timer = threading.Timer(seconds, self._auto_off)
        self._timer.start()

    # --- ЗМІНЕНО: Додано аргумент user ---
    def toggle(self, user: Optional[Dict] = None):
        if self.state:
            self.set(False, user=user)
        else:
            self.set_timed(15.0, user=user)

class IRSender:
    """IR sender using piir library"""

    def __init__(self, pin=IR_PIN):
        self.pin = pin
        self.remote = None
        self.config_file = os.path.join(BASE_DIR, "light.json")

        if piir:
            try:
                self.remote = piir.Remote(self.config_file, self.pin)
                logger.info(f"IR Remote initialized with piir from {self.config_file}")
            except Exception as e:
                logger.warning(f"IR Remote initialization failed: {e}")
                self.remote = None
        else:
            logger.info("piir not available, IR will run in MOCK mode")

    def send_command(self, command: str, user: Optional[Dict] = None):
        """Send command using piir"""
        if not self.remote:
            log_event("Control", f"[MOCK] IR command sent: {command}", user=user)
```

```
            return True

        try:
            self.remote.send(command)
            log_event("Control", f"IR command sent: {command}", user=user)
            return True
        except Exception as e:
            logger.error(f"IR send error: {e}")
            log_event("error", f"IR send error: {str(e)}", user=user)
            return False

    def send_nec(self, addr, cmd, user: Optional[Dict] = None):
        """Legacy NEC format support"""
        log_event("Control", f"[MOCK] IR sent addr={addr} cmd={cmd}", user=user)
        return True

    def ac_cool(self, temp: int, user: Optional[Dict] = None) -> bool:
        """Send AC cool command for specified temperature (16-30)"""
        if temp < 16 or temp > 30:
            return False

        command = f"{temp} cool"

        if self.send_command(command, user=user):
            log_event("Control", f"AC cool command sent: {temp}°C", user=user)
            return True
        return False

    def ac_off(self, user: Optional[Dict] = None) -> bool:
        """Send AC off command"""
        if self.send_command("OFF", user=user):
            log_event("Control", f"AC off command sent", user=user)
            return True
        return False

# ---------- RFID background monitor ----------
class RFIDMonitor(threading.Thread):
    def __init__(self, pn532=None, db_path=DB_PATH, relay: RelayController=None):
        super().__init__(daemon=True)
        self.pn532 = pn532
        self.db_path = db_path
        self.relay = relay
        self.running = True

    def run(self):
        logger.info("RFID monitor started")
        while self.running:
            uid = self.read_uid()
            if uid:
                logger.info(f"RFID seen UID: {uid}")
                conn = get_db_connection("main")
                cur = conn.cursor()
                cur.execute("SELECT * FROM users WHERE rfid_uid = ?", (uid,))
                row = cur.fetchone() # Отримуємо користувача за картою
                conn.close()
                if row:
                    user_data = dict(row)
                    # --- ЗМІНЕНО: Передаємо знайденого користувача в логер ---
                    log_event("Access", f"RFID matched UID {uid}. Activating relay.",
user=user_data)
```

```
                if self.relay:
                    # Передаємо користувача в реле, щоб там теж залогувалося
                    self.relay.set_timed(15.0, user=user_data)
            else:
                log_event("Access", f'RFID unknown UID {uid} denied.")
        time.sleep(0.5)

    def read_uid(self) -> Optional[str]:
        if PN532_I2C and busio and board:
            try:
                if not self.pn532:
                    i2c = busio.I2C(board.SCL, board.SDA)
                    self.pn532 = PN532_I2C(i2c)
                uid = self.pn532.read_passive_target(timeout=0.5)
                if uid:
                    return ''.join('{:02X}'.format(b) for b in uid)
            except Exception as e:
                logger.warning(f'PN532 read error: {e}")
                return None
        return None

    def stop(self):
        self.running = False

# ---------- App-level singletons ----------
dht = DHTReader()
relay = RelayController()
ir = IRSender()
rfid_monitor = RFIDMonitor(relay=relay)

