Галицький фаховий коледж імені В'ячеслава Чорновола
відділення комп'ютерних технологій
циклова комісія інформатики та комп'ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп'ютерних технологій
Наталія СТЕФУРАК _____
(підпис)
«____» _____ 2025р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 123 «Комп'ютерна інженерія»
на тему:
«Автоматична система контролю доступу автотранспорту
на закриту територію»

Студент групи КІ-41	Олег КИРИЧ	_____ (підпис)
Керівник роботи	Василь ПАВЛЮС	_____ (підпис)
Консультанти:		
з техніко-економічного обґрунтування	Любов МЕЛЕНЧУК	_____ (підпис)
Нормоконтролер	Василь КУЗИК	_____ (підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В'ячеслава Чорновола

відділення комп'ютерних технологій

циклова комісія інформатики та комп'ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділення
комп'ютерних технологій
Наталія СТЕФУРАК/_____/
« ___» _____2024р.

ЗАВДАННЯ

на кваліфікаційну роботу
на здобуття освітньо-кваліфікаційного рівня «фаховий молодший бакалавр»
студенту Киричу Олегу Володимировичу

1. Тема кваліфікаційної роботи: Автоматична система контролю доступу автотранспорту на закриту територію затверджено наказом по коледжу
Від "25"11 2024р.,№ 253
2. Термін здачі студентом завершеної роботи "24" 06 2025р.
3. Вихідні дані до роботи: актуальні технології нові методи ініціалізації об'єктів. Актуальні технології контролю доступу
4. Перелік питань, які повинні бути розроблені в роботі:
а) основна частина: аналіз предметної області та наявних рішень, формалізація вимог до системи, алгоритми взаємодії користувача з системою, функціонування та взаємодія основних компонентів системи, технічні аспекти реалізації основних підсистем, налаштування апаратних ресурсів, реалізація та тестування системи
б) техніко-економічне обґрунтування: обгрунтування розробки та розрахунок витрат напроектування Система має невисоку собівартість, проста в монтажі й обслуговуванні, що робить її придатною для побутового та малого комерційного використання. Вона є технічно доцільною, економічно вигідною і конкурентоспроможною

5. Перелік графічного матеріалу: система контролю доступу компанії, Nedap (RFID), пристрій біометричного терміналу, система доступу компанії Hid global (NFC), структурна схема системи, ER-діаграма, логічне проєктування бази даних, мікроконтролер ESP-WROOM-32, кроковий двигун 28BYJ-48, RFID-зчитувач Mfrc522, електрична принципова схема виконавчого механізму воріт, електрична принципова схема Rfid-зчитувача

6. Консультанти роботи:

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
З техніко-економічного обгрунтування	_____ (П.І.Б. консультанта)	26.04.2025	13.05.2025

КАЛЕНДАРНИЙ ПЛАН

Виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Термін	
		початку	завершення
1.	Вибір теми, ознайомлення з вимогами кваліфікаційної роботи.	23.11.2024	25.11.2024
2.	Дослідження предметної області, огляд наявних рішень.	26.11.2024	10.01.2025
3.	Проєктування системи	11.01.2025	03.03.2025
4.	Встановлення та налаштування середовища реалізації.	04.03.2025	10.03.2025
5.	Реалізація системи.	11.03.2025	25.04.2025
6.	Опрацювання економічної частини кваліфікаційної роботи	26.04.2025	13.05.2025
7.	Тестування системи та усунення недоліків	14.05.2025	28.05.2025
8.	Робота над оформленням пояснювальної записки	29.05.2025	15.06.2025
9.	Попередній захист кваліфікаційної роботи	16.06.2025	16.06.2025
10.	Захист кваліфікаційної роботи	26.06.2025	26.06.2025

7. Дата видачі «__»_____2024р. Керівник _____/ Василь ПАЛЮС

Завдання прийнято до виконання _____/Олег КИРИЧ

Реферат

Кваліфікаційна робота. Автоматична система контролю доступу автотранспорту на закритій території. 83с., рисунків 11, 6 додатки, 5 джерел.

Метою проєкту є створення комплексного рішення, що поєднує апаратні й програмні засоби для забезпечення зручного, безпечного та ефективного контролю в'їзду й виїзду транспортних засобів.

Об'єкт дослідження - процес автоматичного контролю доступу транспортних засобів на обмежену територію.

Результати роботи - розроблено систему контролю доступу на основі ESP32, RFID-зчитувача, камери та сервера з FastAPI. Реалізовано розпізнавання номерних знаків і керування воротами через MQTT.

Новизна роботи полягає у поєднанні відкритих технологій у єдину доступну систему, що дозволяє автоматизувати доступ з мінімальною участі людини.

Практичне застосування - система підходить для використання на приватних територіях, підприємствах, складах і парковок.

Система може бути адаптована для різних умов експлуатації, розширена додатковими модулями або інтегрована з іншими рішеннями які для реалізації використовую MQTT.

Ключові слова: КОНТРОЛЬ ДОСТУПУ, РОЗПІЗНАВАННЯ НОМЕРНИХ ЗНАКІВ, ВЕБСАЙТ, БЕЗПЕКА, АВТОМАТИЧНІ ВОРОТА.

Abstract

Qualification work. Automatic vehicle access control system for a restricted area. 83 pages, 11 figures, 6 appendices, 5 sources. The purpose of the project is to create a comprehensive solution that combines hardware and software tools to ensure convenient, safe, and effective control of vehicle entry and exit.

Object of research – the process of automatic access control of vehicles to a restricted area.

Results of the work – an access control system based on ESP32, RFID reader, camera, and a FastAPI server has been developed. License plate recognition and gate control via MQTT have been implemented.

Novelty of the work lies in combining open technologies into a single accessible system that enables automated access with minimal human involvement.

Practical application – the system is suitable for use on private properties, enterprises, warehouses, and parking lots.

The system can be adapted to various operating conditions, expanded with additional modules, or integrated with other solutions that use MQTT.

Keywords: ACCESS CONTROL, LICENSE PLATE RECOGNITION, WEBSITE, SECURITY, AUTOMATIC GATES.

ЗМІСТ

СКОРОЧЕННЯ І УМОВНІ ПОЗНАКИ

ANPR - Automatic Number Plate Recognition.

GPU - Graphics Processing Unit.

MQTT - Message Queuing Telemetry Transport.

RIFD - Radio Frequency Identification.

SPI - Serial Peripheral Interface.

WIFI - Wireless Fidelity.

ВСТУП

У сучасному світі питання безпеки стають дедалі актуальнішими в умовах зростаючої урбанізації, збільшення кількості приватних підприємств, стратегічних об’єктів і закритих територій. Інформаційні технології відіграють ключову роль у забезпеченні надійного контролю та моніторингу таких об'єктів. Особливої актуальності набувають автоматичні системи контролю доступу, які дозволяють організувати безпечний і керований вхід і вихід на території з обмеженим доступом. Такі системи широко застосовуються на промислових підприємствах, в офісних центрах, державних установах, житлових комплексах та інших об’єктах, де необхідно забезпечити захист від несанкціонованого проникнення та контроль за переміщенням персоналу і відвідувачів.

Автоматичні системи контролю доступу поєднують у собі програмно-апаратні засоби, які працюють у режимі реального часу, взаємодіючи з різноманітними пристроями: зчитувачами карток або RFID-міток, камерами відеоспостереження, електромеханічними замками, шлагбаумами тощо. Завдяки використанню сучасних алгоритмів обробки даних, технологій ідентифікації, такі системи не лише забезпечують високий рівень безпеки, але й спрощують процеси обліку та управління персоналом, дозволяючи вести журнал подій і аналітику.

З урахуванням постійного зростання вимог до надійності й автономності систем безпеки, виникає необхідність створення комплексних рішень, що можуть адаптуватися до конкретних умов експлуатації, бути масштабованими та зручними у використанні. У цьому контексті розробка власної автоматизованої системи контролю доступу набуває практичної цінності, адже дозволяє врахувати специфічні потреби об'єкта, забезпечити гнучке управління доступом, інтегрувати додаткові функції.

Метою даної роботи є розробка ефективної, надійної та масштабованої автоматизованої системи контролю доступу на закриту територію, яка

забезпечить ідентифікацію користувачів, реєстрацію входів та виходів у базі даних і безперебійний доступ лише авторизованим особам. Розробка передбачає як створення апаратної складової, так і реалізацію програмного забезпечення з урахуванням сучасних вимог до зручності, швидкодії та інформаційної безпеки.

Для досягнення цієї мети в роботі вирішуються наступні завдання:

1) аналіз існуючих систем контролю доступу та їх функціональних можливостей;
2) розробка програмного забезпечення системи;
3) реалізація алгоритмів ідентифікації користувачів;
4) тестування та оптимізація функціональних можливостей системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Аналіз предметної області

Автоматична система контролю доступу є важливою складовою безпеки на закритих територіях, таких як промислові підприємства, військові бази, комерційні підприємства та державні установи. Основними завданнями систем контролю доступу є ідентифікація людей, дозвіл або обмеження доступу на територію та ведення обліку подій для аналізу та звітності.

Такі системи працюють, перевіряючи особу користувача, порівнюючи її з базою даних дозволів на доступ і вирішуючи, чи надавати доступ. Можуть використовуватися різні методи ідентифікації, включаючи картки доступу, біометричні дані, коди та мобільні пристрої. Сучасні системи дозволяють встановлювати рівні доступу, обмежуючи пересування персоналу та відвідувачів на об'єкті відповідно до їхніх повноважень.

Автоматизований контроль доступу не тільки підвищує безпеку, але й спрощує управління процесом входу і виходу, зменшуючи потребу в постійних фізичних перевірках. Детальні записи про пересування можна зберігати і використовувати для аналізу, виявлення підозрілої поведінки та управління робочим часом співробітників. Крім того, інтеграція систем контролю доступу з іншими системами безпеки, такими як відеоспостереження та сигналізація, дозволяє комплексно підійти до управління ризиками і швидко реагувати на потенційні загрози. У цьому проєкті система контролю доступу передбачає автоматизацію роботи воріт і має забезпечувати контроль в'їзду транспортних засобів на територію та виїзду з неї.

Щоб зрозуміти, яка технологія є найкращою для автоматичного контролю доступу на закриту територію, варто розглянути рішення, доступні на ринку. Основна відмінність між ними полягає в методах ідентифікації користувачів, таких як RFID-картки, біометричні сканери,

розпізнавання номерних знаків. Кожен з цих підходів має переваги та обмеження залежно від вимог безпеки, простоти використання та вартості впровадження. Далі розглянемо основні методи ідентифікації, що використовуються в сучасних системах контролю доступу.

Сучасні рішення в цій сфері використовують різні технології та методи, кожен з яких має свої особливості, переваги та обмеження. Однією з найбільш поширених і зручних технологій є використання безконтактних ідентифікаторів у вигляді RFID-карток або спеціальних міток, які дозволяють користувачам швидко і легко ідентифікувати себе без необхідності вводити паролі або використовувати механічні ключі. Завдяки радіочастотній ідентифікації такі системи значно прискорюють і спрощують процес аутентифікації, передаючи дані між карткою і зчитувачем без необхідності фізичного контакту.

Кожен користувач отримує персональну картку або мітку з унікальним ідентифікатором, який зчитується при наближенні до відповідного пристрою. Потім система автоматично аналізує отриману інформацію і приймає рішення про надання доступу або його обмеження. Це значно підвищує рівень контролю за пересуванням людей і є корисним у різних секторах, включаючи бізнес-сектор, промислові організації, державні установи і навіть житлові комплекси, не тільки через швидкість і простоту використання, але й через те, що веде детальний облік відвідувань.

Широке розповсюдження таких систем пояснюється не лише їх зручністю, але й високим рівнем безпеки: Використання технології RFID надає кожній картці унікальний код, що ускладнює її злам та підробку, а також знижує ризик підробки та шахрайського використання. Крім того, такі системи можна легко інтегрувати з іншими рішеннями безпеки, такими як відеоспостереження, сигналізація або біометрія, забезпечуючи таким чином комплексний підхід до захисту території.

Зручність і швидкість таких рішень значно зменшує навантаження на персонал служби безпеки. Це особливо важливо на великих об'єктах, де потрібні ефективні та надійні механізми контролю доступу.

Головною перевагою такого підходу є неймовірно швидка обробка інформації, що дозволяє зчитувати дані практично миттєво без будь-яких додаткових дій з боку користувача. Завдяки такому механізму значно спрощується процес контролю доступу, оскільки відпадає необхідність витрачати час на введення паролів, пошук ключів та інші рутинні операції. Це робить систему надзвичайно зручною та простою у використанні. Користувачам не потрібно турбуватися про запам'ятовування складних комбінацій або забування паролів.

Такий підхід значно підвищує ефективність системи в цілому і є особливо важливим у місцях з високим трафіком, таких як офісні центри, підприємства, транспортні вузли та багатоквартирні будинки з обмеженим доступом. Велика кількість користувачів може проходити через контроль швидко і безперешкодно, без затримок і черг. Ця технологія також підходить для адміністраторів, оскільки дозволяє автоматизувати багато процесів, пов'язаних з моніторингом і контролем доступу.

Незважаючи на очевидні переваги, цей метод також має недоліки, які впливають на безпеку всієї системи. Одним з основних ризиків є можливість втрати картки або мітки, що автоматично створює потенційну загрозу несанкціонованого доступу. Якщо несанкціонована особа отримує доступ до загубленої картки, вона може безперешкодно користуватися нею, що може призвести до небажаних наслідків. Існує також можливість крадіжки картки, особливо якщо власник картки не приділяє достатньої уваги безпеці та зберігає її у легкодоступному місці.

Ще одним недоліком таких систем є ризик дублювання карток за допомогою спеціальних пристроїв, які можуть копіювати інформацію та створювати ідентичні мітки. Це може призвести до компрометації всієї системи безпеки, оскільки зловмисник може використати підроблену

картку для доступу на об'єкт, що охороняється. Технологія радіочастотної ідентифікації працює безконтактно, тому можуть бути спроби перехопити сигнал і зчитати інформацію віддалено.

