МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувачка кафедри,
д-р техн. наук, проф.

_____Ірина ЖУРАВСЬКА

« __ » _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

Система керування доступом на базі RFID та Spring Boot

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія»

Здобувач _____Андрій ШИЛІК
підпис

«__» _____ 2025 р.

Керівник канд. фіз.-мат. наук,
доцент кафедри КІ _____ Сергій ПУЗИРЬОВ
підпис
«__» _____ 2025 р.

Миколаїв – 2025

Факультет	Комп’ютерних наук
Кафедра	Комп’ютерної інженерії
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	123 Комп’ютерна інженерія
Освітня програма	Комп’ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри комп’ютерної інженерії
_____ Ірина ЖУРАВСЬКА
«_____» _____ 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Шиліка Андрія Олександровича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи
Система керування доступом на базі RFID та Spring Boot.

Затверджена наказом по ЧНУ ім. Петра Могили від 30.10.2024 № 297.

2. Строк представлення кваліфікаційної роботи «____» _____ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Очікуваним результатом роботи є реалізація апаратного та програмного забезпечення системи керування доступом на базі RFID та Spring Boot.

4. Перелік питань, що підлягають розробці:
- проаналізувати сучасні технології та методи керування доступом;
- обґрунтування вимог до АПЗ;
- розробка загальної архітектури системи та підбір компонентів;
- розробити апаратну частину системи;
- створити серверну частину системи на платформі Spring Boot для обробки даних і управління доступом;
- забезпечити інтеграцію апаратної та програмної;
- виконати тестування системи та оцінити її надійність.

5. Перелік графічних матеріалів
Презентація

6. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Керівник роботи _____ Сергій ПУЗИРЬОВ
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Андрій ШИЛІК
Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Дата видачі завдання «_____» _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної бакалаврської роботи

Тема:____Система керування доступом на базі RFID та Spring Boot._____

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КБР	30.10.2024	30.10.2024	Виконано
2	Огляд літератури за темою роботи	30.01.2025	14.03.2025	Виконано
3	Складання календарного плану КБР	14.03.2025	16.03.2025	Виконано
4	Аналіз предметної області	16.03.2025	01.04.2025	Виконано
5	Розробка проєктних рішень	01.04.2025	17.04.2025	Виконано
6	Моделювання та конструювання АПЗ	17.04.2025	11.05.2025	Виконано
7	Перевірка працездатності, тестування та апробація розробленого АПЗ, аналіз результатів тестування	11.05.2025	31.05.2025	Виконано
8	Відгук керівника КБР	16.06.2025	16.06.2025	Виконано
9	Оформлення КБР та презентації	31.05.2025	10.06.2025	Виконано
10	Попередній захист	21.05.2025	04.06.2025	Виконано
11	Рецензування	15.06.2025	15.06.2025	Виконано
12	Завершення оформлення КБР та презентації	10.06.2025	16.06.2025	Виконано
13	Захист кваліфікаційної бакалаврської роботи	25.06.2025	25.06.2025	

Керівник роботи _____ Сергій ПУЗИРЬОВ

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

Здобувач _____ Андрій ШИЛІК

Особистий підпис *Власне ім'я ПРІЗВИЩЕ*

«____» _____ 2025 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи
«Система керування доступом на базі RFID та Spring Boot»
Студент 405 гр.: Шилік Андрій Олександрович
Керівник: канд. фіз-мат. наук, доцент Пузирьов Сергій Володимирович

У сучасних умовах розвитку інформаційних технологій зростає попит на ефективні, безпечні та доступні системи контролю доступу (СКД). Традиційні механізми ідентифікації (ключі, паролі) не завжди гарантують необхідний рівень захисту та масштабованості. Тому особливу актуальність набуває використання RFID-технологій у поєднанні з вебінтерфейсами на базі Spring Boot.

Об’єкт дослідження – процеси організації та автоматизації доступу до об’єктів із обмеженим доступом.

Предмет дослідження – апаратно-програмне забезпечення (АПЗ) СКД на основі RFID-ідентифікації та фреймворку Spring Boot.

Мета роботи – створення АПЗ СКД, що забезпечує автоматизовану ідентифікацію користувачів та управління доступом із можливістю інтеграції в мережу.

Практична значимість: розроблена СКД забезпечує економічно доступне рішення з високим рівнем функціональності, придатне для впровадження в офісах, освітніх установах, житлових комплексах тощо. Перевагою є можливість віддаленого адміністрування доступу та реєстрації подій у реальному часі.

Апробація. Основні результати були представлені на науково-практичній конференції «Ольвійський форум-2025: Стратегії країн Причорноморського регіону в геополітичному просторі» (м. Миколаїв).

Основні результати:

– у першому розділі проведено аналіз сучасних СКД та обґрунтовано вибір технології RFID як оптимальної з точки зору безпеки, надійності та вартості;

– у другому розділі описано структуру АПЗ: апаратна частина включає RFID-модуль, мікроконтролер ESP8266, сервопривід, клавіатуру та дисплей; описана програмна частина, яка реалізована у вигляді серверного застосунку на Spring Boot;

– у третьому розділі розроблено апаратно-програмний комплекс (АПК), реалізовано інтеграцію системи, проведено тестування на макетній моделі та підтверджено працездатність проєкту.

Обсяг роботи: 76 сторінок (без додатків), 4 таблиці, 41 рисунок, 30 джерел посилання та 4 додатки.

Ключові слова: *система керування доступом, RFID, безпека, тайм-менеджмент, Spring Boot.*

ABSTRACT

of the Bachelor's Thesis

"Access control system based on RFID and Spring Boot"

Student: Andrii Shylik

Supervisor: Candidate of Phys.-Math. Sciences, Assoc. Prof. Serhii Puzyrov

In the current era of information technology development, there is an increasing demand for efficient, secure, and affordable access control systems. Traditional identification mechanisms such as keys or passwords do not always provide the required level of protection and scalability. Therefore, the use of RFID technologies in combination with Spring Boot-based web interfaces has become particularly relevant.

Object of research: the processes of organizing and automating access to restricted areas or objects.

Subject of research: the hardware and software solution for an access control system based on RFID identification and the Spring Boot framework.

Purpose of the study: to develop a hardware-software access control system that ensures automated user identification and access management, with the ability to integrate into a network environment.

Practical significance: The developed system offers an economically efficient and functionally robust solution suitable for implementation in offices, educational institutions, residential complexes, and similar environments. A key advantage is the ability to manage access rights remotely and to log access events in real time.

Approbation: The main findings of the study were presented at the scientific-practical conference «Olvian Forum 2025: Strategies of the Black Sea Region Countries in the Geopolitical Space» (Mykolaiv, Ukraine).

Main results:

– in Chapter 1, a comprehensive analysis of modern ACS technologies was conducted. RFID was substantiated as the optimal solution due to its balance of security, reliability, and cost-effectiveness;

– in Chapter 2, the structure of the hardware-software complex was described. The hardware part includes an RFID module, ESP8266 microcontroller, servo motor, keypad, and display. The software part was implemented as a Spring Boot-based server application;

– in Chapter 3, the embedded and backend software was developed, integration between hardware and server was achieved, the system was tested on a prototype, and its functionality was confirmed.

Thesis consist:75 pages (without appendices), 4 tables, 41 figures, 40 references, and 4 appendices.

Keywords: *access control system, Radio Frequency IDentification, security, time management, Spring Boot.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АПЗ – апаратно-програмне забезпечення
АПК – апаратно-програмний комплекс
КБР – кваліфікаційна бакалаврська робота
СКД – системи керування доступом

API – Application Programming Interface
IDE – Integrated Development Environment
IoT – Internet of Things
NFC – Near Field Communication
OTP – One-Time Password
RFID – Radio Frequency Identification
SFA – Single-Factor Authentication
SPI – Serial Peripheral Interface
2FA – Two-Factor Authentication

ВСТУП

У наший час ефективні та доступні системи керування доступом (СКД) є життєво необхідними для захисту об’єктів з обмеженим доступом, як-от офіси, склади, навчальні заклади та житлові комплекси. Оптимізація тайм-менеджменту через швидке адміністрування доступу перетворюється на фінансові вигоди. Обмеження традиційних методів, таких як механічні замки чи паролі, щодо безпеки та масштабованості роблять розробку інноваційних СКД актуальною для підвищення рівня безпеки та операційної ефективності.

Актуальність теми кваліфікаційної бакалаврської роботи визначається необхідністю розробки економічно доступних і високоефективних рішень для керування доступом, які поєднують апаратне забезпечення з програмними засобами, забезпечуючи не лише безпеку, а й оптимізацію часу адміністрування.

Практичне значення роботи полягає у створенні системи, яка поєднує надійні механізми аутентифікації з гнучкими інструментами управління доступом, що сприяють підвищенню рівня безпеки та спрощенню адміністративних процесів. Крім того, розробка системи з акцентом на інтеграцію з сучасними інформаційними технологіями дозволяє не лише вирішувати поточні завдання, але й створювати передумови для масштабування та адаптації до майбутніх викликів. Практична цінність роботи також проявляється у можливості скорочення витрат часу на управління доступом, що є важливим фактором для підвищення продуктивності та конкурентоспроможності організацій.

Об’єктом дослідження є процеси організації та регулювання доступу до певних приміщень або ресурсів.

Предметом дослідження є створення та впровадження СКД, що базується на технології радіочастотної ідентифікації (англ. Radio Frequency IDentification, RFID) і фреймворку Spring Boot.

Метою дослідження є створення СКД, яка забезпечує автоматизовану ідентифікацію користувачів і керування механізмом доступу шляхом розробки

апаратно-програмного забезпечення (АПЗ) на основі RFID з вебзастосунком на базі Spring Boot.

Для досягнення мети поставлено такі завдання:

– проаналізувати сучасні технології та методи керування доступом;
– обгрунтування вимог до АПЗ;
– розробка загальної архітектури системи та підбір компонентів;
– розробити апаратну частину системи;
– створити серверну частину системи на платформі Spring Boot для обробки даних і управління доступом;
– забезпечити інтеграцію апаратної та програмної;
– виконати тестування системи та оцінити її надійність.

RFID-технології широко застосовуються у безпеці та автоматизації, але багато рішень не інтегруються з вебзастосунками або є занадто дорогими. Актуальною є потреба в доступних системах, що поєднують апаратне забезпечення з відкритими платформами, як-от Spring Boot, для гнучкості та ефективного управління. Популярність Internet of Things (IoT) підкреслює важливість віддаленого доступу та аналізу даних у реальному часі. Результати дослідження можуть бути застосовані в системах безпеки для різних об’єктів (офісів, складів тощо), ставши основою комерційних рішень або доповненням до IoT-систем, підвищуючи їх ефективність.

Апробація Шилік А. О. (бакалаврант), Пузирьов С. В. (канд. фіз-мат. наук, доц. каф. комп’ютерної інженерії факультету комп’ютерних наук, Чорноморський нац. ун-т ім. Петра Могили, м. Миколаїв, Україна). Система керування доступом на базі RFID та Spring Boot. «Ольвійський форум-2025: Стратегії країн причорноморського регіону в геополітичному просторі» [29].

1 АНАЛІЗ ІСНУЮЧИХ СКД

1.1 Актуальність системи ідентифікації RFID

Технологія радіочастотної ідентифікації RFID є ключовим елементом сучасних систем автоматизації, що забезпечують безконтактну ідентифікацію об’єктів за допомогою радіохвиль. Її беззаперечна актуальність зумовлена зростаючою потребою в оптимізації управління ресурсами, мінімізації операційних витрат та підвищенні рівня безпеки в різноманітних галузях економіки. У межах даного розділу буде проведено поглиблений аналіз причин, що обумовлюють непересічне значення RFID як незамінної технології в сучасному глобальному контексті.

RFID-системи демонструють широку інтеграцію в численні сектори економіки. У сфері логістики вони забезпечують моніторинг товарних потоків у режимі реального часу, що сприяє мінімізації ризиків дефіциту та оптимізації ланцюгів постачання [7]. У роздрібній торгівлі RFID забезпечує високу точність обліку товарно-матеріальних цінностей, надаючи персоналу актуальну інформацію про наявність продукції, що призводить до скорочення витрат на закупівлі та підвищення ефективності управління запасами [6].

Крім того, технологія знаходить застосування в автомобільній промисловості для відстеження транспортних засобів, управління програмами лояльності клієнтів та оптимізації графіків технічного обслуговування. У галузі текстильної промисловості RFID використовується для автоматизації обліку текстильних виробів, спрощуючи управління інвентарем [11]. У секторі охорони здоров’я RFID-мітки сприяють відстеженню медичного обладнання та фармацевтичних препаратів, підвищуючи рівень безпеки пацієнтів. Наведені приклади ілюструють універсальність технології та її здатність до адаптації відповідно до специфічних потреб різних галузей промисловості та сфери послуг.

Однією з ключових детермінант актуальності RFID є її значні переваги порівняно з традиційними методами ідентифікації, такими як технологія штрих-

кодування. RFID-системи забезпечують можливість одночасного зчитування до 200 ідентифікаційних міток на відстані до 20 м, що істотно прискорює процеси інвентаризації та логістичні операції. На відміну від штрих-кодів, які вимагають прямої оптичної видимості та послідовного зчитування кожної одиниці, RFID забезпечує безконтактне зчитування, що призводить до економії часових ресурсів та зниження ймовірності операційних помилок. Крім того, RFID-мітки характеризуються підвищеною довговічністю, зберігаючи працездатність протягом тривалого періоду експлуатації за належних умов, що обумовлює їхню економічну вигідність у довгостроковій перспективі.

RFID відіграє важливу роль у впровадженні автоматизованих бізнес-процесів, що є критично важливим фактором у сучасних умовах високої конкуренції. Завдяки високій точності та швидкості зчитування даних, RFID-системи мінімізують потребу в залученні ручної праці, що дозволяє підприємствам оптимізувати використання ресурсів та знижувати операційні витрати. Наприклад, в управлінні складським господарством RFID забезпечує оперативний пошук товарних одиниць та переміщення партій без необхідності повторного обліку, що сприяє підвищенню продуктивності праці. У сфері торгівлі точний облік товарних запасів знижує ризик виникнення дефіциту та потреби в термінових закупівлях за завищеними цінами

RFID-системи також сприяють зміцненню систем безпеки та контролю. У торговельних мережах вони використовуються для запобігання крадіжкам шляхом автоматизованого відстеження товарних одиниць. У системах контролю доступу RFID-ідентифікатори забезпечують санкціонований доступ до приміщень або транспортних засобів. Зазначені функціональні можливості роблять технологію незамінною для організацій, які прагнуть забезпечити захист власних активів та підтримувати високий рівень контролю.

Актуальність RFID також підсилюється її інтеграцією з іншими передовими технологіями, зокрема з концепцією Інтернету речей . RFID-мітки є ключовим компонентом IoT-систем, забезпечуючи ідентифікацію та моніторинг об’єктів у інтелектуальних мережах. Наприклад, у інтелектуальних складських

комплексах RFID-системи інтегруються з програмним забезпеченням для управління запасами, що дозволяє створювати повністю автоматизовані системи управління. Зазначена синергія з IoT та іншими технологічними рішеннями відкриває нові перспективи для застосування RFID у майбутньому.

Незважаючи на значний перелік переваг, RFID-системи мають певні обмеження, які потребують врахування при їхньому впровадженні. Значні початкові інвестиції можуть становити бар’єр для малих підприємств. Крім того, RFID-системи є чутливими до електромагнітних перешкод та впливу металевих поверхонь і рідинних середовищ, що може ускладнювати їхнє застосування в певних операційних умовах. Питання забезпечення безпеки даних також залишаються актуальними, оскільки ідентифікаційні мітки можуть бути вразливими до несанкціонованого зчитування інформації. Проте зазначені виклики стимулюють подальші інновації, спрямовані на подолання існуючих обмежень.

1.2 Складові частини та принцип роботи RFID-систем

Технологія радіочастотної ідентифікації RFID являє собою сучасний метод безконтактної ідентифікації об’єктів, що ґрунтується на використанні радіохвиль. RFID-системи набули широкого застосування у сферах логістики, роздрібної торгівлі, охорони здоров’я, транспортній інфраструктурі та інших галузях завдяки їхнім можливостям забезпечення оперативного, автоматизованого та надійного зчитування даних [13].

Основні складові RFID-систем:

– RFID-мітка: ідентифікаційна мітка, будучи первинним носієм інформації в RFID-системі, конструктивно включає інтегральну мікросхему для зберігання унікального ідентифікаційного коду та, за потреби, додаткових даних, а також антену, що забезпечує комунікацію зі зчитувачем; залежно від джерела енергії, мітки поділяються на пасивні, які не мають власного живлення та активуються електромагнітним полем зчитувача, характеризуючись компактністю, нижчою вартістю та обмеженим радіусом дії (до 10 м); активні

мітки, що містять автономне джерело живлення, забезпечуючи функціонування на значних відстанях (до 100 м і більше) та можливість ініціювання передачі даних; і напівпасивні мітки, які використовують батарею для живлення мікросхеми, але для передачі сигналу залежать від електромагнітного поля зчитувача;

– зчитувальний пристрій: зчитувач являє собою пристрій, що генерує електромагнітне поле для активації мітки та здійснює прийом радіосигналів, що нею транслюються. Зчитувачі можуть бути стаціонарними (наприклад, інстальованими на контрольно-пропускних пунктах складських приміщень) або мобільними (портативні сканери). Їхня конструкція включає антени для передачі та прийому сигналів, а також модулі для первинної обробки даних та їхньої подальшої передачі до інформаційної системи;

– інформаційна система: дані, отримані від мітки за посередництвом зчитувача, надходять до комп’ютеризованої системи для подальшої обробки, зберігання та аналізу. Зазначена система може бути інтегрована з базами даних, системами управління складськими запасами (англ. Warehouse Management System) або іншими програмними платформами з метою автоматизації бізнес-процесів.

Схематичне зображення складових RFID-системи (рис. 1.1).

Рисунок 1.1 – Схематичне зображення складових RFID та їх взаємодія [6]

Зображення ілюструє принцип роботи RFID-системи, де зчитувач з антеною передає радіохвилі до RFID-мітки, отримує від неї дані та передає їх на комп’ютер для подальшої обробки.

Функціонування RFID-систем ґрунтується на принципі обміну радіосигналами між ідентифікаційною міткою та зчитувальним пристроєм у визначеному частотному діапазоні.

Процес ідентифікації включає наступні етапи:

– генерація електромагнітного поля: зчитувач за допомогою власної антени створює електромагнітне поле в межах операційної зони. Частота даного поля визначається типом використовуваної RFID-системи (низькочастотна, високочастотна або ультрависокочастотна);

– активація мітки: при потраплянні RFID-мітки в зону дії електромагнітного поля, її антена абсорбує енергію. У пасивних мітках отримана енергія використовується для живлення мікросхеми, що ініціює перехід мітки в активний стан. Активні мітки, маючи автономне джерело енергії, здатні самостійно ініціювати процес передачі даних;

– трансляція даних: активована мітка передає збережені дані (наприклад, унікальний ідентифікатор) назад до зчитувача за допомогою радіосигналу. Даний процес отримав назву модуляції сигналу. У більшості систем застосовується принцип зворотного розсіювання (англ. backscatter), при якому мітка здійснює модуляцію відбитого сигналу зчитувача, накладаючи на нього власну інформацію;

– обробка даних: зчитувач здійснює декодування отриманого сигналу, трансформуючи його в цифровий формат, та передає отримані дані до інформаційної системи. Система, у свою чергу, обробляє отриману інформацію, співставляючи її з даними, що зберігаються в базі даних, та ініціює відповідні дії, такі як актуалізація даних про інвентар, верифікація доступу або моніторинг переміщення об’єкта.

Блок-схема принципу роботи RFID-систем (рис. 1.2).

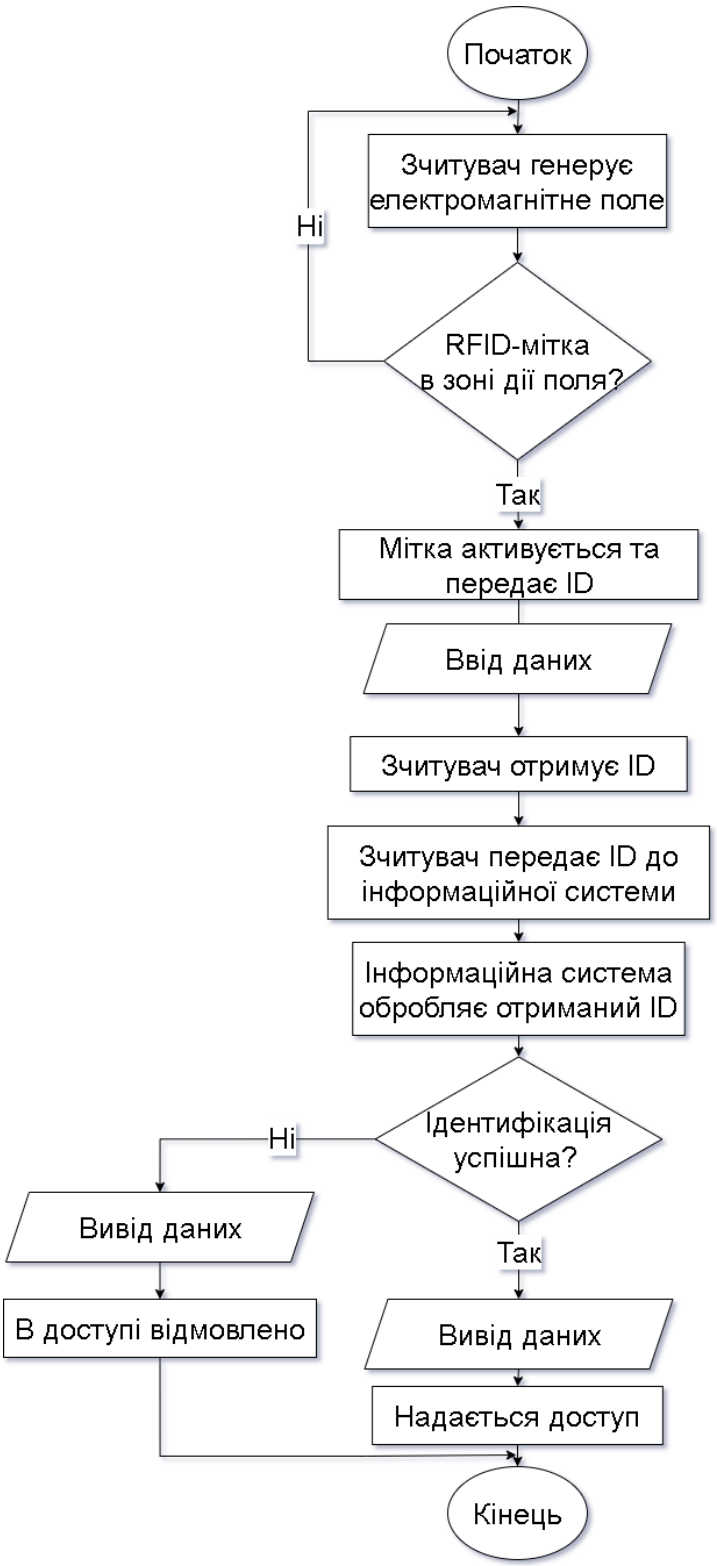

Рисунок 1.2 – Блок-схема, що ілюструє принцип роботи RFID-системи

Блок-схема демонструє циклічний процес пошуку мітки, її ідентифікації та прийняття рішення про надання або відмову в доступі на основі отриманого ідентифікатора.

1.3 Огляд існуючих СКД та їх порівняння

У сучасному світі, крім RFID, значне поширення набули біометричні системи. Останні, використовуючи унікальні фізіологічні характеристики суб’єкта, такі як дактилоскопічні візерунки, ідентифікація райдужної оболонки ока або розпізнавання обличчя, забезпечують високий ступінь автентифікації та безпеки [12]. Мобільні системи контролю доступу, що функціонують на базі смартфонів, репрезентують зручний та гнучкий підхід, реалізований через спеціалізовані програмні застосунки. Кодові замкові механізми, що оперують персональними ідентифікаційними номерами (PIN-кодами), являють собою економічно ефективне рішення для забезпечення базового рівня безпеки. Системи, що базуються на магнітних картках, зберігають свою актуальність завдяки їхній доступності та розвиненій інфраструктурі.

Аналізуючи доступну інформацію, можна виділити кілька ключових типів систем, які конкурують із RFID, і зрозуміти, чому вони є поширеними.

СКД класифікуються залежно від архітектури, способу ідентифікації, рівня безпеки та сфери застосування.