# ---------- Utilities: users management ----------
def get_user_by_username(username: str):
    conn = get_db_connection("main")
    cur = conn.cursor()
    cur.execute("SELECT * FROM users WHERE username = ?", (username,))
    r = cur.fetchone()
    conn.close()
    return r

def get_user_by_id(uid: int):
    conn = get_db_connection("main")
    cur = conn.cursor()
    cur.execute("SELECT * FROM users WHERE id = ?", (uid,))
    r = cur.fetchone()
    conn.close()
    return r

def verify_password(plain, hashed):
    return pwd_context.verify(plain, hashed)

# --- ЗМІНЕНО: Додано аргумент performed_by (хто створив) ---
def create_user(username: str, password: str, role="user", rfid_uid: Optional[str]=None,
performed_by: Optional[Dict]=None):
    secret = pyotp.random_base32()
    backup = "BACKUP-" + pyotp.random_base32()[:10]
    conn = get_db_connection("main")
    cur = conn.cursor()
    cur.execute("INSERT INTO users(username, password_hash, role, rfid_uid, totp_secret,
totp_backup, created_at) VALUES(?,?,?,?,?,?,?)",
```

```
            (username, pwd_context.hash(password), role, rfid_uid, secret, backup,
datetime.utcnow().isoformat()))
    conn.commit()
    uid = cur.lastrowid
    conn.close()

    # Логуємо створення користувача адміном
    log_event("Users", f"User created: {username} (role={role})", user=performed_by)
    return get_user_by_id(uid)

# --- ЗМІНЕНО: Додано аргумент performed_by ---
def update_user(user_id:int, performed_by: Optional[Dict]=None, **kwargs):
    allowed = {"username","password","role","rfid_uid","totp_secret","totp_backup"}
    set_parts=[]
    params=[]
    for k,v in kwargs.items():
        if k not in allowed:
            continue
        if k == "password":
            set_parts.append("password_hash = ?")
            params.append(pwd_context.hash(v))
        else:
            set_parts.append(f"{k} = ?")
            params.append(v)
    if not set_parts:
        return get_user_by_id(user_id)
    params.append(user_id)
    conn = get_db_connection("main")
    cur = conn.cursor()
    cur.execute(f"UPDATE users SET {', '.join(set_parts)} WHERE id = ?", params)
    conn.commit()
    conn.close()

    log_event("Users", f"User updated id={user_id} fields={list(kwargs.keys())}",
user=performed_by)
    return get_user_by_id(user_id)

# --- ЗМІНЕНО: Додано аргумент performed_by ---
def delete_user(user_id:int, performed_by: Optional[Dict]=None):
    conn = get_db_connection("main")
    cur = conn.cursor()
    cur.execute("DELETE FROM users WHERE id = ?", (user_id,))
    conn.commit()
    conn.close()
    log_event("Users", f"User deleted id={user_id}", user=performed_by)

# ---------- Auth + session helpers ----------
def require_login(request: Request):
    user = None
    if "user_id" in request.session:
        user = get_user_by_id(request.session["user_id"])
    return user

def require_user_logged_in(request: Request):
    user = require_login(request)
    if not user:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Not
authenticated")
    # Check session timeout (300 seconds)
    if "last_auth" in request.session:
```

```
        last_auth = datetime.fromisoformat(request.session["last_auth"])
        if datetime.utcnow() - last_auth > timedelta(seconds=300):
            request.session.clear()
            raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Session expired")
    return dict(user) # Return as dict for easier usage

def require_admin(request: Request):
    user = require_user_logged_in(request)
    if user["role"] != "admin":
        raise HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail="Admin
required")
    return user

def generate_totp_qr_uri(username: str, secret: str, issuer="MicroclimateSystem"):
    uri = pyotp.totp.TOTP(secret).provisioning_uri(name=username, issuer_name=issuer)
    return uri

def make_qr_base64_from_uri(uri: str) -> str:
    img = qrcode.make(uri)
    buf = io.BytesIO()
    img.save(buf, format="PNG")
    b64 = base64.b64encode(buf.getvalue()).decode("ascii")
    return f"{b64}"