Іншим ефективним рішенням для ідентифікації користувачів є біометричні системи, які використовують фізіологічні та поведінкові характеристики людини, такі як відбитки пальців, розпізнавання обличчя, аналіз райдужної оболонки ока, ідентифікація голосу або форми долоні. Основною перевагою таких систем є їх точність і надійність, оскільки біометричні дані є унікальними для кожної людини і їх майже неможливо підробити або передати іншим особам.

Завдяки цій особливості біометричні технології широко використовуються в різних галузях, включаючи фінансові установи, державні органи, корпоративні системи безпеки, мобільні пристрої і навіть транспортний сектор. Наприклад, сучасні смартфони оснащені технологією розпізнавання обличчя та відбитків пальців, що забезпечує швидкий і легкий доступ до пристроїв без необхідності вводити паролі. Біометричні сканери також були впроваджені в аеропортах для прискорення паспортного контролю.

Однак, незважаючи на всі ці переваги, біометричні системи мають деякі недоліки. Наприклад, зовнішні фактори, такі як зміни в зовнішності людини, погане освітлення та фізичні пошкодження біометричних характеристик можуть вплинути на точність розпізнавання. Крім того, з точки зору конфіденційності, використання біометричних даних викликає певне занепокоєння, оскільки, на відміну від паролів, біометричні дані не можуть бути змінені, а тому витік такої інформації може мати серйозні наслідки. Іншим важливим аспектом є вартість впровадження біометричної системи. Це пов'язано з тим, що біометричні системи вимагають спеціалізованого обладнання та складних алгоритмів обробки даних. Це робить використання таких рішень у великих масштабах дорогим, особливо в малих і середніх підприємствах та громадських організаціях.

Однак розвиток в галузі машинного навчання допомагають вдосконалювати біометричні системи та підвищувати їхню точність, надійність і зручність для широкого використання.

Завдяки прогресу в галузі машинного навчання нещодавно з'явилася нова технологія для ідентифікації користувачів. Використання машинного зору дозволяє не лише аналізувати біометричні дані, а й розпізнавати об'єкти, пов'язані з конкретними особами. Одним із таких напрямків є розпізнавання номерних знаків автомобілів.

Системи автоматичного розпізнавання номерних знаків (ANPR) використовують камери спостереження та алгоритми комп'ютерного зору для зчитування і аналізу символів на номерних знаках. Вони знаходять широке застосування в контролі доступу, організації платного паркування, правоохоронних органах та на митницях. Такі системи можуть працювати в реальному часі, що дозволяє швидко реагувати на порушення або автоматизувати перевірку транспортних засобів.

Основними перевагами ANPR є висока швидкість і точність роботи. Завдяки сучасним алгоритмам розпізнавання, навіть у складних умовах, таких як погане освітлення, забруднені номерні знаки або різні кути огляду, система здатна коректно визначати символи та співвідносити їх із базами даних. Це дозволяє автоматизувати контроль доступу на території підприємств, житлових комплексів або автостоянок, зменшуючи потребу в людському втручанні.

Проте існують певні виклики. Якість розпізнавання може залежати від технічних характеристик камер, стану дорожньої інфраструктури та регіональних особливостей номерних знаків. Крім того, питання конфіденційності та збереження отриманих даних викликають дискусії щодо правомірності використання таких систем без чітких регламентів.

Незважаючи на ці труднощі, технології розпізнавання номерних знаків продовжують розвиватися, інтегруючись з іншими системами безпеки та автоматизації. Завдяки поєднанню штучного інтелекту, великих

даних і хмарних обчислень ці рішення стають дедалі ефективнішими, відкриваючи нові можливості для контролю та управління транспортними потоками.

1.2 Огляд наявних рішень

Розглянемо приклади систем контролю доступу, їхню архітектуру, методи збору інформації та технології ідентифікації користувачів.

Серед існуючих компаній які є на ринку вибрано наступні компанії:

a) Nedap;

b) ZKTeco;

c) HID Global.

Nedap – компанія, що спеціалізується на RFID-рішеннях для контролю доступу, інтелектуальних системах контролю доступу та інтегрованих рішеннях для великих об'єктів. Продукція компанії відома своєю високою надійністю та масштабованістю. Перевагами систем компанії є гнучкість конфігурації, інтеграція з іншими системами безпеки та висока якість обладнання. Недоліками є висока ціна обладнання та складність конфігурації.

Їхня система контролю доступу орієнтована на автомобільних користувачів. До машини прикріпляють RFID-мітку, яка при наближені до пропускного пункту зчитується і перевіряється, чи має користувач доступ на дану територію (рис. 1.1).

Рисунок 1.1 – Система контролю доступу компанії Nedap (RFID)

Також компанія пропонує систему контролю доступу за технологією розпізнавання номерних знаків. Інформація про цю технологію у відкритому доступі відсутня. За потреби можна надіслати запит до компанії, щоб отримати матеріали.

ZKTeco – виробник біометричних систем контролю доступу (рис. 1.2), що пропонує широкий спектр пристроїв, включаючи розпізнавання відбитків пальців, обличчя та вен на долоні. Рішення компанії зосереджені на підвищенні безпеки та можливості інтеграції з іншими системами безпеки. Перевагами їхньої продукції полягає у доступна ціна, широкий асортимент біометричних пристроїв і простота використання. До недоліків можна віднести потенційні проблеми з точністю розпізнавання за несприятливих умов.

Рисунок 1.2 – Пристрій біометричного терміналу

Окрім біометричних терміналів, ZKTeco також пропонує різноманітні рішення для забезпечення безпеки, такі як контролери доступу, електромеханічні замки, зчитувачі карток та програмне забезпечення для управління системами контролю доступу.

Компанія активно впроваджує новітні технології у свої пристрої, включаючи алгоритми машинного навчання для підвищення точності розпізнавання, а також можливість використання хмарних сервісів для централізованого управління.

HID Global – один із провідних виробників рішень для контролю доступу, що спеціалізується на смарт-картах, мобільних ідентифікаторах та біометричних системах. Продукція компанії широко використовується в корпоративному, урядовому та фінансовому секторах завдяки високому рівню безпеки та надійності. Перевагами систем HID Global є підтримка мультифакторної автентифікації, можливість інтеграції з різними платформами та використання сучасних технологій, таких як мобільний доступ за допомогою смартфонів. До недоліків можна віднести високу вартість обладнання та необхідність складного налаштування для його ефективної роботи. Їхня система контролю доступу може використовуватися як у фізичних об'єктах, так і в цифровому середовищі. Наприклад, користувач може отримувати доступ до приміщення за допомогою смарт-карти або мобільного додатку, що використовує NFC або Bluetooth для ідентифікації (рис. 1.3).

Рисунок 1.3 – Система доступу компанії HID Global (NFC)

Системи контролю доступу використовують різні технології ідентифікації користувачів, зокрема RFID, біометрію та відеоаналітику. Вибір системи залежить від потреб користувача, балансу між безпекою, вартістю та можливістю інтеграції з іншими рішеннями.

1.3 Постановка завдання

Розробка системи контролю доступу має на меті вирішення проблеми безпеки, а також, за можливості, підвищення комфорту. На основі досліджених рішень система повинна забезпечувати швидкий, безпечний та надійний доступ для авторизованих транспортних засобів і осіб. Для забезпечення альтернативного способу ідентифікації планується використання RFID-технології, а для підвищення точності — застосування методів штучного інтелекту, зокрема розпізнавання номерних знаків.

Перейдемо до функціональних вимог пристрою. Система повинна автоматично розпізнавати номерні знаки за допомогою камер відеоспостереження, а також зчитувати RFID-картки і мітки для ідентифікації користувачів. Усі події проходу транспорту та осіб мають реєструватися в базі даних із можливістю ведення історії. Контроль доступу здійснюється на основі «білих» та «чорних» списків, а інтеграція з механічними воротами дозволить автоматизувати їх відкриття та закриття.

Для забезпечення стабільної роботи система повинна використовувати камери з високою роздільною здатністю, що дозволяють розпізнавати номерні знаки незалежно від погодних умов. RFID-зчитувач має бути сумісним із картками стандарту MIFARE. Серверна частина повинна зберігати інформацію про всі події, а також надавати адміністраторам можливість перегляду та управління даними через локальний. Передача та збереження інформації мають бути захищені за допомогою сучасних протоколів безпеки.

На основі проведеного аналізу сформульовано вимоги до майбутньої системи. Це дозволяє перейти до етапу проєктування, де будуть визначені структура системи та алгоритми її роботи.

2 ПРОЄКТУВАННЯ СИСТЕМИ

У цьому розділі буде описано проєктування системи контролю доступу на закриту територію. Проєктування включає визначення структури системи, вибір технологій, розробку алгоритмів роботи.

Проєктування системи базується на результатах аналізу предметної області та існуючих рішень, які були розглянуті в попередньому розділі.

2.1 Проєктування структури системи

Для ефективного проєктування системи необхідно детально проаналізувати технічне завдання, оскільки від цього залежить правильне визначення ключових вимог до системи, її компонентів та алгоритмів взаємодії. Основною функцією даного проєкту є автоматичне відкривання воріт за допомогою технології ANPR. Ця технологія дає змогу розпізнавати номерні знаки транспортних засобів у реальному часі, аналізуючи відеопотік, і на основі отриманих даних приймати рішення щодо доступу. Важливою перевагою такого підходу є висока швидкість і точність розпізнавання, що дозволяє мінімізувати необхідність у додаткових діях з боку водія або оператора системи.

Система складається з кількох ключових частин, кожна з яких виконує важливу функцію. Функція відеозахоплення відповідає за зчитування зображення транспортного засобу та його номерного знака, після чого отримане зображення передається до системи розпізнавання. Остання аналізує зображення, застосовуючи алгоритми машинного зору та штучного інтелекту, і звіряє отримані дані з базою транспортних засобів. Якщо система успішно знаходить відповідний запис у базі, подається сигнал на відкриття воріт. Однак у випадку, коли модель помиляється або не може правильно розпізнати номерний знак через погані умови зйомки, передбачено два додаткові способи контролю воріт. Перший варіант – ручне відкривання через Вебінтерфейс, що дозволяє оператору переглянути

отримане зображення та прийняти відповідне рішення щодо доступу. Другий варіант – напівавтоматичне відкривання за допомогою RFID-картки, яка прив'язана до конкретного користувача або транспортного засобу. Якщо власник має відповідний доступ, система ідентифікує картку та дозволяє проїзд.

Передача даних між пристроями здійснюватиметься за протоколом MQTT. Цю технологію було обрано через її зручність та надійність. Задля наочності створено структурну схему (рис. 2.1).

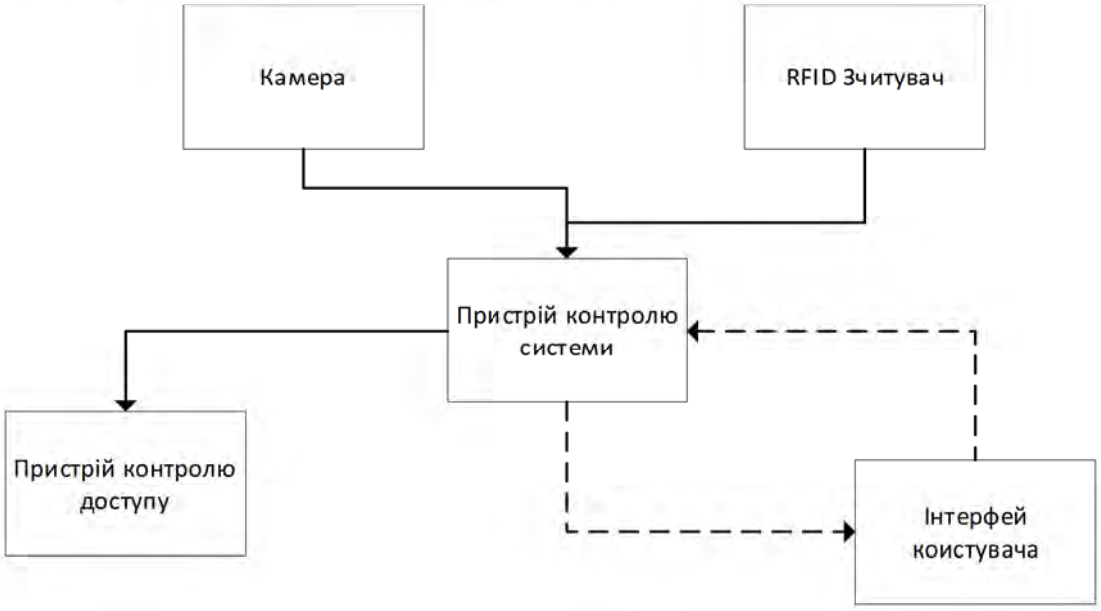


Рисунок 2.1 – Структурна схема системи

Завдяки цій схемі можна чітко простежити взаємозв'язок між основними компонентами системи, визначити маршрути передачі даних та оцінити загальну логіку функціонування.

2.2 Алгоритм роботи системи контролю доступу на закриту територію

Алгоритм роботи системи відіграє ключову роль у структуризації інформації та впорядкуванні всіх процесів, що забезпечують ефективне функціонування автоматизованого контролю доступу. Зважаючи на те, що система повинна виконувати декілька задач одночасно, її робота

організована у вигляді трьох паралельних процесів, які взаємодіють між собою та забезпечують безперебійний контроль доступу

Першим процесом є обробка відеопотоку та розпізнавання номерних знаків. Камера фіксує транспортний засіб, що наближається до контрольно-пропускного пункту, та надсилає отримане зображення до моделі розпізнавання. Застосовуючи алгоритми комп'ютерного зору та нейронні мережі, система аналізує отримане зображення, ідентифікує номерний знак та передає його для зіставлення з базою даних. Якщо дані збігаються з записами в базі даних, тоді ворота відкриваються..

