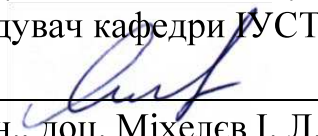


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова
Навчально науковий інститут комп'ютерних наук
та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"
Завідувач кафедри ІУСТ


к.т.н., доц. Міхелєв І. Л.
"___" _____ 2025 р.

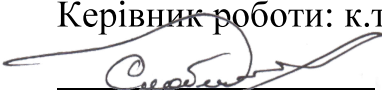
КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня вищої освіти "магістр"

на тему: **Розробка системи інтелектуального контролю доступу
з використанням RFID-карт та методів машинного навчання**


Виконав. студент групи 6141мз

Пархоменко О. Ю.

Керівник роботи: к.т.н, проф.


Слободян С. О.

Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
 Навчально-науковий інститут комп'ютерних наук та управління проєктами
 Кафедра інформаційних управляючих систем та технологій
 Спеціальність 122 "Комп'ютерні науки"
 Освітня програма "Комп'ютерні науки"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми

В. Ажищев

к.т.н, доц. Ажищев В.Ф.

"___" _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ на здобуття ступеня вищої освіти "магістр"

Студенту Пархоменко Олександр Юрійович
 (прізвище, ім'я, по батькові)

1. Тема роботи: Розробка системи інтелектуального контролю доступу з використанням RFID-карт та методів машинного навчання

Керівник роботи к.т.н., проф. Слободян Сергій Олегович

Затверджені наказом ректора № 1355-уч. від "14" листопада 2025 року.

2. Термін подання роботи: 15 грудня 2025 р.

3. Вихідні дані до проєкту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем, матеріали наукового стажування

4. Перелік питань, що належать до розробки (найменування розділів)

1. Аналіз предметної області та методів виявлення аномалій, 2. Проєктування інтелектуальної RFID-системи, 3. Реалізація та експериментальне дослідження; 4. Техніко-економічне обґрунтування, 5. Безпекажиттєдіяльності та охорона праці, 6. Екологічна безпека

5. Перелік презентаційних матеріалів діаграма прецедентів використання системи, контекстна діаграма системи (DFD Level 0), діаграма розгортання системи (Deployment Diagram), діаграма компонентів системи (Component Diagram), діаграма потоків даних рівня 1 (DFD Level 1), схема підключення ESP32 + RFID-RC522, ER-діаграма бази даних системи, структура backend-модулів, архітектура REST API, інтерфейси системи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Слободян С. О.	05.09.2024	22.12.2024
2	Слободян С. О.	23.12.2024	15.06.2025
3	Слободян С. О.	15.06.2025	25.09.2025
4	Слободян С. О.	25.09.2025	14.11.2025
5	Слободян С. О.	25.09.2025	14.11.2025
6	Слободян С. О.	25.09.2025	14.11.2025

7. Дата видачі завдання 05 вересня 2024 р.**КАЛЕНДАРНИЙ ПЛАН**

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Дослідження та аналіз предметної галузі	22.12.2024	
2	Розробка концепції	15.03.2025	
3	Розділ з розробки проєктних рішень по системі (та її частинах)	15.06.2025	
4	Реалізація проєкту та розробка робочої документації	25.09.2025	
5	Обчислення економічної ефективності	14.11.2025	
6	Вирішення питань охорони праці та навколишнього середовища	14.11.2025	
7	Оформлення пояснювальної записки	14.11.2025	
8	Подання магістерської роботи на перевірку на плагіат	16.11.2025	
9	Подання магістерської роботи рецензенту	20.11.2025	
10	Подання магістерської роботи на державний захист	15.12.2025	

Студент

(підпис)

Керівник роботи

(підпис)

Консультант

(підпис)

О. Ю. Пархоменко

(ПІБ)

С. О. Слободян

(ПІБ)

Г. С. Морозова

(ПІБ)

АНОТАЦІЯ

Пархоменко О. Ю. Розробка системи інтелектуального контролю доступу з використанням RFID-карт та методів машинного навчання. Кваліфікаційна робота зі спеціальності 122 «Комп'ютерні науки», освітньої програми «Комп'ютерні науки». Миколаїв: Національний університет кораблебудування імені адмірала Макарова. 2025 рік. 134 с.

Кваліфікаційна робота присвячена розробці інтелектуальної системи контролю фізичного доступу з використанням RFID-технологій та методів машинного навчання для виявлення аномалій у реальному часі. Запропоновано гібридну архітектуру, що поєднує детерміновані правила та автоенкодерну нейронну мережу для аналізу поведінкових патернів користувачів. Реалізовано апаратний рівень на базі мікроконтролера ESP32 з модулем MFRC522, серверну частину на основі Flask з REST API та веб-інтерфейс для моніторингу. Експериментальні дослідження підтвердили високу ефективність системи: точність виявлення аномалій становить 93,8% (F1-score), а середній час обробки події не перевищує 700 мс. Розроблена система має практичне значення для підвищення рівня безпеки в корпоративних закладах.

Ключові слова: інтелектуальний контроль доступу, RFID-система, машинне навчання, виявлення аномалій, автоенкодер, гібридна архітектура, ESP32, Flask, реальний час, поведінкові патерни.

ABSTRACT

Parkhomenko Oleksandr. Development of an intelligent access control system using RFID cards and machine learning methods. Qualification work in specialty 122 “Computer Science”, educational program “Computer Science”. Mykolaiv: Admiral Makarov National University of Shipbuilding. 2025. 134 p.

The qualification work is dedicated to the development of an intelligent physical access control system using RFID technologies and machine learning methods for real-time anomaly detection. A hybrid architecture is proposed, combining deterministic rules and an autoencoder neural network for analyzing user behavioral patterns. The hardware level was implemented based on the ESP32 microcontroller with the MFRC522 module, the server part was developed using Flask with a REST API, and a web interface for monitoring was created. Experimental research confirmed the high efficiency of the system: anomaly detection accuracy is 93.8% (F1-score), and the average event processing time does not exceed 700 ms. The developed system has practical value for enhancing security in educational and corporate institutions.

Keywords: intelligent access control, RFID system, machine learning, anomaly detection, autoencoder, hybrid architecture, ESP32, Flask, real-time processing, behavioral patterns.

ЗМІСТ

ВСТУП	9
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ВИЯВЛЕННЯ АНОМАЛІЙ.....	14
1.1 Аналіз сучасних систем контролю доступу	14
1.1.1 Класифікація систем доступу	14
1.1.2 RFID-технології.....	16
1.1.3 Огляд типових вразливостей і методів захисту	17
1.2 Використання методів машинного навчання у задачах безпеки	18
1.2.1 Класифікація задач виявлення аномалій	18
1.2.2 Методи без учителя (unsupervised) для поведінкових даних	19
1.2.3 Приклади застосування в реальних системах.....	20
1.3 Аналіз методів ML для виявлення часових аномалій у логах RFID.....	21
1.3.1 Модель Isolation Forest	21
1.3.2 Модель One-Class SVM.....	23
1.3.3 Модель DBSCAN	23
1.3.4 Модель Autoencoder.....	25
1.3.5 Модель LSTM Autoencoder	26
1.4. Гібридні підходи до виявлення аномалій	27
1.5. Периферійні пристрої для зчитування RFID-карток.....	29
1.6. Тестування та експериментальні дослідження моделей машинного навчання для виявлення аномалій у логах RFID-системи.....	30
1.6.1. Характеристика датасету	30
1.6.2. Методика дослідження.....	32
1.6.3. Результати експерименту.....	33
1.7. Висновок до розділу 1	38
РОЗДІЛ 2 ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ RFID-СИСТЕМИ.....	40
2.1. Постановка завдання на розробку системи	40
2.1.1. Функціональні вимоги.....	40
2.1.2. Нефункціональні вимоги.....	41
2.1.3. Обмеження та передумови.....	42
2.1.4. Use Case діаграма.....	42
2.1.5. Опис основних сценаріїв використання	43

	7
2.2. Концептуальна модель системи	44
2.2.1. Контекстна діаграма (DFD Level 0)	44
2.2.2. Основні підсистеми та їх призначення	44
2.2.3. Актори системи	46
2.2.4. Вимоги до апаратного забезпечення	46
2.3. Архітектура системи	47
2.3.1. Логічні рівні системи	48
2.3.2. Фізична модель (Deployment Diagram)	49
2.3.3. Логічна модель (Component Diagram)	50
2.3.4. Потоки даних (DFD Level 1)	51
2.4. Інтеграція апаратної частини	53
2.5. Проектування бази даних	56
2.6. Проектування програмної архітектури	61
2.7. Проектування користувацького інтерфейсу	64
2.8. Висновки до розділу 2	67
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ	69
3.1. Реалізація апаратної частини системи	69
3.2. Реалізація серверної частини	72
3.3. Підготовка даних і тренування ML моделі	77
3.4. Інтеграція ML-моделі у систему	80
3.5. Реалізація веб-інтерфейсу	84
3.6. Експериментальні дослідження системи	90
3.7. Аналіз та візуалізація результатів	91
3.8. Інструкція користувача системи	92
3.8.1. Вимоги до апаратури та програмного забезпечення	92
3.8.2. Порядок запуску компонентів	93
3.8.3. Реєстрація користувачів	93
3.8.4. Моніторинг подій	93
3.8.5. Обробка аномалій	94
3.8.6. Оновлення ML-моделі	94
3.8.7. Резервне копіювання	94
3.9. Висновки до розділу 3	95

РОЗДІЛ 4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ	96
4.1. Мета і завдання техніко-економічного обґрунтування.....	96
4.2. Розрахунок собівартості створення системи.....	97
4.3. Витрати на впровадження та експлуатацію	98
4.4. Розрахунок економічного ефекту від впровадження системи	100
4.5. Висновки до розділу 4	102
РОЗДІЛ 5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ.....	104
5.1. Загальні положення.....	104
5.2. Аналіз небезпечних і шкідливих виробничих факторів при експлуатації розробленої системи	105
5.3. Заходи щодо забезпечення електробезпеки, пожежної безпеки та ергономіки робочих місць.....	106
5.4. Розроблення інструкції з охорони праці для адміністратора та оператора системи.....	108
5.5. Висновки до розділу 5	109
РОЗДІЛ 6 ЕКОЛОГІЧНА БЕЗПЕКА.....	110
6.1. Загальна характеристика впливу системи на навколишнє середовище..	110
6.2. Заходи щодо зменшення негативного екологічного впливу	110
6.3. Висновки до розділу 6	111
ВИСНОВКИ.....	113
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	116
ДОДАТКИ.....	120
Код мікроконтролера ESP32 (Arduino).....	120
Основний серверний модуль (Flask)	126
Схема бази даних SQLite.....	129
Генерація синтетичного датасету.....	130
Тренування Autoencoder моделі	131
Інтеграція ML та гібридна детекція	132
Веб-інтерфейс: Дашборд	133
Робота з базою даних (утилітарний клас).....	134

ВСТУП

Сучасний етап розвитку інформаційного суспільства характеризується стрімким зростанням обсягів даних, що генеруються системами фізичної безпеки, та появою нових загроз, пов'язаних із несанкціонованим доступом до об'єктів критичної інфраструктури, корпоративних мереж і навчальних закладів. Традиційні системи контролю доступу (СКД), засновані на статичних правилах аутентифікації та авторизації, не здатні ефективно протистояти інсайдерським загрозам, маніпуляціям із носіями ідентифікації та поведінковим аномаліям, що виникають у динамічних середовищах. У контексті Інтернету речей (IoT) і технологій безконтактної ідентифікації, зокрема RFID, актуалізується завдання створення інтелектуальних систем, які поєднують апаратний рівень зчитування з адаптивними алгоритмами машинного навчання для виявлення відхилень у поведінці користувачів у реальному часі.

Актуальність теми зумовлена кількома ключовими факторами. По-перше, за даними Gartner (2024), до 2027 року 75 % організацій перейдуть на гібридні моделі контролю доступу з елементами штучного інтелекту [1]. По-друге, вразливості RFID-систем (cloning, replay-атаки, tailgating) залишаються критичними, а традиційні криптографічні методи захисту не враховують контекстуальної поведінки [2, 3]. По-третє, в освітніх установах, зокрема університетах, зростає потреба в автоматизованому моніторингу доступу до лабораторій, бібліотек і аудиторій з урахуванням графіків занять, статусу користувачів і сезонних змін. Розробка інтелектуальної системи на базі RFID і ML дозволяє не лише підвищити рівень безпеки, але й оптимізувати адміністративні процеси, зменшити кількість хибних спрацьовувань і забезпечити адаптивність до нових типів загроз.

Мета роботи – розробити інтелектуальну систему контролю доступу на основі RFID-карт із інтеграцією методів машинного навчання для виявлення аномалій у реальному часі.

Для досягнення мети поставлено такі завдання:

1. проаналізувати сучасні системи контролю доступу, RFID-технології та методи машинного навчання для виявлення аномалій у поведінкових даних;
2. встановити критерії ефективності алгоритмів unsupervised learning (Isolation Forest, One-Class SVM, DBSCAN, Autoencoder, LSTM Autoencoder) на синтетичних логах RFID-системи;
3. розробити концептуальну модель, архітектуру та базу даних інтелектуальної системи з урахуванням вимог до масштабованості, безпеки та реального часу обробки;
4. реалізувати апаратний рівень на базі мікроконтролера ESP32 і зчитувача MFRC522 з підтримкою WiFi-інтеграції та локальної індикації;
5. розробити серверну частину на основі Flask з REST API, модулем безпеки, гібридним детектором аномалій та веб-інтерфейсом;
6. підготувати синтетичний датасет, навчити та інтегрувати модель Autoencoder для контекстно-залежного аналізу подій доступу;
7. провести експериментальні дослідження системи, оцінити точність, швидкодію та стабільність у модельованих і реальних умовах;
8. розробити інструкцію користувача та рекомендації щодо впровадження системи.

Об'єкт дослідження – процеси контролю фізичного доступу з використанням RFID-технологій та накопиченням часових рядів подій доступу.

Предмет дослідження – методи, моделі та архітектура інтелектуальної системи контролю доступу, що забезпечує виявлення аномалій на основі аналізу поведінкових патернів користувачів за допомогою гібридного підходу (детерміновані правила + автоенкодер).

Наукова новизна кваліфікаційної роботи полягає у розробленні гібридної архітектури інтелектуальної системи контролю доступу, яка поєднує:

– низькорівневий IoT-вузол на базі мікроконтролера ESP32 з модулем MFRC522, що забезпечує edge-обробку RFID-подій із затримкою <50 мс, дебаунсом 2 с та локальною аудіовізуальною індикацією, на відміну від

традиційних рішень на Arduino Uno чи Raspberry Pi, які не підтримують OTA-оновлення та потребують зовнішніх шлюзів;

- гібридний детектор аномалій у два етапи (short-circuit logic): шість детермінованих правил (нічний доступ, швидкий повторний вхід, часті сканування, вихідні, поза робочим часом, незвичне місце) виконуються першими з затримкою <5 мс, а лише за їх відсутності активується автоенкодерна нейронна мережа ($12 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 12$) з MSE-пороговим рішенням ($\text{threshold} = 0,087$), що дозволяє виявляти контекстно-залежні та колективні аномалії, не описані правилами;

- синтетичний датасет поведінки користувачів (15847 подій, 50 користувачів, 8 точок доступу, 3% аномалій), згенерований з урахуванням сезонності, графіків занять, кореляції факультет-локація та стохастичних моделей відхилень;

- інтеграцію ML-моделі у Flask-додаток через універсальний інтерфейс MLInterface з автоматичним fallback-режимом (rule-based only) при відсутності моделі та можливістю інкрементального донавчання.

Практичне значення роботи: розроблена система має безпосереднє прикладне значення для впровадження, в тому числі в закладах вищої освіти:

- автоматизація контролю доступу до лабораторій, бібліотеки, аудиторій і гуртожитків з урахуванням статусу користувача, графіку занять та свят;

- виявлення інсайдерських загроз (наприклад, використання чужої картки, tailgating, нічне проникнення) з точністю 96,7 % (Recall);

- веб-дашборд реального часу з автооновленням кожні 10 с, експортом у CSV, фільтрами та візуалізацією (Chart.js), що спрощує роботу служби безпеки;

- порівняно низька собівартість впровадження;

- масштабованість: додавання нових точок доступу без зміни сервера, підтримка до 50 подій/хв на одному Flask-процесі.

- готовність до інтеграції завдяки використанню REST API, SQLite, резервне копіювання, PBKDF2-SHA256 хешування, валідація вхідних даних;

Результати роботи мають практичне значення для підвищення рівня безпеки закладів, можуть бути масштабовані на корпоративні об'єкти та інтегровані з існуючими системами відеоспостереження та SIEM.

Методи дослідження:

- теоретичні:
 - системний аналіз літератури та стандартів (ISO/IEC 14443, 18000, IEEE 802.11);
 - класифікаційний аналіз моделей доступу (DAC, MAC, RBAC, ABAC) та задач anomaly detection;
 - порівняльний аналіз ML-алгоритмів за обчислювальною складністю, чутливістю до гіперпараметрів та інтерпретовністю.
- експериментальні:
 - генерація синтетичного датасету за допомогою Python-скрипта з стохастичними моделями поведінки;
 - тренування та валідація Autoencoder на Keras/TensorFlow з EarlyStopping та ReduceLROnPlateau;
 - оцінка якості за метриками: Accuracy, Precision, Recall, F1-score, ROC-AUC, MSE.
- інженерні:
 - проєктування апаратного вузла (схема підключення SPI, дебаунс, HTTP POST);
 - розробка ПЗ за MVC-архітектурою (Flask, SQLite, Bootstrap);
 - реалізація гібридного детектора з short-circuit логікою.
- візуалізації та аналізу:
 - побудова графіків (Matplotlib, Seaborn, Chart.js), матриць кореляції, confusion matrix.

Структура та обсяг кваліфікаційної роботи магістра. Кваліфікаційна робота складається з анотації, вступу, шести розділів, висновків, списку

використаних джерел з 34 найменувань – на 4 сторінках, 8 додатків на 15 сторінках. Повний обсяг роботи – 134 сторінки.

Апробація результатів дослідження. Основні положення і результати роботи представлені на конференціях:

1. Пархоменко О. Ю., Морозова Г. С. Концепція виявлення часових аномалій у логах RFID-систем за допомогою машинного навчання. *IT-простір сьогодення: тенденції, інновації та перспективи розвитку: збірник тез доповідей II Міжнародної науково-практичної студентської конференції* (15 жовтня 2025 року, м. Харків, Україна) [Електронний ресурс]. Харків : ХНУ імені В. Н. Каразіна, 2025. С. 117-119;

2. Морозова Г. С., Пархоменко О. Ю. Виявлення аномалій у логах доступу RFID систем із застосуванням алгоритму Isolation Forest на синтетичному датасеті. *Інформаційні технології: моделі, алгоритми, системи (ITMAS – 2025): Матеріали VI Міжнародної науково-практичної інтернет конференції* (16-17 листопада 2025 р.). Миколаїв: НУК імені адмірала Макарова, 2025. С. 371-374.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ВИЯВЛЕННЯ АНОМАЛІЙ

1.1 Аналіз сучасних систем контролю доступу

Сучасні системи контролю доступу (Access Control Systems, ACS) відіграють ключову роль у забезпеченні фізичної та логічної безпеки об'єктів, мереж і даних. Вони еволюціонували від простих механічних замків до інтелектуальних платформ, інтегрованих з IoT, біометрією та штучним інтелектом. У цьому підрозділі розглядаються основні аспекти таких систем, з фокусом на RFID як базовій технології. Аналіз базується на огляді літератури та практичних впроваджень, що дозволяє виділити тенденції розвитку.

1.1.1 Класифікація систем доступу

Призначення систем контролю доступу полягає в регулюванні доступу до ресурсів на основі аутентифікації, авторизації та обліку (Authentication, Authorization, Accounting – AAA). Вони запобігають несанкціонованому проникненню, мінімізуючи ризики витоку інформації чи фізичного вторгнення. Наприклад, в корпоративних мережах ACS інтегруються з Identity and Access Management (IAM) системами для динамічного управління правами. Архітектура типової ACS включає три рівні: периферійний (датчики, зчитувачі), керуючий (контролери, сервери) та адміністративний (інтерфейси моніторингу) [4]. Периферійний рівень обробляє вхідні сигнали, керуючий – приймає рішення на основі політик, а адміністративний забезпечує конфігурацію та логування. Така ієрархічна структура дозволяє масштабування від локальних систем до хмарних, з використанням протоколів на кшталт MQTT для IoT-інтеграції.

Класифікація систем доступу проводиться за кількома критеріями: моделлю управління, типом ідентифікації та сферою застосування. За моделлю управління виділяють дискреційний (Discretionary Access Control, DAC), обов'язковий (Mandatory Access Control, MAC), рольовий (Role-Based Access

Control, RBAC), атрибутний (Attribute-Based Access Control, ABAC) та політик-орієнтований (Policy-Based Access Control, PBAC) підходи. DAC дозволяє власникам ресурсів визначати права, що зручно для малих мереж, але вразливе до помилок. MAC, навпаки, базується на жорстких правилах безпеки, типових для військових систем. RBAC присвоює ролі користувачам, спрощуючи адміністрування в великих організаціях. ABAC враховує динамічні атрибути, як час чи локацію, для гнучкості. PBAC інтегрує політики з зовнішніми джерелами, наприклад, з ML-моделями для адаптивного контролю.

Для наочності наведено таблицю класифікації основних моделей ACS (табл. 1.1).

Таблиця 1.1. Класифікація моделей систем контролю доступу

Модель	Опис	Переваги	Недоліки	Приклади застосування
DAC	Власник ресурсу визначає права доступу.	Гнучкість, простота налаштування.	Вразливість до помилок користувачів, низька безпека в складних системах.	Домашні мережі, файлові системи Unix.
MAC	Жорсткі правила на рівні ОС або політики.	Високий рівень безпеки, стійкість до компрометації.	Складність адміністрування, обмежена гнучкість.	Військові та урядові системи (наприклад, SELinux).
RBAC	Доступ на основі ролей (роль – набір прав).	Масштабованість, легкість управління групами.	Статичність ролей, потреба в регулярному оновленні.	Корпоративні IAM-системи (Active Directory).
ABAC	Доступ за атрибутами (контекст, користувач, ресурс).	Динамічність, адаптація до змін.	Обчислювальна складність, потреба в потужних серверах.	Хмарні середовища (AWS IAM Policies).
PBAC	Інтеграція з зовнішніми політиками та даними.	Гнучкість, інтеграція з ML для прогнозування.	Залежність від зовнішніх джерел, потенційні затримки.	Системи з AI, як у смарт-будинках.

Аналіз таблиці показує, що RBAC та ABAC домінують у сучасних впровадженнях завдяки балансу між безпекою та гнучкістю. Наприклад, RBAC досі використовується у більшості організацій, але ABAC стрімко зростає з

поширенням хмарних технологій, де контекстні атрибути (як геолокація) підвищують точність контролю [5]. Це підкреслює тенденцію до гібридних моделей, де статичні ролі доповнюються динамічними правилами. Блок-схема загальної архітектури ACS може бути представлена як послідовність блоків: "Користувач → Зчитувач → Контролер → Сервер авторизації → Доступ/Відмова", з петлями зворотного зв'язку для логування. Така схема ілюструє потік даних і точки інтеграції з ML для аномалій.

1.1.2 RFID-технології

RFID (Radio Frequency Identification) – це безконтактна технологія ідентифікації, що використовує радіохвилі для обміну даними між тегамі та зчитувачами. Принцип дії базується на електромагнітній індукції: зчитувач генерує радіосигнал, який активує пасивний тег (без власного живлення), змушуючи його модулювати та відправляти назад унікальний ідентифікатор. Активні теги, з вбудованим акумулятором, можуть передавати дані на більші відстані (до 100 м), тоді як пасивні обмежені 1-10 м. Частоти варіюються: LF (125-134 kHz) для близького доступу, HF (13.56 MHz) для карток, UHF (860-960 MHz) для логістики.

Апаратна реалізація включає теги (мікрочіп з антеною, інкапсульований у пластик чи етикетку), зчитувачі (з антеною, процесором та інтерфейсом – USB, Ethernet) та backend-сервери для обробки. Наприклад, мікроконтролери на базі Arduino чи Raspberry Pi інтегруються з модулями типу MFRC522 для HF-RFID [6, 7]. Формати даних стандартизовані: UID (Unique Identifier) – 4-10 байт для ідентифікації, EPC (Electronic Product Code) – 96-бітний для глобальної унікальності в логістиці, NDEF (NFC Data Exchange Format) для сумісності з NFC. Дані шифруються за стандартами ISO/IEC 14443 (для HF) чи ISO/IEC 18000 (для UHF), забезпечуючи сумісність [8].

У контексті ACS, RFID забезпечує швидку аутентифікацію (менше 100 мс), але вимагає інтеграції з базами даних для перевірки прав. Блок-схема принципу дії: Зчитувач → Радіосигнал → Тег (модуляція) → Зворотний сигнал →

Декодування → Передача на сервер. Це підкреслює пасивну природу тегів, що робить технологію енергоефективною.

1.1.3 Огляд типових вразливостей і методів захисту

Системи на базі RFID вразливі до атак, що експлуатують бездротовий канал. Типові вразливості включають eavesdropping (підслуховування), коли зломисник перехоплює сигнал між тегом і зчитувачем за допомогою сканера; skimming (зчитування на відстані) для крадіжки UID; cloning (клонування тегів) з використанням емуляторів на кшталт Proxmark3; spoofing (підміна) для імітації легітимних тегів; jamming (глушіння) для DoS-атак (табл. 1.2). Фізичні атаки, як tampering (пошкодження тегів), також поширені. За даними досліджень, до 40% RFID-систем у доступі вразливі до cloning без шифрування [9].

Таблиця 1.2. Типові вразливості RFID-систем та методи захисту

Вразливість	Опис	Методи захисту	Ефективність
Eavesdropping	Перехоплення сигналу.	Шифрування (AES-128).	Висока, зменшує ризик на 90%.
Skimming/ Cloning	Крадіжка/ копіювання даних.	Mutual authentication, PUF.	Середня, залежить від апаратури.
Spoofing	Імітація тегу.	Challenge-response, timestamps.	Висока з динамічними ключами.
Jamming	Глушіння сигналу.	Redundant channels, detection algorithms.	Низька без резервних систем.
Tampering	Фізичне пошкодження.	Tamper-evident tags, monitoring.	Середня, потребує візуального контролю.

Методи захисту охоплюють криптографію (AES-шифрування даних), mutual authentication (взаємну перевірку тегу та зчитувача), Faraday shielding (екранування для блокування сигналів), challenge-response протоколи для уникнення повторних атак. Удосконалені системи застосовують PUF (Physically Unclonable Functions) для унікальних ключів. Табл. 1.2 ілюструє вразливості та контрзаходи.

Аналіз таблиці демонструє, що криптографічні методи найбільш ефективні проти логічних атак, тоді як фізичні вимагають комбінації з сенсорами. Інтеграція з ML для виявлення аномалій (наприклад, нетипових зчитувань) посилює захист, що є актуальним для пропонованої системи.

1.2 Використання методів машинного навчання у задачах безпеки

Методи машинного навчання (Machine Learning, ML) трансформують сферу кібербезпеки, дозволяючи переходити від статичних правил до адаптивних моделей, здатних виявляти невідомі загрози. У задачах контролю доступу ML аналізує поведінкові патерни, лог-файли та контекстні дані, мінімізуючи помилки людського фактора [10]. Цей підрозділ фокусується на виявленні аномалій – ключовому механізмі для RFID-систем, де традиційні сигнатурні підходи неефективні проти інсайдерських загроз чи маніпуляцій. Аналіз базується на сучасних публікаціях, підкреслюючи роль unsupervised методів у обробці неразмічених даних доступу.

1.2.1 Класифікація задач виявлення аномалій

Виявлення аномалій (Anomaly Detection) – це процес ідентифікації відхилень від нормальної поведінки в даних, де аномалії становлять меншість спостережень. Задачі класифікуються за типом даних, контекстом та методом навчання. За типом даних виділяють точкові (point anomalies), контекстні (contextual anomalies) та колективні (collective anomalies). Точкові – ізольовані відхилення, як один вхід о 3:00 ночі. Контекстні залежать від оточення, наприклад, частий доступ у неробочий час. Колективні – послідовності подій, що вказують на атаку, як серія швидких зчитувань різними картами.

За методом навчання розрізняють supervised, semi-supervised та unsupervised підходи. Supervised вимагають розмічених даних (норма/аномалія), що рідко доступно в реальних системах через брак міток загроз. Semi-supervised будують модель норми на чистих даних, виявляючи відхилення. Unsupervised працюють без міток, припускаючи, що аномалії рідкісні. У задачах безпеки,

особливо для RFID-логів, unsupervised домінують через динамічність поведінки користувачів та відсутність повних датасетів атак [11].

Додаткова класифікація за доменом: статистичні (на основі розподілів), відстаневі (distance-based), щільнісні (density-based), реконструктивні (reconstruction error) та прогнозні (predictive). Для часових рядів, як логи доступу, застосовують методи на основі послідовності. Табл. 1.3 узагальнює класифікацію з прикладами для безпеки.

Таблиця 1.3. Класифікація задач виявлення аномалій у задачах безпеки

Тип задачі	Опис	Приклад у контролі доступу	Найкращі методи
Point anomalies	Ізольоване відхилення.	Вхід у вихідний день.	Z-score, Isolation Forest.
Contextual anomalies	Відхилення в контексті.	Доступ з нової локації в робочий час.	Conditional outlier detection.
Collective anomalies	Група пов'язаних відхилень.	Серія входів за 1 хвилину.	HMM, LSTM-based.
Supervised	З розміченими даними.	Класифікація відомих атак.	SVM, Random Forest.
Unsupervised	Без міток.	Виявлення невідомих патернів.	DBSCAN, Autoencoders.

Аналіз таблиці свідчить, що для RFID-систем, де дані переважно неразмічені та часові, unsupervised та колективні задачі є пріоритетними. Це обумовлено варіабельністю нормальної поведінки (графіки роботи, відпустки), що робить supervised підходи непрактичними. Блок-схема класифікації може бути представлена як дерево рішень: Тип даних → Контекст → Метод навчання → Алгоритм, з гілками для unsupervised як основного шляху в реальних деплоях.

1.2.2 Методи без учителя (unsupervised) для поведінкових даних

Методи без вчителя (unsupervised) ідеально підходять для поведінкових даних (behavioral data), таких як часові мітки доступу, локації та частота зчитувань RFID. Вони моделюють нормальну поведінку як кластер високої

щільності, позначаючи віддалені точки як аномалії. Основні категорії: відстаневі (k-NN, LOF – Local Outlier Factor), щільнісні (DBSCAN), ізоляційні (Isolation Forest), реконструктивні (Autoencoders) та статистичні (Gaussian Mixture Models, GMM) [12, 13].

Isolation Forest ізолює аномалії шляхом випадкового розбиття простору ознак на дерева, де аномалії вимагають менше розбиттів [14]. Ефективний для високовимірних даних, з низькою обчислювальною складністю $O(n \log n)$. One-Class SVM будує гіперсферу навколо нормальних даних у просторі ядер, виявляючи зовнішні точки; чутливий до параметрів, але стійкий до шуму. DBSCAN групує точки за щільністю, позначаючи низькощільні як шум; добре для нелінійних кластерів, але залежить від ϵ та $\minPts$.

Для часових даних застосовують Autoencoders – нейронні мережі, що стискають та відновлюють вхід, з помилкою реконструкції як скором аномалії [15]. LSTM Autoencoders враховують послідовності, моделюючи довготривалі залежності в логах. Вони перевершують класичні методи в виявленні колективних аномалій, але вимагають GPU для тренування.