# ---------- Routes ----------
@app.on_event("startup")
def startup():
    init_db("logs")
    init_db("main")
    create_admin_if_missing()
    try:
        rfid_monitor.start()
    except Exception as e:
        logger.warning(f"Could not start RFID monitor: {e}")
    log_event("System", "Application startup")

@app.on_event("shutdown")
def shutdown():
    try:
        rfid_monitor.stop()
    except Exception:
        pass
    log_event("System", "Application shutdown")

@app.get("/", response_class=HTMLResponse)
def index(request: Request):
    user = require_login(request)
    return templates.TemplateResponse("index.html", {"request": request, "user": user,
"back_url": None})

# --- AUTH: login/logout / totp verify ---
@app.get("/login", response_class=HTMLResponse)
def login_get(request: Request):
    return templates.TemplateResponse("login.html", {"request": request, "msg": None})

@app.post("/login")
def login_post(request: Request, username: str = Form(...), password: str = Form(...)):
    user = get_user_by_username(username)
    if not user:
```

```
        return templates.TemplateResponse("login.html", {"request": request, "msg": "Invalid
credentials"})
    if not verify_password(password, user["password_hash"]):
        return templates.TemplateResponse("login.html", {"request": request, "msg": "Invalid
credentials"})
    request.session["pre_user_id"] = user["id"]
    return RedirectResponse(url="/totp", status_code=302)

@app.get("/totp", response_class=HTMLResponse)
def totp_get(request: Request):
    if "pre_user_id" not in request.session:
        return RedirectResponse("/login")
    user = get_user_by_id(request.session["pre_user_id"])
    return templates.TemplateResponse("totp.html", {"request": request, "user": user, "msg":
None})

@app.post("/totp")
def totp_post(request: Request, code: str = Form(...)):
    if "pre_user_id" not in request.session:
        return RedirectResponse("/login")
    user = get_user_by_id(request.session["pre_user_id"])
    secret = user["totp_secret"]
    totp = pyotp.TOTP(secret)
    if totp.verify(code):
        request.session.pop("pre_user_id", None)
        request.session["user_id"] = user["id"]
        request.session["last_auth"] = datetime.utcnow().isoformat()

        # --- ЗМІНЕНО: Логуємо вхід з указанням користувача ---
        user_dict = dict(user)
        log_event("Login", "Logged in successfully", user=user_dict)

        return RedirectResponse("/dashboard", status_code=302)
    else:
        return templates.TemplateResponse("totp.html", {"request": request, "user": user,
"msg": "Invalid code"})

@app.get("/logout")
def logout(request: Request):
    # Логируем выход перед очисткой сессии
    user = require_login(request)
    if user:
        log_event("Login", "Logged out", user=dict(user))
    request.session.clear()
    return RedirectResponse("/login")

# --- Dashboard & actions ---
@app.get("/dashboard", response_class=HTMLResponse)
def dashboard(request: Request):
    user = require_user_logged_in(request)
    reading = dht.read()
    print(reading)
    return templates.TemplateResponse("dashboard.html", {"request": request, "user": user,
"reading": reading, "relay_state": relay.state, "back_url": "/"})

@app.post("/relay/toggle")
def relay_toggle(request: Request):
    user = require_user_logged_in(request)
    # --- ЗМІНЕНО: Передаємо користувача в toggle ---
    relay.toggle(user=user)
```

```
    return {"status": "ok", "relay_state": relay.state}

@app.get("/sensor/json")
def sensor_json(request: Request):
    user = require_user_logged_in(request)
    r = dht.read()
    if not r:
        raise HTTPException(status_code=503, detail="Sensor read failed")
    return r

@app.websocket("/ws/sensor")
async def websocket_sensor(websocket: WebSocket):
    await websocket.accept()
    try:
        while True:
            reading = dht.read()
            data_to_send = reading.copy()
            data_to_send["relay_state"] = relay.state
            await websocket.send_text(json.dumps(data_to_send))
            await asyncio.sleep(5)
    except WebSocketDisconnect:
        logger.info("WebSocket client disconnected")
    except Exception as e:
        logger.error(f"WebSocket error: {e}")
        try:
            await websocket.close()
        except:
            pass

