Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Бакалаврська кваліфікаційна робота на тему:
«Засіб контролю доступу до приміщення»

Виконав: студент 4 курсу групи 1БКС-21б
спеціальності 125 Кібербезпека
_____ Ілля КОШЕВЕЦЬ
Керівник: к. т. н., доцент каф. ЗІ
_____ Юрій БАРИШЕВ
«16» червня 2025 р.
Рецензент: к. т. н. доцент каф. ПЗ
_____ Вікторія ВОЙТКО
«16» червня 2025 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«16» червня 2025 р.

Вінниця ВНТУ – 2025 року

Міністерство освіти і науки України
Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти І (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Кібербезпека критичних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
«21» березня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Іллі КОШЕВЦЮ

1. Тема роботи: «Засіб контролю доступу до приміщення»
   керівник роботи: Юрій БАРИШЕВ, к. т. н., доцент кафедри ЗІ,
   затверджені наказом ректора ВНТУ від 20 березня 2025 року №97.

2. Строк подання студентом роботи 16.06 2025 р.

3. Вихідні дані до роботи:
   - фактор автентифікації RFID-картка;
   - модель розмежування прав доступу – рольова;
   - можливість редагування правил доступу.

4. Зміст текстової частини: Вступ. 1. Аналіз засобів контролю доступу. 2. Архітектура засобу контролю доступу. 3. Алгоритм роботи засобу контролю доступу. 4. Тестування засобу. Висновки. Список використаних джерел. Додатки.

5. Перелік ілюстративного матеріалу: аналіз методів автентифікації, порівняльна таблиця типів засобів доступу, методика контролю доступу, узагальнена архітектура, блок автентифікації користувачів, сутності бази даних правил розмежування доступу, узагальнений алгоритм роботи засобу, алгоритм реєстрування користувачів, результати модульного тестування, результати статичного тестування безпеки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	21.03	14.0
2	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	21.05	05.0
3	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	21.03	12.0
4	Ю. БАРИШЕВ, к.т.н., доц. каф.ЗІ	21.03	19.0

7. Дата видачі завдання 21.03 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	24.03.25 – 30.03.25	
2	Аналіз інформаційних джерел за напрямком бакалаврської кваліфікаційної роботи	31.03.25 – 13.04.25	
3	Розробка рішень, моделей, алгоритмів	14.04.25 – 04.05.25	
4	Практична реалізація, моделювання, експериментування, результати	05.05.25 – 14.05.25	
5	Аналіз виконання БКР, висновки	15.05.25 – 18.05.25	
6	Оформлення пояснювальної записки	19.05.25 – 26.05.25	
7	Попередній захист БКР	27.05.25 – 30.05.25	
8	Виправлення зауважень, перевірка на наявність текстових запозичень, підготовка ілюстративного матеріалу	31.05.25 – 10.06.25	
9	Представлення БКР до захисту, рецензування	11.06.25 – 13.06.25	
10	Захист БКР	17.06.25 – 20.06.25	

Студент _____ Ілля КОШЕВЕЦЬ

Керівник роботи _____ Юрій БАРИШЕВ

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 94 сторінок формату А4, на яких є 18 рисунків, 12 таблиць, список використаних джерел містить 25 найменування

Бакалаврська кваліфікаційна робота присвячена розробці програмного засобу для контролю доступу до приміщень. В результаті аналізу систем контролю доступу та методів автентифікації було виявлено низку недоліків, що стосуються обмеженої гнучкості та неефективного контролю за тимчасовими користувачами. Для вирішення цих проблем обрано підхід, який передбачає динамічне обмеження доступу за часом, зоною та кількістю використань.

Для реалізації цього підходу було розроблено архітектуру системи, що включає модель розмежування прав користувачів та структуру бази даних. Після розробки схем функціонування програмного засобу в цілому та алгоритмів його окремих складових зокрема, реєстрації, редагування прав та автентифікації користувачів, здійснено програмну реалізацію. Проведене модульне та статичне тестування безпеки підтвердило функціональність системи та дозволило виявити і виправити потенційні вразливості.

Ключові слова: контроль доступу, автентифікація, розмежування прав, RFID-картки, база даних, кібербезпека, тимчасові правила, тестування.

ABSTRACT

The bachelor thesis consists of 94 A4 pages, which include 18 figures, 12 tables, and a list of references containing 25 titles.

The bachelor qualification thesis is dedicated to the development of a software tool for premises access control. As a result of analyzing access control systems and authentication methods, a number of shortcomings were identified, including limited flexibility and ineffective control over temporary users. To address these issues, an approach was chosen that provides for dynamic access restrictions based on time, zone, and number of uses.

To implement this approach, a system architecture was developed that includes a user access control model and a database structure. After designing the overall operation schemes of the software tool and the algorithms of its individual components—namely, user registration, rights editing, and authentication—software implementation was carried out. Module and static security testing confirmed the functionality of the system and allowed for the identification and correction of potential vulnerabilities.

Keywords: access control, authentication, rights differentiation, RFID cards, database, cybersecurity, temporary rules, testing.

.

ЗМІСТ

ВСТУП

Питання безпеки стають дедалі актуальнішими, зокрема в контексті контролю фізичного доступу до об'єктів з обмеженим режимом. Зокрема, офісні, промислові та державні установи дедалі частіше стають об’єктами несанкціонованого проникнення, що призводить до витоку конфіденційної інформації, матеріальних збитків та порушення роботи інфраструктури. Тому необхідність у надійних, адаптивних та інтелектуальних системах контролю доступу є критично важливою.

Зростання кіберзагроз та фізичних ризиків, пов’язаних із проникненням до захищених об'єктів, виникає потреба у впровадженні інтелектуальних систем доступу, здатних не лише ідентифікувати користувачів, а й гнучко реагувати на зміни поведінки та спроби несанкціонованого входу. Використання систем із часовими обмеженнями доступу, журналами подій та адаптивними механізмами контролю дозволяє значно підвищити рівень безпеки та зменшити вразливості.

Відомі на ринку рішення часто мають низку недоліків, зокрема обмеження в масштабованості, нестачу гнучкості у налаштуванні прав доступу та слабкий захист від атак типу «соціальна інженерія». Крім того, багато систем не дозволяють враховувати тимчасові обмеження чи змінювати рівень доступу в режимі реального часу, що значно знижує їхню ефективність в умовах динамічного середовища. Такі обмеження роблять традиційні підходи до контролю доступу недостатніми для повноцінного захисту.

У зв’язку з цим розробка та впровадження більш гнучких і захищених систем контролю доступу, які враховують як часові обмеження, так і індивідуальні параметри користувачів, є надзвичайно актуальним завданням. Впровадження таких систем дозволяє не лише підвищити безпеку, а й оптимізувати управління доступом до ресурсів, забезпечити прозоре ведення журналів подій і гнучко реагувати на інциденти безпеки.

Тому розробка інтелектуальної системи контролю доступу, що враховує індивідуальні параметри користувачів, часові обмеження та рівні доступу, є не лише актуальною, а й необхідною умовою забезпечення безпеки об’єктів різного

призначення. Така система має забезпечувати гнучке управління, швидку обробку запитів та можливість масштабування відповідно до потреб організації.

Об’єктом бакалаврської кваліфікаційно роботи є процес розмежування прав доступу.

Предметом роботи є методи і засоби розмежування прав доступу до приміщення.

Метою бакалаврської кваліфікаційно роботи є підвищення захисту конфіденційності носіїв даних шляхом впровадження правил розмежування прав доступу до приміщення, що враховують розклад роботи працівників.

Для досягнення мети необхідно виконати такі завдання:

- проаналізувати відомі підходи до розмежування прав доступу до приміщення;
- виконати порівняльний аналіз відомих засобів розмежування прав доступу до приміщення;
- розробити архітектуру засобу;
- розробити даталогічну модель бази даних прав доступу;
- розробити алгоритм роботи засобу на основі архітектури;
- виконати тестування коректності реалізації засобу;
- провести експериментальні дослідження засобу розмежування прав доступу до приміщення.

Результати бакалаврської кваліфікаційно роботи доповідались на LIV Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії [1].

1 АНАЛІЗ ЗАСОБІВ КОНТРОЛЮ ДОСТУПУ ДО ПРИМІЩЕННЯ

1.1Аналіз систем контролю доступу до приміщень

Безпека будівель та приміщень завжди була важливим питанням як для державних установ, так і для приватних підприємств. Із розвитком технологій змінюються й методи, за допомогою яких обмежується доступ до певних зон. Якщо раніше основним засобом була проста механічна охорона або ключі, то з часом на перше місце вийшли автоматизовані системи контролю.

Системи контролю доступу до приміщень (СКД) — це сукупність технічних та програмних засобів, призначених для регулювання переміщення осіб або транспортних засобів через певні точки доступу, такі як двері, ворота чи турнікети. Головна мета СКД — забезпечення безпеки об'єкта шляхом запобігання несанкціонованому доступу та ведення обліку переміщень персоналу [2].

Системи контролю доступу до приміщень призначені для обмеження входу в певні зони тільки тим людям, які мають відповідні права. Це дозволяє захищати приміщення від сторонніх осіб, стежити за рухом працівників і гостей, а також фіксувати факти доступу в електронному журналі.

Класифікація систем контролю доступу залежить від способу їх реалізації, функціональності та рівня взаємодії з іншими елементами безпеки. Загалом, СКД можна поділити на три основні типи: автономні, мережні та універсальні [3]. На рисунку 1.1 наведена схема, яка відображає основні типи систем контролю доступу:

Рисунок 1.1 – Загальна схема класифікації СКД

Автономні системи контролю доступу функціонують без підключення до мережі або централізованого керування. Уся логіка роботи таких систем

зосереджена безпосередньо в зчитувачі або контролері, встановленому на точці доступу. Найчастіше їх застосовують на об’єктах із невисоким рівнем вимог до безпеки — наприклад, у невеликих приміщеннях чи офісах. Вони складаються з простих пристроїв, таких як шлагбауми, механічні замки, кодові панелі, турнікети або ворота [4].

Мережна система контролю та управління доступом (СКУД) має низку розширених можливостей, таких як ведення обліку робочого часу працівників, гнучке налаштування прав доступу відповідно до робочих графіків, днів тижня або інших факторів. Завдяки централізованому керуванню, система дозволяє фіксувати не лише факт входу чи виходу з будівлі, а й переміщення працівників по об'єкту, включаючи відвідування певних зон, перерви на обід чи відсутність на робочому місці [5].

Кожному співробітнику може бути надано індивідуальні права доступу до окремих приміщень, або ж реалізовано спрощену систему з фіксацією лише точок входу/виходу. Такі системи підтримують різні типи ідентифікаторів — від безконтактних карток і брелків до біометричних даних (відбитків пальців тощо). Сучасні контролери можуть працювати в різних режимах: лише з карткою, картка або код, картка та код, або ж комбінації з біометрією [6].

Система контролю доступу може бути складовою частиною більш широкої універсальної системи безпеки, що функціонує на підприємстві. Такі системи поєднують технічні та програмні засоби, спрямовані на забезпечення комплексного захисту об’єкта та підтримку його функціональної діяльності. Вони дозволяють централізовано управляти безпековими процесами та контролювати ключові аспекти доступу до приміщень.

Універсальні системи здатні функціонувати як у автономному, так і в мережному режимах, автоматично переходячи в автономний режим у разі збою центрального пристрою керування [7].

Кожен тип системи контролю доступу має свої особливості, які впливають на вибір рішення залежно від потреб об'єкта. У таблиці 1.1 наведено порівняння основних типів СКД за їх перевагами та недоліками.

Таблиця 1.1 – Типи систем контролю доступу

Тип СКД	Переваги	Недоліки
Автономна	простота встановлення; низька вартість; не потребує мережі.	обмежена функціональність; відсутність централізованого контролю.
Мережна	централізоване керування; гнучке налаштування доступу; статистика.	вища вартість; складність налаштування та обслуговування.
Інтегрована	комплексна безпека; можливість інтеграції з іншими системами; висока автоматизація.	висока вартість; складність проєктування; потреба у фахівцях.

Кожен тип СКД має свої переваги залежно від сфери застосування. Автономні підходять для простих задач, мережні — для офісів і підприємств із більшими вимогами до контролю, а інтегровані — для об’єктів, де критично важливе повне поєднання безпеки й автоматизації.

Загалом, системи контролю доступу відіграють ключову роль у забезпеченні фізичної безпеки об’єктів різного призначення — від невеликих офісів до масштабних підприємств. Вони можуть мати різний рівень складності: від простих автономних пристроїв до інтегрованих рішень, що взаємодіють з іншими системами безпеки. Інвестиції в сучасні СКД сприяють не лише захисту майна, а й підвищенню продуктивності персоналу через оптимізацію процесів доступу. Впровадження комплексних рішень дозволяє створити надійну та гнучку систему безпеки, здатну адаптуватися до змін у вимогах та технологіях.

Правильний вибір типу СКД та її компонентів залежить від специфіки об’єкта, вимог до рівня захисту, бюджету та необхідного функціоналу, що, своєю чергою, впливає на ефективність контролю доступу та загальний рівень безпеки організації.

1.2 Аналіз факторів автентифікації під час контролю доступу до приміщень

Якісний контроль доступу неможливий без ефективного механізму автентифікації, адже саме він дозволяє впевнено встановити особу користувача перед наданням доступу до приміщення чи системи. У цьому підрозділі доцільно розглянути основні етапи автентифікації та їхню роль у забезпеченні фізичної безпеки об'єктів.

Автентифікація — це процес перевірки, який гарантує, що доступ до корпоративних ресурсів отримують лише ті користувачі, служби чи програми, які мають на це відповідні права. Вона є критично важливим елементом системи кібербезпеки, адже одним із головних завдань зловмисників є несанкціонований доступ до інформаційних систем. Для досягнення цього вони намагаються викрасти облікові дані — логіни та паролі легітимних користувачів [8].

Процедура автентифікації зазвичай охоплює три основні етапи: ідентифікація, автентифікація і авторизація.

Ідентифікація — це перший етап процесу автентифікації, який полягає в тому, щоб система дізналася, хто саме намагається отримати доступ. Зазвичай це відбувається через введення імені користувача або ідентифікатора, який є унікальним для кожного користувача чи пристрою. Це можна порівняти з процесом пред'явлення посвідчення особи. Ідентифікація є лише попереднім кроком і не дає жодних прав на доступ до системи. Вона лише дозволяє системі зрозуміти, з ким вона має справу [9].

Автентифікація є основним етапом перевірки особистості користувача. Після того як система визначить, хто перед нею (ідентифікація), наступним кроком є перевірка, чи дійсно ця особа має право отримати доступ. Це досягається через використання певного фактору автентифікації.

Фактори автентифікації — це основа, на якій будується перевірка особи користувача в системах контролю доступу. Залежно від характеру підтвердження, вони поділяються на кілька категорій.

Існують три основні типи факторів автентифікації [10]:

- те, що знає користувач;
- те, що має користувач;
- те, що характеризує тіло користувача.

Першу групу становлять знання, якими володіє користувач. Це може бути пароль, PIN-код, відповідь на секретне запитання тощо. Такий спосіб є найпоширенішим, але й найбільш вразливим до атак, пов’язаних із підбором або викраденням облікових даних.

Друга категорія — це те, що користувач має. Це можуть бути смарт-карти, RFID-мітки, апаратні токени або мобільні пристрої з генераторами кодів. Вони забезпечують вищий рівень захисту, оскільки зловмиснику потрібно мати фізичний доступ до пристрою.

Третій тип — це біометричні характеристики особи: відбитки пальців, геометрія обличчя, райдужка ока, голос тощо. Біометрія важко підроблюється, однак потребує точного [11]. У таблиці 1.2 наведено порівняльний огляд основних методів автентифікації.

Таблиця 1.2 – Аналіз методів автентифікації

Критерій	Пароль / PIN-код	Картка доступу / Брелок	Біометрія
Тип фактору	Знання	Наявність предмета	Біологічна ознака
Стійкість до підбору / зламу	Низька (легко вгадати, фішинг)	Середня (можна скопіювати чи вкрасти)	Висока (важко підробити або відтворити)
Необхідне обладнання	Немає	Карт-рідер, RFID-система	Біометричний сканер
Підходить для	Приміщення з базовим рівнем захисту	Офіси, підприємства	Об'єкти з високими вимогами безпеки

Завдяки використанню кількох факторів автентифікації (так званий багатофакторний підхід) підвищується безпека процесу. Наприклад, користувач може ввести пароль і потім підтвердити свою особистість за допомогою одноразового коду, що надійшов на його телефон.

Після того, як користувач успішно пройшов автентифікацію, наступним етапом є авторизація. Це процес перевірки того, чи має користувач відповідні права для доступу до конкретних ресурсів або виконання певних дій у системі.

Авторизація визначає, що саме може робити користувач після того, як його особистість була підтверджена. Наприклад, один користувач може мати права лише на перегляд певних даних, а інший — на редагування цих даних або виконання адміністративних операцій. Авторизація базується на ролях, групах, політиках або списках контролю доступу (ACLs), які визначають, хто і що може робити в системі.

Зазвичай авторизація включає в себе такі кроки:

- перевірка прав доступу до конкретних ресурсів;
- перевірка ролей користувача в організації (наприклад, адміністратор, менеджер, співробітник тощо);
- перевірка обмежень на виконання певних операцій (наприклад, доступ до конфіденційної інформації, редагування даних, запуск програм).

Загалом ці три етапи разом забезпечують належний рівень безпеки, захищаючи системи від несанкціонованого доступу та неправильного використання ресурсів [11].

Двофакторна автентифікація (2FA) є ефективним способом підвищення безпеки облікових записів. За словами експертів, впровадження цього методу істотно зменшує ймовірність несанкціонованого доступу, особливо коли другим фактором виступає біометрична перевірка. Саме тому рекомендовано якомога раніше налаштувати 2FA, особливо у випадках, коли йдеться про фінансові операції чи доступ до конфіденційної інформації.

Суть двофакторної автентифікації полягає в тому, що користувач проходить перевірку за допомогою двох різних типів даних. Один з них – це інформація, яку знає користувач (наприклад, пароль), а інший – або фізичний носій (наприклад, мобільний телефон із кодом підтвердження), або біометричний параметр (відбиток пальця, скан обличчя тощо). Такий підхід значно ускладнює зловмисникам можливість отримати повний доступ до системи [12].

Порівняння методів автентифікації для контролю доступу до приміщень демонструє важливість вибору підходу залежно від рівня безпеки та зручності для користувачів. Традиційні методи, такі як паролі та PIN-коди, забезпечують базовий рівень захисту, але вони піддаються ризику викрадення або забуття. Методи, що ґрунтуються на чомусь, що має користувач (картки доступу, мобільні пристрої), є більш зручними і менш вразливими до підбору, але потребують фізичної наявності цих пристроїв. Біометричні методи, хоча й забезпечують високий рівень безпеки, мають свої вразливості, такі як можливість підробки або впливу зовнішніх факторів.

Двофакторна автентифікація, що поєднує кілька з цих методів, є найбільш ефективною, оскільки вона значно ускладнює можливість несанкціонованого доступу. Однак впровадження таких систем вимагає значних витрат на обладнання та підтримку. Таким чином, для досягнення високого рівня безпеки необхідно враховувати специфіку приміщення, тип користувачів та фінансові можливості організації.

1.3 Порівняльний аналіз засобів контролю доступу до приміщень

Засоби контролю доступу до приміщень є важливою складовою системи безпеки будь-якого об'єкта. Вони забезпечують фізичний захист, дозволяючи доступ лише уповноваженим особам та запобігаючи несанкціонованому проникненню. Сучасні технології пропонують різноманітні методи автентифікації та засоби доступу, які використовуються в залежності від рівня безпеки, потреб користувачів і фінансових можливостей організації. Вибір засобу контролю доступу безпосередньо залежить від специфіки об'єкта, кількості користувачів та вимог до інтеграції з іншими системами безпеки.

Існує кілька основних типів засобів контролю доступу, які відрізняються за принципом роботи, рівнем безпеки та способом ідентифікації користувача. До них належать механічні (традиційні ключі та замки), електронні (картки, кодові панелі), біометричні (відбитки пальців, сканування обличчя або сітківки) та мобільні (смартфони з NFC, QR-коди або Bluetooth). Кожен тип має свої переваги та

недоліки, які наведено в таблиці 1.3, а вибір конкретного рішення залежить від цілей, бюджету та рівня загроз для об'єкта.

Таблиця 1.3 – Порівняльна таблиця типів засобів доступу

Тип засобів доступу	Потенційні вразливості	Приклади використання	Вартість впровадження
Механічний ключ	Легко скопіювати або втратити	Невеликі офіси, склади, побутові приміщення	Низька
RFID-картка / брелок	Крадіжка або клонування	Бізнес-центри, школи, гуртожитки	Середня
PIN-код	Підглядання, витік пароля	Технічні приміщення, серверні	Низька
Біометрія	Труднощі з точністю, конфіденційність	Лабораторії, дата-центри, урядові об'єкти	Висока

Серед розглянутих засобів особливої уваги заслуговують RFID-картки, які є компромісом між вартістю, зручністю та безпекою. Вони широко застосовуються в офісах, навчальних закладах, гуртожитках та інших об’єктах з помірними вимогами до контролю доступу. Проте їх функціонал здебільшого обмежується статичною автентифікацією без урахування контексту використання — зони доступу, часу, кількості входів тощо. У зв’язку з цим доцільним є вдосконалення RFID-карток шляхом додавання часової логіки, можливості гнучкого управління правами доступу та фіксації подій. Такий підхід дозволить значно підвищити рівень безпеки без суттєвого збільшення вартості впровадження системи, що робить його привабливим для підприємств з обмеженим бюджетом.

Серед усіх засобів контролю доступу, RFID-картки займають важливе місце завдяки своїй простоті використання, відносно невисокій вартості та можливості інтеграції з різними системами. Вони дозволяють ефективно керувати допуском до приміщень, зокрема тимчасовим, що особливо актуально для підприємств, де постійно змінюються потоки персоналу та відвідувачів. Проте не всі RFID-картки однакові — вони відрізняються між собою за технічними характеристиками, функціональністю та рівнем захисту. Тому для правильного вибору та подальшого

вдосконалення системи доступу важливо розуміти, які існують основні типи RFID-карток [13].

Основні типи RFID-карток [13]:

- ID-картки (125 кГц) — ці картки працюють на низькій частоті 125 кГц і мають фіксований ідентифікатор, запрограмований виробником. Вони є лише для читання, не підтримують шифрування та легко копіюються, що робить їх менш безпечними.
- IC-картки (13.56 МГц) — працюють на високій частоті 13.56 МГц, підтримують перезапис, шифрування та складніші протоколи автентифікації. Вони важче піддаються копіюванню та забезпечують вищий рівень безпеки.
- UHF-картки (860–960 МГц) — використовуються для ідентифікації на великих відстанях, наприклад, у логістиці або для контролю доступу на великі об'єкти. Підтримують стандарти EPC Gen2 та забезпечують швидке зчитування.
- Смарт-картки з NFC — ці картки підтримують Near Field Communication (NFC) і можуть взаємодіяти зі смартфонами та іншими пристроями, що підтримують NFC. Вони забезпечують високий рівень безпеки та зручність у використанні.