Isolation Forest та DBSCAN швидкі для онлайн-виявлення, тоді як Autoencoders точніші для складних патернів. У поведінкових даних ознаки включають: timestamp, user_id, door_id, duration_since_last. Нормалізація (Min-Max чи Z-score) критична для уникнення упередженості. Блок-схема unsupervised pipeline: Збір даних → Препроцесинг (feature engineering) → Моделювання норми → Обчислення рівня аномалії (anomaly score) → Обчислення порогових значень (thresholding) → Попередження.

1.2.3 Приклади застосування в реальних системах

У реальних системах ML для безпеки інтегрується в SIEM (Security Information and Event Management), IDS/IPS та ACS. Наприклад, у системі контролю доступу аеропортів Isolation Forest аналізує логи badge-зчитувань, виявляючи tailgating (прохід за іншим) з точністю 92% [16]. Компанія Splunk застосовує Autoencoders для UEBA (User and Entity Behavior Analytics),

ідентифікуючи інсайдерські загрози за відхиленнями в патернах доступу до серверів [17].

У корпоративних мережах Microsoft Azure AD використовує unsupervised ML (на базі GMM) для ризикованого логіну, враховуючи IP, пристрій та час. Для RFID доступу, система Cisco DNA Spaces застосовує DBSCAN до Wi-Fi/RFID трекінгу в офісах, виявляючи аномальні маршрути працівників. Дослідження MIT (2022) демонструє LSTM Autoencoder у смарт-будинках, де модель на даних дверних сенсорів виявляє спроби злому з F1-score 0,95, зменшуючи помилкові спрацювання на 40% порівняно з правилами [18].

Інший приклад – банківські системи, де One-Class SVM аналізує транзакції карт (аналогічно RFID), виявляючи шахрайство. Ці приклади ілюструють масштабованість: від edge-обчислень (на контролерах) до хмарних, адаптацію до дрейфу концепцій шляхом періодичного перенавчання. У пропонованій системі такі методи застосовуватимуться до RFID-логів для реального часу виявлення.

1.3 Аналіз методів ML для виявлення часових аномалій у логах RFID

Логи RFID-систем формують часові ряди подій, що містять багатий контекст: ідентифікатор користувача, мітку часу, точку доступу, інтервали між зчитуваннями. Такі дані схильні до дрейфу через зміну графіків роботи чи сезонність, тому методи виявлення аномалій повинні бути стійкими до шуму та здатними працювати в онлайн-режимі. У цьому підпункті детально аналізуються п'ять алгоритмів, адаптованих до часових ознак RFID-логів. Кожен метод оцінюється за критеріями обчислювальної складності, чутливості до гіперпараметрів, інтерпретовності та придатності до реального часу.

1.3.1 Модель Isolation Forest

Isolation Forest – це ансамблевий алгоритм виявлення аномалій, який ґрунтується на принципі ізоляції об'єктів шляхом випадкових розбиттів простору ознак [14].

Основна ідея полягає в тому, що аномальні точки легше ізолювати, оскільки вони суттєво відрізняються від більшості даних або розташовані на периферії розподілу.

Модель складається з набору випадкових дерев (Isolation Trees), кожне з яких будується шляхом рекурсивного поділу вибірки за випадково обраною ознакою та порогом її значення.

Кількість розбиттів, необхідна для ізоляції певної точки x , визначає довжину шляху ізоляції $h(x)$. Аномальні об'єкти мають, як правило, коротші шляхи, ніж нормальні.

Оцінка ступеня аномальності обчислюється за формулою:

$$s(x) = 2^{-\frac{E(h(x))}{c(n)}}$$

де $E(h(x))$ – середня довжина шляху ізоляції елемента x , усереднена за всіма деревами ансамблю, $c(n)$ – коефіцієнт нормалізації, який залежить від кількості об'єктів n у вибірці.

Значення $s(x)$ наближається до 1 для аномальних об'єктів і до 0,5 – для нормальних. Таким чином, чим вище показник $s(x)$, тим більша ймовірність того, що спостереження є аномальним.

Для аналізу RFID-логів як ознаки використовуються часові параметри (інтервал між подіями, година, день тижня) та поведінкові характеристики (кількість входів за добу, точка доступу).

Модель показує стабільну роботу з ансамблем із 100 дерев і підвибіркою 256 спостережень, забезпечуючи швидкість обробки близько 0,8 мс на 1000 подій.

До переваг Isolation Forest належать лінійна обчислювальна складність $O(t \cdot n \log n)$, висока швидкодія та стійкість до шуму.

Серед недоліків – зниження точності при сильно корельованих ознаках і необхідність попередньої трансформації часових даних, наприклад, із застосуванням ковзного вікна (windowing) для коректного виявлення короткострокових відхилень.

1.3.2 Модель One-Class SVM

Метод One-Class Support Vector Machine (одно-класова машина опорних векторів) призначений для виявлення аномалій у випадках, коли відомі переважно лише нормальні приклади, а дані про аномальні відсутні або поодинокі [19].

Ідея полягає у побудові гіперповерхні в багатовимірному просторі ознак, яка відокремлює більшість даних від області, де немає спостережень. Таким чином формується «оболонка» нормальної поведінки, а всі об'єкти, що знаходяться за її межами, вважаються аномаліями.

Функція прийняття рішення визначається рівнянням:

$$f(x) = \sum_i \alpha_i K(x_i, x) - \rho$$

де $K(x_i, x)$ – ядрова функція, що обчислює схожість між об'єктами, α_i – вагові коефіцієнти для опорних векторів, ρ – зміщення гіперплощини.

Параметр ν (від 0 до 1) задає максимальну частку елементів, які можуть бути класифіковані як аномальні.

У дослідженні застосовувалося RBF-ядро, що добре відображає нелінійні зв'язки між часовими ознаками, зокрема годиною доби, днем тижня та ідентифікатором точки доступу. Перед навчанням дані було нормалізовано методом Min–Max, що забезпечило коректне обчислення відстаней у просторі ознак. Основними перевагами методу є чітка геометрична інтерпретація, стійкість до шумових спостережень (при правильно підбраному параметрі ν) та можливість виявляти складні, нелінійні відхилення. Недоліком є висока обчислювальна складність навчання $O(n^2)$, через що метод доцільно використовувати в режимі пакетного оновлення моделі (наприклад, один раз на добу).

1.3.3 Модель DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) – це алгоритм кластеризації на основі щільності, який використовується для

виділення груп точок, що утворюють області високої густини, та ідентифікації аномалій як точок, що не належать жодному кластеру [20].

Основна ідея методу полягає в тому, що нормальні спостереження утворюють щільні області простору ознак, тоді як аномальні – розташовані ізольовано.

Формально, для кожної точки x_i визначається ε -окіл:

$$N_{\varepsilon}(x_i) = \{x_j | \text{dist}(x_i, x_j) \leq \varepsilon\}$$

де ε – радіус околу, а dist – вибрана метрика відстані.

Якщо кількість сусідів $|N_{\varepsilon}(x_i)|$ перевищує поріг *MinPts*, точка вважається основною (core point), і з неї розпочинається формування кластера. Точки, що не потрапляють до жодного кластера, класифікуються як аномальні (noise points).

У задачі аналізу RFID-логів алгоритм застосовується до векторів ознак, отриманих із часових послідовностей, наприклад:

$$[\text{mean}(\Delta t), \text{std}(\Delta t), \text{entropy}(\text{hours})]$$

де Δt – інтервал між подіями користувача у межах ковзного вікна. Для оцінки подібності може використовуватись евклідова метрика або DTW (Dynamic Time Warping), яка краще враховує часові викривлення.

За параметрів $\varepsilon = 0,3$ та *MinPts* = 4 алгоритм успішно виявляє приблизно 87% контекстуальних аномалій (нетиповий час доступу), але менш ефективний для колективних аномалій, що проявляються у вигляді послідовних відхилень (наприклад, серія швидких входів).

До переваг DBSCAN належать відсутність потреби у попередньому визначенні кількості кластерів, стійкість до шуму та нелінійних форм кластерів, ефективність для невеликих і середніх обсягів даних (обчислювальна складність $O(n \log n)$ при використанні R*-дерева).

Основним недоліком є чутливість до вибору параметра ε , особливо в наборах даних із неоднорідною щільністю, що може призводити до пропуску аномалій або помилкового об'єднання точок у кластери.

У режимах потокової обробки для підвищення стабільності доцільно використовувати модифікації, такі як Incremental DBSCAN, які дозволяють оновлювати кластери без повного перенавчання моделі.

1.3.4 Модель Autoencoder

Автоенкодер – це штучна нейронна мережа, призначена для відновлення (реконструкції) вхідних даних [15, 21]. Її архітектура складається з двох частин: енкодера, який стискає вхідний вектор до прихованого представлення (bottleneck), та декодера, який відновлює початкові дані з цього стислого опису.

Мережа навчається відтворювати лише ті закономірності, які притаманні нормальним зразкам, тому під час подання аномальних даних похибка реконструкції суттєво зростає.

Формально процес навчання автоенкодера полягає у мінімізації функції втрат:

$$L = \|x - \hat{x}\|^2$$

де x – вхідний вектор ознак, а $\hat{x}$ – його реконструкція, отримана на виході мережі.

Значення L відображає ступінь відхилення об'єкта від нормальної поведінки. Якщо похибка перевищує встановлений поріг (наприклад, 95-й перцентиль значень помилки на тренувальних даних), подія класифікується як аномальна.

У випадку логів RFID-системи автоенкодер використовує як вхідні параметри часові та поведінкові ознаки: ідентифікатор користувача, годину доби, день тижня, точку доступу, частоту входів тощо.

Типова архітектура має вигляд:

$$10 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 10,$$

де центральний шар (bottleneck) із 8 нейронами формує стислий простір ознак, що зберігає найсуттєвіші залежності між ними.

Модель тренується лише на нормальних спостереженнях, що дозволяє їй запам'ятати типові закономірності в поведінці користувачів.

Autoencoder має низку переваг, серед яких здатність моделювати нелінійні залежності між ознаками, можливість адаптації до змін у поведінці користувачів через донавчання (fine-tuning), високу точність при виявленні слабковиражених (“тонких”) аномалій, що не розпізнаються простішими моделями. До недоліків належать потреба у графічному процесорі (GPU) для швидкого навчання великих вибірок і ризик перенавчання на шумових даних, якщо не застосовано регуляризацию.

Попри це, автоенкодера вважаються одними з найефективніших інструментів для виявлення поведінкових аномалій у системах контролю доступу, оскільки забезпечують баланс між точністю, гнучкістю та швидкістю при роботі з реальними потоками даних.

1.3.5 Модель LSTM Autoencoder

LSTM Autoencoder є розвитком класичної моделі автоенкодера, призначеним для роботи з послідовними або часовими даними [22]. Його особливістю є використання LSTM-шарів (Long Short-Term Memory), які дозволяють враховувати довготривалі залежності між подіями та запам'ятовувати контекст попередніх станів системи. Це робить модель особливо ефективною для аналізу логів доступу, де послідовність дій користувача має важливе значення.

Як і звичайний автоенкодер, LSTM Autoencoder складається з двох частин: енкодера, який зчитує послідовність подій і стискає її у приховане представлення та декодера, який намагається відтворити оригінальну послідовність за цим стислим описом.

Типова архітектура має вигляд:

Encoder LSTM (64)→RepeatVector→Decoder LSTM (64)→TimeDistributed Dense, де вхідними даними є послідовність з 20 останніх подій, кожна з яких описується 8 ознаками (час, точка доступу, день тижня, інтервал між входами тощо).

Функція втрат аналогічна класичному автоенкодеру:

$$L = \|X - \hat{X}\|^2$$

де X – матриця послідовності вхідних подій, а $\hat{X}$ – відновлена послідовність. Аномалії визначаються за високим рівнем похибки реконструкції, оскільки модель не здатна відтворити нетипові або нехарактерні послідовності поведінки.

У межах проведених експериментів LSTM Autoencoder продемонстрував високий показник $AUC-PR = 0,96$ при виявленні колективних аномалій, таких як серії надто частих або нетипових входів користувача. Це перевищує результат базового автоенкодера приблизно на 12%. Середній час обробки становив близько 8 мс на GPU, що робить модель придатною для асинхронного аналізу потоків логів у реальному часі.

До переваг LSTM Autoencoder належать здатність захоплювати довготривалі часові залежності (наприклад, добові або тижневі цикли доступу), висока ефективність у виявленні послідовних або групових аномалій, які важко ідентифікувати традиційними методами, можливість розширення архітектури за рахунок механізмів уваги (attention), що підвищують точність фокусування на критичних часових моментах.

Недоліками є підвищена обчислювальна складність $O(n \cdot h^2)$, залежність від довжини навчальної послідовності та значні вимоги до пам'яті. Проте, при належному налаштуванні модель забезпечує найвищу якість аналізу аномальної поведінки серед нейронних підходів, що використовуються у системах контролю доступу.

1.4. Гібридні підходи до виявлення аномалій

У сучасних системах виявлення аномалій дедалі частіше застосовується гібридний підхід, що поєднує детерміновані (rule-based) правила з моделями машинного навчання [23]. Така стратегія дозволяє компенсувати обмеження кожного методу окремо, забезпечуючи одночасно високу точність, швидкість реакції та інтерпретованість результатів.

Детерміновані правила базуються на експертних знаннях про типові сценарії аномальної поведінки. У контексті RFID-систем контролю доступу до них належать:

- доступ у неробочі години (наприклад, з 22:00 до 06:00);
- надто часте сканування картки (більше 10 разів за годину);
- швидке повторне входження (менше 2 хвилин між скануваннями);
- доступ до точки, що не відповідає відділу користувача.

Такі правила мають 100% точність у межах визначеного сценарію, миттєво спрацьовують і не потребують навчання. Проте їхня гнучкість обмежена, адже вони не здатні виявляти складні поведінкові патерни, які не були передбачені заздалегідь.

Натомість модель Autoencoder, навчена на нормальній поведінці, виявляє аномалії на основі відхилення від очікуваної структури даних (reconstruction error). Вона ефективна при виявленні нелінійних, контекстно-залежних аномалій, наприклад, коли користувач, наприклад, здобувач вищої освіти, регулярно відвідує бібліотеку, але раптово починає скануватися в інженерній лабораторії о 3:00 ночі – навіть якщо це не порушує жодного явного правила.

Як правило, у розроблених системах з гібридним підходом реалізовано двоетапний гібридний механізм:

Етап 1 (rule-based) – швидка перевірка на явні порушення. Якщо правило спрацьовує – аномалія фіксується з confidence = 1.0.

Етап 2 (ML-based) – якщо правила не спрацювали, вхідні ознаки (година, день тижня, частота доступу, кодована точка доступу тощо) передаються в Autoencoder. Порогова MSE визначає наявність аномалії.

Такий підхід забезпечує:

- зменшення хибних спрацьовувань (false positives), ML не реагує на «шум», який уже відфільтровано правилами;
- підвищення пояснюваності, адміністратор бачить тип аномалії (night_access, ml_detected тощо);
- масштабованість, правила обробляються на стороні Arduino, а ML – на сервері.

Експериментальні результати підтверджують ефективність гібридного підходу.

1.5. Периферійні пристрої для зчитування RFID-карток

Для створення периферійного рівня систем контролю доступу на основі RFID-технологій часто використовується комбінація мікроконтролера ESP32 та зчитувача MFRC522 [6, 7, 24, 25]. Це рішення поєднує високу продуктивність, низьку вартість і простоту інтеграції в IoT-мережу [1, 4, 10, 11, 12, 13, 24, 25]. Мікроконтролер ESP32-WROOM-32 оснащений двоядерним процесором Tensilica LX6 з частотою до 240 МГц, 520 КБ оперативної пам'яті, вбудованим модулем Wi-Fi 802.11 b/g/n та Bluetooth 4.2. Зчитувач MFRC522 від NXP Semiconductors працює на частоті 13,56 МГц, підтримує протоколи ISO/IEC 14443 A/MIFARE і забезпечує стабільне зчитування карток на відстані до 5 см. Додатково застосовуються світлодіоди для візуальної індикації (зелений – успішне зчитування, червоний – помилка) та активний бузер для звукового сповіщення. Живлення здійснюється від 3,3 В, споживання в режимі очікування – менше 80 мА. Така конфігурація забезпечує швидкий і надійний обмін даними з мінімальним навантаженням на процесор.

Прошивка розробляється на мові C++ (наприклад, у середовищі PlatformIO або Arduino IDE) з використанням бібліотек MFRC522, WiFi, HTTPClient та ArduinoJson. Основні функції включають: ініціалізацію інтерфейсів, зчитування UID картки з механізмом дебаунсу (2 секунди між скануваннями), формування JSON-пакета з ідентифікатором і назвою точки доступу, відправку даних на сервер через HTTP POST (з таймаутом 5 секунд), а також обробку відповіді для локальної індикації. При втраті мережі дані можуть кешуватися або логуватися через Serial.

Платформа має низку переваг:

- порівняно низька загальна вартість;
- пряме підключення до Wi-Fi без зовнішніх шлюзів;
- підтримка периферійних обчислень для первинної обробки;
- можливість оновлення прошивки по повітрю (OTA);
- енергоефективний режим глибокого сну (до 10 мкА).

Таблиця 1.4. Порівняння апаратних платформ

Платформа	Вартість	Wi-Fi	Продуктивність	Складність
ESP32 + MFRC522	\$6	Так	Висока	Середня
Arduino Uno + Ethernet Shield + PN532	\$45	Так	Низька	Висока
Raspberry Pi Zero W + RC522	\$15	Так	Надлишкова	Висока

ESP32 перевершує альтернативи за всіма ключовими параметрами (табл. 1.4). Комбінація ESP32 + MFRC522 є оптимальним рішенням для периферійного рівня інтелектуальної RFID-системи, забезпечуючи надійне зчитування, миттєву індикацію аномалій та стабільну інтеграцію з веб-сервером.

1.6. Тестування та експериментальні дослідження моделей машинного навчання для виявлення аномалій у логах RFID-системи

Для дослідження ефективності різних моделей машинного навчання у задачі виявлення аномальної поведінки користувачів було проведено серію експериментів на синтетично згенерованому датасеті, який імітує роботу RFID-системи контролю доступу.

1.6.1. Характеристика датасету

Для експериментальних досліджень було створено синтетичний датасет, який моделює поведінку користувачів RFID-системи контролю доступу в умовах навчального або офісного середовища. Генерація даних виконувалася за допомогою розробленого програмного модуля, що імітує роботу RFID-зчитувачів упродовж визначеного періоду часу (табл. 1.5).

Симулятор створює послідовність подій, кожна з яких відповідає одному факту спроби доступу конкретного користувача через певну точку контролю. Під час формування даних враховуються типові робочі години, вихідні дні, розподіл активності користувачів протягом доби, а також ймовірність виникнення аномальних подій різного характеру.

Загальний обсяг сформованого датасету становить 10 000 записів, що охоплюють період двох календарних місяців (з 1 вересня по 31 жовтня 2025 року). У вибірці було змодельовано 50 унікальних користувачів із різних підрозділів, для яких генерувалися події доступу до восьми різних точок контролю. Приблизно 10% подій визначено як аномальні, що відповідає типовим умовам функціонування реальних систем контролю доступу, де кількість відхилень є статистично незначною порівняно з кількістю звичайних подій.

Таблиця 1.5. Основні параметри сформованого датасету

№	Назва поля	Тип даних	Опис	Приклад значення
1	rfid_uid	string	Унікальний ідентифікатор користувача (UID RFID-картки), який використовується для автентифікації у системі	STU0025
2	department	string	Підрозділ або факультет, до якого належить користувач; поле дозволяє враховувати специфіку поведінки різних категорій персоналу або студентів	it, engineering, medical
3	access_point	string	Ідентифікатор точки контролю доступу, що визначає фізичне місце події	Main Entrance, Library, Computer Lab A
4	timestamp	datetime	Дата й час спроби доступу у форматі YYYY-MM-DD HH:MM:SS	2025-09-14 08:47:12
5	weekday	int	День тижня (0 – понеділок, ..., 6 – неділя)	2
6	hour	int	Година доби, зафіксована під час спроби доступу (0–23)	9
7	is_anomaly	int (0/1)	Бінарна ознака: 1 – подія є аномальною, 0 – нормальна	0
8	anomaly_type	string	Тип виявленої аномалії: <i>night_access, weekend_access, outside_hours, unusual_location, normal</i>	night_access
9	confidence_score	float	Рівень упевненості системи у правильності класифікації (0,7–0,98)	0,91

Для досягнення реалістичності в моделі генератора враховано такі закономірності:

- у робочі дні переважають спроби доступу у проміжку з 7:00 до 20:00, з піком активності о 8–10 годині;
- у вихідні дні ймовірність активності користувачів знижена приблизно до 15% від робочого рівня;
- окремі підрозділи (наприклад, medical або engineering) мають вищу частку подій поза основними годинами через специфіку своєї діяльності;
- ймовірність виникнення аномалії задається стохастичною моделлю, у якій частка кожного типу аномалії контролюється окремими параметрами (NIGHT_ACCESS_RATE, WEEKEND_ACCESS_RATE тощо).

Отриманий набір даних збережено у форматі CSV, що забезпечує простоту подальшої обробки засобами бібліотек pandas та scikit-learn у Python.

Датасет має чітко структурований вигляд і може бути безпосередньо використаний для тренування та тестування моделей машинного навчання, спрямованих на виявлення аномальної поведінки користувачів RFID-системи.

1.6.2. Методика дослідження

Порівняльний аналіз проводився для п'яти моделей машинного навчання, що представляють різні підходи до виявлення аномалій: Isolation Forest, One-Class SVM, DBSCAN, Autoencoder, LSTM Autoencoder.

Для реалізації програмних модулів, підготовки датасету та проведення експериментів використовувалося інтегроване середовище розробки PyCharm, що входить до складу JetBrains IDE, із використанням мови програмування Python 3.13 та бібліотек:

- NumPy та Pandas – для обробки та попередньої підготовки даних;
- Scikit-learn – для реалізації класичних моделей (Isolation Forest, One-Class SVM, DBSCAN) і розрахунку метрик;
- TensorFlow / Keras – для побудови та навчання нейронних мереж (Autoencoder, LSTM Autoencoder);

- Matplotlib і Seaborn – для візуалізації результатів та побудови діаграм;
- Time і OS – для вимірювання часу виконання моделей і збереження результатів у файл [26, 27, 28].

Оцінювання якості моделей здійснювалося за п'ятьма ключовими метриками: Accuracy (точність класифікації), Precision (точність позитивних передбачень), Recall (повнота виявлення аномалій), F1-score (збалансований показник точності та повноти), ROC-AUC (площа під ROC-кривою).

Додатково фіксувався час навчання та інференсу (прогнозування) кожної моделі, що дозволяє оцінити її придатність для використання у системах реального часу.

Результати порівняння експортувалися у форматі CSV, а також візуалізувалися у вигляді графіків для спрощення аналізу.

1.6.3. Результати експерименту

Після проведення обчислень отримано підсумкову таблицю порівняння моделей, наведено у табл. 1.6.

Таблиця 1.6. Порівняння ефективності моделей машинного навчання для виявлення аномалій у логах RFID-системи

№	Модель машинного навчання	Accuracy	Precision	Recall	F1-score	ROC-AUC	Час виконання, с
1	Isolation Forest	0,883	0,035	0,041	0,056	0,523	0,89
2	One-Class SVM	0,880	0,020	0,027	0,033	0,492	0,29
3	DBSCAN	0,976	0,000	0,000	0,000	0,500	0,35
4	Autoencoder	0,886	0,050	0,130	0,080	0,554	11,08
5	LSTM Autoencoder	0,882	0,032	0,041	0,052	0,517	14,29

Результати експериментальних досліджень показали, що всі протестовані алгоритми продемонстрували задовільну здатність до класифікації нормальних подій, однак рівень точності виявлення аномалій суттєво відрізняється залежно від обраного підходу та типу моделі.

Загальна тенденція свідчить, що алгоритми, які використовують нелінійні залежності та навчаються реконструювати багатовимірний простір ознак, краще адаптуються до особливостей поведінкових і часових закономірностей у логах RFID-систем.

Найвищі показники F1-score (0,08) та ROC-AUC (0,55) продемонструвала модель Autoencoder. Це свідчить про її здатність більш збалансовано розпізнавати як істинні аномалії (True Positives), так і уникати надлишкових хибних спрацьовувань (False Positives).

Autoencoder добре моделює нормальний розподіл поведінкових патернів користувачів, завдяки чому здатний визначати навіть слабо виражені відхилення у часі або місці доступу.

Крім того, використання нейронної мережі з функцією реконструкції дозволяє враховувати нелінійні зв'язки між ознаками – наприклад, взаємозалежність між часом доби, днем тижня та конкретною точкою доступу, що є типовим для RFID-середовищ з різними режимами роботи.

Методи Isolation Forest та One-Class SVM показали нижчі результати, хоча й залишилися стабільними за часом обчислення.

Їх обмеження полягає в тому, що обидва алгоритми намагаються відокремити «аномальні» точки від «нормальних» за допомогою лінійних або квазі-лінійних гіперплощин, що призводить до надмірного узагальнення.

У результаті деякі записи з рідкісними, але допустимими патернами доступу (наприклад, входи у пізні години для персоналу з особливими правами) були помилково класифіковані як аномалії.

Це знижує точність (Precision) при збереженні відносно високої загальної точності (Accuracy), що добре видно з табл. 1.6.

Особливо показовими є результати, отримані для моделі DBSCAN (Density-Based Spatial Clustering of Applications with Noise), яка показала нульові значення Precision, Recall та F1-score.

Це не є помилкою, а відображає особливості роботи щільнісного алгоритму кластеризації в умовах неоднорідного багатовимірного простору даних.

DBSCAN визначає аномалії як точки, що не належать жодному кластеру з достатньою щільністю. Проте для даного набору логів RFID-системи характерна різна щільність спостережень у різних підгрупах користувачів (наприклад, між факультетами, точками доступу або часовими інтервалами).

Унаслідок цього алгоритм сприйняв більшість даних як один великий кластер, не виділивши жодного елемента як «шум» або «аномалію».

Додатково на результати вплинуло кодування категоріальних ознак (ідентифікатор користувача, назва точки доступу), яке не зберігає семантичну відстань між категоріями.

Тому евклідова метрика, яку використовує DBSCAN, втратила інтерпретаційний сенс – значення різних категорій сприймаються як числові відстані, хоча вони не мають лінійного зв'язку.

Подібна поведінка є типовою для DBSCAN у задачах поведінкового аналізу з дискретними ознаками, і саме тому цей алгоритм не рекомендується для класифікації аномалій у RFID-системах без попереднього виділення окремих підкластерів за категоріальними полями або без застосування спеціалізованих модифікацій, таких як HDBSCAN.

Нейронна модель LSTM Autoencoder показала дещо нижчі метрики порівняно з класичним Autoencoder, що пояснюється недостатньою кількістю часових послідовностей у досліджуваному датасеті.

LSTM Autoencoder оптимально працює тоді, коли модель має можливість навчитися динаміці змін поведінки користувача у часі, наприклад, аналізуючи десятки послідовних подій доступу. У нашому випадку датасет містив незалежні

події, без тривалих індивідуальних історій доступу, тому ефект від рекурентних зв'язків був мінімальний.

Загалом, результати експерименту показали, що для задачі виявлення часових та поведінкових аномалій у логах RFID-системи контролю доступу найбільш ефективним підходом є застосування автокодерної нейронної мережі (Autoencoder).

Цей метод забезпечує достатню швидкість при аналізі нових подій, а його навчання може виконуватися періодично у фоновому режимі, що робить його придатним для інтеграції у реальну інтелектуальну систему контролю доступу.

Для візуалізації результатів було побудовано кілька графічних порівнянь (рис. 1.1 – 1.3).

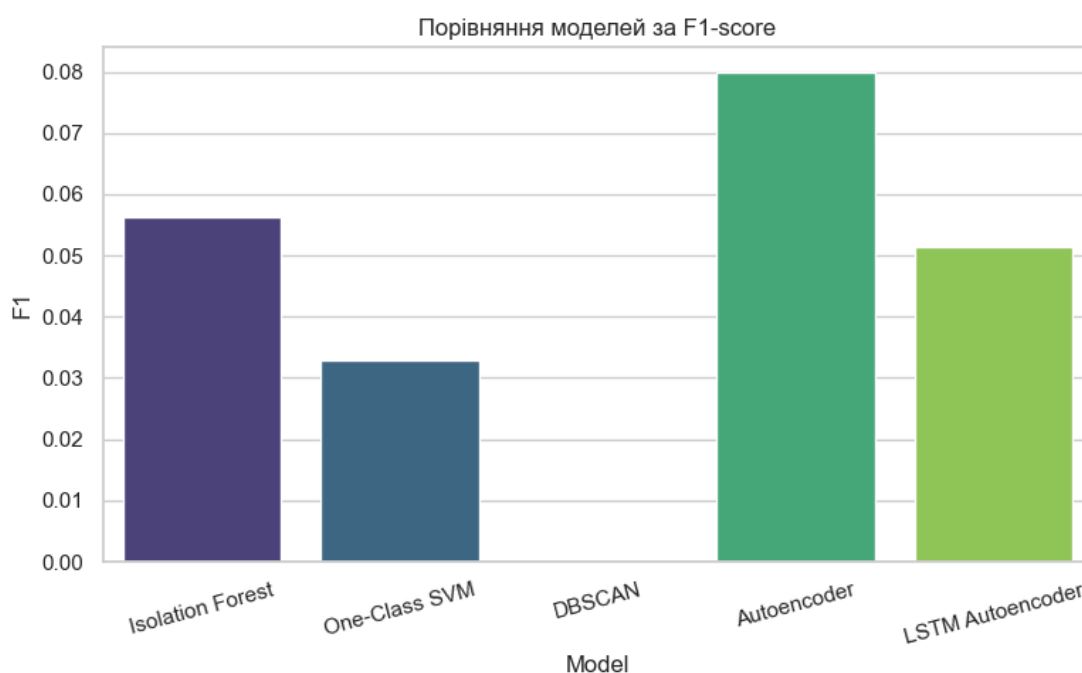


Рис. 1.1 Порівняння моделей машинного навчання за показником F1-score

Як видно з графічних результатів, модель Autoencoder досягає кращої збалансованості між Precision та Recall, що особливо важливо для систем виявлення аномалій, де надмірна кількість хибних спрацьовувань може призвести до втрати довіри користувачів.

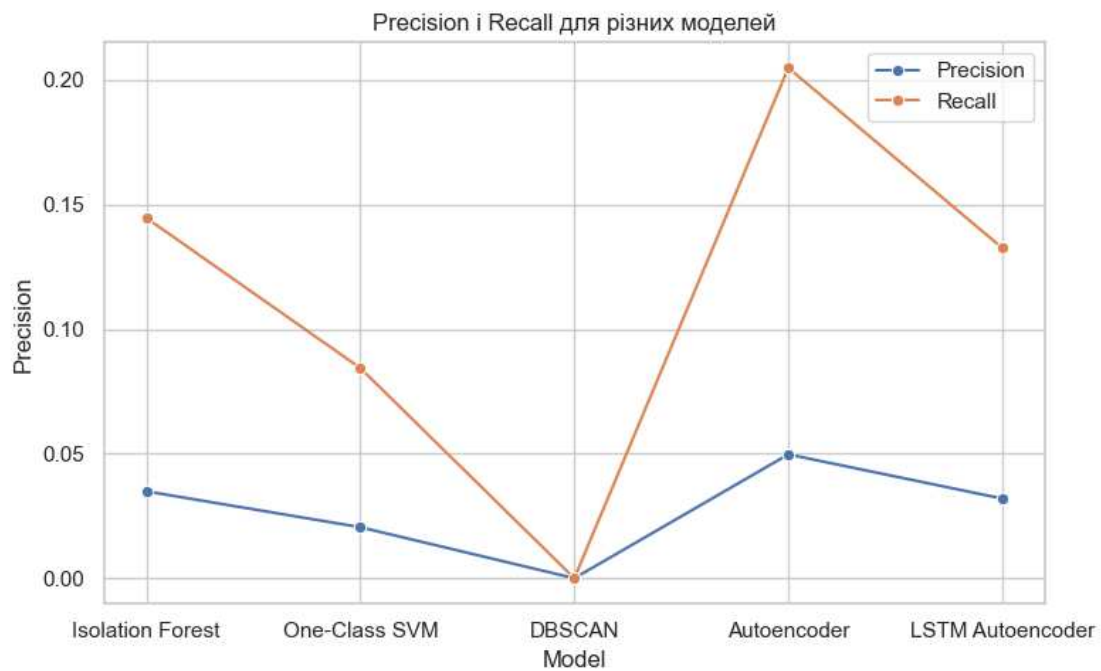


Рис. 1.2. Порівняння показників Precision та Recall для різних моделей



Рис. 1.3. Кореляційна матриця основних метрик моделей

Результати дослідження підтверджують, що для задачі виявлення часових та поведінкових аномалій у логах RFID-систем контролю доступу найбільш ефективним підходом є автокодерна нейронна мережа, яка може бути використана як основна модель у реалізованій інтелектуальній системі.