Другим процесом є робота з RFID-міткою, яка виступає додатковим інструментом ідентифікації для транспортних засобів або водіїв. Під час наближення автомобіля система активує RFID-зчитувач, який виявляє наявність картки у визначеному радіусі та отримує унікальний ідентифікатор. Далі цей код звіряється з даними в базі даних для ідентифікації, якщо дані відповідаю що в базі даних тоді відкриваємо ворота.

Третій процес пов'язаний з роботою з інтерфейсом користувача, який забезпечує адміністрування системи, відображення поточного статусу об'єктів та управління доступом у разі потреби. Через вебінтерфейс користувач має можливість перевіряти журнал подій, додавати або змінювати дані в базі даних, а також вручну відчиняти ворота для транспортних засобів, якщо система не змогла самостійно прийняти рішення. Даний функціонал є особливо корисним у ситуаціях, коли автоматичні способи ідентифікації не спрацювали або коли необхідний контроль доступу для спеціального транспорту. Для кращого розуміння алгоритму представимо у вигляді блок-схеми. Блок схему наведено в додатку А.

Після аналізу блок-схеми можна зробити висновок, що система контролю доступу працює на основі трьох паралельних процесів: розпізнавання номерного знака, зчитування RFID-мітки та взаємодії з

користувацьким інтерфейсом. Завдяки цьому забезпечується гнучкість, надійність та можливість ручного керування у разі збоїв автоматичної ідентифікації.

2.3 Проєктування бази даних

База даних системи контролю доступу проєктується для ефективного зберігання та обробки інформації про транспортні засоби, користувачів, RFID-картки та історію доступу. Основу складають декілька взаємопов'язаних таблиць, які забезпечують цілісність даних та швидкий пошук необхідної інформації.

Основною таблицею, з якою взаємодіятиме система, є таблиця транспортних засобів, де зберігаються дані про номерні знаки, марки, моделі, кольори автомобілів. Кожен запис має унікальний ідентифікатор, що дозволяє однозначно ідентифікувати транспортний засіб.

Таблиця користувачів містить інформацію про власників або водіїв транспортних засобів, включаючи їх імена, контактні телефони та електронні адреси. Ці дані використовуються для ідентифікації осіб, які мають право доступу на територію.

Для альтернативного методу ідентифікації передбачена таблиця RFID-карток, де зберігаються унікальні ідентифікатори карток, інформація про користувачів, до яких вони прив'язані. Це дозволяє використовувати RFID-картки як додатковий спосіб підтвердження права доступу.

Для кращої подальшої реалізації бази даних створюємо ER-діаграму яка покаже зв’язки між таблицями та їхні атрибути. Діаграма зображена на рисунку 2.2.

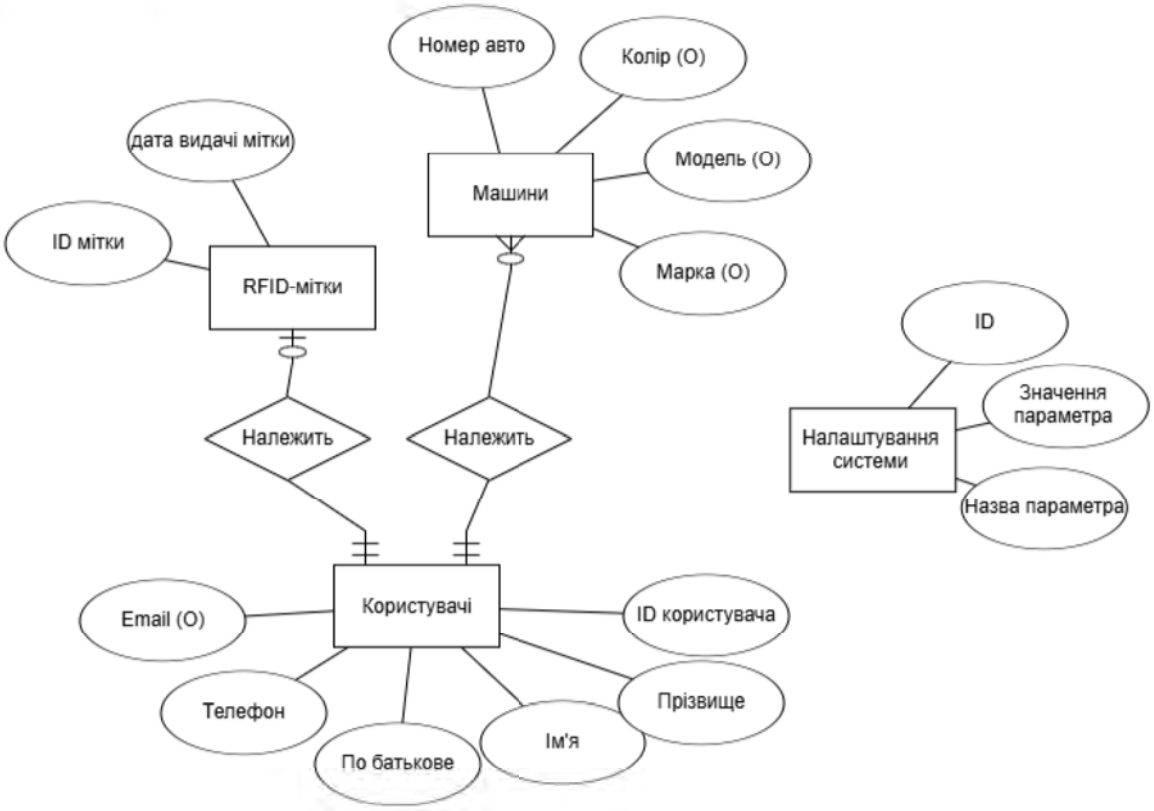


Рисунок 2.2 – ER-діаграма

Наступним кроком є логічне проєктування. У цьому етапі здійснюється деталізоване опрацювання структури бази даних з урахуванням вимог предметної області. Зокрема, на цьому етапі встановлюється, як саме пов’язані між собою таблиці, а також елементи, що виступають ключовими ідентифікаторами.

Крім цього, для кожного поля в таблицях підбирається відповідний тип даних – з урахуванням обсягу інформації, яка буде зберігатися, і способу її обробки в подальшій роботі системи. Логічне проєктування дозволяє побудувати чітку й несуперечливу модель даних, яка буде основою для фізичного проєктування бази даних і реалізації функціоналу всієї інформаційної системи (рис. 2.3).

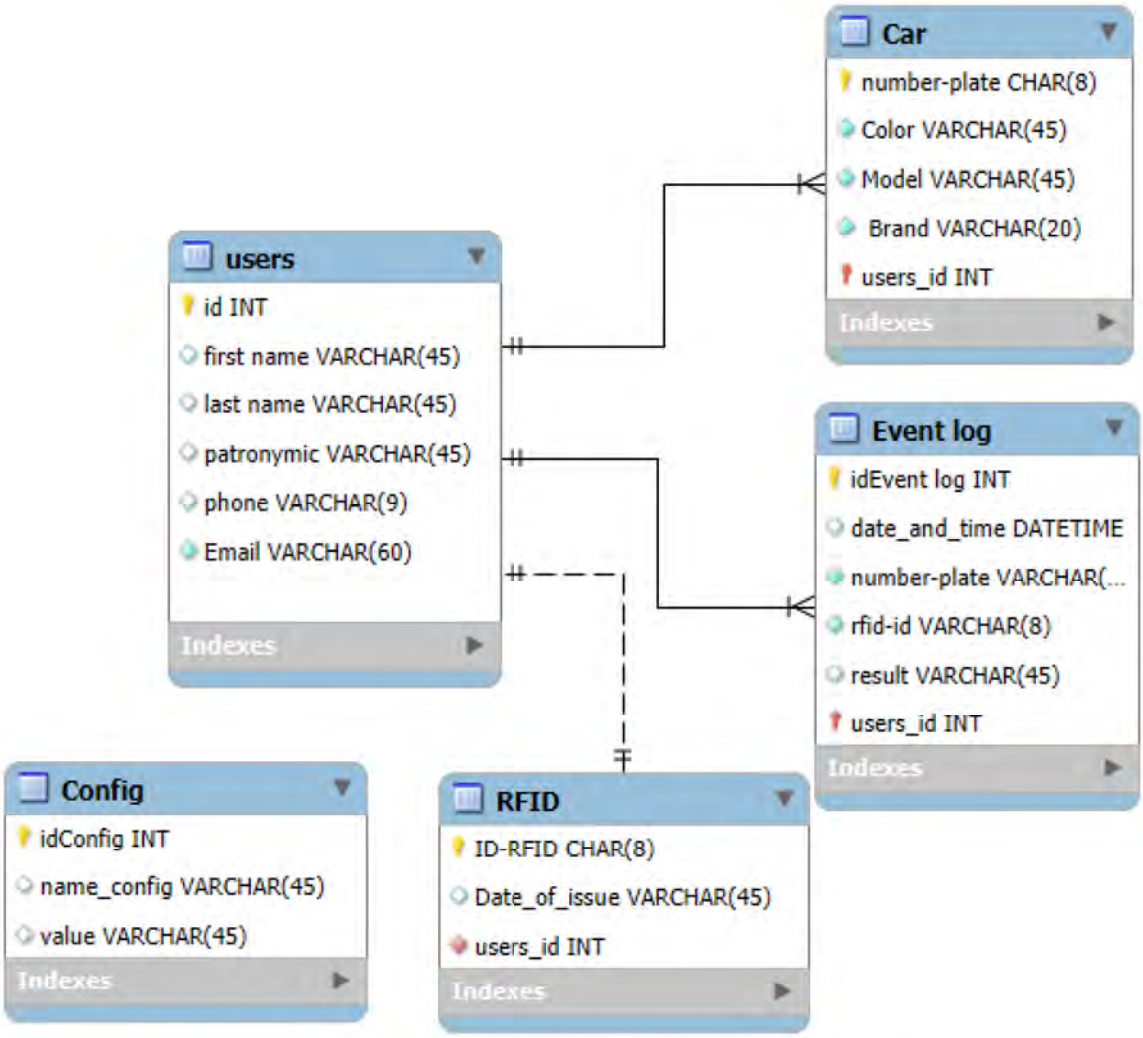


Рисунок 2.3 – Логічне проєктування бази даних

Після побудови логічної структури бази даних завершується етап проєктування системи. Усі компоненти, включаючи архітектуру, алгоритми роботи та модель даних, сформовані з урахуванням поставлених завдань і готові до етапу реалізації.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

У цьому розділі розглянемо детальний опис процесу реалізації системи, що була спроєктована на попередньому етапі. Реалізуємо програмне та апаратне забезпечення, обираємо інструменти розробки, а також проводимо тестування основних функціональних модулів. Метою цього розділу є демонстрація працездатності створеного рішення та підтвердження його відповідності технічним вимогам.

3.1 Апаратна реалізація системи

Апаратна частина автоматизованої системи контролю доступу включає в себе такі основні компоненти:

- Мікроконтролер, кроковий двигун для керування воротами.
- RFID-зчитувач для ідентифікації користувачів.
- Камеру для розпізнавання номерних знаків.
- Сервер, що забезпечує обробку даних і зберігання інформації. Всі ці елементи взаємодіють між собою для забезпечення надійного контролю доступу на закриту територію.

Оскільки система повинна забезпечувати передачу даних між окремими пристроями, доцільно використати бездротове з'єднання через мережу. Для реалізації цієї функції оптимальним варіантом є мікроконтролер серії ESP32, який має вбудований модуль WiFi, що дозволяє йому ефективно взаємодіяти з сервером без потреби у додатковому мережевому обладнанні (рис. 3.1).

Рисунок 3.1 – Мікроконтролер ESP-WROOM-32

Крім того, ESP32 має достатню обчислювальну потужність. Технічні характеристики мікроконтролера наведена в таблиці 3.1.

Таблиця 3.1 – Технічні характеристики мікроконтролера ESP-WROOM-32

Робоча напруга	2.6-3,3В
Частота мікроконтролера	До 240 МГц
Обсяг ОЗУ	520 КБ
Об'єм флеш пам'яті	4МБ
Підтримувані Wi-Fi мережі	802.11 b/g/n

Наступним компонентом, що використовується в проєкті, є кроковий двигун. В даній моделі системи використано 28BYJ-48 у комплекті з драйвером ULN2003 (рис. 3.2). Підключення здійснюється через драйвер, який приймає керуючі сигнали від мікроконтролера ESP32.

В реальній системі кроковий двигун 28BYJ-48 може бути легко замінений на потужніший з ідентичним підключенням, що не потягне за собою змін в програмній частині системи.

Рисунок 3.2 – Кроковий двигун 28BYJ-48

Даний двигун використовується для реалізації механічного керування завдяки можливості точного позиціонування, що є критично важливим для систем автоматизованого доступу.

Технічні характеристики 28BYJ-48 наведено в таблиці 3.2.

Таблиця 3.2 – Технічні характеристики крокового двигуна 28BYJ-48

Напруга живлення	5-12 В
Кількість фаз	4
Кількість мікрокроків	4096
Частота	100 Гц
Номінальна тяга	3500 г/см

Ще одним важливим компонентом системи є RFID-зчитувач MFRC522, який забезпечує безконтактну ідентифікацію користувачів за допомогою RFID-карток або міток (рис. 3.3). Зчитувач працює на частоті 13,56 МГц і підтримує протокол ISO/IEC 14443A. Підключення до мікроконтролера ESP32 здійснюється через інтерфейс SPI, що забезпечує надійну та швидку передачу даних. Використання RFID-технології дозволяє реалізувати ефективний і зручний спосіб авторизації, що не потребує фізичного контакту між користувачем та пристроєм.

Рисунок 3.3 – RFID-зчитувач MFRC522

Його застосування не лише підвищує зручність користування, а й сприяє підвищенню рівня безпеки, оскільки кожна картка має унікальний ідентифікатор, який може бути перевірено у програмному забезпеченні. Завдяки цьому RFID-модуль відіграє ключову роль у реалізації контролю доступу в системі. Технічні характеристики RFID-зчитувача наведено в таблиці 3.3.