# IR control (user)
@app.post("/ir/send")
def ir_send(request: Request, code: str = Form(...)):
    user = require_user_logged_in(request)
    # code format example: "NEC:0x20DF10EF" or "NEC:addr,cmd"
    if code.upper().startswith("NEC:"):
        body = code[4:]
        if "," in body:
            addr, cmd = body.split(",", 1)
            try:
                addr = int(addr.strip(), 0)
                cmd = int(cmd.strip(), 0)
            except ValueError:
                return {"status": "error", "message": "Invalid address or command format"}
        else:
            try:
                val = int(body, 16)
                addr = (val >> 16) & 0xFFFF
                cmd = val & 0xFFFF
            except Exception:
                return {"status": "error", "message": "Invalid hex format"}

        # --- ЗМІНЕНО: Передаємо користувача в метод відправки ---
        ir.send_nec(addr, cmd, user=user)
        return {"status": "ok", "message": f"IR sent addr={addr} cmd={cmd}"}
    else:
        raise HTTPException(status_code=400, detail="Unknown IR format")

# AC (кондиционер) control
@app.post("/ac/cool")
def ac_cool(request: Request, temp: int = Form(...)):
```

```
    user = require_user_logged_in(request)
    if temp < 16 or temp > 30:
        return {"status": "error", "message": "Temperature must be between 16 and 30°C"}

    if ir.ac_cool(temp, user=user):
        return {"status": "ok", "message": f"AC cool command sent: {temp}°C"}
    else:
        return {"status": "error", "message": "Failed to send AC command"}

@app.post("/ac/off")
def ac_off(request: Request):
    user = require_user_logged_in(request)
    ir.ac_off(user=user)
    return {"status": "ok", "message": "AC off command sent"}


# --- Admin: user management ---
@app.get("/admin/users", response_class=HTMLResponse)
def admin_users(request: Request):
    admin = require_admin(request)
    conn = get_db_connection("main")
    cur = conn.cursor()
    cur.execute("SELECT id, username, role, rfid_uid, created_at FROM users")
    users = cur.fetchall()
    conn.close()
    return templates.TemplateResponse("admin_users.html", {"request": request, "user":
admin, "users": users, "back_url": "/dashboard"})

@app.get("/admin/user/create", response_class=HTMLResponse)
def admin_user_create_get(request: Request):
    admin = require_admin(request)
    return templates.TemplateResponse("admin_user_create.html", {"request": request,
"user": admin, "msg": None, "back_url": "/admin/users"})

@app.post("/admin/user/create")
def admin_user_create_post(request: Request, username: str = Form(...), password: str =
Form(...), role: str = Form("user"), rfid_uid: str = Form(None)):
    admin = require_admin(request)
    try:
        # --- ЗМІНЕНО: Передаємо admin як performed_by ---
        user = create_user(username, password, role=role, rfid_uid=rfid_uid,
performed_by=admin)
    except Exception as e:
        return templates.TemplateResponse("admin_user_create.html", {"request": request,
"user": admin, "msg": f"Error: {e}"})
    return RedirectResponse(f"/admin/user/{user['id']}/show_qr", status_code=302)

@app.get("/admin/user/{user_id}/show_qr", response_class=HTMLResponse)
def admin_user_show_qr(request: Request, user_id: int):
    admin = require_admin(request)
    user = get_user_by_id(user_id)
    if not user:
        raise HTTPException(status_code=404)
    uri = generate_totp_qr_uri(user["username"], user["totp_secret"])
    qr = make_qr_base64_from_uri(uri)
    return templates.TemplateResponse("show_qr.html", {"request": request, "user": admin,
"edited_user": user, "qr": qr, "back_url": "/admin/users"})

@app.get("/admin/user/{user_id}/edit", response_class=HTMLResponse)
def admin_user_edit_get(request: Request, user_id: int):
```

```
    admin = require_admin(request)
    user = get_user_by_id(user_id)
    return templates.TemplateResponse("admin_user_edit.html", {"request": request, "user":
admin, "edited_user": user, "msg": None, "back_url": "/admin/users"})