Сучасні системи контролю доступу широко використовують різні типи RFID-карток, кожен з яких має свої технічні особливості та рівень захисту. Вибір відповідного типу картки є критично важливим для забезпечення ефективної та безпечної роботи всієї системи. Перед впровадженням рішення необхідно враховувати такі параметри, як робоча частота, можливість програмування, рівень криптографічного захисту та відповідність умовам експлуатації.

Кожен тип RFID-карток має свої особливості, що визначають сферу їх застосування та рівень захисту від несанкціонованого доступу. Знання цих характеристик дозволяє обрати варіант для конкретних умов експлуатації системи контролю доступу [13]. У таблиці 1.4 наведено основні характеристики найбільш

поширених типів карток, включаючи частоту роботи, рівень безпеки, можливість програмування та сфери застосування.

Таблиця 1.4 – Порівняння типів безконтактних карток

Тип картки	Частота	Безпека	Програмування	Застосування
ID-картка	125 кГц	Низька	Ні	Офіси, склади
IC-картка	13.56 МГц	Висока	Так	Бізнес-центри, навчальні заклади
UHF-картка	860–960 МГц	Середня	Так	Логістика, великі підприємства
NFC-смарт-картка	13.56 МГц	Висока	Так	Мобільні платежі, доступ через смартфон

Таким чином, вибір типу RFID-картки залежить від вимог до безпеки, бюджету та умов використання. Карти з більшою пам’яттю та криптографічним захистом є кращим рішенням для об’єктів з підвищеними вимогами до контролю доступу.

1.4 Постановка завдання

Більшість розглянутих засобів мають певні недоліки — це або низький рівень захисту (як у механічних ключів чи PIN-кодів), або висока вартість впровадження (як у біометричних систем). RFID-картки, попри свою популярність, також не позбавлені вразливостей, зокрема можливості клонування або неконтрольованого доступу. Тому виникає необхідність удосконалити ці системи, підвищивши їхню гнучкість, безпечність і можливості керування правами доступу, зокрема в умовах тимчасового чи обмеженого перебування сторонніх осіб на об’єкті.

Попри широке розмаїття засобів контролю доступу, багато з них мають обмеження в частині гнучкого налаштування прав доступу, особливо для тимчасових користувачів. У більшості випадків системи не передбачають обмеження доступу за часом або зоною перебування, що створює потенційні ризики. Відсутність динамічного керування правами користувача, а також слабкий

рівень контролю за сторонніми особами, які тимчасово перебувають на території об'єкта, знижує загальний рівень безпеки.

Зважаючи на виявлені обмеження, виникає необхідність удосконалення відомих підходів шляхом впровадження гнучкіших засобів контролю, які дозволяють обмежувати доступ не лише за особистістю користувача, а й за контекстом — часом перебування, одноразовістю використання, та визначеними зонами. Такий підхід дає змогу більш ефективно організувати контроль за тимчасовим персоналом і сторонніми відвідувачами, забезпечуючи при цьому прозору та керовану модель доступу.

1.5 Висновки з розділу

У першому підрозділі було проаналізовано основні типи систем контролю доступу до приміщень, методи автентифікації та фактори, що впливають на їхню ефективність. Розглянуто переваги й недоліки механічних, електронних, біометричних та мобільних рішень. Проведене порівняння дозволило оцінити кожен тип за критеріями безпеки, вартості, зручності та можливостей інтеграції.

У результаті аналізу встановлено, що традиційні засоби доступу мають низку недоліків, серед яких — обмежена гнучкість налаштувань, відсутність часових обмежень та слабкий контроль за тимчасовими користувачами. Це створює передумови для вдосконалення цих підходів. Тому поставлено завдання розробити рішення, яке дозволить динамічно обмежувати доступ за часом, зоною, та кількістю використань, що значно підвищить рівень безпеки об’єкта при збереженні простоти впровадження.

2 АРХІТЕКТУРА ЗАСОБУ КОНТРОЛЮ ПРИМІЩЕННЯ

2.1 Методика контролю доступу

Методика контролю доступу передбачає організацію процесу, коли особи (користувачі, працівники, гості) можуть отримати доступ до визначених приміщень підприємства або організації.

Процес доступу включає кілька етапів:

- ідентифікація — особа повинна бути ідентифікована за допомогою відповідного засобу (RFID-картка, біометричні дані, PIN-код і т. д.);
- автентифікація — підтвердження ідентичності особи для того, щоб визначити її право на доступ;
- авторизація — перевірка дозволу на доступ до конкретної зони приміщення або до певних ресурсів;
- фіксація події — кожен доступ (успішний або неуспішний) фіксується в журналі подій для подальшого моніторингу та аналізу.

У системі контролю доступу можуть бути встановлені як тимчасові, так і постійні правила доступу.

Тимчасові правила доступу — це обмеження, що встановлюються на певний час, наприклад, для гостя або тимчасового працівника. Наприклад, картка може бути активована на 30 хвилин або 1 годину, після чого доступ автоматично буде заблокований. Тимчасові правила дозволяють зберігати контроль за доступом без потреби вручну знімати доступ після кожного використання.

Постійні правила доступу — ці правила призначені для співробітників або осіб, які мають постійний доступ до певних приміщень. Вони не обмежені часом і можуть включати різні рівні доступу в залежності від ролі або статусу особи.

Адміністратор є ключовою особою, відповідальною за управління доступом до приміщень. Його обов’язки можуть включати:

- видавання або скасування доступу (як для тимчасових, так і для постійних користувачів);

- налаштування правил доступу (часові обмеження, визначення рівнів доступу, дозволених зон);
- моніторинг активності користувачів та перегляд журналу подій;
- розв'язання ситуацій, коли необхідно змінити рівень доступу або тимчасово припинити доступ.

Основні правила доступу до приміщень включають надання дозволу або його відкликання, що рівноцінно обмеженню доступу.

Наявність дозволу передбачає, що доступ можливий лише тоді, коли особа має відповідний дозвіл для входу до конкретної зони.

Обмеження доступу є станом за замовчуванням, відповідно користувач без наявності дозволу (незалежно від того, чи є картка RFID, чи іншого засобу) матиме заборону входу.

Правила зниження доступу, зокрема можливість блокування картки після певного періоду часу, передбачають відмову в доступі після кількох невдалих спроб входу або скасування дозволу з плином часу. Перед наданням доступу новий користувач реєструється в системі адміністратором.

Це включає такі дії:

- занесення персональних даних (ім’я, роль, UID картки тощо);
- призначення ролі користувача (наприклад: співробітник, охоронець, відвідувач);
- визначення зон доступу;
- встановлення режиму доступу (постійний / тимчасовий / розовий);
- генерація унікального запису в базі прав доступу.

Правила доступу створюються на основі таких параметрів:

- користувач (ID, роль);
- зона / зчитувач (ID зони);
- тип доступу (постійний, тимчасовий, разовий);
- часові обмеження (наприклад, "з 9:00 до 18:00", або "30 хвилин після активації");
- умови блокування (після завершення часу, після 1 входу тощо).

Адміністратор формує або змінює правила через інтерфейс керування і вони зберігаються у базі даних у вигляді записів.

Можливості системи:

- введення нових користувачів і RFID-карток;
- гнучке призначення прав доступу відповідно до ролі;
- можливість видачі тимчасових прав доступу (з автоматичним завершенням дії);
- контроль доступу на основі UID, зони і часу;
- повний аудит дій у журналі подій.

Система контролю доступу підтримує введення тимчасових правил, що дає змогу гнучко керувати правами користувачів у залежності від контексту (події, візити, зміни, аварійні ситуації тощо). Такий підхід особливо ефективний у випадках, коли необхідно надати одноразовий або обмежений у часі доступ для:

- тимчасових працівників;
- відвідувачів підприємства;
- технічного персоналу чи постачальників;
- співробітників, які виконують разові завдання поза своїми постійними зонами.

Тимчасове правило доступу може включати:

- період дії (наприклад, з 14:00 до 14:30 певного дня);
- обмеження по зонах (наприклад, лише склад або серверна);
- одноразове спрацювання (доступ лише один раз, після чого карта блокується);
- автоматичне анулювання прав після завершення часу;
- зв’язок з певним адміністратором, який видав доступ (журналювання).

Таким чином, у бакалаврській кваліфікаційній роботі пропонується методика контролю доступу, що поєднує рольовий підхід до розмежування прав, реєстрацію користувачів, призначення постійних або тимчасових правил доступу, а також автоматизовану логіку перевірки прав на основі ідентифікації та часових обмежень.

Відповідно для реалізації методики необхідно виконати низку кроків.

Крок 1. Користувач звертається до адміністратора з проханням отримати дозвіл на вхід до певної зони. Адміністратор фіксує запит і уточнює параметри доступу (рівень, час).

Крок 2. Адміністратор перевіряє, чи має користувач відповідні права для доступу до цієї зони. Якщо потрібно, адміністратор може запросити додаткову інформацію або підтвердження.

Крок 3. Якщо перевірка пройдена успішно, адміністратор налаштовує права доступу. Це може бути тимчасовий дозвіл (наприклад, на 30 хвилин) або постійний дозвіл, залежно від запиту. Права прив’язуються до ідентифікатора користувача та RFID-картки.

Крок 4. Користувачу видають RFID-картку з налаштованими правами доступу. Картка може бути активною постійно або з часовим обмеженням.

Крок 5. Користувач прикладає картку до зчитувача у потрібній зоні. Система перевіряє права та час доступу.

Крок 6. Система порівнює отримані дані з базою прав доступу. Якщо всі умови дотримані (користувач має дозвіл, картка не прострочена, зона відповідає дозволу), доступ надається. Всі спроби входу фіксуються в журналі для подальшого аудиту. Якщо умови не виконані — доступ блокується.

Запропонована система дозволяє забезпечити гнучке управління доступом до приміщень з урахуванням реальних потреб підприємства, рівня довіри до користувача та тривалості його перебування на об'єкті. Завдяки підтримці тимчасових правил, одноразових доступів і журналювання, методика підвищує загальний рівень безпеки та дозволяє ефективно контролювати всі спроби проходу в захищені зони.

2.2 Модель розмежування прав доступу до приміщень

Модель розмежування прав доступу є ключовим елементом будь-якої системи контролю доступу, особливо коли йдеться про доступ до фізичних приміщень, таких як офіси, склади чи інші об'єкти. Основною метою такої моделі є забезпечення належного рівня безпеки, визначення, хто має право доступу до

певних зон та обмеження доступу для неповноважених осіб. У сучасних системах контролю доступу зазвичай використовуються різні моделі для розмежування прав користувачів, серед яких найпоширенішими є дискреційне та рольове розмежування.

Дискреційне (вибіркове або контрольоване) розмежування доступу — це підхід до управління доступом, за якого самі користувачі можуть у певному обсязі визначати правила безпеки для ресурсів, якими володіють. Найбільш поширеним прикладом реалізації цієї моделі є система контролю доступу до файлів і об'єктів міжпроцесної взаємодії в операційних системах родин Unix і Windows. У таких системах власник файлу має змогу самостійно змінювати права доступу до нього, наприклад, надавати до нього відкритий доступ іншим користувачам [14].

Мандатна модель розмежування доступу передбачає централізоване керування правами доступу, де кожному об’єкту присвоюється рівень секретності, а кожному користувачу — рівень допуску; доступ надається лише у випадку відповідності або перевищення рівня допуску над рівнем секретності об’єкта. Такий підхід характерний для систем з високим рівнем безпеки, зокрема у військовій чи державній сферах, проте в рамках реалізації локальної системи контролю доступу він є надто жорстким і складним. У нашій системі необхідна гнучкість, можливість оперативно змінювати права доступу, призначати тимчасові або виняткові дозволи, а також зручне адміністрування [14].

Для системи контролю доступу до приміщень на основі RFID-карток, де рівень доступу може бути чітко визначений для кожного користувача або групи користувачів, рольова модель управління доступом є найбільш доцільною. Вона дозволяє не призначати права кожному окремому користувачу вручну, а замість цього — призначати їх відповідно до ролі, яку той виконує в організації. Наприклад, ролі можуть включати адміністратора, охоронця, логіста, технічного працівника або офісного співробітника. Кожна з цих ролей має набір прав доступу, що значно спрощує процес адміністрування, дозволяє централізовано змінювати політики доступу та швидко адаптувати систему до змін у структурі персоналу.

Такий підхід також знижує ризик помилок у налаштуваннях доступу та покращує загальну безпеку системи.

У таблиці 2.1 наведено основні категорії користувачів, рівнів доступу, дозволених зон та часових обмежень, що можуть застосовуватись у системі контролю доступу.

Таблиця 2.1 – Категорії користувачів

Категорія користувача	Рівень доступу	Дозволені зони	Часовий обмеження доступу
Адміністратор	Повний доступ	Всі зони	Постійний доступ
Менеджер	Розширений доступ	Офіси, конференц-зали, зони для стратегічних зустрічей	Постійний доступ
Співробітник	Обмежений доступ	Офіси, склад, кабінети	Постійний доступ
Технічний персонал	Обмежений доступ	Серверні кімнати, технічні зони	Постійний доступ
Охоронець	Повний доступ при необхідності	Всі зони в разі надзвичайних ситуацій	В разі потреби
Консультант/Підрядник	Обмежений доступ	Спеціалізовані приміщення, кабінети	За терміном договору
Постачальник	Обмежений доступ	Склад, зони приймання товарів	За терміном договору
Гість	Обмежений доступ	Приймальня, кімната для зустрічей	Тільки на обмежений час

У рамках цієї системи виділяються кілька основних категорій користувачів, кожна з яких має свої особливості та права доступу:

- адміністратори - це користувачі з найвищими правами доступу в системі. Адміністратор має можливість налаштовувати правила доступу, видавати картки доступу, змінювати ролі користувачів і контролювати діяльність інших користувачів;
- співробітники - це постійні користувачі, які мають доступ до основних робочих зон. Вони можуть мати доступ до деяких обмежених або спеціальних зон, таких як офіси, кабінети, складські приміщення тощо, залежно від їхніх обов'язків;
- гості - це користувачі, які мають обмежений доступ до системи або приміщень. Зазвичай гості отримують тимчасовий доступ, наприклад, для проведення зустрічей або інших заходів;
- менеджери - це співробітники, які займають вищі позиції в організації і мають розширений доступ до приміщень, які пов'язані з управлінською діяльністю. Вони можуть мати доступ до більшої кількості зон, ніж звичайні співробітники, наприклад, до конференц-залів, приймальні вищого керівництва та зон, пов'язаних із стратегічними рішеннями;
- технічний персонал - це користувачі, які відповідають за обслуговування та підтримку технічного обладнання, таких як сервери, мережеві пристрої та інше обладнання, що потребує спеціалізованого доступу. Вони зазвичай має доступ до серверних кімнат, телекомунікаційних кімнат та інших спеціалізованих зон;
- охоронці - це співробітники, які відповідають за фізичну безпеку і мають доступ до різних приміщень в разі надзвичайних ситуацій або для здійснення контролю за порядком. Вони можуть мати доступ до всіх приміщень на певний час, особливо у випадках, коли потрібно перевірити зони або забезпечити порядок;
- консультанти або підрядники - це зовнішні фахівці або компанії, які працюють над конкретними проектами, наприклад, розробка програмного забезпечення або впровадження нових систем;

- постачальники, що доставляють товари або послуги в організацію, можуть мати доступ до складських приміщень або інші обмежені зони. Їхній доступ обмежений терміном і конкретними зонами, що стосуються їхніх поставок.

Визначення категорій користувачів і встановлення чітких прав доступу є важливими елементами для забезпечення безпеки в системах контролю доступу. Це дозволяє уникнути несанкціонованого доступу до обмежених або критичних зон та забезпечити ефективне використання ресурсів. Правильне розмежування доступу забезпечує належний контроль за користувачами в системі, покращує безпеку і дозволяє зберігати конфіденційність.

Таблиця 2.2 відображає основну інформацію про кожного користувача системи, включаючи його унікальний ідентифікатор, ім'я, роль, статус доступу та інші атрибути, пов'язані з доступом.

Таблиця 2.2 – Інформація про користувачів у системі

Поле	Тип даних	Опис
id_user	INT	Унікальний ідентифікатор користувача
fullname	VARCHAR(255)	Ім'я користувача
accessLevel	VARCHAR(255)	Роль користувача (менеджер, співробітник, охоронець)
access_start_time	DATETIME	Час активації доступу (для тимчасових користувачів)
access_end_time	DATETIME	Час деактивації доступу
password	VARCHAR(50)	Пароль користувача

У таблиці 2.3 зберігаються правила доступу, які визначають, чи має конкретний користувач право входу до певної зони через визначений зчитувач. Вона поєднує в собі інформацію з таблиць користувачів та зчитувачів за допомогою зовнішніх ключів. Це дозволяє гнучко налаштовувати індивідуальні або групові правила доступу, а також встановлювати часові рамки, в межах яких доступ дозволено.

Таблиця 2.3 – Права, які має користувач

Поле	Тип даних	Опис
id_rule	INT	Унікальний ідентифікатор правила
id_user	INT	Ідентифікатор користувача (зовнішній ключ на таблицю "Користувачі")
id_reader	INT	Ідентифікатор зчитувача (зовнішній ключ на таблицю "Зчитувачі")
access_granted	BOOLEAN	Доступ надано (TRUE/ FALSE)
access_start_time	DATETIME	Час активації доступу
access_end_time	DATETIME	Час деактивації доступу

У таблиці 2.4 наведено основні характеристики зчитувачів, які встановлені в системі контролю доступу. Кожен зчитувач має унікальний ідентифікатор, розміщується у певному приміщенні або зоні доступу

Таблиця 2.4 – Інформація про дозвіл у зону, яку має користувач

Поле	Тип даних	Опис
id_reader	INT	Унікальний ідентифікатор зчитувача
location_name	VARCHAR(255)	Назва або опис розташування зчитувача (наприклад, "Склад 1")
room_number	VARCHAR(50)	Номер або код приміщення
access_zone	VARCHAR(100)	Зона доступу, до якої належить зчитувач (наприклад, "Серверна")
reader_type	VARCHAR(100)	Тип зчитувача (наприклад, RFID, NFC, біометричний)
status	VARCHAR(50)	Статус зчитувача (активний, неактивний, у ремонті)

Адміністратор відповідальний за створення, активацію та деактивацію тимчасових правил доступу. Для цього він запускає спеціальну процедуру в системі, яка дозволяє задати обмежений період дії дозволу для конкретного користувача або групи користувачів. По закінченню заданого часу система автоматично відключає ці тимчасові права, але адміністратор може вручну припинити їх дію раніше у разі потреби.

Модуль часу в системі контролю доступу відповідає за точне відстеження та управління часовими обмеженнями доступу користувачів. Він забезпечує активацію тимчасових прав доступу відповідно до заданого проміжку часу. Модуль використовує внутрішній годинник реального часу (RTC) або таймер контролера для відліку часу, після чого автоматично деактивує права доступу. Це дозволяє реалізувати гнучкі політики безпеки, де доступ надається не лише за роллю чи ідентифікатором, а й з урахуванням часових параметрів, що є особливо важливим для тимчасових користувачів або обмежених зон.

2.3 Узагальнена архітектура засобу

Для ефективного функціонування системи контролю доступу важливо не лише реалізувати окремі технічні модулі, а й розуміти, як вони взаємодіють між собою в межах єдиної архітектури. У цій моделі користувачі, адміністратори, серверна логіка, база даних і механізми журналювання утворюють узгоджену структуру, де кожен компонент виконує свою роль. На рисунку 2.1 наведено узагальнену схему, яка демонструє основні елементи системи та їхні зв’язки на концептуальному рівні.

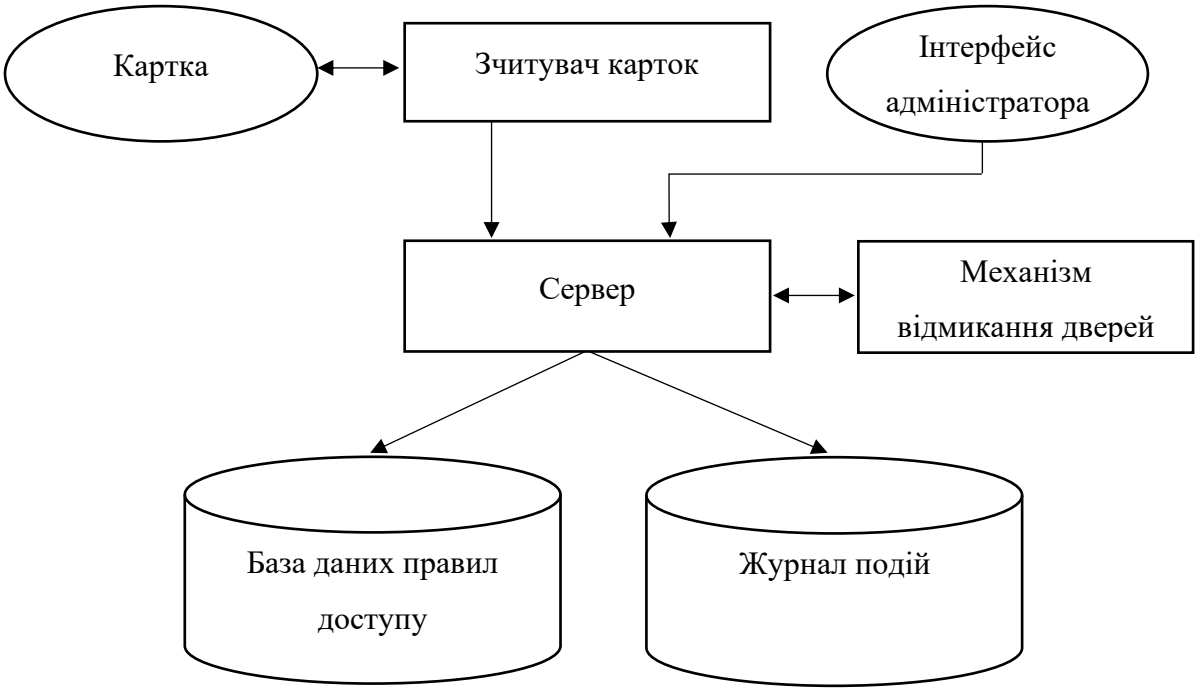


Рисунок 2.1 – Узагальнена архітектура засобу контролю доступу

У запропонованій архітектурі системи контролю доступу взаємодія між основними компонентами побудована таким чином, щоб забезпечити ефективну, надійну та гнучку перевірку прав доступу користувачів. Взаємодія починається з користувача, який за допомогою RFID-картки або іншого засобу ідентифікації звертається до точки входу (зчитувача). Цей зчитувач передає UID картки у серверну логіку.