Таблиця 3.3 – Технічні характеристики MFRC522

Напруга живлення	3.3 В
Робоча частота	13.56 МГц
Дальність зчитування	0 – 60 мм
Інтерфейс	SPI
Швидкість передачі даних	10 Мбіт/с

Окрім основних апаратних компонентів, система передбачає зв'язок із сервером, що виконує функції обробки та збереження даних. У ролі сервера може виступати будь-який комп'ютер або інший пристрій з відповідним програмним забезпеченням та мережею з'єднання. Таким чином, апаратна частина формує основу функціонування системи, забезпечуючи її взаємодію з користувачем і зовнішнім середовищем. Мінімальні системні потреби сервера наведені в таблиці 3.4.

Таблиця 3.4 – Мінімальні апаратні характеристики сервера

CPU	2-ядерний, не нижче Intel Core i3 або AMD Ryzen 3
RAM	Від 4 ГБ
Місце на диску	Від 10 ГБ

Розглянувши основні компоненти системи перейдемо до створення електричних схем компонентів.

Схеми реалізовано в середовищі Fritzing, яке дає змогу наочно моделювати з'єднання між компонентами, створювати електричні схеми та підготовлювати їх до подальшого монтажу.

Оскільки система складається з двох окремих фізичних пристроїв – модуля з RFID-зчитувачем та виконавчого модуля з кроковим двигуном – для повного представлення електричної частини було створено дві схеми.

Почнемо з виконавчого механізму. Його електрична принципова схема зображено на рисунку 3.4.

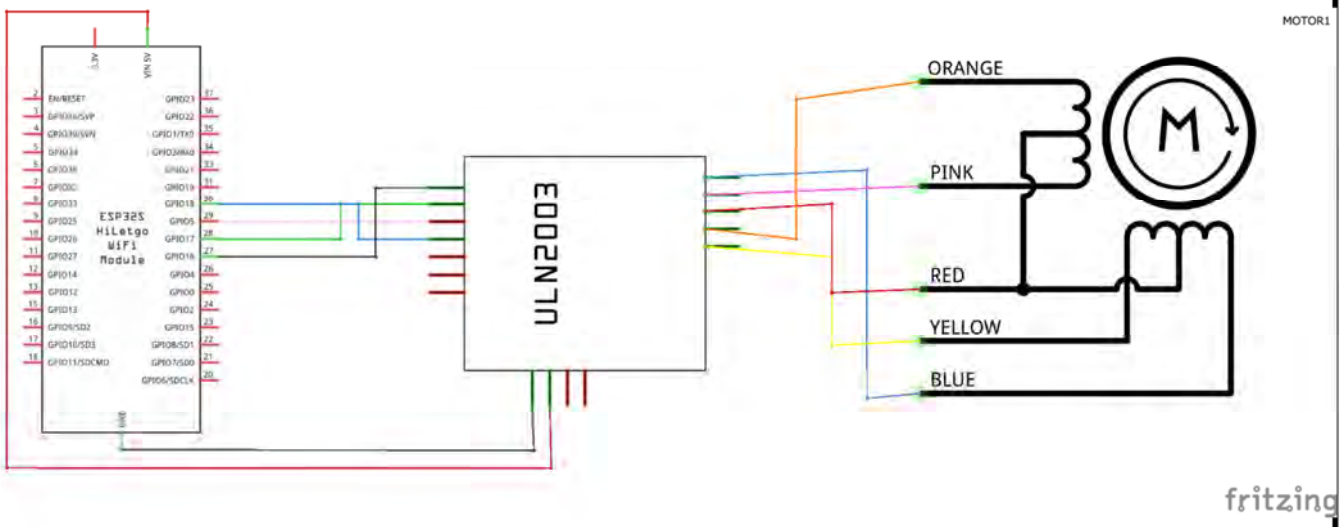


Рисунок 3.4 – Електрична принципова схема виконавчого механізму воріт

Наступним елементом є модуль RFID-зчитування. Його електрична схема зображена на рисунку 3.5.

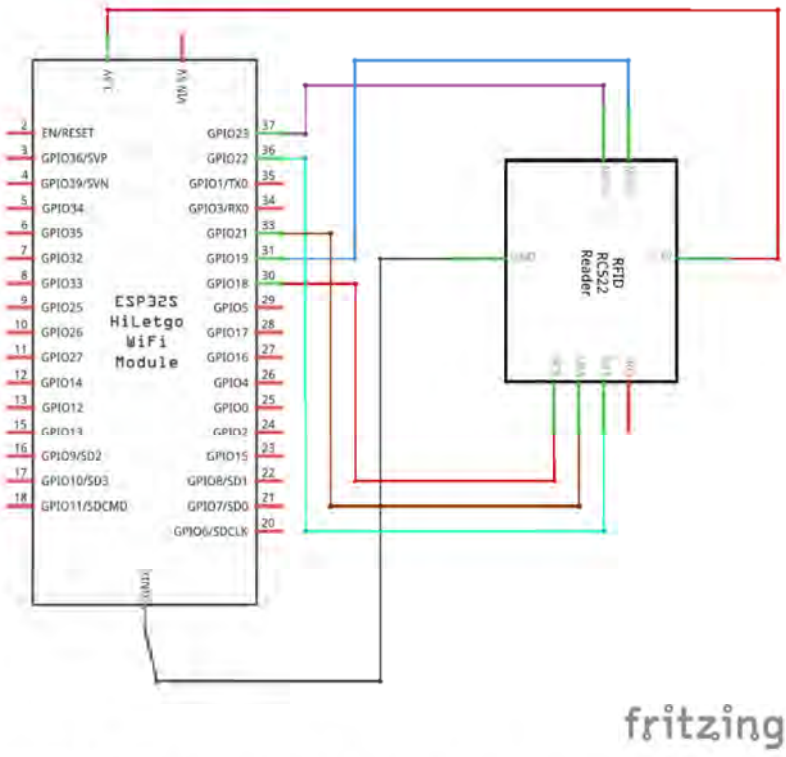


Рисунок 3.5 – Електрична принципова схема RFID-зчитувача

3.2 Програмна реалізація

Спочатку було прийнято рішення випробувати різні бібліотеки для оптичного розпізнавання символів, зокрема для розпізнавання тексту з фотографій. Серед протестованих рішень були PaddleOCR, EasyOCR та Tesseract. Обрано для проєкту PaddleOCR через його високу точність можливість роботи без обов'язкового використання GPU-ядер.

На наступному етапі постало питання: навчати власну нейромережу чи використати вже готову модель. Після тривалого аналізу даного питання було прийнято рішення використати попередньо навчену модель щоб зекономити час і обчислювальні ресурси. Серед доступних варіантів найбільш популярною виявилася модель haarcascade_russian_plate_number. Незважаючи на те, що вона навчена на російських номерних знаках, під час тестування вона демонструвала здатність виявляти номерні знаки різних країн.

У рамках реалізації системи було створено функціональний фрагмент програмного коду, що відповідає за виявлення номерного знака транспортного засобу в режимі реального часу. Цей компонент є частиною загального серверного застосунку й інтегрований у його життєвий цикл. Для захоплення відеопотоку використовується бібліотека OpenCV, яка ініціалізує підключення до камери та забезпечує постійний потік кадрів. Кожен отриманий кадр обробляється із застосуванням попередньо навченої моделі, яка використовується для детекції області, що потенційно містить номерну табличку. У разі успішного виявлення відповідна частина кадру обрізається, зберігається у файл та передається на подальше оптичне розпізнавання.

Наступним етапом реалізації системи став вибір технологічного стеку для побудови серверної частини системи. Розглядалися декілька популярних варіантів, зокрема розробка серверу на основі Node.js із фронтендом на React, або ж повноцінна реалізація усієї логіки на Python із використанням фреймворків, таких як Flask чи FastAPI. Перевагою першого

підходу була висока швидкодія та розділення логіки бекенду й клієнтської частини, проте він вимагав додаткових зусиль для забезпечення взаємодії з Python-бібліотеками, що використовуються для розпізнавання номерів.

Після аналізу варіантів було прийнято рішення реалізувати сервер на Python із використанням FastAPI, що дозволило поєднати засоби обробки зображень, роботу з OCR, базою даних та інтерфейс користувача в єдиному середовищі. FastAPI забезпечує зручний та швидкий механізм створення REST- та WebSocket-інтерфейсів.

Вебінтерфейс системи реалізовано з використанням звичайного HTML, стилізовано за допомогою CSS, а також додано трохи JavaScript для оновлення даних без перезавантаження сторінки. Основні сторінки, такі як головна, вхід, історія подій, керування базою даних користувачів і автомобілів. Створюються сервером за допомогою шаблонізатора Jinja2. Вся логіка обробки даних виконується на сервері, а браузер лише відображає готові сторінки. Для передачі розпізнаного номерного знака та стану воріт у реальному часі використано WebSocket-з'єднання, яке дозволяє отримувати дані з сервера без затримок і автоматично оновлювати інформацію на сторінці.

Наступним постало питання щодо організації зберігання даних у системі. Розглядалося два основні варіанти: використання повноцінного серверного середовища для баз даних, такого як MySQL Server, або ж зберігання інформації локально у вигляді файлу за допомогою вбудованої SQLite. Перший варіант дає більше можливостей для масштабування та одночасної роботи декількох клієнтів, проте потребує додаткового налаштування сервера, драйверів, мережевого з'єднання та адміністрування. Для цілей даного проєкту, де система розгортається на одному пристрої і не потребує складної мережевої архітектури, було вирішено використовувати SQLite.

Оскільки спочатку здавалося, що такий підхід дозволить забезпечити більш захищений доступ до бази даних, не підключаючи її до зовнішнього

серверного середовища, передбачалося, що база, збережена у вигляді локального файлу, буде недоступна ззовні, а функції роботи з нею будуть захищені всередині коду. Однак, у процесі реалізації стало зрозуміло, що код на Python є відкритим, оскільки мова не компілюється у вигляді бінарного виконуваного файлу. Це означає, що будь-хто, хто має доступ до файлів проєкту, може переглянути логіку функцій та структуру взаємодії з базою даних. Незважаючи на це, обраний підхід залишається зручним і достатньо безпечним у рамках локального розгортання на одному пристрої, де фізичний доступ до коду обмежено.

Для повноцінної взаємодії із пристроєм ESP32 у серверну частину було додано функції для прослуховування MQTT-топіків, а також для публікації повідомлень у відповідні канали. Це дозволяє системі в режимі реального часу реагувати на події та надсилати команди пристрою. Таким чином, після реалізації механізму розпізнавання зображення, перевірки даних у базі, збереження історії подій, формування вебінтерфейсу та інтеграції з MQTT-брокером, серверна частина проєкту вважається повністю завершеною та готовою до роботи в реальному середовищі. Програмний код наведено в додатках Б, В та Г.

Черговим етапом реалізації стало створення механізму віддаленого керування кроковим двигуном за допомогою мікроконтролера ESP32. Для цього використовувався вже налаштований MQTT-брокер, через який сервер надсилає відповідні команди у вигляді повідомлень у топік, на який підписаний ESP32.

Спочатку планувалося використати підхід, при якому ESP32 перевіряє наявність зв'язку із сервером через регулярні HTTP-запити. Проте в рамках цього проєкту взаємодія мікроконтролера з сервером відбувається лише через API та MQTT, тобто ініціатива надсилання команд належить саме серверу, а не ESP32.

У зв'язку з цим основна частина коду була запозичена з реалізації налаштування мережевого з'єднання мікроконтролера через вебінтерфейс

У цьому інтерфейсі користувач може ввести необхідні параметри підключення до Wi-Fi. Дані зберігаються у вбудованій пам'яті мікроконтролера, що дозволяє уникнути повторного введення при кожному перезавантаженні. Після збереження налаштувань пристрій автоматично підключається до мережі та встановлює MQTT-з'єднання для подальшої роботи. Приклад коду наведений в додатку Ґ.

Далі реалізовано RFID-зчитувач на базі ESP32, який реалізує альтернативний канал ідентифікації. Цей мікроконтролер працює автономно та не взаємодіє безпосередньо з камерою чи кроковим двигуном. Його єдиною функцією є зчитування унікального ідентифікатора з RFID-картки або брелока, який прикладає користувач. Після зчитування UID передається через MQTT. Приклад коду наведений в додатку Д.

На цьому етапі програмна реалізація системи була завершена, тож далі перейдемо до етапу тестування.

3.3 Тестування системи

Було створено чек-лист функціональних перевірок, який включає ключові компоненти інтегрованого рішення: від зв'язку ESP32 з мережею Wi-Fi до стабільності роботи OCR-модуля при різних умовах освітлення. Кожен пункт чек-листа відповідає окремому функціональному блоку системи, який було протестовано при симуляції типових умов експлуатації.

Тестування проводилося за наступною методикою:

– Кожен компонент системи активувався вручну або автоматично, відповідно до заданих сценаріїв.

– Для перевірки MQTT-зв'язку та реакції двигуна використовувалися програмно сформовані команди через локальний брокер Mosquitto.

– Розпізнавання номерних знаків тестувалося на попередньо підготовленому наборі зображень, які включали як ідеальні умови (чітке фронтальне зображення), так і складні – з низьким рівнем освітлення, тінями, відблисками, нахилом тощо.

Перевірка безпеки включала спробу доступу до серверного коду без авторизації, а також аналіз структури директорій на предмет відкритих шляхів.

На основі зібраних даних і спостережень було створено зведену таблицю (табл. 3.5), що дозволяє наочно представити функціональний стан кожної частини системи.