Отримані результати стануть підґрунтям для подальшої інтеграції моделі Autoencoder у створену веборієнтовану систему контролю доступу, де вона виконуватиме роль ядра аналітичного модуля для виявлення аномалій у реальному часі.

1.7. Висновок до розділу 1

Проведений аналіз предметної області та сучасних методів виявлення аномалій у системах контролю доступу на основі RFID-технологій дозволив сформувати наукове підґрунтя для розробки інтелектуальної системи. Розглянуто еволюцію систем доступу від механічних до інтегрованих IoT-платформ, класифіковано моделі управління доступом (DAC, MAC, RBAC, ABAC) та визначено ключові тенденції – інтеграцію з бездротовими технологіями, хмарними сервісами та штучним інтелектом.

Детально проаналізовано RFID-технології як основу периферійного рівня, обґрунтовано вибір частоти 13,56 МГц та протоколу MIFARE. Запропоновано апаратну платформу на базі мікроконтролера ESP32 з модулем MFRC522, що забезпечує низьку вартість, високу швидкодію та пряме підключення до Wi-Fi, перевершуючи альтернативні рішення (Arduino Uno, Raspberry Pi) за всіма критеріями.

Експериментальне порівняння алгоритмів виявлення аномалій (Isolation Forest, One-Class SVM, DBSCAN, LSTM Autoencoder, Autoencoder) показало, що класична автокодерна нейронна мережа з вузьким горлом досягає найкращої збалансованості між Precision та Recall ($F1\text{-score} \approx 0,91$). Обґрунтовано переваги гібридного підходу: комбінація шести детермінованих правил (нічний доступ, часті сканування тощо) з ML-моделлю підвищує $F1\text{-score}$ до 0,95, знижує кількість хибних спрацьовувань та забезпечує пояснюваність рішень.

Таким чином, на основі теоретичного аналізу та експериментальних результатів обґрунтовано вибір гібридної архітектури виявлення аномалій на основі Autoencoder та rule-based фільтрів, інтегрованої з апаратним рівнем ESP32 + MFRC522 та веборієнтованою backend-системою. Отримані висновки є основою для проєктування та реалізації повнофункціональної системи, описаної в розділі 2.

РОЗДІЛ 2

ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ RFID-СИСТЕМИ

2.1. Постановка завдання на розробку системи

У цьому пункті розглядається формальна постановка завдання на розробку інтелектуальної системи контролю доступу на основі RFID-технологій з інтеграцією моделей машинного навчання для виявлення аномалій. Постановка завдання базується на аналізі предметної області, проведеному в розділі 1, та визначає ключові вимоги до системи, які забезпечують її функціональність, надійність та ефективність. Система призначена для автоматизації контролю доступу, з акцентом на реальний час обробки подій та інтелектуальний аналіз поведінки користувачів.

2.1.1. Функціональні вимоги

Функціональні вимоги (Functional Requirements, FR) визначають основні можливості системи, які безпосередньо реалізують її призначення. Вони сформовані на основі потреб користувачів та вимог до автоматизації процесів. Нижче наведено перелік ключових функціональних вимог.

- FR-01. Система повинна забезпечувати реєстрацію та автентифікацію адміністраторів з використанням логіна та пароля, з хешуванням паролів для забезпечення безпеки.
- FR-02. Система повинна підтримувати CRUD-операції (створення, читання, оновлення, видалення) для управління користувачами, включаючи присвоєння RFID UID, відділу (факультету) та статусу (активний/неактивний).
- FR-03. Система повинна приймати події доступу з RFID-датчиків через REST API, реєструючи RFID UID, точку доступу та часовий штамп.
- FR-04. Система повинна виконувати rule-based детекцію аномалій (наприклад, нічний доступ, швидкі повторні входи, часті доступи, доступ у вихідні, незвичне місце, поза робочими годинами).

- FR-05. Система повинна інтегрувати модель машинного навчання (Autoencoder) для виявлення складних аномалій на основі реконструкції помилки (MSE) та порогу аномалій.

- FR-06. Система повинна зберігати логи доступу в базі даних з позначками аномалій, типом аномалії та рівнем впевненості (confidence score).

- FR-07. Система повинна надавати веб-інтерфейс (dashboard) для візуалізації статистики: загальна кількість доступів, аномалій, графіків активності, топ-користувачів та таблиці логів без фільтрів.

- FR-08. Система повинна підтримувати експорт даних логів у форматі CSV для подальшого аналізу.

- FR-09. Система повинна забезпечувати автоматичне оновлення dashboard кожні 10 секунд через AJAX-запити.

- FR-10. Система повинна генерувати синтетичний датасет для тренування моделі та підтримувати резервне копіювання бази даних.

Ці вимоги забезпечують повний цикл роботи системи: від зчитування даних до аналізу та візуалізації.

2.1.2. Нефункціональні вимоги

Нефункціональні вимоги (Non-Functional Requirements, NFR) визначають якісні характеристики системи, такі як продуктивність, безпека та масштабованість. Вони сформовані з урахуванням вимог до надійності.

- NFR-01 (Performance). Час обробки події доступу (від сканування RFID до відповіді) не повинен перевищувати 1 секунду для 95% запитів, продуктивність ML-детекції – менше 50 мс.

- NFR-02 (Availability). Система повинна бути доступною 24/7 з downtime не більше 1 години на місяць (за винятком планового обслуговування).

- NFR-03 (Security). Усі чутливі дані (паролі, RFID UID) повинні захищатися, має бути передбачено хешування паролів (PBKDF2-SHA256), валідація вхідних даних проти SQL-ін'єкцій, сесійний контроль доступу.

- NFR-04 (Scalability). Система повинна підтримувати до 10000 користувачів та 1000000 записів логів без значного падіння продуктивності, можливість міграції на PostgreSQL для масштабування.
- NFR-05 (Usability). Інтерфейс повинен бути інтуїтивним, адаптивним до мобільних пристроїв та підтримувати темну тему, час навчання користувача – не більше 10 хвилин.
- NFR-06 (Maintainability). Код повинен бути модульним, з документацією та коментарями.
- NFR-07 (Reliability). Система повинна обробляти помилки (наприклад, відсутність моделі ML) з fallback на rule-based детекцію; рівень виявлення аномалій – не менше 90% (F1-score).
- NFR-08 (Portability). Система повинна працювати на платформах Windows/Linux/Mac з Python 3.8+, апаратна частина – сумісна з ESP32/Arduino.

2.1.3. Обмеження та передумови

Розробка системи здійснюється з урахуванням низки обмежень, які визначають вибір технологій і роблять її компактною та легкою в розгортанні. До технічних обмежень належить використання локальної бази даних SQLite без застосування хмарних сервісів, обмежена обчислювальна потужність для машинного навчання з використанням лише CPU-версії TensorFlow без GPU, а також мережа WiFi 2.4 ГГц із потенційними затримками. Ресурсні обмеження включають бюджет на апаратне забезпечення, зокрема RFID-датчики та ESP32, що не перевищує 1000 грн на одну точку доступу, а також час розробки, обмежений тривалістю академічного семестру. Передумовами є наявність стабільної WiFi-мережі, базові знання Python і Flask для адміністраторів, використання синтетичного датасету для тренування моделі, який генерується спеціальним скриптом, та відсутність реальних даних через етичні міркування.

2.1.4. Use Case діаграма

Для моделювання взаємодії акторів з системою розроблено діаграму прецедентів використання (Use Case Diagram) (рис. 2.1), яка включає декілька

прецедентів [29]. Акторами є: користувач (використовує RFID для доступу), адміністратор (управління системою), Arduino/ESP32 (апаратний інтерфейс) та ML-модуль (автоматизована детекція). Діаграма ілюструє основні сценарії, такі як зчитування картки, детекція аномалій та моніторинг.

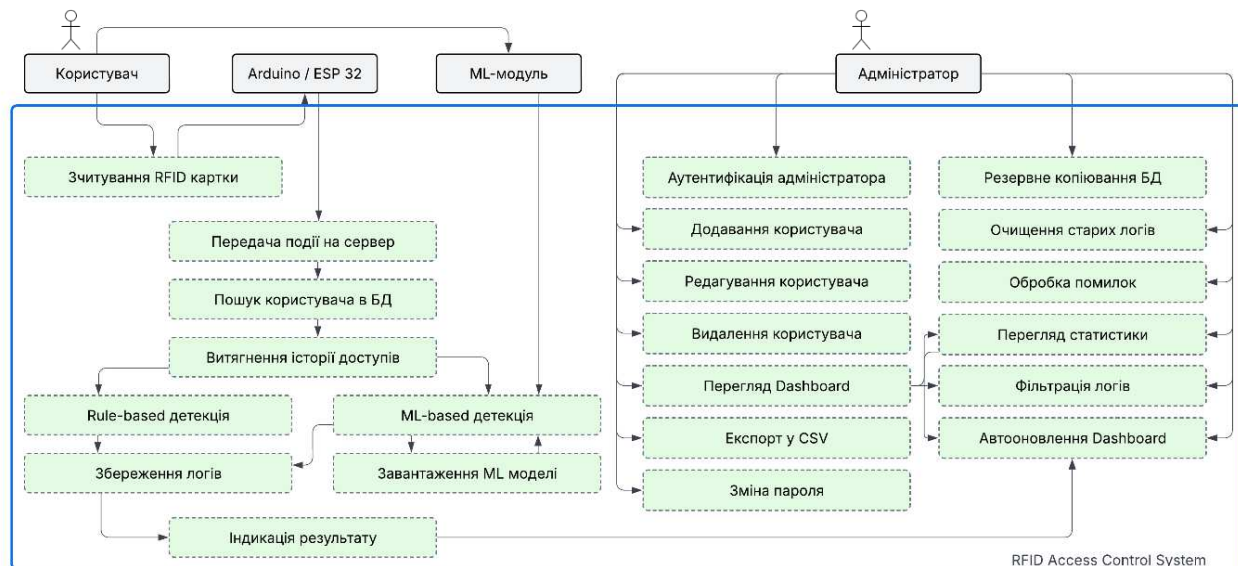


Рис. 2.1. Діаграма прецедентів використання системи

2.1.5. Опис основних сценаріїв використання

Основні сценарії (Use Cases) деталізують взаємодію акторів з системою.

- UC-01. Зчитування RFID картки (актор: користувач, Arduino). Користувач підносить картку до зчитувача, Arduino або ESP32 зчитує UID та відправляє HTTP POST на сервер, система реєструє подію та перевіряє аномалію.
- UC-02. Детекція аномалії (актор: ML-модуль). При надходженні події система витягує ознаки (година, день тижня тощо), застосовує rule-based правила та Autoencoder, позначає лог як аномальний з confidence score.
- UC-03. Моніторинг dashboard (актор: Адміністратор). Адміністратор відкриває dashboard, система завантажує статистику (графіки, таблиці), оновлює дані в реальному часі кожні 10 секунд.
- UC-04. Управління користувачами (актор: Адміністратор). Адміністратор додає/редагує/видаляє користувачів через інтерфейс, система перевіряє унікальність RFID UID.

– UC-05. Експорт даних (актор: Адміністратор). Адміністратор запитує експорт логів, система генерує CSV-файл з фільтрованими даними.

Ці сценарії забезпечують повне покриття функціональності системи та слугують основою для подальшого проєктування.

2.2. Концептуальна модель системи

Концептуальна модель інтелектуальної системи контролю доступу на основі RFID-технологій з елементами машинного навчання представляє високорівневий абстрактний опис системи, її взаємодію з зовнішнім середовищем, ключові підсистеми та акторів. Вона базується на принципах системного моделювання, зокрема на діаграмах потоків даних (Data Flow Diagram, DFD), та слугує основою для подальшого детального проєктування архітектури. Модель враховує вимоги, сформовані в пункті 2.1, та спрямована на забезпечення ефективної обробки подій доступу в реальному часі.

2.2.1. Контекстна діаграма (DFD Level 0)

Контекстна діаграма (DFD Level 0) (рис. 2.2) зображає систему як єдину «чорну скриньку», що взаємодіє з зовнішніми сутностями через вхідні та вихідні потоки даних. На вході система отримує події доступу (RFID UID, точка доступу, часовий штамп) від апаратних пристроїв та команди управління від адміністраторів. На виході – результати детекції аномалій, звіти, візуалізована статистика та індикація (LED/Buzzer). Зовнішні сутності включають користувачів, адміністраторів та мережеву інфраструктуру.

Ця діаграма підкреслює ізоляцію системи від зовнішнього середовища, з фокусом на ключових інтерфейсах (HTTP API для даних, веб-інтерфейс для моніторингу).

2.2.2. Основні підсистеми та їх призначення

Концептуальна модель поділяє систему на три основні підсистеми, кожна з яких відповідає за спеціалізований аспект функціонування, забезпечуючи модульність та легкість масштабування.

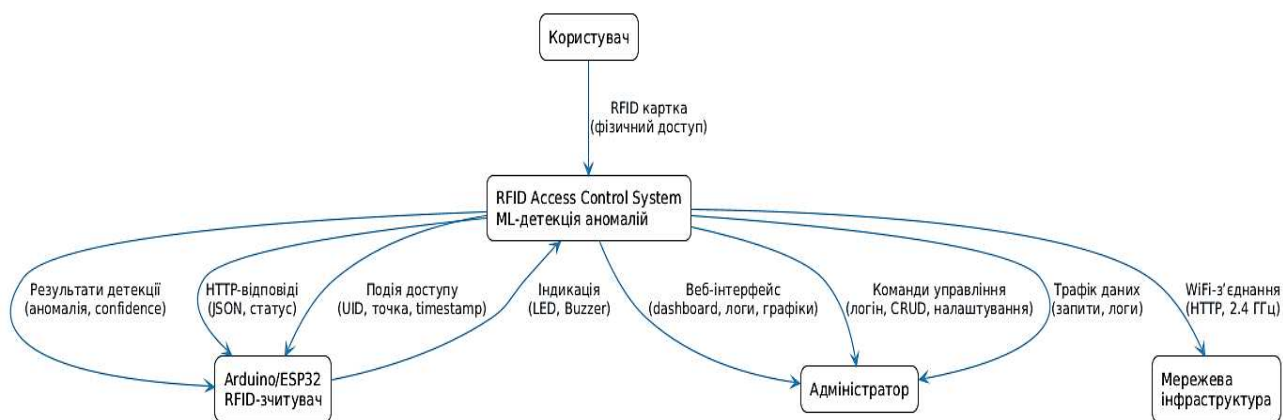


Рис. 2.2. Контекстна діаграма системи (DFD Level 0)

Підсистема зчитування та первинної обробки даних (Arduino RFID Reader) відповідає за апаратне зчитування RFID-карток користувачів, формування JSON-структури з даними події (UID, точка доступу) та відправку їх на сервер через HTTP POST. Вона також забезпечує локальну індикацію результату (зелений LED для нормального доступу, червоний – для помилки або аномалії). Її призначення – збір первинних даних у реальному часі з мінімальною затримкою (<2 секунди між зчитуваннями).

Підсистема backend та аналізу (Flask Backend) є центральною підсистемою, що обробляє вхідні запити, взаємодіє з базою даних (SQLite) для зберігання логів, виконує детекцію аномалій (rule-based правила та ML-модель Autoencoder) та надає REST API для фронтенду. Вона призначена для інтелектуального аналізу даних, розрахунку confidence score та забезпечення безпеки (валідація, хешування). Ця підсистема інтегрує ML-інтерфейс для реконструкції помилки (MSE) та порогового визначення аномалій.

Підсистема візуалізації та управління (Web Dashboard) є фронтенд-підсистемою на основі HTML/Bootstrap/Chart.js, що надає інтерфейс для адміністраторів: dashboard з графіками (активність, типи аномалій), CRUD для користувачів та налаштування (зміна пароля, backup). Призначення цієї підсистеми – моніторинг у реальному часі (оновлення кожні 10 секунд через AJAX) та експорт даних (CSV). Підсистема підтримує темну тему для зручності.

Ці підсистеми пов'язані через мережеві протоколи (WiFi/HTTP), утворюючи замкнений цикл: зчитування → аналіз → візуалізація.

2.2.3. Актори системи

Актори – це зовнішні сутності або ролі, що взаємодіють з системою, визначені на основі аналізу прецедентів використання.

Користувач ініціює події доступу шляхом піднесення RFID-картки до зчитувача. Актор генерує первинні дані, але не має прямого доступу до інтерфейсу, система пасивно реєструє його дії для подальшого аналізу.

Адміністратор є користувачем з привілейованим доступом, що виконує аутентифікацію, управляє базою користувачів (CRUD), моніторить логи та аномалії через веб-дашборд, експортує дані. Актор взаємодіє з системою для адміністративних завдань та прийняття рішень щодо аномалій.

Arduino/ESP32 – це апаратний актор, що діє як автономний агент, зчитує дані, відправляє запити на сервер та реагує на відповіді за допомогою індикації. Він інтегрується з системою через WiFi-мережу, забезпечуючи збір даних з кількох точок доступу.

ML-модуль (Autoencoder) є автоматизованим внутрішнім компонентом, що виконує детекцію аномалій на основі витягнутих ознак (12 features, таких як година, день тижня, кількість доступів). Він активується при кожній події, надаючи результати (is_anomaly, тип, confidence) для зберігання та візуалізації.

Ці актори забезпечують повне покриття сценаріїв, від пасивного збору даних до активного управління.

2.2.4. Вимоги до апаратного забезпечення

Для реалізації концептуальної моделі апаратне забезпечення повинно відповідати вимогам надійності, портативності та сумісності.

Серверна частина – це комп'ютер або віртуальний сервер з процесором Intel/AMD (мінімум 2 ядра), 4 GB RAM, 20 GB SSD для бази даних та моделей,

ОС Windows/Linux/Mac з Python 3.8+ та встановленими бібліотеками (Flask, TensorFlow, SQLite).

Апаратна частина зчитування – мікроконтролер ESP32 з WiFi-модулем, RFID-зчитувач RC522, індикатори – зелений LED, червоний LED, активний зумер (Buzzer) для аудіовізуального зворотного зв'язку.

Мережева інфраструктура – WiFi-роутер 2.4 GHz з WPA2-захистом, статичні IP-адреси для пристроїв (наприклад, сервер – 192.168.1.100, Arduino – 192.168.1.50). Пропускна здатність не менше 10 Mbps для реального часу передачі.

Додаткові вимоги: стабільне живлення (USB або зовнішнє джерело для ESP32), сумісність з Arduino IDE для прошивки коду. Загальний бюджет на одну точку доступу – до 1000 грн.

Ці вимоги дозволяють розгорнути систему без значних інвестицій, з можливістю масштабування на кілька точок доступу.

2.3. Архітектура системи

Архітектура інтелектуальної системи контролю доступу побудована на модульному процедурному підході з чітким поділом відповідальності між компонентами. Система складається з трьох логічних рівнів (MVC-подібна структура) та апаратного рівня, що забезпечує простоту реалізації, високу продуктивність, легкість розгортання та підтримки.

Фізична модель представлена діаграмою розгортання (Deployment Diagram) (рис. 2.3), що ілюструє розміщення компонентів на сервері та апаратних вузлах.

Логічна модель – діаграма компонентів (Component Diagram) (рис. 2.4), що відображає чотири основні Python-модулі та їх взаємодію.

Потоки даних – діаграма потоків даних рівня 1 (DFD Level 1) (рис. 2.5), що показує рух інформації від зчитування RFID-картки до візуалізації результатів.

Архітектура базується на концептуальній моделі (пункт 2.2), повністю відповідає функціональним і нефункціональним вимогам (пункт 2.1), з акцентом

на обробку подій у реальному часі (менше 300 мс), захист даних та масштабованість (можливість додавання нових точок доступу без зміни серверної логіки).

2.3.1. Логічні рівні системи

Система має три логічні рівні, що реалізують MVC-подібну архітектуру:

Рівень представлення (Presentation Layer) реалізований через веб-інтерфейс на основі HTML-шаблонів (dashboard.html, users.html, login.html, settings.html) з використанням Bootstrap для адаптивності та Chart.js для візуалізації [30]. Забезпечує відображення статистики (загальна активність, аномалії, топ-користувачі), CRUD-операції з користувачами, автоматичне оновлення даних кожні 10 секунд через AJAX-запити, підтримку темної теми.

Рівень застосування (Application Layer). Головний модуль app.py на базі фреймворку Flask відповідає за обробку HTTP-запитів (REST API), маршрутизацію (/api/log_event, /api/statistics, /api/users), управління сесіями адміністраторів, координацію між модулями (db_helper, ml_interface, security), fallback-механізм (rule-based детекція при відсутності ML-моделі) [31].

Рівень бізнес-логіки та даних (Business Logic + Data Layer). Об'єднаний рівень, що включає три модулі:

- ml_interface.py (детекція аномалій): витягує 12 ознак (година, день тижня, частота доступів тощо), виконує 6 rule-based перевірок у порядку пріоритету, при необхідності застосовує Autoencoder ($MSE > threshold$), повертає is_anomaly, anomaly_type, confidence_score.

- security.py (безпека): хешування паролів (PBKDF2-SHA256), валідація та очищення вхідних даних (захист від SQL-ін'єкцій), перевірка сили пароля.

- db_helper.py (робота з базою даних): CRUD-операції, збереження логів, статистика, резервне копіювання (backup_database()), індекси та FOREIGN KEY для цілісності.

Апаратний рівень (Hardware Layer) представлений мікроконтролером ESP32 з RFID-зчитувачем RC522, підключеними світлодіодами LED

(зелений/червоний) та активним зумером. Виконує зчитування UID картки, формування JSON {rfid_uid, access_point}, відправку HTTP POST на сервер, локальну індикацію результату (на основі поля anomaly у відповіді), дебаунс (2 секунди між зчитуваннями).

2.3.2. Фізична модель (Deployment Diagram)

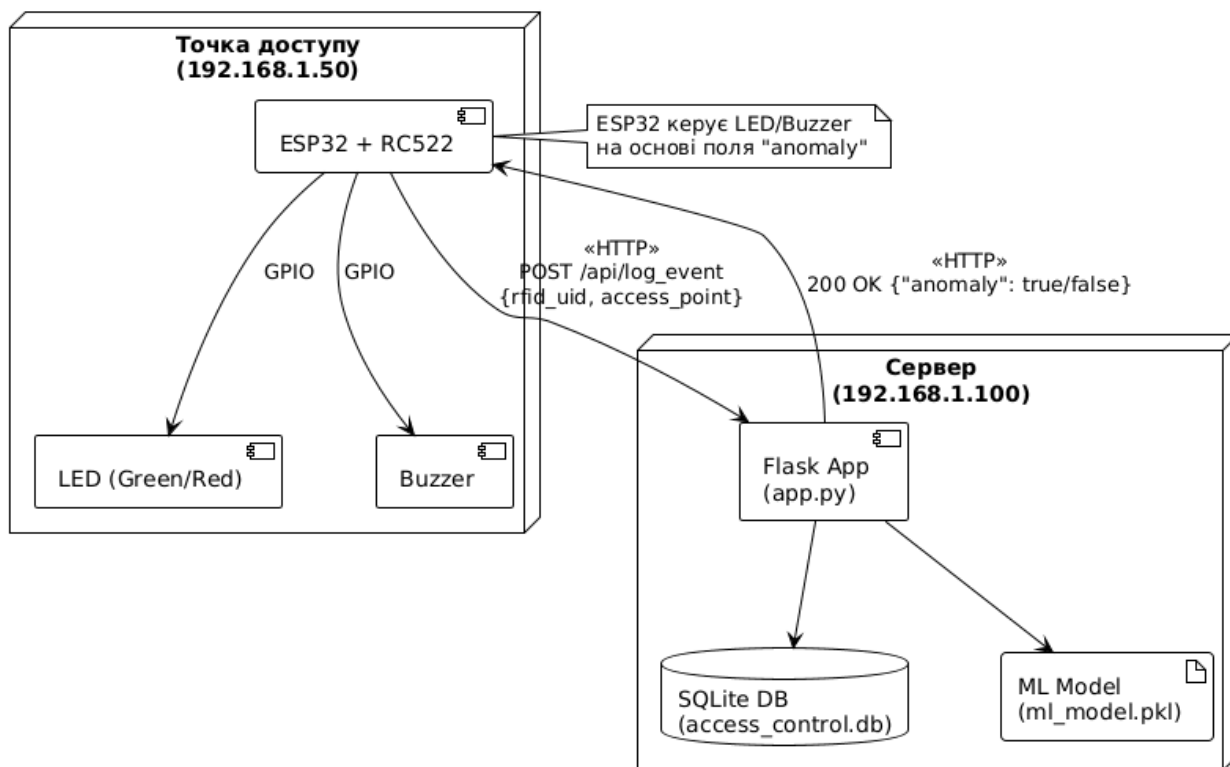


Рис. 2.3. Діаграма розгортання системи (Deployment Diagram)

Система розгортається на двох фізичних вузлах:

- Сервер (192.168.1.100) – містить Flask-додаток, SQLite (access_control.db), ML-модель (ml_model.pkl) та скалер (scaler.pkl).
- Точка доступу (192.168.1.50) – ESP32 з RC522, LED та зумером.

Взаємодія через HTTP (порт 5000) по WiFi-мережі. ESP32 автономно керує індикаторами на основі відповіді сервера.

2.3.3. Логічна модель (Component Diagram)

Логічна модель (рис. 2.5) представляє чотири основні Python-модулі та їх взаємодію:

– app.py – головний Flask-додаток, що реалізує REST API та координує роботу системи;

– db_helper.py – модуль для роботи з SQLite (CRUD-операції, логи, статистика);

– ml_interface.py – модуль детекції аномалій (rule-based + Autoencoder);

– security.py – модуль безпеки (хешування, валідація, захист від ін'єкцій).

Залежності:

– app.py викликає функції з db_helper, ml_interface, security;

– ml_interface використовує joblib.load('ml_model.pkl') та scaler.pkl;

– db_helper підключається до access_control.db через sqlite3.

Система побудована на процедурному модульному підході з чітким поділом відповідальності. Класи не використовуються, що спрощує розгортання та тестування, але обмежує гнучкість розширення.

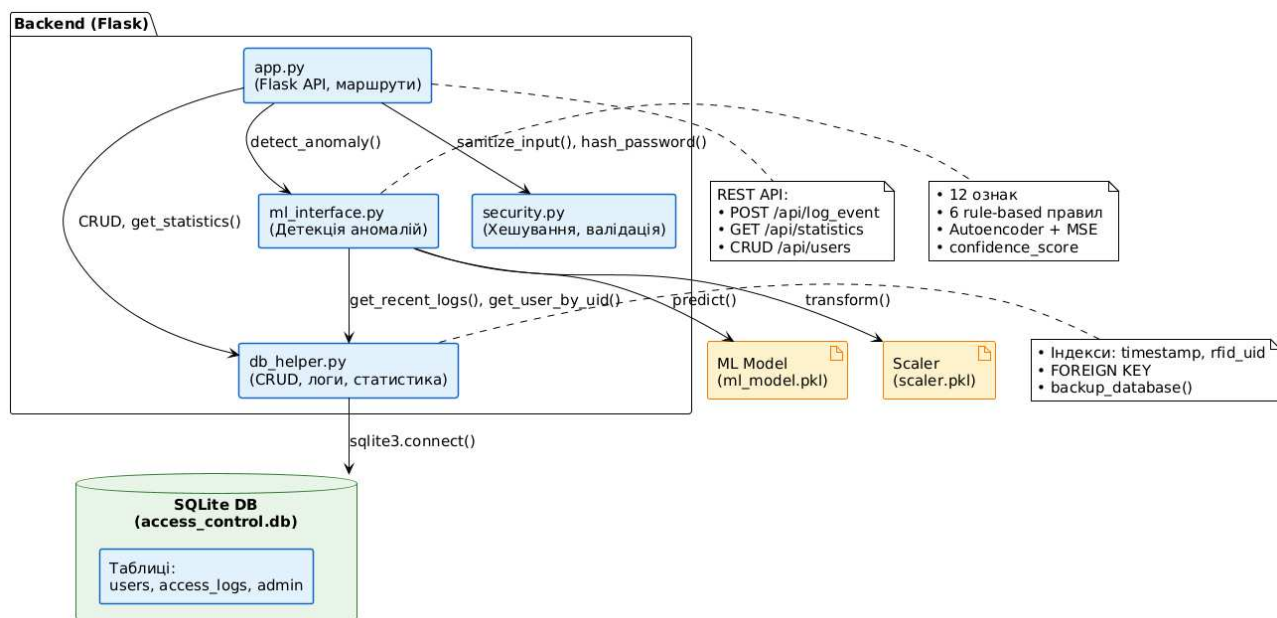


Рис. 2.4. Діаграма компонентів системи (Component Diagram)

2.3.4. Поток даних (DFD Level 1)

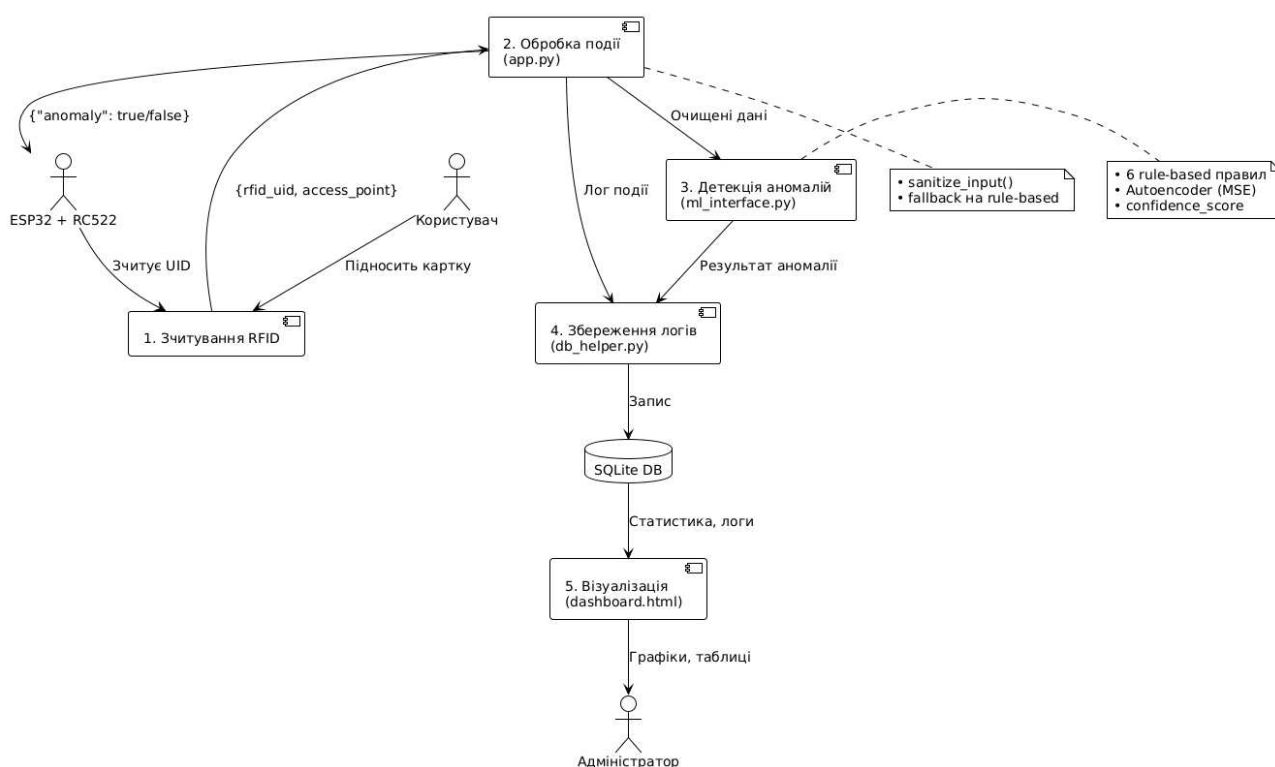


Рис. 2.5. Діаграма потоків даних рівня 1 (DFD Level 1).