@app.post("/admin/user/{user_id}/edit")
def admin_user_edit_post(request: Request, user_id: int, username: str = Form(...),
password: str = Form(None), role: str = Form("user"), rfid_uid: str = Form(None)):
    admin = require_admin(request)
    kwargs = {"username": username, "role": role, "rfid_uid": rfid_uid}
    if password:
        kwargs["password"] = password
    try:
        # --- ЗМІНЕНО: Передаємо admin як performed_by ---
        update_user(user_id, performed_by=admin, **kwargs)
    except Exception as e:
        return templates.TemplateResponse("admin_user_edit.html", {"request": request,
"user": admin, "edited_user": get_user_by_id(user_id), "msg": f"Error: {e}"})
    return RedirectResponse("/admin/users", status_code=302)

@app.post("/admin/user/{user_id}/reset_totp")
def admin_user_reset_totp(request: Request, user_id: int):
    admin = require_admin(request)
    new_secret = pyotp.random_base32()
    new_backup = "BACKUP-" + pyotp.random_base32()[:10]
    # --- ЗМІНЕНО: Передаємо admin ---
    update_user(user_id, totp_secret=new_secret, totp_backup=new_backup,
performed_by=admin)
    return RedirectResponse(f"/admin/user/{user_id}/show_qr", status_code=302)

@app.post("/admin/user/{user_id}/delete")
def admin_user_delete(request: Request, user_id: int):
    admin = require_admin(request)
    # --- ЗМІНЕНО: Передаємо admin ---
    delete_user(user_id, performed_by=admin)
    return RedirectResponse("/admin/users", status_code=302)

@app.get("/admin/logs", response_class=HTMLResponse)
def admin_logs(request: Request):
    admin = require_admin(request)
    conn = get_db_connection("logs")
    cur = conn.cursor()
    cur.execute("SELECT * FROM logs ORDER BY ts DESC LIMIT 500")
    logs = cur.fetchall()
    conn.close()
    return templates.TemplateResponse("admin_logs.html", {"request": request, "user":
admin, "logs": logs, "back_url": "/dashboard"})

@app.post("/admin/user/{user_id}/bind_rfid")
def admin_bind_rfid(request: Request, user_id: int, rfid_uid: str = Form(...)):
    admin = require_admin(request)
    # --- ЗМІНЕНО: Передаємо admin ---
    update_user(user_id, rfid_uid=rfid_uid, performed_by=admin)
    return RedirectResponse("/admin/users", status_code=302)

@app.get("/me/totp", response_class=HTMLResponse)
def me_totp(request: Request):
    user = require_user_logged_in(request)
    uri = generate_totp_qr_uri(user["username"], user["totp_secret"])
    qr = make_qr_base64_from_uri(uri)
```

```
    return templates.TemplateResponse("show_qr.html", {"request": request, "user": user,
"edited_user": user, "qr": qr, "back_url": "/dashboard"})

@app.get("/api/users")
def api_users(request: Request):
    admin = require_admin(request)
    conn = get_db_connection("main")
    cur = conn.cursor()
    cur.execute("SELECT id, username, role, rfid_uid FROM users")
    rows = cur.fetchall()
    conn.close()
    return {"users": [dict(r) for r in rows]}

@app.get("/health")
def health():
    return {"status": "ok", "time": datetime.utcnow().isoformat()}

@app.exception_handler(HTTPException)
def http_exception_handler(request: Request, exc: HTTPException):
    if exc.status_code == status.HTTP_401_UNAUTHORIZED:
        return RedirectResponse("/login")
    elif exc.status_code == status.HTTP_403_FORBIDDEN:
        return templates.TemplateResponse("403.html", {"request": request, "status_code":
exc.status_code, "detail": "Forbidden"}, status_code=exc.status_code)

@app.exception_handler(status.HTTP_404_NOT_FOUND)
def not_found_handler(request: Request, exc: HTTPException):
    return templates.TemplateResponse("404.html", {"request": request, "status_code":
exc.status_code, "detail": "Not Found"}, status_code=exc.status_code)
```

Файл *light.json*

```
{
  "format": {
    "preamble": [
      8,
      8
    ],
    "coding": "ppm",
    "zero": [
      1,
      1
    ],
    "one": [
      1,
      3
    ],
    "byte_separator": [
      8,
      8
    ],
    "postamble": [
      1
    ],
    "pre_data": "4D B2",
```