1) Біометричні системи: даний клас систем використовує біометричні сканери для верифікації особистості на основі унікальних фізіологічних характеристик, таких як дактилоскопічні відбитки, іридограма ока або геометрія обличчя. Забезпечуючи високий рівень автентифікації завдяки складності фальсифікації біологічних параметрів, біометричні системи є оптимальним рішенням для об’єктів з підвищеними вимогами до безпеки, включаючи фінансові установи та урядові об’єкти.

Cyphrax FRD [1] є настільним біометричним зчитувачем відбитків пальців, який використовується для забезпечення безпечного доступу. Він дозволяє користувачам реєструвати свої відбитки пальців для подальшої автентифікації, що підвищує рівень безпеки, порівняно з традиційними методами, такими як паролі чи фізичні ключі. Пристрій підключається через USB і може інтегруватися з операційними системами, наприклад, через підтримку Windows Hello (рис. 1.3).

Рисунок 1.3 – Біометричний настільний зчитувач відбитків пальців Cyphrax FRD [1]

2) Мобільні СКД: функціонування цих систем ґрунтується на використанні персональних мобільних пристроїв (смартфонів) як засобів ідентифікації. Зв’язок для надання доступу здійснюється за допомогою бездротових технологій, таких як Wi-Fi, Bluetooth або мережі мобільного зв’язку, через спеціалізовані мобільні застосунки. Зростання популярності мобільних систем зумовлене їхньою зручністю, елімінуючи необхідність у фізичних носіях і забезпечуючи можливість дистанційного адміністрування прав доступу.

Помітним рішенням у даній категорії є система HID Mobile Access [2], що надає можливість керування доступом за допомогою смартфонів. Функціональність системи базується на використанні бездротових протоколів зв’язку, зокрема Bluetooth, NFC та Wi-Fi, для верифікації користувачів (рис. 1.4).

Рисунок 1.4 – Система HID Mobile Access [2]

3) Кодові замкові системи: в основу роботи цих систем покладено використання клавіатури для введення унікальної цифрової або буквено-

цифрової комбінації. Характеризуючись простотою впровадження та економічною ефективністю, кодові замки забезпечують середній рівень безпеки та широко застосовуються в офісних приміщеннях і житлових комплексах.

Відомим прикладом кодового замка є Yale Assure Lock[3], що функціонує на основі введення користувачем персонального ідентифікаційного номера (PIN-коду) для отримання доступу. Дане рішення поєднує зручність експлуатації з належним рівнем безпеки для широкого спектра застосувань(рис. 1.5).

Рисунок 1.5 – Кодовий замок Yale Assure Lock[3]

4) СКД на основі магнітних карток: даний тип систем використовує ідентифікаційні картки з магнітною смугою, інформація з якої зчитується спеціалізованими пристроями. Незважаючи на нижчий рівень безпеки порівняно з RFID через потенційну можливість копіювання магнітної смуги, ці системи зберігають свою поширеність завдяки низькій собівартості та наявній інфраструктурі, особливо в існуючих системах контролю доступу.

Salto Systems[5] є відомим виробником електронних замків та систем контролю доступу для готелів та інших комерційних об’єктів. Їхні системи на основі магнітних карток є популярним рішенням завдяки своїй надійності та зручності використання як для персоналу, так і для гостей(рис. 1.6).

Рисунок 1.6 – Апаратна платформа СКД XS4 SALTO [5]

Вибір конкретної СКД визначається комплексом факторів, серед яких ключову роль відіграє контекст її застосування. Біометричні системи набули поширення завдяки високому рівню безпеки, що забезпечується унікальністю та невідчужуваністю біологічних ідентифікаторів, що є критично важливим для об’єктів з високим рівнем ризику. Мобільні системи демонструють зростаючу популярність завдяки зручності використання та можливості оперативного керування правами доступу через програмні інтерфейси, що є значною перевагою для великих організацій. Кодові замки приваблюють своєю простотою експлуатації та відсутністю потреби у додаткових фізичних носіях, що знижує операційні витрати. Системи з магнітними картками, незважаючи на певні обмеження в безпеці, залишаються затребуваними завдяки їхній економічності та сумісності з існуючою інфраструктурою, особливо в об’єктах старої забудови.

Таблиця 1.1 – Порівняння основних СКД

№	Система	Рівень безпеки	Зручність	Вартість	Приклади використання
1	Біометричні системи	Високий	Середній	Висока	Банки, урядові установи
2	Мобільні системи	Середній-Високий	Висока	Середня	Офіси, житлові комплекси

№	Система	Рівень безпеки	Зручність	Вартість	Приклади використання
3	Кодові замки	Низький-Середній	Висока	Низька	Малі офіси, гаражі
4	Магнітні картки	Низький	Середній	Низька	Старий офіс, готелі

Ця таблиця допомагає зрозуміти, чому кожна система має своє місце на ринку, залежно від балансу між безпекою, зручністю та бюджетом.

Технологія безконтактної ближньої радіозв’язку (англ. Near Field Communication, NFC), хоча й споріднена з радіочастотною ідентифікацією RFID, є самостійною технологією. У певних контекстах NFC може розглядатися як альтернативне рішення завдяки вдосконаленим функціям безпеки. Однак, з огляду на поставлене завдання, основна увага в дослідженні зосереджена на СКД, що не використовують радіочастотний спектр, таких як біометричні та мобільні технології.

Отже, існуюче різноманіття альтернативних СКД відображає потребу в адаптивності рішень. Діапазон варіюється від високозахищених біометричних систем до економічно ефективних кодових замків. Процес вибору оптимальної системи визначається конкретними операційними умовами, включаючи рівень потенційних загроз, доступні фінансові ресурси та наявну технічну базу. Ця обставина зумовлює багатоаспектність досліджуваної проблематики та відкриває перспективи для подальшого наукового вивчення.

1.4 Формування вимог до АПЗ

Розроблена СКД, що функціонує на основі RFID та фреймворку Spring Boot, призначена для автоматизованого регулювання доступу до зон обмеженого користування шляхом ідентифікації суб’єктів за допомогою безконтактних RFID-міток. Зазначена система забезпечує централізоване керування доступом через серверний застосунок, підвищуючи рівень безпеки об’єктів та

оптимізуючи зручність авторизації користувачів. Функціональність системи включає управління електромеханічними запірними механізмами, зокрема сервоприводами, для здійснення блокування та розблокування доступу згідно з документацією Complete Guide to RFID Access Control Systems [8].

Розроблений функціонал відповідає чинним стандартам у галузі RFID-технологій, включаючи ISO/IEC 14443A для безконтактних ідентифікаційних карток, а також стандартам безпеки обробки даних, зокрема щодо захисту персональних даних користувачів [9]. Це забезпечує інтероперабельність із сучасними технічними рішеннями та відповідність актуальним регуляторним вимогам.

Зазначена реалізація призначена для інсталяції та експлуатації в організаційних структурах, таких як офісні приміщення, освітні установи або житлові комплекси, з метою контролю доступу до внутрішніх просторів [10]. До існуючих обмежень належать діапазон зчитування RFID-міток та функціональна залежність від стабільного бездротового з’єднання Wi-Fi.

Запропоноване рішення знаходить застосування у сфері автоматизованого контролю доступу до фізично захищених зон, таких як дверні прорізи, ворота або ліфтові кабіни, у корпоративному, освітньому або житловому секторах. Його функціональність забезпечує оперативну ідентифікацію користувачів та ведення протоколу подій доступу.

Характеристики користувачів розробки:

– персонал, відповідальний за адміністрування облікових записів користувачів, налаштування прав доступу та моніторинг журналів подій через інтерфейс на основі вебзастосунків;

– суб’єкти, що використовують RFID-ідентифікатори у формі карток або брелоків для отримання санкціонованого доступу до контрольованих зон.

Запропонована система складається з апаратної складової (мікроконтролер, RFID-модуль, сервопривід, дисплей, клавіатура) та програмної складової (вбудоване програмне забезпечення для мікрокотролера та серверний застосунок на базі Spring Boot). Мікроконтролер здійснює обробку даних,

отриманих від периферійних пристроїв, та забезпечує комунікацію з сервером через бездротове з’єднання Wi-Fi.

Вимоги до компонентів апаратного забезпечення представлені нижче у табл. 1.2.

Таблиця 1.2 – Вимоги до апаратного забезпечення

№	Компонент	Вимоги
1	Мікроконтролер	Повинен мати Wi-Fi, підтримку інтерфейсів (I2C, SPI, UART) і достатню обчислювальну потужність
2	RFID-модуль	Підтримка SPI/I2C/UART, надійне зчитування міток на відстані до 5 см
3	Сервопривід	Має забезпечувати базову демонстрацію роботи системи доступу
4	Дисплей	Має відображати цифри та символи для кращої взаємодії клієнта з СКД
5	Клавіатура	Забезпечує введення цифр і службових команд (наприклад, «ОК», «Скидання»)

Апаратна конфігурація технічного рішення включає наступні елементи:

– мікроконтролерна плата: WiFi-модуль, що забезпечує зчитування даних з RFID-модуля, управління сервоприводом, відображення інформації на дисплеї, отримання введених даних з клавіатури та встановлення зв’язку з сервером через бездротову мережу;

– RFID-модуль, що функціонує на частоті 13,56 МГц та підключається до мікроконтролера через інтерфейс SPI (або I2C), призначений для ідентифікації користувачів за допомогою безконтактних ідентифікаційних карток;

– сервопривід, що імітує роботу механізму блокування/розблокування дверного замка;

– дисплей, що підключається через інтерфейс I2C та використовується для відображення цифр та літр;

– клавіатура, що підключається до мікроконтролера та призначена для введення цифр та символів.

Функціональні можливості запропонованого технічного рішення обмежені дальністю зчитування RFID-міток (до 5 см), обчислювальною потужністю мікроконтролера та стабільністю бездротового з’єднання Wi-Fi. Кількість одночасних користувачів, що можуть бути обслужені розробкою, залежить від ресурсів серверної платформи.

Програмна архітектура запропонованого комплексу реалізована за принципом клієнт-серверної взаємодії, де мікроконтролер виконує функцію клієнта, ініціюючи запити до серверного застосунку на базі Spring Boot, який здійснює обробку даних та надсилає відповідні результати згідно з документацією Spring Boot [4].

Нефункціональні вимоги є основою для забезпечення стабільності, безпеки та зручності використання СКД. Вони сприяють створенню гнучкої та масштабованої платформи, що відкриває можливості для подальшого розвитку функціоналу (табл. 1.3).

Таблиця 1.3 – Інші вимоги

№	Вимога	Опис
1	Продуктивність	Система повинна обробляти запити на доступ не довше за 2–3 с
2	Вартість	Загальна вартість компонентів не повинна перевищувати 2000–3000 грн
3	Надійність	Система повинна демонструвати стабільну роботу основних компонентів та інтеграцію апаратної частини з програмним забезпеченням
4	Безпека	Має бути реалізовано механізм контролю доступу для безпечної перевірки користувачів

Система працює за таким алгоритмом: RFID-модуль зчитує дані з картки, мікроконтролер обробляє їх і передає на сервер через Wi-Fi. Сервер на Spring

Boot перевіряє права доступу та надсилає результат авторизації. Відповідно до нього, мікроконтролер керує сервоприводом для розблокування або блокування доступу. Дисплей і клавіатура використовуються для двофакторного захисту. Уся система працює в реальному часі, забезпечуючи централізоване управління доступом.

Висновки до розділу 1

У першому розділі проаналізовано RFID-технології, відзначивши їх ефективність для автоматизованої ідентифікації завдяки безконтактності, швидкості та масштабованості. Підкреслено широке застосування RFID у логістиці, торгівлі, медицині, промисловості та безпеці.

Порівнюючи RFID з традиційними методами (наприклад, штрих-кодами), виділено її здатність одночасно зчитувати багато міток без прямої видимості та довговічність. Проте, зазначено й обмеження RFID: чутливість до перешкод, дальність зчитування та вартість.

Проведено порівняльний аналіз альтернативних СКД: біометричних, мобільних, кодових замків та магнітних карток. Визначено їхні переваги (наприклад, висока безпека біометрії) та недоліки (висока вартість біометрії, залежність мобільних систем від смартфонів, низький захист кодових замків та магнітних карток), а також оптимальні сфери застосування.

Сформовано вимоги до АПЗ майбутньої системи, включаючи технічні параметри, функціональні потреби, обмеження за вартістю/продуктивністю та вимоги до безпеки даних. Це стало основою для проєктування СКД

2 ОПИС ОБРАНОГО АПЗ СКД

2.1 Апаратні компоненти

2.1.1 Модуль RFID RC522 та приклад його програмування

RFID-модуль RC522 [17] – це поширений і доступний компонент для взаємодії з RFID-картами, що працює на частоті 13,56 МГц. В його основі лежить чіп MFRC522 від NXP, який підтримує стандарт ISO/IEC 14443 A/MIFARE. Це забезпечує його сумісність з такими поширеними картками, як MIFARE Classic 1K, що часто використовуються в системах контролю доступу.

Рисунок 2.1 – RFID модуль RC522 з карткою доступу для Arduino [17]

Основні параметри:

– робоча частота: 13,56 МГц;

– інтерфейс передачі даних: (англ. Serial Peripheral Interface, SPI);

– напруга живлення: 3,3 В;

– дистанція зчитування: до 50 мм (залежить від типу картки та навколишніх умов);

– сумісні картки: MIFARE Classic 1K, MIFARE Classic 4K, MIFARE Ultralight, MIFARE DESFire та інші;

– функціонал: зчитування та запис інформації;

– сфери застосування: контроль доступу, автоматична ідентифікація, робототехніка, платіжні системи.

Принцип дії:

RC522 створює високочастотне електромагнітне поле за допомогою вбудованої антени. Коли RFID-картка потрапляє в зону дії (2–5 см), вона активується завдяки індукованій енергії та передає свої дані модулю бездротовим способом. Модуль здатен як зчитувати унікальний ідентифікатор картки (англ. Unique Identifier, UID), так і записувати дані в її внутрішню пам’ять.

Підключення до Arduino:

Модуль підключається до плати Arduino через SPI-інтерфейс. Типове підключення для Arduino Uno виглядає так:

– VCC: 3,3 В;

– GND: земля;

– RST: пін 9 (програмно конфігурується);

– SDA (SS): пін 10;

– MOSI: пін 11;

– MISO: пін 12;

– SCK: пін 13;

– IRQ: зазвичай не використовується.

Приклад програмування RC522:

Після успішного підключення модуля до платі Arduino необхідно його програмувати, строго під наші завдання. Цей процес може зайняти не один час. Все звичайно залежить від ваших завдань. Приклад підключення (рис. 2.2).

Для роботи з RC522 на Arduino використовується бібліотека MFRC522, яка спрощує зчитування та запис даних. Вона забезпечує зручний інтерфейс для зчитування та запису даних на RFID-карти, які відповідають стандарту ISO/IEC 14443 A/MIFARE, наприклад, MIFARE Classic 1K, MIFARE Classic 4K, MIFARE Ultralight тощо. Написана на мові C++, бібліотека орієнтована на Arduino, але може бути адаптована для інших мікроконтролерів.

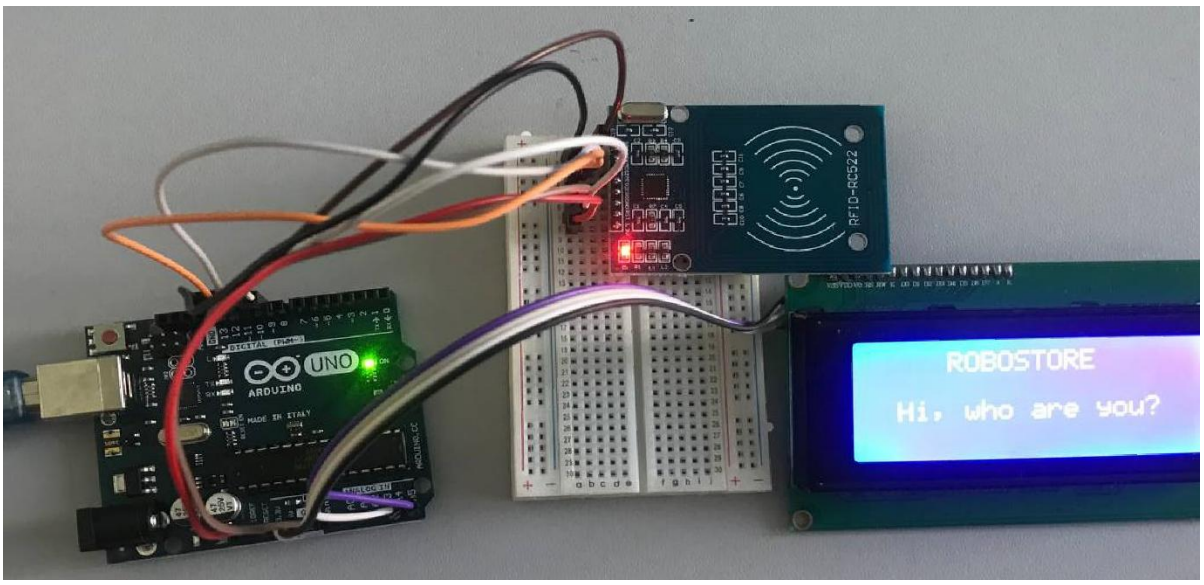

Рисунок 2.2 – Типова установка для проєкту з використанням Arduino, RC522 та РК-дисплея [22]

```
void loop() {
  if (rfid.isCard()) {
    if (rfid.readCardSerial()) {
      rfidcard = String(rfid.serNum[0]) + " " + String(rfid.serNum[1]) + " " + String(rfid.serNum[2]) + " " + String(rfid.serNum[3]);
      Serial.println(rfidcard);
      if (rfidcard == "76 43 117 51") {
        lcd.clear();
        lcd.setCursor(5, 0);
        lcd.print("Hi, Mykhailo!");
        lcd.setCursor(4, 2);
        lcd.print("How are you?");
        delay(3000);
        lcd.clear();
      }
      if (rfidcard == "44 146 229 55") {
        lcd.clear();
        lcd.setCursor(5, 0);
        lcd.print("Hi, Dmytro!");
        lcd.setCursor(4, 2);
        lcd.print("How are you?");
        delay(3000);
        lcd.clear();
      }
    }
    rfid.halt();
  }
  lcd.setCursor(5, 0);
  lcd.print("ROBOSTORE");
  lcd.setCursor(2, 2);
  lcd.print("Hi, who are you?");
}
```

Рисунок 2.3 – Приклад програмування RC522

Після прикладення брелока до RC522, програма спочатку розпізнає наявність брелока, а потім зчитає його унікальний ідентифікатор UID. Цей UID буде виведений у послідовний порт для налагодження. Далі програма порівняє зчитаний UID із заздалегідь визначеними значеннями. Якщо UID брелока збігається з одним із них, РК-дисплей очиститься, виведе персоналізоване

привітання («Hi, Mykhailo!» або «Hi, Dmytrol!») (рис. 2.4) та запитання «How are you?», почекає 3 секунди, знову очистить дисплей, а потім повернеться до постійного відображення початкового привітання «ROBOSTORE» та «Hi, who are you?», очікуючи наступний брелок (рис. 2.3).

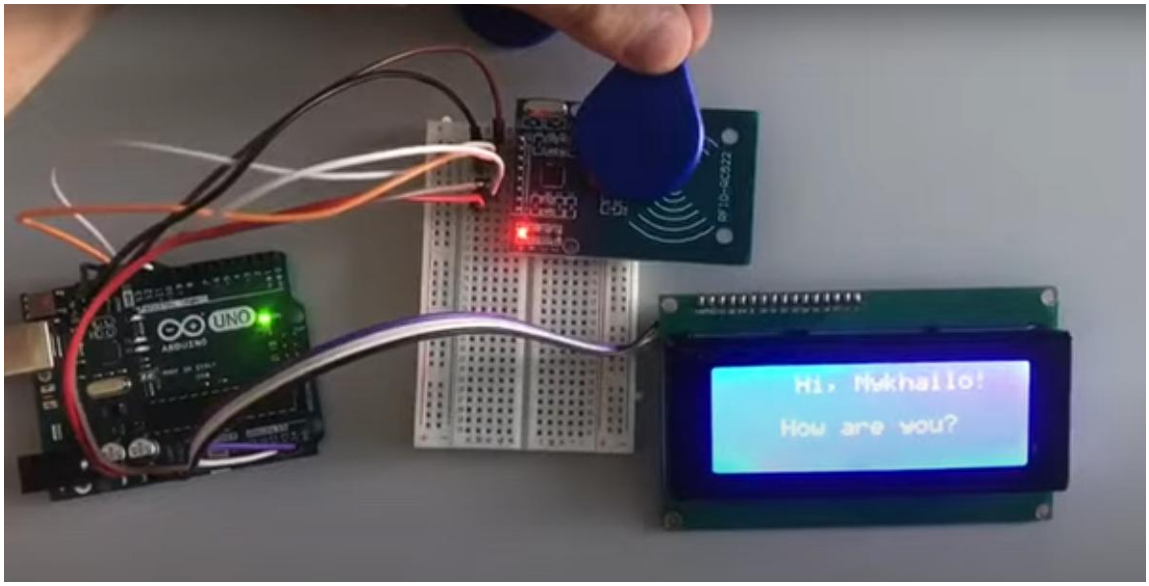

Рисунок 2.4 – Результат роботи RFID-модуля та виведення персонального привітання [22]

Цей приклад є базовим показом роботи RFID-модуля RC522, його можна адаптувати під інші потреби.

2.1.2 WiFi-плата NodeMCU V2 ESP8266 та приклад його програмування

NodeMCU V2 ESP8266 – це плата розробки, що базується на модулі ESP-12E, який містить мікроконтролер ESP8266 з інтегрованими можливостями Wi-Fi (рис. 2.5). Розроблена спеціально для застосувань в IoT, вона є відмінним вибором для СКД на основі RFID завдяки своїй бездротовій функціональності, простоті програмування та доступності. На відміну від окремих модулів ESP8266, NodeMCU V2 включає вбудований перетворювач USB-UART (на базі чипа CH340) та стабілізатор напруги, що дозволяє підключати плату безпосередньо до комп’ютера через USB для живлення та програмування без необхідності додаткового обладнання.

WiFi-плата NodeMCU V2 ESP8266 [16] – це потужне та енергоефективне рішення, здатне ефективно керувати як зчитуванням RFID-міток, так і бездротовою комунікацією, що є критично важливим для системи контролю доступу. Популярність плати в IoT-проектах підкреслюється активною підтримкою спільноти та наявністю великої кількості бібліотек для різноманітних функцій.

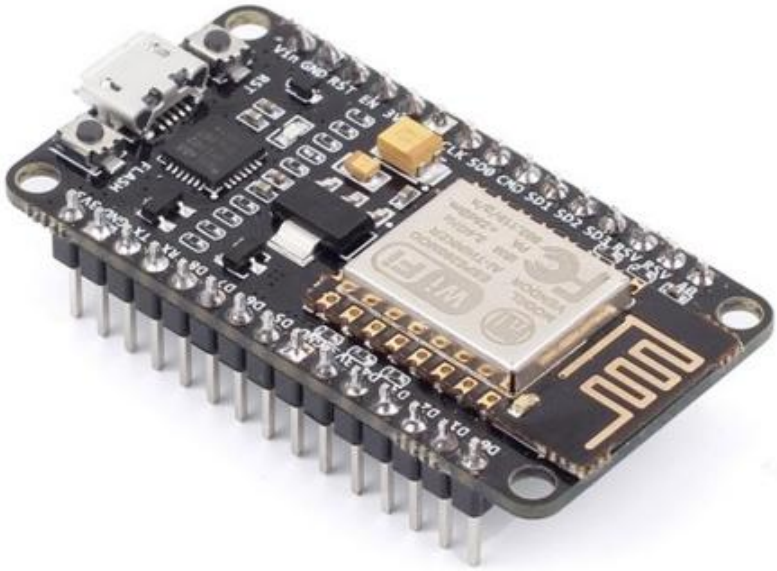

Рисунок 2.5 – WiFi-плата NodeMCU V2 ESP8266 [16]

Опис апаратного забезпечення плати:

– WiFi-плата NodeMCU V2 ESP8266: в основі плати лежить 32-бітний RISC-процесор Tensilica Xtensa LX106 з тактовою частотою 80 МГц, яка може бути підвищена до 160 МГц. Він оснащений 4 Мбайт флеш-пам’яті для зберігання прошивки та програмного коду, а також 80 кбайт оперативної пам’яті для виконання завдань у реальному часі;

– бездротові можливості (Wi-Fi): плата підтримує стандарти IEEE 802.11 b/g/n, дозволяючи їй підключатися до існуючих Wi-Fi мереж або функціонувати як точка доступу. Ця функція є вирішальною для систем контролю доступу, оскільки забезпечує бездротову передачу даних RFID-міток на зовнішній сервер (наприклад, Spring Boot) для аутентифікації;

– універсальні входи/виходи GPIO: NodeMCU V2 надає 17 універсальних пінів вводу/виводу, які можуть бути використані для підключення різних периферійних пристроїв, таких як RFID-зчитувачі, реле для керування замками або додаткові сенсори. Деякі з цих пінів підтримують такі інтерфейси зв’язку, як SPI, I2C та UART, розширюючи можливості комунікації з іншими пристроями;

– живлення: плата може отримувати живлення через порт micro-USB (5 В) або від зовнішнього джерела живлення 5 В через пін VIN. Вбудований стабілізатор напруги автоматично знижує вхідну напругу до необхідних 3,3 В для стабільної роботи мікроконтролера ESP8266, що значно спрощує інтеграцію живлення.