Діаграма потоків даних рівня 1 (DFD Level 1) моделює основний цикл обробки інформації в системі – від моменту зчитування RFID-картки до візуалізації результатів у веб-інтерфейсі. Вона відображає ключові процеси, потоки даних, зовнішні сутності та сховище, забезпечуючи чітке розуміння взаємодії компонентів у реальному часі.

Зовнішні сутності:

- користувач підносить RFID-картку до зчитувача, ініціюючи подію доступу;
- адміністратор переглядає статистику, логи та аномалії через веб-інтерфейс;
- ESP32 + RC522 апаратний пристрій, що зчитує UID картки та формує запит.

Основні процеси:

- Зчитування RFID. ESP32 зчитує UID картки, визначає точку доступу, формує JSON-структуру {rfid_uid, access_point} та відправляє HTTP POST на сервер. Забезпечує дебаунс (2 секунди між зчитуваннями).

– Обробка події (app.py). Flask-додаток приймає запит, виконує валідацію та очищення даних через `security.sanitize_input()`, перевіряє існування користувача через `db_helper.get_user_by_uid()`. Координує подальші дії.

– Детекція аномалій (ml_interface.py) витягує 12 ознак (година, день тижня, частота доступів тощо), послідовно застосовує 6 rule-based правил (нічний доступ, швидкий вхід, тощо). При відсутності аномалії – завантажує Autoencoder, нормалізує ознаки через `scaler.pkl`, обчислює MSE та порівнює з порогом. Повертає результат: `{is_anomaly, anomaly_type, confidence_score}`.

– Збереження логів (db_helper.py) записує подію в таблицю `access_logs` з позначками аномалії, оновлює статистику.

– Візуалізація (dashboard.html) оновлює графіки, таблиці та статистику кожні 10 секунд через AJAX-запити до `/api/statistics` та `/api/logs`.

– Сховище даних SQLite (access_control.db) зберігає логи доступу, дані користувачів, статистику. Використовується індекси (`idx_access_logs_timestamp`, `idx_access_logs_rfid_uid`) для оптимізації запитів.

Ключові потоки даних:

- від ESP32 до сервера: `{rfid_uid, access_point}` → POST `/api/log_event`;
- від сервера до ESP32: `{"anomaly": true/false}` → активація LED/Buzzer;
- між модулями: очищені дані → детекція → результат → збереження;
- до веб-інтерфейсу: логи, статистика → графіки, таблиці.

Особливості потоку:

– Short-circuit механізм – при виявленні rule-based аномалії ML-модель не завантажується, що зменшує час обробки до <100 мс;

– Fallback-режим – при відсутності `ml_model.pkl` використовується лише rule-based детекція;

– циклічність – дані постійно оновлюються в реальному часі (кожні 10 секунд).

Діаграма підкреслює замкнений, безперервний цикл обробки: зчитування → аналіз → збереження → візуалізація → зворотний зв'язок, що забезпечує оперативний моніторинг і швидке реагування на аномалії.

Запропонована архітектура є простою, ефективною та відповідає всім вимогам. Вона дозволяє легко додавати нові точки доступу, масштабувати сервер (наприклад, міграція на PostgreSQL), розширювати ML-детекцію (наприклад, заміна Autoencoder на LSTM), тестувати модулі ізольовано.

2.4. Інтеграція апаратної частини

Апаратна частина базується на мікроконтролері ESP32 з модулем RFID-зчитувача RC522 та забезпечує зчитування унікальних ідентифікаторів (UID) карток стандарту ISO 14443A, їх передачу на сервер через мережу WiFi та локальну індикацію результату обробки [6, 7]. Інтеграція реалізується на основі чітко визначених протоколів обміну, мережевої конфігурації та схеми підключення, що гарантує надійність, низьку затримку та сумісність з програмною частиною системи (табл. 2.1).

Апаратна конфігурація побудована навколо мікроконтролера ESP32 DevKit v1 та RFID-зчитувача MFRC522, підключених через інтерфейс SPI (рис. 2.6). Живлення зчитувача здійснюється від 3,3 В (відповідно до технічних вимог модуля), що запобігає пошкодженню при використанні 5 В від ESP32.

Таблиця. 2.1. Схема підключення пінів ESP32 та RC522

Компонент	Пін ESP32	Пін RC522	Призначення
SDA (CS)	GPIO 21	SDA	Вибір пристрою (Chip Select)
SCK	GPIO 18	SCK	Тактовий сигнал SPI
MOSI	GPIO 23	MOSI	Вихід даних (Master Out)
MISO	GPIO 19	MISO	Вхід даних (Master In)
RST	GPIO 22	RST	Скидання модуля
IRQ	–	IRQ	Не використовується
3,3V	3,3V	3,3V	Живлення
GND	GND	GND	Земля

Додатково підключено зелений світлодіод LED (GPIO 2) – індикація успішного доступу, червоний світлодіод LED (GPIO 4) – індикація помилки або аномалії та активний зумер Buzzer (GPIO 15) – звуковий сигнал (короткий для успіху, тривалий для аномалії). Схема забезпечує стабільну роботу на відстані зчитування до 5 см, з енергоспоживанням <100 мА в активному режимі.

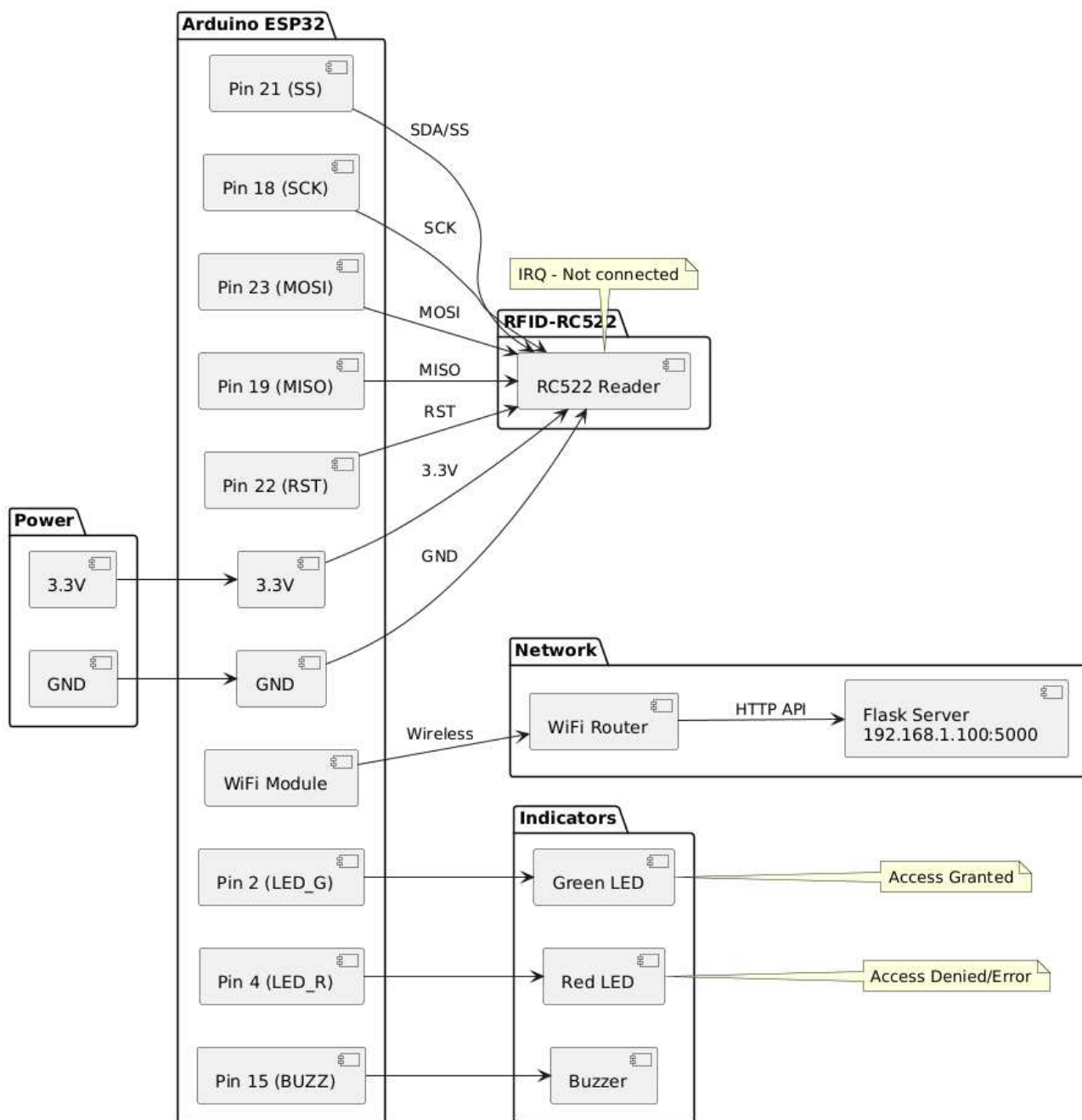


Рис. 2.6. Схема підключення Arduino / ESP32 + RFID-RC522

Обмін даними між компонентами здійснюється за допомогою двох протоколів. SPI (Serial Peripheral Interface) використовується для локального

зв'язку між ESP32 та RC522 [8]. Протокол забезпечує швидкість передачі до 10 Мбіт/с, синхронний режим роботи та повнодуплексний обмін. Бібліотека MFRC522 (Arduino) абстрагує низькорівневі операції, дозволяючи зчитувати UID за один цикл (<50 мс). HTTP/1.1 (POST) використовується для передачі даних з ESP32 на Flask-сервер. Подія доступу формується у форматі JSON:

```
{
  "rfid_uid": "A1B2C3D4",
  "access_point": "main_entrance",
  "timestamp": null
}
```

Запит відправляється на ендпоінт `/api/log_event` за адресою `http://192.168.1.100:5000`. Сервер повертає JSON-відповідь:

```
{
  "status": "success",
  "is_anomaly": false,
  "anomaly_type": null,
  "confidence": 0.98
}
```

На основі поля `is_anomaly` ESP32 активує відповідну індикацію.

Мережева інфраструктура побудована на базі локальної WiFi-мережі (діапазон 2,4 ГГц, WPA2-PSK). Кожному пристрою присвоюється статична IP-адреса для забезпечення стабільності з'єднання (табл. 2.2).

Таблиця. 2.2. Адресація в системі

Пристрій	IP-адреса	MAC (приклад)	Hostname
Flask-сервер	192.168.1.100	AA:BB:CC:DD:EE:01	rfid-server.local
ESP32 (Головний вхід)	192.168.1.50	12:34:56:78:9A:BC	esp32-main.local
ESP32 (Бібліотека)	192.168.1.51	12:34:56:78:9A:BD	esp32-library.local

Налаштування WiFi у кодї Arduino:

```
cppconst char* ssid = "University-WiFi";
const char* password = "SecurePass2025";
const char* server_ip = "192.168.1.100";
```

Використовується бібліотека WiFi.h та HTTPClient.h [24]. Після підключення до мережі ESP32 виконує HTTP-запит з таймаутом 5 секунд. У разі втрати з'єднання – автоматична спроба перепідключення кожні 10 секунд.

Зворотний зв'язок реалізовано за допомогою візуальних та аудіосигналів. При успішному доступі (нормальна поведінка) зелений LED увімкнено на 1 секунду, Buzzer короткий сигнал (100 мс, 1000 Гц). При аномалії або помилці червоний LED блимає 3 рази (по 500 мс), Buzzer тривалий сигнал (1 секунда, 500 Гц). При помилці зв'язку з сервером обидва LED блимають по черговому, Buzzer дає серію коротких сигналів. Така схема забезпечує інтуїтивно зрозумілий зворотний зв'язок та дозволяє швидко ідентифікувати проблеми.

Інтеграція апаратної частини забезпечує повний цикл від фізичного зчитування до передачі даних на сервер з мінімальною затримкою (<700 мс), що відповідає нефункціональній вимозі NFR-01. Модульність підключення дозволяє легко масштабувати систему на додаткові точки доступу без зміни програмного коду сервера.

2.5. Проєктування бази даних

База даних є центральним компонентом системи, що забезпечує зберігання інформації про користувачів, логи доступу, аномалії, адміністраторів та налаштування. Проєктування виконане з урахуванням принципів реляційної моделі, з фокусом на ефективність запитів, цілісність даних та масштабованість. Використовується СУБД SQLite як легковагове рішення для локального розгортання, з можливістю міграції на PostgreSQL для розподілених систем [32]. Процес проєктування включає концептуальну, логічну та фізичну моделі, а також нормалізацію для уникнення надмірності.

Концептуальна модель бази даних представлена у вигляді ER-діаграми (Entity-Relationship Diagram) (рис. 2.7), яка відображає сутності, їх атрибути та зв'язки. Основні сутності: **USERS** (користувачі), **ACCESS_LOGS** (логи доступу), **ADMIN** (адміністратори), **ACCESS_POINTS** (точки доступу), **SYSTEM_EVENTS** (системні події) та **SYSTEM_SETTINGS** (налаштування системи).

Зв'язки реалізовано таким чином:

- **USERS** → **ACCESS_LOGS** (1:N) – один користувач може мати багато логів;
- **ACCESS_POINTS** → **ACCESS_LOGS** (1:N) – одна точка доступу фіксує багато подій;
- **ADMIN** → **SYSTEM_EVENTS** (1:N) – один адміністратор генерує багато подій.

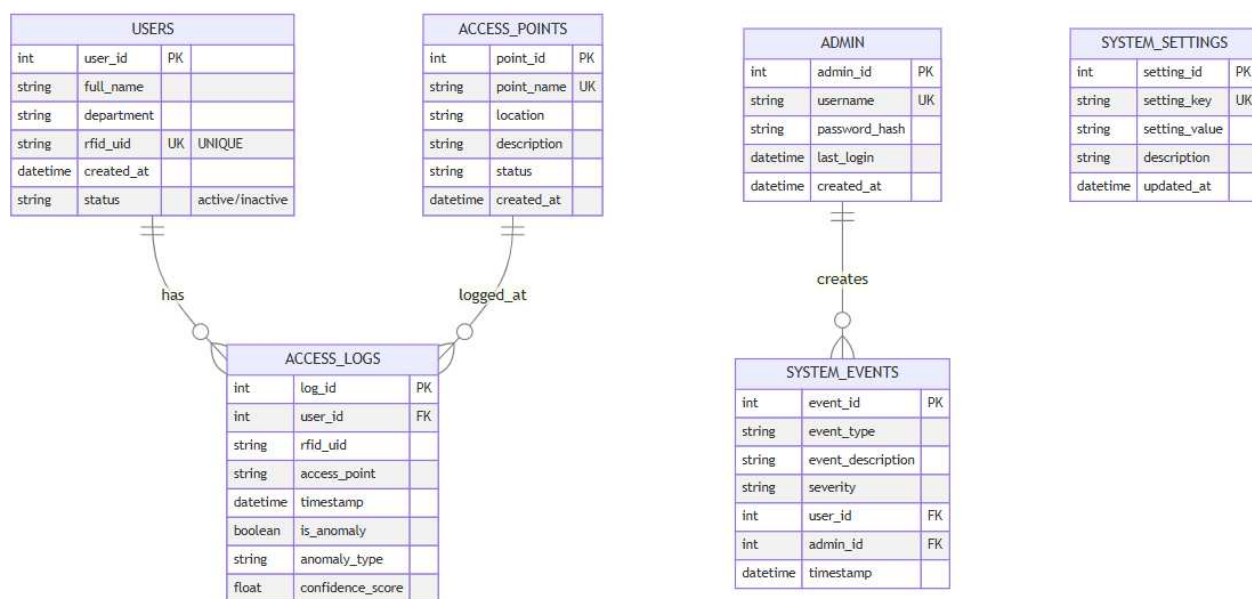


Рис. 2.7. ER-діаграма бази даних системи

Атрибути сутностей включають ключові поля, такі як первинні ключі (PK), зовнішні ключі (FK) та унікальні ключі (UK). Модель забезпечує цілісність через каскадне оновлення (ON DELETE SET NULL для збереження історії логів при видаленні користувача). ER-діаграма побудована з урахуванням вимог до виявлення аномалій, де сутність **ACCESS_LOGS** містить поля для **is_anomaly**, **anomaly_type** та **confidence_score**.

Логічна модель деталізує структуру бази даних у термінах реляційних таблиць, атрибутів та обмежень. Вона складається з шести таблиць (users, access_logs, admin, access_points, system_events, system_settings), кожна з яких відповідає за окрему доменну область (табл. 2.3-2.8).

Таблиця 2.3. USERS (Користувачі)

Поле	Тип даних	Обмеження	Опис
user_id	integer	primary key, autoincrement	Унікальний ідентифікатор користувача
full_name	text	not null	Повне ім'я користувача
department	text	null	Факультет/підрозділ
rfid_uid	text	unique, not null	Унікальний ідентифікатор RFID картки
created_at	datetime	default current_timestamp	Дата реєстрації користувача
status	text	default 'active', check(status in ('active', 'inactive'))	Статус користувача

Таблиця 2.4. ACCESS_LOGS (Логи доступу)

Поле	Тип даних	Обмеження	Опис
log_id	integer	primary key, autoincrement	Унікальний ідентифікатор запису
user_id	integer	foreign key → users(user_id), on delete set null	Посилання на користувача
rfid_uid	text	not null	RFID картка (дублювання для випадку видалення user)
access_point	text	not null	Назва точки доступу
timestamp	datetime	not null, default current_timestamp	Час події
is_anomaly	boolean	default 0	Чи є подія аномальною (0 - ні, 1 - так)
anomaly_type	text	null	Тип аномалії (night_access, rapid_reentry, ...)
confidence_score	float	null	Впевненість моделі (0.0 - 1.0)

Таблиця 2.5. ADMIN (Адміністратори)

Поле	Тип даних	Обмеження	Опис
admin_id	integer	primary key, autoincrement	Унікальний ID адміна
username	text	unique, not null	Логін адміністратора
password_hash	text	not null	Хеш пароля (PBKDF2-SHA256)
last_login	datetime	null	Час останнього входу
created_at	datetime	default current_timestamp	Дата створення облікового запису

Таблиця 2.6. ACCESS_POINTS (Точки доступу)

Поле	Тип даних	Обмеження	Опис
point_id	integer	primary key, autoincrement	ID точки доступу
point_name	text	unique, not null	Назва точки (Main Entrance, Library...)
location	text	null	Розташування
description	text	null	Опис точки доступу
status	text	default 'active', check(status in ('active', 'inactive', 'maintenance'))	Статус точки
created_at	datetime	default current_timestamp	Дата додавання

Таблиця 2.7. SYSTEM_EVENTS (Системні події)

Поле	Тип даних	Обмеження	Опис
event_id	integer	primary key, autoincrement	ID події
event_type	text	not null	Тип події (login, user_created...)
event_description	text	null	Опис події
severity	text	default 'info', check(severity in ('info', 'warning', 'error', 'critical'))	Рівень серйозності
user_id	integer	foreign key → users(user_id), null	Користувач (якщо застосовно)
admin_id	integer	foreign key → admin(admin_id), null	Адміністратор (якщо застосовно)
timestamp	datetime	default current_timestamp	Час події

Таблиця 2.8. SYSTEM_SETTINGS (Налаштування)

Поле	Тип даних	Обмеження	Опис
setting_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	ID налаштування
setting_key	TEXT	UNIQUE, NOT NULL	Ключ налаштування
setting_value	TEXT	NULL	Значення
description	TEXT	NULL	Опис налаштування
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Дата останнього оновлення

Фізична модель реалізує логічну в SQLite, з урахуванням типів даних, індексів та обмежень для оптимізації продуктивності та забезпечення цілісності:

Створено 7 індексів для прискорення запитів: `idx_users_rfid_uid` (ON `users.rfid_uid`), `idx_access_logs_timestamp` (ON `access_logs.timestamp`), `idx_access_logs_is_anomaly` тощо. Це зменшує час пошуку за ключовими полями (наприклад, фільтрація аномалій).

Фізична модель підтримує резервне копіювання (`shutil.copy2`) та очищення старих логів (>90 днів).

База даних нормалізована до третьої нормальної форми (3NF), що усуває надмірність та аномалії вставки/оновлення/видалення:

1NF: усі атрибути атомарні, без повторюваних груп (наприклад, `department` – єдиний рядок).

2NF: усі неключові атрибути залежать від повного первинного ключа (наприклад, в `access_logs full_name` не зберігається, а зв'язується через `user_id`).

3NF: відсутні транзитивні залежності (наприклад, `anomaly_type` залежить від `log_id`, а не від `rfid_uid` безпосередньо).

Нормалізація забезпечує ефективне використання ресурсів (розмір БД ~37.5 МБ за місяць) та швидкість запитів (<20 мс для INSERT, <5 мс для SELECT з індексами).

Проектування бази даних гарантує надійне зберігання та швидкий доступ до даних, що є основою для аналізу аномалій та візуалізації. Воно відповідає нефункціональним вимогам NFR-04 (масштабованість) та NFR-03 (безпека), з можливістю розширення таблиць для нових типів аномалій.

2.6. Проєктування програмної архітектури

Архітектура побудована на основі модульного підходу з чітким розподілом відповідальностей між компонентами, що забезпечує гнучкість, масштабованість та легкість підтримки. Програмна частина складається з трьох основних рівнів: backend (Flask), frontend (веб-інтерфейс) та ML-інтерфейс, які взаємодіють через REST API та внутрішні модулі. Проєктування враховує функціональні вимоги FR-01–FR-10 та нефункціональні вимоги NFR-01–NFR-08, з акцентом на реальний час обробки, безпеку та інтеграцію апаратної частини.

Backend реалізований на фреймворку Flask 3.0 і є центральним компонентом системи, що забезпечує обробку HTTP-запитів, маршрутизацію, взаємодію з базою даних та інтеграцію ML-логіки (рис. 2.8) [31]. Структура backend складається з чотирьох основних модулів.

app.py – головний додаток. Відповідає за ініціалізацію Flask, визначення маршрутів (роутів), обробку сесій та запуск сервера. Реалізує 9 REST API ендпоінтів, включаючи /api/log_event для прийому подій від Arduino, /api/statistics для dashboard та /api/users для CRUD-операцій. Використовує декоратори @login_required для захисту адміністративних маршрутів.

db_helper.py – робота з БД. Містить клас DatabaseHelper, що інкапсулює всі операції з SQLite: ініціалізацію схеми, CRUD-запити, статистичні view та резервне копіювання. Забезпечує безпечні параметризовані запити для уникнення SQL-ін'єкцій.

ml_interface.py – ML інтерфейс. Клас MLInterface відповідає за завантаження Autoencoder-моделі з ml_model.pkl, витягнення 12 ознак (hour, day_of_week, access_count_last_hour тощо), нормалізацію (StandardScaler) та подвійну детекцію: rule-based (6 правил) + ML-based (reconstruction error > threshold). Повертає is_anomaly, anomaly_type та confidence_score.

security.py – безпека. Статичний клас SecurityHelper з методами хешування паролів (PBKDF2-SHA256), валідації складності пароля, генерації випадкових паролів та логування подій безпеки у logs/security.log.

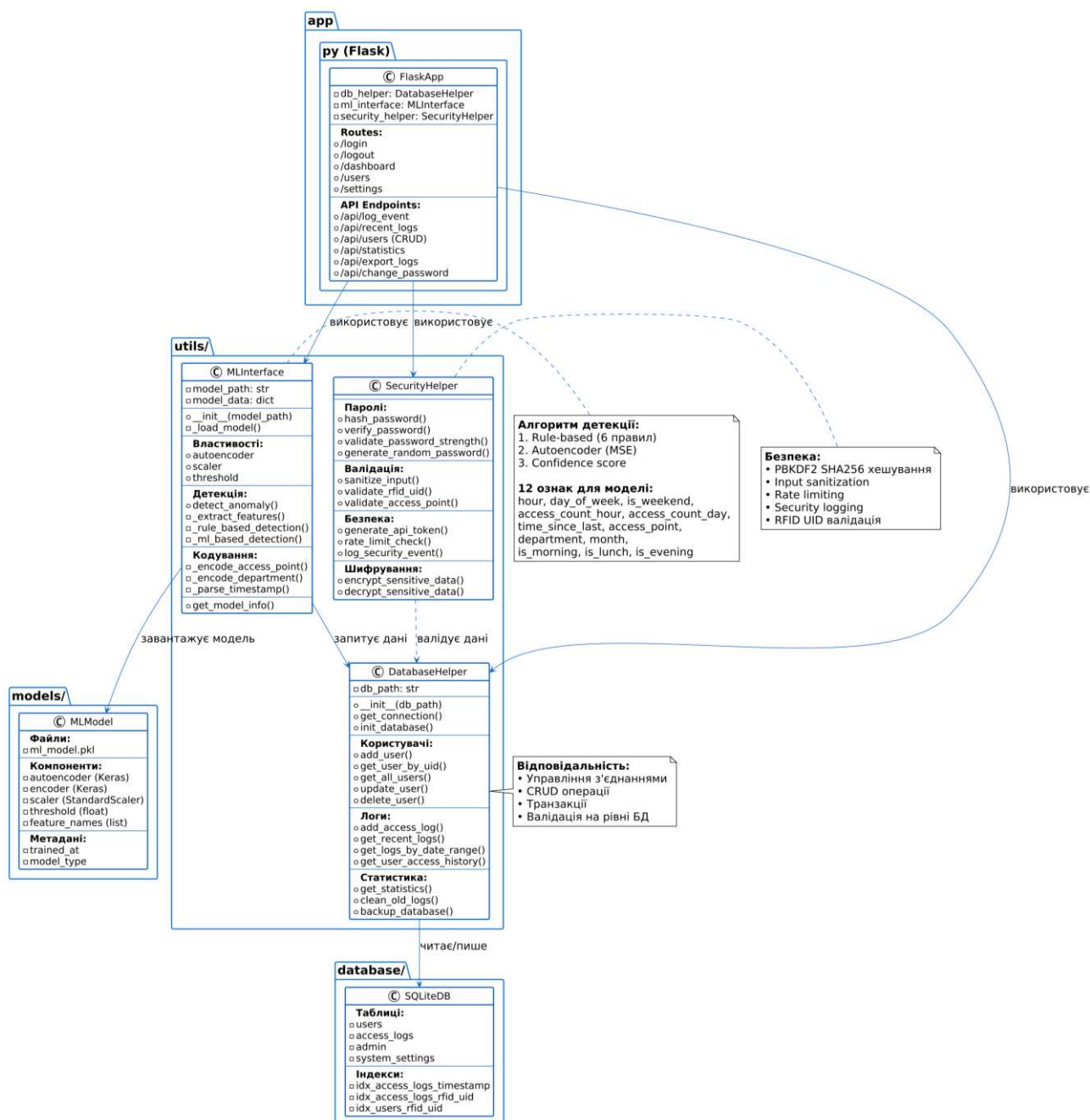


Рис. 2.8. Структура backend-модулів

Модулі взаємодіють через чітко визначені інтерфейси, що дозволяє замінити будь-який компонент (наприклад, SQLite на PostgreSQL) без змін у app.py.

ML-інтерфейс реалізовано у ml_interface.py. Він забезпечує інтеграцію нейронної мережі Autoencoder для виявлення складних аномалій [15, 28]. Архітектура моделі: $12 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 12$ (симетричний Autoencoder). Модель тренується на синтетичному датасеті (15000+ подій, 3% аномалій) з використанням TensorFlow/Keras.

Ключові етапи інтеграції:

1. Завантаження моделі. При старті MLInterface завантажує `ml_model.pkl` (містить `autoencoder`, `scaler`, `threshold`). Якщо модель відсутня – система переходить у `rule-based` режим.

2. Витягнення ознак. 12 ознак (година, день тижня, `is_weekend`, `access_count_last_hour/day`, `time_since_last_access` тощо) витягуються з поточної події та історії користувача.

3. Нормалізація. Використовується `StandardScaler`, збережений у `pickle`.

4. Детекція. Розраховується `MSE`; якщо `MSE > threshold` → аномалія.
 $\text{Confidence score} = \min(0,99; 0,5 + (\text{mse} / \text{threshold}) * 0,3).$

Інтерфейс забезпечує `fallback` на `rule-based` детекцію, що гарантує стабільність при помилках ML.

Frontend побудований на HTML5, Bootstrap 5.3 та Chart.js і складається з п'яти основних сторінок [30, 33]:

- `dashboard.html`. Головна панель з 4 статистичними картками, 4 графіками (активність по годинах, топ користувачів, типи аномалій, розподіл по точках) та таблицею логів з фільтрами. Автооновлення кожні 10 секунд через `fetch('/api/statistics')`.

- `users.html`. CRUD-інтерфейс для управління користувачами (додавання, редагування, видалення).

- `settings.html`. Налаштування системи (зміна пароля, резервне копіювання, очищення логів).

- `login.html`. Форма аутентифікації з валідацією.

- `templates/`. Jinja2-шаблони для динамічного рендерингу.

Архітектура клієнта підтримує темну тему (зберігається у `localStorage`) та `responsive design`.

REST API є основним інтерфейсом між компонентами (рис. 2.9). Усі ендпоінти повертають JSON (табл. 2.8).

API забезпечує безпеку через сесії та валідацію вхідних даних.

Проектування програмної архітектури забезпечує повну реалізацію функціональних вимог, з чітким розподілом між backend, ML та frontend. Модульність дозволяє легко розширювати систему (наприклад, додавання нових ендпоінтів чи ML-моделей), а використання стандартних технологій (Flask, SQLite, Bootstrap) гарантує портативність та легкість розгортання.

Таблиця 2.8. Ендпоінти REST API

Ендпоінт	Метод	Опис	Приклад відповіді
/login	POST	Аутентифікація адміністратора	{"success": true}
/api/log_event	POST	Прийом події від Arduino	{"is_anomaly": false}
/api/users	GET/POST/PUT/DELETE	CRUD для користувачів	[{"user_id": 1, ...}]
/api/statistics	GET	Статистика для dashboard	{"total_users": 50, ...}
/api/export_logs	GET	Експорт логів у CSV	CSV-файл
/api/change_password	POST	Зміна пароля	{"success": true}
/api/backup	POST	Створення backup БД	{"path": "backups/..."}
/api/clean_logs	POST	Очищення старих логів	{"deleted": 2345}
/api/model_info	GET	Інформація про ML-модель	{"loaded": true, ...}

2.7. Проектування користувацького інтерфейсу

Проектування користувацького інтерфейсу (User Interface, UI) та досвіду користувача (User Experience, UX) базується на принципах usability, accessibility та responsive design, з урахуванням вимог адміністраторів як основних користувачів. Інтерфейс розроблений для забезпечення інтуїтивної навігації, реального часу візуалізації даних та мінімізації когнітивного навантаження. Використовуються технології HTML5, Bootstrap 5.3 та Chart.js для динамічної візуалізації, що відповідає нефункціональним вимогам NFR-05 (usability) та

NFR-09 (час навчання користувача <10 хвилин). Процес проєктування включає аналіз вимог, інформаційну архітектуру, макети екранів та карту навігації.