Таблиця 3.5 – Чек-лист функціонального тестування системи

№	Функціонал	Очікувана поведінка	Результат
1	Встановлення з'єднання ESP32 з Wi-Fi	Після введення даних через вебінтерфейс автоматичне підключення	+
2	Прослуховування MQTT-топіка	ESP32 отримує команду open, close, stop, а сервер UID RFID-міток	+
3	Реакція двигуна на команди	Двигун обертається/зупиняється відповідно до команди	+
4	Розпізнавання номерного знака	Навчена модель розпізнає номерні знаки а PaddleOCR «витягує» текст із зображення	+
5	Автоматичне збереження подій у базі даних	Дані про доступ зберігаються у таблиці event_log	+
6	Відновлення після перезавантаження ESP32	Після перезапуску зберігаються налаштування Wi-Fi і відбувається підключення до MQTT	+
7	Захист сервера	Вихідний код і логіка обробки недоступні стороннім користувачам без авторизації	-

Усі функції системи продемонстрували стабільну роботу в тестовому середовищі. Затримка між надсиланням команди із сервера та виконанням дії ESP32 становила в середньому менше 1 секунди. Система впевнено працює в автономному режимі, не потребує повторної конфігурації після перезавантаження. Серед виявлених недоліків системи варто відмітити, що PaddleOCR може розпізнавати номерні знаки некоректно при поганому освітленні або нахилі зображення. Однак, цей недолік був зафіксований як потенційний напрям для подальшого вдосконалення системи.

4. ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Все більше організацій прагнуть автоматизувати свою діяльність, щоб підвищити ефективність роботи, зменшити витрати та забезпечити безперебійне функціонування ключових процесів. Одним із таких напрямків є автоматизація системи контролю доступу.

За своєю суттю система є вдосконаленням існуючих рішень. Вона не є абсолютно новою, але відрізняється від аналогів доступністю, відкритістю та економічністю. Більшість подібних продуктів на ринку є закритими комерційними рішеннями з високою ціною, тоді як дана система може бути реалізована значно дешевше без втрати основного функціоналу.

Основними потенційними замовниками такої системи є підприємства, склади, паркінги, житлові комплекси, автостоянки та інші об'єкти з обмеженим доступом, де важлива як безпека, так і швидкість обслуговування. Ринок збуту охоплює як приватний, так і корпоративний сегмент, а зростання інтересу до цифрової безпеки створює сприятливі умови для розповсюдження подібних продуктів. Оскільки система є бюджетною, вона орієнтована на малий та середній бізнес, а також на установи, які шукають ефективне, але недороге рішення для організації доступу.

Попри загальну позитивну динаміку в розвитку автоматизованих систем, ситуація в Україні зазнала змін унаслідок повномасштабного вторгнення. За статистикою з 2019 по 2023 рік ринок автоматики для воріт та систем контролю доступу скоротився на 24,54%. Найбільше падіння зафіксовано в період 2022-2023 років, що пояснюється економічною нестабільністю, зниженням інвестиційної активності та пошкодженням інфраструктури в зоні бойових дій. Водночас у західних регіонах країни, а також у містах з високим рівнем внутрішньої міграції, спостерігається

зростання інтересу до встановлення бюджетних систем контролю доступу, зокрема в житлових комплексах, логістичних центрах і на складах.

На українському ринку представлені як вітчизняні, так і зарубіжні виробники рішень для автоматизації доступу. Серед основних конкурентів можна виокремити компанії Rital, CAME, Nice, BFT, AnMotors, які пропонують комплексні рішення з високим рівнем надійності, але й відповідною вартістю. Більшість таких систем є закритими, з обмеженими можливостями кастомізації, що ускладнює їх інтеграцію в унікальні інфраструктурні умови замовника.

На відміну від них запропонована система є відкритою для модифікацій, ґрунтується на доступних програмних та апаратних компонентах, а її ціна у базовій конфігурації може бути в декілька разів нижчою за імпортні аналоги. За умови подальшої оптимізації виробництва та грамотного просування, така система має перспективи зайняти нішу бюджетних рішень для малого та середнього бізнесу. Додатковим конкурентним фактором є можливість локального обслуговування та технічної підтримки, що знижує витрати кінцевого користувача та забезпечує оперативне реагування на зміни в умовах експлуатації.

4.2 Розрахункова частина

Техніко-економічна ефективність створеної системи контролю доступу визначається через аналіз витрат на її реалізацію та оцінку вигод, які вона приносить кінцевому користувачу. У контексті побутового або малопромислового застосування головна мета полягає не стільки в прямій економії коштів, скільки у підвищенні зручності, надійності, безпеки та автоматизації повсякденних рутинних дій. Водночас проєкт, навіть у спрощеній тестовій конфігурації, дозволяє досягнути відчутної оптимізації часу та ресурсів користувача.

Проте навіть за умов, коли економічна вигода не є ключовим критерієм, детальне техніко-економічне обґрунтування дає змогу об'єктивно оцінити доцільність впровадження розробки, виявити основні статті витрат та спрогнозувати потенційну окупність системи. Такий підхід є особливо корисним для обґрунтування впровадження аналогічного рішення на підприємствах або в комерційних структурах, де необхідно обґрунтувати інвестиції в автоматизацію.

Щоб повною мірою оцінити вартість створення системи, слід врахувати не лише витрати на придбання технічних компонентів, але й витрати на працю розробників, використання програмного забезпечення, витратні матеріали, логістичні та супутні ресурси. Нижче наведено обрахунки витрат на заробітну плату фахівців (табл. 4.1).

Таблиця 4.1 – Витрати на працю

Назва посади	Кількість годин роботи	Вартість роботи, грн	ЄСВ(22%)	Сума, грн
Проєктувальник системи	90	100	1980	10980
Програміст	150	80	2640	14640
Інженер з підтримки програмного забезпечення	120	80	2112	11712

На основі розрахунку витрат на працю можна оцінити загальні трудозатрати, необхідні для створення системи. Однак, важливо враховувати, що значну частину загального бюджету складають матеріальні витрати пов'язані з придбанням апаратних компонентів та програмного забезпечення. Ці витрати безпосередньо залежать від обраної архітектури, типу контролерів, сенсорів, модулів зв'язку та програмних рішень, які будуть інтегровані в систему.

Крім того, при виборі компонентів враховувалася доступність на ринку України, вартість доставки, можливість заміни у разі виходу з ладу, а також технічна сумісність між окремими модулями. Усі ці аспекти було

проаналізовано з метою мінімізації витрат без шкоди для функціональності та надійності системи. Нижче наведено таблицю 4.2, яка охоплює витрати на обладнання та програмне забезпечення, необхідні для реалізації проєкту.

Таблиця 4.2 – Витрати на компоненти та програмне забезпечення

Назва	Кількість	Вартість за одиницю, грн	Сума, грн
ESP32	2	200	400
Камера	1	1100	1100
28BYJ-48	1	100	100
MFRC522	1	60	60
Pycharm (1 місяць)	1	830	830

Окрім основних апаратних компонентів і програмного забезпечення, у процесі реалізації проєкту виникає низка супутніх витрат. Вони включають витрати на електроенергію під час тестування, витратні матеріали, витрати на оформлення документації, логістичні витрати, а також можливі витрати на сервісне обслуговування або навчання персоналу.

Такі витрати хоч і мають меншу частку в загальному бюджеті, є важливими для забезпечення повноцінного функціонування системи, особливо на етапах монтажу, введення в експлуатацію та супроводу. У таблиці нижче наведено перелік цих витрат із відповідними сумами.

Таблиця 4.3 – Інші витрати

Назва	Сума, грн
Електроенергія (160 год × 0.15 кВт × 6 грн)	144
Канцелярія	450
Інтернет	300
Транспортні витрати	300

Ці таблиці дозволяють комплексно оцінити фінансове навантаження, пов'язане з реалізацією системи. Вони показую, що проєкт має досить помірну вартість і може бути реалізований навіть у межах обмеженого бюджету, що робить його доступним для приватних користувачів. Велика частка витрат припадає на оплату праці, однак завдяки використанню доступних апаратних рішень та безкоштовного програмного забезпечення більшість витрат вдалося оптимізувати.

Оскільки система встановлюється у побутовому середовищі, її ефективність варто оцінювати не через економію зарплат працівників, а через виграний час, комфорт і безпеку для користувача. Уявімо, що власнику потрібно вручну відкривати і закривати ворота щоразу при виїзді та в'їзді. На цю дію щодня йде приблизно 4 хвилини (2 вранці та 2 ввечері). Упродовж року, за умови щоденного користування, це понад 20 годин часу, який можна заощадити. Наступна таблиця допомагає визначити, скільки коштує цей виграний час, якщо умовно оцінити його як ресурс.

Таблиця 4.4 – Оцінка часу і економії

Показник	Значення
Час, що економиться щодня	4 хв
Кількість днів на рік	300
Загальний зекономлений час	1200 хв
Умовна оцінка години часу	100 грн
Економія за рік	2 000 грн

Завдяки цій таблиці можна побачити, що навіть без прямого фінансового прибутку, система дозволяє заощаджувати час, який має значну вартість. Ба більше, ця вигода доповнюється комфортом: система працює повністю автономно і не вимагає виходити з автомобіля у негоду, вночі або зупинятися перед воротами. Це не лише зручно, а й підвищує безпеку, бо дозволяє мінімізувати людський фактор.

Оскільки цей проєкт є лише моделлю автоматичних воріт, вирахувати окупність кінцевого продукту в повному обсязі неможливо. Реалізація в реальних умовах потребує дещо іншої апаратної бази. Запчастини та програмне забезпечення будуть відрізнятися в залежності від вимог замовника або об'єкта. Усі розрахунки в цьому проєкті виконані на основі тестової конфігурації, яка була зібрана власноруч у навчальних цілях.

4.3 Обґрунтування необхідності розробки

Зростання потреб у підвищенні безпеки доступу до територій обмеженого користування, а також тенденції до автоматизації побутових і промислових процесів зумовлюють актуальність створення систем автоматичного контролю доступу. У сучасних умовах дедалі частіше виникає потреба в організації автономного контролю в'їзду автотранспорту без участі людини – як у приватному секторі, так і на території підприємств, закритих житлових дворів, складських зон тощо.

На практиці на таких об'єктах відсутня постійна охорона, або ж її утримання є економічно невигідним. Крім того, ручне керування воротами або шлагбаумами є незручним, особливо у несприятливих погодних умовах або в нічний час. Тому автоматизована система, здатна самостійно ідентифікувати транспортні засоби забезпечує суттєве покращення як у плані зручності, так і безпеки.

Проєкт було реалізовано в навчальних цілях в рамках дипломного проєктування. Основною метою була не комерційна реалізація, а демонстрація на практиці принципів побудови подібних систем, інтеграції апаратного та програмного забезпечення, а також відпрацювання навичок, пов'язаних з IoT-рішеннями, вебтехнологіями, комп'ютерним зором та базами даних. Усі розрахунки базуються на умовній моделі, яка використовується для тестування і демонстрації функціоналу системи.

Попри це, розроблена система має достатній потенціал, щоб бути доопрацьованою та представлена як готовий продукт на ринку. Вона також

має переваги в експлуатації: мінімальні витрати на обслуговування, автономна робота, просте налаштування та можливість адаптації до різних типів об’єктів. У порівнянні з аналогічними комерційними рішеннями, запропонована розробка є бюджетною, що робить її доступною для широкого кола користувачів.

Таким чином, система задовольняє потреби в безпечному, зручному та ефективному контролі доступу на території, де відсутня постійна охорона. Вона дозволяє економити ресурси, покращує безпеку об'єкта та сприяє підвищенню загальної ефективності функціонування інфраструктури. Очікуваний ефект від впровадження – це економія часу користувача, відмова від послуг охорони, зменшення кількості інцидентів, пов’язаних із несанкціонованим проникненням, та підвищення комфортності повсякденного життя.

ВИСНОВКИ

У межах даної кваліфікаційної роботи було виконано повноцінне проєктування, реалізацію та тестування автоматичної системи контролю доступу автотранспорту на закриту територію. Реалізація такого проєкту стала відповіддю на зростаючі вимоги до забезпечення безпеки, ефективності та автоматизації процесів, пов'язаних із контролем в'їзду та виїзду транспортних засобів на закриту територію. Актуальність обраної теми зумовлена потребою у створенні доступних та надійних рішень для приватного та корпоративного сектору, де відсутність постійної охорони або бажання мінімізувати людський фактор спонукає до впровадження інтелектуальних систем.

У ході дослідження було виконано ґрунтовний аналіз предметної області, розглянуто наявні на ринку рішення, зокрема ті, що ґрунтуються на RFID-технологіях, біометричних ідентифікаторах, а також системах розпізнавання номерних знаків. Особливу увагу було приділено архітектурі та принципам функціонування таких систем. В результаті обґрунтовано доцільність поєднання методів комп'ютерного зору та RFID-ідентифікації, що дозволило створити більш гнучкий і надійний підхід до контролю доступу.

Розроблена система має модульну структуру, що поєднує апаратні та програмні компоненти. Було успішно реалізовано передачу даних між окремими частинами системи за допомогою протоколу MQTT, що забезпечує стабільну комунікацію між сервером, ESP32 та виконавчими механізмами. Камера відеоспостереження, з'єднана із серверною частиною, дозволяє виконувати автоматичне розпізнавання номерних знаків у режимі реального часу. У випадках, коли розпізнавання ускладнене, система дозволяє використовувати RFID-мітки або ручне підтвердження через Вебінтерфейс.

Окрему увагу було приділено розробці інтерфейсу користувача, що забезпечує просте та зручне адміністрування системи, перегляд історії подій та керування базами даних транспортних засобів і користувачів. Структура бази даних спроєктована таким чином, щоб забезпечити цілісність інформації, можливість розширення та аналітики. Серверна частина створена на основі сучасного вебфреймворку FastAPI, що дало змогу об'єднати обробку зображень, роботу з базами даних, фронтендом та MQTT-з'єднанням в одному середовищі.

Після завершення програмної та апаратної реалізації було проведено комплексне тестування системи у реальних умовах. Результати перевірок засвідчили стабільну роботу всіх функціональних модулів, відповідність проєкту технічному завданню, а також можливість подальшого масштабування або інтеграції з додатковими рішеннями безпеки. Незначні недоліки, виявлені під час тестування, не впливають критично на загальну функціональність системи, однак окреслюють потенційні напрямки для вдосконалення.