У серверному модулі відбувається кілька ключових перевірок: автентифікація UID, визначення ролі користувача, перевірка прав доступу відповідно до зони та часових обмежень. Для цього сервер звертається до бази даних, де зберігається інформація про користувачів, їхні ролі, дозволені зони, доступи й часові правила.

У разі позитивного рішення сервер надсилає сигнал на розблокування точки доступу. Паралельно, незалежно від результату (успішний чи відмовлений доступ), у журналі подій фіксується запис про дію: час, користувач, зона, результат.

Окрему роль у системі відіграє адміністратор, який має повний доступ до керування користувачами, призначення ролей, налаштування зчитувачів та створення/видалення тимчасових правил доступу. Він також має змогу переглядати журнал подій для моніторингу безпеки та прийняття управлінських рішень.

Таким чином, архітектура засобу складається з логічно пов’язаних модулів: користувацького рівня, зчитування даних, обробки запитів на сервері, централізованого зберігання інформації в базі даних, системи логування та керування адміністратором.

2.4 Блок автентифікації користувачів

Автентифікація — це ключовий етап у системі контролю доступу, який дозволяє перевірити, чи є особа, що намагається отримати доступ, дійсно тією, за кого себе видає. У нашій системі автентифікація реалізується на основі RFID-карток, які прив’язані до конкретного користувача в базі даних.

Для забезпечення надійного контролю доступу в системі розроблено спеціалізований блок автентифікації користувачів. Його головна мета —

перевірити особу користувача, оцінити відповідність його прав доступу до запитуваної зони, а також врахувати часові обмеження, якщо такі встановлені. Робота блоку базується на обробці унікального ідентифікатора, отриманого з RFID-картки, і на порівнянні цього ідентифікатора з даними в базі. Нижче наведена узагальнена схема, яка демонструє основні етапи функціонування блоку автентифікації та прийняття рішення про надання або відмову в доступі. На рисунку 2.2, наведено узагальнену схему автентифікації користувачів.

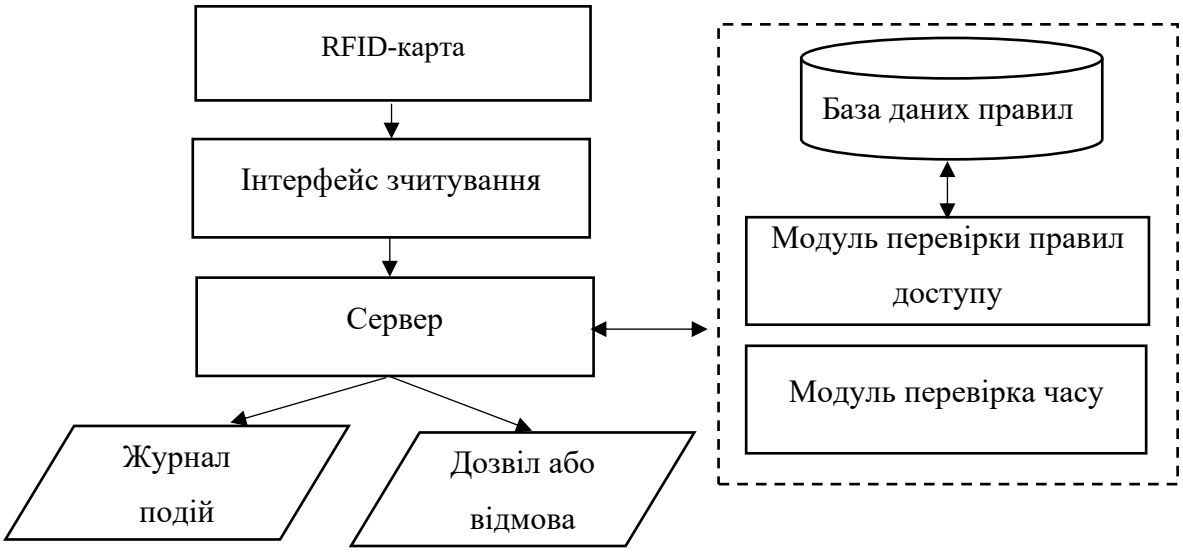


Рисунок 2.2 – Архітектура блоку автентифікації користувачів

Користувач є ініціатором взаємодії із системою. Для проходження автентифікації він використовує RFID-карту, яка містить унікальний ідентифікатор (UID), прив’язаний до облікового запису в базі даних.

Далі інформація з карти потрапляє до інтерфейсу зчитування, яким виступає RFID-картридер, підключений до комп’ютера або мікроконтролера. Саме через цей інтерфейс UID передається на обробку сервером або центральним програмним модулем.

Сервер (або локальна обчислювальна система) виконує ключові функції з обробки запитів. Він отримує ідентифікатор карти, звіряє його з базою даних користувачів, а також здійснює перевірку додаткових параметрів, таких як правила доступу і поточний час. Ці дії представлені модулями:

- перевірка правил доступу — визначає, чи має користувач право входу в конкретну зону або доступ до певного ресурсу;
- перевірка часу — перевіряє, чи відповідає запитаний час доступу встановленим обмеженням (наприклад, дозвіл лише у робочі години).

У результаті обробки запиту система приймає рішення — дозволити або відмовити у доступі. Це рішення надсилається до виконавчого пристрою (наприклад, реле, яке відкриває турнікет або двері).

Незалежно від результату, інформація про спробу входу, включаючи UID, час, статус перевірки та результат (дозвіл або відмова), записується у журнал подій. Такий підхід дозволяє здійснювати аудит подій та відстеження активності користувачів у системі.

Таким чином, узагальнена архітектура демонструє логічну послідовність дій та компоненти, необхідні для ефективної реалізації процесу автентифікації з урахуванням політик безпеки.

Цей блок забезпечує перевірку ідентифікатора картки (UID), визначає, чи активна вона в поточний момент, чи дозволений доступ до конкретної зони, а також виконує логування кожного запиту (таблиця 2.5).

Таблиця 2.5 – Етапи автентифікації користувачів

Етап	Опис
Зчитування UID	З RFID-картки зчитується унікальний ідентифікатор.
Перевірка в базі даних	Система шукає UID у таблиці користувачів або access_tokens.
Часова перевірка	Перевіряється, чи не закінчився термін дії картки (30 хв або заданий інтервал).
Перевірка зони доступу	Порівнюється UID картки із дозволами на поточний зчитувач (зону).
Надання або відмова в доступі	Якщо всі перевірки пройдені — активується доступ (реле, LED), інакше — відмова.
Журналювання події	Фіксація результату в лог (час, користувач, зона, результат).

Блок автентифікації користувачів виконує ключову функцію у забезпеченні контролю доступу, перевіряючи унікальні ідентифікатори карток, відповідність користувача дозволеним зонам, а також дотримання часових обмежень. Він забезпечує прийняття рішення про дозвіл або відмову у доступі на основі визначених правил, що дозволяє ефективно керувати безпековими сценаріями.

2.5 Обґрунтування вибору складових засобу

У даному проєкті для реалізації процесу автентифікації користувачів було обрано технологію RFID, що забезпечує безконтактну ідентифікацію за допомогою радіохвиль. Основними причинами такого вибору є зручність використання, швидкість зчитування даних, а також широке розповсюдження відповідного обладнання. На відміну від паролів, які легко піддаються компрометації, або біометричних систем, які потребують складних сенсорів і налаштувань, RFID дозволяє побудувати доступний і надійний засіб автентифікації. Крім того, використання RFID-карток дозволяє реалізувати гнучку систему контролю доступу з можливістю швидкої заміни носія та його повторного програмування.

Для забезпечення ефективної автентифікації користувачів необхідно обрати відповідний апаратний модуль зчитування RFID-карт. Основними критеріями вибору виступають тип інтерфейсу, сумісність із програмним забезпеченням, підтримувані стандарти карт, а також вартість пристрою.

Окрім вибору апаратного забезпечення, важливо також враховувати безпекові аспекти використання RFID-технологій. Зокрема, необхідно зменшити ризики несанкціонованого зчитування або клонування карток шляхом використання сучасних стандартів або захищені канали обміну даними. Також доцільним є впровадження багаторівневої системи автентифікації, яка поєднує RFID з іншими методами — наприклад, PIN-кодами або тимчасовими обмеженнями доступу. Такий підхід дозволяє значно підвищити загальний рівень інформаційної та фізичної безпеки об’єкта.

У табличці 2.6 наведено порівняння трьох популярних моделей RFID-картридерів та відповідні до них карток.

Таблиця 2.6 – Порівняльний аналіз моделей картридерів

№	Модель рідера	Ціна (грн)	Тип карток	Модель картки	Ціна картки (грн)	Особливості
1	RFID id R20D [15]	~420	EM4100 / TK4100	EM4001	~10–15	Plug and play, вводить код як клавіатура
2	ZKTeco ProID102 [16]	~2300	EM4100	Em-Marine, Mifare	~12	Металевий корпус, стабільна робота
3	ACR122U (режим RFID) [17]	~2000	MIFARE Classic (тільки RFID-режим)	MIFARE 1K	~25	Підтримка MIFARE 1K через RFID

Представлена таблиця порівнює три моделі рідерів RFID-карток за ціною, сумісністю з типами карток та особливостями. Найдорожчим є ACR122U, який підтримує MIFARE 1K — картки з більшою пам’яттю, але працює виключно в RFID-режимі. ZKTeco ProID102 вирізняється металевим корпусом і стабільністю, працює з недорогими картками EM4100**.** Найдешевшим варіантом є RUIDO R20D, який підтримує EM4100 / TK4100 і працює як клавіатура (plug and play), що спрощує інтеграцію. Таким чином, вибір моделі залежить від бюджету, потреб у типі карток і вимог до надійності або простоти підключення.

2.4 Висновки з розділу

У другому розділі було детально розглянуто методику контролю доступу до приміщень, побудовано модель розмежування прав користувачів, розроблено структуру бази даних для управління доступом, описано методику призначення прав і механізми автентифікації. Також було обґрунтовано вибір програмних і технічних складових, необхідних для реалізації системи, визначено категорії користувачів і рівні їх доступу, а також представлено архітектуру блоку перевірки

прав доступу на вхід до приміщення. Усі ці елементи формують цілісну архітектуру системи доступу, що відповідає вимогам безпеки, гнучкості та ефективності.

Розроблена система дозволяє реалізувати гнучке управління доступом до приміщень, поєднуючи декілька важливих підходів — розмежування за ролями, часові обмеження доступу та централізоване адміністрування. Кожен користувач отримує права відповідно до своєї ролі (наприклад, працівник, охоронець або технічний персонал), а можливість задавати часові інтервали дозволяє точно контролювати, коли саме дозволено вхід. Взаємодія всіх елементів — програмної частини, бази даних і зчитувачів — забезпечує цілісність функціонування системи.

Запропонована система також відзначається високою адаптивністю до змін у структурі підприємства чи організації. У разі розширення штату, відкриття нових приміщень або зміни політики безпеки, адміністратор може оперативно змінювати параметри доступу без необхідності значної перебудови системи. Такий підхід мінімізує витрати часу й ресурсів на технічне обслуговування та дозволяє зберігати актуальність системи у динамічних умовах роботи.

Хоча запропоноване рішення не гарантує абсолютного захисту від усіх потенційних загроз, воно створює надійну основу для впровадження сучасних механізмів контролю доступу. Адміністратор системи може оперативно оновлювати налаштування, аналізувати журнали подій і своєчасно виявляти порушення. Це дозволяє не лише підвищити загальний рівень безпеки, а й адаптувати систему до змін у структурі персоналу чи режимі роботи установи.

3 АЛГОРИТМ РОБОТИ ЗАСОБУ КОНТРОЛЮ ДОСТУПУ

3.1 Узагальнений алгоритм роботи засобу

Для забезпечення ефективної та надійної роботи системи контролю доступу необхідно чітко визначити загальний алгоритм її функціонування. Такий алгоритм описує послідовність дій від моменту ініціації запиту на доступ до фіксації результату в системному журналі. Упорядкованість процесів дозволяє легко масштабувати систему, забезпечити стабільну роботу з різними типами користувачів і забезпечити централізований контроль над усіма точками входу.

Робота системи контролю доступу реалізується у вигляді послідовних етапів, що забезпечують перевірку прав користувача та відповідну реакцію системи.

Етап 1 ініціація доступу. Користувач підносить RFID-картку до зчитувача. Пристрій зчитує унікальний ідентифікатор (UID) та передає його на обробку до центрального модуля (наприклад, сервер або локальний контролер).

Етап 2 передача та обробка даних. UID картки передається на сервер, де виконується звернення до бази даних для перевірки зареєстрованих користувачів і їхніх прав доступу.

Етап 3 перевірка прав доступу.

Сервер перевіряє:

- Чи існує UID у базі зареєстрованих користувачів;
- Чи має користувач право доступу до цієї конкретної зони;
- Чи поточний час входить у дозволений інтервал доступу;
- Чи не є картка заблокованою.

Етап 4 прийняття рішення.

На основі перевірки формується рішення:

- Якщо умови виконано — дозволити доступ;
- Якщо ні — відхилити запит.

Етап 5 керування виконавчим пристроєм. У разі позитивного рішення сервер/контролер подає сигнал на виконавчий механізм (наприклад, електрозамок), який відкриває двері. Інакше — двері залишаються зачиненими.

Етап 6 логування події. Система фіксує спробу доступу в журналі подій із зазначенням UID, дати, часу, результату перевірки, точки входу та інших параметрів.

Етап 7 завершення сесії. Після виконання дій система переходить у стан очікування наступного запиту.

На рисунку 3.1 відображено логічну послідовність етапів взаємодії користувача із системою.

Рисунок 3.1 – Узагальнений алгоритм роботи програми

Таким чином, узагальнений алгоритм роботи засобу контролю доступу демонструє логічну і структуровану послідовність етапів взаємодії користувача із

системою — від моменту ініціації запиту до прийняття рішення про дозвіл або заборону доступу. Такий підхід забезпечує не лише базову функціональність, а й гнучкість у керуванні правами доступу

3.2 Алгоритм реєстрування користувачів

У процесі реєстрації користувача система перевіряє наявність облікового запису та, залежно від результату, виконує відповідні дії. Алгоритм забезпечує як створення нових записів, так і оновлення існуючих прав доступу, гарантуючи правильну інтеграцію користувача в систему контролю.

На рисунку 3.2 зображено основні етапи алгоритму реєстрування користувачів:

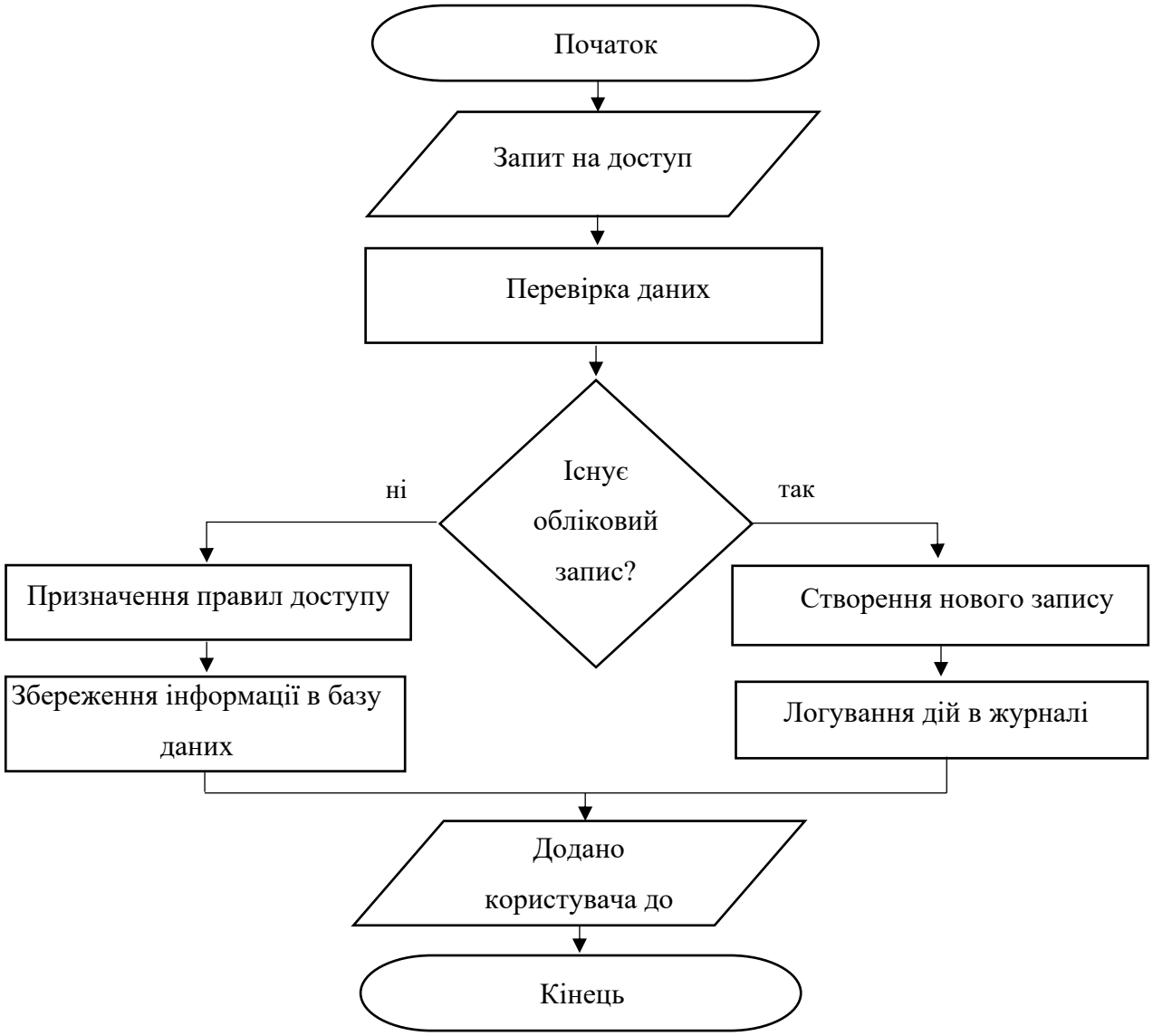


Рисунок 3.2 – Алгоритм реєстрації користувачів

Етап 1 запит на доступ. Користувач ініціює запит на реєстрацію або ідентифікацію в системі.

Етап 2 перевірка даних. Система аналізує введені або зчитані дані для визначення, чи існує обліковий запис користувача.

Етап 3 перевірка наявності облікового запису. Відбувається перевірка бази даних на предмет існуючого запису. Далі можливі два варіанти розвитку подій.

Якщо обліковий запис вже існує:

- створення нового запису – актуалізація даних або їх уточнення;
- логування дій у журналі – фіксація реєстраційної події в журналі аудиту.

Якщо обліковий запис відсутній:

- призначення правил доступу – визначаються рівні та умови доступу для нового користувача.
- збереження інформації в базу даних – збереження облікового запису з правами доступу.

Етап 4 додавання користувача до системи. Завершальний етап реєстрації, після якого користувач вважається повноправним суб’єктом СКД.

Алгоритм реєстрації користувачів дозволяє гнучко реагувати як на появу нових користувачів, так і на зміни у вже зареєстрованих облікових записах. Вбудоване логування дій забезпечує аудит, а автоматизоване призначення прав – контрольований і безпечний процес управління доступом.

3.3 Алгоритм редагування прав доступу

Редагування прав доступу є ключовою функцією системи контролю доступу, яка дозволяє змінювати рівень повноважень користувача відповідно до зміни його обов’язків або ролі. Такий процес повинен бути захищеним, гнучким і мати можливість зберігати історію змін для подальшого аудиту. У результаті адміністратор може оперативно призначати або скасовувати доступ до певних зон або ресурсів без необхідності повторної реєстрації користувача.

На рисунку 3.3 зображено основні етапи алгоритму редагування прав доступу користувачів:

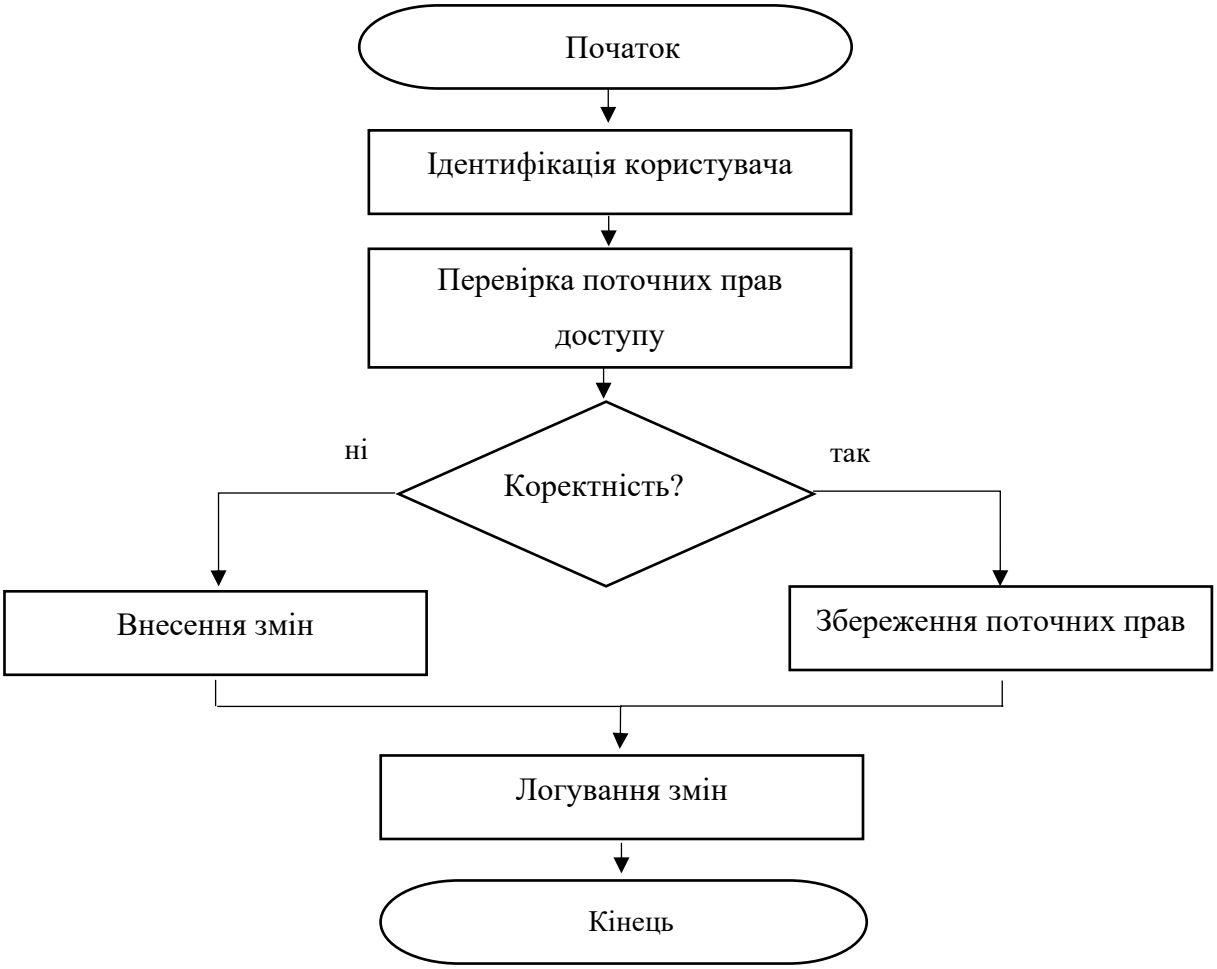


Рисунок 3.3 – Алгоритм редагування прав доступу

Етап 1 Ініціація редагування. Адміністратор запускає процес зміни прав через інтерфейс системи.