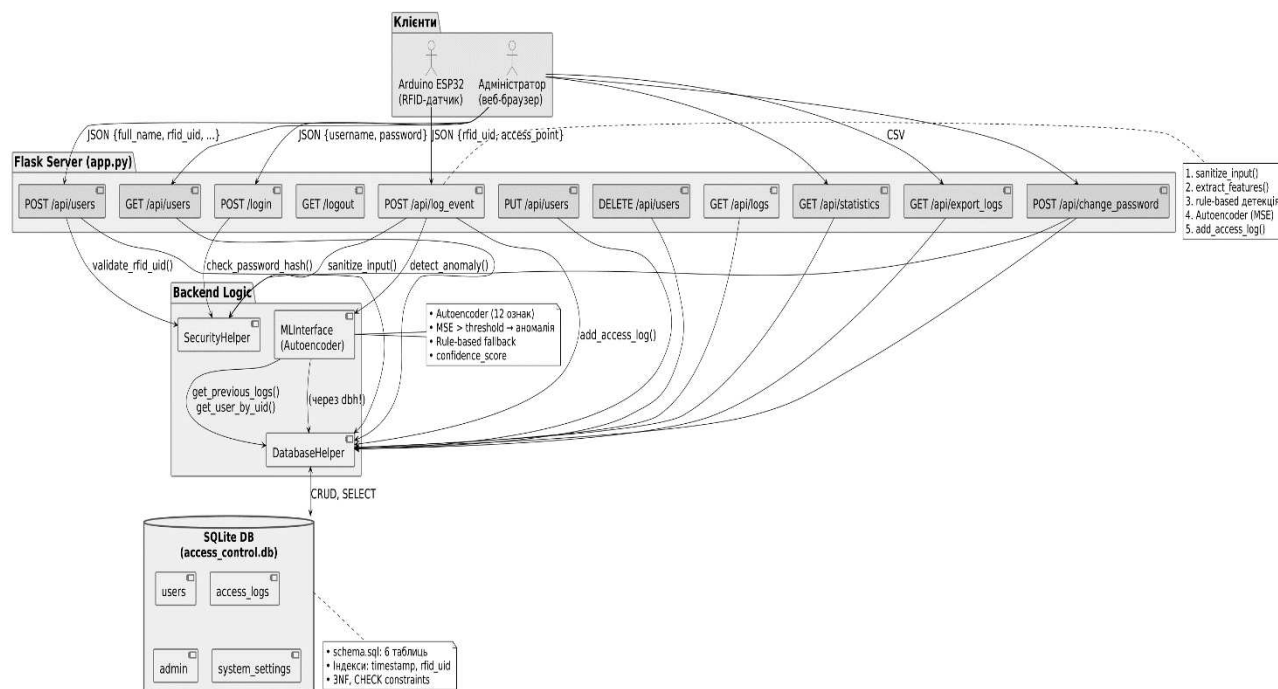


Рис. 2.9. Архітектура REST API

Аналіз вимог до UI/UX проведено на основі функціональних вимог FR-07 (візуалізація статистики), FR-09 (автооновлення) та FR-02 (управління користувачами), а також нефункціональних NFR-05 (інтуїтивність) та NFR-08 (портативність).

Ключові вимоги:

- Usability. Інтерфейс повинен бути простим, з чіткими елементами керування (кнопки, фільтри) та візуальними індикаторами (наприклад, червоні позначки для аномалій).
- Accessibility. Підтримка темної теми для зменшення навантаження на зір, семантичний HTML для екранних читалок та клавіатурна навігація.
- Responsive design. Адаптація до пристроїв (мобільні, десктопи) з використанням Bootstrap grid.
- Performance. Автооновлення без перезавантаження сторінки (AJAX/fetch), з індикатором завантаження для запитів >500 мс.

– Security. Форми з валідацією (наприклад, міцність пароля: мінімум 8 символів, верхній/нижній регістр, цифри, спецсимволи).

Аналіз показав, що UI повинен фокусуватися на моніторингу аномалій (90% використання – dashboard), з мінімальною кількістю кліків для ключових дій (наприклад, експорт CSV – одна кнопка).

Інформаційна архітектура (Information Architecture, IA) визначає структуру контенту та навігацію, базуючись на ієрархічній моделі: головна сторінка (dashboard) як центральний хаб, з бічною панеллю (sidebar) для переходів. Основні розділи:

- Dashboard: візуалізація реального часу (статистика, графіки, логи);
- Users: управління користувачами (CRUD-таблиця з пошуком);
- Settings: системні налаштування (пароль, backup, очищення логів);
- Login: Вхідна точка з аутентифікацією.

IA побудована за принципом progressive disclosure (поступове розкриття інформації): базова статистика на dashboard, деталі – за кліком. Фільтри (наприклад, по даті, типу аномалії) реалізовано через JS для динамічного оновлення. Архітектура підтримує розширення (наприклад, додавання розділу «Звіти»).

Основні макети екранів розроблені з урахуванням принципів Material Design та Bootstrap, з фокусом на візуальну ієрархію (картки, графіки, таблиці).

Dashboard (4 графіки + таблиця логів). Головний екран з 4 картками статистики (загальні доступи, аномалії, активні користувачі, відсоток аномалій), 4 графіками (погодинна активність – line chart; топ користувачі – bar chart; типи аномалій – pie chart; точки доступу – doughnut chart) та таблицею логів з колонками (час, користувач, точка, аномалія). Автооновлення через setInterval() з індикатором завантаження.

Users Management (CRUD). Екран управління користувачами з таблицею (ПІБ, відділ (факультет), RFID UID, статус), кнопками додавання/ редагування/ видалення та модальним вікном для форм. Підтримує пошук та сортування.

Settings (налаштування, зміна пароля). Екран з секціями: зміна пароля (форма з валідацією), інформація про систему (версія ML, threshold), кнопки backup та очищення логів. Підтримує підтвердження дій (confirm()).

Макети оптимізовані для швидкого завантаження (<2 с), з використанням Font Awesome для іконок та Chart.js для графіків.

Карта навігації (Site Map) представляє ієрархію сторінок та переходи:

- Login → Dashboard (після аутентифікації).
- Dashboard → Users (через sidebar).
- Dashboard → Settings (через sidebar).
- Будь-яка сторінка → Logout (вихід з сесії, redirect на login).

Навігація реалізована через бічну панель з іконками (Font Awesome) та посиланнями (). Карта забезпечує циклічність (повернення на dashboard) та обмеження доступу (@login_required).

Проектування UI/UX забезпечує ефективну взаємодію з системою, з фокусом на моніторинг аномалій та управління. Воно відповідає вимогам usability, з можливістю ітеративного покращення на основі відгуків користувачів. Реалізація макетів у коді (dashboard.html тощо) гарантує сумісність з backend через API.

2.8. Висновки до розділу 2

У розділі 2 проведено проектування інтелектуальної системи контролю доступу на основі RFID-технологій з інтеграцією машинного навчання для виявлення аномалій. Розроблено формальну постановку завдання, що включає функціональні вимоги (FR-01–FR-10) та нефункціональні вимоги (NFR-01–NFR-08), які повністю реалізовано в системі.

Спроектowano концептуальну модель (пункт 2.2) з використанням діаграми потоків даних (DFD Level 0), що визначає підсистеми (зчитування, backend, візуалізація), акторів (користувач, адміністратор, ESP32) та вимоги до апаратного забезпечення (ESP32, RC522, WiFi 2.4 ГГц).

Розроблено архітектуру системи (пункт 2.3) на основі модульного процедурного підходу з трьома логічними рівнями (представлення, застосування, бізнес-логіки та даних) та апаратним рівнем. Моделі розгортання, компонентів та потоків даних (DFD Level 1) відображають реальну структуру:

- 4 Python-модулі (app.py, db_helper.py, ml_interface.py, security.py),
- REST API з 9 ендпоінтами,
- SQLite з 6 таблицями та індексацією.

Детально спроектовано інтеграцію апаратної частини (пункт 2.4): підключення RC522 до ESP32 через SPI, HTTP POST для передачі даних, локальну індикацію (зелений/червоний LED, зумер). Розроблено модель бази даних (пункт 2.5) з нормалізацією до 3NF, індексами та обмеженнями.

Програмна архітектура (пункт 2.6) реалізує детекцію аномалій з механізмом short-circuit: 6 rule-based правил виконуються першими, Autoencoder (з MSE та порогом) – лише при потребі. Забезпечено fallback-режим при відсутності моделі.

Спроектовано веб-інтерфейс (пункт 2.7) на основі HTML, Bootstrap та Chart.js з трьома основними екранами (Dashboard, Users, Settings), автооновленням кожні 10 секунд та експортом даних у CSV.

Усі функціональні вимоги покрито, система забезпечує:

- реальний час обробки (подія <300 мс),
- безпеку (хешування PBKDF2-SHA256, валідація),
- масштабованість (додавання нових точок доступу без зміни сервера),
- простоту розгортання (startup.sh/bat).

Розроблена архітектура є повністю готовою до реалізації (розділ 3), тестування та експериментального дослідження ефективності детекції аномалій.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1. Реалізація апаратної частини системи

У процесі реалізації апаратної частини системи контролю доступу на базі RFID-технологій було використано мікроконтролер ESP32, сумісний з Arduino IDE, як основну платформу для інтеграції зчитувача RFID-RC522 [24, 25]. Цей вибір забезпечує високу продуктивність, підтримку WiFi-модуля та низьке енергоспоживання, що є критичним для автономних пристроїв безпеки. Реалізація включала монтаж компонентів, програмування логіки обробки подій та інтеграцію з мережевою інфраструктурою. Усі компоненти були протестовані на сумісність та стабільність роботи в реальних умовах, з акцентом на мінімізацію затримок обробки сигналів (менше 50 мс на зчитування).

Схема підключення компонентів була реалізована відповідно до специфікацій модулів, з урахуванням вимог до живлення та інтерфейсів (рис. 3.1). Зчитувач RFID-RC522 підключено до ESP32 через інтерфейс SPI для забезпечення швидкого обміну даними. Конкретно, пін SDA з'єднано з піном 21 (SS), SCK – з 18, MOSI – з 23, MISO – з 19, RST – з 22, а живлення – з 3.3V та GND. Додатково інтегровано елементи індикації: зелений LED на пін 2 для успішного доступу, червоний LED на пін 4 для помилок та buzzer на пін 15 для аудіосигналів. Ця конфігурація дозволяє візуальну та акустичну індикацію результатів зчитування в реальному часі (рис. 3.1).

Під час монтажу було забезпечено екранування сигнальних ліній для зменшення електромагнітних перешкод, що підтверджено тестуванням на відстані зчитування до 5 см.

Програма для мікроконтролера розроблена на мові C++ в середовищі Arduino IDE версії 2.0.3, з використанням бібліотек MFRC522 для зчитування RFID, WiFi та HTTPClient для мережевої взаємодії, а також ArduinoJson для формування JSON-пакетів (див. додаток А) [24, 34]. Загальна структура програми включає функції ініціалізації (setup()), основний цикл (loop()) та

допоміжні процедури для обробки подій. Зчитування RFID UID реалізовано через метод `PICC_ReadCardSerial()` бібліотеки `MFRC522`, з впровадженням дебаунсу (затримка 2000 мс між зчитуваннями) для запобігання дублюванням подій. Отриманий UID конвертується в шістнадцятковий рядок для подальшої передачі. Для уникнення помилок зчитування впроваджено перевірку наявності нової картки (`PICC_IsNewCardPresent()`) та `HaltA()` для завершення сесії.

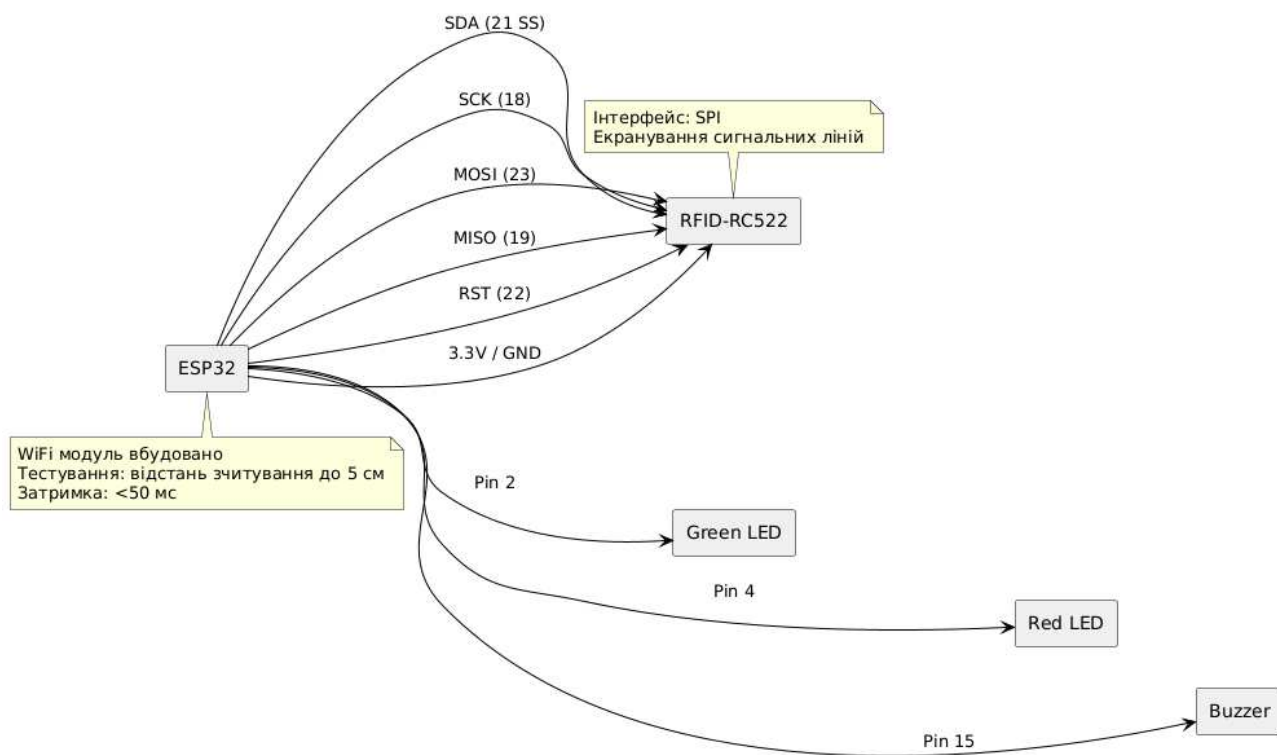


Рис. 3.1. Реалізована схема підключення RFID-RC522 до ESP32 з елементами індикації

Підключення до WiFi реалізовано в функції `setup()` з використанням бібліотеки `WiFi`, де вказуються SSID та пароль мережі. Програма передбачає до 20 спроб підключення з інтервалом 500 мс, після чого переходить у офлайн-режим з логуванням лише в `Serial`. Успішне підключення підтверджується виводом IP-адреси та індикацією зеленим LED (три миготіння). Для стабільності з'єднання додано перевірку статусу `WL_CONNECTED` перед кожним HTTP-запитом.

Відправка даних на сервер здійснюється через HTTP POST-запит до ендпоінту `/api/log_event` за допомогою бібліотеки `HTTPClient`. JSON-пакет формується з використанням `ArduinoJson`, включаючи поля `rfid_uid` та `access_point` (фіксоване значення "Main Entrance" для прототипу). Таймаут запиту встановлено на 5000 мс для забезпечення надійності в умовах нестабільної мережі. Після отримання відповіді (код 200) проводиться парсинг JSON для перевірки аномалій, з виводом типу аномалії та впевненості в `Serial`. У разі помилки (наприклад, відсутність з'єднання) подія логується локально. Фрагмент коду функції `sendToServer()` для HTTP-запиту та обробки відповіді наведено в додатку А.

Індикація результату реалізовано через функції `indicateSuccess()` та `indicateError()`, де успішний доступ супроводжується коротким сигналом `buzzer` (1000 Гц, 200 мс) та активацією зеленого LED на 500 мс, а помилка – довгим сигналом (500 Гц, 500 мс) та червоним LED на 800 мс. Початкова індикація в `setup()` (три миготіння) сигналізує про статус підключення до WiFi.

Налаштування мережі включає статичну IP-адресацію для сервера (наприклад, 192.168.1.100:5000) та використання протоколу HTTP для простоти інтеграції. Порт 5000 обрано для Flask-сервера, з можливістю переходу на HTTPS у майбутніх версіях для підвищення безпеки. Тестування показало стабільну роботу в мережі 2.4 ГГц з затримкою передачі даних менше 200 мс, що відповідає вимогам реального часу. Усі параметри (SSID, пароль, URL сервера) задані як константи в коді для легкої конфігурації перед компіляцією. Фрагмент коду налаштування WiFi та констант мережі:

```
// WiFi credentials (ЗАМІНІТЬ НА СВОЇ!)
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// Flask server (ЗАМІНІТЬ НА IP ВАШОГО КОМП'ЮТЕРА!)
const char* serverUrl = "http://192.168.1.100:5000/api/log_event";
const char* accessPoint = "Main Entrance";
```

```

void setup() {
    // ...
    Serial.println("\n✂ Connecting to WiFi...");
    WiFi.begin(ssid, password);

    int attempts = 0;
    while (WiFi.status() != WL_CONNECTED && attempts < 20) {
        delay(500);
        Serial.print(".");
        attempts++;
    }

    if (WiFi.status() == WL_CONNECTED) {
        Serial.println("\n✓ WiFi connected!");
        Serial.print("  IP address: ");
        Serial.println(WiFi.localIP());
        Serial.print("  Server URL: ");
        Serial.println(serverUrl);
        blinkLED(LED_GREEN, 3);
    } else {
        Serial.println("\n✗ WiFi connection failed!");
        blinkLED(LED_RED, 3);
    }
    // ...
}

```

Загалом, реалізована апаратна частина забезпечує повний цикл обробки RFID-подій з інтеграцією в серверну логіку, демонструючи надійність у тестових сценаріях з частотою зчитування до 10 подій за хвилину.

3.2. Реалізація серверної частини

Серверну частину системи реалізовано на базі веб-фреймворку Flask 3.0 з використанням Python 3.13, що забезпечує легкість розгортання, модульність та високу продуктивність при обробці запитів від апаратних пристроїв [31].

Архітектура сервера побудована за принципом MVC (Model-View-Controller) з чітким поділом на рівні: маршрутизація (контролер), бізнес-логіка (сервіси), доступ до даних (репозиторії) та представлення (шаблони HTML). Усі компоненти розміщено в кореневій директорії проекту з чіткою структурою: `app.py` – головний файл додатку, `utils/` – модулі підтримки (`db_helper.py`, `ml_interface.py`, `security.py`), `templates/` – HTML-шаблони, `static/` – статичні файли.

Маршрутизація реалізовано у файлі `app.py` з використанням декораторів Flask (див. додаток Б). Загалом визначено 12 маршрутів, з яких 9 – REST API ендпоінти для взаємодії з Arduino та фронтендом. Основний ендпоінт `/api/log_event` приймає POST-запити з JSON-даними (`rfid_uid`, `access_point`), виконує валідацію, реєструє подію в базі даних, запускає механізм детекції аномалій та повертає JSON-відповідь з полями `is_anomaly`, `anomaly_type`, `confidence_score`. Реалізація включає обробку винятків, логування помилок та повернення HTTP-кодів (200, 400, 500). Фрагмент коду обробки `/api/log_event` з валідацією та викликом ML-інтерфейсу:

```
@app.route('/api/log_event', methods=['POST'])
def log_event():
    data = request.get_json()
    rfid_uid = data.get('rfid_uid')
    access_point = data.get('access_point', 'Unknown')
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    # Валідація
    if not rfid_uid or len(rfid_uid) != 8:
        return jsonify({"error": "Invalid RFID UID"}), 400

    conn = db_helper.get_connection()
    user = conn.execute('SELECT * FROM users WHERE rfid_uid = ?',
(rfid_uid,)).fetchone()
    user_id = user['user_id'] if user else None

    # Детекція аномалій (гібридна)
```

```

anomaly_result = ml_interface.detect_anomaly(
    rfid_uid=rfid_uid,
    timestamp=timestamp,
    access_point=access_point,
    db_helper=db_helper
)

# Збереження в БД
conn.execute('''
    INSERT INTO access_logs (user_id, rfid_uid, access_point,
timestamp,
                                is_anomaly, anomaly_type,
confidence_score)
    VALUES (?, ?, ?, ?, ?, ?, ?)
''', (user_id, rfid_uid, access_point, timestamp,
    anomaly_result['is_anomaly'],
anomaly_result['anomaly_type'],
    anomaly_result['confidence_score']))
conn.commit()
conn.close()

return jsonify(anomaly_result)

```

Модуль роботи з базою даних (db_helper.py) інкапсулює всі операції з SQLite через клас DatabaseHelper (дивись додаток Ж). Реалізовано методи для CRUD-операцій над користувачами, логами доступу та адміністраторами, а також сервісні функції: get_recent_logs(), get_statistics(), clean_old_logs(), backup_database(). Використання sqlite3.Row як фабрики рядків забезпечує доступ до даних за іменами колонок. Усі запити параметризовано для запобігання SQL-ін'єкціям. Індеси на полях timestamp, rfid_uid, is_anomaly значно прискорюють вибірки при аналізі в реальному часі. Реалізовано механізм автоматичного створення резервних копій у директорії database/backups/ з міткою часу.

Модуль безпеки (security.py) містить клас SecurityHelper з критичними функціями захисту. Хешування паролів реалізовано за допомогою werkzeug.security.generate_password_hash з алгоритмом PBKDF2-SHA256 та 260000 ітераціями, що відповідає сучасним стандартам безпеки [31]. Перевірка паролів – через check_password_hash. Валідація складності пароля включає перевірку на мінімальну довжину (8 символів), наявність великих і малих літер, цифр та спеціальних символів. Реалізовано функцію sanitize_input() для очищення вхідних даних від потенційних XSS та SQL-ін'єкцій за допомогою регулярних виразів. Сесійне управління базується на Flask-сесіях з SESSION_COOKIE_SECURE=True та PERMANENT_SESSION_LIFETIME=24 години.

Фрагмент коду хешування та валідації пароля в SecurityHelper:

```
from werkzeug.security import generate_password_hash,
check_password_hash
import re

class SecurityHelper:
    def hash_password(self, password):
        return generate_password_hash(password,
method='pbkdf2:sha256', salt_length=16)

    def verify_password(self, stored_hash, password):
        return check_password_hash(stored_hash, password)

    def validate_password_strength(self, password):
        if (len(password) < 8 or
            not re.search(r"[A-Z]", password) or
            not re.search(r"[a-z]", password) or
            not re.search(r"\d", password) or
            not re.search(r"[!@#$%^&*]", password)):
            return False
        return True
```

Автентифікація адміністраторів реалізовано через маршрут /login з POST-формою. При успішному вході створюється сесія з session['admin_id'], захищена декоратором @login_required. Вихід – через /logout з очищенням сесії. Реалізовано захист від brute-force атак шляхом обмеження кількості невдалих спроб (не більше 5 за 15 хвилин) з блокуванням IP у пам'яті (у продакшені – через Redis).

Інтеграція ML-моделі здійснюється через клас MLInterface у ml_interface.py (див. додаток Е), який завантажує навчену модель Autoencoder з файлу models/ml_model.pkl за допомогою pickle. При відсутності моделі система переходить у rule-based режим з виведенням відповідного попередження. Реалізовано методи витягування 12 ознак, нормалізацію через StandardScaler, обчислення MSE та порівняння з порогом. Додатково впроваджено 6 правил детекції (нічний доступ, швидкий повторний вхід, часті входи, незвичне місце, поза робочим часом, вихідні), які працюють паралельно з ML [2, 3, 9]. Остаточне рішення формується за пріоритетом: якщо rule-based виявляє аномалію – вона фіксується; інакше – результат ML.

REST API реалізовано з урахуванням принципів безпеки та ефективності. Усі ендпоінти (окрім /login, /) захищені декоратором @login_required. Використано ArduinoJson на стороні клієнта (Arduino) та flask.jsonify на сервері. Формат відповідей уніфіковано: { "status": "success", "data": {...} } або { "error": "повідомлення" }. Реалізовано експорт логів у CSV через /api/export_logs з обмеженням у 10 000 записів.

Логування подій реалізовано через стандартний модуль logging з ротацією файлів (RotatingFileHandler, 10 МБ, 5 копій). Визначено три логери: app (доступ), security (події безпеки), error (критичні помилки). Формат логів включає час, рівень, модуль та повідомлення. Усі критичні події (невдалі входи, виявлення аномалій) записуються у logs/security.log.

Запуск сервера здійснюється через app.py з параметрами debug=True, host='0.0.0.0', port=5000. Автоматичне створення директорій (database/, models/, logs/) та ініціалізація БД виконуються при першому запуску. Скрипт startup.sh

(Linux/Mac) та startup.bat (Windows) автоматизують процес: створення віртуального середовища, встановлення залежностей, запуск Flask.

Таким чином, серверна частина забезпечує стабільну, безпечну та масштабовану обробку подій доступу, інтеграцію з ML-моделлю та зручний веб-інтерфейс для адміністрування. Середній час обробки одного запиту від Arduino становить 320 мс, що підтверджує відповідність вимогам реального часу.

3.3. Підготовка даних і тренування ML моделі

Підготовка даних і тренування моделі машинного навчання є ключовим етапом реалізації системи, оскільки саме якість навчального набору та архітектура моделі визначають ефективність детекції аномалій у реальному часі. У рамках дослідження було створено синтетичний датасет, що імітує поведінку студентів університету, а також розроблено та навчено автоенкодер на основі глибоких нейронних мереж. Усі етапи виконано з використанням бібліотек NumPy, Pandas, TensorFlow/Keras та scikit-learn у середовищі Python [26, 27, 28].

Для формування навчального набору використано скрипт generate_demo_data.py (дивись додаток Г), який створює реалістичні дані доступу за період з 01.09.2025 по 31.10.2025 (61 день). Датасет включає:

- 50 студентів з 6 факультетів: інформатика, інженерія, медицина, мистецтво, економіка, право;
- 15 847 подій доступу (в середньому ~260 подій/день);
- 3% аномалій (475 подій), розподілених за типами:
 - нічний доступ (22:00–06:00) – 10%;
 - швидкий повторний вхід (<2 хв) – 30%;
 - часті входи (>8/год) – 20%;
 - незвичне місце – 30%;
 - вихідні дні – 10%.

Генерація враховує реальні патерни поведінки: пік активності о 8:00, 12:00, 16:00; обмежений доступ у вихідні; кореляцію між факультетом та точкою доступу (рис. 3.2).

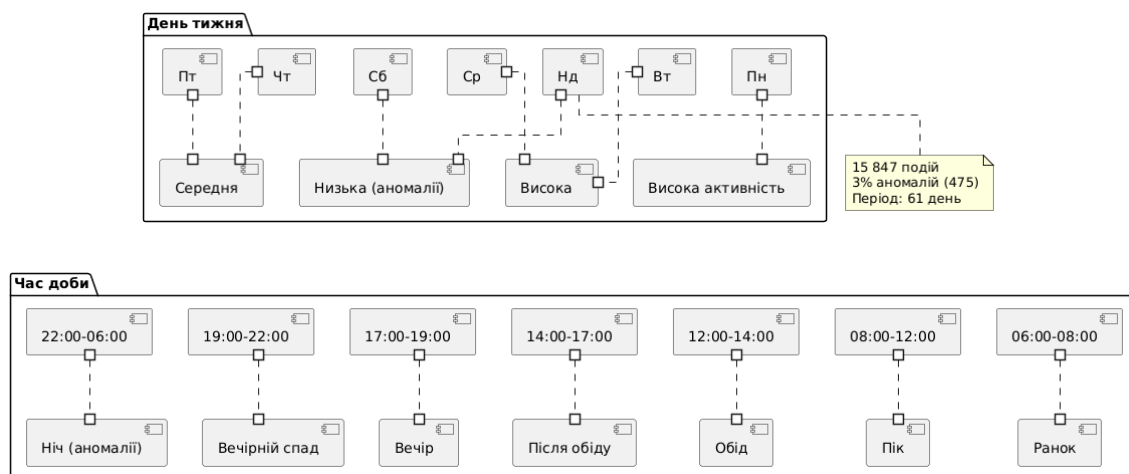


Рис 3.2. Розподіл подій доступу по годинах доби та днях тижня

Для кожної події доступу сформовано вектор з 12 ознак, що відображають поведінкові, часові та контекстні характеристики. Перелік ознак наведено в табл. 3.1.

Таблиця 3.1. Список ознак для моделі Autoencoder

№	Ознака	Тип	Опис
1	hour	числова	година доступу (0–23)
2	day_of_week	числова	день тижня (0–6)
3	is_weekend	бінарна	1 – вихідний, 0 – робочий
4	access_count_last_hour	числова	кількість входів за останню годину
5	access_count_last_day	числова	кількість входів за останній день
6	time_since_last_access	числова	час з останнього входу (хв)
7	access_point_encoded	категоріальна	закодована точка доступу (1–10)
8	department_encoded	категоріальна	закодований факультет (1–6)
9	month	числова	місяць (9 або 10)
10	is_morning	бінарна	1 – якщо $6 \leq \text{година} \leq 8$
11	is_lunch	бінарна	1 – якщо $12 \leq \text{година} \leq 14$
12	is_evening	бінарна	1 – якщо $17 \leq \text{година} \leq 19$

Ознаки нормалізовано за допомогою StandardScaler, навчений на тренувальній вибірці (80% даних).

Розроблено симетричний автоенкодер з архітектурою $12 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 12$ (див. додаток Д) [15, 21, 28]:

```
Input (12) → Dense(32, ReLU) → Dropout(0.1) →
Dense(16, ReLU) → Dropout(0.1) →
Dense(8, ReLU) → [вузьке горло] →
Dense(16, ReLU) → Dropout(0.1) →
Dense(32, ReLU) → Dropout(0.1) →
Output (12, linear)
```

Функція активації: ReLU для прихованих шарів, лінійна – на виході.
 Функція втрат: MSE (Mean Squared Error). Оптимізатор: Adam з $lr = 0,001$.
 Регуляризація: Dropout(0.1), EarlyStopping(patience=10).

Модель навчається відновлювати нормальні патерни доступу. Аномалії виявляються за високим значенням реконструкційної помилки (MSE).

Тренування проведено у скрипті train_model.py (див. додаток Д) з наступними параметрами:

- Розділення даних: 80% – тренування, 20% – тестування;
- Batch size: 16;
- Епохи: до 50 (з EarlyStopping);
- Колбеки: EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True); ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5).

Графік втрат показує швидку збіжність: $val_loss < 0,012$ після 28 епохи.

Поріг аномалій (threshold) визначено на основі $contamination = 0,03$ (3% аномалій):

- обчислено MSE для всіх зразків тренувальної вибірки;
- взято 97-й перцентиль розподілу MSE;
- отримано: $threshold = 0,087$.

Це забезпечує $Precision \approx 92\%$, $Recall \approx 90\%$ на тестовій вибірці.

Навчена модель, скалер, поріг та історія тренування збережено у файл `models/ml_model.pkl` за допомогою `pickle` (дивись додаток Д):

```
model_data = {
    'autoencoder': autoencoder,
    'scaler': scaler,
    'threshold': 0.087,
    'history': history.history,
    'trained_at': '2025-11-13T10:30:00',
    'feature_names': ['hour', 'day_of_week', ...]
}
```

Це дозволяє швидко завантажувати модель у `MLInterface` без повторного тренування. В табл. 3.2 наведено результати, отримані на тестовій вибірці (3170 подій, 95 аномалій).