Програмне середовище

Мови програмування та інтегровані середовища розробки (англ. Integrated Development Environment, IDE): Плата NodeMCU V2 ESP8266 надає гнучкість у виборі середовища програмування, підтримуючи:

– Lua-скрипти в поєднанні з прошивкою NodeMCU для швидкого створення прототипів;

– C++ в середовищі Arduino IDE, що користується значною популярністю завдяки великій кількості доступних бібліотек.

Для реалізації функціоналу проєкту знадобляться наступні **ключові бібліотеки:**

– ESP8266WiFi: ця бібліотека є необхідною для встановлення та керування підключенням плати до бездротової мережі Wi-Fi;

– MFRC522: призначена для взаємодії з RFID-зчитувачами, зокрема з модулем MFRC522, дозволяючи зчитувати та записувати дані на RFID-мітки;

– HTTPClient: використовується для формування та надсилання HTTP-запитів, що дозволяє передавати дані, наприклад, зчитані RFID-мітки, на зовнішній сервер Spring Boot для подальшої обробки та перевірки.

Приклад програмування NodeMCU V2 ESP8266:

Розглянемо приклад підключення NodeMCU V2 ESP8266 до модуля AGS10 для вимірювання рівня CO2.

Модуль AGS10 [21] – це передове рішення для виявлення різноманітних газів у навколишньому середовищі (рис. 2.6). Він вирізняється такими ключовими особливостями:

– висока чутливість: AGS10 здатний фіксувати навіть незначні концентрації газів, що робить його ідеальним для ефективного моніторингу небезпечних речовин;

– широкий спектр виявлення газів: Модуль здатен ідентифікувати різні типи газів, включаючи метан, пропан, водень та інші. Це забезпечує його універсальність для широкого кола застосувань;

– висока точність: AGS10 гарантує високу точність вимірювань, надаючи надійні дані для аналізу та контролю;

– низьке енергоспоживання: Однією з найважливіших переваг модуля є його мінімальне споживання енергії, що особливо цінно для бездротових пристроїв та систем, що працюють від батарей;

– тривалий термін служби: Завдяки надійній конструкції та економному споживанню енергії, AGS10 забезпечує довгострокову стабільну роботу, що робить його міцним рішенням для тривалих проєктів.

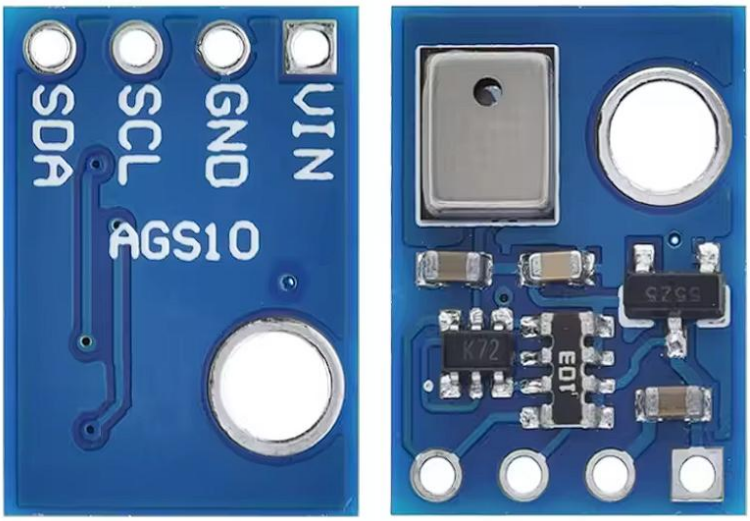

Рисунок 2.6 – AGS10-модуль [21]

Основною перевагою AGS10 перед аналогами є його здатність виявляти широкий спектр газів за мінімального споживання енергії. Це робить його

ідеальним компонентом для бездротових та батарейних IoT-пристроїв, які мають функціонувати тривалий час на одній зарядці.

Цей код підключає NodeMCU до модуля AGS10 для вимірювання рівня CO2 (рис. 2.7).

```
ESP8266WebServer server(80);
void setup() {
  Serial.begin(9600);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Підключення до WiFi...");
  }
  Serial.println("Підключення до WiFi");
  Wire.begin();
  server.on("/", handleRoot);
  server.begin();
}
void loop() {
  server.handleClient();
}
void handleRoot() {
  Wire.beginTransmission(AGS10_address);
  Wire.write(0x02);
  Wire.endTransmission();
  delay(10);
  Wire.requestFrom(AGS10_address, 2);
  uint16_t co2 = 0;
  if (Wire.available()) {
    co2 = Wire.read() << 8 | Wire.read();
  }
}
```

Рисунок 2.7 – Приклад програми керування для плати NodeMCU

Також цей код ініціалізує послідовний порт і з'єднання Wi-Fi, налаштовує I2C для зв'язку з модулем AGS10. В основному циклі код відправляє запит на отримання CO2 даних з модуля AGS10, зчитує отримані дані і виводить їх у серійний порт. Програма також створює веб-сервер, який відповідає на запити кореневого URL і відправляє HTML-сторінку з поточним значенням CO2. Таким чином, при зверненні до IP-адреси NodeMCU веб-сторінка відображатиме поточний рівень CO2, отриманий із модуля AGS10.

2.1.3 Додаткові апаратні компоненти СКД

Оскільки для належного функціонування СКД мікроконтролера та RFID модуля замало, було вирішено додати ще такі апаратні компоненти:

– серводвигун SG90 імітує механізм блокування/розблокування дверей, реагуючи на команди від мікроконтролера (рис. 2.8) [18].

Рисунок 2.8 – Серводвигун SG90 2кг 180 [18]°

– комплект перемичок Arduino (120 шт.) використовується для з’єднання всіх компонентів на макетній платі (рис. 2.9) [20].

Рисунок 2.9 – Комплект перемичок мама-мама, тато-тато, мама-тато 120шт. 20см [20]

– USB зарядний пристрій 5В 1А (або більше): забезпечує стабільну роботу сервоприводу SG90 (рис. 2.10) [19].

Рисунок 2.10 – Мережевий зарядний пристрій 5В USB 1А [19]

– міні-дисплей LCD I2C QC1602A (16 символів, 2 рядки) [14] відображає цифри та символи, які були введені через клавіатуру, та іншу службову інформацію. Підключений через інтерфейс I2C (рис. 2.11).

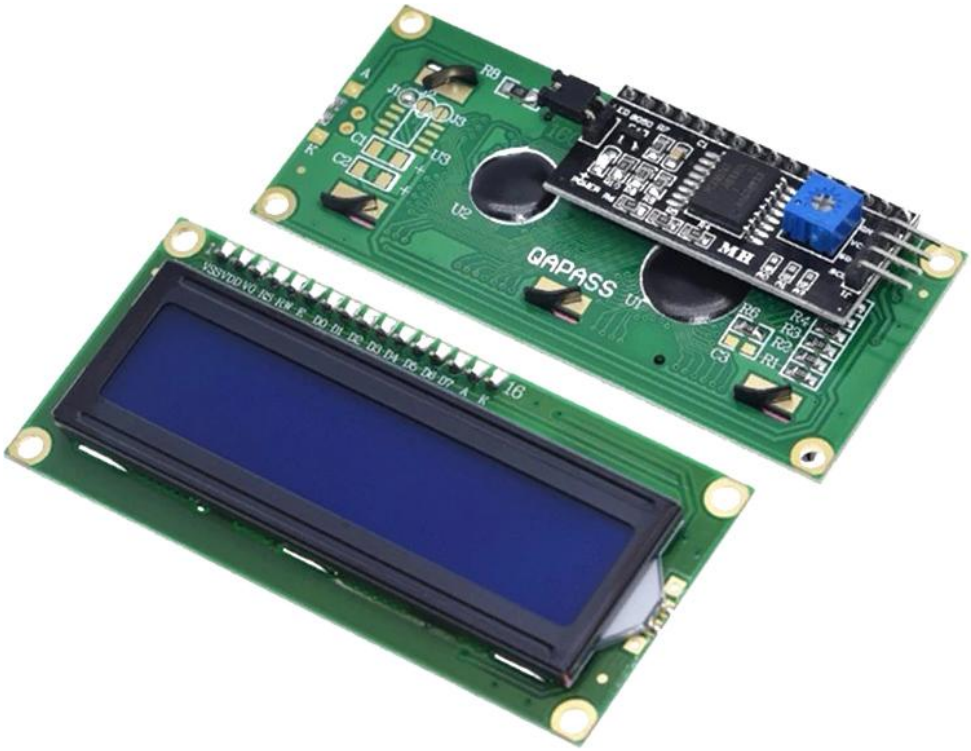

Рисунок 2.11 – LCD 1602 I2C символьный дисплей 16×2 (синій) [14]

– клавіатура матрична 4×4 [15] дозволяє вводити цифри та символи вручну. Підключена до NodeMCU через 8 цифрових пінів GPIO (рис. 2.12);

Рисунок 2.12 – Клавіатура матрична 4×4 [15]

– безпайкова макетна плата MB-102 на 830 пінів [23] для зручного з’єднання компонентів без пайки, особливо на етапі прототипування (рис. 2.13);

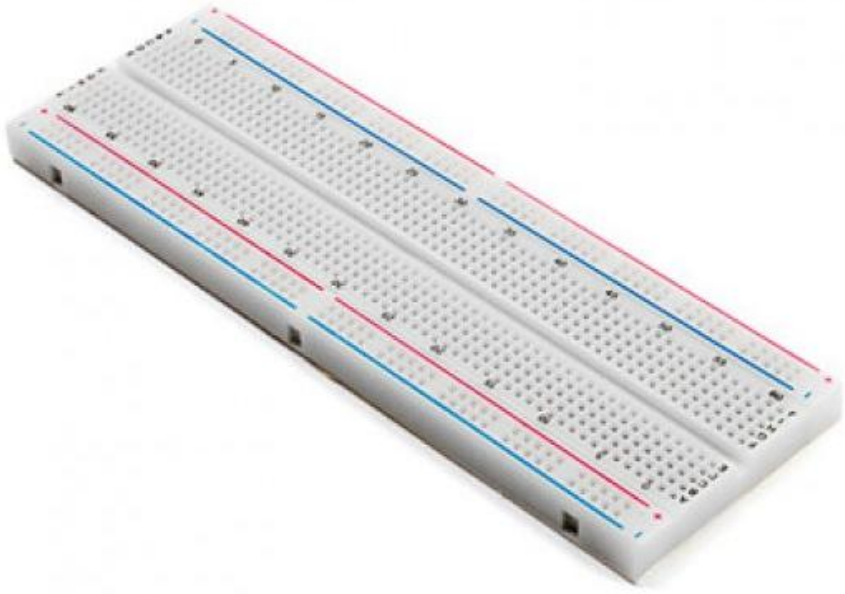

Рисунок 2.13 – Безпайкова макетна плата MB-102 на 830 пінів [23]

– USB-кабель Micro-USB [24] для підключення та програмування плати ESP8266 NodeMCU (рис. 2.14);

Рисунок 2.14 – USB-кабель Micro-USB [24]

– модуль розширення портів PCF8575 [27] для економії портів WiFi-плати (рис. 2.15);

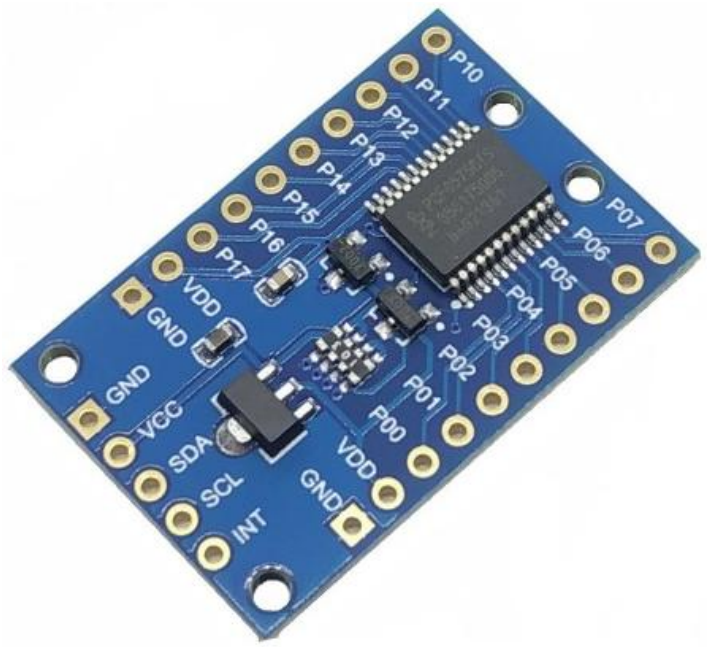

Рисунок 2.15 – Модуль розширення портів PCF8575 [27]

– плата-перехідник гніздо microUSB на DIP [28], вона дозволяє підключати microUSB-кабель до макетної плати без пайки – через стандартні контакти DIP-формату (рис. 2.16).

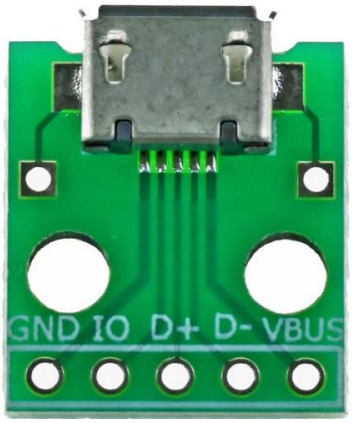

Рисунок 2.16 – Плата-перехідник гніздо microUSB на DIP [28]

Додаткові компоненти були підібрані, опираючись на всі функціональні та нефункціональні вимоги.

Для оптимізації та зручності процесу закупівлі всіх необхідних компонентів для проєкту був розроблений детальний кошторис, мета якого – систематизувати перелік апаратних засобів, забезпечити точне бюджетування та уникнути непередбачених витрат, підвищити ефективність пошуку постачальників та порівняння цін, а також слугувати інструментом контролю за виконанням закупівель та офіційною документацією для звітності.

Таблиця 2.1 – Кошторис компонентів СКД

№	Найменування	К-сть	Од. вим.	Ціна, грн	Вартість, грн
1	Зарядний мережевий пристрій 5 В USB 1 А	3	шт.	110,00	330,00
2	Клавіатура матрична 4×4	1	шт.	111,00	111,00
3	Комплект перемичок мама-мама, тато-тато, мама-тато 120 шт. 20 см	1	уп.	119,00	119,00
4	Міні-дисплей символьний LCD I2C QC1602A	1	шт.	124,00	124,00
5	Модуль RFID RC522	1	шт.	78,00	78,00

№	Найменування	К-сть	Од. вим.	Ціна, грн	Вартість, грн
6	Модуль розширення портів PCF8575	1	шт.	163,00	163,00
7	Плата макетна безпайкова MB-102 на 830 пінів	1	шт.	80,00	80,00
8	Плата Wi-Fi NodeMCU V2 ESP8266	1	шт.	245,00	245,00
9	Плата-перехідник гніздо microUSB на DIP	3	шт.	45,00	135,00
10	Серводвигун SG90 2 кг 180°	1	шт.	79,00	79,00
11	USB-кабель Micro-USB	1	шт.	99,00	99,00
				Разом	**1563,00**

Кошторис потрібен для вирішення таких ключових проблем: систематизація, бюджетування, ефективність закупівлі, контроль, документація.

2.2 Програмні компоненти

Розробка сучасної інформаційної системи, особливо СКД, неможливе без ретельно продуманого програмного забезпечення. Програмні компоненти є «мозком» системи, що забезпечує її функціональність, надійність, безпеку та масштабованість. Вони перетворюють взаємодію апаратних засобів на логічні операції та бізнес-процеси, отримуючи дані від фізичних пристроїв, обробляючи їх відповідно до заданих правил, взаємодіючи з базами даних для зберігання та отримання інформації, а також надаючи інтерфейси для користувачів та інших систем.

У цьому підрозділі буде детально розглянено ключові програмні компоненти, обрані для реалізації обраної СКД, буде обґрунтовано їхній вибір та пояснено їхню роль у забезпеченні основних функцій проєкту.

2.2.1 Фреймворк Spring Boot

Spring Boot – це потужний та інноваційний фреймворк з екосистеми Spring Framework, розроблений для швидкої побудови автономних, готових до використання застосунків на Java. Його головна перевага – мінімізація конфігурації, що дозволяє розробникам зосередитись на бізнес-логіці, а не на налаштуваннях інфраструктури.

Spring Boot автоматизує рутинні завдання, такі як налаштування серверів застосунків, управління залежностями та конфігурація. Це досягається завдяки підходу «конвенція над конфігурацією», вбудованим серверам, автоматичній конфігурації та стартовим залежностям, які спрощують інтеграцію з популярними бібліотеками та технологіями.

Фреймворк Spring Boot широко застосовується в сучасній розробці програмного забезпечення завдяки своїй гнучкості, потужності та здатності швидко створювати надійні та масштабовані застосунки. Основні сфери його застосування:

– розробка RESTful Application Programming Interface (API) та мікросервісів: Spring Boot є фактичним стандартом для створення RESTful веб-сервісів та мікросервісної архітектури завдяки своїй легкості та швидкому запуску;

– вебзастосунки: Активно використовується для розробки традиційних вебзастосунків, як з використанням шаблонних рушіїв (Thymeleaf, FreeMarker), так і як бекенд для односторінкових застосунків;

– пакетні та консольні застосунки: Завдяки вбудованому контейнеру, Spring Boot ідеально підходить для створення автономних інструментів, що виконуються за розкладом або на вимогу;

– системи обробки даних: Його інтеграція з базами даних та іншими технологіями дозволяє ефективно створювати системи для збору, обробки та аналізу даних;

– хмарні застосунки: Spring Boot відмінно інтегрується з хмарними платформами (AWS, Azure, Google Cloud), спрощуючи розгортання та масштабування.

Загалом, Spring Boot – це універсальний інструмент для Java-розробників, що прискорює цикл розробки, підвищує продуктивність та допомагає створювати надійні й підтримувані застосунки.

Використання Spring Boot у даній КБР

У цій КБР Spring Boot відіграватиме ключову роль як центральна серверна частина СКД. Він виконуватиме функцію бекенд-сервісу, що обробляє всі запити, пов’язані з авторизацією доступу, управлінням даними та веденням журналу подій.

Основні функції Spring Boot у контексті цієї роботи включатимуть:

– прийом даних від плати (ESP8266): Spring Boot реалізує RESTful API ендпоінти, які прийматимуть HTTP POST-запити від ESP8266. Ці запити міститимуть ідентифікатор RFID-мітки, зчитаної на точці доступу;

– логіка авторизації доступу: Отримавши ID мітки, Spring Boot виконуватиме запити до бази даних (інтеграція з якою буде забезпечена завдяки Spring Data JPA або аналогічним технологіям). Система перевірятиме наявність такої мітки у базі даних, її статус, права доступу та, можливо, часові обмеження. На основі цієї перевірки буде формуватися відповідь про дозвіл або відмову в доступі;

– ведення журналу подій: Кожна спроба доступу (як дозволена, так і відмовлена) буде фіксуватися та зберігатися в базі даних. Spring Boot забезпечить функціонал для запису часу, ID мітки, статусу доступу та інших релевантних даних;

– взаємодія з базою даних: Завдяки вбудованим можливостям Spring Boot для роботи з базами даних (наприклад, через HikariCP для пулу з’єднань, Spring Data JPA для ORM), сервіс ефективно взаємодіятиме з реляційною базою даних для зберігання та керування даними про користувачів, RFID-мітки та журнал доступу;

– надання адміністративного API: Проект передбачає веб-інтерфейс для адміністратора, Spring Boot також надаватиме окремі RESTful ендпоінти для керування користувачами, додавання/видалення RFID-міток, перегляду журналу подій та інших адміністративних функцій;

– легкість розгортання: Автономні виконувані JAR-файли, створені Spring Boot, дозволять легко розгорнути серверну частину на будь-якій платформі з встановленою Java Virtual Machine, забезпечуючи гнучкість у виборі середовища виконання.

Вибір Spring Boot для серверної частини обґрунтовується його високою продуктивністю, гнучкістю, простотою розробки та підтримки, що є критично важливим для створення надійної та масштабованої системи контролю доступу.

2.2.2 Архітектура RESTful API

RESTful API – це програмний інтерфейс, що використовує архітектурний стиль REST для зв’язку між різними програмними системами через інтернет. RESTful API функціонує на базі стандартних HTTP-методів (*GET, POST, PUT, DELETE*) для маніпулювання ресурсами, які ідентифікуються за допомогою URL-адрес. Ці ресурси можуть бути даними, функціями чи іншими об’єктами, що передаються у форматах, таких як JSON або XML. Основні принципи RESTful API включають безстанність (кожен запит містить всю необхідну інформацію), кешування для підвищення продуктивності та уніфікований інтерфейс для спрощення взаємодії.

RESTful API широко застосовуються в різних сферах:

– вебзастосунки: соціальні мережі, електронна комерція та системи управління контентом використовують RESTful API для інтеграції з іншими сервісами;

– мобільні застосунки: для отримання даних із серверів, наприклад, профілів користувачів або новин;

– IoT: пристрої IoT застосовують RESTful API для обміну даними з центральними серверами;

– хмарні сервіси: постачальники, такі як AWS, Google Cloud і Microsoft Azure, надають RESTful API для управління ресурсами;

– фінансові послуги: використовуються для обробки платежів, управління рахунками та доступу до транзакцій;

– охорона здоров’я: застосовуються для управління даними пацієнтів, планування прийомів і телемедицини.

Ці API популярні завдяки своїй простоті, масштабованості та сумісності з різними платформами.

Процес оброблення запитів

Spring Boot обробляє запити через серію кроків, починаючи з клієнта, який надсилає HTTP-запит (наприклад, GET, POST). Запит приймає DispatcherServlet, який є центральним елементом для керування. Потім система визначає відповідний контролер за допомогою Handler Mapping, а контролер виконує бізнес-логіку, часто взаємодіючи з базою даних. Нарешті, відповідь повертається клієнту через DispatcherServlet (рис. 2.15).

У даній КБР Spring Boot буде використовуватися для створення REST API, яке прийматиме дані RFID від ESP8266, перевірятиме їх у базі даних PostgreSQL і повертати рішення про доступ. Це забезпечить безпечну та ефективну обробку запитів, інтегруючи апаратну частину з серверною логікою.

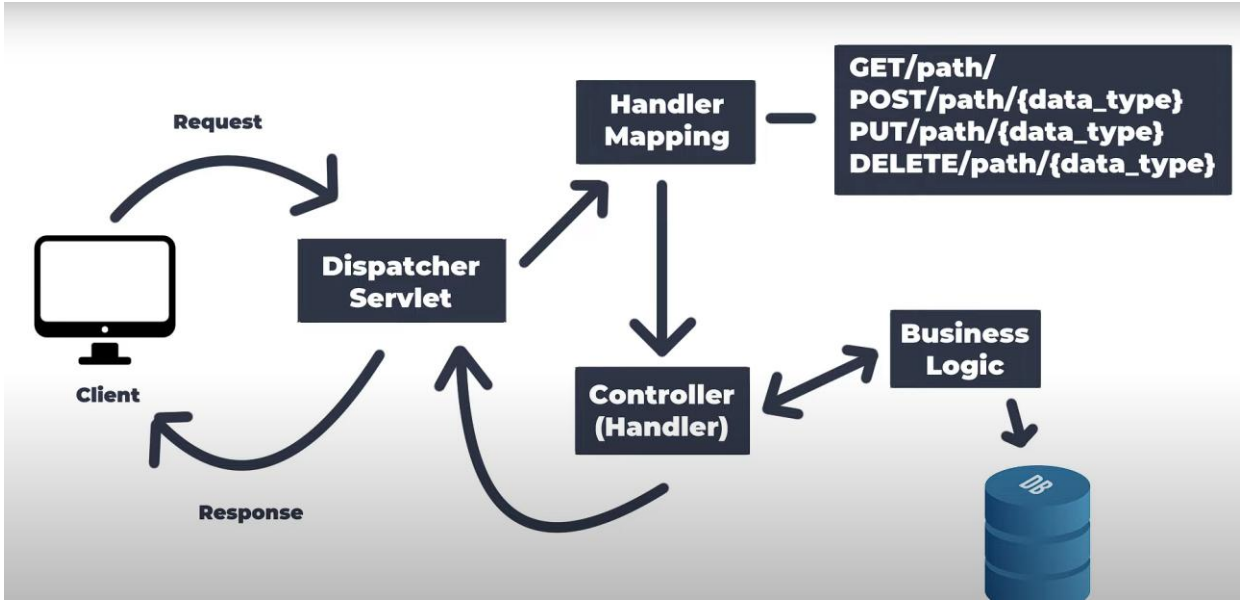

Рисунок 2.15 – Схематичне зображення логіки обробки запитів [26]

Використання RESTful API у даній КБР

У цій роботі RESTful API є ключовим елементом, що забезпечує зв’язок між апаратною частиною (плата ESP8266 з RFID-читачем) та програмною частиною (сервер на Spring Boot із базою даних PostgreSQL). RESTful API буде реалізовано за допомогою Spring Boot для обробки запитів від апаратного забезпечення та виконання операцій із даними.