Етап 2 Ідентифікація користувача. Система запитує або автоматично отримує унікальний ідентифікатор користувача, права якого підлягають редагуванню.

Етап 3 Завантаження поточних прав доступу. Система витягує з бази даних поточні параметри доступу для вибраного користувача.

Етап 4 Внесення змін. Адміністратор визначає нові права доступу: це можуть бути як нові зони, час доступу, так і обмеження.

Етап 5 Перевірка коректності. Перед збереженням змін система проводить перевірку на конфлікти або недопустимі комбінації прав.

Етап 6 Збереження змін. Оновлені дані зберігаються в базу даних.

Етап 7 Логування змін. Усі зміни фіксуються в журналі подій, з вказанням часу, автора змін і попередніх параметрів доступу.

Етап 8 Завершення процедури. Після підтвердження змін користувач автоматично отримує нові параметри доступу при наступній автентифікації.

Редагування прав доступу в системі реалізовано з дотриманням принципів надійності, безпеки та простоти використання. Завдяки журналюванню змін забезпечується прозорість і підзвітність дій адміністратора, що є критично важливим у корпоративному середовищі.

3.4 Алгоритм автентифікації користувачів

Автентифікація є ключовим етапом у забезпеченні безпеки системи контролю доступу. Вона передбачає перевірку достовірності облікових даних користувача (наприклад, ідентифікатора або токена), які передаються при спробі отримати доступ до захищених об’єктів. Основна мета — переконатися, що суб'єкт доступу є тим, за кого себе видає. Алгоритм автентифікації гарантує, що доступ надається лише зареєстрованим і перевіреним користувачам.

Подана схема демонструє типовий процес автентифікації користувача в системі контролю доступу. Вона охоплює всі основні етапи взаємодії між користувачем і системою під час спроби отримати доступ до об'єкта.

Етапи алгоритму:

Етап 1 Початок процесу. Користувач ініціює запит на доступ — наприклад, за допомогою прикладання RFID-картки.

Етап 2 Введення автентифікаційних даних. На цьому етапі користувач вводить свої облікові дані: ідентифікатор (ID) та пароль, або ж здійснюється автоматичне зчитування даних з RFID-картки/мітки.

Етап 3 Перевірка даних у базі. Отримані дані передаються до сервера, де система звіряє їх із записами в базі даних зареєстрованих користувачів. Перевіряється:

- відповідність ідентифікатора;
- відповідність паролю/ключа;
- статус користувача (активний/заблокований).

Етап 4 Рішення системи:

- Якщо дані вірні — користувачу надається доступ до відповідної зони.
- Якщо дані не збігаються — система відмовляє у доступі та реєструє інцидент у журналі подій.

Етап 5 Завершення процесу. Система повертається до очікування наступного запиту на автентифікацію.

На рисунку 3.4 представлено цей алгоритм.

Рисунок 3.4 – Алгоритм автентифікації користувачів

Реалізований алгоритм автентифікації забезпечує надійний механізм перевірки особи користувача та його прав доступу. Завдяки використанню унікального ідентифікатора та централізованої перевірки на сервері, система оперативно приймає рішення про надання чи відмову у доступі. Важливою

перевагою є фіксація всіх подій у журналі, що дозволяє проводити аудит та швидко виявляти порушення безпеки. Такий підхід дозволяє зберігати баланс між зручністю для користувача та вимогами до кіберзахисту.

3.5 Обґрунтування вибору засобів розробки

У процесі створення програмного забезпечення для системи контролю доступу важливим етапом є вибір мови програмування, яка дозволить ефективно реалізувати усі необхідні функції — від взаємодії з апаратними засобами до обробки даних користувача, перевірки прав доступу та збереження інформації у базі даних. Вибір мови залежить від багатьох факторів: складності реалізації, швидкодії, наявності бібліотек для роботи з базами даних, простоти підтримки та розширення коду. У рамках цього проєкту було розглянуто три найбільш поширені мови програмування — C, C++ та Python — з метою порівняння їх можливостей та визначення найдоцільнішої для поставлених задач.

Мова програмування C++ є однією з найпоширеніших у світі, і активно застосовується для створення програмного забезпечення різного рівня складності. Вона поєднує у собі високу швидкодію, гнучкість і можливості об’єктно-орієнтованого програмування, що робить її придатною як для низькорівневого коду, так і для великих проектів. Завдяки сумісності з мовою C та широкому набору інструментів, C++ залишається актуальною для реалізації системного програмного забезпечення, драйверів, ігор, додатків реального часу тощо.

Серед відомих програмних продуктів, створених за допомогою C++, можна виділити такі як, браузер Mozilla Firefox, різноманітні компоненти операційної системи Windows, а також інструменти Microsoft Office і Visual Studio. Багато професійного мультимедійного софту, зокрема Adobe Photoshop, Illustrator і Premiere Pro, також написані на C++. Крім того, C++ використовується у розробці ігрових рушіїв, зокрема в основі популярного Unity [18].

Мова програмування C є універсальним інструментом для розробки програмного забезпечення та залишається актуальною протягом десятиліть. Вона належить до мов високого рівня, але водночас дозволяє працювати з

низькорівневими ресурсами, що забезпечує гнучкість та ефективність у використанні системних можливостей. Завдяки цьому С активно використовується в розробці драйверів, прошивок, операційних систем та різноманітного системного ПЗ.

С надає розробникам глибокий контроль над пам’яттю та апаратними ресурсами, що особливо важливо при створенні продуктивних та стабільних програм. Саме з використанням цієї мови були створені такі важливі системи, як Windows і Linux, а також велика кількість програм, бібліотек і програмних інтерфейсів, які активно застосовуються у сучасній IT-інфраструктурі [19].

Python — це високорівнева мова програмування, яка посідає одне з провідних місць за популярністю у світі. Її активно використовують для створення вебзастосунків, програмного забезпечення, а також у сферах машинного навчання та аналізу даних. Завдяки простому синтаксису, широким можливостям і підтримці різних платформ, Python став інструментом вибору для багатьох відомих компаній, таких як Google, NASA, IBM, Facebook, Dropbox та Netflix.

Python є об’єктноорієнтованою мовою, що дозволяє структурувати програмний код у вигляді модулів та пакетів. Його логічний та зрозумілий синтаксис робить код доступним для читання та підтримки, а вбудовані механізми дозволяють легко працювати з об'єктами, аналізувати їхню структуру та типи під час виконання програми. Мова підтримує масштабування, що дозволяє будувати як прості скрипти, так і складні, розширювані системи.

Попри численні переваги, Python має й деякі обмеження. Основним недоліком є відносно низька швидкість виконання коду, що може бути критичним для ресурсомістких задач. Також програми на Python можуть споживати більше пам’яті порівняно з аналогами, а деякі компоненти залежать від системних бібліотек. Проте завдяки великій і активній спільноті розробників більшість проблем вирішується швидко, що робить Python надзвичайно гнучким інструментом у багатьох сферах програмування [20].

Щоб обґрунтувати вибір мови програмування для реалізації проєкту, було проведено порівняння трьох популярних мов: C, C++ та Python. Кожна з них має

свої особливості, сильні сторони та обмеження, що безпосередньо впливають на процес розробки, підтримки й масштабування програмного забезпечення. У таблиці 3.1 наведено ключові переваги та недоліки кожної з мов, що дозволяє наочно оцінити їхню придатність до задач системного програмування.

Таблиця 3.1 – Порівняльний аналіз мов програмування

Мова	Переваги	Недоліки
C	- висока швидкість виконання - прямий доступ до пам’яті - портативність - використовується для створення ОС, драйверів	- важко читати та підтримувати код - відсутність ООП - високий ризик помилок через ручне керування пам’яттю
C++	- підтримка об’єктноорієнтованого програмування - швидкість, сумісність із C - використовується в ігровій індустрії, системному ПЗ, великомасштабних проєктах	- складність синтаксису - довгий цикл розробки - може бути важко відлагоджувати
Python	- простий та зрозумілий синтаксис - швидка розробка - велика кількість бібліотек - мультиплатформність - потужна спільнота	- менша продуктивність - вища витрата ресурсів - залежність від інтерпретатора та бібліотек

На основі аналізу було обрано Python як основну мову програмування для розробки програмного забезпечення. Завдяки простоті синтаксису, широким функціональним можливостям та гнучкості Python ідеально підходить для реалізації логіки системи доступу, роботи з базою даних та журналами подій. Попри деякі недоліки, такі як нижча швидкодія порівняно з C або C++, Python забезпечує значно швидший та зручніший цикл розробки, що є критично важливим у рамках даного проєкту.

Для реалізації проєкту також було необхідно обрати зручну, функціональну та стабільну систему управління базами даних (СУБД). Серед найбільш відомих і використовуваних варіантів було розглянуто SQL Server (SSMS)**,** Oracle Database

та SQLite**.** Кожна з цих СУБД має власні особливості, які варто враховувати при виборі залежно від типу застосунку, вимог до безпеки, масштабованості та швидкодії. У таблиці 1.3 наведено основні переваги та недоліки цих систем.

SQL Server — це потужна реляційна система керування базами даних, яка забезпечує зберігання, обробку та захист даних у різних масштабах — від локальних серверів до хмарних рішень. Для ефективної роботи з нею використовується SQL Server Management Studio (SSMS) — інтегроване середовище, що дозволяє керувати інфраструктурою SQL, включаючи як локальні інсталяції SQL Server, так і хмарні сервіси на кшталт SQL Azure.

SSMS надає зручні інструменти для адміністрування, налаштування, моніторингу, доступу та розробки компонентів баз даних. Воно підтримує роботу з SQL Server, SQL Azure, керованими екземплярами Azure SQL, SQL Server на віртуальних машинах Azure, а також із платформою аналітики Azure Synapse Analytics. Завдяки своїй гнучкості та функціональності SSMS є ефективним інструментом як для розробників, так і для адміністраторів баз даних [21].

Oracle Database — це одне з провідних комерційних рішень у сфері баз даних для підприємств, яке працює як система керування реляційними базами даних (RDBMS). Вона організовує дані у вигляді пов'язаних таблиць і стовпців, а також підтримує об'єктно-орієнтоване зберігання, що дозволяє зберігати структури даних разом із їхніми зв’язками. Oracle відзначається високим рівнем налаштовуваності, масштабованості та сумісності з різними операційними системами, зокрема Windows і macOS. Продукт доступний у п’яти редакціях: Enterprise, Standard, Personal, Express і Lite, що дозволяє обрати рішення відповідно до конкретних потреб користувача або компанії.

Остання стабільна версія — Oracle 19c (де «c» означає «cloud»), являє собою перше об'єднане рішення, яке підтримує кілька моделей даних і типів навантажень. Це дозволяє працювати з мультимедійним вмістом, текстовими даними, геопросторовими структурами тощо без необхідності створення окремих баз даних. Завдяки підтримці аналітики в режимі реального часу Oracle спрощує керування даними, забезпечує швидкий доступ і ефективну обробку інформації. Отримання

даних здійснюється за допомогою стандартної мови SQL, що робить роботу з системою зручною та уніфікованою для більшості спеціалістів [22].

SQLite — це легка вбудована реляційна база даних з відкритим вихідним кодом, яка здобула популярність завдяки своїй простоті й ефективності. На відміну від серверних рішень, таких як MySQL чи PostgreSQL, SQLite не потребує встановлення окремого серверу, оскільки є бібліотекою, яку безпосередньо інтегрують у програму. Це робить її зручною для використання в мобільних додатках, вбудованих системах та невеликих проєктах, де необхідне локальне зберігання даних. Вона є однією з найвикористовуваніших СУБД у світі та має нагороду Google-O'Reilly Open Source Awards [23].

У таблиці 3.2 наведено ключові переваги та недоліки трьох популярних СУБД — SQL Server Management Studio (SSMS)**,** Oracle Database та SQLite.

Таблиця 3.2 – Порівняльний аналіз систем управління баз даних

СУБД	Переваги	Недоліки
SSMS	Інтегроване середовище для адміністрування SQL Server; підтримка Azure; зручний GUI; надійність	Потребує встановлення SQL Server; високе споживання ресурсів.
Oracle	Масштабованість; підтримка хмарних технологій; багатомодельна структура; потужна аналітика	Висока вартість ліцензії; складність налаштування та адміністрування
SQLite	Легка вага; не потребує встановлення серверу; проста інтеграція; open-source	Не підходить для великих систем; обмежена багатокористувацька підтримка

Проаналізувавши основні характеристики трьох систем, обрано SQL Server Management Studio (SSMS) як оптимальне рішення. Воно забезпечує широку функціональність, зручне керування, підтримку масштабування та інтеграцію з хмарними сервісами, що ідеально підходить для сучасних задач адміністрування баз даних.

3.6 Розробка програми

Основні компоненти системи контролю доступу реалізовані з урахуванням безпеки та зручності використання. Вони включають механізми реєстрації користувачів, управління їхніми правами, а також контроль доступу на основі часових параметрів і блокувань. Для збереження конфіденційності даних застосовано паролі, а також ведеться журналювання дій для подальшого аналізу та виявлення потенційних загроз.

Імпортовані бібліотеки:

- бібліотека pyodbc є обов'язковою для встановлення зв'язку та виконання операцій (запитів) з SQL Server через ODBC драйвер. Вона дозволяє програмі на Python "спілкуватися" з базою даних;
- використовується datetime для роботи з датами та часом. Це критично для генерації міток часу для логів, зберігання та перевірки часових обмежень доступу користувачів (AccessStart, AccessEnd) та розрахунку тривалості блокування (timedelta);
- модуль os операційної системи, що дозволяє взаємодіяти з файловою системою. У даному випадку використовується для формування повного шляху до лог-файлу та створення каталогу для логів, якщо він не існує (os.makedirs);
- криптографічна бібліотека bcrypt, призначена для безпечного зберігання паролів. Її використання є ключовим елементом безпеки системи, оскільки вона дозволяє зберігати паролі в базі даних.

Модуль реєстрації нового користувача є важливою складовою системи контролю доступу, що дозволяє адміністратору додавати нових співробітників або об'єктів у базу даних. Ця функція відповідає за збір усієї необхідної інформації про нового користувача та безпечне її збереження у таблиці бази даних. Програма перевіряє чи є користувач у базі даних

Одним з найважливіших аспектів безпеки є обробка паролів. Система просить користувача ввести пароль. Після успішної валідації, пароль зберігається у системі.

Для гнучкого управління доступом, можна визначити часові інтервали, протягом яких дозволено вхід. Система запитує час початку та кінця доступу. Якщо ці поля залишаються пустими або введення є некоректним, відповідні значення в базі даних будуть встановлені, що означає відсутність часових обмежень для даного користувача.

Функція check_access є центральним елементом системи, відповідальним за верифікацію особи та надання або відмову в доступі на основі наявних правил.

На початку функція запитує унікальний ідентифікатор (UID) користувача. Перед виконанням будь-яких подальших перевірок, система оперативно звертається до тимчасового сховища failed_attempts, щоб визначити, чи не заблоковано вже даний UID через попередні невдалі спроби введення пароля. Якщо користувач заблокований і час блокування ще не минув, доступ негайно відхиляється

Якщо пароль виявився некоректним, лічильник невдалих спроб для цього UID збільшується. Коли кількість невдалих спроб досягає максимуму, користувач тимчасово блокується, і встановлюється час, до якого блокування буде активним.

Модуль редагування користувачів надає адміністратору можливість оновлювати інформацію про існуючих користувачів системи. Це дозволяє коригувати повне ім'я, рівень доступу, часові обмеження та, за потреби, змінювати пароль.

Функція (edit_user) починається із запиту UID користувача, дані якого потрібно змінити. Далі виконується запит до бази даних, щоб витягнути поточні дані цього користувача. Якщо користувача з введеним UID не знайдено, система виводить відповідне повідомлення та завершує операцію.

Користувачеві пропонується ввести нові значення для кожного поля. Якщо поле залишається пустим (користувач просто натискає Enter), то зберігається поточне значення цього поля. Це забезпечує гнучкість, дозволяючи змінювати лише необхідні параметри. Для часових полів передбачена можливість явно вказати "null", щоб скинути їх значення в базі даних.

Функція (update_user_password) окремо запитує, чи потрібно змінити пароль. Якщо користувач відповідає ствердно, програма вимагає ввести новий пароль.

Модуль видалення користувачів дозволяє адміністратору безпечно видаляти записи користувачів із системи. Ця функція включає додаткові кроки для запобігання випадковому видаленню та підтримки цілісності даних.

Функція (delete_user) починається із запиту унікального ідентифікатора (UID) користувача, якого потрібно видалити. Після ідентифікації користувача (витягується його повне ім'я для відображення), система просить явне підтвердження дії від адміністратора. Цей крок є критично важливим для запобігання випадковому або несанкціонованому видаленню даних.

Функція (log_event()) для забезпечення аудиту безпеки та моніторингу всіх значущих подій у системі контролю доступу. Вона відповідає за запис інформації про дії користувачів, спроби доступу, зміни конфігурації та системні помилки у спеціально призначений лог-файл.

Кожен запис логу починається з точної мітки часу, що генерується в момент виникнення події. Це дозволяє хронологічно відстежувати всі дії. Повідомлення про подію, передане функції, об'єднується з міткою часу для створення повного запису логу. Програмний код має модульну структуру, що забезпечує зручність у підтримці та подальшому розширенні. Повний код основної програми наведено в додатку А.

3.7 Висновки з розділу

У даному розділі було розроблено загальний алгоритм роботи засобу, що дозволяє сформувати цілісну картину функціонування системи. Описано ключові етапи — від запуску програми до обробки дій користувача, що забезпечує узгоджену логіку між усіма модулями. Завдяки цьому стало можливим забезпечити стабільність, логічну послідовність та передбачуваність поведінки системи.

Особливу увагу приділено алгоритмам реєстрації користувачів та редагування прав доступу. Усі дії в системі передбачають чітку структуру перевірок та обробки даних, що підвищує безпечність роботи. Реєстрація

виконується з урахуванням перевірки на унікальність, а редагування прав дозволяє гнучко змінювати ролі відповідно до потреб адміністрування.

Розглянуто й алгоритм автентифікації, що є ключовим для контролю доступу. Він забезпечує перевірку достовірності введених даних і запобігає несанкціонованому входу до системи. Для його реалізації враховано сучасні підходи до обробки паролів та зберігання інформації, що покращує загальний рівень безпеки.

На основі порівняння різних мов програмування та систем управління базами даних обґрунтовано вибір Python як основної мови розробки завдяки її простоті та ефективності, а також SSMS як середовища для зберігання та керування даними. У підсумку, всі етапи — від проектування до реалізації — дозволили створити функціональний програмний засіб з можливістю розширення та подальшої інтеграції.

Також важливо відзначити, що під час створення системи було враховано принцип модульності, що дозволяє легко масштабувати та модернізувати програму в майбутньому. Кожен з реалізованих алгоритмів виконує чітко визначену функцію, що спрощує підтримку та оновлення системи. Такий підхід забезпечує гнучкість розробки та адаптацію до змін у вимогах користувачів чи середовища експлуатації. Крім того, у процесі розробки особливу увагу було приділено безпеці та зручності користувача. Завдяки реалізації алгоритмів автентифікації та контролю прав доступу, система гарантує надійний захист даних та обмежує доступ до функцій відповідно до ролі користувача.

4 ТЕСТУВАННЯ ЗАСОБУ

4.1 Модульне тестування

Модульне тестування є фундаментальним етапом у життєвому циклі розробки програмного забезпечення, спрямованим на перевірку коректності функціонування окремих, найменших логічних частин коду – так званих "модулів" У контексті розробленої системи контролю доступу, модульне тестування дозволяє ізольовано оцінити працездатність ключових функцій та компонентів. Це забезпечує впевненість у тому, що кожен елемент системи виконує свої завдання правильно, незалежно від взаємодії з іншими модулями, що є критично важливим для надійності та безпеки системи контролю доступу.

Тестування проводилося за допомогою фреймворку «unittest» і було спрямоване на перевірку коректності функціонування ключових модулів програми.

На рисунку 4.1 зображено процес запуску модульного тестування в командному рядку. За допомогою певної команди, було виконано 6 тестів, що стосуються функції реєстрації нового користувача.

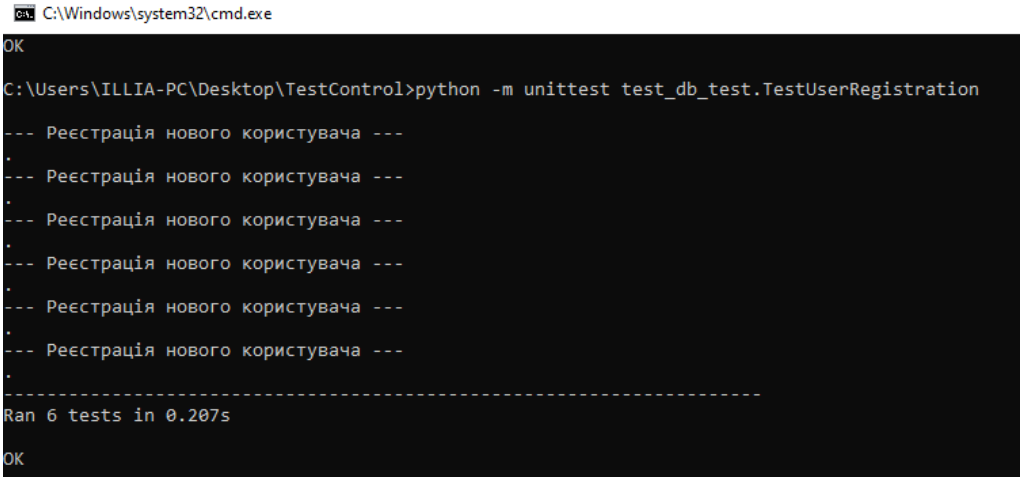


Рисунок 4.1 – Результат модульного тестування реєстрування користувачів

Перевіряється, чи функція реєстрації користувачів правильно обробляє різні сценарії введення даних та взаємодії з базою даних:

1. Тест успішної реєстрації. Тестується, чи система може коректно додати нового користувача до бази даних, коли надані всі необхідні та дійсні дані. Перевіряється, що дані зберігаються, а система підтверджує успішну операцію.