```
    "byte_by_byte_complement": true,
    "timebase": 540,
    "carrier": 38000
  },
  "keys": {
    "OFF": "DE 07 4D DE 07",
    "20 cool": "F8 14 4D F8 14",
    "22 cool": "F8 1E 4D F8 1E",
    "24 cool": "F8 12 4D F8 12",
    "26 cool": "F8 1B 4D F8 1B",
    "28 cool": "F8 11 4D F8 11",
    "30 cool": "F8 1D 4D F8 1D"
  }
}
```

Файл *network_manager_script.service*

```
[Unit]
Description=Raspberry Pi Network Manager Script
After=network-online.target wpa_supplicant.service
Wants=network-online.target

[Service]
Type=simple

ExecStart=/usr/bin/python3 /usr/local/bin/network_manager_script.py

WorkingDirectory=/usr/local/bin/

User=root
Group=root

Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Файл *network_manager_script.service*

```
import subprocess
import time
import sys
import os
import signal

# --- Конфігурація МЕРЕЖІ ---
TARGET_SSID = "TP-Link_C2D4"
TARGET_PASSWORD = "45404580"
HOTSPOT_SSID = "PiHotspot"
HOTSPOT_PASSWORD = "HotspotPassword123"
HOTSPOT_IP = "192.168.1.181/24"
```

```
ETH_INTERFACE = "eth0"
WIFI_INTERFACE = "wlan0"
PING_HOST = "8.8.8.8"

# --- Конфігурація ДОДАТКА ---
# Розширення шляху до домашньої папки (~/project/env)
#VENV_PATH = os.path.expanduser("/home/user/project/env")
PROJECT_PATH = "/home/user/project"
VENV_PATH = f"{PROJECT_PATH}/env"
UVICORN_EXE = f"{VENV_PATH}/bin/uvicorn" # Повний шлях до виконуваного файлу uvicorn
UVICORN_CMD_ARGS = f"app:app --host 0.0.0.0 --port 443"
SCAN_INTERVAL = 30 # Інтервал сканування цільової мережі, коли Hotspot активний (сек)

# --- Глобальні змінні для процесів ---
uvicorn_process = None
# --------------------

def run_command(command, check=False):
    """Виконує системну команду і повертає stdout."""
    try:
        result = subprocess.run(
            command,
            capture_output=True,
            text=True,
            check=check,
            shell=True
        )
        return result.stdout.strip(), result.stderr.strip(), result.returncode
    except subprocess.CalledProcessError as e:
        print(f"Помилка виконання команди: {e.cmd}\n{e.stderr}", file=sys.stderr)
        return None, e.stderr, e.returncode
    except FileNotFoundError:
        print(f"Помилка: Команда '{command.split()[0]}' не знайдена.", file=sys.stderr)
        return None, "Command not found", 127

# --- Керування додатком Uvicorn ---

def start_uvicorn_app():
    """Запускає Uvicorn у фоновому процесі, використовуючи віртуальне середовище."""
    global uvicorn_process

    if uvicorn_process and uvicorn_process.poll() is None:
        return

    # 1. КРИТИЧНА ПЕРЕВІРКА ШЛЯХУ
    if 'ВАШ_КОРИСТУВАЧ' in PROJECT_PATH:
        print("Критична помилка конфігурації: Будь ласка, замініть 'ВАШ_КОРИСТУВАЧ' у змінній PROJECT_PATH на фактичне ім'я користувача.", file=sys.stderr)
        return

    if not os.path.exists(UVICORN_EXE):
        print(f"Критична помилка: Виконуваний файл Uvicorn не знайдено за шляхом: {UVICORN_EXE}", file=sys.stderr)
        return
    if not os.path.exists(f"{PROJECT_PATH}/app.py"):
        print(f"Критична помилка: Файл app.py не знайдено у каталозі: {PROJECT_PATH}", file=sys.stderr)
        return
```

```
    print(f"Запуск Uvicorn ({UVICORN_CMD_ARGS}) з каталогу {PROJECT_PATH}...")
    try:
        log_file = open("/tmp/uvicorn_app.log", "a")

        uvicorn_process = subprocess.Popen(
            f"{UVICORN_EXE} {UVICORN_CMD_ARGS}",
            shell=True,
            stdout=log_file,
            stderr=log_file,
            preexec_fn=os.setsid,
            # Встановлюємо правильний робочий каталог
            cwd=PROJECT_PATH
