Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

# ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
**за першим (бакалаврським) рівнем вищої освіти**
на тему
**“Програмне забезпечення системи "розумний замок" з автентифікацією даних через мобільний додаток”**



Виконав здобувач вищої освіти
IV курсу, групи КІ-21-1
ОПП «Кібербезпека»
спеціальності 123 «Комп’ютерна інженерія»
_____ Лісевич Д.С.
«____» _____ 2025 р.

Керівник проекту
доктор технічних наук, професор
_____ Минайленко Р.М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

# ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

*Лісевичу Дмитру Сергійовичу*
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи "розумний замок" з автентифікацією даних через мобільний додаток*

2. Керівник роботи *Минайленко Роман Миколайович, канд. тех. наук, доцент*
( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 46-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи "розумний замок" з автентифікацією даних через мобільний додаток*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

*1. Призначення та область використання.*

*2.Перегляд аналогічних існуючих систем.*

*3. Опис і обґрунтування проектних рішень.*

*4. Етапи програмування системи.*

*5. Впровадження системи в промислову експлуатацію*.

*6. Висновки*

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

| | |
|---|---|
| *Структурна схема системи* | *1 аркуш* |
| *Функціональна схема системи* | *1 аркуш* |
| *Діаграма процесів* | *1 аркуш* |
| *Блок-схема алгоритму роботи додатку* | *2 аркуша* |

7. Дата видачі завдання « 17 » січня 2025 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти | Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти | Примітка |
|---|---|---|---|
| 1. | Аналіз існуючих систем | 10.03.2025 р. | |
| 2. | Постановка задачі, оформлення ТЗ | 15.03.2025 р. | |
| 3. | Розробка моделі компонента | 20.03.2025 р. | |
| 4. | Розробка структур даних | 25.03.2025 р. | |
| 5. | Розробка алгоритмів зв’язку та відображення | 30.03.2025 р. | |
| 6. | Програмування алгоритмів | 10.04.2025 р. | |
| 7. | Оформлення ПЗ | 17.04.2025 р. | |
| 8. | Попередній захист роботи | 23.05.2025 р. | |
| | | | |
| | | | |
| | | | |
| | | | |

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Минайленко Р.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Лісевич Д.С.
(прізвище та ініціали)

# АНОТАЦІЯ

**Лісевич Д.С. Програмне забезпечення системи "розумний замок" з автентифікацією даних через мобільний додаток. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.**

У даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення для системи «розумний замок» з автентифікацією користувачів за допомогою мобільного додатку.

Метою розробки є створення програмного забезпечення, що забезпечує безпечний доступ до системи замикання дверей за допомогою мобільного застосунку з функціями автентифікації.

Результатом роботи є програмна реалізація системи «розумний замок» з підтримкою зв'язку між мобільним додатком і сервером через протокол HTTP.

У процесі розробки виконано аналіз аналогічних рішень, описано апаратну та програмну частину системи. Особливу увагу приділено безпеці зберігання та передачі даних.

Розроблено зручний графічний інтерфейс користувача для мобільного додатку, що дозволяє здійснювати вхід до системи, відкриття замка та перегляд журналу подій.

Програма може використовуватися на мобільних пристроях з ОС Android, а серверна частина – на комп'ютерах архітектури IBM PC з ОС Windows або Linux.

Програму розроблено з використанням Python (Flask) для серверної частини та Kivy для мобільного додатку.

**Ключові слова:** розумний замок, мобільний додаток, автентифікація, Flask, Kivy

## ABSTRACT

**Lisevych D.S. Smart lock system software with data authentication via a mobile application. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.**

In this bachelor's graduation thesis, software has been developed for a smart lock system with user authentication via a mobile application.

The purpose of the development is to create software that ensures secure access to a door locking system using a mobile app with authentication features.

The result of the work is a software implementation of a smart lock system supporting communication between the mobile application and the server via the HTTP protocol.

During the development, an analysis of existing solutions was carried out, and both the hardware and software parts of the system were described. Special attention was paid to data storage and transmission security.

A convenient graphical user interface was developed for the mobile application, allowing users to log in, unlock the lock, and view the event log.

The program can be used on mobile devices running Android OS, while the server part runs on IBM PC architecture computers with Windows or Linux OS.

The software was developed using Python (Flask) for the server and Kivy for the mobile application.