2. Тест реєстрації з дублюючим ідентифікатором. Тестується, чи система правильно виявляє спробу зареєструвати користувача з ідентифікатором, який вже існує. Перевіряється, що реєстрація відхиляється, і дані не дублюються в базі.

3. Тест реєстрації з невірним форматом рівня доступу. Перевіряється, як система реагує, коли рівень доступу вводиться у некоректному (наприклад, нечисловому) форматі. Очікується, що реєстрація буде відхилена.

4. Тест реєстрації з рівнем доступу поза діапазоном. Тестується, чи система відхиляє реєстрацію, якщо вказаний рівень доступу виходить за межі дозволених значень (наприклад, менше 1 або більше 3).

5. Тест реєстрації з невірним форматом часу початку доступу. Перевіряється, як система обробляє введення часу початку доступу у некоректному форматі. Очікується відхилення реєстрації.

6. Тест реєстрації з невірним форматом часу закінчення доступу. Аналогічно попередньому, тестується обробка некоректного формату часу закінчення доступу, що також має призвести до відхилення реєстрації.

Усі тести завершились успішно — про це свідчить повідомлення "Ran 6 tests in 0.207s" і "OK", тобто жодних помилок не виникло, і модуль працює коректно. Це означає, що реєстрація користувачів у системі функціонує стабільно.

На рисунку 4.2 зображено виконання ще одного етапу модульного тестування — цього разу тестується доступ користувачів до системи. Команда запускає клас, в якому реалізовані тести для перевірки правильності надання доступу на основі UID користувача та допустимого часового інтервалу.

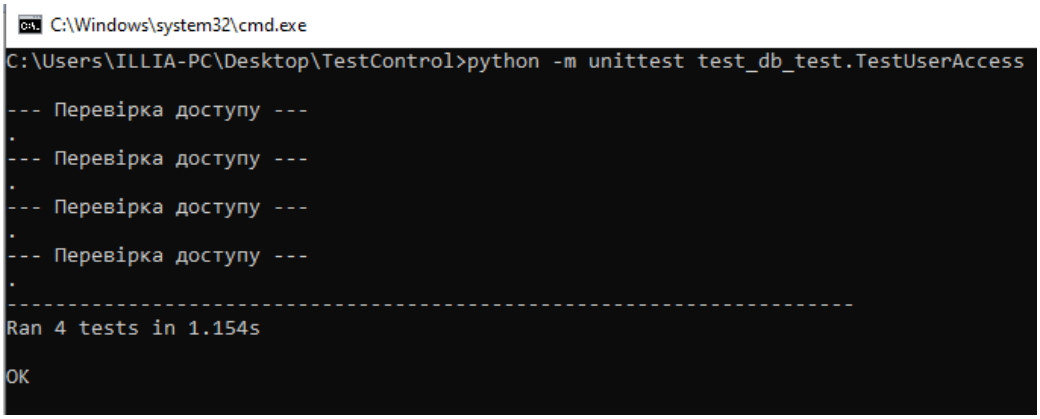


Рисунок 4.2 – Результат модульного тестування доступу користувачів до системи

У виводі видно повідомлення, що повторюється чотири рази — це означає, що було виконано 4 окремих тести. Кожен із них перевіряє різні сценарії:

- перевірка неіснуючого UID;
- доступ дозволений у межах робочого часу;
- доступ заборонений поза межами дозволеного часу;
- граничні умови (наприклад, рівно о 8:00 чи рівно о 17:00).

Результати показують, що всі тести завершилися успішно — система вивела «Ran 4 tests in 1.154s» та «OK», що свідчить про коректну роботу модуля перевірки доступу. Це підтверджує, що логіка перевірки часу входу й UID працює надійно та відповідає очікуваному функціоналу.

На рисунку 4.3 відображено процес модульного тестування механізму блокування користувача після трьох невдалих спроб введення пароля.

У процесі тестування імітується кілька спроб доступу до системи. Кожна з них супроводжується повідомленням, що вказує на виконання окремого тесту. Після трьох невдалих спроб вхід має бути заблоковано. Саме це і перевіряється: чи система правильно розпізнає кількість помилкових спроб і активує блокування відповідно до заданої логіки безпеки.

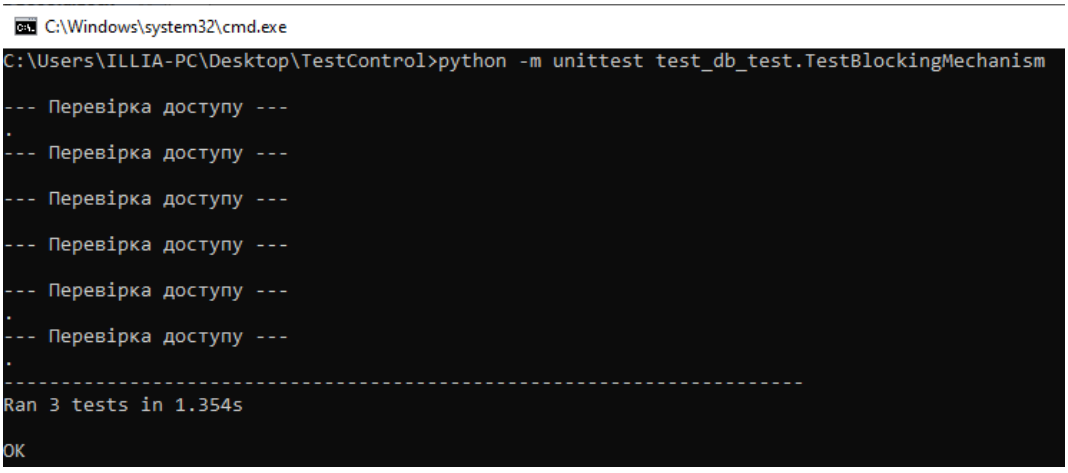


Рисунок 4.3 – Вигляд інтерфейсу блокування доступу після трьох помилкових спроб

У процесі тестування імітується кілька спроб доступу до системи. Кожна з них супроводжується повідомленням, що вказує на виконання окремого тесту. Після трьох невдалих спроб вхід має бути заблоковано. Саме це і перевіряється: чи

система правильно розпізнає кількість помилкових спроб і активує блокування відповідно до заданої логіки безпеки. Цей клас тестів зосереджений на перевірці логіки блокування користувачів після невдалих спроб входу.

1. Тест блокування користувача після кількох невдалих спроб. Перевіряється чи система коректно рахує невдалі спроби входу і чи блокує користувача, коли кількість цих спроб досягає встановленого ліміту. Також перевіряється, чи доступ відмовляється під час періоду блокування.

2. Тест скидання лічильника невдалих спроб при успішному вході. Перевіряється чи скидається лічильник невдалих спроб для користувача, якщо він успішно входить в систему після кількох (але не критичних для блокування) помилкових спроб.

3. Тест розблокування користувача після закінчення часу блокування. Перевіряється чи система дозволяє користувачеві знову спробувати ввійти в систему, коли період блокування, встановлений раніше, закінчився.

Підсумок виводиться у вигляді повідомлення «Ran 3 tests in 1.354s», що означає успішне проходження трьох тестових сценаріїв протягом 1.354 секунд.

На рисунку 4.4 проводиться етап тестування модуля керування користувачами в системі контролю доступу.

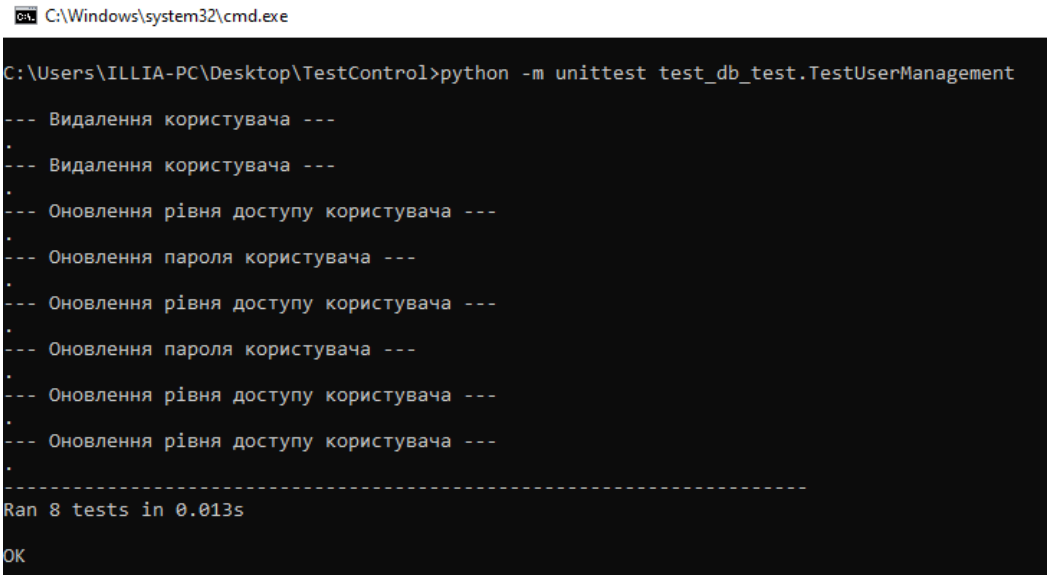


Рисунок 4.4 – Вигляд тестування керування правилами розмежування доступу

У процесі виконання тестів система проходить кілька сценаріїв:

- видалення користувача — двічі перевіряється, чи коректно видаляється користувач із бази;
- оновлення рівня доступу користувача — декілька разів перевіряється, чи правильно змінюється рівень доступу конкретного користувача;
- оновлення пароля користувача — окремо перевіряється, чи вдається успішно змінити пароль.

Кожна операція супроводжується повідомленням у консолі, що дозволяє легко простежити, яка саме дія виконується під час тесту.

Після завершення тестування виводиться результат, який означає, що всі вісім тестів були виконані успішно менш ніж за 0.013 секунди, без жодних помилок. Таким чином, можна зробити висновок, що всі функції керування користувачами реалізовані правильно й працюють стабільно.

На рисунку 4.5 виконуються два тести, що перевіряють, чи правильно виводиться список усіх користувачів із бази і чи містять дані правильні поля (UID, ім’я, рівень доступу тощо), або чи правильно працює фільтрація, якщо вона реалізована.

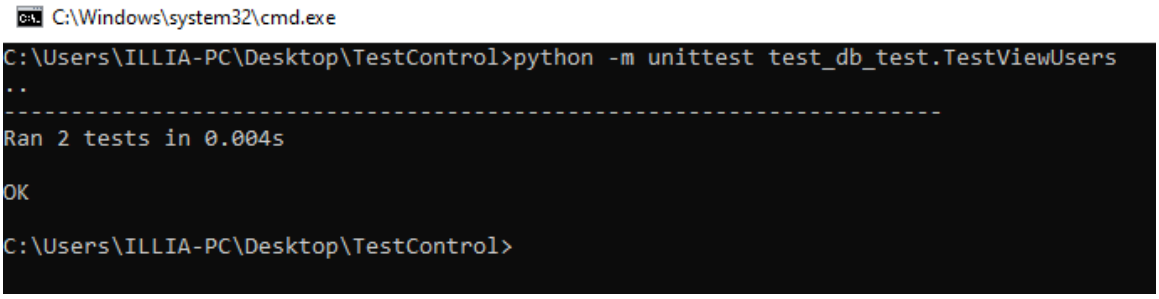


Рисунок 4.5 – Вигляд результатів юніт-тестування модуля перегляду користувачів

Цей клас тестів зосереджений на функціональності перегляду списку користувачів.

1. Тест перегляд списку користувачів, коли в базі даних немає жодного користувача. Перевіряється чи коректно функція відображає повідомлення про відсутність користувачів, коли таблиця Users порожня.

2. Тест перегляд списку користувачів, коли в базі даних є дані. Система перевіряє чи коректно функція витягує дані користувачів з бази даних та форматує їх для відображення на консолі.

Результат свідчить про те, що всі тести виконано успішно — отже, модуль перегляду користувачів працює коректно і видає очікувані результати. Це важливо, оскільки саме через цей функціонал адміністратор може перевіряти, хто має доступ до системи.

На рисунку 4.6 наведено результат тестування всієї програми, що підтверджує коректну та стабільну роботу всіх функцій.

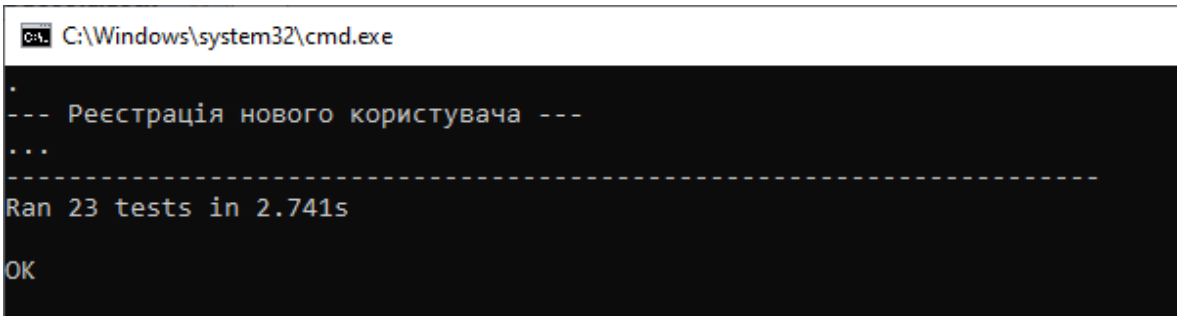


Рисунок 4.6 – Результати виконання повного набору юніт-тестів

Результати успішного проходження 23 тестів підтвердили коректність реалізації основних функціональних вимог. Код юніт-тестів наведено в додатку Б.

На рисунку 4.7 представлено фрагмент журналу подій (файл access_log.txt), який фіксує ключові дії користувачів у системі контролю доступу.

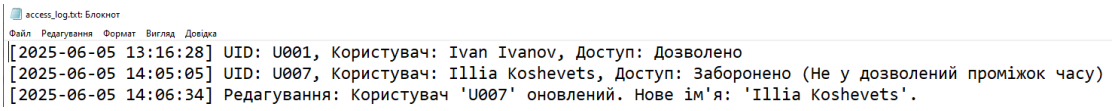


Рисунок 4.7 – Вигляд логування подій в журналі

Як видно з рисунку 4.7 кожен запис містить точну дату й час події, UID користувача, його ім’я, а також результат виконаної дії — наприклад, спробу входу або редагування облікового запису. Такий підхід до логування дозволяє не лише здійснювати моніторинг активності системи в реальному часі, але й забезпечує можливість аудиту дій користувачів у разі необхідності.

4.2 Статичне тестування безпеки

Статичне тестування безпеки застосунків (SAST) — це метод тестування безпеки, що використовується для перевірки вихідного, бінарного та байт-коду застосунку на наявність вразливостей. Це допомагає виявляти та усувати недоліки безпеки на ранніх стадіях розробки, знижуючи ризик експлуатації зловмисниками [24]. SAST-інструменти аналізують вихідний код, байт-код або бінарний код програми, не виконуючи її. Це дозволяє ідентифікувати широкий спектр потенційних проблем, таких як ін'єкції (SQL Injection, Command Injection), витік конфіденційних даних, небезпечне використання криптографії, відсутність належної валідації вхідних даних та помилки конфігурації.

Перевага SAST полягає в його здатності виявляти вразливості "глибоко" в коді, навіть у частинах, які можуть бути неактивними під час виконання програми або важкодоступними для динамічного тестування. Завдяки інтеграції SAST у процеси розробки та CI/CD (Continuous Integration/Continuous Delivery), команди можуть оперативно отримувати зворотний зв'язок про безпеку коду, що дозволяє виправляти проблеми набагато швидше та дешевше, ніж на пізніших етапах.

Після успішного запуску інструменту статичного аналізу безпеки Bandit щодо розробленого мовою Python-коду, було отримано такий звіт (рис. 4.8).

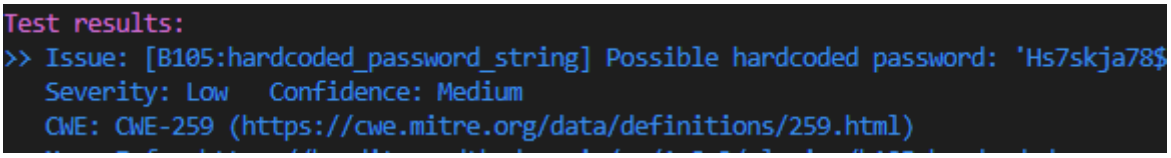


Рисунок 4.8 – Помилка про жорстке кодування паролів

Як видно на рисунку 4.8 Bandit проаналізував код і знайшла 2 можливі проблеми. Обидві проблеми стосуються того, що в коді “зашиті” паролі. Це називається “жорстке кодування паролів”. Перша і більш критична проблема полягає в тому, що пароль для підключення до бази даних був записаний безпосередньо в файлі db_test.py. Це становить серйозну загрозу безпеці, оскільки доступ до вихідного коду автоматично надає доступ до конфіденційних облікових даних бази даних.

Друга проблема стосується тестового пароля (рис. 4.9) захардкодженого у файлі test_db_test.py, що хоча й не є прямою загрозою для "бойової" системи, вказує на потенційно небезпечні практики кодування.

```
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'block_pass'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
```

Рисунок 4.9 – Помилка про тестовий пароль

Для усунення цих вразливостей, необхідно негайно прибрати пароль до бази даних з файлу db_test.py і замість цього використовувати безпечніші методи зберігання, такі як змінні середовища або окремі файли конфігурації, що не входять до системи контролю версій. Щодо тестового пароля, його можна або ігнорувати за допомогою спеціальної директиви (# nosec B105), або змінити на інше значення. Виявлення та виправлення цих проблем, хоча й мають низький рівень серйозності за класифікацією Bandit, є важливим кроком до підвищення загальної безпеки нашого програмного рішення та його стійкості до потенційних атак.

Результати статичного тестування безпеки (SAST), наведені на рисунку 4.10 представлено зведений звіт за результатами статичного тестування безпеки (SAST), виконаного за допомогою інструменту Bandit. Цей звіт надає загальну статистику щодо аналізу коду.

```
Code scanned:
        Total lines of code: 689
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 2
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 2
                High: 0
Files skipped (0):
```

Рисунок 4.10 – Результати статичного тестування безпеки

У розділі "Run metrics" видно, що було виявлено дві потенційні проблеми (Issues). Обидві проблеми класифіковані як такі, що мають низький рівень серйозності (Low Severity) та середній рівень впевненості (Medium Confidence).

Це означає, що програма-аналізатор виявила дві ділянки коду, які можуть містити вразливості, пов'язані з безпекою, і хоча вони не є критичними, їх варто перевірити. Ці дві проблеми стосувалися жорсткого кодування паролів у коді, що є поганою практикою безпеки.

Для усунення цієї загрози було реалізовано перенесення всіх конфіденційних облікових даних – серверу, бази даних, імені користувача та пароля – у зовнішній конфігураційний файл, названий protect.env. Цей файл розміщується окремо від основного коду програми і, що найважливіше, виключається зі системи контролю версій.

Таким чином, пароль та інші чутливі дані більше не зберігаються безпосередньо у вихідному коді, а завантажуються динамічно під час виконання програми за допомогою бібліотеки python-dotenv.

Повторне сканування за допомогою Bandit, рисунок 4.11, підтвердило успішне усунення виявленої вразливості, забезпечивши значне підвищення загального рівня безпеки розробленої системи контролю доступу.

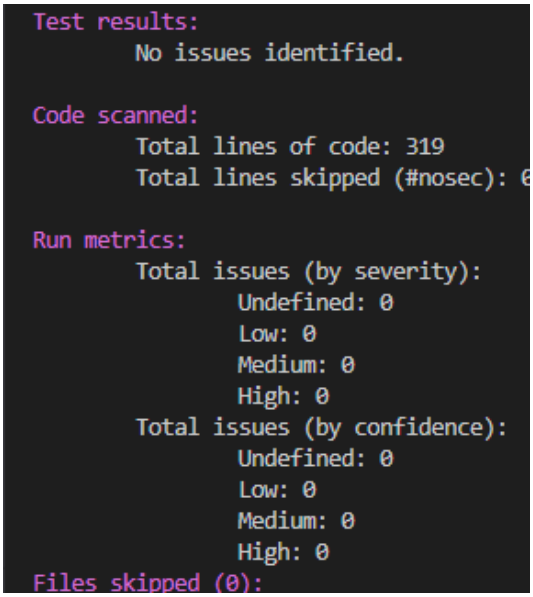


Рисунок 4.11 – Результати тестування після виправлення помилок

Для усунення цього ризику було впроваджено практику зберігання конфіденційних даних у зовнішньому конфігураційному файлі, який не є частиною основного репозиторію коду. Це значно підвищило рівень захищеності розробленого засобу контролю доступу. Таким чином, забезпечено не лише функціональність, а й надійний фундамент безпеки системи.

4.3 Висновки з розділу

Проведені тестування відіграли ключову роль у забезпеченні якості та надійності розробленої системи контролю доступу. Юніт-тести, організовані у кілька класів, дозволили перевірити кожен окремий компонент програми, починаючи від процесу реєстрації нових користувачів, перевірки доступу та функціоналу блокування, до механізмів управління користувачами (видалення, оновлення паролів та рівнів доступу) та перегляду списку користувачів. Загальний успішний прогін 23 тестів підтвердив, що всі основні функціональні вимоги до системи були реалізовані коректно, а логіка обробки даних та взаємодії з базою даних працює згідно з очікуваннями. Це створює міцну основу для подальшої розробки та розгортання, дозволяючи вчасно виявляти та виправляти помилки на етапі розробки.

Окрім функціональної перевірки, важливим етапом стало статичне тестування безпеки (SAST), проведене за допомогою інструменту Bandit. Цей аналіз дозволив ідентифікувати потенційні вразливості без необхідності запуску програми, що є критично важливим для раннього виявлення загроз у безпеці. SAST-сканування виявило дві проблеми, пов'язані з жорстким кодуванням паролів. Зокрема, пароль для підключення до бази даних був безпосередньо вбудований у код, що є серйозним недоліком безпеки, оскільки відкриває шлях для несанкціонованого доступу до конфіденційної інформації у разі компрометації вихідного коду. Хоча інша проблема стосувалася тестового пароля, обидва випадки підкреслюють необхідність дотримання кращих практик у роботі з чутливими даними.