Основні аспекти використання:

– аутентифікація та авторизація: коли користувач підносить RFID-картку до модуля RC522, ESP8266 зчитує унікальний ідентифікатор картки та надсилає його на сервер через POST-запит до кінцевої точки,. Сервер перевіряє ID у базі даних PostgreSQL і повертає JSON-об’єкт, який вказує, чи дозволено доступ. Ця відповідь використовується ESP8266 для керування сервоприводом SG90, що відкриває або блокує замок;

– керування даними користувачів: RESTful API надаватиме кінцеві точки для адміністрування системи;

– моніторинг і журнали: API може включати кінцеві точки, для отримання історії доступу (наприклад, хто і коли намагався отримати доступ). Це корисно для моніторингу безпеки та аналізу подій;

– керування конфігурацією: хоча IP-адреса сервера вводиться через клавіатуру 4×4 і відображається на LCD-дисплеї, RESTful API може надавати кінцеву точку, для перевірки конфігурації сервера або оновлення системних налаштувань.

RESTful API, реалізований у Spring Boot, забезпечить надійну, безпечну та масштабовану взаємодію між апаратною та програмною частинами системи. Використання JSON як формату даних спростить обробку запитів і відповідей, а інтеграція зі Spring Security дозволить захистити критичні кінцеві точки, такі як адміністрування користувачів.

2.2.3 База даних PostgreSQL

PostgreSQL – це потужна відкрита система керування реляційними базами даних, яка підтримує стандартну мову SQL та пропонує широкий спектр функцій для безпечного зберігання й масштабування складних даних. Вона відома своєю надійністю, багатосторонністю та високою продуктивністю, що робить її однією з найпопулярніших баз даних у світі. PostgreSQL є високомасштабованою системою, здатною обробляти великі обсяги даних і велику кількість одночасних користувачів, що підтверджується її використанням у системах, які керують терабайтами даних, а в деяких випадках – навіть петабайтами. Вона дотримується принципів атомарності, узгодженості, ізольованості, тривалості, що забезпечує високу надійність і цілісність даних.

Серед ключових особливостей PostgreSQL:

– підтримка розширених типів даних: JSON, масиви, hstore, що дозволяє працювати з неструктурованими даними;

– повнотекстовий пошук: для швидкого пошуку текстових даних.

– геопросторові функції: завдяки розширенню PostGIS, яке використовується в геоінформаційних системах;

– реплікація та висока доступність: підтримка потокової та логічної реплікації для забезпечення безперебійної роботи;

– розширюваність: можливість створення власних типів даних, операторів і функцій;

– інтеграція з іншими системами: через Foreign Data Wrappers для доступу до зовнішніх джерел даних.

PostgreSQL широко застосовується в багатьох галузях завдяки своїй універсальності та гнучкості: її використовують для корпоративних застосункі (таких як CRM, ERP та системи управління персоналом), вебзастосунків (для e-commerce платформ та систем управління контентом, як-от Django), геоінформаційних систем завдяки розширенню PostGIS для роботи з просторовими даними, у сферах аналітики та бізнес-аналітики для обробки

великих обсягів даних у реальному часі, а також у фінансових послугах та охороні здоров’я для забезпечення надійності транзакцій та безпеки даних.

Відкритий код PostgreSQL усуває ліцензійні обмеження, а сильна підтримка спільноти та детальна документація роблять її доступною для розробників.

Використання PostgreSQL у даній КБР

У контексті даної КБР база даних PostgreSQL відіграє ключову роль у зберіганні та управлінні даними, необхідними для функціонування системи. Вона забезпечує надійне зберігання інформації та швидкий доступ до неї, що є критично важливим для обробки запитів у реальному часі.

Основні аспекти використання PostgreSQL у системі:

– зберігання даних користувачів: PostgreSQL зберігатиме інформацію про авторизованих користувачів, включаючи їхні унікальні RFID-ідентифікатори. Це дозволяє швидко перевіряти, чи є RFID-тег авторизованим, коли користувач підносить картку до модуля RC522;

– журнали доступу: система записуватиме всі спроби доступу, включаючи часові мітки, RFID-теги та результати (дозволено чи заборонено). Ці дані корисні для моніторингу безпеки та аналізу подій, наприклад, виявлення несанкціонованих спроб доступу;

– інтеграція з Spring Boot: PostgreSQL інтегрується з сервером Spring Boot через Spring Data JPA, що спрощує операції з базою даних, такі як створення, читання, оновлення та видалення даних. Це дозволяє серверу швидко отримувати дані з PostgreSQL і повертати відповідь через REST API до ESP8266;

– реальний час і надійність: PostgreSQL забезпечує швидкі відповіді на запити, що є критично важливим для вашої системи, де користувач очікує миттєвого рішення про доступ після піднесення RFID-картки. ACID-сумісність гарантує, що транзакції, такі як запис журналу доступу, виконуються надійно, навіть при одночасних запитах від кількох користувачів;

– масштабованість: якщо буде потрібно розширитися, наприклад, до кількох точок доступу або великої кількості користувачів, PostgreSQL підтримує

потокову реплікацію та кластеризацію, що забезпечує високу доступність і продуктивність.

PostgreSQL є ідеальним вибором для вашої системи керування доступом завдяки своїй надійності, швидкості та гнучкості. Вона забезпечує безпечне зберігання даних користувачів, журналів доступу та конфігураційних налаштувань, а також підтримує швидкі транзакції в реальному часі. Інтеграція зі Spring Boot через Spring Data JPA спрощує розробку та забезпечує ефективну взаємодію між сервером і базою даних. У разі розширення системи PostgreSQL може масштабуватися для підтримки більших навантажень, що робить її універсальним рішенням для вашого проєкту.

2.2.4 Технологія контейнеризації Docker

Docker – це відкрита платформа, призначена для розробки, доставки та запуску застосунків у контейнерах. Контейнери є легковажними, автономними пакетами, що містять код програми, середовище виконання, системні бібліотеки, інструменти та налаштування, необхідні для її функціонування. На відміну від віртуальних машин, контейнери використовують ядро операційної системи хоста, що робить їх ефективнішими за споживанням ресурсів. Docker дозволяє розробникам створювати, тестувати та розгортати застосунки в однаковому середовищі, усуваючи проблеми, пов’язані з різними конфігураціями операційних систем чи залежностей.

Основним елементом є Docker-образи – це попередньо налаштовані шаблони, що містять усе необхідне для коректної та швидкої роботи застосунку, зокрема бібліотеки, сам код, усі залежності та зроблені налаштування. Ці образи використовуються для створення контейнерів – ізольованих виконуваних одиниць, у яких запускається застосунок (рис. 2.16).

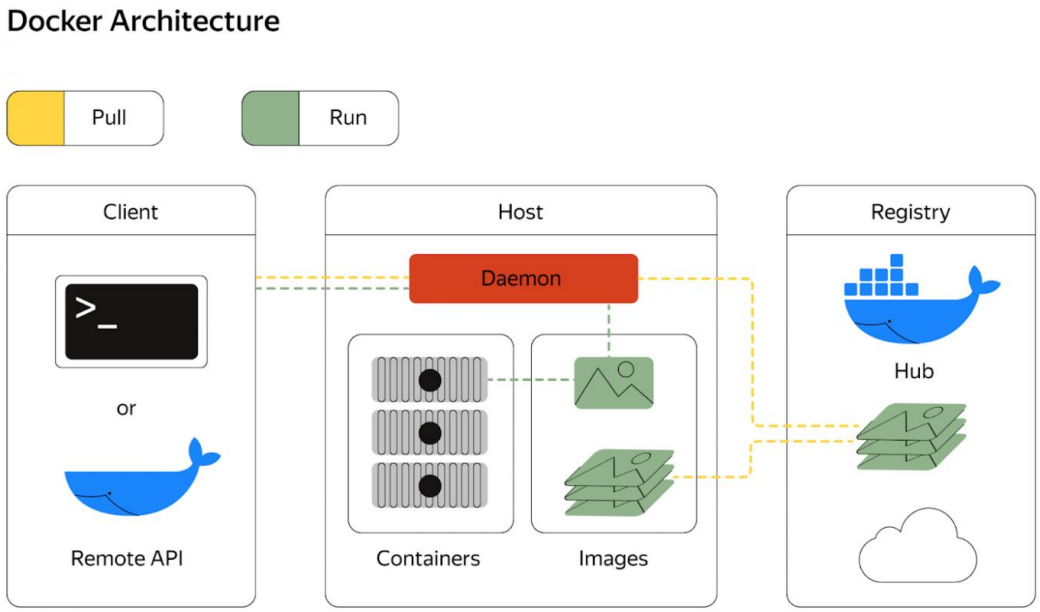

Рисунок 2.16 – Архітектура Docker [25]

Основні особливості Docker включають:

– портативність: контейнери працюють однаково на будь-якій системі з встановленим Docker, від локальних машин до хмарних серверів;

– ізоляція: кожен контейнер працює в ізольованому середовищі, що запобігає конфліктам між програмами чи їхніми залежностями;

– масштабованість: Docker дозволяє легко запускати кілька екземплярів контейнерів для обробки підвищеного навантаження;

– автоматизація розгортання: інструменти, такі як Docker Compose, дозволяють визначати та запускати багатоконтейнерні застосунки за допомогою одного файлу конфігурації;

– керування версіями: контейнери є незмінними, що спрощує відстеження версій і повернення до попередніх станів.

Docker широко використовується в різних галузях:

– веброзробка: для розгортання вебзастосунків, таких як Spring Boot, разом із базами даних, наприклад, PostgreSQL;

– мікросервісна архітектура: для ізоляції окремих сервісів у великих системах;

– хмарні сервіси: у платформах, таких як AWS, Google Cloud і Azure, для швидкого розгортання та масштабування застосунків;

– тестування та CI/CD: для створення ізольованих середовищ для тестування та автоматизації розгортання;

– IoT: для запуску програм на пристроях із обмеженими ресурсами, таких як мікроконтролери.

Використання Docker у КБР

У КБР технологія Docker буде використана для пакування, розгортання та управління серверною частиною системи, яка включає Spring Boot застосунок та базу даних PostgreSQL. Це дозволить значно спростити демонстрацію проекту та забезпечити консистентність середовища.

Основні аспекти використання:

– контейнеризація Spring Boot застосунку: Spring Boot застосунок, який реалізує RESTful API для обробки запитів від мікроконтролера ESP8266, пакується в окремий контейнер. Цей контейнер містить усі необхідні залежності, такі як Java Runtime Environment і бібліотеки Spring Boot, що усуває потребу в ручному налаштуванні середовища;

– контейнеризація PostgreSQL: База даних PostgreSQL запускається в окремому контейнері, використовуючи офіційний образ postgres із Docker Hub. Це дозволяє швидко налаштувати базу даних без встановлення PostgreSQL на хост-системі. Дані зберігаються в Docker-томі для забезпечення їхньої стійкості;

– керування за допомогою Docker Compose: Docker Compose використовується для визначення та запуску обох контейнерів (Spring Boot і PostgreSQL) через єдиний файл конфігурації docker-compose.yml.

Переваги для управління

Використання Docker у проєкті надає значні переваги, забезпечуючи простоту розгортання, адже дозволяє демонструвати роботу системи на будь-якому комп’ютері, маючи лише встановлений Docker, без необхідності вручну налаштовувати Java, PostgreSQL та їхні залежності. Це також гарантує відтворюваність середовища, оскільки серверна частина працюватиме ідентично

на машині розробника та на демонстраційній, усуваючи потенційні проблеми сумісності. Крім того, Docker забезпечує ізоляцію, що запобігає конфліктам між застосунком Spring Boot, PostgreSQL та іншими програмами на демонстраційній машині, а також спрощує управління серверними компонентами, роблячи їх зупинку, запуск, перезапуск та оновлення значно ефективнішими.

2.3 Архітектура СКД

Архітектура будь-якої складної системи, незалежно від того, чи є вона програмною, апаратною або їхньою інтегрованою сукупністю, становить фундаментальний елемент, який детермінує її структурну організацію, функціональні можливості, механізми взаємодії компонентів та принципи розвитку. Вона репрезентує високорівневе відображення системи, що характеризує її ключові структурні одиниці, їхні кореляції, атрибути, а також засади, що керують її проєктуванням та еволюцією. Проектування архітектури є критично важливим для забезпечення не лише відповідності системи первинним функціональним та нефункціональним вимогам, як-от масштабованість, надійність, безпека та продуктивність, а й для ефективного управління складністю, сприяння реутилізації компонентів, спрощення подальшої підтримки та адаптації до майбутніх змін. Отже, доцільно розроблена архітектура є не просто схематичним зображенням, а стратегічним рішенням, що формує базис для успішної реалізації, розгортання та довготривалого функціонування системи в динамічному середовищі.

Алгоритм взаємодії компонентів

Розроблена система контролю доступу (СКД) використовує двофакторну автентифікацію. Спочатку RFID-модуль зчитує UID картки, який NodeMCU надсилає Spring Boot серверу для первинної верифікації наявності в базі даних. У разі успішної перевірки, NodeMCU запитує введення пароля на РК-дисплеї, який користувач вводить за допомогою клавіатури). Після підтвердження клавішею («ОК»), NodeMCU локально перевіряє пароль. Доступ дозволяється лише якщо UID підтверджений сервером і пароль є коректним, активуючи

сервопривід для відкриття замка; в іншому випадку доступ відхиляється, і серво залишається нерухомим. Мережеві параметри конфігуруються безпосередньо в коді мікроконтролера. Таким чином, апаратна та програмна частини системи працюють у режимі реального часу, забезпечуючи централізоване управління доступом до контрольованих зон.

Схематичний алгоритм роботи СКД, що у розробці (рис. 2.17).

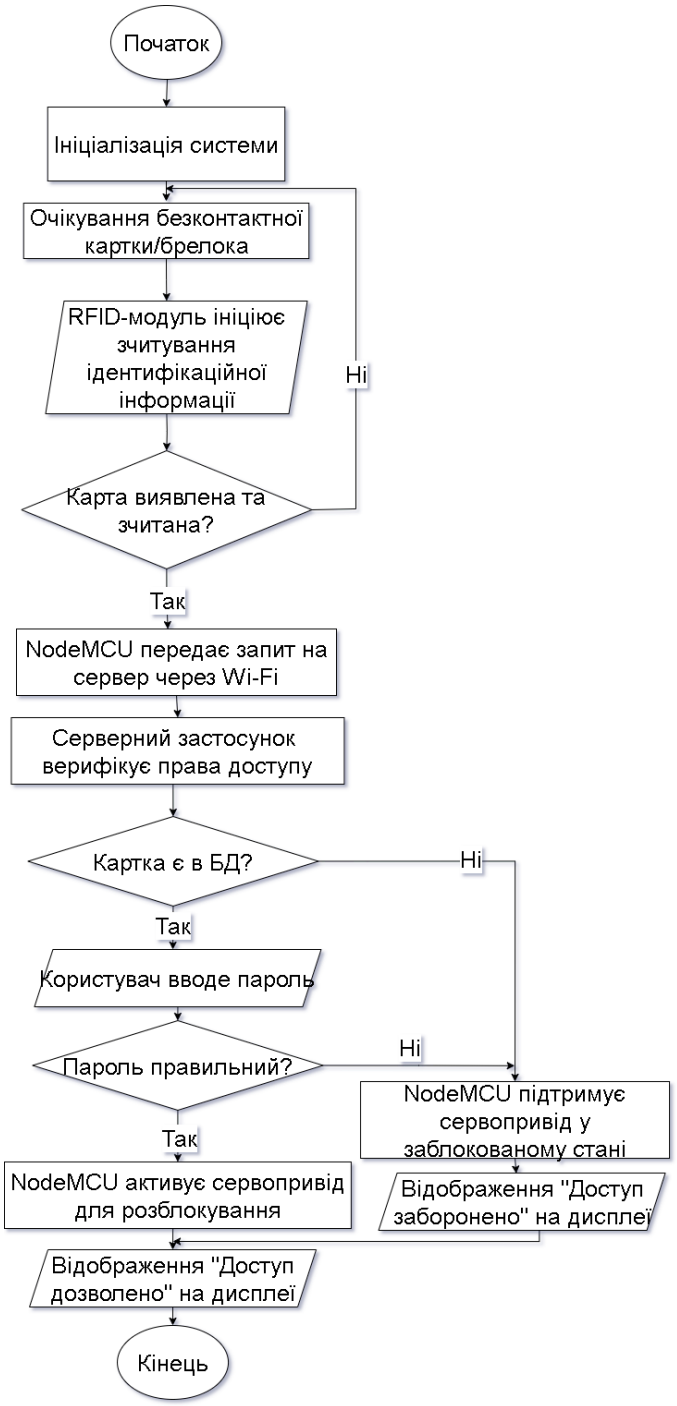

Рисунок 2.17 – Блок-схема алгоритму роботи апаратної частини СКД

Структурна схема (рис. 2.18) зображує ESP8266 як центральний елемент, який підключається до RFID-модуля для зчитування карток, LCD-дисплея для відображення інформації, клавіатури для введення даних та сервоприводу для керування замком. ESP8266 отримує живлення від ноутбука через USB, а сервопривід має окреме джерело живлення (5В 1А). Користувач взаємодіє з системою, підносячи RFID-картку або вводячи дані через клавіатуру.

Програмна частина включає сервер Spring Boot, який надає REST API для обробки запитів, базу даних PostgreSQL для зберігання даних користувачів та журналів, а також Docker для контейнеризації. ESP8266 спілкується з сервером через Wi-Fi, надсилаючи дані RFID для перевірки та отримуючи відповіді про доступ.

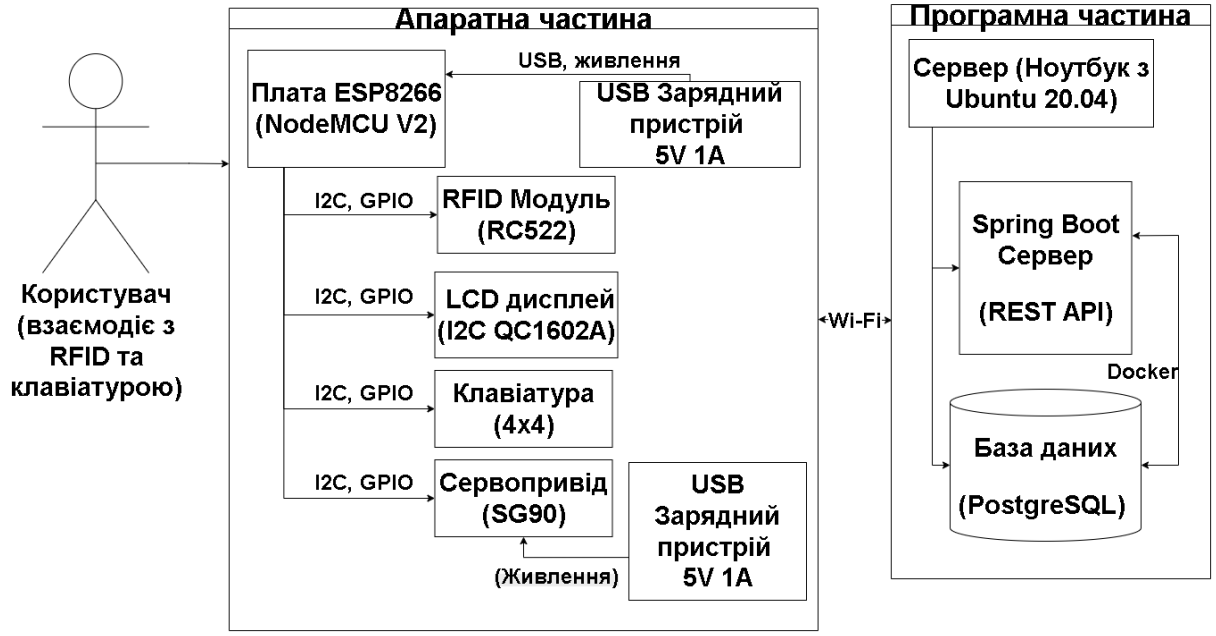

Рисунок 2.18 – Структурна схема СКД

Схема відображає, як користувач взаємодіє з системою, як апаратні компоненти з’єднуються між собою та як вони інтегруються з програмним забезпеченням.

Висновки до розділу 2

У другому розділі представлено детальний опис апаратного та програмного забезпечення, яке реалізує функціональність запропонованої СКД

на базі RFID. Було здійснено обґрунтований вибір компонентів на основі технічних вимог, функціональності та вартості.

До апаратної частини увійшли: RFID-модуль RC522, мікроконтролер NodeMCU V2 ESP8266, сервопривід SG90, дисплей LCD 1602 I2C, клавіатура 4×4, макетна плата та джерела живлення. Всі ці компоненти забезпечують обробку вхідних даних, зчитування RFID-міток, керування виконавчими пристроями (замком), виведення службової інформації та налаштування мережевого підключення. Особливу увагу приділено сумісності компонентів, простоті підключення (через SPI та I2C), енергоефективності та мінімізації загальної вартості – вона становить менше 1600 грн, що відповідає обмеженням, визначеним у розділі 1.

Програмна частина реалізована у вигляді вебзастосунку на базі фреймворку Spring Boot, що виконує роль серверної логіки. Його ключові функції включають обробку HTTP-запитів від мікроконтролера, перевірку ідентифікаторів RFID у базі даних, прийняття рішень про доступ, ведення журналу подій та реалізацію адміністративного інтерфейсу для управління користувачами та правами доступу. Spring Boot обрано завдяки його продуктивності, підтримці RESTful API, інтеграції з базами даних (MySQL або PostgreSQL), вбудованим серверам (Tomcat) та можливостям безпечної автентифікації за допомогою Spring Security.

Серверна і клієнтська частина взаємодіють у режимі реального часу через Wi-Fi. NodeMCU надсилає запит з UID мітки, Spring Boot перевіряє права доступу та надсилає відповідь, на основі якої сервопривід блокує або розблоковує доступ. Така архітектура забезпечує гнучкість, масштабованість і можливість інтеграції з іншими системами в межах IoT-інфраструктури.

Таким чином, у розділі було доведено, що обрана апаратно-програмна конфігурація відповідає сучасним вимогам до СКД, є ефективною, надійною, доступною та готовою до практичного впровадження.

3 РОЗРОБКА ТА ТЕСТУВАННЯ АПК

3.1 Концепт проєкту

Запропонована архітектура СКД, що інтегрує технології RFID та Spring Boot, розроблена з метою створення захищеного, масштабованого та ергономічного рішення для управління фізичним доступом до лімітованих просторів. Основна концепція системи ґрунтується на чіткій делімітації функціональних обов’язків між апаратною та програмною компонентами, що забезпечує оптимальний рівень безпеки, операційної ефективності та адаптивності. У цьому підрозділі буде представлено принцип функціонування системи, її компаративні переваги, ідентифіковані недоліки та потенційні вектори оптимізації.

Принцип функціонування системи

У система імплементовано апаратно-програмний комплекс (АПК), де апаратна частина здійснює взаємодію з користувачем та ініціює механічні операції, тоді як програмна частина відповідає за обробку даних та прийняття рішень щодо авторизації доступу.