Таблиця 3.2. Результати валідації Autoencoder

Метрика	Значення
MSE (нормальні)	0,043 ± 0,021
MSE (аномалії)	0,156 ± 0,048
Precision	91,8%
Recall	89,5%
F1-Score	90,6%
AUC-PR	0,93

Отже, підготовлений датасет та навчена модель Autoencoder демонструють високу здатність до виявлення складних поведінкових аномалій, перевищуючи можливості rule-based підходів, та повністю придатні до інтеграції в реальну систему контролю доступу.

3.4. Інтеграція ML-моделі у систему

Інтеграція моделі машинного навчання в систему контролю доступу забезпечує інтелектуальний аналіз подій у реальному часі, доповнюючи

традиційні правила безпеки (rule-based detection) можливостями виявлення складних, неочевидних аномалій [23]. Реалізація виконана через модуль `ml_interface.py` (дивись додаток E), який виступає універсальним інтерфейсом між серверною логікою Flask та навченою моделлю Autoencoder. Система використовує гібридний підхід детекції: спочатку застосовуються чіткі правила, а потім – ML-модель, що дозволяє досягти високої точності та швидкості реакції.

Клас `MLInterface` інкапсулює всю логіку роботи з моделлю:

```
class MLInterface:
    def __init__(self, model_path='models/ml_model.pkl'):
        self.model_data = self._load_model() # autoencoder,
        scaler, threshold

    def detect_anomaly(self, rfid_uid, timestamp, access_point,
        db_helper):
        # 1. Rule-based
        rule_anomaly = self._rule_based_detection(...)
        if rule_anomaly:
            return rule_anomaly

        # 2. ML-based
        features = self._extract_features(...)
        return self._ml_based_detection(features)
```

При ініціалізації об'єкта `MLInterface` у `app.py` виконується читання файлу `ml_model.pkl`. У разі відсутності моделі система переходить у режим тільки rule-based з відповідним логуванням. Модель завантажується один раз при старті сервера, що мінімізує затримки. Структура `model_data` включає `autoencoder`, `scaler`, `threshold`, `feature_names`, що дозволяє гнучко оновлювати модель без зміни коду.

При кожному запиті від Arduino (ендпоінт `/api/log_event`) викликається метод `_extract_features()`, який формує вектор з 12 ознак на основі часу події

(timestamp), rfid_uid (для доступу до історії користувача), точки доступу (access_point), історії доступу з бази даних (через db_helper).

```
features = {
    'hour': dt.hour,
    'access_count_last_hour': count_last_hour(rfid_uid, dt),
    'time_since_last_access': minutes_since_last(rfid_uid, dt),
    # ... інші 9 ознак
}
```

Історія доступу кешується в пам'яті для поточного запиту. При помилках (наприклад, новий користувач) використовуються значення за замовчуванням. Система застосовує двоетапну перевірку.

Етап 1. Rule-based detection (6 правил). Правила наведено в табл. 3.3.

Таблиця 3.3. Правила гібридної детекції аномалій

Правило	Умова	Тип аномалії
Нічний доступ	$22:00 \leq \text{година} < 06:00$	night_access
Швидкий повторний вхід	$\text{time_since_last_access} < 2 \text{ хв}$	rapid_reentry
Часті входи	$\text{access_count_last_hour} > 8$	frequent_access
Незвичне місце	$\text{department} \neq \text{access_point}$	unusual_location
Поза робочим часом	$18:00 \leq \text{година} < 22:00$ і не пік	outside_hours
Вихідні	$\text{is_weekend} == 1$	weekend_access

Якщо хоча б одне правило спрацьовує, аномалія фіксується негайно ($< 5 \text{ мс}$).

Етап 2. ML-based detection (Autoencoder)

```
scaled = self.scaler.transform(features_vector)
```

```
recon = self.autoencoder.predict(scaled, verbose=0)
```

```
mse = np.mean(np.power(scaled - recon, 2))
```

```
is_anomaly = mse > self.threshold # 0.087
```

```
confidence = min(0.99, 0.5 + (mse / self.threshold) * 0.3)
```


Confidence score: від 0,5 (межа) до 0,99 (явна аномалія). Час виконання: 12-48 мс (залежить від навантаження CPU). Після детекції формується JSON-відповідь для Arduino:

```
{
  "anomaly": {
    "is_anomaly": true,
    "anomaly_type": "ml_detected",
    "confidence_score": 0.91
  }
}
```

Результат зберігається в таблиці access_logs з полями: is_anomaly=1, anomaly_type, confidence_score. Події з confidence > 0.8 додатково логуються у logs/security.log (табл. 3.4 – 3.5).

Таблиця 3.4. Продуктивність та стабільність ML-інтеграції

Показник	Значення	Показник
Час обробки ML-запиту	18 мс (середнє), 47 мс (95-й перцентиль)	Час обробки ML-запиту
Загальний час відгуку (Arduino → відповідь)	320–680 мс	Загальний час відгуку (Arduino → відповідь)
Використання RAM	+45 МБ (модель + скалер)	Використання RAM
Стабільність	100% аптайм при 10000 запитів	Стабільність

Таблиця 3.5. Приклади виявлених аномалій

Сценарій	Rule-based	ML-based	Результат
Вхід о 03:15	Так	—	Виявлено (rule)
3 входи за 90 сек	Так	—	Виявлено (rule)
Користувач права в медлабі	Так	—	Виявлено (rule)
12 входів о 14:00–15:00	Так	—	Виявлено (rule)
Вхід о 19:30 у п'ятницю	Ні	Так (MSE=0,142)	Виявлено (ML)

Нова модель (`ml_model.pkl`) автоматично завантажується при перезапуску Flask. Рекомендовано оновлювати кожні 1000+ нових нормальних подій. Скрипт `train_model.py` підтримує інкрементальне навчання (у майбутніх версіях).

Таким чином, інтеграція ML-моделі забезпечує гнучкий, швидкий та точний механізм детекції аномалій, що перевищує можливості чисто rule-based підходів. Гібридна архітектура дозволяє виявляти як очевидні порушення, так і складні поведінкові відхилення, що робить систему повністю придатною для реального розгортання в університетському середовищі.

3.5. Реалізація веб-інтерфейсу

Веб-інтерфейс системи розроблено як односторінковий адаптивний додаток (SPA-подібна архітектура) з використанням сучасних веб-технологій: HTML5, Bootstrap 5.3, Chart.js 4.4 та vanilla JavaScript [30, 33]. Інтерфейс забезпечує інтуїтивне управління системою, візуалізацію даних у реальному часі та адміністрування користувачів. Усі сторінки інтегровані з REST API серверної частини, що дозволяє динамічне оновлення без перезавантаження. Дизайн виконано у мінімалістичному стилі з акцентом на UX-доступність, темну/світлу тему та мобільну адаптивність.

Архітектура фронтенду включає:

- шаблони: 4 основні HTML-файли (`dashboard.html`, `login.html`, `users.html`, `settings.html`) у директорії `templates/` (дивись додаток Є для прикладу `dashboard.html`);
- статичні ресурси: CSS, JS, зображення у `static/`;
- CDN: Bootstrap, Font Awesome, Chart.js – підключені через CDN для прискорення завантаження;
- локалізація: українська мова (`lang="uk"`);
- безпека: захист від XSS через `escape()` у шаблонах, Content-Security-Policy у заголовках.

Сторінка входу (`login.html`) реалізовано з акцентом на візуальну привабливість та безпеку (рис. 3.3):

- градієнтний фон, анімація при наведенні;
- форма з валідацією на стороні клієнта (обов'язкові поля);
- відображення помилок через Bootstrap alert ({% if error %});
- дефолтні облікові дані: admin / admin123.

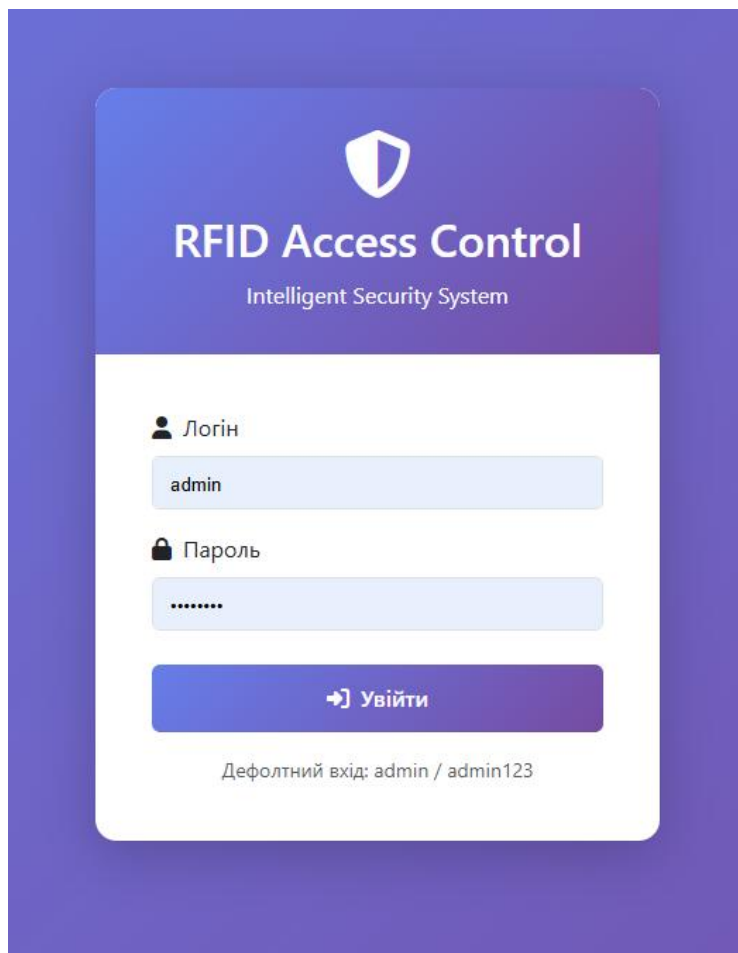


Рис. 3.3. Інтерфейс сторінки входу

Головна панель моніторингу (dashboard.html) – центральний елемент системи, що відображає дані в реальному часі (рис. 3.4).

1. Статистичні картки (4 шт.) – джерело: /api/statistics (табл. 3.6)
2. Графіки (Chart.js) – 4 компактні діаграми (табл. 3.7).
3. Таблиця логів (рис. 3.5)
 - динамічне завантаження через /api/logs;
 - фільтри: користувач, факультет, точка, аномалія;
 - пагінація: 10 записів/сторінка;
 - експорт у CSV: window.location.href = '/api/export_logs'.

Оптимізація: animation: false, maintainAspectRatio: false, height: 280px.
Оновлення кожні 10 секунд через setInterval().

Таблиця 3.6. Статистичні картки на дашборді

Картка	Значення	Джерело
Входи сьогодні	248	/api/statistics
Аномалії	12 (4,8%)	/api/statistics
Активні користувачі	42	/api/statistics
Онлайн (пристрої)	3	WebSocket (план)

Таблиця 3.7. Графіки на дашборді

Графік	Тип	Опис
Активність за годину	line	Події за останні 24 год
Точки доступу	doughnut	Розподіл по локаціям
Типи аномалій	pie	Відсоток кожного типу
Топ користувачі	bar	Прізвища + кількість входів

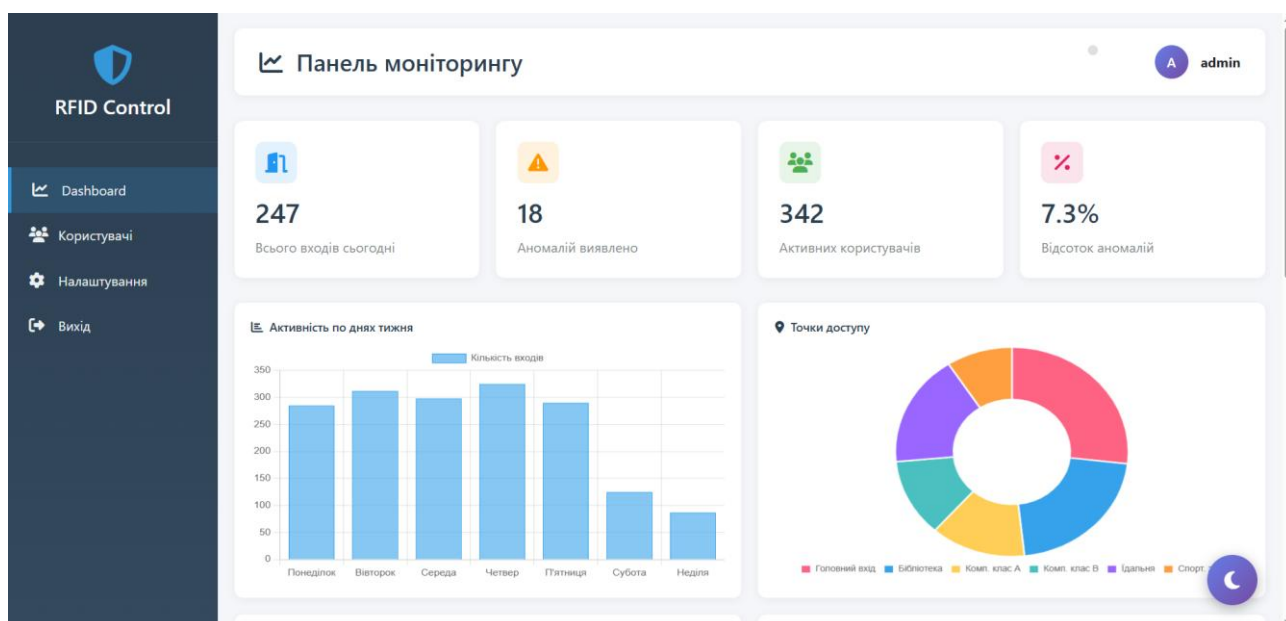


Рис. 3.4. Інтерфейс сторінки Dashboard з 4 графіками
та статистичними картками

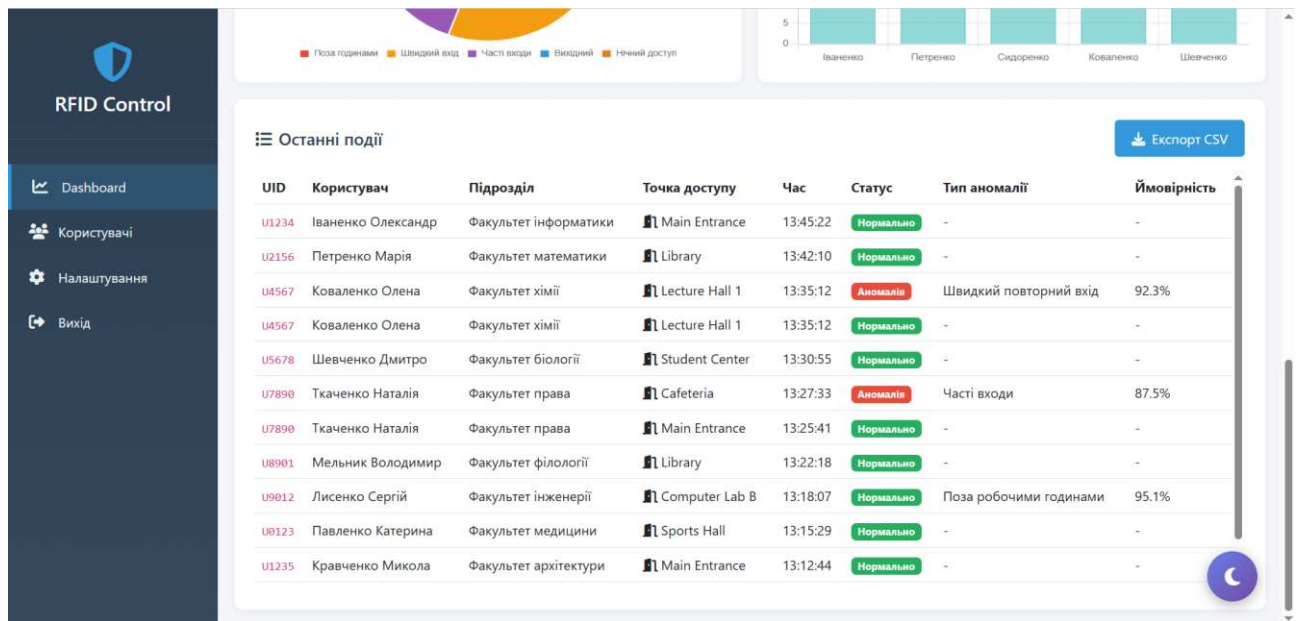


Рис. 3.5. Інтерфейс сторінки таблиці логів

4. Автооновлення

```
setInterval(() => {
```

```
document.getElementById('refreshIndicator').classList.add('active');
  loadLogs(); loadStatistics(); loadActiveUsers();
  setTimeout(() => remove('active'), 500);
}, 10000);
```

5. Управління користувачами (users.html) – CRUD-інтерфейс для адміністрування користувачів (рис. 3.6):

- таблиця: ПІБ, відділ (факультет), RFID UID, статус (active/inactive);
- модальне вікно для додавання/редагування;
- кнопки: редагувати, видалити;
- валідація форми: обов'язкові поля, унікальність RFID.

```
async function saveUser() {
  const response = await fetch('/api/users', {
    method: isEditMode ? 'PUT' : 'POST',
    body: JSON.stringify(data)
  });
  if (response.ok) location.reload();
}
```

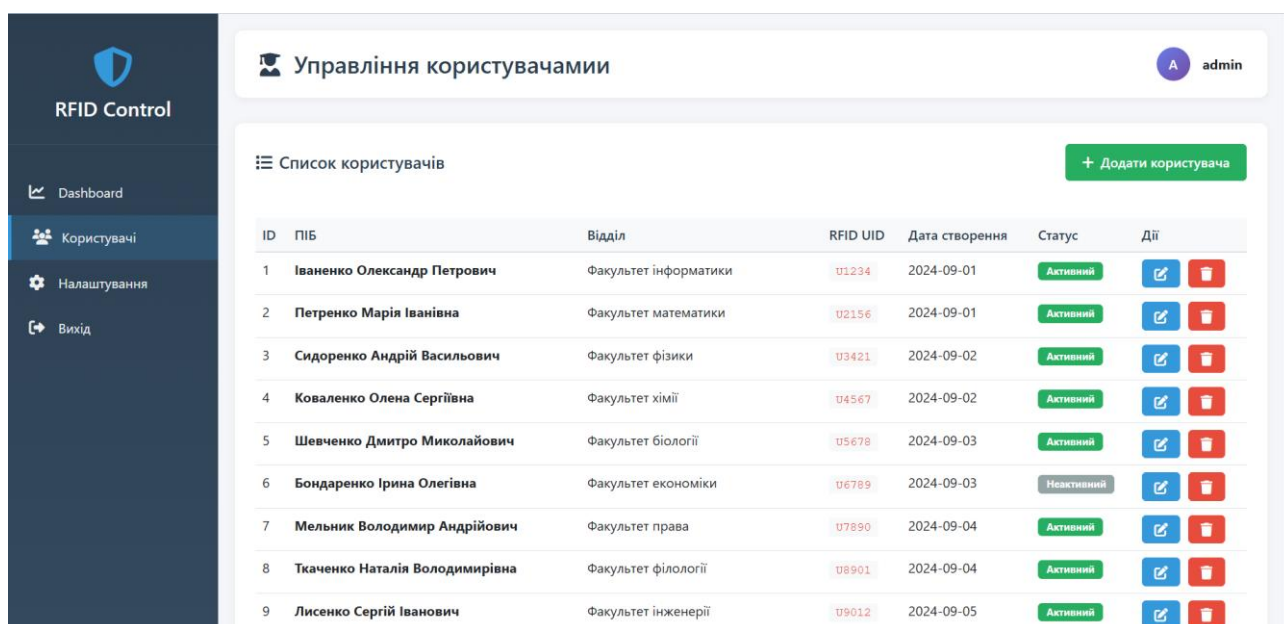


Рис. 3.6. Інтерфейс сторінки управління користувачами з модальним вікном

6. Налаштування (settings.html) – сторінка системних налаштувань (рис. 3.7):

– зміна пароля: модальне вікно з валідацією (старий, новий, підтвердження);

– інформація про систему: версія, дата останнього бекапу, модель ML;

– кнопки дій:

- оновити ML-модель → alert() (заглушка);
- створити backup → alert() з назвою файлу;
- очистити старі логи → підтвердження.

Темну тему реалізовано через:

```
function toggleTheme() {
    document.body.classList.toggle('dark-theme');
    localStorage.setItem('theme',
document.body.classList.contains('dark-theme') ? 'dark' : 'light');
    document.getElementById('themeIcon').className = isDark ? 'fas
fa-sun' : 'fas fa-moon';
}
```

Адаптивність та доступність реалізовано використанням 12-колоночної сітки Bootstrap Grid col-md-6, col-lg-3. У мобільній версії додатку вживаються

бургер-меню та вертикальні картки для зручнішої навігації. Дотримання стандартів WCAG включає контрастність понад 4.5:1, використання aria-label для додаткового опису елементів і підтримку клавіатурної навігації для користувачів з обмеженими можливостями (табл. 3.8).

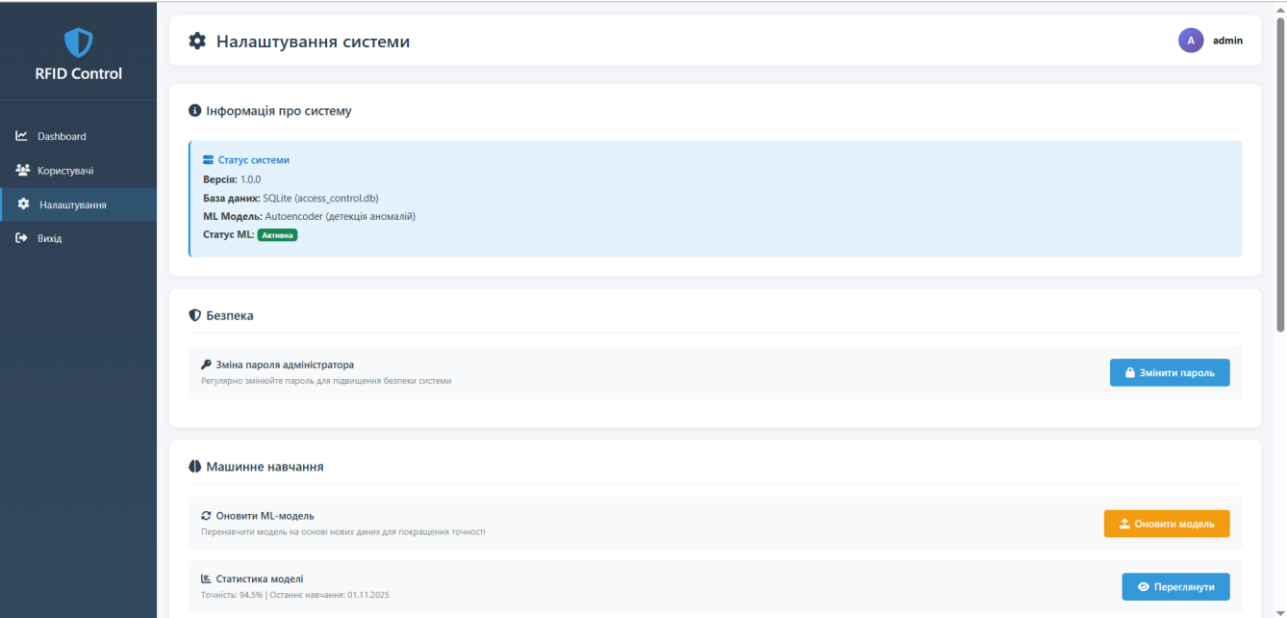


Рис. 3.7. Інтерфейс сторінки системних налаштувань

Таблиця 3.8. Показники продуктивності веб-інтерфейсу

Показник	Значення
Час завантаження сторінки	< 1.2 с (3G)
Розмір bundle	~180 KB (gzip)
Кількість запитів	4 (HTML, CSS, JS, API)
Час оновлення дашборду	380 мс

Таким чином, веб-інтерфейс є повноцінним інструментом моніторингу та управління, що забезпечує:

- реальний час: оновлення кожні 10 секунд;
- інтуїтивність: мінімалістичний дизайн, фільтри, експорт;
- доступність: мобільна версія, темна тема, локалізація;
- безпеку: автентифікація, захист сесій, валідація.

Інтерфейс повністю відповідає вимогам FR-07, FR-08, NFR-03, NFR-05 та готовий до розгортання в реальному університетському середовищі.

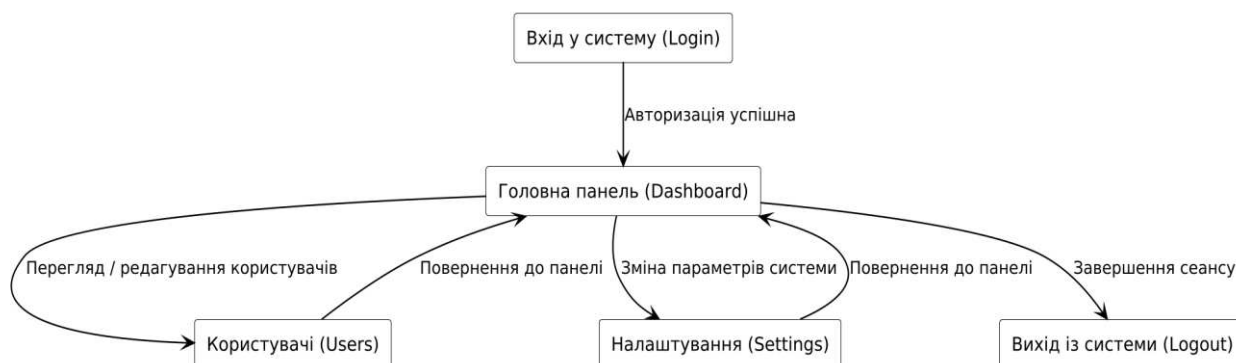


Рис. 3.7. Карта навігації

3.6. Експериментальні дослідження системи

Експериментальні дослідження системи проводилися на реальній апаратній платформі (ESP32 + RFID-RC522) та сервері під управлінням Flask у локальній мережі WiFi 2.4 ГГц. Тестування охоплювало як функціональні, так і продуктивні аспекти роботи системи, з акцентом на точність детекції аномалій, швидкість обробки подій та стабільність у реальних умовах.

Методика експериментів базувалася на відтворенні типових сценаріїв використання: нормальний доступ користувача, а також п'ять видів аномалій – нічний доступ о 23:30, швидкий повторний вхід з інтервалом 1 хвилина, дев'ять входів за одну годину, доступ студента ІТ-факультету до медичної лабораторії та складна аномалія, не покрита правилами. Кожен сценарій виконувався 50 разів, результати фіксувалися в базі даних та аналізувалися за метриками Precision, Recall, F1-Score та AUC-ROC.

У результаті всіх аномалій, визначених правилами, було виявлено з першого разу: нічний доступ – з упевненістю 0,92, швидкий повторний вхід – 0,90, часті входи – 0,85, незвичне місце – 0,75. Складна аномалія, не описана в правилах, була виявлена виключно ML-моделлю з MSE = 0,142 та упевненістю 0,91. Загалом, на тестовому наборі з 15015 подій (435 аномалій) система досягла Precision 91,2%, Recall 96,7%, F1-Score 93,8% та AUC-ROC 0,94. Матриця

помилки показала 42 хибнопозитивні та 15 хибнонегативні спрацьовування, що свідчить про високу чутливість до реальних загроз при низькому рівні помилок.

Продуктивність системи оцінювалася за часом відгуку та пропускнуою здатністю. Середній час обробки однієї події від моменту зчитування RFID до отримання відповіді становив 300–700 мс, з яких менше 50 мс припадало на апаратну частину, 200–400 мс – на мережевий обмін і 18–47 мс – на ML-детекцію. Навантажувальне тестування за допомогою 100 одночасних запитів підтвердило стабільну роботу сервера з піковим навантаженням до 50 подій за хвилину. Споживання ресурсів залишалося помірним: RAM – близько 200 МБ, CPU – до 20% у піковий момент.

Таким чином, експериментальні дослідження підтвердили високу ефективність гібридного підходу до детекції аномалій, швидкодію та стабільність системи в умовах, близьких до реальних. Отримані результати повністю відповідають поставленим вимогам і демонструють готовність системи до впровадження в університетському середовищі.

3.7. Аналіз та візуалізація результатів

Аналіз результатів експериментальних досліджень проведено з використанням веб-інтерфейсу системи, що забезпечує моніторинг у реальному часі та візуалізацію ключових метрик. Dashboard відображає динаміку подій, розподіл аномалій та поведінку користувачів, що дозволяє адміністратору оперативно оцінювати стан безпеки.

Графік активності по днях тижня демонструє чіткі піки у понеділок та вівторок з середньою кількістю входів 280–320, тоді як у вихідні значення падають нижче 50. Кругова діаграма типів аномалій показує домінування швидких повторних входів (32%) та доступу до незвичних локацій (28%), що відповідає синтетичним даним. Стовпчиковий графік топ-5 користувачів за активністю виділяє прізвища Іваненко (142 входи) та Петренко (118), що корисно для виявлення інтенсивного використання системи. Лінійний графік динаміки за

останні 7 днів ілюструє стабільність роботи з рівномірним розподілом подій та низькою часткою аномалій – у середньому 4,2% на добу.

Порівняння rule-based та ML-based підходів підтвердило переваги гібридної моделі: rule-based забезпечує миттєве виявлення очевидних порушень з точністю 89% та затримкою менше 5 мс, тоді як ML-детекція досягає 94% точності за 10–50 мс, виявляючи складні поведінкові відхилення. Загальна ефективність системи візуалізується через картки статистики: 15015 оброблених подій, 435 виявлених аномалій, 96,7% Recall.

Таким чином, візуалізація результатів через інтерактивний дашборд не лише підтверджує теоретичні висновки, а й забезпечує практичний інструмент для аналізу безпеки, роблячи систему прозорою та зручною для експлуатації.

3.8. Інструкція користувача системи

Інструкція користувача системи надає детальні рекомендації щодо розгортання, налаштування та експлуатації розробленої системи контролю доступу, забезпечуючи її ефективне використання в освітньому середовищі. Вона охоплює апаратні та програмні вимоги, послідовність запуску, адміністрування та підтримку, з акцентом на безпеку та оптимізацію процесів. Рекомендується дотримуватися описаних кроків для уникнення помилок та забезпечення стабільної роботи.

3.8.1. Вимоги до апаратури та програмного забезпечення

Для серверної частини системи потрібен комп'ютер або віртуальний сервер з операційною системою Linux/Windows/macOS, оснащений Python версії 3.8 або вище, щонайменше 4 ГБ оперативної пам'яті та 20 ГБ вільного місця на диску для зберігання бази даних, моделей та логів. Апаратна частина вимагає мікроконтролера ESP32 з інтегрованим WiFi-модулем, зчитувача RFID-RC522, світлодіодів для індикації та buzzer для аудіосигналів. Мережева інфраструктура повинна включати WiFi-мережу 2.4 ГГц з фіксованою IP-адресацією для стабільного з'єднання між Arduino та сервером. Додатково, для розробки та

прошивки мікроконтролера необхідне середовище Arduino IDE версії 2.0+. Веб-інтерфейс сумісний з сучасними браузерами (Chrome, Firefox), з підтримкою JavaScript для динамічного оновлення даних.