З техніко-економічної точки зору розробка показала себе як доступне та раціональне рішення для впровадження в умовах обмеженого бюджету. Використання доступних компонентів і відкритих програмних засобів забезпечило значну економію коштів, зберігаючи при цьому високу ефективність. Розрахунки продемонстрували доцільність впровадження подібних систем навіть у побутових умовах, адже окрім підвищеного рівня безпеки, користувач отримує економію часу, підвищення комфорту та зменшення потреби у фізичному втручанні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Крокoвий двигун 28BYJ-48 с драйвером ULN2003. Інтернет-магазин радіоаматорських товарів radiostore.ua | Паяльні станції, джерела живлення, витратні матеріали, радіокомпоненти. URL: https://radiostore.ua/products/shagovyy-dvigatel-28byj-48-s-drayverom-uln2003-2 (дата звернення: 08.05.2025).

2. HID Global completa los primeros pilotos con tecnología NFC en smartphones para apertura de puertas. IDentico SAS. URL: https://identicosas.blogspot.com/2012/10/hid-global-completa-los-primeros.html (дата звернення: 14.03.2025).

3. RFID-модуль RC522. Огляд, застосування, характеристики. I Блог 3DIY shop. 3DiY shop – Інтернет-магазин комплектующих для 3D принтера, Arduino, ЧПУ і робототехніка | 3DIY. URL: https://salo.li/5191F0d (дата звернення: 09.05.2025).

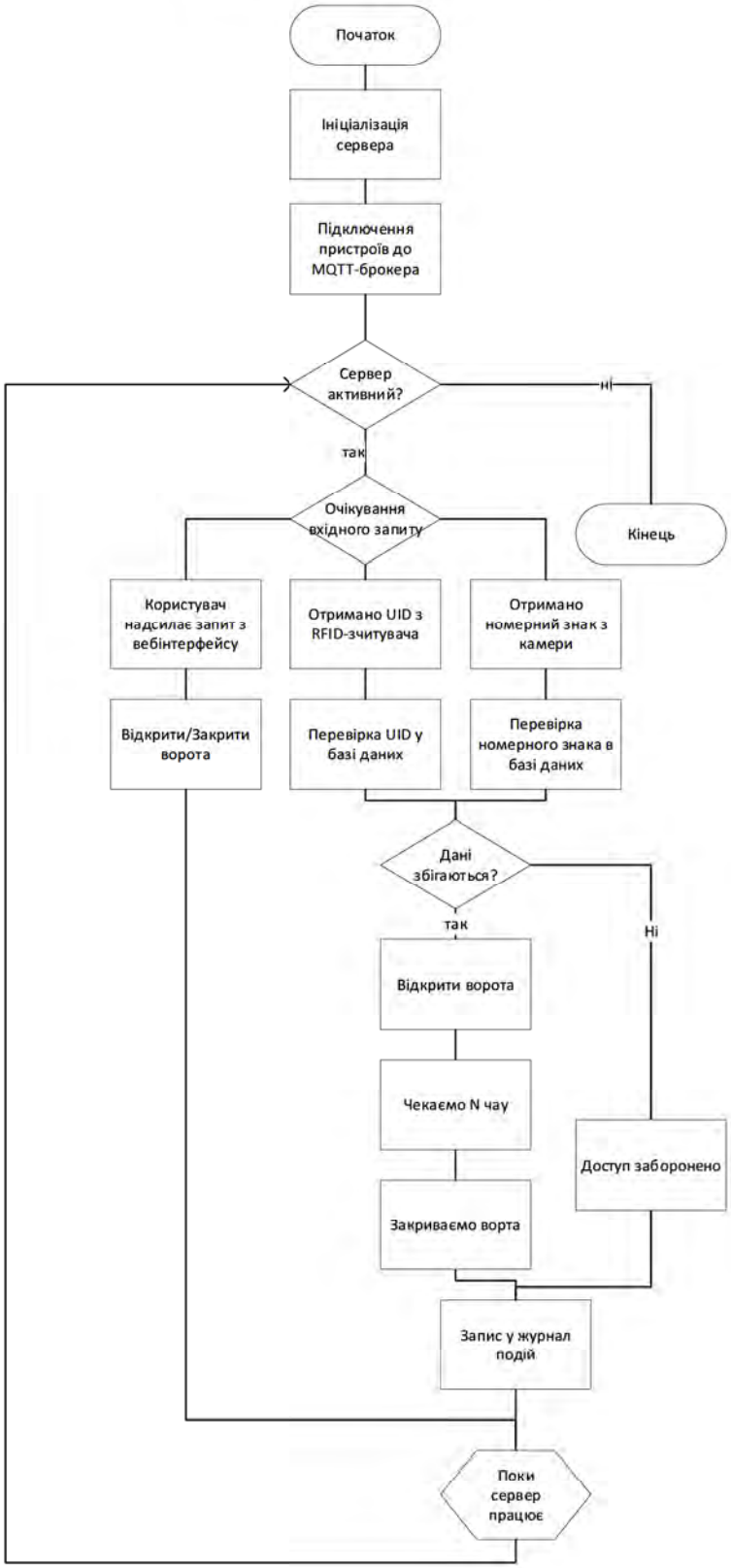
4. Semi-active RFID solutions – Nedap Identification Systems. Nedap Identification Systems. URL: https://www.nedapidentification.com/technologies/semi-active-rfid-solutions (дата звернення: 08.06.2025).

5. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 123 «Ком п'ютерна інженерія» освітньої програми «Інженерія Інтернету речей» URL: https://moodle.gi.edu.ua/mod/resource/view.php?id=83881

ДОДАТКИ

Додаток А

Блок схема системи



Додаток Б

Лістинг програмного коду сервера головного файлу

```
import asyncio
import cv2
from fastapi import FastAPI, WebSocket, Request, Form,
HTTPException
from starlette.middleware.sessions import SessionMiddleware
from fastapi.responses import StreamingResponse, HTMLResponse,
RedirectResponse
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
from contextlib import asynccontextmanager
from paddleocr import PaddleOCR
import SQLLITE
import os
from MQTT import start as start_mqtt , publish
import concurrent.futures
import  io
import time

GATE_TOPIC = "esp/gate/control"
executor =
concurrent.futures.ThreadPoolExecutor(max_workers=1)

cap = None
recognized_plate = "Очікування..."
ocr = PaddleOCR(use_angle_cls=True, lang="en")
carplate_haar_cascade =
cv2.CascadeClassifier('haarcascade_russian_plate_number.xml')

def read_image_as_blob(file_path):
    with open(file_path, 'rb') as f:
        blob_data = f.read()
    return blob_data
```

```
@asynccontextmanager
async def lifespan(app: FastAPI):
    global cap
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Помилка: Не вдалося відкрити камеру")
    start_mqtt()
    asyncio.create_task(process_video())
    yield
    if cap:
        cap.release()


app = FastAPI(lifespan=lifespan)
app.mount("/static", StaticFiles(directory="static"),
name="static")
app.add_middleware(SessionMiddleware,
secret_key="supersecretkey")
templates = Jinja2Templates(directory="templates")

def carplate_extract(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    carplate_rects =
carplate_haar_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5)

    if len(carplate_rects) == 0:
        return None

    for x, y, w, h in carplate_rects:
        carplate_img = image[y + 15:y + h - 10, x + 15:x + w -
20]

        if carplate_img.size == 0:
            print("⚠ Вирізаний номерний знак пустий!")
```

```
            return None

        filename = "carplate.png"
        cv2.imwrite(filename, carplate_img)
        print(f"📸 Збережено: {filename}")
        print(f"Збережено у: {os.path.abspath(filename)}")
        return filename

    return None


def recognize_text(image_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = ocr.ocr(image, cls=True)
    if not results:
        return None
    text_list = []
    for line in results:
        if line:
            for word_info in line:
                text, _ = word_info[1]
                text_list.append(text)
                print(text)
    return "".join(text_list) if text_list else None

async def process_video():
    global recognized_plate
    while True:
        if cap and cap.isOpened():
            ret, frame = cap.read()
            if ret:
                filename = carplate_extract(frame)
                if filename:
```

```
                    loop = asyncio.get_event_loop()
                    recognized_plate = await
loop.run_in_executor(
                        executor,
                        recognize_text,
                        filename
                    )
                    if recognized_plate:
                        match =
SQLLITE.check_plate_in_db(recognized_plate)
                        if match:
                            print("Номер виявлено в базі")
                            publish("esp/gate/control","open")
                            plate_img_blob =
read_image_as_blob("carplate.png")

SQLLITE.add_event(recognized_plate,
plate_image=plate_img_blob, result='access granted',
users_id=SQLLITE.get_user_id_by_plate(recognized_plate))

                            await asyncio.sleep(15)
                            publish("esp/gate/control",
"close")
                            # Таймер і надсилання топіку на
закритя?
                        else:
                            print("Номер не в базі")
                            plate_img_blob =
read_image_as_blob("carplate.png")

SQLLITE.add_event(recognized_plate,plate_image=plate_img_blob)
                            # Якась інтеграція з користувачем
НЕЗНАЮ треба виводити сповіщення де і як?
        await asyncio.sleep(2)

@app.websocket("/ws")
```

```
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    try:
        current_gate_status=SQLLITE.gate_shec()
        while True:
            await websocket.send_json({
                "plate": recognized_plate,
                "gateStatus": current_gate_status
            })
            await asyncio.sleep(1)
    except Exception as e:
        print(f"WebSocket error: {e}")

def generate_frames():
    while True:
        if cap is None or not cap.isOpened():
            break
        ret, frame = cap.read()
        if not ret:
            continue
        _, buffer = cv2.imencode('.jpg', frame)
        frame_bytes = buffer.tobytes()
        yield (b'--frame\r\nContent-Type: image/jpeg\r\n\r\n'
+ frame_bytes + b'\r\n')

@app.post("/gate/open")
async def open_gate():
    publish(GATE_TOPIC, "open")
    return {"status": "success"}

@app.post("/gate/close")
async def close_gate():
    publish(GATE_TOPIC, "close")
    return {"status": "success"}

@app.post("/gate/stop")
```

```
async def stop_gate():
    publish(GATE_TOPIC, "stop")
    return {"status": "success"}

@app.get("/video")
def video_feed():
    return StreamingResponse(generate_frames(),
media_type="multipart/x-mixed-replace; boundary=frame")

@app.get("/", response_class=HTMLResponse)
async def home(request: Request):
    user = request.session.get("user")
    if not user:
        return RedirectResponse("/login")
    return templates.TemplateResponse("index.html",
{"request": request, "user": user})

@app.get("/login", response_class=HTMLResponse)
async def login_get(request: Request):
    return templates.TemplateResponse("login.html",
{"request": request})


@app.post("/login")
async def login_post(request: Request, login: str = Form(...),
password: str = Form(...)):
    if SQLLITE.check_credentials(login, password):
        request.session["user"] = True
        return RedirectResponse(url="/", status_code=302)
    else:
        return templates.TemplateResponse("login.html",
{"request": request, "error": "Невірний логін або пароль"})

@app.get("/history", response_class=HTMLResponse)
async def history(request: Request):
    user = request.session.get("user")
```

```
    if not user:
        return RedirectResponse("/login")

    events =SQLLITE.get_event_history()
    return templates.TemplateResponse("history.html", {
        "request": request,
        "user": user,
        "events": events
    })

@app.get("/users", response_class=HTMLResponse)
async def edit_db(request: Request):
    user = request.session.get("user")
    if not user:
        return RedirectResponse("/login")
    table_data = SQLLITE.print_table('users')
    return templates.TemplateResponse("users.html",
{"request": request, "user": user,  "columns":
table_data["columns"],
        "rows": table_data["rows"]})

@app.get("/car", response_class=HTMLResponse)
async def edit_db(request: Request):
    user = request.session.get("user")
    if not user:
        return RedirectResponse("/login")
    table_data = SQLLITE.print_table('Car')
    return templates.TemplateResponse("car.html", {"request":
request, "user": user,  "columns": table_data["columns"],
        "rows": table_data["rows"]})

@app.get("/config", response_class=HTMLResponse)
async def edit_db(request: Request):
    user = request.session.get("user")
    if not user:
        return RedirectResponse("/login")
```

```
    table_data = SQLLITE.print_table('Config')
    return templates.TemplateResponse("config.html",
{"request": request, "user": user,  "columns":
table_data["columns"],
        "rows": table_data["rows"]})

@app.get("/plate_image/{event_id}")
async def get_plate_image(event_id: int):
    try:
        image_data = SQLLITE.img_re(event_id)
        if image_data is None:
            raise HTTPException(status_code=404)
        if isinstance(image_data, str):
            return
StreamingResponse(io.BytesIO(image_data.encode('latin1')),
media_type="image/png")
        if isinstance(image_data, bytes):
            return StreamingResponse(io.BytesIO(image_data),
media_type="image/png")
        raise HTTPException(status_code=404)
    except Exception as e:
        print(f"Помилка отримання зображення: {e}")
        raise HTTPException(status_code=500)

@app.get("/rfid", response_class=HTMLResponse)
async def edit_db(request: Request):
    user = request.session.get("user")
    if not user:
        return RedirectResponse("/login")
    table_data = SQLLITE.print_table('RFID')
    return templates.TemplateResponse("rfid.html", {"request":
request, "user": user,  "columns": table_data["columns"],
        "rows": table_data["rows"]})

@app.get("/edit/{table_name}/{record_id}",
response_class=HTMLResponse)
```

```
async def edit_record_form(request: Request, table_name: str,
record_id: str):
    user = request.session.get("user")
    if not user:
        return RedirectResponse("/login")
    record = SQLLITE.get_record_by_id(table_name, record_id)
    if not record:
        raise HTTPException(status_code=404, detail="Record
not found")
    table_data = SQLLITE.print_table(table_name)
    return templates.TemplateResponse("edit_form.html", {
        "request": request,
        "user": user,
        "table_name": table_name,
        "record_id": record_id,
        "record": record,
        "columns": table_data["columns"]
    })

@app.post("/edit/{table_name}/{record_id}")
async def update_record(request: Request, table_name: str,
record_id: str):
    user = request.session.get("user")
    if not user:
        return RedirectResponse("/login")
    form_data = await request.form()
    data_to_update = {k: v for k, v in form_data.items() if k
!= "table_name" and k != "record_id"}
    success = SQLLITE.update_record(table_name, record_id,
data_to_update)
    if not success:
        raise HTTPException(status_code=400, detail="Failed to
update record")
    return RedirectResponse(f"/{table_name}", status_code=303)

@app.get("/add/{table_name}", response_class=HTMLResponse)
```

```
async def add_record_form(request: Request, table_name: str):
    form =f"add_form_{table_name}.html"
    user = request.session.get("user")
    if not user:
        return RedirectResponse("/login")
    users= []
    db = SQLLITE.sqlite3.connect("server.db")
    cursor = db.cursor()
    cursor.execute("SELECT id, first_name, last_name FROM
users")
    users = [{"id": row[0], "first_name": row[1], "last_name":
row[2]} for row in cursor.fetchall()]
    cursor.execute(f"PRAGMA table_info({table_name})")
    columns_info = cursor.fetchall()
    db.close()
    columns = [col[1] for col in columns_info]
    primary_key = None
    for col in columns_info:
        if col[5] == 1:
            primary_key = col[1]
            break
    return templates.TemplateResponse(form, {
        "request": request,
        "user": user,
        "table_name": table_name,
        "columns": columns,
        "primary_key": primary_key,
        "users":users
    })

@app.post("/add/{table_name}")
async def insert_record_post(request: Request, table_name:
str):
    user = request.session.get("user")
    if not user:
        return RedirectResponse("/login")
```

```
    form_data = await request.form()
    data = dict(form_data)
    SQLLITE.insert_record(table_name, data)
    return RedirectResponse(f"/{table_name}", status_code=303)

@app.get("/delete/{table_name}/{item_id}")
def delete_item(table_name: str, item_id: str, request:
Request):
    if table_name == "Config" and
SQLLITE.is_config_protected(item_id):
        return RedirectResponse(url="/config",
status_code=303)
    SQLLITE.delet(table_name, item_id)
    return RedirectResponse(url=f"/{table_name.lower()}",
status_code=303)

@app.get("/logout")
async def logout(request: Request):
    request.session.clear()
    return RedirectResponse("/login")

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("server_pass:app", host="0.0.0.0", port=5000,
reload=True)
```