**Keywords:** smart lock, mobile application, authentication, Flask, Kivy

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

TOTP – Time-based One-Time Password

LDAP – Lightweight Directory Access Protocol

2FA – Two-Factor Authentication

RFID – Radio-Frequency Identification

JWT – JSON Web Token

BLE – Bluetooth Low Energy

ORM – Object-Relational Mapping

QR – Quick Response

HTTP – Hypertext Transfer Protocol

REST – Representational State Transfer

DB – Data Base

PID – Process Identifier

SQLite – Structured Query Language Lite

JSON – JavaScript Object Notation

API – Application Programming Interface

# ВСТУП

**Актуальність теми.** У сучасному світі зростає потреба в безпечних, зручних та інтелектуальних системах контролю доступу. Звичні засоби – механічні ключі, кодові замки чи магнітні картки – часто є вразливими до втрати, копіювання або несанкціонованого застосування. У відповідь на ці виклики активно розвиваються системи «розумного замка», які використовують мобільні додатки для автентифікації користувачів і віддаленого керування доступом.

Застосування мобільного додатку у зв'язці з безпечним серверним з'єднанням відкриває нові перспективи для управління замками: видача тимчасових цифрових ключів, логування подій, контроль доступу в реальному часі та інше. Така інтеграція значно збільшує зручність для користувача та рівень безпеки системи. Актуальність теми зумовлена стрімким розвитком Інтернету речей(IoT), збільшенням попиту на smart-рішення у побуті та бізнесі, а також необхідністю створення надійного програмного забезпечення, що забезпечить збереження персональних даних і захист від несанкціонованого доступу.

**Мета та завдання дослідження.** Метою роботи є створення програмного забезпечення системи «розумний замок» з автентифікацією користувачів через мобільний додаток.

Для досягнення цієї мети були поставлені такі завдання:

- Здійснити огляд наявних рішень у сфері інтелектуальних замків та мобільної автентифікації;
- Розробити архітектуру системи, що включає клієнтський(мобільний) застосунок і серверний компоненти;
- Реалізувати мобільний додаток для керування доступом та авторизації користувача;
- Розробити серверну частину для зберігання, обробки запитів і перевірки автентичності;

- Провести тестування системи в різних сценаріях використання та проаналізувати її ефективність.

**Практична цінність здобутих результатів** полягає у можливості застосування розробленого програмного забезпечення для організації безпечного доступу до помешкань що збільшує рівень автоматизації та контролю. Запропонована система може бути впроваджена як у побуті (квартири, приватні будинки), так і в комерційній сфері (офіси, склади, готелі). Реалізовані рішення можуть також стати основою для подальшого розвитку більш складних IoT-систем.

Отже, розробка програмного забезпечення системи «розумний замок» з автентифікацією даних через мобільний додаток є актуальним завданням, що вимагає реалізації в рамках цієї випускної кваліфікаційної робити за першим (бакалаврським) рівнем вищої освіти.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

## 1.1 Призначення системи

Розроблена система призначена для безпечного, зручного та ефективного дистанційного управління електронним замком із використанням сучасних інформаційних технологій. Вона забезпечує автентифікацію користувача з використанням двофакторного методу (2FA), що суттєво підвищує рівень безпеки порівняно з традиційними підходами, які передбачають лише парольний захист.

Основним призначенням цієї системи є створення надійного та інтуїтивно зрозумілого інструменту для користувача, який дозволяє віддалено відкривати чи закривати замок, переглядати історію дій, а також гарантує, що доступ до пристрою мають тільки авторизовані особи. Система може застосовуватись у побутових умовах (наприклад, у розумному будинку), в офісних приміщеннях, у складських комплексах, у коворкінгах або навіть у готелях, де потрібне керування доступом до приміщень на основі автентифікації.

На відміну від звичайних механічних замків або простих електронних систем, які використовують лише код або RFID, запропонована система поєднує в собі переваги серверного управління та сучасних методів автентифікації, включаючи використання одноразових паролів (TOTP) на основі часу. Це дозволяє мінімізувати ризики злому пароля або викрадення даних авторизації.

Крім того, реалізована система включає логіку, яка автоматично блокує обліковий запис після кількох невдалих спроб входу, що унеможливлює brute-force атаки. Також реалізовано механізм таймерного автоматичного закриття замка після заданого інтервалу часу, що особливо корисно в умовах, коли користувач може забути замкнути приміщення власноруч.

Інтерфейс користувача реалізовано з використанням бібліотеки Kivy, що забезпечує кросплатформенність застосунку: користувачі можуть користуватися

системою як на мобільних пристроях, так і на комп'ютерах. Серверна частина, створена на Flask, відповідає за зберігання інформації про користувачів, автентифікацію, авторизацію, обробку запитів управління замком та збереження логів. Зв'язок між клієнтською та серверною частинами здійснюється через HTTP(S)-протокол із використанням JSON-формату для передачі даних.