Таким чином, комбіноване використання модульного та статичного тестування безпеки забезпечило комплексний підхід до оцінки якості та захищеності програмного забезпечення. Результати цих тестувань підтверджують функціональну працездатність системи та виявляють критичні точки для покращення її безпеки, що є основою для подальшої розробки надійного та захищеного рішення.

ВИСНОВКИ

Розроблена система контролю доступу до приміщень є відповіддю на значні обмеження відомих рішень, які часто страждають від відсутності гнучкості, часових обмежень та ефективного контролю над тимчасовими користувачами. Поглиблений аналіз цих недоліків привів до чіткого завдання: створити рішення, здатне динамічно обмежувати доступ за часом, зоною та кількістю використань, значно підвищуючи загальний рівень безпеки.

Архітектура системи була розроблена з урахуванням гнучкої методики контролю доступу, що включає модель розмежування прав користувачів та оптимізовану структуру бази даних. Ця архітектура дозволяє реалізувати комплексне управління доступом, поєднуючи розмежування за ролями, часові інтервали та централізоване адміністрування. На основі розробленої архітектурі було побудовано узагальнений алгоритм роботи засобу. На основі узагальненого алгоритму були розроблені більше детально алгоритми ключових процедур: алгоритм реєстрування користувачів, редагування прав доступу та їх автентифікації. Застосування принципу модульності у розробці гарантує легке масштабування та модернізацію системи в майбутньому. Було виконано обґрунтування засобів вибору розробки.

Коректність реалізації системи були підтверджені тестуванням. Модульне тестування успішно підтвердило коректність реалізації всіх ключових функціональних вимог, від реєстрації до управління користувачами. Водночас, статичне тестування безпеки (SAST) за допомогою інструменту Bandit виявило важливі аспекти для подальшого вдосконалення, зокрема необхідність усунення жорсткого кодування паролів. Ці результати є цінною основою для подальшого підвищення безпеки системи. У сукупності, розроблена система є засобом, що відповідає сучасним вимогам до безпеки, гнучкості та ефективності, а також має значний потенціал для подальшого розвитку та інтеграції до систем кібербезпеки об’єктів критичної інфраструктури.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кошевець І. А., Баришев Ю. В. Удосконалення систем контролю доступу до приміщення. LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025) Вінниця, 2025. 3 с. URL https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025 (дата звернення 27.05.2025)
2. Система контролю і управління доступом від А до Я. URL: https://deps.ua/ua/knowegable-base/reference-information/7824.html (дата звернення 06.04.2025).
3. Системи контролю доступу. URL: https://zktecoua.com/ua/solutions/skud/ (дата звернення 10.04.2025).
4. Як працює система контролю доступу (СКД або СКУД). URL: https://u-prox.systems/yak-praczyuye-sistema-kontrolyu-dostupu-skd-abo-skud/?utm_source=chatgpt.com (дата звернення 11.04.2025).
5. Контроль доступу: мережева система. URL: https://expert112.com.ua/kontrol-dostupa/index_ua.html (дата звернення 13.04.2025).
6. Система контролю доступу. Види та відмінності СКУД. Особливості вибору СКД. URL: https://revel.com.ua/info/articles/skud-vidy-i-otlichiya/?lang=ua (дата звернення 16.04.2025).
7. Універсальна система контролю доступу. URL: https://ukrinfosystems.com.ua/uk/design-and-construction/access-control-systems (дата звернення 19.04.2025).
8. Що таке автентифікація? URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-authentication (дата звернення 21.04.2025).
9. Аутентифікація, авторизація та ідентифікація: як не сплутати. URL: https://training.qatestlab.com/blog/technical-articles/authentication-authorization-and-identification/?utm_source=chatgpt.com (дата звернення 28.04.2025).

10. Що таке двофакторна автентифікація або 2FA? URL: https://www.dropbox.com/uk_UA/resources/what-is-2fa#:~:.(дата звернення 10.05.2025).
11. Що таке автентифікація? URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-authentication (дата звернення 14.05.2025).
12. Автентифікація (аутентифікація): що це, методи та особливості. URL: https://www.zen.com/ua/blog/personal-finance-uk/authentication-definition-methods-features/ (дата звернення 17.05.2025).
13. Understanding the confusing world of RFID tags and readers in access control. URL: https://www.nedapidentification.com/insights/understanding-the-confusing-world-of-rfid-tags-and-readers-in-access-control/ (access 23.05.2025).
14. Вільне суспільство моделі розмежування доступу. URL https://jak.bono.odessa.ua/articles/vilne-suspilstvo-modeli-rozmezhuvannja-dostupu.php (дата звернення 24.05.2025).
15. Usb rfid id РЧІД r20d-usb зчитувач карт em4001. URL https://rozetka.com.ua/ua/84958722/p84958722/ (дата звернення 26.05.2025).
16. EM-Marine і Mifare ZKTeco ProID102 (ID) Black. URL https://smart-home.com.ua/index.php?route=product/product&path=77_90&product_id=13769 (дата звернення 26.05.2025).
17. Зчитувач Mifare ACS ACR122U. URL https://idcard.com.ua/ua/schityvatel-beskontaktnykh-smart-kart-acr122u-nfc/?srsltid=afmbooo1kaf6s2024kpvrwvzprgeedgh1lar99s8bsytwt4xyx6jdkio (дата звернення 26.05.2025).
18. Уроки програмування на C++. URL https://acode.com.ua/uroki-po-cpp/ (дата звернення 02.06.2025).
19. Вивчити мову програмування C. URL https://w3schoolsua.github.io/c/index.html#gsc.tab=0 (дата звернення 02.06.2025).

20. Що таке мова програмування Python? URL https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovanija-python/ (дата звернення 02.06.2025).
21. У чому різниця між SQL Server і SQL Management Studio? URL https://bard.kultura.cx.ua/maysternist/u-chomu-riznicya-mizh-sql-server-i-sql-management-studio.html (дата звернення 03.06.2025).
22. What Is an Oracle Database? URL https://www.solarwinds.com/resources/it-glossary/oracle-database (access 03.06.2025).
23. Що таке SQlite? URL https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/ (дата звернення 04.06.2025).
24. Статичне тестування безпеки застосунків (Static Application Security Testing, SAST). URL https://www.vpnunlimited.com/ua/help/cybersecurity/sast?srsltid=AfmBOooIIbc-vwstD2COwJwYcuXigrS98l9_Py3k9_IY4Bv1YK4rlwtP (дата звернення 07.06.2025).
25. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 125 «Кібербезпека» освітньо-професійні програми «Безпека інформаційних і комунікаційних систем» та «Кібербезпека критичних систем» / Уклад. В. А. Каплун, О. П. Войтович. Вінниця: ВНТУ, 2023. 61 с.

ДОДАТКИ

Додаток А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Засіб контролю доступу до приміщення

Автор роботи: Кошевець Ілля Анатолійович

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ , група 1 БКС-21 б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism **0,81** %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

√ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Завідувач кафедри ЗІ д. т. н., проф. _____ Володимир ЛУЖЕЦЬКИЙ

Гарант освітньої програми «Кібербезпека критичних систем» к. т. н., доц.
_____ Юрій БАРИШЕВ

Особа, відповідальна за перевірку _____ Валентина КАПЛУН

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____

Здобувач _____ _____

Додаток Б. Текст програми

db_test.py

```
import pyodbc
from datetime import datetime, timedelta, time # Додано time до імпорту
import os
import bcrypt

server = 'ILIAPC\\SQLEXPRESS'
database = 'AccessControlDB'
username = 'sa'
password = 'Hs7skja78$ksjT'
driver = '{ODBC Driver 17 for SQL Server}' # Переконайтеся, що цей драйвер встановлено!

# --- Конфігурація ---
LOG_FILE = 'access_log.txt'
log_file_full_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), LOG_FILE)

MAX_FAILED_ATTEMPTS = 3
BLOCK_DURATION_SECONDS = 30 # Тривалість блокування в секундах

# Словник для відстеження невдалих спроб та часу блокування
failed_attempts = {}

# --- Функції безпеки ---
def hash_password(password):
    """Хешує пароль за допомогою bcrypt."""
    hashed = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
    return hashed.decode('utf-8')

def verify_password(password, hashed_password):
    """Перевіряє, чи відповідає пароль хешованому паролю."""
    try:
        return bcrypt.checkpw(password.encode('utf-8'), hashed_password.encode('utf-8'))
    except ValueError:
        return False

# --- Функції логування ---
def log_event(message):
    """Записує подію в лог-файл з часовою міткою."""
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open(log_file_full_path, "a", encoding="utf-8") as f:
        f.write(f"[{timestamp}] {message}\n")
    print(message) # Виводимо також у консоль

# --- Функції роботи з базою даних (ОНОВЛЕНО ДЛЯ SQL SERVER) ---
def get_db_connection():
    """Створює та повертає з'єднання з базою даних SQL Server."""
    try:
        conn_str = (
            f"DRIVER={driver};"
            f"SERVER={server};"
            f"DATABASE={database};"
            f"UID={username};"
```

```
                f"PWD={password}"
            )
            conn = pyodbc.connect(conn_str)
            return conn
        except pyodbc.Error as ex:
            sqlstate = ex.args[0]
            log_event(f"Помилка підключення до БД: {sqlstate} -
{ex.args[1]}")
            raise # Rethrow the exception to stop execution if connection
fails

    def create_users_table():
        """Створює таблицю користувачів у SQL Server, якщо вона не існує."""
        conn = None
        try:
            conn = get_db_connection()
            cursor = conn.cursor()
            cursor.execute("""
                IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'Users'
AND type = 'U')
                CREATE TABLE Users (
                    ID INT PRIMARY KEY IDENTITY(1,1),
                    UID NVARCHAR(50) UNIQUE NOT NULL,
                    FullName NVARCHAR(255) NOT NULL,
                    AccessLevel INT NOT NULL,
                    AccessStart TIME(7), -- Змінено на TIME(7)
                    AccessEnd TIME(7),   -- Змінено на TIME(7)
                    PasswordHash NVARCHAR(255) NOT NULL
                )
            """)
            conn.commit()
            log_event("Таблиця 'Users' перевірена/створена в SQL Server.")
        except pyodbc.Error as ex:
            sqlstate = ex.args[0]
            log_event(f"Помилка БД при створенні таблиці: {sqlstate} -
{ex.args[1]}")
            if conn:
                conn.rollback()
        finally:
            if conn:
                conn.close()


    def register_user(cursor, conn):
        print("\n--- Реєстрація нового користувача ---")
        uid = input("Введіть UID нового користувача: ").strip()
        full_name = input("Введіть повне ім'я нового користувача:
").strip()
        password = input("Введіть пароль для нового користувача: ").strip()

        # Перевірка наявності UID
        cursor.execute("SELECT COUNT(*) FROM Users WHERE UID = ?", (uid,))
        if cursor.fetchone()[0] > 0:
            log_event(f"Помилка реєстрації: UID '{uid}' вже існує.")
            return False

        # Введення рівня доступу з валідацією
        access_level_str = input("Введіть рівень доступу (1, 2 або 3):
").strip()
```

```
        try:
            access_level = int(access_level_str)
            if not (1 <= access_level <= 3):
                log_event(f"Помилка реєстрації: Некоректний рівень доступу
'{access_level_str}'. Введіть число від 1 до 3.")
                return False
        except ValueError:
            log_event(f"Помилка реєстрації: Некоректний формат рівня
доступу. Введіть число від 1 до 3.")
            return False

        # Введення часу початку доступу
        access_start_str = input("Введіть час початку доступу (HH:MM:SS,
залиште порожнім для відсутності обмежень): ").strip()
        try:
            access_start       =       datetime.strptime(access_start_str,
'%H:%M:%S').time() if access_start_str else time.min
        except ValueError:
            log_event(f"Помилка реєстрації: Некоректний формат часу початку
доступу. Будь ласка, використовуйте HH:MM:SS або залиште порожнім.")
            return False

        # Введення часу закінчення доступу
        access_end_str = input("Введіть час закінчення доступу (HH:MM:SS,
залиште порожнім для відсутності обмежень): ").strip()
        try:
            access_end         =         datetime.strptime(access_end_str,
'%H:%M:%S').time() if access_end_str else time.max
        except ValueError:
            log_event(f"Помилка реєстрації: Некоректний формат часу
закінчення доступу. Будь ласка, використовуйте HH:MM:SS або залиште
порожнім.")
            return False

        hashed_password = hash_password(password)

        try:
            cursor.execute(
                "INSERT INTO Users (UID, FullName, AccessLevel,
AccessStart, AccessEnd, PasswordHash) VALUES (?, ?, ?, ?, ?, ?)",
                (uid, full_name, access_level, access_start, access_end,
hashed_password) # Передаємо об'єкти time
            )
            conn.commit()
            log_event(f"Успішна реєстрація: UID '{uid}', FullName
'{full_name}'.")
            return True
        except pyodbc.Error as e:
            conn.rollback()
            log_event(f"Помилка БД при реєстрації UID '{uid}':
{e.args[1]}")
            return False


    def check_access(cursor):
        print("\n--- Перевірка доступу ---")
        uid = input("Введіть UID: ").strip()
        password = input("Введіть пароль: ").strip()
```

```
        now = datetime.now()
        current_time = now.time()

        if uid in failed_attempts and
failed_attempts[uid].get('blocked_until'):
            blocked_until = failed_attempts[uid]['blocked_until']
            if now < blocked_until:
                log_event(f"Спроба доступу: UID '{uid}' заблокований до
{blocked_until.strftime('%Y-%m-%d %H:%M:%S')}.")
                return False
            else:
                log_event(f"Блокування: UID '{uid}' розблоковано після
{BLOCK_DURATION_SECONDS} секунд.")
                failed_attempts.pop(uid)

        cursor.execute("SELECT FullName, AccessLevel, AccessStart,
AccessEnd, PasswordHash FROM Users WHERE UID = ?", (uid,))
        user_row = cursor.fetchone()

        if user_row:
            # Отримуємо значення з user_row. Вони можуть бути рядками або
datetime.time об'єктами.
            # Перетворюємо їх на datetime.time, якщо вони рядки, або
використовуємо time.min/time.max
            access_start_from_db = user_row[2]
            access_end_from_db = user_row[3]

            user_access_start = time.min # Значення за замовчуванням, якщо
access_start_from_db порожній або некоректний
            if access_start_from_db is not None:
                if isinstance(access_start_from_db, str):
                    try:
                        user_access_start =
datetime.strptime(access_start_from_db, '%H:%M:%S.%f').time() # Спробуємо з
мілісекундами
                    except ValueError:
                        try:
                            user_access_start =
datetime.strptime(access_start_from_db, '%H:%M:%S').time() # Якщо без
мілісекунд
                        except ValueError:
                            user_access_start = time.min # Fallback, якщо
не вдалося розпарсити
                elif isinstance(access_start_from_db, time):
                    user_access_start = access_start_from_db

            user_access_end = time.max # Значення за замовчуванням, якщо
access_end_from_db порожній або некоректний
            if access_end_from_db is not None:
                if isinstance(access_end_from_db, str):
                    try:
                        user_access_end =
datetime.strptime(access_end_from_db, '%H:%M:%S.%f').time() # Спробуємо з
мілісекундами
                    except ValueError:
                        try:
                            user_access_end =
datetime.strptime(access_end_from_db, '%H:%M:%S').time() # Якщо без
мілісекунд
```

```
                        except ValueError:
                            user_access_end = time.max # Fallback, якщо не
вдалося розпарсити
                elif isinstance(access_end_from_db, time):
                    user_access_end = access_end_from_db

            user = {
                "FullName": user_row[0],
                "AccessLevel": user_row[1],
                "AccessStart": user_access_start,
                "AccessEnd": user_access_end,
                "PasswordHash": user_row[4]
            }

            if verify_password(password, user['PasswordHash']):
                # Порівнюємо об'єкти time
                if     user['AccessStart']     <=     current_time     <=
user['AccessEnd']:
                    if uid in failed_attempts:
                        log_event(f"Успішний вхід: UID '{uid}', FullName
'{user['FullName']}'. Лічильник спроб скинуто.")
                        failed_attempts.pop(uid)
                    else:
                        log_event(f"Успішний вхід: UID '{uid}', FullName
'{user['FullName']}'.")
                    return True
                else:
                    log_event(f"Відмовлено в доступі: UID '{uid}' за межами
дозволеного     часу     ({user['AccessStart'].strftime('%H:%M:%S')}     -
{user['AccessEnd'].strftime('%H:%M:%S')}).")
                    update_failed_attempts(uid)
                    return False
            else:
                log_event(f"Невдала спроба входу: Невірний пароль для UID
'{uid}'.")
                update_failed_attempts(uid)
                return False
        else:
            log_event(f"Невдала спроба входу: UID '{uid}' не знайдено.")
            return False

    def update_failed_attempts(uid):
        """Оновлює лічильник невдалих спроб і блокує користувача, якщо
досягнуто ліміту."""
        if uid not in failed_attempts:
            failed_attempts[uid] = {'attempts': 0}

        attempts_info = failed_attempts[uid]
        attempts_info['attempts'] += 1

        log_event(f"Невдала    спроба    входу:    UID    '{uid}'.    Спроб:
{attempts_info['attempts']}/{MAX_FAILED_ATTEMPTS}")

        if attempts_info['attempts'] >= MAX_FAILED_ATTEMPTS:
            attempts_info['blocked_until']     =     datetime.now()     +
timedelta(seconds=BLOCK_DURATION_SECONDS)
            log_event(f"Блокування:    UID    '{uid}'    заблоковано    на
{BLOCK_DURATION_SECONDS} секунд через забагато невдалих спроб.")
        else:
```

```
            attempts_info.pop('blocked_until', None)


    def delete_user(cursor, conn):
        """Видаляє користувача за UID."""
        print("\n--- Видалення користувача ---")
        uid_to_delete = input("Введіть UID користувача для видалення:
").strip()

        cursor.execute("SELECT COUNT(*) FROM Users WHERE UID = ?",
(uid_to_delete,))
        if cursor.fetchone()[0] == 0:
            log_event(f"Помилка видалення: Користувача з UID
'{uid_to_delete}' не знайдено.")
            return False

        try:
            cursor.execute("DELETE FROM Users WHERE UID = ?",
(uid_to_delete,))
            conn.commit()
            log_event(f"Успішне видалення: Користувач з UID
'{uid_to_delete}' видалений.")
            return True
        except pyodbc.Error as e:
            conn.rollback()
            log_event(f"Помилка БД при видаленні UID '{uid_to_delete}':
{e.args[1]}")
            return False

    def view_users(cursor):
        """Виводить список всіх зареєстрованих користувачів."""
        print("\n--- Список користувачів ---")
        cursor.execute("SELECT UID, FullName, AccessLevel, AccessStart,
AccessEnd FROM Users ORDER BY UID")
        users = cursor.fetchall()

        if not users:
            print("Користувачів не знайдено.")
            return

        print("UID\tFullName\tAccessLevel\tAccessStart\tAccessEnd")
        print("-" * 70)
        for user in users:
            uid = user[0]
            full_name = user[1]
            access_level = user[2]
            # Перетворюємо об'єкти time у потрібний формат для виводу
            access_start = user[3].strftime('%H:%M:%S') if user[3] else
"N/A"
            access_end = user[4].strftime('%H:%M:%S') if user[4] else "N/A"


print(f"{uid:<8}{full_name:<16}{access_level:<12}{access_start:<15}{access_
end:<15}")


    def update_user_password(cursor, conn):
        """Оновлює пароль для існуючого користувача за UID."""
        print("\n--- Оновлення пароля користувача ---")
```

```
        uid = input("Введіть UID користувача, чий пароль потрібно оновити:
").strip()
        new_password = input("Введіть новий пароль: ").strip()

        cursor.execute("SELECT COUNT(*) FROM Users WHERE UID = ?", (uid,))
        if cursor.fetchone()[0] == 0:
            log_event(f"Помилка оновлення: Користувача з UID '{uid}' не
знайдено.")
            return False

        hashed_new_password = hash_password(new_password)

        try:
            cursor.execute("UPDATE Users SET PasswordHash = ? WHERE UID =
?", (hashed_new_password, uid))
            conn.commit()
            log_event(f"Успішне оновлення: Пароль для UID '{uid}'
оновлено.")
            return True
        except pyodbc.Error as e:
            conn.rollback()
            log_event(f"Помилка БД при оновленні пароля для UID '{uid}':
{e.args[1]}")
            return False

    def update_user_access_level(cursor, conn):
        """Оновлює рівень доступу для існуючого користувача за UID."""
        print("\n--- Оновлення рівня доступу користувача ---")
        uid = input("Введіть UID користувача, чий рівень доступу потрібно
оновити: ").strip()
        new_access_level_str = input("Введіть новий рівень доступу (1, 2
або 3): ").strip()

        try:
            new_access_level = int(new_access_level_str)
            if not (1 <= new_access_level <= 3):
                log_event(f"Помилка оновлення: Некоректний рівень доступу
'{new_access_level_str}'. Введіть число від 1 до 3.")
                return False
        except ValueError:
            log_event(f"Помилка оновлення: Некоректний формат рівня
доступу. Введіть число від 1 до 3.")
            return False

        cursor.execute("SELECT COUNT(*) FROM Users WHERE UID = ?", (uid,))
        if cursor.fetchone()[0] == 0:
            log_event(f"Помилка оновлення: Користувача з UID '{uid}' не
знайдено.")
            return False

        try:
            cursor.execute("UPDATE Users SET AccessLevel = ? WHERE UID =
?", (new_access_level, uid))
            conn.commit()
            log_event(f"Успішне оновлення: Рівень доступу для UID '{uid}'
оновлено на {new_access_level}.")
            return True
        except pyodbc.Error as e:
            conn.rollback()
```

```
            log_event(f"Помилка БД при оновленні рівня доступу для UID
'{uid}': {e.args[1]}")
            return False


    # --- Головна функція ---
    def main():
        create_users_table()
        conn = None
        try:
            conn = get_db_connection()
            cursor = conn.cursor()

            while True:
                print("\n--- Меню ---")
                print("1. Зареєструвати нового користувача")
                print("2. Перевірити доступ")
                print("3. Видалити користувача")
                print("4. Переглянути список користувачів")
                print("5. Оновити пароль користувача")
                print("6. Оновити рівень доступу користувача")
                print("7. Вийти")

                choice = input("Оберіть опцію: ")

                if choice == '1':
                    register_user(cursor, conn)
                elif choice == '2':
                    check_access(cursor)
                elif choice == '3':
                    delete_user(cursor, conn)
                elif choice == '4':
                    view_users(cursor)
                elif choice == '5':
                    update_user_password(cursor, conn)
                elif choice == '6':
                    update_user_access_level(cursor, conn)
                elif choice == '7':
                    print("Вихід...")
                    break
                else:
                    print("Невірний вибір. Будь ласка, спробуйте ще раз.")
        except pyodbc.Error as ex:
            log_event(f"Критична помилка в main: {ex.args[1]}")
        finally:
            if conn:
                conn.close()

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

test_db_test.py

```
    import unittest
    from unittest.mock import patch, MagicMock
    from datetime import datetime, timedelta, time
    import os
    import sys

    # Додаємо шлях до батьківської директорії, щоб імпортувати db_test
```

```
    sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

    # Імпортуємо функції з db_test.py
    from db_test import (
        register_user,
        check_access,
        delete_user,
        update_user_password,
        update_user_access_level,
        log_event,
        hash_password,
        verify_password,
        failed_attempts,
        BLOCK_DURATION_SECONDS,
        MAX_FAILED_ATTEMPTS
    )

    # Важливо: скидаємо failed_attempts перед кожним запуском тестів,
    # щоб не впливати на результати наступних тестів.
    def reset_failed_attempts_for_tests():
        global failed_attempts
        failed_attempts.clear()

    class TestUserRegistration(unittest.TestCase):
        @classmethod
        def setUpClass(cls):
            cls.patcher_log_event = patch('db_test.log_event')
            cls.mock_log_event = cls.patcher_log_event.start()
            cls.patcher_pyodbc_connect = patch('db_test.pyodbc.connect')
            cls.mock_pyodbc_connect = cls.patcher_pyodbc_connect.start()

        @classmethod
        def tearDownClass(cls):
            cls.patcher_log_event.stop()
            cls.patcher_pyodbc_connect.stop()

        def setUp(self):
            self.mock_log_event.reset_mock()
            reset_failed_attempts_for_tests()

            self.mock_conn = MagicMock()
            self.mock_cursor = MagicMock()
            self.mock_conn.cursor.return_value = self.mock_cursor

        # Тести для register_user
        @patch('builtins.input', side_effect=['test_uid_1', 'Test User 1',
'password123', '2', '', ''])
        def test_successful_registration(self, mock_input):
            self.mock_cursor.fetchone.return_value = (0,) # UID не існує
            result = register_user(self.mock_cursor, self.mock_conn)
            self.assertTrue(result)
            self.mock_conn.commit.assert_called_once()
            self.mock_log_event.assert_called_once_with("Успішна
реєстрація: UID 'test_uid_1', FullName 'Test User 1'.")