Додаток В

Лістинг програмного коду сервера SQLLITE файлу

```
import sqlite3
from datetime import datetime
def create_DB(c):
    c.executescript("""
CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    first_name VARCHAR(45),
    last_name VARCHAR(45),
    patronymic VARCHAR(45),
    phone VARCHAR(9),
    email VARCHAR(60)
);

CREATE TABLE Car (
    number_plate CHAR(8) PRIMARY KEY,
    Color VARCHAR(45),
    Model VARCHAR(45),
    Brand VARCHAR(20),
    users_id INTEGER,
    FOREIGN KEY (users_id) REFERENCES users(id)
);

CREATE TABLE RFID (
    ID_RFID CHAR(8) PRIMARY KEY,
    Date_of_issue VARCHAR(45),
    users_id INTEGER,
    FOREIGN KEY (users_id) REFERENCES users(id)
);

CREATE TABLE event_log (
    idEvent_log INTEGER PRIMARY KEY,
    date_and_time DATETIME,
    number_plate CHAR(8),
```

```
    rfid_id CHAR(8),
    result VARCHAR(45),
    users_id INTEGER,
    plate_image BLOB,
    FOREIGN KEY (number_plate) REFERENCES Car(number_plate),
    FOREIGN KEY (rfid_id) REFERENCES RFID(ID_RFID),
    FOREIGN KEY (users_id) REFERENCES users(id)
);

CREATE TABLE Config (
    idConfig INTEGER PRIMARY KEY,
    name_config VARCHAR(45),
    value VARCHAR(45)
);
""")


def print_table(table_name):
    db = sqlite3.connect("server.db")
    cursor = db.cursor()

    # Отримуємо дані з таблиці
    cursor.execute(f"SELECT * FROM {table_name}")
    rows = cursor.fetchall()

    # Отримуємо назви колонок
    col_names = [description[0] for description in
cursor.description]

    db.close()

    return {
        "columns": col_names,
        "rows": rows
    }
```

```
def get_event_history():
    db = sqlite3.connect("server.db")
    cursor = db.cursor()
    cursor.execute("""
        SELECT el.idEvent_log, el.date_and_time,
el.number_plate, el.plate_image, el.rfid_id, el.source,
el.result,
               u.first_name || ' ' || u.last_name AS full_name
        FROM event_log el
        LEFT JOIN users u ON el.users_id = u.id
        ORDER BY el.date_and_time DESC
    """)
    events = cursor.fetchall()
    db.close()
    return events

#update
def get_primary_key_column(cursor, table_name):
    cursor.execute(f"PRAGMA table_info({table_name})")
    columns_info = cursor.fetchall()
    for column in columns_info:
        if column[5] == 1:  # первинний ключ
            return column[1]
    return 'id'  # значення за замовчуванням, якщо не знайдено

def get_record_by_id(table_name, record_id):
    conn = sqlite3.connect("server.db")
    cursor = conn.cursor()

    pk_column = get_primary_key_column(cursor, table_name)

    cursor.execute(f"SELECT * FROM {table_name} WHERE
{pk_column} = ?", (record_id,))
    record = cursor.fetchone()

    if not record:
```

```
            conn.close()
            return None

        columns = [column[0] for column in cursor.description]
        conn.close()

        return dict(zip(columns, record))

    def update_record(table_name, record_id, data):
        conn = sqlite3.connect("server.db")
        cursor = conn.cursor()

        try:
            pk_column = get_primary_key_column(cursor, table_name)
            data.pop(pk_column, None)

            if not data:
                return False

            set_clause = ", ".join([f"{key} = ?" for key in
    data.keys()])
            query = f"UPDATE {table_name} SET {set_clause} WHERE
    {pk_column} = ?"

            cursor.execute(query, (*data.values(), record_id))
            conn.commit()
            return cursor.rowcount > 0
        except Exception as e:
            conn.rollback()
            raise e
        finally:
            conn.close()


    def insert_record(table_name, data):
        conn = sqlite3.connect("server.db")
```

```
    cursor = conn.cursor()

    try:

        cursor.execute(f"PRAGMA table_info({table_name})")
        columns_info = cursor.fetchall()

        table_columns = [col[1] for col in columns_info]

        filtered_data = {k: v for k, v in data.items() if k in
table_columns}

        if not filtered_data:
            raise ValueError("Жодне з переданих полів не
знайдено в таблиці.")

        columns_str = ", ".join(filtered_data.keys())
        placeholders = ", ".join(["?"] * len(filtered_data))

        query = f"INSERT INTO {table_name} ({columns_str})
VALUES ({placeholders})"
        cursor.execute(query, tuple(filtered_data.values()))
        conn.commit()

        return cursor.lastrowid
    except Exception as e:
        conn.rollback()
        raise e
    finally:
        conn.close()


def check_plate_in_db (plate_text):
    conn = sqlite3.connect("server.db")
    cursor = conn.cursor()
```

```
    cursor.execute("SELECT * FROM Car WHERE number_plate = ?",
(plate_text,))
    result = cursor.fetchone()
    conn.close()
    return result


PROTECTED_CONFIGS = ['gate_status']

def is_config_protected(item_id):
    conn = sqlite3.connect("server.db")
    cur = conn.cursor()
    cur.execute("SELECT name_config FROM Config WHERE idConfig
= ?", (item_id,))
    row = cur.fetchone()
    conn.close()
    return row and row[0] in PROTECTED_CONFIGS

def add_event(number_plate=None, rfid_id=None,
result='denied', users_id=None, plate_image=None):
    conn = sqlite3.connect("server.db")
    cursor = conn.cursor()
    if rfid_id:
        source = "RFID"
    elif number_plate:
        source = "Plate"
    else:
        source = "Site"
    data = {
        'date_and_time': datetime.now().strftime("%Y-%m-%d
%H:%M:%S"),
        'number_plate': number_plate,
        'rfid_id': rfid_id,
        'result': result,
        'users_id': users_id,
        'plate_image': plate_image,
```

```
            'source': source
        }

        try:
            columns = ", ".join(data.keys())
            placeholders = ", ".join(["?"] * len(data))
            query = f"INSERT INTO event_log ({columns}) VALUES
({placeholders})"
            cursor.execute(query, tuple(data.values()))
            conn.commit()
            print("Подію успішно збережено.")
        except Exception as e:
            conn.rollback()
            print("Помилка під час збереження події:", e)
        finally:
            conn.close()

    def delet(table_name: str, item_id: str):
        conn = sqlite3.connect("server.db")
        cur = conn.cursor()
        key_column = {
            "users": "id",
            "Car": "number_plate",
            "RFID": "ID_RFID",
            "event_log": "idEvent_log",
            "Config": "idConfig"
        }

        column = key_column.get(table_name)
        if not column:
            conn.close()
            return

        cur.execute(f"DELETE FROM {table_name} WHERE {column} =
?", (item_id,))
        conn.commit()
```

```
    conn.close()

def img_re(event_id):
    db = sqlite3.connect("server.db")
    cursor = db.cursor()
    cursor.execute("SELECT plate_image FROM event_log WHERE
idEvent_log = ?", (event_id,))
    image_data = cursor.fetchone()
    db.close()
    return image_data[0] if image_data else None

def get_user_id_by_plate(plate):
    conn = sqlite3.connect("server.db")
    cursor = conn.cursor()
    cursor.execute("SELECT users_id FROM Car WHERE
number_plate = ?", (plate,))
    result = cursor.fetchone()
    conn.close()
    return result[0] if result else None

def get_user_id_by_rfid(rfid_id):
    conn = sqlite3.connect("server.db")
    cursor = conn.cursor()
    cursor.execute("SELECT users_id FROM RFID WHERE ID_RFID =
?", (rfid_id,))
    result = cursor.fetchone()
    conn.close()
    return result[0] if result else None

def get_config():
    db = sqlite3.connect('server.db')
    cursor = db.cursor()
    cursor.execute("SELECT name_config, value FROM Config")
    rows = cursor.fetchall()
    db.close()
    return {name: value for name, value in rows}
```

```
def gate_shec():
    db = sqlite3.connect("server.db")
    cursor = db.cursor()
    cursor.execute("SELECT value FROM Config WHERE name_config
= 'gate_status'")
    gate_status = cursor.fetchone()[0]
    db.close()
    return gate_status

def check_credentials(login, password):
    conn = sqlite3.connect("server.db")
    cursor = conn.cursor()
    cursor.execute("SELECT value FROM Config WHERE name_config
= 'login'")
    db_login = cursor.fetchone()
    cursor.execute("SELECT value FROM Config WHERE name_config
= 'password'")
    db_password = cursor.fetchone()
    conn.close()
    if db_login and db_password:
        return login == db_login[0] and password ==
db_password[0]
    return False

def check_rfid(uid):
    conn = sqlite3.connect("server.db")  # заміни на назву
твоєї БД
    cursor = conn.cursor()
    cursor.execute(
        """SELECT rfid.ID_RFID,
                  users.first_name || ' ' || users.last_name
||
                  CASE WHEN users.patronymic IS NOT NULL THEN
' ' || users.patronymic ELSE '' END
            FROM rfid
```

```
            JOIN users ON rfid.users_id = users.id
            WHERE rfid.ID_RFID = ?""",
        (uid,))
    row = cursor.fetchone()
    if row:
        print(f"✅ Доступ дозволено: UID = {row[0]},
Користувач = {row[1]}")
        add_event( rfid_id="UID: {uid}", result='access
granted',users_id=get_user_id_by_rfid (uid))
        conn.close()
        return True
    else:
        print(f"❌ Доступ заборонено: UID {uid} не знайдено")
        add_event(rfid_id=f"UID: {uid}")
    conn.close()

def main():
    tables = ['users', 'Car', 'RFID', 'event_log', 'Config']
    db = sqlite3.connect('server.db')
    c = db.cursor()
    # for table in tables:
    #     print_table(c, table)
    db.commit()
    db.close()

if __name__ == "__main__":
    main()
```

Додатко Г

Лістинг прогамного коду сервера MQTT файлу

```
import paho.mqtt.client as mqtt
import SQLLITE
import threading
import time
last_uid = ""
last_time = 0


mqttBroker = "localhost"
client = mqtt.Client(protocol=mqtt.MQTTv311)


def on_connect(client, userdata, flags, rc):
    print("✅ MQTT connected:", rc)
    client.subscribe("esp/response")
    client.subscribe("rfid/UID")


def on_message(client, userdata, msg):
    global last_uid, last_time
    topic = msg.topic
    payload = msg.payload.decode("utf-8")
    now = time.time()

    if payload == last_uid and now - last_time < 5:
        print("⚠️ Повтор UID — пропущено")
        return

    last_uid = payload
    last_time = now

    print(f"[MQTT] {topic}: {payload}")
    if SQLLITE.check_rfid(payload):
```

```
            print("RFID знайдено, відкриваємо двері")
            client.publish("esp/gate/control", "open")
            threading.Timer(15.0, lambda:
client.publish("esp/gate/control", "close")).start()
        else:
            print("RFID не знайдено")

def publish(topic, payload):
    client.publish(topic, payload)

def config_rfid_rider(config_rfid):
    for name, value in config_rfid.items():
        publish("rfid-rider/config/" + name, value)


def start():
    config_rfid_rider(SQLLITE.get_config())
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(mqttBroker)
    client.loop_start()

def get_client():
    return client
```