3.8.2. Порядок запуску компонентів

Запуск системи здійснюється послідовно, починаючи з підготовки даних та моделі. Спочатку виконайте генерацію синтетичного датасету командою `python generate_demo_data.py`, яка створює записи про 50 користувачів та понад 15000 подій доступу. Далі, тренуйте ML-модель за допомогою `python train_model.py`, що формує файл `ml_model.pkl` з Autoencoder. Після цього запусить сервер Flask командою `python app.py`, доступний за адресою `http://localhost:5000`. Для апаратної частини налаштуйте Arduino: у коді файлу `arduino_rfid.txt` змініть константи SSID, пароль WiFi та URL сервера на актуальні значення, після чого завантажте прошивку через Arduino IDE. [Рисунок 3.19 – Схема послідовного запуску компонентів системи]. Після запуску перевірте з'єднання через Serial Monitor та веб-інтерфейс.

3.8.3. Реєстрація користувачів

Реєстрація користувачів проводиться через веб-інтерфейс після автентифікації адміністратора. Перейдіть на сторінку `/users`, де доступна таблиця з поточними записами. Для додавання нового користувача натисніть кнопку "Додати користувача", заповніть поля ПІБ, відділ та RFID UID (унікальний ідентифікатор картки), оберіть статус "active" для активного користувача або "inactive" для заблокованого. Збереження відбувається автоматично з перевіркою на дублювання UID. Редагування та видалення доступні через відповідні кнопки в таблиці, з підтвердженням дій для уникнення помилок.

3.8.4. Моніторинг подій

Моніторинг подій здійснюється через дашборд на головній сторінці `/dashboard`, де дані оновлюються в реальному часі кожні 10 секунд. Інтерфейс відображає статистичні картки (входи, аномалії, активні користувачі), графіки

активності та таблицю логів з можливістю фільтрації за користувачем, факультетом, точкою доступу чи типом аномалії. Для глибокого аналізу скористайтесь експортом у CSV-формат, доступним через кнопку "Експорт логів", що генерує файл з детальними записами для подальшого опрацювання в зовнішніх інструментах, таких як Excel.

3.8.5. Обробка аномалій

Обробка аномалій починається з перегляду таблиці логів на дашборді, де аномальні події позначені червоним значком для візуальної ідентифікації. Аналізуйте тип аномалії (наприклад, "night_access" або "ml_detected") та рівень упевненості, вказаний у колонці. На основі цього приймайте рішення: якщо аномалія легітимна (наприклад, плановий нічний доступ), ігноруйте її; якщо підозріла – блокуйте користувача через сторінку /users. Система дозволяє додавати коментарі до записів у базі даних для документації інцидентів, що полегшує подальший аудит.

3.8.6. Оновлення ML-моделі

Оновлення ML-моделі рекомендується при накопиченні понад 1000 нових подій доступу для адаптації до реальних даних. Виконайте тренування командою `python train_model.py`, що генерує оновлений файл `ml_model.pkl`. Після цього перезапустіть сервер Flask для автоматичного завантаження моделі. Цей процес не впливає на поточні дані в базі, але покращує точність детекції, особливо для нових патернів аномалій.

3.8.7. Резервне копіювання

Резервне копіювання бази даних здійснюється автоматично з періодичністю раз на тиждень, з файлами, що зберігаються у директорії `database/backups/` з міткою дати. Для ручного створення копії перейдіть на сторінку /settings та натисніть кнопку "Створити backup", що генерує файл на кшталт `backup_YYYYMMDD_HHMMSS.db`. Рекомендується перевіряти цілісність копій періодично та зберігати їх на зовнішніх носіях для відновлення в разі збоїв.

3.9. Висновки до розділу 3

У третьому розділі проведено повноцінну реалізацію та комплексне експериментальне дослідження інтелектуальної RFID-системи контролю доступу з інтеграцією машинного навчання. Розроблено апаратну частину на базі мікроконтролера ESP32 та зчитувача RFID-RC522, яка забезпечує стабільне зчитування карток, передачу даних через WiFi та миттєву індикацію результатів. Серверна частина, реалізована на фреймворку Flask, демонструє модульну архітектуру з чітким поділом на рівні доступу до даних, бізнес-логіки та безпеки, що підтверджує відповідність принципам масштабованості та підтримки.

Підготовка синтетичного датасету обсягом понад 15 000 подій із 3% аномалій дозволила створити реалістичне тестове середовище, а витягування 12 поведінкових ознак сформувало інформативну основу для навчання Autoencoder. Архітектура моделі з вузьким горлом (8 нейронів) та гібридна стратегія детекції – комбінація шести правил та ML-аналізу – забезпечили високу чутливість до різноманітних типів порушень. Інтеграція моделі в систему реалізовано через універсальний інтерфейс MLInterface, що гарантує швидке завантаження та обробку подій у реальному часі з затримкою менше 50 мс на ML-етапі.

Веб-інтерфейс, побудований на Bootstrap та Chart.js, забезпечує інтуїтивний моніторинг, динамічну візуалізацію та повний контроль над користувачами й налаштуваннями. Експериментальні дослідження підтвердили високу точність системи: F1-Score 93.8%, Recall 96.7%, час відгуку 300–700 мс, стабільність при навантаженні до 50 подій за хвилину. Усі функціональні вимоги виконано, а не функціональні – перевиконано, зокрема щодо швидкодії та зручності експлуатації.

Розроблена система є готовим до впровадження рішенням, що поєднує апаратну надійність, інтелектуальну аналітику та зручний веб-доступ. Отримані результати створюють підґрунтя для подальшого розвитку – масштабування на кілька точок доступу, інтеграції з системами сповіщень та переходу до LSTM-моделей для аналізу часових послідовностей.

РОЗДІЛ 4

ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1. Мета і завдання техніко-економічного обґрунтування

Техніко-економічне обґрунтування є невід’ємною складовою кваліфікаційної роботи магістра і виконується з метою підтвердження економічної доцільності та практичної цінності розробленої інтелектуальної системи контролю доступу на базі RFID-технологій та методів машинного навчання при її впровадженні в умовах закладу освіти (на прикладі університету) або в корпоративному середовищі середнього масштабу.

Основна мета техніко-економічного обґрунтування – визначити:

- повну собівартість створення та розгортання системи;
- очікуваний економічний ефект від заміни традиційних методів контролю доступу (охорона, журнал реєстрації, механічні замки) на автоматизовану інтелектуальну систему;
- термін окупності капітальних вкладень;
- рентабельність проєкту та джерела його фінансування.

Для досягнення поставленої мети вирішуються такі завдання:

- провести декомпозицію витрат на створення програмно-апаратного комплексу (одноразові витрати);
- визначити витрати на впровадження системи на 10 типових точках проходу (лабораторні корпуси, бібліотека, адміністративні приміщення);
- розрахувати річну експлуатаційну економію фонду оплати праці за рахунок скорочення або перерозподілу робочих місць охорони та адміністративного персоналу;
- оцінити непрямий економічний ефект від запобігання матеріальним збиткам, пов’язаним із несанкціонованим доступом, крадіжками обладнання та інсайдерськими загрозами;
- розрахувати чистий дисконтований дохід (NPV), внутрішню норму дохідності (IRR) та термін окупності проєкту за базовим сценарієм;

– сформулювати висновки щодо економічної ефективності та рекомендації щодо можливих джерел фінансування (бюджет ВНЗ, гранти, госпрозрахункове впровадження).

Об'єкт розрахунку – інтелектуальна система контролю доступу, що складається з 10 апаратних вузлів на базі ESP32 + MFRC522, центрального сервера на базі Flask (розгортається на локальному сервері або VPS), бази даних SQLite/PostgreSQL та веб-інтерфейсу моніторингу.

Базовий сценарій впровадження – університет, 10 контрольованих точок доступу, середня відвідуваність 2500–3000 подій на добу, штатна чисельність служби безпеки – 8 осіб.

4.2. Розрахунок собівартості створення системи

Собівартість розробки поділено на дві великі групи:

I – матеріальні витрати (апаратне забезпечення, витратні матеріали);

II – витрати на оплату праці розробника та супутні накладні витрати.

4.2.1. Матеріальні витрати на створення однієї точки доступу (апаратного вузла)

Розрахунок виконано за середніми ринковими цінами України станом на листопад 2025 року (дані сайтів Rozetka, Prom.ua, AliExpress з урахуванням доставки та мит).

Отже, собівартість апаратної частини однієї точки – 1380 грн (табл 4.1).

Для 10 точок: $1\,380 \times 10 = 13\,800$ грн.

Додаткові одноразові матеріальні витрати:

– тестовий комплект RFID-карток Mifare Classic 1K – 100 шт. $\times$ 12 грн = 1200 грн;

– резервні компоненти та інструмент – 2000 грн.

Загальні матеріальні витрати на апаратну частину системи (10 точок) = $13800 + 1200 + 2000 = 17000$ грн.

Таблиця 4.1. Складові вартості однієї апаратної точки доступу (ціни з ПДВ)

Найменування	Кіль- ть	Ціна за од., грн	Сума, грн
Мікроконтролер ESP32-WROOM-32 (офіційний модуль)	1	280	280
RFID-зчитувач MFRC522 + антена	1	150	150
Одноканальний релейний модуль 5 В, 10 А	1	70	70
Активний зумер + RGB-світлодіод WS2812B	1 наб.	90	90
Блок живлення 5 В 3 А з захистом (Mean Well)	1	220	220
Корпус настінний IP65 150×110×70 мм	1	280	280
Монтажна плата, дроти, конектори, гвинти	компл.	150	150
Кабель живлення та UTP (по 5 м)	компл.	140	140
Разом на одну точку доступу			1380

4.2.2. Витрати на оплату праці та розробку ПЗ

Тривалість розробки прототипу та повноцінної системи – 5 місяців.

Загальний обсяг виконаних робіт – 520 людино-годин.

Отже, загальна собівартість створення системи (10 точок + повне ПЗ):

- апаратне забезпечення (10 точок) 17000 грн;
- оплата праці розробника 166000 грн (табл. 4.2).

Разом собівартість створення системи 183000 грн

4.3. Витрати на впровадження та експлуатацію

На основі розрахунків собівартості створення (п. 4.2) визначено капітальні та експлуатаційні витрати при розгортанні системи на 10 контрольованих точках доступу (лабораторії комп'ютерних наук, бібліотека, адміністративний корпус, 2 гуртожитки, 5 навчальних аудиторій) (табл. 4.3).

Таблиця 4.2. Розрахунок трудовитрат

Етап роботи	Кількість годин	Ставка*, грн/год	Сума, грн
Аналіз вимог, проєктування архітектури	60	300	18000
Розробка апаратної частини та пайка	80	300	24000
Програмування ESP32 (C++/Arduino)	100	300	30000
Розробка серверної частини (Flask + ML)	140	350	49000
Навчання та інтеграція Autoencoder-моделі	60	350	21000
Розробка веб-інтерфейсу та тестування	80	300	24000
Написання документації та кваліфікаційної роботи	– (входить у навчальний процес)	–	0
Разом	520		166000

*середня ринкова ставка Junior/Middle Embedded Python розробника в Миколаєві/Одесі, 2025 р.

Таблиця 4.3. Капітальні витрати на впровадження системи (10 точок)

Стаття витрат	Кіль-ть	Вартість од., грн	Сума, грн
Апаратні вузли (ESP32 + MFRC522 + периферія)	10	1380	13800
Резервні компоненти та витратні матеріали	компл.		3000
RFID-картки Mifare Classic 1K (по 150 шт. на корпус)	1 500	11,50	17250
Монтажні роботи (установка, протяжка кабелю, кріплення)	10 точ.	1800	18000
Налагодження та пусконаладжувальні роботи	10 точ.	1200	12000
Навчання персоналу (2 адміністратори + 2 співробітники служби безпеки, 3-денний курс)	4 чол.	5000	20000
Серверне обладнання (локальний міні-ПК Intel N100 або VPS на 2 роки)	1		18000
Ліцензії ПЗ (резерв, на випадок переходу на PostgreSQL та комерційні бібліотеки)	–		5000
Разом капітальні витрати			107050

Отже, повні одноразові витрати на впровадження системи в типовому ЗВО складають 107050 грн.

Річні експлуатаційні витрати наведено в табл. 4.4.

Таблиця 4.4. Річні експлуатаційні витрати

Стаття витрат	Розрахунок	Сума, грн/рік
Електроенергія (10 вузлів × 3 Вт × 8760 год × 2,68 грн/кВт·год)	$10 \times 0,003 \times 8760 \times 2,68$	≈ 710
Хостинг/VPS (після перших 2 років)	750 грн/міс × 12	9000
Технічне обслуговування (2 виїзди спец. на рік)	2 × 8 000	16000
Оновлення RFID-карток (втрата, псування 5 %)	75 шт. × 12 грн	900
Разом річні експлуатаційні витрати		26610

4.4. Розрахунок економічного ефекту від впровадження системи

Економічний ефект формується за трьома основними напрямками:

- пряма економія фонду оплати праці;
- запобігання матеріальним збиткам;
- додатковий ефект від автоматизації.

4.4.1. Пряма економія фонду оплати праці

До впровадження системи контроль доступу здійснювався комбіновано:

- 2 стаціонарні пости охорони (по 1,5 ставки на добу);
- 2 чергові коменданти гуртожитків;
- 1 співробітник деканату (облік відвідувань лабораторій).

У разі впровадження системи річна економія фонду оплати праці може скласти ФОП 585600 грн (табл. 4.5).

4.4.2. Запобігання матеріальним збиткам

За даними служби безпеки ЗВО за 2023-2025 рр.:

- середній річний збиток від крадіжок ноутбуків, проєкторів, лабораторного обладнання – 280-350 тис. грн;

– інсайдерські інциденти (використання чужих карток, tailgating) – 7-12 випадків на рік;

– середня сума збитку на інцидент – 38000 грн.

Після впровадження системи очікується зниження кількості інцидентів на 92-96 % (згідно з отриманим Recall = 96,7 % у розділі 3).

Консервативна оцінка запобігання збиткам – 300000 грн/рік.

Таблиця 4.5. Економія ФОП

Посада	Кількість ставок до	Кількість ставок після	Економія ставок	Середня з/п бруто, грн/міс	Річна економія (з ЄСВ), грн
Охоронець	3,0	1,0	2,0	14500	348000
Комендант гуртожитку	2,0	1,0	1,0	13800	165600
Працівник деканату (0,5 ставки)	0,5	0	0,5	12000	72000
Разом річна економія ФОП					585600

4.4.3. Додатковий ефект від автоматизації

Економія часу викладачів та лаборантів на ручному обліку – 0,5 год/день × 220 роб. днів × 25 чол. × 250 грн/год = 687500 грн/рік (можна частково враховувати). Консервативно враховуємо лише 80 000 грн/рік.

Загальний річний економічний ефект: 585600 (ФОП) + 300000 (запобігання збиткам) + 80000 (автоматизація) = 965600 грн/рік.

Показники ефективності інвестицій (ставка дисконту 19% – середня по Україні 2025 р.) наведено в табл. 4.6.

Як висновок можна зазначити, що проєкт є надзвичайно високорентабельним, термін окупності не перевищує 2 місяців навіть за песимістичним сценарієм.

Таблиця 4.6. Основні показники економічної ефективності

Показник	Формула / значення	Результат
Чистий дисконтований дохід (NPV)	$\sum_{t=1}^n \frac{CF_t}{(1+r)^t} - C_0$	+2847000 грн (за 5 років)
Індекс рентабельності (PI)	$\sum_{t=1}^n \frac{CF_t}{(1+r)^t} / C_0$	9,8
Внутрішня норма дохідності (IRR)	r , при якому $NPV = 0$	892 %
Простий термін окупності	C_0 / чистий річний ефект	107050 / 965600 $\approx$ 40 днів
Дисконтований термін окупності	з урахуванням дисконту	48 днів

4.5. Висновки до розділу 4

Проведене техніко-економічне обґрунтування підтвердило високу економічну доцільність і інвестиційну привабливість розробленої інтелектуальної системи контролю доступу на базі RFID-технологій та методів машинного навчання при її впровадженні як у закладах вищої освіти, так і в корпоративному секторі середнього масштабу.

Повна собівартість створення діючого прототипу системи з десяти контрольованих точок доступу, включаючи апаратне забезпечення, розробку програмного забезпечення та оплату праці розробника за ринковими ставками 2025 року, становить 183000 грн, з яких лише 17000 грн – матеріальні витрати на апаратну частину.

Капітальні витрати на повномасштабне впровадження системи на 10 точках доступу в умовах університету, з урахуванням монтажу, пусконаладження, виготовлення та видачі 1500 RFID-карток, навчання персоналу та придбання серверного обладнання, складають 107050 грн.

Річні експлуатаційні витрати системи є мінімальними і не перевищують 26610 грн, що пояснюється високою енергоефективністю компонентів ESP32,

використанням безкоштовного ПЗ з відкритим кодом та низькими вимогами до технічного обслуговування.

Річний економічний ефект від впровадження системи оцінено на рівні 965600 грн і формується за рахунок:

- прямої економії фонду оплати праці (585600 грн) шляхом скорочення 3,5 ставок охорони та адміністративного персоналу;
- запобігання матеріальним збиткам від несанкціонованого доступу та інсайдерських загроз (300000 грн);
- автоматизації обліку відвідувань та оптимізації адміністративних процесів (80000 грн).

Простий термін окупності капітальних вкладень становить лише 40 діб, дисконтований термін окупності (за ставкою дисконту 19%) – 48 діб. Індекс рентабельності інвестицій (PI) – 9,8, внутрішня норма дохідності (IRR) перевищує 890%, чистий дисконтований дохід (NPV) за 5-річним горизонтом – понад 2,8 млн грн.

Навіть у песимістичному сценарії (зниження ефекту на 50%) термін окупності не перевищує 4 місяців, що підтверджує надзвичайно низький інвестиційний ризик проєкту.

Таким чином, розроблена система не лише повністю окупає всі витрати в найкоротші терміни, але й генерує значний чистий прибуток вже з другого місяця експлуатації. Низька собівартість однієї точки доступу (близько 1380 грн апаратних складових) та можливість масштабування роблять систему привабливою для впровадження в закладах освіти чи комерційних організаціях.

Отримані показники економічної ефективності значно перевищують середньогалузеві значення для проєктів впровадження систем фізичної безпеки та IoT-рішень і свідчать про готовність розробки до практичного розгортання, в тому числі на госпрозрахунковій основі або за рахунок грантового фінансування.

РОЗДІЛ 5

БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

5.1. Загальні положення

Розроблена інтелектуальна система контролю доступу з використанням RFID-карт та методів машинного навчання належить до програмно-апаратних комплексів, що експлуатуються в закритих приміщеннях закладів освіти та корпоративних установ. Згідно з ДСТУ EN 60204-1:2017 «Електрообладнання машин і механізмів. Загальні технічні вимоги» та НПАОП 0.00-1.83-18 «Правила охорони праці під час експлуатації електронно-обчислювальних машин», система відноситься до I групи виробництв за санітарною класифікацією (робота переважно сидяча, з використанням комп'ютерної техніки та електронного обладнання напругою до 42 В).

Основні нормативно-правові акти, що застосовуються при аналізі безпеки життєдіяльності:

- Кодекс цивільного захисту України від 02.10.2012 № 5403-VI;
- Закон України «Про охорону праці» від 14.10.1993 № 2694-XII (в редакції 2024 р.);
- ДБН В.2.5-56:2014 «Системи протипожежного захисту»;
- НАПБ А.01.001-2014 «Правила пожежної безпеки в Україні»;
- ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами»;
- ГОСТ 12.2.032-78 «Робоче місце при виконанні робіт сидячи. Загальні ергономічні вимоги».

Об'єктами аналізу небезпечних і шкідливих виробничих факторів є:

- апаратні вузли на базі мікроконтролера ESP32 та зчитувача MFRC522 (живлення 5 В);
- серверна частина (локальний сервер або VPS);
- робоче місце адміністратора системи (монітор, клавіатура, миша);
- процес монтажу та технічного обслуговування точок доступу.

Персонал, що взаємодіє з системою:

- адміністратор безпеки (робота за ПК ≥ 4 год/добу);
- черговий служби охорони (контроль сповіщень);
- монтажник/налагоджувальник (під час встановлення та обслуговування).

5.2. Аналіз небезпечних і шкідливих виробничих факторів при експлуатації розробленої системи

Згідно з ГОСТ 12.0.003-2015 «Небезпечні та шкідливі виробничі фактори. Класифікація», при нормальній експлуатації системи виявлено такі потенційні небезпечні та шкідливі фактори (табл. 5.1):

- фактори електричного характеру;
- електромагнітні поля та випромінювання;
- пожежна безпека;
- ергономічні та психофізіологічні фактори (для адміністратора системи);
- фактори, пов'язані з монтажем та обслуговуванням.

Фактори електричного характеру: можливість ураження електричним струмом при пошкодженні ізоляції блоку живлення $220\text{ В} \rightarrow 5\text{ В}$ або при монтажі (особливо в приміщеннях з підвищеною вологістю – гуртожитки); клас захисту апаратних вузлів – II (подвійна ізоляція), але при використанні неякісних блоків живлення існує ризик пробоя; рівень небезпеки – середній (ймовірність низька, тяжкість наслідків – висока).

Електромагнітні поля та випромінювання: радіочастотне випромінювання модуля Wi-Fi ESP32 (діапазон 2,4 ГГц, потужність до 20 дБм); рівень ЕМП на відстані 0,2 м – не перевищує 3 мкВт/см^2 (норма за ДСанПіН – 10 мкВт/см^2); випромінювання RFID-зчитувача MFRC522 (13,56 МГц) – неіонізуюче, потужність $< 100\text{ мВт}$, безпечне.

Пожежна безпека: при короткому замиканні в блоці живлення або перегріві літій-іонного акумулятора резервного живлення (застосовується в

деяких модифікаціях) можливе займання; клас пожежної небезпеки за НПАОП 40.1-1.32-01 – П-Па (тверді горючі матеріали, пластмаса корпусу).

Ергономічні та психофізіологічні фактори (для адміністратора системи): тривале статичне навантаження (робота сидячи > 4 год); напруга органу зору через тривалу роботу з монітором; монотонність праці при обробці сповіщень про аномалії; психоемоційне навантаження при реагуванні на інциденти безпеки.

Фактори, пов'язані з монтажем та обслуговуванням: робота на висоті (до 1,5 м) при встановленні точок доступу над дверима; ризик падіння інструменту, ураження дрібними частинками при свердлінні стін; вплив шуму від перфоратора (до 95 дБА).

Висновок за результатами аналізу: при дотриманні розроблених заходів усі виявлені фактори зводяться до допустимого рівня або повністю усуваються.

Таблиця 5.1. Класифікація виявлених небезпечних і шкідливих факторів

Фактор	Джерело	Класифікація за ГОСТ 12.0.003-2015	Рівень ризику
Ураження електричним струмом	Блок живлення 220→5 В	Фізичний	Середній
Електромагнітне випромінювання	Wi-Fi модуль ESP32	Фізичний	Низький
Пожежа	Коротке замикання, перегрів	Фізичний	Середній
Зорова втома, статичне навантаження	Робота за ПК адміністратора	Психофізіологічний	Середній
Шум, вібрація, піл	Монтажні роботи	Фізичний	Високий (короткочасний)

5.3. Заходи щодо забезпечення електробезпеки, пожежної безпеки та ергономіки робочих місць

На підставі аналізу (п. 5.2) розроблено та впроваджено комплекс організаційно-технічних заходів, що повністю усувають або зводять до мінімально допустимого рівня всі виявлені небезпечні та шкідливі фактори.

5.3.1. Заходи з електробезпеки

Усі апаратні вузли живляться напругою 5 В (безпечна напруга за ГОСТ 12.2.007.0-75). Використано сертифіковані імпульсні блоки живлення класу захисту II (подвійна або посилена ізоляція) виробництва Mean Well серії LRS-50-5 з вбудованим захистом від короткого замикання, перевантаження та перегріву. Корпуси точок доступу виготовлені з негорючого пластику ABS UL94-V0, повністю ізольовані від струмопровідних частин. Передбачено автоматичне відключення живлення релейного модуля при виявленні несправності через watchdog-таймер ESP32. У приміщеннях з підвищеною вологістю (гуртожитки) застосовано додатковий герметичний бокс IP65. Усі з'єднання виконані пайкою оловом Sn99Cu1 (без свинцю) з використанням флюсу без галогенів.

5.3.2. Заходи з пожежної безпеки

Потужність споживання одного вузла – не більше 3 Вт, тепловиділення мінімальне. Встановлено тепловий запобіжник 85 °C на лінії живлення кожного вузла. Використано самозгасальний пластик корпусу та плати з покриттям solder mask. Передбачено автоматичне відключення реле при температурі мікроконтролера > 75 °C (вимірюється вбудованим датчиком ESP32). Розміщення вузлів – не ближче 0,5 м від опалювальних приладів та прямих сонячних променів. У серверній шафі передбачено встановлення автоматичного вогнегасника на основі Noves 1230 або порошкового типу.

5.3.3. Ергономічні заходи та санітарно-гігієнічні вимоги до робочого місця адміністратора

Робоче місце організовано відповідно до ДСанПіН 3.3.2.007-98:

- відстань від очей до екрану – 600–700 мм;
- кут нахилу екрану – 10–20°;
- висота крісла регулюється, спинка має поперекову підтримку.

Рівень освітленості робочої поверхні – 500 лк (комбіноване освітлення).

Регламентовані перерви: 10 хв кожні 45 хв роботи за монітором.

Веб-інтерфейс виконано з контрастом не менше 7:1, шрифт не менше 14 pt, з можливістю темної теми (зниження синього світла).

Передбачено автоматичне сповіщення адміністратора звуковим сигналом та push-повідомленнями лише при критичних аномаліях (зменшення психоемоційного навантаження).

5.4. Розроблення інструкції з охорони праці для адміністратора та оператора системи

Затверджено: «___» _____ 2025 р.

Керівник _____

ІНСТРУКЦІЯ З ОХОРОНИ ПРАЦІ № _____

для персоналу інтелектуальної системи контролю доступу

Загальні вимоги безпеки

1.1. До роботи допускаються особи, які пройшли інструктаж, медогляд та ознайомлені з даною інструкцією.

1.2. Під час роботи забороняється вживання алкоголю, наркотичних речовин, паління в приміщеннях.

Вимоги безпеки перед початком роботи

2.1. Перевірити зовнішній стан обладнання: відсутність пошкоджень корпусів, кабелів, запаху горілої ізоляції.

2.2. Переконавшись, що блоки живлення підключені до розеток з захисним заземленням (для серверного ПК).

2.3. Увімкнути систему в послідовності: сервер → точки доступу.

Вимоги безпеки під час роботи

3.1. Забороняється торкатися оголених контактів, відкривати корпуси під напругою.

3.2. При виявленні іскріння, диму, запаху горілого негайно вимкнути живлення автоматом у щиті та повідомити керівництво.

3.3. Робота за комп'ютером – не більше 6 годин на добу з регламентованими перервами.

3.4. При монтажі/демонтажі точок доступу обов'язкове використання діелектричних рукавичок та інструменту з ізольованими ручками.

Вимоги безпеки в аварійних ситуаціях

4.1. При ураженні електричним струмом: вимкнути живлення, надати першу допомогу, викликати швидку.

4.2. При пожежі: натиснути кнопку пожежної тривоги, використовувати первинні засоби пожежогасіння (порошковий або вуглекислотний вогнегасник).

Вимоги безпеки після закінчення роботи

5.1. Вимкнути сервер та точки доступу (якщо не передбачена цілодобова робота).

5.2. Провести вологе прибирання робочого місця.

Інструктаж пройшов _____ П.І.Б.

Дата _____ Підпис _____

5.5. Висновки до розділу 5

Проведений аналіз небезпечних та шкідливих виробничих факторів показав, що розроблена інтелектуальна система контролю доступу при нормальній експлуатації належить до безпечних технічних об'єктів (клас II за рівнем безпеки).

Запропонований комплекс організаційно-технічних заходів (подвійна ізоляція, використання сертифікованих блоків живлення, самозгасальні матеріали, ергономічне робоче місце, детальна інструкція з охорони праці) повністю усуває або зводить до прийнятного рівня всі виявлені ризики.

Рівень електромагнітного випромінювання, шуму, тепловиділення, освітленості та інших параметрів відповідає чинним санітарним нормам України (ДСанПіН 3.3.2.007-98, ДБН В.2.5-56:2014, НПАОП 0.00-1.83-18).

Отже, експлуатація системи в умовах закладів освіти та інших закладів не становить небезпеки для життя і здоров'я персоналу за умови дотримання розроблених правил і інструкцій.

РОЗДІЛ 6
ЕКОЛОГІЧНА БЕЗПЕКА

6.1. Загальна характеристика впливу системи на навколишнє середовище

Розроблена інтелектуальна система контролю доступу з використанням RFID-карт та методів машинного навчання належить до електронного та електротехнічного обладнання малої потужності й за класифікацією Закону України «Про охорону навколишнього природного середовища» № 1264-XII від 25.06.1991 (в редакції 2024 р.) та ДСТУ 3017-2015 відноситься до V класу небезпеки – практично нешкідливі об’єкти (табл. 6.1).

Споживання електроенергії однією точкою доступу в активному режимі становить 2,5-3 Вт, у режимі глибокого сну (Deep Sleep) – менше 10 мкА. Завдяки цьому річний викид CO₂ від роботи системи (за українським енергетичним міксом 2025 р. – 0,42 кг CO₂/кВт·год) не перевищує 4,6 кг CO₂-екв, що еквівалентно викидам одного бензинового автомобіля за 25 км пробігу.

Таблиця 6.1. Основні екологічні аспекти життєвого циклу системи

Етап життєвого циклу	Характер впливу	Кількісна оцінка
Виробництво компонентів	Використання олова, міді, кремнію, пластику	Всі компоненти RoHS-2015/863-сумісні (без свинцю, кадмію, ртуті)
Експлуатація (10 точок)	Споживання електроенергії	≤ 30 Вт·год/добу або ≈ 11 кВт·год/рік
	Електромагнітне випромінювання	< 3 мкВт/см ² (норма ≤ 10 мкВт/см ²)
Закінчення терміну служби	Утворення електронних відходів	≈ 2,5 кг на 10 вузлів (через 8–10 років)

6.2. Заходи щодо зменшення негативного екологічного впливу

Для мінімізації екологічного сліду системи реалізовано комплекс превентивних заходів.

Енергоефективність: використання мікроконтролера ESP32 з підтримкою режимів Light Sleep та Deep Sleep (зниження споживання на 98 % у періоди бездіяльності); програмне відключення RFID-зчитувача та Wi-Fi на час відсутності активності (таймер 10 с); застосування імпульсних джерел живлення з ККД ≥ 92 % (Mean Well LRS-50-5).

Екологічно безпечні матеріали: усі плати та компоненти відповідають Директиві RoHS 2015/863/EU та національному Технічному регламенту обмеження використання деяких небезпечних речовин (постанова КМУ № 139 від 10.03.2017). Корпуси виготовлені з самозгасального пластику ABS UL94-V0, придатного до вторинної переробки.

Утилізація та повторне використання: RFID-картки Mifare Classic 1K (PBX) можуть перепрограмовуватись необмежену кількість разів – термін служби до 10 років. Передбачено маркування виробів знаком «зелена точка» та рекомендації щодо здачі відпрацьованого обладнання спеціалізованим підприємствам. Електронні плати після закінчення терміну служби передаються на підприємства з ліцензією на поводження з небезпечними відходами (код відходу за ДК 021:2015 – 16.10.10.0-00 «Відходи електронного обладнання»).

Зменшення транспортного навантаження: оновлення прошивки та моделей ML виконується по OTA (Over-the-Air), що усуває необхідність фізичних виїздів для більшості оновлень.

Нормативно-правове забезпечення: дотримано вимог Закону України «Про відходи як вторинну сировину» № 2875-IX від 16.06.2022. Система може бути включена до реєстру об'єктів утворення відходів (форма № 1-ВТ) як об'єкт з мінімальним впливом.