            # Перевіряємо, що execute був викликаний ДВІЧІ (один SELECT,
один INSERT)
            self.assertEqual(self.mock_cursor.execute.call_count, 2)
```

```
            # Перевіряємо, що були зроблені конкретні виклики
            self.mock_cursor.execute.assert_has_calls([
                unittest.mock.call("SELECT COUNT(*) FROM Users WHERE UID =
?", ('test_uid_1',)),
                unittest.mock.call("INSERT INTO Users (UID, FullName,
AccessLevel, AccessStart, AccessEnd, PasswordHash) VALUES (?, ?, ?, ?, ?,
?)",
                                   ('test_uid_1', 'Test User 1', 2, time(0,
0, 0), time(23, 59, 59, 999999), unittest.mock.ANY))
            ])


        @patch('builtins.input', side_effect=['existing_uid', 'Test User',
'password', '1', '', ''])
        def test_registration_with_existing_uid(self, mock_input):
            self.mock_cursor.fetchone.return_value = (1,) # UID існує
            result = register_user(self.mock_cursor, self.mock_conn)
            self.assertFalse(result)
            self.mock_cursor.execute.assert_called_once_with("SELECT
COUNT(*) FROM Users WHERE UID = ?", ('existing_uid',))
            self.mock_conn.commit.assert_not_called()
            self.mock_log_event.assert_called_once_with("Помилка
реєстрації: UID 'existing_uid' вже існує.")

        @patch('builtins.input', side_effect=['new_uid', 'New User',
'password', 'invalid_level', '', ''])
        def test_registration_with_invalid_access_level_format(self,
mock_input):
            self.mock_cursor.fetchone.return_value = (0,) # UID не існує
            result = register_user(self.mock_cursor, self.mock_conn)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Помилка
реєстрації: Некоректний формат рівня доступу. Введіть число від 1 до 3.")
            self.mock_conn.commit.assert_not_called()

        @patch('builtins.input', side_effect=['new_uid', 'New User',
'password', '0', '', ''])
        def test_registration_with_access_level_out_of_range(self,
mock_input):
            self.mock_cursor.fetchone.return_value = (0,) # UID не існує
            result = register_user(self.mock_cursor, self.mock_conn)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Помилка
реєстрації: Некоректний рівень доступу '0'. Введіть число від 1 до 3.")
            self.mock_conn.commit.assert_not_called()

        @patch('builtins.input', side_effect=['new_uid', 'New User',
'password', '1', 'invalid_time', ''])
        def test_registration_with_invalid_access_start_time_format(self,
mock_input):
            self.mock_cursor.fetchone.return_value = (0,)
            result = register_user(self.mock_cursor, self.mock_conn)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Помилка
реєстрації: Некоректний формат часу початку доступу. Будь ласка,
використовуйте HH:MM:SS або залиште порожнім.")
            self.mock_conn.commit.assert_not_called()
```

```
        @patch('builtins.input', side_effect=['new_uid', 'New User',
'password', '1', '10:00:00', 'invalid_time'])
        def test_registration_with_invalid_access_end_time_format(self,
mock_input):
            self.mock_cursor.fetchone.return_value = (0,)
            result = register_user(self.mock_cursor, self.mock_conn)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Помилка
реєстрації: Некоректний формат часу закінчення доступу. Будь ласка,
використовуйте HH:MM:SS або залиште порожнім.")
            self.mock_conn.commit.assert_not_called()

    class TestUserAccess(unittest.TestCase):
        @classmethod
        def setUpClass(cls):
            cls.patcher_log_event = patch('db_test.log_event')
            cls.mock_log_event = cls.patcher_log_event.start()
            cls.patcher_pyodbc_connect = patch('db_test.pyodbc.connect')
            cls.mock_pyodbc_connect = cls.patcher_pyodbc_connect.start()

        @classmethod
        def tearDownClass(cls):
            cls.patcher_log_event.stop()
            cls.patcher_pyodbc_connect.stop()

        def setUp(self):
            self.mock_log_event.reset_mock()
            reset_failed_attempts_for_tests()
            self.mock_conn = MagicMock()
            self.mock_cursor = MagicMock()
            self.mock_conn.cursor.return_value = self.mock_cursor

        # Тести для check_access
        @patch('builtins.input', side_effect=['test_uid',
'correct_password'])
        @patch('db_test.datetime')
        def test_successful_access(self, mock_datetime, mock_input):
            mock_datetime.now.return_value = datetime(2025, 1, 1, 12, 0, 0)
# Поточний час 12:00:00
            mock_datetime.strptime = datetime.strptime # Передаємо реальну
функцію strptime
            mock_datetime.time = time # Передаємо реальний клас time

            # Дані користувача: повний доступ за часом
            user_data = ('Test User', 1, time(0, 0, 0), time(23, 59, 59),
hash_password('correct_password'))
            self.mock_cursor.fetchone.return_value = user_data

            result = check_access(self.mock_cursor)
            self.assertTrue(result)
            self.mock_log_event.assert_called_once_with("Успішний вхід:
UID 'test_uid', FullName 'Test User'.")

        @patch('builtins.input', side_effect=['test_uid',
'wrong_password'])
        @patch('db_test.datetime')
        def test_access_with_wrong_password(self, mock_datetime,
mock_input):
            mock_datetime.now.return_value = datetime(2025, 1, 1, 12, 0, 0)
```

```
            mock_datetime.strptime = datetime.strptime
            mock_datetime.time = time

            user_data = ('Test User', 1, time(0, 0, 0), time(23, 59, 59),
hash_password('correct_password'))
            self.mock_cursor.fetchone.return_value = user_data

            result = check_access(self.mock_cursor)
            self.assertFalse(result)
            self.mock_log_event.assert_any_call("Невдала   спроба   входу:
Невірний пароль для UID 'test_uid'.")
            self.assertIn('test_uid', failed_attempts)
            self.assertEqual(failed_attempts['test_uid']['attempts'], 1)

        @patch('builtins.input',          side_effect=['non_existent_uid',
'any_password'])
        def test_access_with_non_existent_uid(self, mock_input):
            self.mock_cursor.fetchone.return_value = None # Користувача не
знайдено

            result = check_access(self.mock_cursor)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Невдала    спроба
входу: UID 'non_existent_uid' не знайдено.")

        @patch('builtins.input',                  side_effect=['test_uid',
'correct_password'])
        @patch('db_test.datetime')
        def       test_access_out_of_time_range(self,       mock_datetime,
mock_input):
            mock_datetime.now.return_value = datetime(2025, 1, 1, 6, 0, 0)
# Поточний час 06:00:00
            mock_datetime.strptime = datetime.strptime
            mock_datetime.time = time

            # Доступ дозволено з 08:00:00 до 18:00:00
            user_data = ('Test User', 1, time(8, 0, 0), time(18, 0, 0),
hash_password('correct_password'))
            self.mock_cursor.fetchone.return_value = user_data

            result = check_access(self.mock_cursor)
            self.assertFalse(result)
            self.mock_log_event.assert_any_call("Відмовлено в доступі: UID
'test_uid' за межами дозволеного часу (08:00:00 - 18:00:00).")
            self.assertIn('test_uid', failed_attempts)
            self.assertEqual(failed_attempts['test_uid']['attempts'], 1)

    class TestBlockingMechanism(unittest.TestCase):
        @classmethod
        def setUpClass(cls):
            cls.patcher_log_event = patch('db_test.log_event')
            cls.mock_log_event = cls.patcher_log_event.start()
            cls.patcher_pyodbc_connect = patch('db_test.pyodbc.connect')
            cls.mock_pyodbc_connect = cls.patcher_pyodbc_connect.start()

        @classmethod
        def tearDownClass(cls):
            cls.patcher_log_event.stop()
            cls.patcher_pyodbc_connect.stop()
```

```
        def setUp(self):
            self.mock_log_event.reset_mock()
            reset_failed_attempts_for_tests()  # Скидаємо перед кожним
тестом
            self.mock_conn = MagicMock()
            self.mock_cursor = MagicMock()
            self.mock_conn.cursor.return_value = self.mock_cursor

            # Дані для тестового користувача
            self.user_uid = 'blocked_user'
            self.user_password = 'block_pass' # nosec B105
            self.user_data = ('Blocked User', 1, time(0, 0, 0), time(23,
59, 59), hash_password(self.user_password))

        @patch('builtins.input',              side_effect=['blocked_user',
'wrong_pass', # 1-ша спроба в check_access
                                           'blocked_user', 'wrong_pass',
# 2-га спроба в check_access
                                           'blocked_user', 'wrong_pass',
# 3-тя спроба в check_access (блокування)
                                           'blocked_user',
'wrong_pass'])# 4-та спроба в check_access (під час блокування)
        @patch('db_test.datetime')
        def test_user_gets_blocked_after_max_attempts(self, mock_datetime,
mock_input):
            mock_datetime.now.side_effect = [
                datetime(2025, 1, 1, 10, 0, 0), # 1-ша спроба (для `now =
datetime.now()`)
                datetime(2025, 1, 1, 10, 0, 1), # 2-га спроба (для `now =
datetime.now()`)
                datetime(2025, 1, 1, 10, 0, 2), # 3-тя спроба (для `now =
datetime.now()`)
                datetime(2025, 1, 1, 10, 0, 2), # Для `blocked_until` в
`update_failed_attempts` після 3-ї спроби
                datetime(2025, 1, 1, 10, 0, 3)  # 4-та спроба (для `now =
datetime.now()`)
            ]
            mock_datetime.strptime = datetime.strptime
            mock_datetime.time = time
            mock_datetime.timedelta = timedelta # Забезпечуємо доступ до
timedelta

            self.mock_cursor.fetchone.return_value  =  self.user_data  #
Завжди повертаємо дані користувача

            # 1-ша невдала спроба
            check_access(self.mock_cursor)
            self.assertEqual(failed_attempts[self.user_uid]['attempts'],
1)
            self.assertNotIn('blocked_until',
failed_attempts[self.user_uid])

            # 2-га невдала спроба
            self.mock_log_event.reset_mock()  #  Скидаємо  для  перевірки
наступного логу
            check_access(self.mock_cursor)
            self.assertEqual(failed_attempts[self.user_uid]['attempts'],
2)
```

```
            self.assertNotIn('blocked_until',
failed_attempts[self.user_uid])

            # 3-тя невдала спроба (має заблокувати)
            self.mock_log_event.reset_mock()
            check_access(self.mock_cursor)
            self.assertEqual(failed_attempts[self.user_uid]['attempts'],
3)
            self.assertIn('blocked_until', failed_attempts[self.user_uid])
            self.mock_log_event.assert_any_call(f"Блокування:          UID
'{self.user_uid}'  заблоковано  на  {BLOCK_DURATION_SECONDS}  секунд  через
забагато невдалих спроб.")

            # Спроба під час блокування
            self.mock_log_event.reset_mock()
            result = check_access(self.mock_cursor)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with(f"Спроба  доступу:
UID              '{self.user_uid}'              заблокований              до
{failed_attempts[self.user_uid]['blocked_until'].strftime('%Y-%m-%d
%H:%M:%S')}.")

        @patch('builtins.input',              side_effect=['blocked_user',
'correct_pass'])
        @patch('db_test.datetime')
        @patch('db_test.verify_password')
        def             test_successful_login_resets_failed_attempts(self,
mock_verify_password, mock_datetime, mock_input):
            mock_verify_password.return_value = True

            mock_datetime.now.side_effect = [
                datetime(2025, 1, 1, 10, 0, 2),
                datetime(2025, 1, 1, 10, 0, 3),
            ]
            mock_datetime.strptime = datetime.strptime
            mock_datetime.time = time
            mock_datetime.timedelta = timedelta

            self.mock_cursor.fetchone.return_value = self.user_data

            failed_attempts[self.user_uid] = {'attempts': 2}

            result = check_access(self.mock_cursor)
            self.assertTrue(result)
            mock_verify_password.assert_called_once_with('correct_pass',
self.user_data[4])
            self.mock_log_event.assert_any_call(f"Успішний    вхід:    UID
'{self.user_uid}', FullName 'Blocked User'. Лічильник спроб скинуто.")
            self.assertNotIn(self.user_uid, failed_attempts)

        @patch('builtins.input',              side_effect=['blocked_user',
'wrong_pass'])
        @patch('db_test.datetime')
        def    test_user_unblocked_after_duration(self,    mock_datetime,
mock_input):
            mock_datetime.now.side_effect = [
                datetime(2025, 1, 1, 10, 0, 0),
                datetime(2025,     1,     1,     10,     0,     0)     +
timedelta(seconds=BLOCK_DURATION_SECONDS + 1)
```

```
            ]
            mock_datetime.strptime = datetime.strptime
            mock_datetime.time = time
            mock_datetime.timedelta = timedelta

            self.mock_cursor.fetchone.return_value = self.user_data

            failed_attempts[self.user_uid] = {
                'attempts': MAX_FAILED_ATTEMPTS,
                'blocked_until': datetime(2025, 1, 1, 10, 0, 0)
            }

            self.mock_log_event.reset_mock()
            result = check_access(self.mock_cursor)
            self.assertFalse(result)
            self.mock_log_event.assert_any_call(f"Блокування:          UID
'{self.user_uid}' розблоковано після {BLOCK_DURATION_SECONDS} секунд.")
            self.mock_log_event.assert_any_call(f"Невдала  спроба  входу:
Невірний пароль для UID '{self.user_uid}'.")
            self.assertIn(self.user_uid, failed_attempts)
            self.assertEqual(failed_attempts[self.user_uid]['attempts'],
1)


    class TestUserManagement(unittest.TestCase):
        @classmethod
        def setUpClass(cls):
            cls.patcher_log_event = patch('db_test.log_event')
            cls.mock_log_event = cls.patcher_log_event.start()
            cls.patcher_pyodbc_connect = patch('db_test.pyodbc.connect')
            cls.mock_pyodbc_connect = cls.patcher_pyodbc_connect.start()

        @classmethod
        def tearDownClass(cls):
            cls.patcher_log_event.stop()
            cls.patcher_pyodbc_connect.stop()

        def setUp(self):
            self.mock_log_event.reset_mock()
            reset_failed_attempts_for_tests()
            self.mock_conn = MagicMock()
            self.mock_cursor = MagicMock()
            self.mock_conn.cursor.return_value = self.mock_cursor

        # Тести для delete_user
        @patch('builtins.input', side_effect=['uid_to_delete'])
        def test_delete_existing_user(self, mock_input):
            self.mock_cursor.fetchone.side_effect   =   [(1,),   None]   #
Користувач існує, потім видаляється
            result = delete_user(self.mock_cursor, self.mock_conn)
            self.assertTrue(result)
            self.mock_cursor.execute.assert_any_call("SELECT COUNT(*) FROM
Users WHERE UID = ?", ('uid_to_delete',))
            self.mock_cursor.execute.assert_any_call("DELETE  FROM  Users
WHERE UID = ?", ('uid_to_delete',))
            self.mock_conn.commit.assert_called_once()
            self.mock_log_event.assert_called_once_with("Успішне
видалення: Користувач з UID 'uid_to_delete' видалений.")
```

```
        @patch('builtins.input', side_effect=['non_existent_uid'])
        def test_delete_non_existent_user(self, mock_input):
            self.mock_cursor.fetchone.return_value = (0,) # Користувача не
знайдено
            result = delete_user(self.mock_cursor, self.mock_conn)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Помилка
видалення: Користувача з UID 'non_existent_uid' не знайдено.")
            self.mock_conn.commit.assert_not_called()

        # Тести для update_user_password
        @patch('builtins.input',             side_effect=['uid_to_update',
'new_password'])
        def test_update_existing_user_password(self, mock_input):
            self.mock_cursor.fetchone.return_value  =  (1,)  #  Користувач
існує
            # Очікуємо, що bcrypt.hashpw буде викликано
            with                              patch('db_test.hash_password',
return_value='hashed_new_password') as mock_hash_password:
                result      =      update_user_password(self.mock_cursor,
self.mock_conn)
                self.assertTrue(result)

mock_hash_password.assert_called_once_with('new_password')
                self.mock_cursor.execute.assert_any_call("UPDATE Users SET
PasswordHash = ? WHERE UID = ?", ('hashed_new_password', 'uid_to_update'))
                self.mock_conn.commit.assert_called_once()
                self.mock_log_event.assert_called_once_with("Успішне
оновлення: Пароль для UID 'uid_to_update' оновлено.")

        @patch('builtins.input',          side_effect=['non_existent_uid',
'new_password'])
        def test_update_non_existent_user_password(self, mock_input):
            self.mock_cursor.fetchone.return_value = (0,) # Користувача не
знайдено
            result        =        update_user_password(self.mock_cursor,
self.mock_conn)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Помилка
оновлення: Користувача з UID 'non_existent_uid' не знайдено.")
            self.mock_conn.commit.assert_not_called()

        # Тести для update_user_access_level
        @patch('builtins.input', side_effect=['uid_to_update', '3'])
        def test_update_existing_user_access_level(self, mock_input):
            self.mock_cursor.fetchone.return_value  =  (1,)  #  Користувач
існує
            result      =      update_user_access_level(self.mock_cursor,
self.mock_conn)
            self.assertTrue(result)
            self.mock_cursor.execute.assert_any_call("UPDATE   Users   SET
AccessLevel = ? WHERE UID = ?", (3, 'uid_to_update'))
            self.mock_conn.commit.assert_called_once()
            self.mock_log_event.assert_called_once_with("Успішне
оновлення: Рівень доступу для UID 'uid_to_update' оновлено на 3.")

        @patch('builtins.input', side_effect=['non_existent_uid', '2'])
        def test_update_non_existent_user_access_level(self, mock_input):
```

```
            self.mock_cursor.fetchone.return_value = (0,) # Користувача не
знайдено
            result      =      update_user_access_level(self.mock_cursor,
self.mock_conn)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Помилка
оновлення: Користувача з UID 'non_existent_uid' не знайдено.")
            self.mock_conn.commit.assert_not_called()

        @patch('builtins.input',            side_effect=['uid_to_update',
'invalid_level'])
        def            test_update_user_access_level_invalid_format(self,
mock_input):
            self.mock_cursor.fetchone.return_value  =  (1,)  #  Користувач
існує, але рівень некоректний
            result      =      update_user_access_level(self.mock_cursor,
self.mock_conn)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Помилка
оновлення: Некоректний формат рівня доступу. Введіть число від 1 до 3.")
            self.mock_conn.commit.assert_not_called()

        @patch('builtins.input', side_effect=['uid_to_update', '4'])
        def test_update_user_access_level_out_of_range(self, mock_input):
            self.mock_cursor.fetchone.return_value  =  (1,)  #  Користувач
існує, але рівень поза діапазоном
            result      =      update_user_access_level(self.mock_cursor,
self.mock_conn)
            self.assertFalse(result)
            self.mock_log_event.assert_called_once_with("Помилка
оновлення: Некоректний рівень доступу '4'. Введіть число від 1 до 3.")
            self.mock_conn.commit.assert_not_called()


    class TestViewUsers(unittest.TestCase):
        @classmethod
        def setUpClass(cls):
            cls.patcher_log_event = patch('db_test.log_event')
            cls.mock_log_event = cls.patcher_log_event.start()
            cls.patcher_pyodbc_connect = patch('db_test.pyodbc.connect')
            cls.mock_pyodbc_connect = cls.patcher_pyodbc_connect.start()

        @classmethod
        def tearDownClass(cls):
            cls.patcher_log_event.stop()
            cls.patcher_pyodbc_connect.stop()

        def setUp(self):
            self.mock_log_event.reset_mock()
            self.mock_conn = MagicMock()
            self.mock_cursor = MagicMock()
            self.mock_conn.cursor.return_value = self.mock_cursor

            # Мокуємо builtins.print для захоплення ВСЬОГО виводу консолі
            self.patcher_print = patch('builtins.print')
            self.mock_print = self.patcher_print.start()

        def tearDown(self):
```

```
            self.patcher_print.stop() # Зупиняємо мок print після кожного
тесту

        def test_view_users_no_users(self):
            self.mock_cursor.fetchall.return_value = []
            from db_test import view_users
            view_users(self.mock_cursor)

            # Правильне отримання рядків з mock_print.call_args_list
            actual_print_calls = [call_arg.args[0] for call_arg in
self.mock_print.call_args_list]

            self.assertIn("\n--- Список користувачів ---",
actual_print_calls)
            self.assertIn("Користувачів не знайдено.", actual_print_calls)


        def test_view_users_with_data(self):
            test_users_data = [
                ('U001', 'User One', 1, time(9, 0, 0), time(17, 0, 0)),
                ('U002', 'User Two', 2, time(10, 0, 0), time(18, 0, 0))
            ]
            self.mock_cursor.fetchall.return_value = test_users_data

            from db_test import view_users
            view_users(self.mock_cursor)

            # Правильне отримання рядків з mock_print.call_args_list
            actual_output_lines = [call_arg.args[0] for call_arg in
self.mock_print.call_args_list]

            self.assertIn("\n--- Список користувачів ---",
actual_output_lines)

self.assertIn("UID\tFullName\tAccessLevel\tAccessStart\tAccessEnd",
actual_output_lines)
            self.assertIn("-" * 70, actual_output_lines)

            found_u001 = False
            found_u002 = False
            for line in actual_output_lines:
                if "U001" in line and "User One" in line and "1" in line
and "09:00:00" in line and "17:00:00" in line:
                    found_u001 = True
                if "U002" in line and "User Two" in line and "2" in line
and "10:00:00" in line and "18:00:00" in line:
                    found_u002 = True

            self.assertTrue(found_u001, "Expected User U001 details in
output")
            self.assertTrue(found_u002, "Expected User U002 details in
output")


    if __name__ == '__main__':
        unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

84

Додаток В

ІЛЮСТРАТИВНА ЧАСТИНА

ЗАСІБ КОНТРОЛЮ ДОСТУПУ ДО ПРИМІЩЕННЯ

Виконав: студент 4 курсу групи 1БКС-21б
спеціальності 125 Кібербезпека
_____ Ілля КОШЕВЕЦЬ
16 червня 2025 р.
Керівник: к. т. н., доцент каф. ЗІ
_____ Юрій БАРИШЕВ
16 червня 2025 р.