Додато Ґ

Лістинг програмного коду приводу для воріт

```
#include <WiFi.h>
#include <Preferences.h>
#include <PubSubClient.h>
#include <WebServer.h>

Preferences preferences;

const char* default_ssid = "TP-Link_0B0B";
const char* default_password = "fnaf_009";
const char* default_mqtt_ip = "192.168.188.61";
const int mqtt_port = 1883;
const char* mqtt_topic_control = "esp/gate/control";
const char* mqtt_topic_status = "esp/gate/status";
const char* ap_password = "123456789";

WebServer server(80);
WiFiClient espClient;
PubSubClient client(espClient);

const int IN1 = 16, IN2 = 17, IN3 = 5, IN4 = 18;
const int stepSequence[8] = {B01000, B01100, B00100, B00110,
B00010, B00011, B00001, B01001};
int speedMotor = 1200, stepIndex = 0, stepsRemaining = 0,
currentPosition = 0;
const int fullTravel = 2048;
bool motorRunning = false;
bool apStarted = false;
bool webStarted = false;

enum State {STOPPED, OPENING, CLOSING, OPENED, CLOSED};
State currentState = STOPPED;

void handleMainMenu() {
```

```
  String html = "<html><head><meta charset='utf-
8'></head><body><h1>Налаштування ESP32</h1>"
                "<ul><li><a href=\"/wifi\">Wi-
Fi</a></li><li><a href=\"/server\">MQTT
IP</a></li></ul></body></html>";
  server.send(200, "text/html; charset=utf-8", html);
}

void handleWiFiForm() {
  String html = "<html><head><meta charset='utf-
8'></head><body><h1>Введіть дані Wi-Fi</h1>"
                "<form action=\"/save\" method=\"POST\">SSID:
<input name=\"ssid\"><br>"
                "Пароль: <input name=\"password\"
type=\"password\"><br>"
                "<input type=\"submit\"
value=\"Зберегти\"></form></body></html>";
  server.send(200, "text/html; charset=utf-8", html);
}

void handleSaveWiFi() {
  preferences.putString("ssid", server.arg("ssid"));
  preferences.putString("password", server.arg("password"));
  server.send(200, "text/html; charset=utf-8",
"<html><body><h1>Wi-Fi збережено!
Перезавантаження...</h1></body></html>");
  delay(1000);
  ESP.restart();
}

void handleCerver() {
  String html = "<html><head><meta charset='utf-
8'></head><body><h1>Введіть IP MQTT сервера</h1>"
                "<form action=\"/saveServer\"
method=\"POST\">IP сервера: <input name=\"IP_Server\"><br>"
```

```
                "<input type=\"submit\"
value=\"Зберегти\"></form></body></html>";
  server.send(200, "text/html; charset=utf-8", html);
}

void handleSave2() {
  preferences.putString("IP_Server", server.arg("IP_Server"));
  server.send(200, "text/html; charset=utf-8",
"<html><body><h1>IP збережено!
Перезавантаження...</h1></body></html>");
  delay(1000);
  ESP.restart();
}

void startWebInterface() {
  if (!webStarted) {
    server.on("/", handleMainMenu);
    server.on("/wifi", handleWiFiForm);
    server.on("/save", HTTP_POST, handleSaveWiFi);
    server.on("/server", handleCerver);
    server.on("/saveServer", HTTP_POST, handleSave2);
    server.begin();
    webStarted = true;
    Serial.println("Вебсервер запущено");
  }
}

void stopWebInterface() {
  if (webStarted) {
    server.close();
    webStarted = false;
    Serial.println(" Вебсервер вимкнено");
  }
}

void startAP() {
```

```
  if (!apStarted) {
    String chipID = String((uint32_t)ESP.getEfuseMac(), HEX);
    String apSSID = "ESP32_Config_" + chipID;
    WiFi.softAP(apSSID.c_str(), ap_password);
    Serial.println(" Запущено AP: " + apSSID);
    Serial.println("Доступ: http://" +
WiFi.softAPIP().toString());
    apStarted = true;
  }
}

void stopAP() {
  if (apStarted) {
    WiFi.softAPdisconnect(true);
    Serial.println(" AP вимкнено");
    apStarted = false;
  }
}

void connectToWiFi() {
  String ssid = preferences.getString("ssid", default_ssid);
  String pass = preferences.getString("password",
default_password);
  WiFi.begin(ssid.c_str(), pass.c_str());
  Serial.println("Підключення до Wi-Fi: " + ssid);
}

void stepMotor(bool forward) {
  if (stepsRemaining > 0) {
    stepIndex = (stepIndex + (forward ? 1 : 7)) % 8;
    digitalWrite(IN1, bitRead(stepSequence[stepIndex], 0));
    digitalWrite(IN2, bitRead(stepSequence[stepIndex], 1));
    digitalWrite(IN3, bitRead(stepSequence[stepIndex], 2));
    digitalWrite(IN4, bitRead(stepSequence[stepIndex], 3));
    delayMicroseconds(speedMotor);
    stepsRemaining--;
```

```
    currentPosition += (forward ? 1 : -1);

    if (stepsRemaining == 0) {
      motorRunning = false;
      currentState = (currentPosition <= 0) ? CLOSED :
(currentPosition >= fullTravel) ? OPENED : STOPPED;
      const char* status = (currentState == OPENED) ? "opened"
: (currentState == CLOSED) ? "closed" : "stopped";
      client.publish(mqtt_topic_status, status);
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length)
{
  String cmd;
  for (unsigned int i = 0; i < length; i++) cmd +=
(char)payload[i];
  cmd.trim();
  Serial.println(" Команда: " + cmd);

  if (motorRunning && (cmd == "open" || cmd == "close"))
return;

  if (cmd == "open" && currentPosition < fullTravel) {
    stepsRemaining = fullTravel - currentPosition;
    currentState = OPENING;
    motorRunning = true;
    client.publish(mqtt_topic_status, "opening");
  } else if (cmd == "close" && currentPosition > 0) {
    stepsRemaining = currentPosition;
    currentState = CLOSING;
    motorRunning = true;
    client.publish(mqtt_topic_status, "closing");
  } else if (cmd == "stop") {
    stepsRemaining = 0;
```

```
    motorRunning = false;
    currentState = STOPPED;
    client.publish(mqtt_topic_status, "stopped");
  }
}

void reconnectMQTT() {
  if (!client.connected()) {
    String mqtt_ip = preferences.getString("IP_Server",
default_mqtt_ip);
    client.setServer(mqtt_ip.c_str(), mqtt_port);
    if (client.connect("ESP32_Gate")) {
      Serial.println(" MQTT підключено");
      client.subscribe(mqtt_topic_control);
      client.publish(mqtt_topic_status, "ready");
    } else {
      Serial.println(" MQTT недоступний");
    }
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT); pinMode(IN3,
OUTPUT); pinMode(IN4, OUTPUT);
  preferences.begin("wifi-creds", false);
  WiFi.mode(WIFI_AP_STA);
  connectToWiFi();

  if (WiFi.waitForConnectResult() != WL_CONNECTED) {
    startAP();
    startWebInterface();
  }

  client.setCallback(callback);
  currentState = CLOSED;
```

```
}

void loop() {
  if (webStarted) server.handleClient();

  bool wifiConnected = WiFi.status() == WL_CONNECTED;
  bool mqttConnected = client.connected();

  if (!wifiConnected) {
    if (!apStarted) startAP();
    if (!webStarted) startWebInterface();
  } else {
    if (apStarted) stopAP();
    if (mqttConnected && webStarted) stopWebInterface();
  }

  if (wifiConnected) {
    if (!mqttConnected) reconnectMQTT();
    else client.loop();
  }

  if (motorRunning) stepMotor(currentState == OPENING);
}
```

Додаток Д

Лістинг програмного коду RFID-зчитувача

```
#include <SPI.h>
#include <MFRC522.h>
#include <WiFi.h>
#include <WebServer.h>
#include <Preferences.h>
#include <PubSubClient.h>

#define RST_PIN 22
#define SS_PIN 21

Preferences preferences;
MFRC522 rfid(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;

const char* default_ssid = "TP-Link_0B0B";
const char* default_password = "fnaf_009";
const char* default_mqtt_ip = "192.168.188.61";
const int mqtt_port = 1883;
const char* mqtt_topic = "rfid/UID";
const char* ap_password = "123456789";

WebServer server(80);
WiFiClient espClient;
PubSubClient mqtt(espClient);

String lastUID = "";
unsigned long lastScanTime = 0;
const unsigned long debounceDelay = 5000;

bool apStarted = false;
bool webStarted = false;
bool mqttStatus = false;
bool wifiConnected = false;
```

```
String currentSSID = "";
String currentIP = "";
String currentMQTT = "";

void handleMainMenu() {
  String html = "<html><head><meta charset='utf-
8'></head><body><h1>ESP32: Налаштування</h1><ul>";
  html += "<li><a href=\"/wifi\">Wi-Fi</a></li>";
  html += "<li><a href=\"/server\">MQTT IP</a></li></ul>";
  html += "<h3>Стан:</h3><ul>";
  html += "<li>Wi-Fi SSID: " + currentSSID + "</li>";
  html += "<li>ESP IP: " + currentIP + "</li>";
  html += "<li>MQTT сервер: " + currentMQTT + "</li>";
  html += "<li>MQTT статус: " + String(mqttStatus ?
"Підключено ✅" : "Не підключено ❌") + "</li>";
  html += "</ul></body></html>";
  server.send(200, "text/html; charset=utf-8", html);
}

void handleWiFiForm() {
  String html = "<html><head><meta charset='utf-
8'></head><body><h1>Wi-Fi</h1>"
                "<form action=\"/save\" method=\"POST\">SSID:
<input name=\"ssid\"><br>"
                "Пароль: <input name=\"password\"
type=\"password\"><br>"
                "<input type=\"submit\"
value=\"Зберегти\"></form></body></html>";
  server.send(200, "text/html; charset=utf-8", html);
}

void handleSaveWiFi() {
  preferences.putString("ssid", server.arg("ssid"));
  preferences.putString("password", server.arg("password"));
```

```
  server.send(200, "text/html", "<html><body><h1>Wi-Fi
збережено. Перезапуск...</h1></body></html>");
  delay(1000);
  ESP.restart();
}

void handleServerForm() {
  String html = "<html><head><meta charset='utf-
8'></head><body><h1>MQTT сервер</h1>"
                "<form action=\"/saveServer\"
method=\"POST\">IP: <input name=\"IP_Server\"><br>"
                "<input type=\"submit\"
value=\"Зберегти\"></form></body></html>";
  server.send(200, "text/html; charset=utf-8", html);
}

void handleSaveServer() {
  preferences.putString("IP_Server", server.arg("IP_Server"));
  server.send(200, "text/html", "<html><body><h1>IP збережено.
Перезапуск...</h1></body></html>");
  delay(1000);
  ESP.restart();
}

void startWebInterface() {
  if (!webStarted) {
    server.on("/", handleMainMenu);
    server.on("/wifi", handleWiFiForm);
    server.on("/save", HTTP_POST, handleSaveWiFi);
    server.on("/server", handleServerForm);
    server.on("/saveServer", HTTP_POST, handleSaveServer);
    server.begin();
    webStarted = true;
    Serial.println(" Вебсервер запущено: http://" +
currentIP);
  }
```

```
}

void stopWebInterface() {
  if (webStarted) {
    server.stop();
    webStarted = false;
    Serial.println(" Вебсервер вимкнено");
  }
}

void startAP() {
  if (!apStarted) {
    String chipID = String((uint32_t)ESP.getEfuseMac(), HEX);
    String apSSID = "ESP32_Config_" + chipID;
    WiFi.softAP(apSSID.c_str(), ap_password);
    apStarted = true;
    currentIP = WiFi.softAPIP().toString();
    Serial.println(" Точка доступу: " + apSSID);
    Serial.println(" http://" + currentIP);
    startWebInterface();
  }
}

void stopAP() {
  if (apStarted) {
    WiFi.softAPdisconnect(true);
    apStarted = false;
    Serial.println(" AP режим вимкнено");
  }
}

void connectToWiFi() {
  String ssid = preferences.getString("ssid", default_ssid);
  String pass = preferences.getString("password",
default_password);
  WiFi.mode(WIFI_STA);
```

```
  WiFi.begin(ssid.c_str(), pass.c_str());
  currentSSID = ssid;
  Serial.println(" Підключення до Wi-Fi: " + ssid);

  unsigned long startAttemptTime = millis();
  const unsigned long timeout = 10000;

  while (WiFi.status() != WL_CONNECTED && millis() -
startAttemptTime < timeout) {
    delay(500);
    Serial.print(".");
  }

  if (WiFi.status() == WL_CONNECTED) {
    wifiConnected = true;
    currentIP = WiFi.localIP().toString();
    Serial.println("\n Підключено до Wi-Fi: " + currentIP);
    stopAP();
    startWebInterface();
  } else {
    wifiConnected = false;
    Serial.println("\n Wi-Fi не підключено");
    startAP();
  }
}

void reconnectMQTT() {
  if (!mqtt.connected()) {
    currentMQTT = preferences.getString("IP_Server",
default_mqtt_ip);
    mqtt.setServer(currentMQTT.c_str(), mqtt_port);
    if (mqtt.connect("ESP32_RFID")) {
      mqttStatus = true;
      Serial.println(" Підключено до MQTT: " + currentMQTT);
      stopWebInterface();
    } else {
```

```
      mqttStatus = false;
      Serial.println(" Помилка підключення до MQTT");
    }
  }
}

void setup() {
  Serial.begin(115200);
  SPI.begin(18, 19, 23, SS_PIN);
  rfid.PCD_Init();
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;

  preferences.begin("wifi-creds", false);

  connectToWiFi();

  if (wifiConnected) {
    reconnectMQTT();
  }
}
void loop() {
  if (webStarted) server.handleClient();

  if (wifiConnected && WiFi.status() == WL_CONNECTED) {
    mqtt.loop();
    if (!mqtt.connected()) reconnectMQTT();
  }

  if (!rfid.PICC_IsNewCardPresent() ||
!rfid.PICC_ReadCardSerial()) return;

  String uidString = "";
  for (byte i = 0; i < rfid.uid.size; i++) {
    if (rfid.uid.uidByte[i] < 0x10) uidString += "0";
    uidString += String(rfid.uid.uidByte[i], HEX);
  }
```

```
  uidString.toUpperCase();

  unsigned long now = millis();
  if (uidString == lastUID && now - lastScanTime <
debounceDelay) return;

  lastUID = uidString;
  lastScanTime = now;

  Serial.println(" UID: " + uidString);
  mqtt.publish(mqtt_topic, uidString.c_str());

  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
  delay(500);
}
```