        )
        print(f"Uvicorn запущено. PID: {uvicorn_process.pid}. Логи: /tmp/uvicorn_app.log")
        time.sleep(2)
    except Exception as e:
        print(f"Критична помилка при запуску Uvicorn: {e}", file=sys.stderr)
        uvicorn_process = None

def stop_uvicorn_app():
    """Надсилає сигнал завершення процесу Uvicorn."""
    global uvicorn_process
    if uvicorn_process and uvicorn_process.poll() is None:
        print("Зупинка Uvicorn...")
        try:
            # Надсилаємо SIGTERM групі процесів
            os.killpg(os.getpgid(uvicorn_process.pid), signal.SIGTERM)
            uvicorn_process.wait(timeout=10)
            print("Uvicorn зупинено.")
        except Exception as e:
            print(f"Помилка при зупинці Uvicorn (примусове завершення): {e}", file=sys.stderr)
            try:
                # Примусове завершення, якщо SIGTERM не спрацював
                os.killpg(os.getpgid(uvicorn_process.pid), signal.SIGKILL)
            except:
                pass
        uvicorn_process = None

# --- Мережеві функції (залишені з попередньої версії) ---

def check_internet_connection(interface=None):
    """Перевіряє наявність Інтернет-з'єднання через вказаний інтерфейс (за
бажанням)."""
    interface_option = f"-I {interface}" if interface else ""
    cmd = f"ping -c 3 -W 1 {interface_option} {PING_HOST}"
    _, _, code = run_command(cmd)
    return code == 0

def connect_wifi(ssid, password):
    """Підключається до Wi-Fi мережі за допомогою nmcli."""
    print(f"Спроба підключення до Wi-Fi: {ssid}...")
    run_command(f'nmcli con delete id \"{ssid}\"')
    cmd_add = f'nmcli dev wifi connect \"{ssid}\" password \"{password}\" ifname
{WIFI_INTERFACE} name \"{ssid}\"'
    _, stderr, code = run_command(cmd_add)

    if code == 0:
        print("Wi-Fi підключення успішно налаштовано.")
        time.sleep(10)
```

```
        return check_internet_connection(WIFI_INTERFACE)
    else:
        print(f"Помилка підключення до Wi-Fi. Деталі: {stderr.splitlines()[-1]}")
        return False

def start_hotspot():
    """Створює і запускає точку доступу (Hotspot) на wlan0."""
    print(f"Запуск Hotspot ({HOTSPOT_SSID})...")
    run_command(f'nmcli con delete id \"{HOTSPOT_SSID}\"')
    cmd_add = (
        f'nmcli connection add type wifi ifname {WIFI_INTERFACE} con-name
\"{HOTSPOT_SSID}\" autoconnect no '
        f'ssid \"{HOTSPOT_SSID}\" mode ap"
    )
    run_command(cmd_add, check=True)
    cmd_sec = (
        f'nmcli connection modify \"{HOTSPOT_SSID}\" wifi-sec.key-mgmt wpa-psk '
        f'wifi-sec.psk \"{HOTSPOT_PASSWORD}\""
    )
    run_command(cmd_sec, check=True)
    cmd_ip = f'nmcli connection modify \"{HOTSPOT_SSID}\" ipv4.method shared
ipv4.addresses {HOTSPOT_IP}"
    run_command(cmd_ip, check=True)

    cmd_up = f'nmcli connection up \"{HOTSPOT_SSID}\""
    _, stderr, code = run_command(cmd_up)

    if code == 0:
        print(f'Hotspot '{HOTSPOT_SSID}' успішно запущено. IP: {HOTSPOT_IP.split('/')[0]}")
        return True
    else:
        print(f"Помилка запуску Hotspot: {stderr}")
        return False

def stop_hotspot():
    """Зупиняє Hotspot, видаляючи підключення."""
    print(f"Зупинка Hotspot '{HOTSPOT_SSID}'...")
    run_command(f'nmcli connection down \"{HOTSPOT_SSID}\"')
    run_command(f'nmcli connection delete \"{HOTSPOT_SSID}\"')
    run_command("nmcli radio wifi on")
    print("Hotspot вимкнено.")

def scan_for_target_wifi(target_ssid):
    """Сканує доступні мережі і перевіряє, чи присутня цільова мережа."""
    print(f"Сканування мереж для пошуку '{target_ssid}'...")