Загалом, система виконує наступні основні функції:

– забезпечення безпечної реєстрації та входу до облікового запису з використанням двофакторної автентифікації;

– віддалене відкривання та закривання електрозамка;

– ведення журналу дій користувачів;

– автоматичне блокування доступу за підозрілих активностей;

– інтеграція з мобільними або стаціонарними пристроями користувача.

Завдяки використанню стандартних технологій та протоколів, система легко масштабується та інтегрується з іншими системами — наприклад, системами відеоспостереження, сигналізації чи централізованого контролю доступу.

Призначення системи також полягає у навчальній та дослідницькій корисності — вона демонструє принципи побудови клієнт-серверної архітектури, реалізацію авторизації через JWT, роботу з REST API, взаємодію з базами даних та практичне застосування TOTP для 2FA. Це робить систему цінною не лише в прикладному, а й в освітньому аспекті, зокрема для студентів та розробників, котрі вивчають основи кібербезпеки, розробки інтерфейсів та роботи з інтернетом речей.

Отже, розроблена система є універсальним рішенням, яке поєднує у собі безпеку, мобільність, зручність у користуванні та сучасні технології розробки програмного забезпечення.

Додатково, важливим аспектом призначення системи є збільшення рівня фізичної безпеки приміщень без необхідності у встановленні коштовного обладнання. В сучасних умовах зростаючих загроз кібератак та фізичного зламу,

традиційні замки вже не гарантують відповідного рівня захисту. Розроблена система дозволяє керувати доступом на основі цифрових токенів, які неможливо фізично скопіювати, на відміну від ключів чи магнітних карток.

Застосування TOTP як другої фази автентифікації суттєво підвищує стійкість до атак типу "перехоплення сесії" або "фішингу". Користувач генерує одноразовий пароль на своєму мобільному пристрої (наприклад, у додатку Google Authenticator), і цей пароль дійсний лише протягом короткого інтервалу часу (зазвичай 30 секунд). Навіть якщо зловмисник отримає доступ до логіна і пароля, без TOTP він не зможе потрапити в систему.

Ще одним ключовим призначенням системи є гнучке управління ролями та правами доступу. Адміністратор може надати тимчасовий або постійний доступ певним користувачам, що дозволяє, наприклад, впроваджувати систему у готельному секторі — клієнт отримує цифровий ключ, який діє лише протягом терміну бронювання. Після завершення терміну, доступ автоматично анулюється.

Система також має потенціал для подальшої інтеграції з хмарними сервісами, що дозволяє реалізувати зберігання журналів подій у хмарі, доступ до керування з будь-якої точки світу, та сповіщення про події (наприклад, спроби злому) в режимі реального часу через e-mail, Telegram або мобільні пуш-повідомлення.

Крім безпосереднього використання для контролю доступу, розроблена система може застосовуватись як навчальний стенд для студентів, які вивчають основи інформаційної безпеки, IoT, клієнт-серверну архітектуру та мобільну розробку. Вона демонструє практичні аспекти побудови безпечних рішень, що мають реальне прикладне значення.

Слід також зазначити, що система не потребує сталого інтернет-з’єднання між пристроєм користувача та замком. Всі критично важливі операції можуть бути реалізовані у локальній мережі, що забезпечує автономність рішення — у випадку аварійного відключення зовнішнього зв’язку або спроби DDoS-атаки, користувач все ще має змогу управляти замком у межах локальної

інфраструктури.

Отже, призначення системи виходить за межі звичайного механізму "відкривання/закривання" — вона є комплексною цифровою платформою безпечного доступу, яка може бути масштабована, модифікована та адаптована під різні задачі користувача: від контролю доступу до окремого приміщення до централізованої системи в будівлях чи установах.

### 1.2 Область застосування

Система електронного доступу з використанням тимчасових кодів має широкий спектр застосування в різних галузях, де важливо забезпечити контрольований, безпечний і гнучкий доступ до фізичних чи логічних об’єктів. Завдяки використанню тимчасових кодів, ця система помітно підвищує рівень безпеки, виключаючи можливість повторного використання одноразових паролів та мінімізуючи ризик несанкціонованого доступу.

Однією з ключових сфер застосування системи є офісні та корпоративні приміщення. У великих компаніях та організаціях з великою кількістю працівників контроль доступу до службових приміщень відіграє важливу роль. Запровадження системи TOTP дозволяє гнучко керувати правами доступу співробітників: наприклад, обмежувати доступ за часом, днем тижня або навіть конкретними подіями. Крім того, система забезпечує просту інтеграцію з наявними корпоративними інфраструктурами, такими як LDAP, Active Directory або внутрішні CRM-системи.