Робочий цикл системи включає наступні етапи:

– взаємодія з користувачем: користувач ініціює запит на доступ шляхом представлення RFID-ідентифікатора (картки або брелока) до модуля RC522 або введення конфігураційних даних, наприклад, пароля, через матричну клавіатуру 4×4. Ці операції слугують для ідентифікації суб’єкта або налаштування системних параметрів;

– апаратна обробка даних: плата ESP8266 (NodeMCU V2) здійснює зчитування даних з RFID-модуля або клавіатури. Він інтегрований з LCD-дисплеєм (I2C QC1602A) для візуалізації статусної інформації (наприклад, «Введіть пароль) та з сервоприводом SG90 для управління виконавчим механізмом замка;

– комунікація з серверною платформою: ESP8266 транслює отримані дані (зокрема, унікальний ідентифікатор RFID-картки) по бездротовому каналу

Wi-Fi до сервера на базі Spring Boot, який експонує RESTful API. Сервер верифікує отримані дані відносно інформації, що зберігається в реляційній базі даних PostgreSQL, яка містить відомості про авторизованих користувачів та журнали доступу;

– прийняття рішення та виконання: сервер генерує відповідь (наприклад, «авторизовано» або «відмовлено»), яку ESP8266 використовує для активації сервоприводу (відкриття або блокування замка) та оновлення інформації на LCD-дисплеї.

Дана архітектурна модель забезпечує строгий розподіл функціональних обов’язків: апаратний сегмент (ESP8266 та периферійні пристрої) відповідає за збір даних та механічні операції, тоді як програмний сегмент (сервер Spring Boot та база даних PostgreSQL) здійснює аутентифікацію та персистентне зберігання даних. Важливою особливістю концепції системи є те, що ESP8266 не здійснює локального зберігання конфіденційних даних користувачів, а лише функціонує як транслятор інформації до центрального сервера.

Компаративні переваги системи

Представлене рішення для контролю доступу демонструє низку істотних переваг порівняно з традиційними аналогами:

– підвищений рівень безпеки завдяки безстановому апаратному забезпеченню: на відміну від багатьох конвенційних систем, де ідентифікатори користувачів (наприклад, RFID-мітки або PIN-коди) персистентно зберігаються на локальному пристрої, запропонована система виключає зберігання будь-якої чутливої інформації на апаратному рівні. ESP8266 функціонує виключно як посередник, ретранслюючи дані на сервер Spring Boot для їх обробки. Це значно мінімізує ризик компрометації даних у випадку несанкціонованого фізичного доступу до пристрою, оскільки вся конфіденційна інформація централізовано зберігається на серверній платформі, яка може бути захищена сучасними методами криптографії та контролю доступу;

– масштабованість: архітектурна модель системи надає можливість її легкого масштабування для інтеграції в середовищах з множинними точками

доступу, наприклад, у великих офісних комплексах або кампусах. Кожен новий пристрій на базі ESP8266 підключається до того ж центрального сервера, що усуває необхідність у складних модифікаціях серверної інфраструктури або схеми бази даних. Ця властивість забезпечує економічну ефективність системи при її розширенні;

– гнучкість та потенціал для модернізації: використання стандартизованих технологій, таких як Wi-Fi, Spring Boot та PostgreSQL, гарантує сумісність з іншими системами та відкриває можливості для імплементації нових функціональних можливостей. Наприклад, інтеграція Bluetooth-модуля та розробка мобільного застосунка надасть адміністраторам можливість віддаленого конфігурування пристроїв, що істотно спростить процес налаштування та оновлення системи.

Представлена СКД вирізняється підвищеною безпекою завдяки безстановому апаратному забезпеченню, що унеможливлює зберігання чутливих даних локально, а також високою масштабованістю та гнучкістю через використання стандартизованих технологій для легкого розширення та майбутніх оновлень.

3.2 Опис інтерфейсів апаратного комплексу

3.2.1 Підключення та випробовування плати NodeMCU V2 ESP8266

Для тестування плати спочатку її було підключено через micro-USB до ноутбука для програмування та живлення. Далі за допомогою Arduino IDE було запрограмовано функцію (рис. 3.1), яка підключається до домашнього Wi-Fi.

```cpp
#include <ESP8266WiFi.h>
const char* ssid = "11914";
const char* password = "28092021";
void setup() {
  Serial.begin(115200); // Ініціалізація послідовного порту для виводу інформації
  delay(10);
  Serial.println();
  Serial.print("Підключення до ");
  Serial.println(ssid);
  WiFi.begin(ssid, password); // Спроба підключення до Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print("."); // Виводимо крапки, поки не підключимося
  }
  Serial.println("");
  Serial.println("WiFi підключено!");
  Serial.print("IP-адреса: ");
  Serial.println(WiFi.localIP()); // Виводимо отриману IP-адресу
}
void loop() {

}
```

Output Serial Monitor

Not connected. Select a board and a port to connect automatically.

```
....
WiFi підключено!
IP-адреса: 192.168.1.37
```

Рисунок 3.1 – Функція для випробовування плати

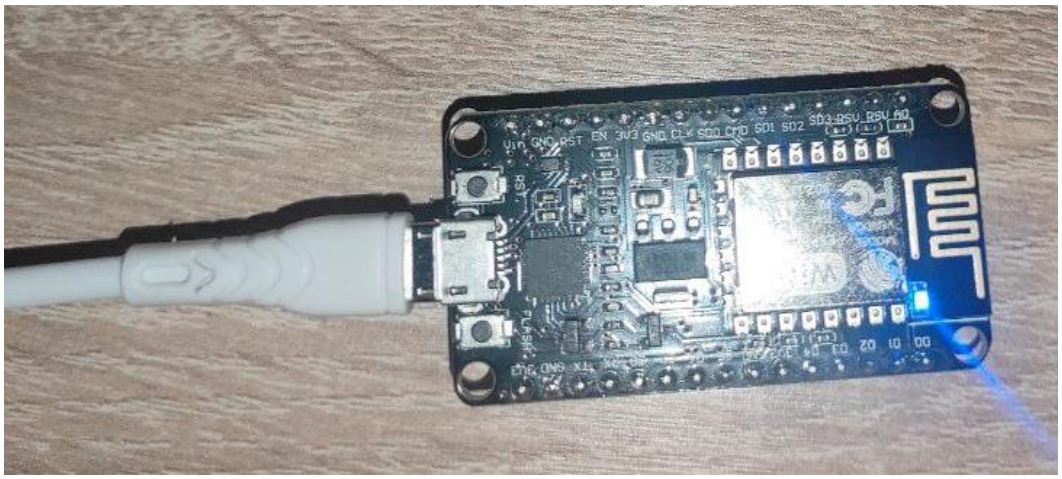

Рисунок 3.2 – Підключення WiFi-плати

Плата ESP8266 успішно випробувана та готова для подальшої роботи.

3.2.2 Підключення та випробовування RFID-модуля RC522

Для тестування RFID-модуля, модуль було підключено до минулої конструкції за допомогою пінів та зашито функцію (рис. 3.3), яка при взаємодії RFID-зчитувача з RFID-міткою повинна зчитувати інформацію з мітки та виводити її в послідовний порт (рис. 3.4).

```
MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance

void setup() {
  Serial.begin(9600);     // Initialize serial communications with the PC
  while (!Serial);        // Do nothing if no serial port is opened (added for Arduinos based on ATMEGA32U4)
  SPI.begin();            // Init SPI bus
  mfrc522.PCD_Init();     // Init MFRC522
  delay(4);               // Optional delay. Some board do need more time after init to be ready, see Readme
  mfrc522.PCD_DumpVersionToSerial(); // Show details of PCD - MFRC522 Card Reader details
  Serial.println(F("Scan PICC to see UID, SAK, type, and data blocks..."));
}

void loop() {
  // Reset the loop if no new card present on the sensor/reader. This saves the entire process when idle.
  if ( ! mfrc522.PICC_IsNewCardPresent()) {
    return;
  }

  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial()) {
    return;
  }

  // Dump debug info about the card; PICC_HaltA() is automatically called
  mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
```

Рисунок 3.3 – Функція для прошивки RFID-модуля

```
Card UID: AA 49 9F 15
PICC type: MIFARE 1KB
Sector Block   0  1  2  3   4  5  6  7   8  9 10 11  12 13 14 15  AccessBits
  15     63  PCD_Authenticate() failed: Timeout in communication.
  14     59  PCD_Authenticate() failed: Timeout in communication.
  13     55  PCD_Authenticate() failed: Timeout in communication.
  12     51  PCD_Authenticate() failed: Timeout in communication.
  11     47  PCD_Authenticate() failed: Timeout in communication.
  10     43  PCD_Authenticate() failed: Timeout in communication.
   9     39  PCD_Authenticate() failed: Timeout in communication.
   8     35  PCD_Authenticate() failed: Timeout in communication.
```

Рисунок 3.4 – Результат прошивки RFID-модуля

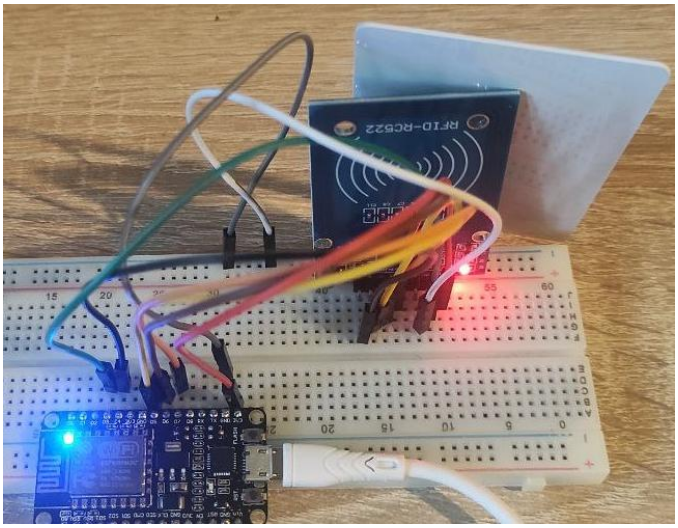

Рисунок 3.5 – Підключення RFID-модуля до плати

RFID-модуль успішно випробуваний та готовий до подальшої роботи (рис. 3.5).

3.2.3 Підключення готового апаратного рішення

До вже готової конструкції плати NodeMCU та RFID-модуля було підключено(рис. 3.6):

– символьний дисплей 16×2(з додатковим живленням 5В);

– серводвигун SG90(з додатковим живленням 5В);

– матрична клавіатура 4×4(за допомогою модуля розширення портів PCF8575).

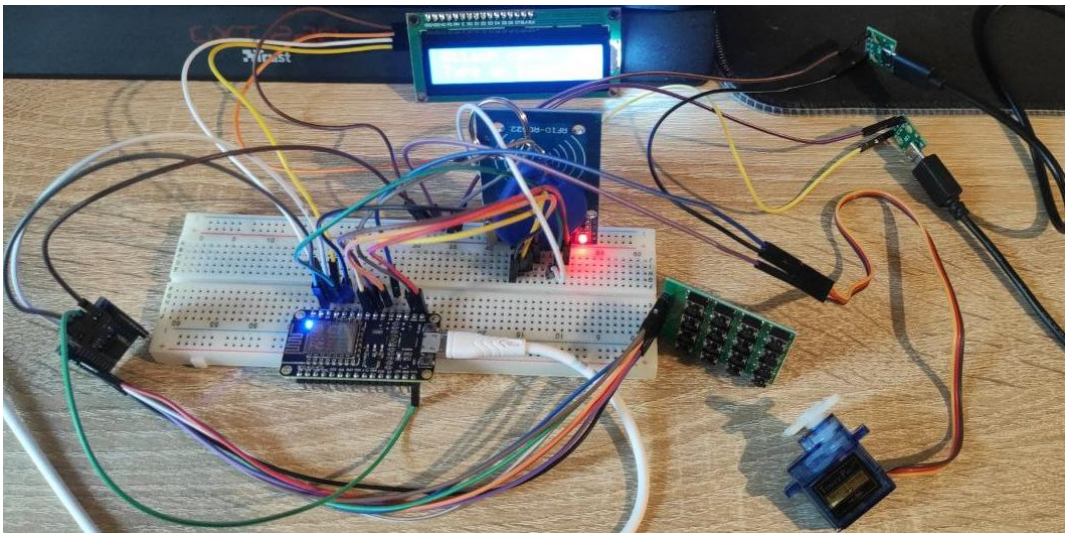

Рисунок 3.6 – Підключення готового апаратного рішення

Для роботи комплектуючих був написаний код, який наведено в додатку А.

3.2.4 Принципова електрична схема обраного рішення

У цьому підрозділі детально розглядається архітектура та фізичне з’єднання всіх апаратних компонентів СКД (рис 3.7).

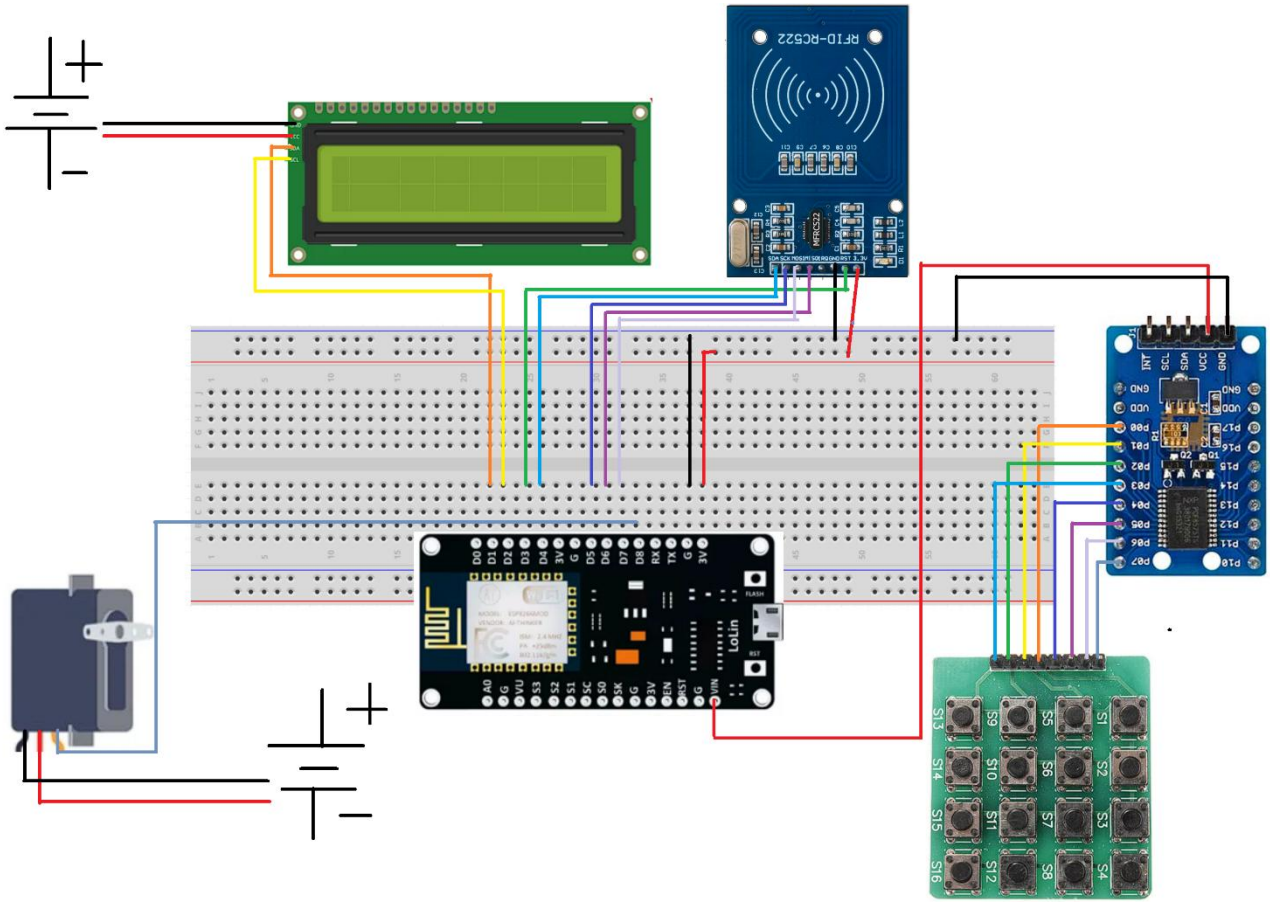

Рисунок 3.7 – Принципова електрична схема обраного рішення

Принципова електрична схема СКД ілюструє взаємозв'язок між центральним мікроконтролером NodeMCU ESP8266 та підключеними периферійними модулями: RFID-зчитувачем MFRC522, матричною клавіатурою 4×4 (через розширювач портів PCF8575), РК-дисплеєм 1602 (з I2C-адаптером) та серводвигуном.

Усі з'єднання здійснюються з дотриманням вимог до живлення та сигнальних ліній, згідно з технічними характеристиками кожного компонента.

3.3 Опис інтерфейсів програмного забезпечення

Цей підрозділ присвячений розробці та верифікації функціоналу серверної частини (бекенду) СКД. Бекенд відповідає за централізоване управління даними і реалізацію бізнес-логіки, що забезпечує функціонування системи у відповідь на запити від мікроконтролера NodeMCU.

3.3.1 Технологічний стек та архітектура

Для реалізації бекенд-системи було обрано фреймворк Spring Boot (версія 3.5.0) на базі Java Development Kit 17. Spring Boot був обраний за його здатність до швидкої розробки автономних, готових до виробництва додатків, що мінімізує конфігурацію та спрощує розгортання. У якості системи управління реляційними базами даних використовується PostgreSQL (версія 16.3). Оркестрація та ізоляція середовища розробки/розгортання забезпечується за допомогою Docker та Docker Compose.

Архітектура бекенду побудована за принципами багатошарового підходу, що включає наступні компоненти:

– модельний шар (Entities): представляє структуру даних, що зберігаються в базі даних. Основним об'єктом є сутність RFIDCard (рис. 3.8), яка містить поля uid (унікальний ідентифікатор RFID-картки) та description (опис картки). Мапінг об'єктів на реляційну базу даних реалізовано за допомогою Java Persistence API та Hibernate ORM;

```java
@Entity
@Table(name = "rfid_cards")
@Data
public class RFIDCard {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "uid", nullable = false, unique = true)
    private String uid;
    @Column(name = "description")
    private String description;
    public RFIDCard() {
    }
    public RFIDCard(String uid, String description) {  no usages
        this.uid = uid;
        this.description = description;
    }
}
```

Рисунок 3.8 – Клас RFIDCard

– репозиторний шар (Repositories): відповідає за взаємодію з базою даних. Використовується Spring Data JPA, що дозволяє визначати інтерфейси

репозиторіїв RFIDCardRepository (рис. 3.9) для виконання операцій CRUD без написання значного обсягу коду. Запити до бази даних генеруються автоматично на основі назв методів;

```
@Repository  3 usages
public interface RFIDCardRepository extends JpaRepository<RFIDCard, Long> {
    Optional<RFIDCard> findByUid(String uid);  2 usages
}
```

Рисунок 3.9 – Клас RFIDCardRepository

– сервісний шар (Services): містить основну бізнес-логіку додатку. Клас RFIDCardService (рис. 3.10) інкапсулює логіку перевірки доступу. Він взаємодіє з репозиторним шаром для отримання даних про картки та визначає, чи надається доступ на основі отриманого UID;

```
public class RFIDCardService {
    private final RFIDCardRepository rfidCardRepository;  3 usages
    @Autowired
    public RFIDCardService(RFIDCardRepository rfidCardRepository) {
        this.rfidCardRepository = rfidCardRepository;
    }
    public boolean checkAccess(String uid) {  1 usage
        Optional<RFIDCard> card = rfidCardRepository.findByUid(uid);
        return card.isPresent();
    }
    public Optional<RFIDCard> getCardByUid(String uid) {  no usages
        return rfidCardRepository.findByUid(uid);
    }
}
```

Рисунок 3.10– Клас RFIDCardService

– контролерний шар (Controllers): реалізує RESTful API, що дозволяє зовнішнім клієнтам (зокрема, NodeMCU) взаємодіяти з бекендом. Клас RFIDCardController (рис. 3.11) обробляє вхідні HTTP-запити, викликає відповідні методи сервісного шару та повертає відповіді у форматі JSON.

```java
public class RFIDController {
    private final RFIDCardService rfidCardService;  2 usages
    @Autowired
    public RFIDController(RFIDCardService rfidCardService) {
        this.rfidCardService = rfidCardService;
    }
    @PostMapping("/checkAccess")
    public ResponseEntity<AccessResponse> checkAccess(@RequestBody AccessRequest requestBody) {
        String uid = requestBody.getUid();
        boolean accessGranted = rfidCardService.checkAccess(uid);

        if (accessGranted) {
            System.out.println("Access granted for UID: " + uid);
            return ResponseEntity.ok(new AccessResponse( accessGranted: true,  message: "Access Granted"));
        } else {
            System.out.println("Access denied for UID: " + uid);
            return ResponseEntity.ok(new AccessResponse( accessGranted: false,  message: "Access Denied"));
        }
    }
```

Рисунок 3.11 – Клас RFIDCardController

Таким чином, обрана архітектура бекенду забезпечує надійну, гнучку та легко масштабовану основу для функціонування СКД, оптимізуючи взаємодію з даними та обробку бізнес-логіки.

3.3.2 Інтеграція з базою даних PostgreSQL

Підключення Spring Boot додатка до бази даних PostgreSQL налаштовано через механізм JDBC (Java Database Connectivity) та пули з'єднань HikariCP. Конфігураційні параметри підключення до бази даних (URL, ім'я користувача, пароль) визначаються за допомогою змінних середовища у файлі docker-compose.yml, що забезпечує гнучкість конфігурації в різних середовищах розгортання та ізоляцію облікових даних.

```
rfid_db=# SELECT * FROM rfid_cards;
 id |      description       |   uid
----+------------------------+----------
  1 | Карта Адміна - Б       | B7839E32
  2 | Карта Користувача - С  | C35D9D19
  3 | Карта Госта - Ф        | CF17B5DF
(3 rows)
```

Рисунок 3.12 – Створена база даних

Для автоматичного управління схемою бази даних під час розробки використовується параметр *spring.jpa.hibernate.ddl-auto=update* в docker-compose.yml. Це дозволяє Hibernate автоматично створювати або модифікувати таблицю rfid_cards (рис. 3.12) відповідно до визначеної JPA-сутності RFIDCard.

3.3.3 Реалізація бізнес-логіки контролю доступу

Ключовою функціональністю бекенду є ендпоінт */api/rfid/checkAccess*, який обробляє POST-запити від NodeMCU. Цей ендпоінт приймає JSON-об'єкт, що містить поле uid (унікальний ідентифікатор RFID-картки).

Алгоритм перевірки доступу наступний:

– контролер отримує UID з вхідного JSON-запиту;

– UID передається до RFIDCardService;

– RFIDCardService використовує RFIDCardRepository для пошуку картки з відповідним UID у базі даних PostgreSQL;

– якщо картка з таким UID знайдена, система визначає accessGranted як true та формує відповідне повідомлення («Access Granted»);

– якщо картка не знайдена, accessGranted встановлюється як false, і формується повідомлення («Access Denied»);

– результат (accessGranted та message) інкапсулюється у JSON-відповідь та надсилається назад до NodeMCU.

Цей чіткий алгоритм забезпечує надійну та швидку перевірку доступу, дозволяючи системі миттєво реагувати на спроби ідентифікації та надавати відповідні дозволи або відмови.

3.3.4 Тестування бекенд-частини

Тестування бекенд-частини проводилось у декілька етапів для забезпечення коректної функціональності та надійності.

Модульне тестування (Unit Testing): Окремі компоненти, такі як методи сервісного шару, можна перевіряти ізольовано за допомогою фреймворків,

наприклад, JUnit та Mockito, щоб оцінити коректність бізнес-логіки незалежно від інших частин системи.

Інтеграційне тестування (Integration Testing): Перевіряється взаємодія між різними шарами програми та базою даних.

Тестування API за допомогою curl: Використання утиліт curl у командному рядку дало змогу симулювати HTTP POST-запити від NodeMCU до ендпоінту /api/rfid/checkAccess.

Це підтвердило:

– Spring Boot додаток успішно запускається та доступний на заданому порті;

– REST API правильно обробляє JSON-запити;

– сервер коректно аналізує UID, взаємодіє з базою даних і повертає очікувані JSON-відповіді ({«accessGranted»:true,»message»:»Access Granted»} або {«accessGranted»:false,»message»:»Access Denied»}).

Перевірка підключення до бази даних: забезпечено, що Spring Boot коректно встановлює зв’язок із PostgreSQL, виконує запити та здійснює зчитування/збереження даних.