    stdout, _, code = run_command(f'nmcli --fields SSID dev wifi list | grep -w
\"{target_ssid}\"')

    if code == 0 and target_ssid in stdout.splitlines():
        print(f" Цільова мережа '{target_ssid}' знайдена.")
        return True

    print(f"Цільова мережа '{target_ssid}' не знайдена.")
    return False

# --- Головний цикл ---

def main_loop():
    """Головний цикл перевірки та управління мережею."""
```

```
    run_command("sudo pigpiod")
    hotspot_active = False
    last_scan_time = 0

    print("Запуск мережевого менеджера.")

    # Спробувати підключитися до цільової мережі при запуску
    if connect_wifi(TARGET_SSID, TARGET_PASSWORD):
        print(f"Стартове підключення до {TARGET_SSID} успішне.")

    try:
        while True:
            # Перевіряємо, чи є Інтернет
            is_internet_ok = check_internet_connection(ETH_INTERFACE) or
check_internet_connection(WIFI_INTERFACE)

            # --- СЦЕНАРІЙ 1: Інтернет працює ---
            if is_internet_ok:
                if hotspot_active:
                    print(" Інтернет відновлено. Вимикаємо Hotspot.")
                    stop_hotspot()
                    hotspot_active = False
                else:
                    print(" Інтернет-з'єднання активне.")

                # Якщо ми не підключені до цільового Wi-Fi, намагаємося підключитися
                is_target_active = run_command(f"nmcli con show --active | grep -w
'{TARGET_SSID}'")[2] == 0
                if not is_target_active:
                    connect_wifi(TARGET_SSID, TARGET_PASSWORD)

            # --- СЦЕНАРІЙ 2: Інтернет відсутній ---
            else:
                print(" Інтернет-з'єднання відсутнє.")

                if hotspot_active:
                    current_time = time.time()
                    if current_time - last_scan_time >= SCAN_INTERVAL:
                        last_scan_time = current_time

                        if scan_for_target_wifi(TARGET_SSID):
                            # Цільова мережа знайдена! Перемикаємося.
                            print(" Цільова мережа з'явилася! Вимкнення Hotspot та підключення.")
                            stop_hotspot()
                            hotspot_active = False
                            # Логіка повернеться до "Hotspot неактивний" і спробує підключитися

                        else:
                            print(f"Hotspot активний. Цільова мережа не знайдена. Сканування
через {SCAN_INTERVAL} сек.")

                # Якщо Hotspot неактивний
                if not hotspot_active:
                    if connect_wifi(TARGET_SSID, TARGET_PASSWORD):
                        print(" Успішно підключено до цільового Wi-Fi.")
                        continue

                    # Якщо підключення не вдалося - запускаємо Hotspot
                    if start_hotspot():
```

```
                hotspot_active = True

            # --- ОСНОВНА ВИМОГА: Запуск додатка Uvicorn у будь-якому сценарії ---
            start_uvicorn_app()

            time.sleep(5) # Пауза перед наступною перевіркою

    except KeyboardInterrupt:
        print("\nОтримано сигнал зупинки (Ctrl+C). Завершення роботи...")
    finally:
        # Очищення при виході
        stop_uvicorn_app()
        if hotspot_active:
            stop_hotspot()
        print("Скрипт завершено.")

if __name__ == "__main__":
    if sys.platform.startswith('linux'):
        print("!!! Для коректної роботи необхідні права root (sudo) та встановлений
NetworkManager (nmcli).")
        main_loop()
    else:
        print("Цей скрипт призначений для виконання на Linux-системах.")
```