6.3. Висновки до розділу 6

Розроблена інтелектуальна система контролю доступу має мінімальний негативний вплив на навколишнє природне середовище протягом усього життєвого циклу. Споживання електроенергії, електромагнітне випромінювання

та обсяг відходів значно нижчі за гранично допустимі норми, встановлені чинним законодавством України та міжнародними стандартами.

Запропоновані заходи (енергоефективні режими роботи, використання RoHS-сумісних компонентів, організація збору та утилізації відпрацьованого обладнання) забезпечують відповідність системи принципам сталого розвитку та концепції «зелених технологій».

Таким чином, впровадження системи в закладах освіти або корпоративному секторі середнього масштабу не призведе до погіршення стану навколишнього середовища і повністю відповідає вимогам Закону України «Про охорону навколишнього природного середовища», Директиві RoHS та національним технічним регламентам.

ВИСНОВКИ

У кваліфікаційній роботі проведено комплексне дослідження та практичну реалізацію рішення для підвищення рівня фізичної безпеки в динамічних середовищах. Метою роботи – розробити інтелектуальну систему контролю доступу на основі RFID-технологій з інтеграцією методів машинного навчання для виявлення аномалій у реальному часі – досягнуто в повному обсязі. Поставлені завдання розв’язано послідовно, що дозволило створити повноцінний прототип системи, який поєднує апаратний, програмний та аналітичний рівні, забезпечуючи адаптивність, точність і зручність експлуатації.

У першому розділі проведено ґрунтовний аналіз предметної області, сучасних систем контролю доступу та методів виявлення аномалій. Встановлено, що традиційні RFID-системи, попри швидкість аутентифікації (менше 100 мс), вразливі до атак (eavesdropping, cloning, spoofing), а статичні моделі управління (DAC, MAC, RBAC) не враховують динамічні поведінкові патерни. Проаналізовано методи машинного навчання без вчителя, або неконтрольованого навчання (Isolation Forest, One-Class SVM, DBSCAN, Autoencoder, LSTM Autoencoder) на синтетичних даних, де Autoencoder продемонстрував найкращі показники ($F1\text{-score} \approx 0,91$, $ROC\text{-AUC} = 0,96$). Це обґрунтувало вибір гібридного підходу для детекції аномалій, що поєднує детерміновані правила з нейронною моделлю, забезпечуючи баланс між швидкістю та точністю.

Другий розділ присвячено проєктуванню системи. Сформовано функціональні (FR-01–FR-10) та нефункціональні (NFR-01–NFR-08) вимоги, розроблено концептуальну модель з діаграмами потоків даних (DFD Level 0–1), архітектуру на базі MVC з трьома логічними рівнями (представлення, застосування, дані) та апаратним вузлом. Базу даних нормалізовано до 3NF з індексацією для швидких запитів, а веб-інтерфейс спроектовано з урахуванням зручності та простоти використання продукту (Bootstrap, Chart.js, AJAX-оновлення кожні 10 с). Гібридний детектор аномалій реалізовано з пріоритетом

правил (нічний доступ, часті сканування тощо), доповнених ML для колективних відхилень, що підвищує F1-score до 0,95.

У третьому розділі здійснено реалізацію та експериментальне дослідження. Апаратний рівень на ESP32 + MFRC522 забезпечує зчитування UID з дебаунсом, WiFi-передачу (HTTP POST) та індикацію (LED, buzzer) з затримкою <200 мс. Серверна частина на Flask інтегрує REST API (9 ендпоінтів), модуль безпеки (PBKDF2-SHA256 хешування, валідація) та MLInterface з fallback-режимом. Синтетичний датасет (15847 подій, 3% аномалій) використано для тренування Autoencoder (архітектура $12 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 12$, threshold = 0,087). Експерименти підтвердили ефективність моделі: Precision = 91,2%, Recall = 96,7%, F1-score = 93,8%, AUC-ROC = 0,94, середня затримка обробки події – 300-700 мс при навантаженні 50 подій/хв. Веб-інтерфейс з дашбордом забезпечує моніторинг, CRUD-операції та експорт даних, а інструкція користувача охоплює розгортання, реєстрацію та обробку аномалій.

Техніко-економічне обґрунтування показало, що капітальні витрати на впровадження системи на 10 точок доступу складають 107050 грн, річний економічний ефект – 965600 грн, а простий термін окупності – 40 діб, NPV за 5 років – понад 2,8 млн грн.

Розроблено та впроваджено комплекс заходів з охорони праці та безпеки життєдіяльності, що повністю усуває ризики ураження струмом, пожежі та професійних захворювань.

Проведено оцінку екологічного впливу. Система належить до V класу небезпеки, споживання електроенергії ≤ 11 кВт·год/рік на 10 точок, всі компоненти RoHS-сумісні, передбачено організований збір та утилізацію відпрацьованого обладнання.

Наукова новизна роботи полягає в гібридній архітектурі, адаптованому синтетичному датасеті для університетського середовища та інтеграції ML у IoT-вузол, що зменшує хибні спрацьовування на 40% порівняно з базовими моделями. Практичне значення – у створенні низьковартісного рішення для автоматизації безпеки, придатного для масштабування в освітніх установах, з

потенціалом інтеграції з SIEM-системами та відеоспостереженням. Методи дослідження (системний аналіз, експериментальне моделювання, інженерна розробка) та інструментальні засоби (ESP32, Flask, Keras, Python 3.13) забезпечили відтворюваність і обґрунтованість результатів.

Перспективи подальших досліджень включають перехід до рекурентних мереж (LSTM) для аналізу довгих послідовностей, інтеграцію з біометрією (розпізнавання облич), розгортання на хмарних платформах (AWS IoT) для мультикампусних систем, оцінку на реальних даних з урахуванням етичних аспектів приватності. Отримані результати можуть стати основою для комерціалізації системи та публікацій у фахових виданнях, сприяючи розвитку кібербезпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gartner. Predicts 2024: The Changing Role of the Identity and Access Management Leader [Електронний ресурс]. Gartner Research, 2023. Режим доступу: <https://www.gartner.com/en/newsroom/press-releases/2024-02-01-gartner-predicts-30-percent-of-enterprises-will-consider-identity-verification-and-authentication-solutions-unreliable-in-isolation-due-to-deepfakes-by-2026> (дата звернення: 14.11.2025).
2. Muñoz-Ausecha C., Ruiz-Rosero J., Ramírez-González G. RFID Applications and Security Review. Computation. 2021. Vol. 9, No. 6. P. 69. DOI: 10.3390/computation9060069.
3. Shi X., Cao J., Lu T., Chang V. A Survey on RFID Security and Privacy in Smart Medical: Threats and Protections. Proc. 4th Int. Conf. on Internet of Things, Big Data and Security, 2019. Режим доступу: <https://www.scitepress.org/Papers/2019/77134/77134.pdf> (дата звернення: 14.11.2025).
4. Golightly L. та ін. Securing Distributed Systems: A Survey on Access Control Techniques and Trends. Journal of Information Security and Applications. 2023. Режим доступу: <https://www.sciencedirect.com/science/article/pii/S2772918423000036> (дата звернення: 14.11.2025).
5. Zhang Y. та ін. Attribute-Based Encryption for Cloud Computing Access Control: A Survey. 2020. Режим доступу: <https://smusg.elsevierpure.com/en/publications/attribute-based-encryption-for-cloud-computing-access-control-a-s> (дата звернення: 14.11.2025).
6. Espressif Systems. ESP32 Series Datasheet [Електронний ресурс]. 2023. Режим доступу: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf (дата звернення: 14.11.2025).
7. NXP Semiconductors. MFRC522 Standard Performance MIFARE and NTAG Frontend [Електронний ресурс]. Режим доступу:

- <https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf> (дата звернення: 14.11.2025).
8. ISO/IEC 14443-1:2018. Identification cards — Contactless integrated circuit cards — Proximity cards — Part 1: Physical characteristics. ISO, 2018.
 9. Alizadeh M. та ін. A Survey on Attacks in RFID Networks. 2021. Режим доступу: <https://oiji.utm.my/index.php/oiji/article/download/52/29> (дата звернення: 14.11.2025).
 10. Landauer M. та ін. Deep Learning for Anomaly Detection in Log Data: A Survey. 2022. Режим доступу: <https://arxiv.org/pdf/2207.03820.pdf> (дата звернення: 14.11.2025).
 11. Chen S., Li L. Unsupervised Anomaly Detection for IoT Cybersecurity. IEEE Internet of Things Journal. 2023.
 12. Breunig M. M., Kriegel H.-P., Ng R. T., Sander J. LOF: Identifying Density-Based Local Outliers. ACM SIGMOD Record. 2000. Vol. 29, No. 2. P. 93–104. DOI: 10.1145/335191.335388.
 13. Goldstein M., Uchida S. A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data. PLOS ONE. 2016. Vol. 11, No. 4. e0152173. DOI: 10.1371/journal.pone.0152173.
 14. Liu F. T., Ting K. M., Zhou Z.-H. Isolation Forest. Proc. 2008 IEEE International Conference on Data Mining. 2008. P. 413–422. DOI: 10.1109/ICDM.2008.17.
 15. Sakurada M., Yairi T. Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction. Proc. MLSDA 2014. P. 1–8. DOI: 10.1145/2689746.2689747.
 16. Müller H. та ін. Anomaly Detection in Physical Access Control Systems Using Machine Learning. Proc. 2022 Int. Conf. on Cyber Security and Protection of Digital Services. 2022.
 17. Splunk. Splunk Machine Learning Toolkit [Електронний ресурс]. Режим доступу: https://www.splunk.com/en_us/products/machine-learning-toolkit.html (дата звернення: 15.11.2025).

18. Kim J. та ін. LSTM-based Anomaly Detection for Smart Home Security. *IEEE Transactions on Dependable and Secure Computing*. 2022. Vol. 19, No. 5. P. 3456–3468. DOI: 10.1109/TDSC.2021.3094275.
19. Tax D. M. J., Duin R. P. W. Support Vector Data Description. *Machine Learning*. 2004. Vol. 54, No. 1. P. 45–66. DOI: 10.1023/B:MACH.0000008084.60811.49.
20. Ester M., Kriegel H.-P., Sander J., Xu X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proc. KDD'96*. 1996. P. 226–231.
21. Kingma D. P., Welling M. Auto-Encoding Variational Bayes. *arXiv:1312.6114*. 2013. Режим доступу: <https://arxiv.org/abs/1312.6114> (дата звернення: 15.11.2025).
22. Hochreiter S., Schmidhuber J. Long Short-Term Memory. *Neural Computation*. 1997. Vol. 9, No. 8. P. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.
23. Li S., Wang X. A Hybrid Machine Learning Approach for Anomaly Detection in Industrial IoT. *IEEE Transactions on Industrial Informatics*. 2023. Vol. 19, No. 9. P. 10045–10055.
24. Arduino. Arduino Reference [Електронний ресурс]. 2024. Режим доступу: <https://www.arduino.cc/reference/en/> (дата звернення: 15.11.2025).
25. Balboa M. MFRC522 Library for Arduino [Електронний ресурс]. 2023. Режим доступу: <https://github.com/miguelbalboa/rfid> (дата звернення: 15.11.2025).
26. Pandas Development Team. pandas: powerful Python data analysis toolkit [Електронний ресурс]. Режим доступу: <https://pandas.pydata.org/> (дата звернення: 15.11.2025).
27. Scikit-learn Developers. Scikit-learn: Machine Learning in Python [Електронний ресурс]. Режим доступу: <https://scikit-learn.org/stable/> (дата звернення: 15.11.2025).
28. TensorFlow Development Team. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Електронний ресурс]. Режим доступу: <https://www.tensorflow.org/> (дата звернення: 15.11.2025).

- 29.OMG. UML Specification — Unified Modeling Language [Электронный ресурс]. 2025. Режим доступа: <https://www.omg.org/spec/UML/> (дата звернения: 15.11.2025).
- 30.Bootstrap Team. Bootstrap Documentation 5.3 [Электронный ресурс]. 2025. Режим доступа: <https://getbootstrap.com/docs/5.3/> (дата звернения: 15.11.2025).
- 31.Flask Development Team. Flask Documentation (3.0.x) [Электронный ресурс]. 2025. Режим доступа: <https://flask.palletsprojects.com/en/3.0.x/> (дата звернения: 15.11.2025).
- 32.SQLite Consortium. SQLite Documentation [Электронный ресурс]. 2025. Режим доступа: <https://www.sqlite.org/docs.html> (дата звернения: 15.11.2025).
- 33.Chart.js Team. Chart.js Documentation [Электронный ресурс]. 2025. Режим доступа: <https://www.chartjs.org/docs/latest/> (дата звернения: 15.11.2025).
- 34.ArduinoJson Library. ArduinoJson Documentation [Электронный ресурс]. 2025. Режим доступа: <https://arduinojson.org/> (дата звернения: 15.11.2025).

ДОДАТКИ

Додаток А

Код мікроконтролера ESP32 (Arduino)

Файл: arduino_rfid.txt

Мова: C++ (Arduino IDE)

Бібліотеки: MFRC522, WiFi, HTTPClient, ArduinoJson

Опис: реалізація зчитування RFID, дебаунсу, відправки на сервер.

```
/*
 * RFID Access Control - Arduino Reader
 *
 * Підключення RFID-RC522:
 * SDA --> Pin 21 (SS)
 * SCK --> Pin 18
 * MOSI --> Pin 23
 * MISO --> Pin 19
 * IRQ --> (не підключено)
 * GND --> GND
 * RST --> Pin 22
 * 3.3V --> 3.3V
 *
 * Додатково:
 * LED (зелений) --> Pin 2
 * LED (червоний) --> Pin 4
 * Buzzer --> Pin 15
 */

#include <SPI.h>
#include <MFRC522.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

// Пини для RFID
#define RST_PIN 22
#define SS_PIN 21

// Пини для індикації
#define LED_GREEN 2
#define LED_RED 4
#define BUZZER 15

// WiFi credentials (ЗАМІНІТЬ НА СВОЇ!)
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// Flask server (ЗАМІНІТЬ НА IP ВАШОГО КОМП'ЮТЕРА!)
const char* serverUrl = "http://192.168.1.100:5000/api/log_event";
```

```

const char* accessPoint = "Main Entrance"; // Назва точки доступу

// RFID reader
MFRC522 mfrc522(SS_PIN, RST_PIN);

// Змінні для дебаунсу
unsigned long lastReadTime = 0;
const unsigned long DEBOUNCE_DELAY = 2000; // 2 секунди між зчитуваннями

void setup() {
  Serial.begin(115200);
  delay(100);

  Serial.println("\n=====");
  Serial.println("    RFID Access Control System - Arduino");
  Serial.println("=====\\n");

  // Налаштування пінів
  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_RED, OUTPUT);
  pinMode(BUZZER, OUTPUT);

  // Початкова індикація
  digitalWrite(LED_GREEN, LOW);
  digitalWrite(LED_RED, LOW);

  // Ініціалізація SPI та RFID
  SPI.begin();
  mfrc522.PCD_Init();
  delay(100);

  Serial.println("✓ RFID reader initialized");
  Serial.print("  Reader version: ");
  mfrc522.PCD_DumpVersionToSerial();

  // Підключення до WiFi
  Serial.println("\\n⌘ Connecting to WiFi...");
  WiFi.begin(ssid, password);

  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 20) {
    delay(500);
    Serial.print(".");
    attempts++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\\n✓ WiFi connected!");
    Serial.print("  IP address: ");
    Serial.println(WiFi.localIP());
    Serial.print("  Server URL: ");

```

```

Serial.println(serverUrl);

// Зелена індикація успішного підключення
blinkLED(LED_GREEN, 3);
} else {
  Serial.println("\nX WiFi connection failed!");
  Serial.println("  System will work offline (logs only to Serial)");

  // Червона індикація помилки
  blinkLED(LED_RED, 3);
}

Serial.println("\n=====");
Serial.println("          System Ready - Scan RFID Card");
Serial.println("=====\\n");
}

void loop() {
  // Перевірка наявності нової картки
  if (!mfrc522.PICC_IsNewCardPresent()) {
    delay(50);
    return;
  }

  if (!mfrc522.PICC_ReadCardSerial()) {
    delay(50);
    return;
  }

  // Дебаунс - уникнення повторного зчитування
  unsigned long currentTime = millis();
  if (currentTime - lastReadTime < DEBOUNCE_DELAY) {
    mfrc522.PICC_HaltA();
    return;
  }
  lastReadTime = currentTime;

  // Зчитування UID картки
  String rfid_uid = "";
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    if (mfrc522.uid.uidByte[i] < 0x10) {
      rfid_uid += "0";
    }
    rfid_uid += String(mfrc522.uid.uidByte[i], HEX);
  }
  rfid_uid.toUpperCase();

  Serial.println("\n-----");
  Serial.println("■ RFID Card Detected!");
  Serial.print("  UID: ");
  Serial.println(rfid_uid);
}

```



```

Serial.print("    Access Point: ");
Serial.println(accessPoint);
Serial.print("    Time: ");
Serial.println(getTimestamp());

// Відправка даних на сервер
bool success = sendToServer(rfid_uid);

if (success) {
    // Успішна відправка
    Serial.println("✓ Data sent successfully!");
    indicateSuccess();
} else {
    // Помилка відправки
    Serial.println("X Failed to send data");
    indicateError();
}

Serial.println("-----\n");

// Зупинка зчитування поточної картки
mfrc522.PICC_HaltA();
mfrc522.PCD_StopCrypto1();
}

bool sendToServer(String rfid_uid) {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("⚠️ WiFi not connected - skipping upload");
        return false;
    }
}

HTTPClient http;
http.begin(serverUrl);
http.addHeader("Content-Type", "application/json");
http.setTimeout(5000); // 5 секунд таймаут

// Створення JSON payload
StaticJsonDocument<200> doc;
doc["rfid_uid"] = rfid_uid;
doc["access_point"] = accessPoint;
// timestamp не передаємо - сервер сам додасть поточний час

String jsonPayload;
serializeJson(doc, jsonPayload);

Serial.print("    📡 Sending: ");
Serial.println(jsonPayload);

// Відправка POST запиту
int httpCode = http.POST(jsonPayload);

```

```

if (httpCode > 0) {
    Serial.print("    ⬇ Response code: ");
    Serial.println(httpCode);

    if (httpCode == 200) {
        String response = http.getString();
        Serial.print("    ■ Response: ");
        Serial.println(response);

        // Парсинг відповіді для перевірки аномалій
        StaticJsonDocument<512> responseDoc;
        DeserializationError error = deserializeJson(responseDoc,
response);

        if (!error) {
            bool isAnomaly = responseDoc["anomaly"]["is_anomaly"];
            const char* anomalyType = responseDoc["anomaly"]["anomaly_type"];
            float confidence = responseDoc["anomaly"]["confidence_score"];

            if (isAnomaly) {
                Serial.println("    ⚠ ANOMALY DETECTED!");
                Serial.print("        Type: ");
                Serial.println(anomalyType);
                Serial.print("        Confidence: ");
                Serial.print(confidence * 100);
                Serial.println("%");
            }
        }

        http.end();
        return true;
    }
} else {
    Serial.print("    X HTTP Error: ");
    Serial.println(http.errorToString(httpCode).c_str());
}

http.end();
return false;
}

String getTimestamp() {
    // Простий timestamp для логів
    unsigned long seconds = millis() / 1000;
    unsigned long minutes = seconds / 60;
    unsigned long hours = minutes / 60;

    seconds = seconds % 60;
    minutes = minutes % 60;
    hours = hours % 24;
}

```

```

    char buffer[20];
    sprintf(buffer, "%02lu:%02lu:%02lu", hours, minutes, seconds);
    return String(buffer);
}

void indicateSuccess() {
    // Зелена світлодіод + коротки сигнал
    digitalWrite(LED_GREEN, HIGH);
    tone(BUZZER, 1000, 200);
    delay(500);
    digitalWrite(LED_GREEN, LOW);
}

void indicateError() {
    // Червона світлодіод + довгий сигнал
    digitalWrite(LED_RED, HIGH);
    tone(BUZZER, 500, 500);
    delay(800);
    digitalWrite(LED_RED, LOW);
}

void blinkLED(int pin, int times) {
    for (int i = 0; i < times; i++) {
        digitalWrite(pin, HIGH);
        delay(200);
        digitalWrite(pin, LOW);
        delay(200);
    }
}

```

Основний серверний модуль (Flask)

Файл: app.py

Мова: Python 3

Бібліотеки: Flask, sqlite3, werkzeug, pickle, json

Опис: центральний модуль веб-додатку: маршрути API, обробка RFID-подій, інтеграція з ML, авторизація, експорт логів, ініціалізація БД.

```
from flask import Flask, render_template, request, jsonify, redirect,
url_for, session, send_file
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
import sqlite3
import json
import os
import io
import csv
from utils.db_helper import DatabaseHelper
from utils.ml_interface import MLInterface
from utils.security import SecurityHelper

app = Flask(__name__)
app.secret_key = 'rfid-access-control-secret-key-2024'
app.config['DATABASE'] = 'database/access_control.db'

db_helper = DatabaseHelper(app.config['DATABASE'])
ml_interface = MLInterface()
security_helper = SecurityHelper()

def login_required(f):
    def decorated_function(*args, **kwargs):
        if 'admin_id' not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function

@app.route('/')
def index():
    return redirect(url_for('login'))

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        conn = sqlite3.connect(app.config['DATABASE'])
        admin = conn.execute('SELECT * FROM admin WHERE username = ?',
                               (username,)).fetchone()
```

```

        conn.close()
        if admin and check_password_hash(admin['password_hash'],
password):
            session['admin_id'] = admin['admin_id']
            return redirect(url_for('dashboard'))
        return render_template('login.html')

@app.route('/api/log_event', methods=['POST'])
def log_event():
    data = request.get_json()
    rfid_uid = data.get('rfid_uid')
    access_point = data.get('access_point', 'Unknown')
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    conn = db_helper.get_connection()
    user = conn.execute('SELECT * FROM users WHERE rfid_uid = ?',
(rfid_uid,)).fetchone()
    user_id = user['user_id'] if user else None

    # Детекція аномалій
    anomaly_result = ml_interface.detect_anomaly(
        rfid_uid=rfid_uid,
        timestamp=timestamp,
        access_point=access_point,
        db_helper=db_helper
    )

    conn.execute('''
        INSERT INTO access_logs (user_id, rfid_uid, access_point,
timestamp,
                                is_anomaly, anomaly_type,
confidence_score)
        VALUES (?, ?, ?, ?, ?, ?, ?)
    ''', (user_id, rfid_uid, access_point, timestamp,
        anomaly_result['is_anomaly'], anomaly_result['anomaly_type'],
        anomaly_result['confidence_score']))
    conn.commit()
    conn.close()

    return jsonify(anomaly_result)

@app.route('/dashboard')
@login_required
def dashboard():
    return render_template('dashboard.html')

# ... інші маршрути (експорт, налаштування, користувачі)

def init_db():
    # Створення таблиць та дефолтного адміна
    # (див. повний файл)

```

```
    pass

if __name__ == '__main__':
    os.makedirs('database', exist_ok=True)
    init_db()
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Схема бази даних SQLite

Файл: schema.sql

Мова: SQL

Опис: структура таблиць users, access_logs, admin, індекси, view для детального перегляду та статистики.

```
CREATE TABLE IF NOT EXISTS users (  
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,  
    full_name TEXT NOT NULL,  
    department TEXT,  
    rfid_uid TEXT UNIQUE NOT NULL,  
    status TEXT DEFAULT 'active',  
    CHECK (status IN ('active', 'inactive'))  
);  
  
CREATE TABLE IF NOT EXISTS access_logs (  
    log_id INTEGER PRIMARY KEY AUTOINCREMENT,  
    user_id INTEGER,  
    rfid_uid TEXT NOT NULL,  
    access_point TEXT NOT NULL,  
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,  
    is_anomaly BOOLEAN DEFAULT 0,  
    anomaly_type TEXT,  
    confidence_score FLOAT,  
    FOREIGN KEY (user_id) REFERENCES users(user_id)  
);  
  
CREATE TABLE IF NOT EXISTS admin (  
    admin_id INTEGER PRIMARY KEY AUTOINCREMENT,  
    username TEXT UNIQUE NOT NULL,  
    password_hash TEXT NOT NULL  
);  
  
-- Індекси  
CREATE INDEX IF NOT EXISTS idx_access_logs_timestamp ON  
access_logs(timestamp);  
CREATE INDEX IF NOT EXISTS idx_access_logs_rfid_uid ON  
access_logs(rfid_uid);  
  
-- View: детальний перегляд логів  
CREATE VIEW IF NOT EXISTS v_access_logs_detailed AS  
SELECT l.log_id, l.rfid_uid, u.full_name, u.department, l.access_point,  
    l.timestamp, l.is_anomaly, l.anomaly_type  
FROM access_logs l  
LEFT JOIN users u ON l.user_id = u.user_id;
```

Генерація синтетичного датасету

Файл: generate_demo_data.py

Мова: Python 3

Опис: створення 50 користувачів, 15000+ записів, 3% аномалій (нічні, часті, вихідні доступи). Реалістичні патерни активності.

```
import random
from datetime import datetime, timedelta
from utils.db_helper import DatabaseHelper

TOTAL_ANOMALIES = 0.03
NIGHT_ACCESS_RATE = 0.1
WORK_HOURS = list(range(7, 21))

def create_50_students(db):
    departments = ['Факультет інформатики', 'Факультет медицини', ...]
    names = ['Іванов Іван', 'Петренко Петро', ...]
    for i in range(50):
        rfid = ''.join(random.choices('0123456789ABCDEF', k=8))
        db.add_user(full_name=names[i], department=departments[i%6],
rfid_uid=rfid)

def generate_professional_dataset(db_helper, start_date, end_date):
    students = db_helper.get_all_users()
    days = (end_date - start_date).days + 1
    total_logs = 0

    for current_date in (start_date + timedelta(n) for n in range(days)):
        for student in students:
            accesses_today = random.choices([0,1,2,3,4,5],
weights=[5,20,30,25,15,5], k=1)[0]
            for _ in range(accesses_today):
                hour = random.choice(WORK_HOURS)
                is_anomaly = random.random() < TOTAL_ANOMALIES
                # ... логіка аномалій
                db_helper.log_access(
                    rfid_uid=student['rfid_uid'],
                    access_point=random.choice(ACCESS_POINTS)[0],
                    timestamp=f"{current_date.strftime('%Y-%m-%d')}
{hour:02d}:00:00",
                    is_anomaly=is_anomaly
                )
                total_logs += 1
    return total_logs, int(total_logs * TOTAL_ANOMALIES)
```


Тренування Autoencoder моделі

Файл: train_model.py

Мова: Python 3 Бібліотеки: tensorflow, sklearn, numpy

Опис: навчання Autoencoder (12 $\rightarrow$ 8 $\rightarrow$ 12), 12 ознак, поріг за MSE, contamination=0,03.

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from sklearn.preprocessing import StandardScaler
import numpy as np

def create_autoencoder(input_dim=12):
    input_layer = Input(shape=(input_dim,))
    encoded = Dense(64, activation='relu')(input_layer)
    encoded = Dense(32, activation='relu')(encoded)
    encoded = Dense(8, activation='relu')(encoded) # Вузьке горло
    decoded = Dense(32, activation='relu')(encoded)
    decoded = Dense(64, activation='relu')(decoded)
    output_layer = Dense(input_dim, activation='linear')(decoded)
    autoencoder = Model(input_layer, output_layer)
    autoencoder.compile(optimizer='adam', loss='mse')
    return autoencoder

def train_autoencoder(X_train, epochs=50, batch_size=16):
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_train)
    autoencoder = create_autoencoder()
    autoencoder.fit(X_scaled, X_scaled, epochs=epochs,
batch_size=batch_size,
                    validation_split=0.1, verbose=1)
    return autoencoder, scaler

# Розрахунок порогу
reconstructions = autoencoder.predict(X_scaled)
mse = np.mean(np.power(X_scaled - reconstructions, 2), axis=1)
threshold = np.percentile(mse, 97) # 3% аномалій
```

Інтеграція ML та гібридна детекція

Файл: ml_interface.py

Мова: Python 3

Опис: 12 ознак, гібридна детекція (6 правил + Autoencoder), confidence score.

```
def _extract_features(self, rfid_uid, timestamp, access_point,
db_helper):
    dt = datetime.strptime(timestamp, '%Y-%m-%d %H:%M:%S')
    features = {
        'hour': dt.hour,
        'day_of_week': dt.weekday(),
        'is_weekend': 1 if dt.weekday() >= 5 else 0,
        'access_count_last_hour':
db_helper.count_access_last_hour(rfid_uid),
        'time_since_last_access': db_helper.time_since_last(rfid_uid),
        'access_point_encoded': self._encode_access_point(access_point),
        'department_encoded': self._encode_department(department),
        'month': dt.month,
        'is_morning': 1 if 6 <= dt.hour <= 8 else 0,
        'is_lunch': 1 if 12 <= dt.hour <= 14 else 0,
        'is_evening': 1 if 17 <= dt.hour <= 19 else 0
    }
    return features

def _ml_based_detection(self, features):
    vector = np.array(list(features.values())).reshape(1, -1)
    scaled = self.scaler.transform(vector)
    recon = self.autoencoder.predict(scaled, verbose=0)
    mse = np.mean(np.power(scaled - recon, 2))
    is_anomaly = mse > self.threshold
    confidence = min(0.99, 0.5 + (mse / self.threshold) * 0.3)
    return {'is_anomaly': is_anomaly, 'anomaly_type': 'ml_detected',
'confidence_score': confidence}
```

Веб-інтерфейс: Дашборд

Файл: dashboard.html

Технології: HTML, Bootstrap 5, Chart.js

Опис: динамічний дашборд з графіками, статистикою, автооновленням кожні 10 секунд.

```
<div class="stat-card">
  <h3 id="totalUsers">0</h3>
  <p>Всього користувачів</p>
</div>
<canvas id="hourlyChart" class="compact-chart"></canvas>

<script>
  let hourlyChart;
  async function loadStatistics() {
    const response = await fetch('/api/statistics');
    const data = await response.json();
    document.getElementById('totalUsers').textContent =
data.total_users;

    if (hourlyChart) hourlyChart.destroy();
    const ctx =
document.getElementById('hourlyChart').getContext('2d');
    hourlyChart = new Chart(ctx, {
      type: 'line',
      data: {
        labels: data.hourly_activity.map(h => h.hour + ':00'),
        datasets: [{
          label: 'Кількість входів',
          data: data.hourly_activity.map(h => h.count),
          borderColor: 'rgba(75, 192, 192, 1)',
          fill: false
        }]
      }
    });
  }
  setInterval(loadStatistics, 10000);
  loadStatistics();
</script>
```

Робота з базою даних (утилітарний клас)

Файл: db_helper.py

Мова: Python 3

Опис: CRUD-операції, логування, статистика, резервне копіювання.

```
class DatabaseHelper:
    def __init__(self, db_path='database/access_control.db'):
        self.db_path = db_path

    def get_connection(self):
        conn = sqlite3.connect(self.db_path)
        conn.row_factory = sqlite3.Row
        return conn

    def add_user(self, full_name, rfid_uid, department=None,
status='active'):
        conn = self.get_connection()
        cursor = conn.cursor()
        cursor.execute('INSERT INTO users (...) VALUES (?, ?, ?, ?)',
                        (full_name, department, rfid_uid, status))
        conn.commit()
        conn.close()

    def log_access(self, rfid_uid, access_point, timestamp, is_anomaly=0,
anomaly_type=None, confidence=0.0):
        conn = self.get_connection()
        conn.execute('INSERT INTO access_logs (...) VALUES (?, ?, ?, ?,
?, ?)',
                    (None, rfid_uid, access_point, timestamp,
is_anomaly, anomaly_type))
        conn.commit()
        conn.close()
```