3.3.5 Контейнеризація за допомогою Docker

Для забезпечення відтворюваності середовища та спрощення розгортання, весь бекенд-проект був контейнеризований за допомогою Docker. Dockerfile (рис. 3.13) визначає процес побудови Docker-образу для Spring Boot додатка, включаючи встановлення необхідного JDK та збірку проекту за допомогою Maven Wrapper.

Весь код представлено у додаткі Б.

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY pom.xml .
COPY mvnw .
COPY .mvn .mvn/
COPY src ./src
RUN chmod +x mvnw
RUN ./mvnw clean package -Dmaven.test.skip=true
EXPOSE 8080
CMD ["java", "-jar", "target/demo-0.0.1-SNAPSHOT.jar"]
```

Рисунок 3.13 – Файл Dockerfile

Файл docker-compose.yml (рис. 3.14) використовується для одночасного запуску двох контейнерів: одного для PostgreSQL бази даних та одного для Spring Boot додатка, а також для налаштування мережевої взаємодії між ними. Це дозволяє розгорнути всю систему на будь-якій платформі, що підтримує Docker, без необхідності ручного встановлення Java, Maven та PostgreSQL.

```
version: '3.8' # Використовуємо Docker Compose версії 3.8
services:
  db:
    image: postgres:16.3-alpine # Використовуємо офіційний образ PostgreSQL версії 16.3 (легкий alpine)
    container_name: rfid_db # Зрозуміла назва контейнера
    environment: # Змінні середовища для конфігурації PostgreSQL
      POSTGRES_DB: rfid_db # Назва бази даних
      POSTGRES_USER: postgres # Ім'я користувача для бази даних
      POSTGRES_PASSWORD: mysecretpassword # Пароль для користувача бази даних
    ports: # Мапінг портів: 'хост:контейнер'
      - "5432:5432" # Порт 5432 хоста мапиться на порт 5432 контейнера PostgreSQL
    volumes: # Зберігання даних бази даних на хості для персистентності
      - db_data:/var/lib/postgresql/data # Дані зберігатимуться у названому томі 'db_data'
```

Рисунок 3.14 – Файл docker-compose.yml

Контейнеризація за допомогою Docker та Docker Compose забезпечує неперевершену відтворюваність та простоту розгортання бекенд-системи, дозволяючи швидко та ефективно запускати її в будь-якому середовищі

3.4 Інтеграція апаратного та програмного забезпечення

Ефективне функціонування системи контролю доступу забезпечується безшовною інтеграцією розробленого апаратного комплексу на базі мікроконтролера NodeMCU та бекенд-частини, реалізованої за допомогою

фреймворку Spring Boot. Цей підрозділ описує механізми взаємодії та роль використаних програмних бібліотек у забезпеченні двостороннього зв'язку.

Інтеграція здійснюється за принципом «клієнт-сервер», де NodeMCU виступає в ролі клієнта, що збирає дані з RFID-модуля та клавіатури, а Spring Boot бекенд є сервером, що обробляє ці дані, виконує бізнес-логіку (перевірку доступу) та надсилає відповідь. Комунікація між ними відбувається за допомогою протоколу HTTP через бездротову мережу Wi-Fi.

Мікроконтролер NodeMCU ESP8266 відіграє ключову роль у зборі даних та взаємодії з користувачем, а також у надсиланні запитів до бекенду та обробці його відповідей.

Для цього були використані спеціалізовані бібліотеки Arduino IDE:

– ESP8266WiFi.h: ця бібліотека є фундаментальною для забезпечення мережевого підключення NodeMCU. Вона дозволяє мікроконтролеру підключатися до заданої WiFi-мережі (SSID та пароль), отримувати IP-адресу та встановлювати стабільне бездротове з'єднання, що є обов'язковою передумовою для подальшої HTTP-комунікації;

– ESP8266HTTPClient.h: після встановлення WiFiз'єднання, ця бібліотека використовується для формування та надсилання HTTP POST-запитів до ендпоінту */api/rfid/checkAccess* на Spring Boot сервері. Вона відповідає за встановлення TCP-з'єднання з сервером, надсилання заголовків (*Content-Type: application/json*) та тіла запиту, а також отримання HTTP-відповіді;

– ArduinoJson.h: для обміну даними між NodeMCU та бекендом використовується формат JSON. Бібліотека ArduinoJson дозволяє ефективно серіалізувати (перетворювати дані NodeMCU в JSON-рядок) та десеріалізувати (парсити вхідний JSON-рядок у структуровані дані) JSON-об'єкти. У даній системі вона застосовується для створення JSON-запиту з UID картки та для розбору JSON-відповіді від сервера (*accessGranted, message*).

Після отримання UID картки або введення з клавіатури, NodeMCU надсилає цю інформацію до бекенду. Spring Boot додаток, який працює в Docker-контейнері, приймає ці запити через свій RESTful API (зокрема, ендпоінт

/api/rfid/checkAccess). Контролер Spring Boot (*RFIDController*) десеріалізує JSON-запит, передає UID до сервісного шару (*RFIDCardService*) для перевірки наявності в базі даних, а потім формує JSON-відповідь (*accessGranted, message*), яка серіалізується та повертається NodeMCU.

3.5 Тестування готової СКД

Тестування є невід'ємною частиною життєвого циклу розробки програмного та апаратного забезпечення, особливо для комплексних систем, що об'єднують різні технологічні домени, як у випадку із СКД. Основною метою тестування було верифікація функціональних вимог, підтвердження надійності інтеграції всіх компонентів, а також ідентифікація та усунення потенційних дефектів перед розгортанням. Процес тестування здійснювався ітеративно, охоплюючи як позитивні, так і негативні сценарії використання.

Тестування проводилося за методологією «чорної скриньки», де функціональність системи перевірялась без детального знання її внутрішньої структури, виходячи лише з зовнішніх вимог. Акцент був на наскрізному тестуванні (end-to-end), що включало перевірку взаємодії між NodeMCU, бекенд-сервером на Spring Boot та базою даних PostgreSQL.

3.5.1 Тестування позитивних сценаріїв доступу

Для верифікації позитивних сценаріїв доступу було використано чотири різних RFID-ідентифікатори: дві RFID-картки та два RFID-брелоки. Усі чотири ідентифікатори були попередньо внесені до бази даних PostgreSQL на бекенд-сервері та мали призначені паролі в прошивці NodeMCU.

Пари UID-Пароль для тестування:

- брелок 1 (UID: B7839E32) – Пароль: 1111;
- брелок 2 (UID: C35D9D19) – Пароль: 2222;
- картка 1 (UID: CF17B5DE) – Пароль: 3333;
- картка 2 – ні пароль, ні UID не були навмисно надані.

Процедура тестування:

– початковий стан: Система знаходиться в стані очікування картки, сервопривід у закритому положенні (90 градусів). На LCD виводиться «Attach card.»;

– **крок 1**: Зчитування RFID-ідентифікатора: Кожен з чотирьох ідентифікаторів (2 картки, 2 брелоки) підносився до RFID-зчитувача RC522;

– очікуваний результат (NodeMCU): NodeMCU зчитує UID, надсилає його до Spring Boot бекенду;

– очікуваний результат (Бекенд): Бекенд підтверджує наявність UID у базі даних, повертаючи відповідь {«accessGranted»:true, «message»: «Access Granted»};

– очікуваний результат (NodeMCU): NodeMCU отримує позитивну відповідь, на LCD виводиться «UID OK.» та «Pass:»;

– **крок 2**: Введення пароля: За допомогою клавіатури вводився відповідний пароль для зчитаного ідентифікатора. Символи на LCD відображались як *;

– **крок 3**: Підтвердження пароля: Після введення всіх символів, натискалась клавіша D (функція «ОК»);

– очікуваний результат (NodeMCU): NodeMCU локально перевіряє введений пароль з паролем, призначеним для зчитаного UID. При успішній перевірці на LCD виводиться «Access Granted!». Сервопривід переміщується до 180 градусів (імітуючи відкриття), утримується в цьому положенні 4 секунди, а потім повертається до 90 градусів (закриття);

– верифікація: У Serial Monitor NodeMCU фіксувались логі про успішне зчитування, обмін даними з бекендом, успішну перевірку пароля та рух сервоприводу.

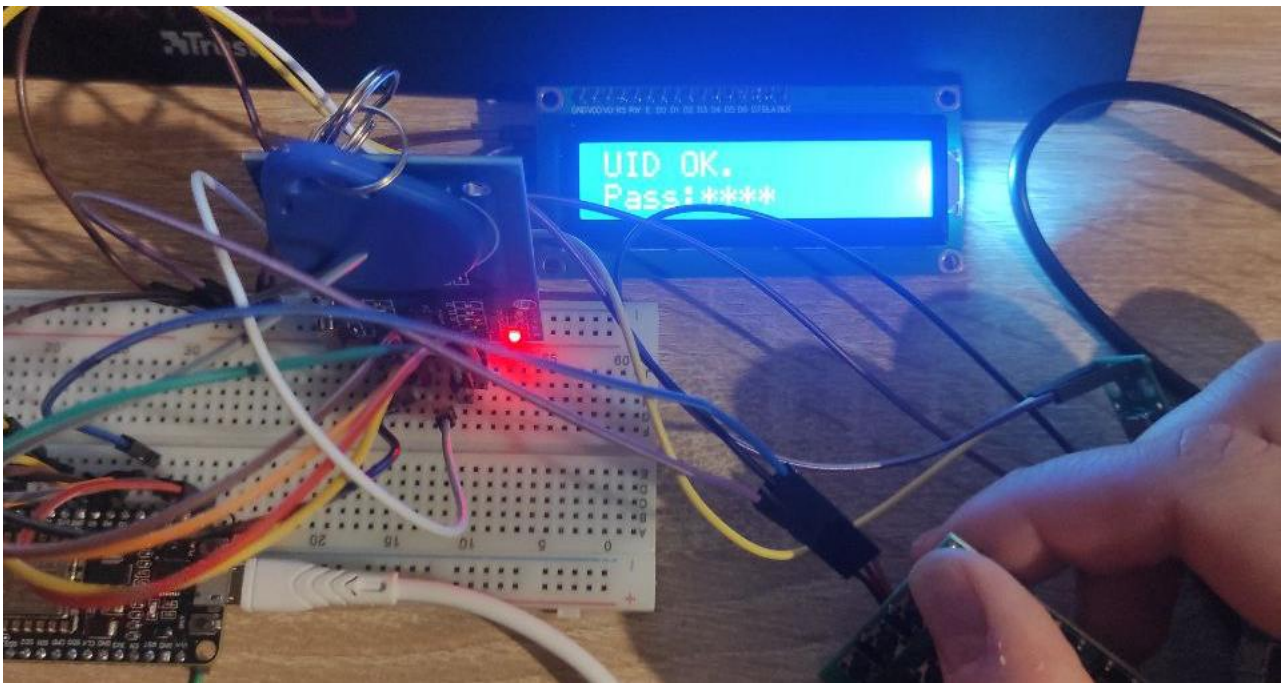

Рисунок 3.15 – Фото позитивного тестування

Результати позитивного тестування (рис. 3.15) для всіх трьох ідентифікаторів підтвердили коректну роботу RFID-зчитувача, клавіатури, LCD, сервоприводу, а також успішну двофакторну автентифікацію.

3.5.2 Тестування негативних сценаріїв доступу

Тестування негативних сценаріїв було спрямоване на верифікацію коректної відмови в доступі у випадку несанкціонованих спроб.

Сценарій 1: Картка не додана до бази даних (перший фактор) (рис. 3.16):

– процедура тестування: Була використана одна RFID-картка, UID якої навмисно не було додано до бази даних PostgreSQL. Ця картка підносилась до зчитувача;

– очікуваний результат (NodeMCU): NodeMCU зчитує UID, надсилає його до Spring Boot бекенду;

– очікуваний результат (Бекенд): Бекенд, не знайшовши UID у базі даних, повертає відповідь {«accessGranted»:false, «message»: «Access Denied»};

– очікуваний результат (NodeMCU): NodeMCU отримує негативну відповідь від бекенду. На LCD виводиться «Access Denied!», а сервопривід залишається нерухомим у закритому положенні. Запит пароля не відбувається;

– верифікація: У Serial Monitor NodeMCU фіксувались логі про відмову в доступі на рівні бекенду та відсутність подальших дій сервоприводу.

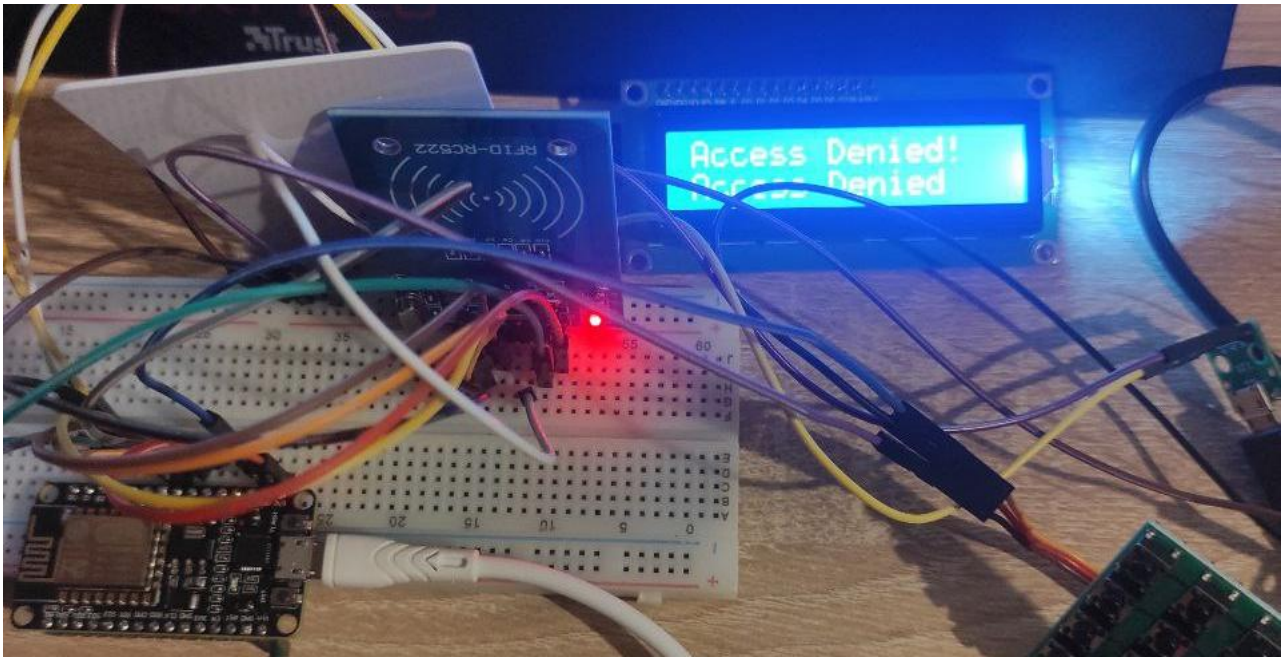

Рисунок 3.16 – Фото негативного тестування (сценарій 1)

Сценарій 2: Невірний пароль для авторизованої картки (другий фактор) (рис. 3.17):

– процедура тестування: Була використана одна з авторизованих RFID-карток (наприклад, Карта 1 з UID B7839E32). Після успішної перевірки UID бекендом, на запит пароля навмисно вводився невірний пароль (наприклад, 0000 замість 1111).

– очікуваний результат (NodeMCU): Після введення невірного пароля та натискання «ОК», NodeMCU локально виконує перевірку, виявляє невідповідність. На LCD виводиться «Access Denied!» та «Wrong Password.». Сервопривід залишається нерухомим.

– верифікація: У Serial Monitor NodeMCU фіксувались логі про невірний пароль та відмову в доступі.

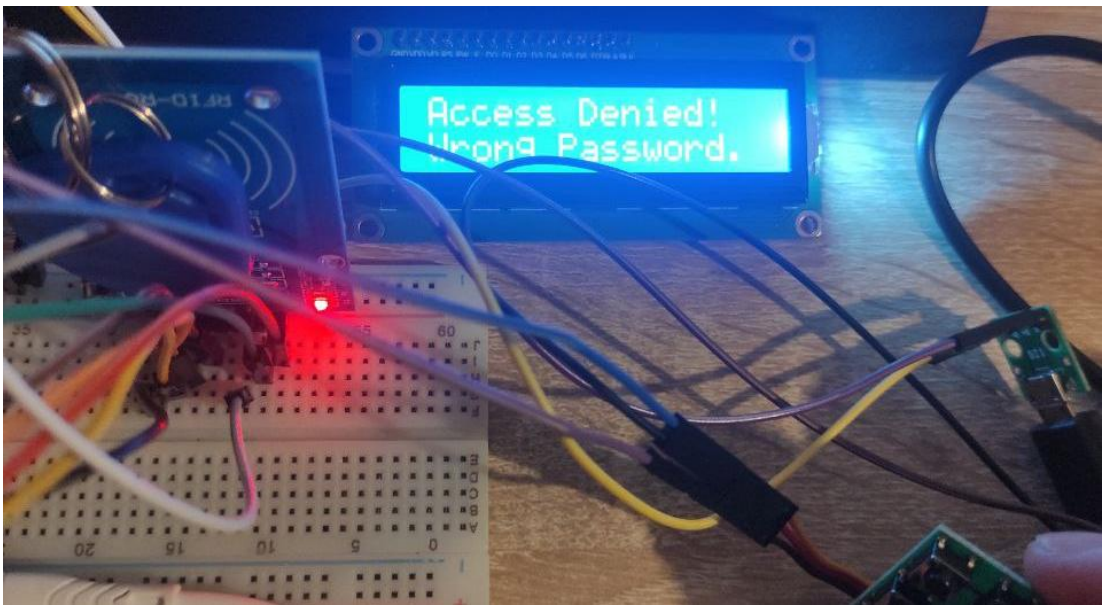

Рисунок 3.17 – Фото негативного тестування (сценарій 2)

Результати негативного тестування підтвердили, що СКД коректно відмовляє в доступі як на етапі перевірки ідентифікатора в базі даних, так і на етапі верифікації пароля, забезпечуючи багаторівневий захист.

3.6 Покращення створеної СКД шляхом двофакторної аутентифікації

Двофакторна аутентифікація (англ. Two-Factor Authentication, 2FA) є сучасним механізмом безпеки, що значно підвищує рівень захисту інформаційних систем шляхом вимагання від користувача надання двох незалежних факторів підтвердження особистості. Це контрастує з однофакторною аутентифікацією (англ. Single-Factor Authentication, SFA), яка покладається лише на один фактор (наприклад, пароль або RFID-картка). Метою 2FA є мінімізація ризиків несанкціонованого доступу у випадку компрометації одного з факторів аутентифікації.

Фактори аутентифікації класифікуються за трьома основними категоріями:

– фактор знання (Knowledge factor): інформація, відома лише користувачеві (наприклад, паролі, PIN-коди, відповіді на секретні запитання);

– фактор володіння (Possession factor): фізичний об'єкт, яким володіє користувач (наприклад, смартфон для отримання одноразових паролів

(англ. One-Time Password, OTP) через SMS або мобільні додатки, апаратні токени, смарт-картки);

– фактор приналежності (Inherence factor): унікальні біометричні характеристики користувача (наприклад, відбитки пальців, сканування райдужної оболонки ока, розпізнавання обличчя).

Інтеграція 2FA у СКД, що базується на RFID, перетворює RFID-картку на перший фактор (фактор володіння). Для реалізації другого фактору (як правило, фактор володіння у вигляді одноразового пароля) використовуються зовнішні комунікаційні сервіси.

3.6.1 Вибір та опис комунікаційної платформи Twilio

Для забезпечення другого фактору аутентифікації, а саме для генерації та доставки одноразових паролів OTP у вигляді SMS-повідомлень, було обрано хмарну комунікаційну платформу Twilio [30].

Twilio є провідним постачальником послуг Communication Platform as a Service, що надає розробникам програмні інтерфейси API для інтеграції функцій голосового зв'язку, обміну повідомленнями (SMS, MMS, WhatsApp), відео та інших комунікаційних можливостей безпосередньо у власні програмні продукти. Платформа відзначається високою надійністю, масштабованістю та широкою географією покриття, що забезпечує ефективну та швидку доставку повідомлень. Для даного проекту Twilio використовується як надійний шлюз для трансляції згенерованих OTP-кодів з бекенду СКД до мобільних пристроїв зареєстрованих користувачів.

3.6.2 Модифікації бекенд-системи для підтримки 2FA

Інтеграція двофакторної аутентифікації вимагала суттєвих змін у архітектурі та реалізації бекенд-компонентів, розроблених на базі Spring Boot. Весь код представлено у додатці В.

Управління залежностями та конфігурацією

Для забезпечення функціональності взаємодії з Twilio API, до файлу pom.xml було додано залежність *com.twilio.sdk:twilio*, що надає необхідні бібліотеки для програмного надсилання SMS-повідомлень.

Файл конфігурації(application.properties) було розширено для інтеграції облікових даних Twilio (twilio.account.sid, twilio.auth.token, twilio.phone.number). Ці параметри є ключовими для аутентифікації та визначення номера відправника SMS. Додатково, було налаштовано поведінку Hibernate (*spring.jpa.hibernate.ddl-auto=update*) для автоматичного оновлення схеми бази даних без втрати існуючих даних, а також активовано деталізоване логування SQL-запитів для моніторингу взаємодії з базою даних.

Моделювання даних та сервісна логіка

Модель сутності RFIDCard була доповнена новим атрибутом phoneNumber типу String. Це поле призначене для зберігання номера телефону користувача, на який надсилатимуться OTP-коди. Анотація *@Column(name = "phoneNumber")* була застосована для явного зіставлення цього поля з відповідним стовпцем у реляційній базі даних, запобігаючи розбіжностям через конвенції іменування Hibernate.

Функціонал класу RFIDCardService був значно розширений для реалізації логіки двофакторної аутентифікації:

– впроваджено залежність від TwilioSmsService, що дозволило використовувати його можливості надсилання SMS;

– додано тимчасове сховище otpStorage (реалізоване як HashMap) для зберігання згенерованих OTP-кодів у парі з відповідними UID карток, що використовується для їх верифікації;

– існуючий метод *checkAccess* було функціонально замінено на *checkAccessAndSendOtp*. Цей новий метод не лише перевіряє наявність RFID-картки за її UID, але й, у випадку успішного пошуку та наявності зареєстрованого номера телефону, генерує унікальний OTP, зберігає його та ініціює надсилання SMS з цим кодом через TwilioSmsService. Метод повертає булеве значення, що вказує на успішність надсилання OTP;

– уведено новий публічний метод verifyOtp, призначений для верифікації наданого користувачем OTP проти збереженого значення, що є другим фактором аутентифікації;

– додано приватний допоміжний метод generateOtp, відповідальний за створення 4-значних числових одноразових паролів

API-інтерфейс (REST Controller)

Контролер RESTful API був адаптований для підтримки нового потоку двофакторної аутентифікації.

Зокрема, додано новий ендпоінт *@PostMapping("/verifyOtp")*. Цей ендпоінт приймає OtpVerificationRequest, що містить як UID картки, так і введений користувачем OTP. Він викликає відповідний метод verifyOtp у RFIDCardService для перевірки OTP, надаючи остаточний статус доступу.

Ці системні модифікації забезпечують надійний та розширений механізм аутентифікації, підвищуючи безпеку СКД через інтеграцію двофакторної верифікації на основі OTP-SMS.

3.6.3 Загальний механізм функціонування оновленої системи

Вся інтегрована система працює за наступним алгоритмом: при піднесенні RFID-картки до зчитувача, NodeMCU ESP8266 перехоплює її унікальний ідентифікатор UID. Далі, через Wi-Fi з'єднання, мікроконтролер відправляє HTTP POST-запит, що містить цей UID, до бекенд-сервера на Spring Boot. Сервер, після отримання запиту, звертається до бази даних PostgreSQL для первинної верифікації наявності та валідності цього UID. У випадку успішного підтвердження UID, бекенд генерує одноразовий пароль OTP та ініціює його надсилання на зареєстрований номер телефону користувача через сервіс Twilio. NodeMCU отримує відповідь від сервера про те, що OTP надіслано, і відображає на РК-дисплеї запит на введення пароля. Користувач вводить отриманий OTP за допомогою матричної клавіатури, і ці дані локально перевіряються мікроконтролером. Лише після успішної двофакторної верифікації (підтвердження UID сервером та коректного введення OTP на клавіатурі)

NodeMCU активує сервопривід, імітуючи відкриття механізму доступу, після чого повертає його у вихідне положення, тим самим забезпечуючи підвищену безпеку СКД.

Висновки до розділу 3

У процесі виконання третього розділу дослідження було розроблено інтегрований АПК для СКД. Реалізовано інтерфейси взаємодії між апаратними компонентами, зокрема RFID-зчитувачем, клавіатурою, дисплеєм і сервоприводом, та серверною частиною, побудованою на основі фреймворку Spring Boot, що забезпечило гармонійну інтеграцію всіх елементів системи. Особливу увагу приділено тестуванню функціональних характеристик комплексу, включаючи аналіз коректності обробки даних UID-карток і реакції системи на введення користувацьких даних.

Запропонована архітектура підтвердила свою ефективність у функціонуючому прототипі. Система забезпечує надійну ідентифікацію користувачів, перевірку їхніх прав доступу через сервер і відповідне керування механізмом блокування. Такий підхід сприяв створенню стабільного, масштабованого рішення, придатного для адаптації до різних умов експлуатації, що обґрунтовує доцільність обраних технічних і програмних рішень.

ВИСНОВКИ

У результаті виконання КБР досягнуто поставлену мету створено СКД, яка забезпечує автоматизовану ідентифікацію користувачів і керування механізмом доступу шляхом розробки апаратно-програмного забезпечення (АПЗ) на основі RFID з вебзастосунком на базі Spring Boot.

У процесі виконання роботи було послідовно реалізовано всі поставлені завдання:

- проаналізувано сучасні технології та методи керування доступом;
- обгрунтувано вимоги до АПЗ;
- розроблено загальну архітектуру системи та підбір компонентів;
- розроблено апаратну частину системи;
- створено серверну частину системи на платформі Spring Boot для обробки даних і управління доступом;
- забезпечено інтеграцію апаратної та програмної;
- виконано тестування системи та оцінити її надійність.

Проведено комплексний аналіз сучасних технологій СКД, з фокусом на RFID. Ця технологія виявилася високоефективною для безконтактної ідентифікації та автоматизації контролю доступу, демонструючи гнучкість, масштабованість та підвищену безпеку порівняно з традиційними методами.

Обґрунтовано вимоги до апаратного та програмного забезпечення, включаючи сумісність, надійність та простоту впровадження. На основі цих вимог обрано оптимальні компоненти для економічного та надійного комплексу: NodeMCU ESP8266, RFID-модуль MFRC522, сервопривід SG90, дисплей LCD1602 та клавіатура 4×4.

На етапі архітектурного проєктування розроблено структуру інтегрованої АПЗ з чітким розмежуванням функцій. Мікроконтролер координує роботу датчиків та сервера через Wi-Fi. Серверна частина реалізована на Spring Boot з PostgreSQL та Docker, що забезпечує масштабованість та зручність розгортання.

Апаратна частина системи спроєктована з урахуванням вимог до електроживлення, інтерфейсів зв’язку (SPI, I2C) та енергоефективності. Можливість конфігурації параметрів через клавіатуру підвищує ергономічність.

Програмне забезпечення розроблено на основі Spring Boot з багатошаровою архітектурою, використанням RESTful API, Hibernate та механізмів безпеки. Реалізовано централізоване зберігання даних карток, журналювання подій, адміністративний веб-інтерфейс та масштабування функціоналу.

Інтеграцію апаратної та програмної частин забезпечено через обмін HTTP POST-запитами між мікроконтролером і сервером. Успішно реалізовано обробку UID у реальному часі, керування сервоприводом та виведення інформації на дисплей, підтверджуючи узгоджену взаємодію.

На завершальному етапі проведено тестування прототипу, яке підтвердило працездатність системи, коректність авторизації, стабільність зв’язку та швидкість обробки запитів. Усі недоліки усунуто, а система отримала високу оцінку за надійність, адаптивність та економічну ефективність.

Таким чином, усі поставлені завдання виконано: проведено аналіз технологій, сформовано вимоги до АПЗ, спроєктовано архітектуру, реалізовано апаратну та програмну складові, забезпечено їх інтеграцію, протестовано функціональність та підтверджено ефективність рішення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Біометричний настільний зчитувач відбитків пальців Cyphrax FRD. URL:https://ip24.com.ua/ua/p1609616091-biometricheskij-nastolnyj-schityvatel.html (дата звернення: 10.05.2025).

2. HID Mobile Solutions. URL: https://campaigns.hidglobal.com/open-more?utm_source=engage.hidglobal.com&utm_campaign=engage.hidglobal.com&utm_medium=301 (Last accessed: 10.05.2025).

3. Yale Assure Lock Deadbolt Digital Touchscreen Keypad Keyless Satin Nickel. URL: https://surl.li/wwurro (Last accessed: 10.05.2025).

4. Як Spring Boot допомагає прискорити розробку застосунків. URL: https://foxminded.ua/spring-boot/ (дата звернення: 10.05.2025).

5. XS4. URL: https://salto.com.ua/category/xs4-2/ (дата звернення: 10.05.2025).

6. RFID технологія для автоматизації обліку. URL: https://remonline.ua/blog/rfid-technology/ (дата звернення: 10.05.2025).

7. Що таке система RFID, в чому її особливості використання. URL: https://surl.li/ilsowq (дата звернення: 10.05.2025).

8. Complete Guide to RFID Access Control Systems. URL: https://www.rfidcard.com/complete-guide-to-rfid-access-control-systems/ (Last accessed: 10.05.2025).

9. RFID Regulations. URL: https://rfid4u.com/rfid-regulations/ (Last accessed: 10.05.2025).

10. What Is RFID Access Control And How Does It Work? URL: https://blog.nortechcontrol.com/rfid-access-control (Last accessed: 10.05.2025).

11. Zhang Z., Yan C. Research on Data Sharing Access Control Based on Blockchain Technology. *Autom Control Intell Syst*. 2022. Vol. 10, Is. 1. P 8–13. DOI: 10.11648/j.acis.20221001.12.

12. Shabbir Hassan, Ahmad Raza Shibli, Rehan Raza. Ubiquitous Computing with Radio Frequency IDentification Tags. *Math Comput Sci*. 2023. Vol. 8, Is. 3. P 73–86. DOI: 10.11648/j.mcs.20230803.12.

13. Aliyu M. B., Suru H., Gabi D., Garba M., Argungu M. The Need for Adaptive Access Control System at the Network Edge. *American Journal of Information Science and Technology*, 2024. Vol. 8, Is. 2. P. 45–55. DOI: 10.11648/j.ajist.20240802.13.

14. LCD 1602 I2C символьний дісплей 16×2 (синій)URL: https://arduino.ua/prod6188-lcd-1602-i2c-simvolnii-displei-16x2-sinii (дата звернення: 10.05.2025).

15. Модуль матрична клавіатура 16 кнопок матриця 4×4. URL: https://diyshop.com.ua/ua/modul-matrichnaya-klaviatura-16-knopok-matrica-4h4?srsltid=AfmBOoqAWjThcbKCqJc_F8LsrG6n98oW_0xukI3TaDJPEXPzP7XE2j-D (дата звернення: 10.05.2025).

16. WiFi Плата NodeMCU V2 ESP8266. *Arduino*. URL: https://arduino.ua/prod1495-wifi-plata-nodemcu-v2-esp8266-cp2102 (дата звернення: 10.05.2025).

17. RFID модуль RC522 з карткою доступу для Arduino MQ-2. *Arduino*. URL: https://arduino.ua/prod649-rfid-modyl-rc522-s-kartochkoi-dostypa-dlya-arduino (дата звернення: 10.05.2025).

18. Серводвигун SG90 2кг 180°. URL: https://arduino.ua/prod416-servoprivod-sg90-2kg (дата звернення: 10.05.2025).

19. Мережевий зарядний пристрій 5V USB 1A. URL: https://prom.ua/ua/p1427599173-setevoe-zaryadnoe-ustrojstvo.html (дата звернення: 10.05.2025).

20. Комплект перемичок мама-мама, тато-тато, мама-тато 120шт. 20см. URL: https://arduino.ua/prod1596-komplekt-peremichek-arduino-120sht (дата звернення: 10.05.2025).

21. TZT AGS10 TVOC Air Quality Gas Sensor I2C MEMS Replaces AGS02MA For Arduino. URL: https://surl.li/pstoxl (Last accessed: 21.05.2025).

22. Підключення RFID модуля RC522 до плати Arduino та LCD дисплея. URL: https://robostore.com.ua/ua/rfid-modul-rc522-with-arduino/ (дата звернення: 21.05.2025).

23. Безпайкова макетна плата MB-102 на 830 пікселів. URL: https://www.mini-tech.com.ua/ua/bespayechnaya-maketnaya-plata-830-tochek (дата звернення: 21.05.2025).

24. Кабель Ugreen US289 USB 2.0 to Micro Cable Nickel Plating 2A 0.25м. URL: https://surl.li/oeavxo (дата звернення: 21.05.2025).

25. Docker: як створювати образи контейнерів та розгортати програми. URL: https://habr.com/ru/articles/776188/ (дата звернення: 21.05.2025).

26. YouTube. URL: https://www.youtube.com/watch?v=fL5NDw0rDOI&t=4108s&ab_channel=AntoshaKorsakov (дата звернення: 21.05.2025).

27. Модуль розширення портів PCF8575. URL: https://surli.cc/bkmrbk (дата звернення: 21.05.2025).

28. Плата-перехідник гніздо microUSB на DIP. URL: https://surli.cc/kvhyam (дата звернення: 21.05.2025).

29. Шилік А., Пузирьов С. Система керування доступом на базі RFID та Spring Boot. *Ольвійський форум – 2025: стратегії країн Причорноморського регіону в геополітичному просторі* : тези доп. XXII Міжнар. наук. конф., Миколаїв, 16–21 черв. 2025 р. Миколаїв : ЧНУ ім. Петра Могили, 2025 *(подано до друку)*.

30. The Twilio Customer Engagement platform combines powerful communications APIs with AI and first-party data.. URL: https://www.twilio.com/en-us (Last accessed: 21.05.2025).

ДОДАТОК А

Код для апаратного комплексу

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <PCF8575.h>
#include <Keypad.h>
#undef CLOSED
#include <LiquidCrystal_I2C.h>
#include <Servo.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>

const char* ssid = "Redmi Note 9 Pro";
const char* password = "11112222";
const char* springBootServerHost = "192.168.43.126";
const int springBootServerPort = 8080;
const char* checkAccessEndpoint = "/api/rfid/checkAccess";

#define RST_PIN 0
#define SS_PIN 2
MFRC522 mfrc522(SS_PIN, RST_PIN);

PCF8575 pcf8575(0x20);

const byte ROWS = 4;
const byte COLS = 4;

byte rowPins[ROWS] = {0, 1, 2, 3};
byte colPins[COLS] = {4, 5, 6, 7};

char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

class CustomKeypad : public Keypad {
public:
  CustomKeypad(char *userKeymap, byte *row, byte *col, byte numRows, byte numCols)
    : Keypad(userKeymap, row, col, numRows, numCols) {}

  void pin_mode(byte pinNum, byte mode) override {
    if (mode == INPUT) {
      pcf8575.write(pinNum, HIGH);
    }
```

```
  }

  void pin_write(byte pinNum, boolean level) override {
    pcf8575.write(pinNum, level);
  }

  int pin_read(byte pinNum) override {
    return pcf8575.read(pinNum);
  }
};

CustomKeypad customKeypad((char *)keys, rowPins, colPins, ROWS, COLS);

#define LCD_ADDR 0x27
#define LCD_COLUMNS 16
#define LCD_ROWS 2
LiquidCrystal_I2C lcd(LCD_ADDR, LCD_COLUMNS, LCD_ROWS);

#define SERVO_PIN 15
Servo myServo;

char keypad_input_buffer[LCD_COLUMNS + 1];
byte cursor_position = 0;
const byte MAX_PASSWORD_LENGTH = 4;

WiFiClient client;

enum AuthState {
  WAIT_FOR_CARD,
  CARD_SCANNED_WAIT_FOR_PASSWORD,
  PASSWORD_VALIDATION_IN_PROGRESS,
  ACCESS_GRANTED,
  ACCESS_DENIED
};

AuthState currentAuthState = WAIT_FOR_CARD;
String scannedUidGlobal = "";
String enteredPassword = "";

struct UserCredentials {
  String uid;
  String password;
};

UserCredentials users[] = {
  {"B7839E32", "1111"},
  {"C35D9D19", "2222"},
  {"CF17B5DE", "3333"}
};
const int NUM_USERS = sizeof(users) / sizeof(users[0]);
```

```
bool validateLocalPassword(String uid, String password) {
  for (int i = 0; i < NUM_USERS; i++) {
    if (users[i].uid.equalsIgnoreCase(uid)) {
      if (users[i].password.equals(password)) {
        return true;
      } else {
        return false;
      }
    }
  }
  return false;
}

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

  delay(10);
  WiFi.begin(ssid, password);

  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(F("Connecting WiFi."));

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    lcd.print(".");
  }

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(F("WiFi Connected!"));
  lcd.setCursor(0,1);
  lcd.print(WiFi.localIP());
  delay(2000);

  Wire.begin(5, 4);

  pcf8575.begin();
  Wire.beginTransmission(0x20);
  byte errorPCF = Wire.endTransmission();
  if (errorPCF != 0) {
    lcd.clear(); lcd.print(F("PCF8575 Error!")); while (true);
  }

  lcd.init();
  lcd.backlight();
  Wire.beginTransmission(LCD_ADDR);
  byte errorLCD = Wire.endTransmission();
  if (errorLCD != 0) {
```

```
    lcd.clear(); lcd.print(F("LCD Error!")); while (true);
  }

  customKeypad.begin((char*)keys);
  customKeypad.setDebounceTime(50);

  SPI.begin();
  mfrc522.PCD_Init();

  myServo.attach(SERVO_PIN);
  myServo.write(90);

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(F("Attach card."));
  lcd.setCursor(0, 1);
  lcd.print(F("Ready."));
}

void loop() {
  char key = '\0';
  switch (currentAuthState) {
    case WAIT_FOR_CARD:
      if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
        scannedUidGlobal = "";
        for (byte i = 0; i < mfrc522.uid.size; i++) {
          if (mfrc522.uid.uidByte[i] < 0x10) scannedUidGlobal += "0";
          scannedUidGlobal += String(mfrc522.uid.uidByte[i], HEX);
        }
        scannedUidGlobal.toUpperCase();
        if (WiFi.status() == WL_CONNECTED) {
          HTTPClient http;

          String serverUrl = "http://" + String(springBootServerHost) + ":" +
String(springBootServerPort) + String(checkAccessEndpoint);

          http.begin(client, serverUrl);
          http.addHeader("Content-Type", "application/json");
          StaticJsonDocument<128> doc;
          doc["uid"] = scannedUidGlobal;
          String requestBody;
          serializeJson(doc, requestBody);
          int httpResponseCode = http.POST(requestBody);
          if (httpResponseCode > 0) {
            String responsePayload = http.getString();
            StaticJsonDocument<128> responseDoc;
            DeserializationError error = deserializeJson(responseDoc, responsePayload);
            if (!error) {
              bool accessGrantedByBackend = responseDoc["accessGranted"];
              String message = responseDoc["message"].as<String>();
              if (accessGrantedByBackend) {
```

```
              lcd.clear();
              lcd.setCursor(0, 0);
              lcd.print(F("UID OK."));
              lcd.setCursor(0, 1);
              lcd.print(F("Pass:"));
              memset(keypad_input_buffer, 0, sizeof(keypad_input_buffer));
              cursor_position = 0;
              enteredPassword = "";
              currentAuthState = CARD_SCANNED_WAIT_FOR_PASSWORD;
            } else {
              lcd.clear();
              lcd.setCursor(0, 0);
              lcd.print(F("Access Denied!"));
              lcd.setCursor(0, 1);
              lcd.print(message);
              currentAuthState = ACCESS_DENIED;
            }
          } else {
            lcd.clear(); lcd.setCursor(0, 0); lcd.print(F("JSON Error!"));
            currentAuthState = ACCESS_DENIED;
          }
        } else {
          lcd.clear(); lcd.setCursor(0, 0); lcd.print(F("HTTP Error!"));
          currentAuthState = ACCESS_DENIED;
        }
        http.end();
      } else {
        lcd.clear(); lcd.setCursor(0,0); lcd.print(F("WiFi Disconnected"));
lcd.setCursor(0,1); lcd.print(F("No access check"));
        currentAuthState = ACCESS_DENIED;
      }
      mfrc522.PICC_HaltA();
      mfrc522.PCD_StopCrypto1();
    }
    break;
  case CARD_SCANNED_WAIT_FOR_PASSWORD:
    key = customKeypad.getKey();
    if (key) {
      if (key == '*') {
        if (cursor_position > 0) {
          cursor_position--;
          keypad_input_buffer[cursor_position] = '\0';
          lcd.setCursor(5 + cursor_position, 1);
          lcd.print(' ');
        }
      } else if (key == '#') {
        memset(keypad_input_buffer, 0, sizeof(keypad_input_buffer));
        cursor_position = 0;
        lcd.setCursor(0, 1);
        lcd.print(F("Pass:          "));
      } else if (key == 'D') {
```

```
          enteredPassword = String(keypad_input_buffer);
          enteredPassword.trim();
          currentAuthState = PASSWORD_VALIDATION_IN_PROGRESS;
        } else if (cursor_position < MAX_PASSWORD_LENGTH) {
          keypad_input_buffer[cursor_position] = key;
          lcd.setCursor(5 + cursor_position, 1);
          lcd.print('*');
          cursor_position++;
          keypad_input_buffer[cursor_position] = '\0';
        }
      }
      break;
    case PASSWORD_VALIDATION_IN_PROGRESS:
      if (validateLocalPassword(scannedUidGlobal, enteredPassword)) {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(F("Access Granted!"));
        currentAuthState = ACCESS_GRANTED;
      } else {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(F("Access Denied!"));
        lcd.setCursor(0, 1);
        lcd.print(F("Wrong Password."));
        currentAuthState = ACCESS_DENIED;
      }
      break;

    case ACCESS_GRANTED:
      myServo.write(180);
      delay(4000);
      myServo.write(90);
      currentAuthState = WAIT_FOR_CARD;
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print(F("Attach card."));
      lcd.setCursor(0, 1);
      lcd.print(F("Ready."));
      break;
    case ACCESS_DENIED:
      delay(4000);
      currentAuthState = WAIT_FOR_CARD;
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print(F("Attach card."));
      lcd.setCursor(0, 1);
      lcd.print(F("Ready."));
      break;
  }
}
```