Аналіз методів автентифікації

Критерій	Пароль / PIN-код	Картка доступу / Брелок	Біометрія
Тип фактору	Знання	Наявність предмета	Біологічна ознака
Стійкість до підбору / зламу	Низька (легко вгадати, фішинг)	Середня (можна скопіювати чи вкрасти)	Висока (важко підробити або відтворити)
Необхідне обладнання	Немає	Карт-рідер, RFID-система	Біометричний сканер
Підходить для	Приміщення з базовим рівнем захисту	Офіси, підприємства	Об'єкти з високими вимогами безпеки

Порівняльна таблиця типів засобів доступу

Тип засобів доступу	Потенційні вразливості	Приклади використання	Вартість впровадження
Механічний ключ	Легко скопіювати або втратити	Невеликі офіси, склади, побутові приміщення	Низька
RFID-картка / брелок	Крадіжка або клонування	Бізнес-центри, школи, гуртожитки	Середня
PIN-код	Підглядання, витік пароля	Технічні приміщення, серверні	Низька
Біометрія	Труднощі з точністю, конфіденційність	Лабораторії, дата-центри, урядові об'єкти	Висока

Методика контролю доступу

Крок 1. Користувач звертається до адміністратора з проханням отримати дозвіл на вхід до певної зони. Адміністратор фіксує запит і уточнює параметри доступу (рівень, час).

Крок 2. Адміністратор перевіряє, чи має користувач відповідні права для доступу до цієї зони. Якщо потрібно, адміністратор може запросити додаткову інформацію або підтвердження.

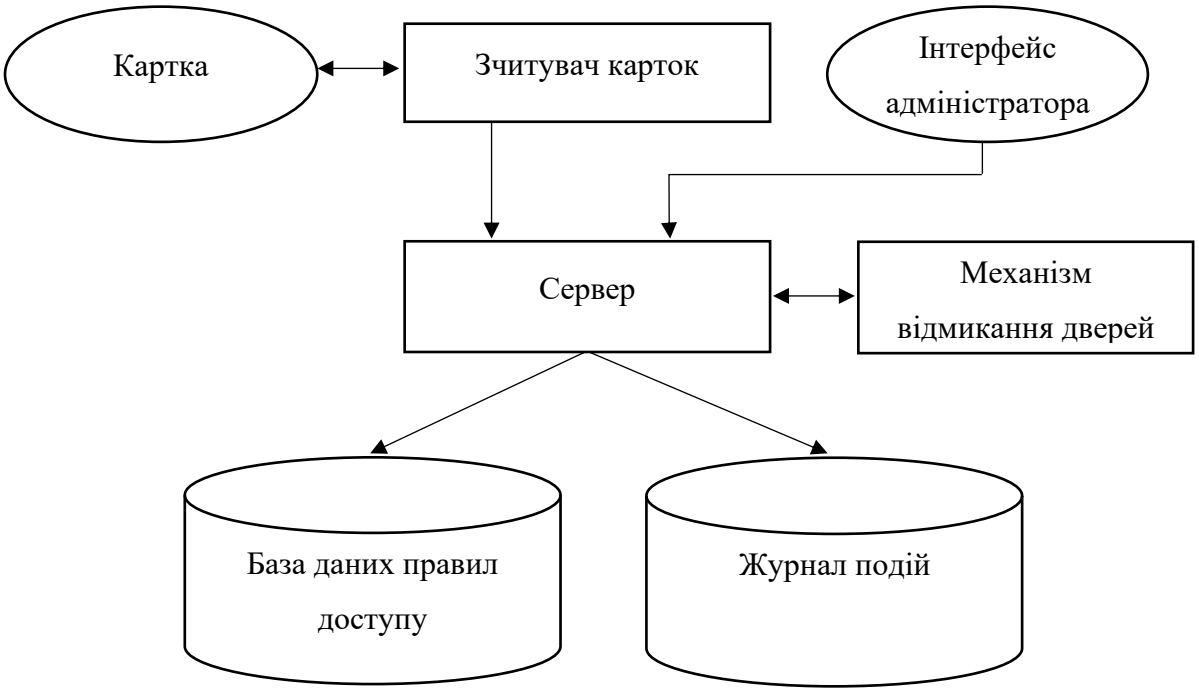
Крок 3. Якщо перевірка пройдена успішно, адміністратор налаштовує права доступу. Це може бути тимчасовий дозвіл (наприклад, на 30 хвилин) або постійний дозвіл, залежно від запиту. Права прив’язуються до ідентифікатора користувача та RFID-картки.

Крок 4. Користувачу видають RFID-картку з налаштованими правами доступу. Картка може бути активною постійно або з часовим обмеженням.

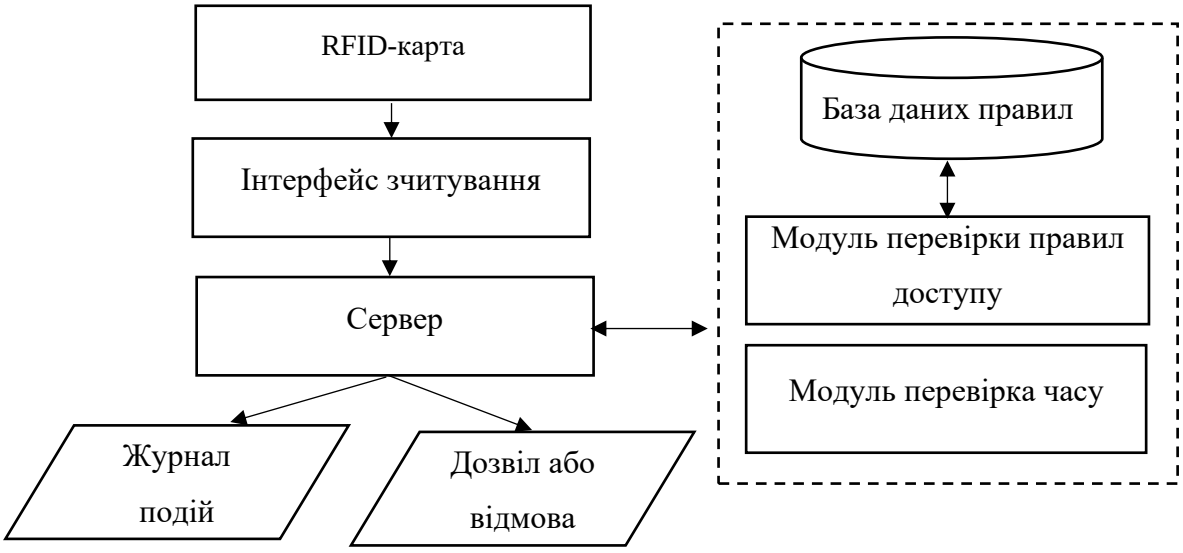
Крок 5. Користувач прикладає картку до зчитувача у потрібній зоні. Система перевіряє права та час доступу.

Крок 6. Система порівнює отримані дані з базою прав доступу. Якщо всі умови дотримані (користувач має дозвіл, картка не прострочена, зона відповідає дозволу), доступ надається. Всі спроби входу фіксуються в журналі для подальшого аудиту. Якщо умови не виконані — доступ блокується.

Узагальнена архітектура



Блок автентифікації користувачів



Сутності бази даних правил розмежування доступу

Сутність бази даних користувач

Поле	Тип даних	Опис
id_user	INT	Унікальний ідентифікатор користувача
fullname	VARCHAR(255)	Ім'я користувача
accessLevel	VARCHAR(255)	Роль користувача (менеджер, співробітник, охоронець)
access_start_time	DATETIME	Час активації доступу (для тимчасових користувачів)
access_end_time	DATETIME	Час деактивації доступу
password	VARCHAR(50)	Пароль користувача

Сутність бази даних право доступу

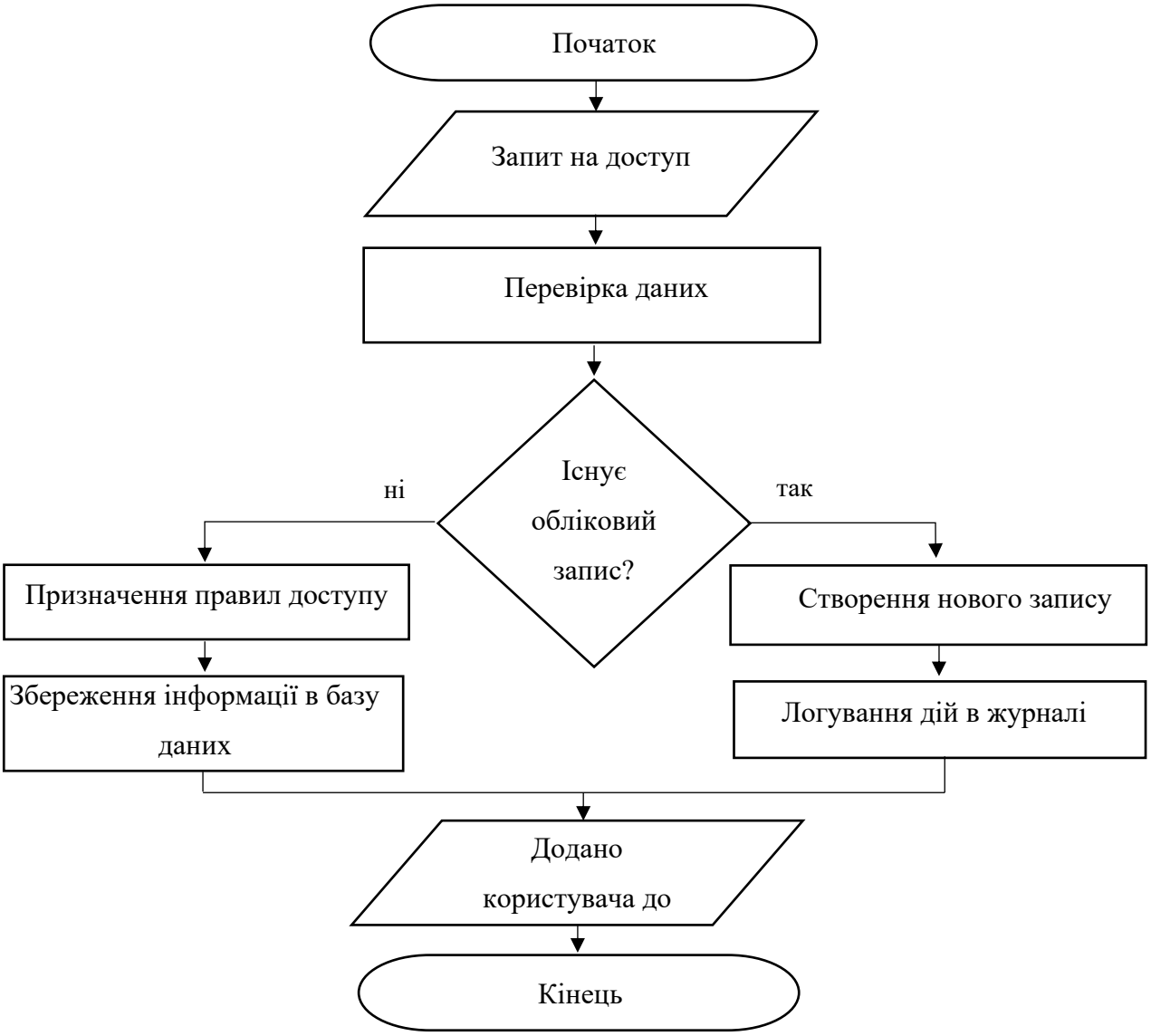
Поле	Тип даних	Опис
id_rule	INT	Унікальний ідентифікатор правила
id_user	INT	Ідентифікатор користувача (зовнішній ключ на таблицю "Користувачі")
id_reader	INT	Ідентифікатор зчитувача (зовнішній ключ на таблицю "Зчитувачі")
access_granted	BOOLEAN	Доступ надано (TRUE/ FALSE)
access_start_time	DATETIME	Час активації доступу
access_end_time	DATETIME	Час деактивації доступу

Сутність бази даних зчитувач картки

Поле	Тип даних	Опис
id_reader	INT	Унікальний ідентифікатор зчитувача
location_name	VARCHAR(255)	Назва або опис розташування зчитувача (наприклад, "Склад 1")
room_number	VARCHAR(50)	Номер або код приміщення
access_zone	VARCHAR(100)	Зона доступу, до якої належить зчитувач (наприклад, "Серверна")
reader_type	VARCHAR(100)	Тип зчитувача (наприклад, RFID, NFC, біометричний)
status	VARCHAR(50)	Статус зчитувача (активний, неактивний, у ремонті)

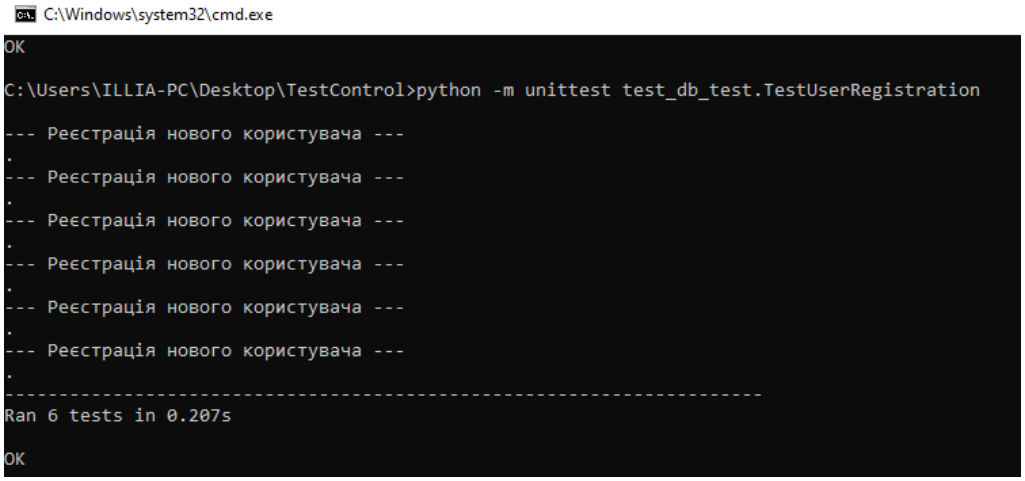
Узагальнений алгоритм роботи засобу

Алгоритм реєстрування користувачів

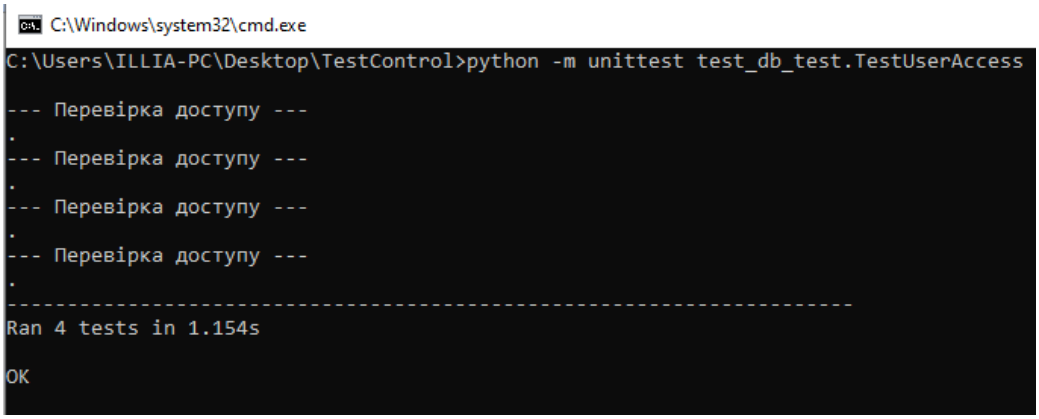


Результати модульного тестування

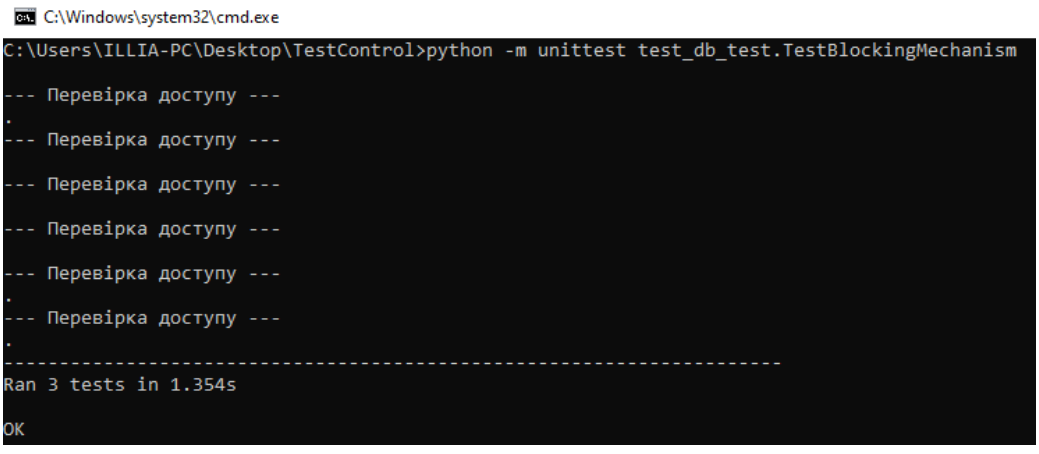
Тестування реєстрування користувачів



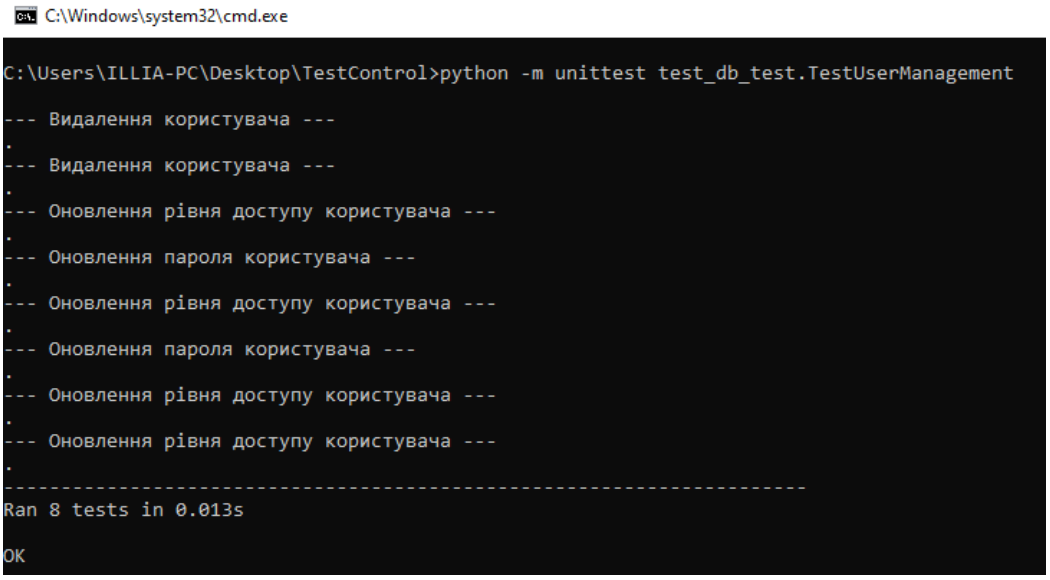
Тестування доступу користувачів до системи



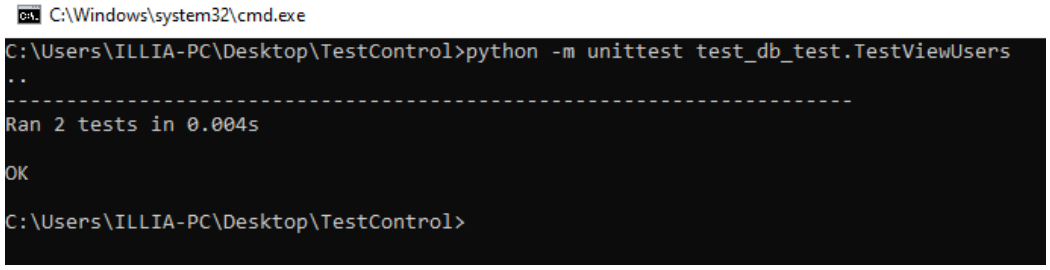
Тестування блокування доступу після трьох помилкових спроб



Тестування модуля керування користувачем



Тестування модуля перегляду користувачів



Результати статичного тестування безпеки

Помилка про жорстке кодування паролів

```
Test results:
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'Hs7skja78$
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
```

Помилка про тестовий пароль

```
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'block_pass'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
```

Результати первинного тестування

```
Code scanned:
        Total lines of code: 689
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 2
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 2
                High: 0
Files skipped (0):
```

Результати тестування після виправлення помилок

```
Test results:
        No issues identified.

Code scanned:
        Total lines of code: 319
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
Files skipped (0):
```