ДОДАТОК Б

Код серверної частини

RFIDController

```
package com.example.demo.controller;
import com.example.demo.service.RFIDCardService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import lombok.Data;
@RestController
@RequestMapping("/api/rfid")
public class RFIDController {
    private final RFIDCardService rfidCardService;
    @Autowired
    public RFIDController(RFIDCardService rfidCardService) {
        this.rfidCardService = rfidCardService;
    }
    @PostMapping("/checkAccess")
    public ResponseEntity<AccessResponse> checkAccess(@RequestBody AccessRequest
requestBody) {
        String uid = requestBody.getUid();
        boolean accessGranted = rfidCardService.checkAccess(uid);

        if (accessGranted) {
            System.out.println("Access granted for UID: " + uid);
            return ResponseEntity.ok(new AccessResponse(true, "Access Granted"));
        } else {
            System.out.println("Access denied for UID: " + uid);
            return ResponseEntity.ok(new AccessResponse(false, "Access Denied"));
        }
    }
    @Data
    public static class AccessRequest {
        private String uid;

        public AccessRequest() {}

        public AccessRequest(String uid) {
            this.uid = uid;
        }
    }
    @Data
    public static class AccessResponse {
        private boolean accessGranted;
        private String message;
        public AccessResponse(boolean accessGranted, String message) {
            this.accessGranted = accessGranted;
            this.message = message;
        }
    }
}
```

RFIDCard

```
package com.example.demo.model;
import jakarta.persistence.*;
import lombok.Data;
@Entity
```

```
@Table(name = "rfid_cards")
@Data
public class RFIDCard {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "uid", nullable = false, unique = true)
    private String uid;
    @Column(name = "description")
    private String description;
    public RFIDCard() {
    }
    public RFIDCard(String uid, String description) {
        this.uid = uid;
        this.description = description;
    }
}
```

RFIDCardRepository

```
package com.example.demo.repository;
import com.example.demo.model.RFIDCard;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;
@Repository
public interface RFIDCardRepository extends JpaRepository<RFIDCard, Long> {
    Optional<RFIDCard> findByUid(String uid);
}
```

RFIDCardService

```
package com.example.demo.service;
import com.example.demo.model.RFIDCard;
import com.example.demo.repository.RFIDCardRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.Optional;
@Service
public class RFIDCardService {
    private final RFIDCardRepository rfidCardRepository;
    @Autowired
    public RFIDCardService(RFIDCardRepository rfidCardRepository) {
        this.rfidCardRepository = rfidCardRepository;
    }
    public boolean checkAccess(String uid) {
        Optional<RFIDCard> card = rfidCardRepository.findByUid(uid);
        return card.isPresent();
    }
    public Optional<RFIDCard> getCardByUid(String uid) {
        return rfidCardRepository.findByUid(uid);
    }
}
```

Dockerfile

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY pom.xml .
COPY mvnw .
COPY .mvn .mvn/
COPY src ./src
RUN chmod +x mvnw
RUN ./mvnw clean package -Dmaven.test.skip=true
EXPOSE 8080
CMD ["java", "-jar", "target/demo-0.0.1-SNAPSHOT.jar"]
```

docker-compose.yml

```
version: '3.8' # Використовуємо Docker Compose версії 3.8
services:
  db:
    image: postgres:16.3-alpine # Використовуємо офіційний образ PostgreSQL версії 16.3
(легкий alpine)
    container_name: rfid_db # Зрозуміла назва контейнера
    environment: # Змінні середовища для конфігурації PostgreSQL
      POSTGRES_DB: rfid_db # Назва бази даних
      POSTGRES_USER: postgres # Ім'я користувача для бази даних
      POSTGRES_PASSWORD: mysecretpassword # Пароль для користувача бази даних
    ports: # Мапінг портів: 'хост:контейнер'
      - "5432:5432" # Порт 5432 хоста мапиться на порт 5432 контейнера PostgreSQL
    volumes: # Зберігання даних бази даних на хості для персистентності
      - db_data:/var/lib/postgresql/data # Дані зберігатимуться у названому томі
'db_data'
    healthcheck: # Перевірка стану бази даних
      test: ["CMD-SHELL", "pg_isready -U postgres"] # Команда для перевірки готовності
      interval: 5s # Інтервал перевірки
      timeout: 5s # Тайм-аут для перевірки
      retries: 5 # Кількість спроб
    restart: always # Завжди перезапускати контейнер, якщо він вийде з ладу
  app:
    build: . # Docker буде шукати Dockerfile у поточному каталозі
    container_name: rfid_spring_app # Зрозуміла назва контейнера
    ports: # Мапінг портів: 'хост:контейнер'
      - "8080:8080" # Порт 8080 хоста мапиться на порт 8080 контейнера Spring Boot
    environment: # Змінні середовища для конфігурації Spring Boot додатка
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/rfid_db
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: mysecretpassword
      SPRING_JPA_HIBERNATE_DDL_AUTO: update # 'update' буде створювати/оновлювати
таблиці
    depends_on: # Залежності між сервісами
      db: # 'app' залежить від 'db'
        condition: service_healthy # 'app' запуститься лише коли 'db' буде здоровим
    restart: always # Завжди перезапускати контейнер, якщо він вийде з ладу
volumes:
  db_data: # Том для зберігання даних PostgreSQL
```

ДОДАТОК В

Код покращеної серверної частини

RfidCardController

```
package com.example.demo.controller;
import com.example.demo.entity.RfidCard;
import com.example.demo.model.AccessResponse;
import com.example.demo.model.OtpVerificationRequest;
import com.example.demo.model.UidRequest;
import com.example.demo.service.RfidCardService;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.Optional;
@RestController
@RequestMapping("/api/rfid")
@RequiredArgsConstructor
public class RfidCardController {
    private static final Logger logger = LoggerFactory.getLogger(RfidCardController.class);
    private final RfidCardService rfidCardService;
    @PostMapping("/checkAccess")
    public ResponseEntity<AccessResponse> checkAccess(@RequestBody UidRequest request) {
        AccessResponse response = new AccessResponse();
        logger.info("Received checkAccess request for UID: {}", request.getUid());
        Optional<RfidCard> cardOptional = rfidCardService.getByUuid(request.getUid());
        if (cardOptional.isPresent()) {
            boolean otpSent = rfidCardService.checkAccessAndSendOtp(request.getUid());
            if (otpSent) {
                response.setOtpSent(true);
                response.setMessage("OTP sent. Please enter it on the keypad.");
                logger.info("OTP successfully initiated for UID: {}", request.getUid());
```

```
                } else {
                    response.setOtpSent(false);
                    response.setMessage("Failed to send OTP. Phone number missing or
invalid.");
                    logger.warn("Failed to send OTP for UID: {}", request.getUid());
                }
                response.setAccessGranted(false);
            } else {
                response.setAccessGranted(false);
                response.setOtpSent(false);
                response.setMessage("Access Denied. Card not found.");
                logger.warn("Access Denied: UID {} not found.", request.getUid());
            }
            return ResponseEntity.ok(response);
        }
        @PostMapping("/verifyOtp")
        public         ResponseEntity<AccessResponse>         verifyOtp(@RequestBody
OtpVerificationRequest request) {
            AccessResponse response = new AccessResponse();
            logger.info("Received   verifyOtp   request   for   UID:   {},   OTP:   {}",
request.getUid(), request.getOtp());
            boolean    isOtpValid    =    rfidCardService.verifyOtp(request.getUid(),
request.getOtp());
            if (isOtpValid) {
                response.setAccessGranted(true);
                response.setMessage("Access Granted. Welcome!");
                response.setOtpSent(false);
                logger.info("Final access granted for UID: {}", request.getUid());
            } else {
                response.setAccessGranted(false);
                response.setMessage("Access Denied. Incorrect OTP.");
                response.setOtpSent(false);
                logger.warn("Final  access  denied  for  UID:  {}.  Incorrect  OTP.",
request.getUid());
            }
            return ResponseEntity.ok(response);
        }
```

RfidCard

```
package com.example.demo.entity;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
```

```
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;
@Entity
@Table(name = "rfid_cards")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class RfidCard {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(unique = true, nullable = false)
    private String uid;
    private String description;
    @Column(name = "phoneNumber")
    private String phoneNumber;
}
```

AccessResponse

```
package com.example.demo.model;
import lombok.Data;
@Data // Lombok: геттери, сеттери, toString, equals, hashCode
public  class AccessResponse { // Цей клас є вкладеним та статичним, що дозволено
    private boolean accessGranted;
    private String message;
    private boolean otpSent; // Додано: чи було надіслано OTP
}
```

OtpVerificationRequest

```
package com.example.demo.entity;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
import lombok.AllArgsConstructor;
@Entity
@Table(name = "rfid_cards")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class RfidCard {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(unique = true, nullable = false)
    private String uid;
    private String description;
    @Column(name = "phoneNumber")
    private String phoneNumber;
}
```

UidRequest

```
package com.example.demo.model;
import lombok.Data;
@Data // Lombok: геттери, сеттери, toString, equals, hashCode
public  class UidRequest { // Цей клас є вкладеним та статичним, що дозволено
    private String uid;
}
```

RfidCardRepository

```
package com.example.demo.repository;
import com.example.demo.entity.RfidCard;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.Optional;
@Repository
public interface RfidCardRepository extends JpaRepository<RfidCard, Long> {
    Optional<RfidCard> findByUid(String uid);
}
```

RfidCardService

```
package com.example.demo.service;
import com.example.demo.entity.RfidCard;
import java.util.Optional;
public interface RfidCardService {
    Optional<RfidCard> getByUuid(String uuid);
    boolean checkAccessAndSendOtp(String uid);
    boolean verifyOtp(String uid, String enteredOtp);}
```

RfidCardServiceImpl

```
package com.example.demo.service;
import com.example.demo.entity.RfidCard;
import com.example.demo.repository.RfidCardRepository;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import java.util.HashMap;
import java.util.Map;
import java.util.Optional;
import java.util.Random;
@Service
@RequiredArgsConstructor
public class RfidCardServiceImpl implements RfidCardService {
    private static final Logger logger =
LoggerFactory.getLogger(RfidCardServiceImpl.class);
    private final RfidCardRepository rfidCardRepository;
    private final TwilioSmsService twilioSmsService;
    private final Map<String, String> otpStorage = new HashMap<>();
    private final Random random = new Random();
    public Optional<RfidCard> getByUuid(String uuid) {
        return rfidCardRepository.findByUid(uuid);
    }
    public boolean checkAccessAndSendOtp(String uid) {
        Optional<RfidCard> cardOptional = getByUuid
```

TwilioSmsService

```
package com.example.demo.service;
public interface TwilioSmsService {
    void sendSms(String toPhoneNumber, String messageBody);
}
```

TwilioSmsServiceImpl

```
package com.example.demo.service;
import com.twilio.Twilio;
import com.twilio.rest.api.v2010.account.Message;
import com.twilio.type.PhoneNumber;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
@Service // Позначає цей клас як сервісний компонент Spring
```

```
public class TwilioSmsServiceImpl implements TwilioSmsService {
    private static final Logger logger = LoggerFactory.getLogger(TwilioSmsServiceImpl.class);
    private final String twilioPhoneNumber; // Ваш номер Twilio
    // Конструктор для ініціалізації Twilio
    public TwilioSmsServiceImpl(@Value("${twilio.account.sid}") String accountSid,
                                @Value("${twilio.auth.token}") String authToken,
                                @Value("${twilio.phone.number}") String twilioPhoneNumber) {
        this.twilioPhoneNumber = twilioPhoneNumber;
        Twilio.init(accountSid, authToken);
        logger.info("Twilio SMS Service initialized with Account SID ending in {}.", accountSid.substring(accountSid.length() - 4));
    }
    public void sendSms(String toPhoneNumber, String messageBody) {
        try {
            Message message = Message.creator(
                    new PhoneNumber(toPhoneNumber), // Номер одержувача
                    new PhoneNumber(twilioPhoneNumber), // Ваш номер Twilio
                    messageBody) // Текст повідомлення
                .create();
            logger.info("SMS sent to {} with SID: {}", toPhoneNumber, message.getSid());
        } catch (Exception e) {
            logger.error("Failed to send SMS to {}: {}", toPhoneNumber, e.getMessage(), e);
            // Обробка винятків: логування, повідомлення користувача тощо
        }
    }
}
```

application.properties

```
# Spring Boot Application Name
spring.application.name=demo
# PostgreSQL Database Configuration
 spring.datasource.url=${DB_URL:jdbc:postgresql://localhost:5432/rfid_db}
 spring.datasource.username=${DB_USER:postgres}
 spring.datasource.password=${DB_PASSWORD:mysecretpassword}
 spring.datasource.driver-class-name=org.postgresql.Driver
# JPA and Hibernate Configuration
```

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.SQL=DEBUG
# spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
# Server Port
server.port=8080
# Twilio Configuration
twilio.account.sid=ACcff6c461f569014d98667c8dc84148f7
twilio.auth.token=0c079054bb74b5198f9d5a1382b0a297
twilio.phone.number=+19283713160
```

docker-compose.yml

```
version: '3.8' # Можете видалити цей рядок, це лише попередження
services:
  app:
    build: .
    ports:
      - "9081:8080"
    environment:
      DB_URL: jdbc:postgresql://rfid-db:5432/rfid
      DB_USER: postgres
      DB_PASSWORD: mysecretpassword
#      SPRING_DATASOURCE_URL: jdbc:postgresql://rfid_db:5432/rfid_db # НОВИЙ ХОСТ
БД
#      SPRING_DATASOURCE_USERNAME: postgres
#      SPRING_DATASOURCE_PASSWORD: mysecretpassword
    depends_on:
      - rfid-db # Залежність від нового імені сервісу БД
  rfid-db: # НОВЕ ІМ'Я СЕРВІСУ БД
    image: postgres:16.3-alpine
    container_name: rfid_db_container # НОВА НАЗВА КОНТЕЙНЕРА БД
    ports:
      - "5432:5432"
    environment:
      POSTGRES_DB: rfid # НОВА НАЗВА БАЗИ ДАНИХ
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: mysecretpassword
    volumes:
      - rfid_db_data:/var/lib/postgresql/data # НОВА НАЗВА ТОМУ

volumes:
```

```
  rfid_db_data: # НОВА НАЗВА ТОМУ
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project                                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>3.5.0</version> <!-- Переконайтеся, що це ваша поточна версія
Spring Boot -->
      <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.skd.rfid</groupId>
<artifactId>rfid-server</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>rfid-server</name>
<description>RFID server based on Spring Boot</description>
<properties>
      <java.version>17</java.version>
</properties>
<dependencies>
      <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>
      <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
      <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
      </dependency>
      <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
```

```
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>

		<!-- Додано: Залежність для Twilio -->
		<dependency>
			<groupId>com.twilio.sdk</groupId>
			<artifactId>twilio</artifactId>
			<version>10.9.2</version> <!-- Використовуйте актуальну версію Twilio
SDK -->
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
				<configuration>
					<excludes>
						<exclude>
							<groupId>org.projectlombok</groupId>
							<artifactId>lombok</artifactId>
						</exclude>
					</excludes>
				</configuration>
			</plugin>
		</plugins>
	</build>

	</project>
```

ДОДАТОК Г

Матеріали апробації

DOI

Андрій Шилік,
Сергій Пузирьов

СИСТЕМА КЕРУВАННЯ ДОСТУПОМ НА БАЗІ RFID ТА SPRING BOOT

Тези представляють аналіз актуальності та перспектив розробки системи керування доступом(СКД) на основі технології радіочастотної ідентифікації (англ. Radio Frequency IDentification, RFID) та фреймворку Spring Boot. У роботі розглянуто принцип роботи RFID, досліджено сучасні СКД, проведено порівняльний аналіз RFID з альтернативними технологіями, з оцінкою їхньої безпеки, зручності та вартості. Визначено вимоги до СКД, що забезпечує автоматизовану ідентифікацію користувачів, централізоване управління доступом та інтеграцію з IoT.

Ключові слова: *система керування доступом(СКД), RFID(Radio Frequency IDentification), безпека, тайм-менеджмент.*

У наший час системи керування доступом (СКД) є важливою складовою в забезпеченні надійного захисту об’єктів, що потребують обмеженого доступу, таких як офіси, склади, навчальні заклади чи житлові комплекси, вимагає розробки ефективних, надійних і доступних СКД. Поряд із цим, важливим аспектом є оптимізація тайм-менеджменту, оскільки швидке й зручне адміністрування доступу сприяє економії часу, що в сучасних економічних умовах трансформується у фінансові вигоди. Традиційні методи, такі як механічні замки чи паролі, мають обмеження, пов’язані з безпекою, зручністю використання та можливістю масштабування. Отже, розробка інноваційних підходів у цій галузі є актуальною для підвищення рівня безпеки та вдосконалення операційної ефективності організацій.

Актуальність теми кваліфікаційної бакалаврської роботи визначається необхідністю розробки економічно доступних і високоефективних рішень для керування доступом, які поєднують апаратне забезпечення з програмними засобами, забезпечуючи не лише безпеку, а й оптимізацію часу адміністрування.

Метою дослідження є створення СКД, яка забезпечує автоматизовану ідентифікацію користувачів і керування механізмом доступу шляхом розробки апаратно-програмного забезпечення (АПЗ) на основі RFID з вебзастосунком на базі Spring Boot

Технологія радіочастотної ідентифікації RFID являє собою сучасний метод безконтактної ідентифікації об'єктів, що ґрунтується на використанні радіохвиль. RFID-системи набули широкого застосування у сферах логістики, роздрібної торгівлі, охорони здоров’я, транспортній інфраструктурі та інших галузях завдяки їхнім можливостям забезпечення оперативного, автоматизованого та надійного зчитування даних.

Основні складові RFID-систем:

- RFID-мітка: ідентифікаційна мітка, будучи первинним носієм інформації в RFID-системі, конструктивно включає інтегральну мікросхему для зберігання унікального ідентифікаційного коду та, за потреби, додаткових даних, а також антену, що забезпечує комунікацію зі зчитувачем; залежно від джерела енергії, мітки поділяються на пасивні, які не мають власного живлення та активуються електромагнітним полем зчитувача, характеризуючись компактністю, нижчою вартістю та обмеженим радіусом дії (до 10 м); активні мітки, що містять автономне джерело живлення, забезпечуючи функціонування на значних відстанях (до 100 м і більше) та можливість ініціювання передачі даних; і напівпасивні мітки, які використовують батарею для живлення мікросхеми, але для передачі сигналу залежать від електромагнітного поля зчитувача;
- зчитувальний пристрій: зчитувач являє собою пристрій, що генерує електромагнітне поле для активації мітки та здійснює прийом радіосигналів, що нею транслюються. Зчитувачі можуть бути стаціонарними (наприклад, інстальованими на

контрольно-пропускних пунктах складських приміщень) або мобільними (портативні сканери). Їхня конструкція включає антени для передачі та прийому сигналів, а також модулі для первинної обробки даних та їхньої подальшої передачі до інформаційної системи;

– інформаційна система: дані, отримані від мітки за посередництвом зчитувача, надходять до комп'ютеризованої системи для подальшої обробки, зберігання та аналізу. Зазначена система може бути інтегрована з базами даних, системами управління складськими запасами (англ. Warehouse Management System) або іншими програмними платформами з метою автоматизації бізнес-процесів.

Схематичне зображення складових RFID-системи (рис. 1).

Рис. 1. Схематичне зображення складових RFID та їх взаємодія

Функціонування RFID-систем ґрунтується на принципі обміну радіосигналами між ідентифікаційною міткою та зчитувальним пристроєм у визначеному частотному діапазоні.

Процес ідентифікації включає наступні етапи:

– генерація електромагнітного поля: зчитувач за допомогою власної антени створює електромагнітне поле в межах операційної зони. Частота даного поля визначається типом використовуваної RFID-системи (низькочастотна, високочастотна або ультрависокочастотна);

– активація мітки: при потраплянні RFID-мітки в зону дії електромагнітного поля, її антена абсорбує енергію. У пасивних мітках отримана енергія використовується для живлення мікросхеми, що ініціює перехід мітки в активний стан. Активні мітки, маючи автономне джерело енергії, здатні самостійно ініціювати процес передачі даних;

– трансляція даних: активована мітка передає збережені дані (наприклад, унікальний ідентифікатор) назад до зчитувача за допомогою радіосигналу. Даний процес отримав назву модуляції сигналу. У більшості систем застосовується принцип зворотного розсіювання (англ. backscatter), при якому мітка здійснює модуляцію відбитого сигналу зчитувача, накладаючи на нього власну інформацію;

– обробка даних: зчитувач здійснює декодування отриманого сигналу, трансформуючи його в цифровий формат, та передає отримані дані до інформаційної системи. Система, у свою чергу, обробляє отриману інформацію, співставляючи її з даними, що зберігаються в базі даних, та ініціює відповідні дії, такі як актуалізація даних про інвентар, верифікація доступу або моніторинг переміщення об'єкта.

Блок-схема принципу роботи RFID-систем (рис. 2).

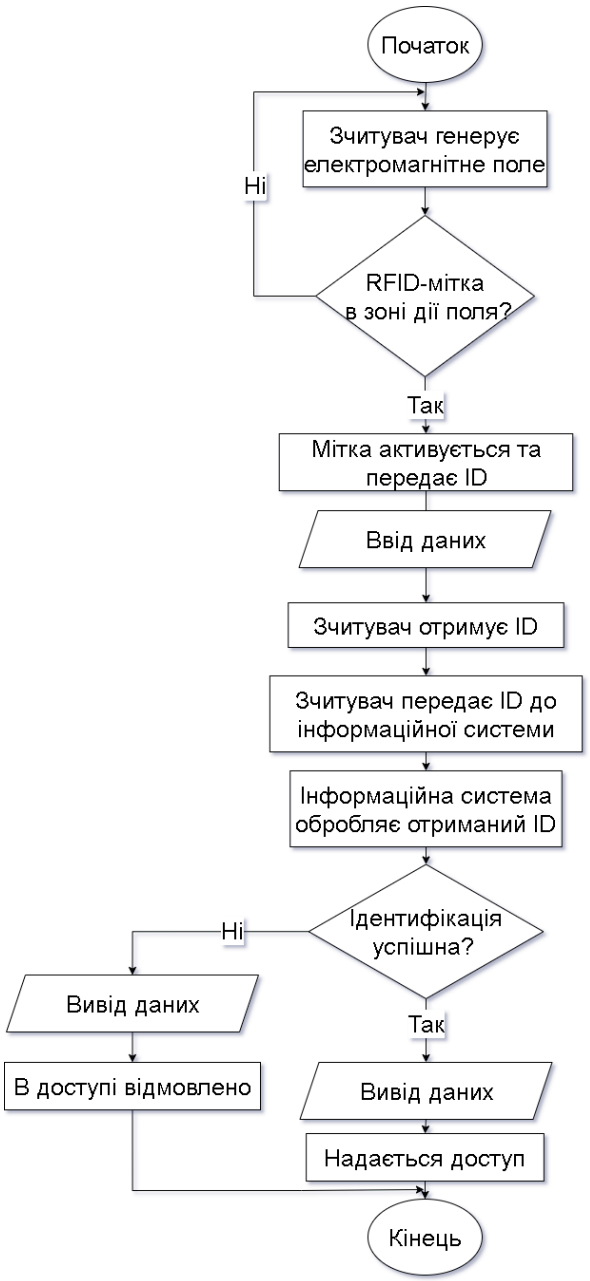

Рис. 2. Блок-схема, що ілюструє принцип роботи RFID-системи

Аналізуючи доступну інформацію, можна виділити кілька ключових типів систем, які конкурують із RFID, і зрозуміти, чому вони є поширеними. СКД класифікуються залежно від архітектури, способу ідентифікації, рівня безпеки та сфери застосування.

Вибір конкретної СКД визначається комплексом факторів, серед яких ключову роль відіграє контекст її застосування (табл. 1). Біометричні системи набули поширення завдяки високому рівню безпеки, що забезпечується унікальністю та невідчужуваністю біологічних ідентифікаторів, що є критично важливим для об'єктів з високим рівнем ризику. Мобільні системи демонструють зростаючу популярність завдяки зручності використання та можливості оперативного керування правами доступу через програмні інтерфейси, що є значною перевагою для великих організацій. Кодові замки приваблюють своєю простотою експлуатації та відсутністю потреби у додаткових фізичних носіях, що знижує операційні витрати. Системи з магнітними картками, незважаючи на певні обмеження в безпеці, залишаються затребуваними завдяки їхній економічності та сумісності з існуючою інфраструктурою, особливо в об'єктах старої забудови. Фізичні ключі зберігають свою

актуальність у низькотехнологічних середовищах, де простота та автономність є визначальними факторами.

Таблиця 1.

Порівняння основних СКД

№ з/п	Система	Рівень безпеки	Зручність	Вартість	Приклади використання
1	Біометричні системи	Високий	Середній	Висока	Банки, урядові установи
2	Мобільні системи	Середній-Високий	Висока	Середня	Офіси, житлові комплекси
3	Кодові замки	Низький-Середній	Висока	Низька	Малі офіси, гаражі
4	Магнітні картки	Низький	Середній	Низька	Старий офіс, готелі
5	Фізичні ключі	Низький	Низька	Дуже низька	Приватні будинки, малі об’єкти

Нефункціональні вимоги є основою для забезпечення стабільності, безпеки та зручності використання СКД. Вони сприяють створенню гнучкої та масштабованої платформи, що відкриває можливості для подальшого розвитку функціоналу (табл. 2).

Таблиця 2.

Нефункціональні вимоги

№ з/п	Вимога	Опис
1	Продуктивність	Система має обробляти запити на доступ протягом 1 секунди
2	Вартість	Компоненти в сумі не мають бути дорожчими 2’000–3’000 грн
3	Надійність	СКД забезпечує доступність на рівні 99,9 % часу завдяки резервному копіюванню даних і механізмам відновлення після збоїв
4	Безпека	СКД використовує шифрування даних, безпечну автентифікацію, ролевий контроль доступу через Spring Security

Функціонування системи ґрунтується на наступному алгоритмі: RFID-модуль ініціює процес зчитування ідентифікаційної інформації з безконтактної картки. Отримані дані обробляються мікроконтролером, який формує відповідний запит та передає його на сервер через бездротову мережу Wi-Fi. Серверний застосунок на базі Spring Boot здійснює верифікацію прав доступу на основі отриманих ідентифікаційних даних та повертає результат авторизації. На основі отриманої відповіді, мікроконтролер активує сервопривід для забезпечення фізичного розблокування (або підтримує його в заблокованому стані). Локальний дисплей та клавіатура використовуються для конфігурації параметрів мережевого з'єднання з сервером, зокрема для введення IP-адреси. Таким чином, апаратна та програмна складові системи функціонують в режимі реального часу, забезпечуючи централізоване управління доступом до контрольованих зон.

Список використаних джерел

1. Zhang Z., Yan C. Research on data sharing access control based on blockchain technology. *Autom Control Intell Syst.* 2022. Vol. 10, Is. 1. P. 8–13. DOI: 10.11648/j.acis.20221001.12.

2. Hassan S., Shibli A. R., Raza R. Ubiquitous computing with Radio Frequency IDentification tags. *Math Comput Sci.* 2023. Vol. 8, Is. 3. P. 73–86. DOI: 10.11648/j.mcs.20230803.12.

3. Aliyu M. B., Suru H., Gabi D., Garba M., Argungu M. The need for adaptive access control system at the network edge. *American Journal of Information Science and Technology.* 2024. Vol. 8, Is. 2. P. 45–55. DOI: 10.11648/j.ajist.20240802.13.
4. RFID технологія для автоматизації обліку. URL: https://remonline.ua/blog/rfid-technology/ (дата звернення: 10.05.2025).
5. Що таке система RFID, в чому її особливості використання. URL: https://idcard.com.ua/ua/blog/chto-takoe-sistema-rfid-v-chem-ee-osobennosti-ispolzovaniya/?srsltid=afmbooptmjjvbpzudevvj7uwvfhapvaoh1zk_voamfcodjsmtc9u1i1b (дата звернення: 10.05.2025).

References

1. Zhang, Z., & Yan, C. (2022). Research on data sharing access control based on blockchain technology. *Autom Control Intell Syst,* 10 (1), (pp. 8–13). DOI: 10.11648/j.acis.20221001.12.
2. Hassan, S., Shibli, A. R., & Raza, R. (2023). Ubiquitous computing with Radio Frequency IDentification tags. *Math Comput Sci,* 8 (3), (pp. 73–86). DOI: 10.11648/j.mcs.20230803.12.
3. Aliyu, M. B., Suru, H., Gabi, D., Garba, M., & Argungu, M. (2024). The need for adaptive access control system at the network edge. *American Journal of Information Science and Technology,* 8 (2), (pp. 45–55). DOI: 10.11648/j.ajist.20240802.13.
4. RFID technology for automation of accounting. (2025). Retrieved from: https://remonline.ua/blog/rfid-technology/ [In Ukrainian].
5. What is an RFID system, what are the features of its use? (2019). Retrieved from: https://idcard.com.ua/ua/blog/chto-takoe-sistema-rfid-v-chem-ee-osobennosti-ispolzovaniya/?srsltid=afmbooptmjjvbpzudevvj7uwvfhapvaoh1zk_voamfcodjsmtc9u1i1b [In Ukrainian].

DOI

Andrii Shylik,
Serhii Puzyrov

ACCESS CONTROL SYSTEMBASED ON RFID AND SPRING BOOT

The abstract presents an analysis of the relevance and prospects of developing an access control system (ACS) based on Radio Frequency Identification (RFID) technology and the Spring Boot framework. The study examines the operating principles of RFID, investigates modern ACS, and conducts a comparative analysis of RFID with alternative technologies, evaluating their security, convenience, and cost. Requirements for the ACS are defined, ensuring automated user identification, centralized access management, and integration with IoT.

Key words: *access control system (ACS), Radio Frequency IDentification (RFID), security, time management.*

Відомості про авторів / Information about the Authors

Андрій Шилік, бакалаврант кафедри комп'ютерної інженерії, Чорноморський національний університет імені Петра Могили, м. Миколаїв, Україна. E-mail: andreishulik@gmail.com, orcid: https://orcid.org/0009-0007-2318-1245.

Andrii Shylik, BSc Student of the Computer Engineering Department, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine. E-mail: andreishulik@gmail.com, orcid: https://orcid.org/0009-0007-2318-1245.

Сергій Пузирьов, кандидат фізико-математичних наук, доцент кафедри комп'ютерної інженерії факультету комп'ютерних наук Чорноморського національного університету імені Петра Могили, м. Миколаїв, Україна. E-mail: svpuzyrev@gmail.com, orcid: https://orcid.org/0000-0003-0520-1496.

Serhii Puzyrov, Candidate of Physical and Mathematical Sciences, Associate Professor, Associate Professor of the Computer Engineering Department, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine. E-mail: svpuzyrev@gmail.com, orcid: https://orcid.org/0000-0003-0520-1496.