Ще однією важливою сферою є готельно-ресторанний бізнес (HoReCa). Готелі, хостели та гостьові будинки активно впроваджують системи безключового доступу для спрощення процесу заселення та підвищення зручності для клієнтів. За допомогою TOTP система може генерувати тимчасові коди для гостей, що діють лише протягом періоду бронювання. Це дозволяє повністю автоматизувати процес доступу до номерів без потреби фізичного

контакту з адміністратором, що особливо актуально в умовах пандемії або при обслуговуванні віддалених об’єктів.

В освітньому середовищі — в університетах, школах, лабораторіях — система може застосовуватися для керування доступом до комп’ютерних класів, дослідницьких приміщень або серверних кімнат. Завдяки можливості централізованого управління правами доступу, адміністрація навчального закладу може контролювати, хто, коли і до чого має доступ. Крім того, така система може використовуватися як частина навчального процесу в рамках дисциплін з інформаційної безпеки, програмування або проєктування систем автоматизації.

У сфері оренди майна — наприклад, оренда гаражів, складів, комірок, офісів або приватних будинків — система надає орендодавцям інструмент для забезпечення безпечного доступу без потреби передавати фізичні ключі. Користувач отримує TOTP-код через мобільний додаток, електронну пошту або SMS, і може використовувати його лише в обмежений період часу. Після завершення терміну оренди доступ автоматично блокується.

В умовах промислових підприємств, де часто існує розмежування зон з обмеженим доступом, система може застосовуватись для диференціації прав доступу співробітників до різних цехів, лабораторій, машинних відділень. Також важливим є захист від доступу сторонніх осіб або підрядників, які можуть мати тимчасовий дозвіл. Для них можуть генеруватись окремі одноразові коди, що не надають доступу до критичних ділянок підприємства.

У сфері охорони здоров’я система може бути інтегрована в роботу лікарень, приватних клінік, лабораторій, де потрібно регулювати доступ до аптек, складів медикаментів, кабінетів або архівів із персональними медичними даними. Забезпечення контролю доступу у цій галузі є критично важливим, оскільки стосується як безпеки обладнання, так і конфіденційності пацієнтів.

Система також є корисною для використання у сфері IT та дата-центрів, де часто потрібне багаторівневе управління доступом до серверних приміщень або

мережевих шаф. У поєднанні з системами моніторингу, журналюванням та відеоспостереженням система з ТОТР дозволяє створити потужну інфраструктуру інформаційної та фізичної безпеки.

Варто також зазначити можливість використання подібної системи в особистому побуті — для контролю доступу до квартир, будинків, гаражів, сховищ. Зокрема, інтеграція з системами "розумного дому" відкриває шлях до широкого застосування у приватних житлових об'єктах, де власник може видавати тимчасові коди для родичів, гостей або обслуговуючого персоналу (прибиральники, ремонтники тощо).

Ще одна цікава сфера застосування — тимчасові заходи або події, такі як конференції, виставки, форуми чи фестивалі, де необхідно обмежити доступ до певних зон (наприклад, прес-зони, VIP-зони, технічних приміщень). Завдяки використанню ТОТР можна уникнути втрат або підробок фізичних бейджів або перепусток, адже коди діють лише короткий проміжок часу і можуть бути перевірені в режимі реального часу.

Окремо варто відзначити потенціал використання в державних установах та органах місцевого самоврядування, де безпека інформації та фізичний доступ до службових приміщень мають велике значення. Система може бути інтегрована в загальну концепцію цифрової трансформації та використана для реалізації електронного документообігу, захищених архівів або обмеженого доступу до залів засідань.

Таким чином, система електронного доступу на базі ТОТР є універсальним рішенням, що може адаптуватися під потреби як великого підприємства, так і окремої особи. Її гнучкість, масштабованість і підвищена безпека роблять її ефективним інструментом у сучасному цифровому суспільстві.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

## 2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Системи електронного контролю доступу стали невід'ємною складовою сучасної інфраструктури безпеки в офісних центрах, державних установах, навчальних закладах, житлових комплексах та промислових підприємствах. Їх головне завдання полягає у забезпеченні обмеженого, контрольованого та зафіксованого доступу до визначених об’єктів або територій на основі попередньо встановлених прав ідентифікованих користувачів.

З розвитком цифрових технологій системи доступу еволюціонували від простих механічних рішень (ключі та замки) до високотехнологічних рішень, що включають використання електронних ідентифікаторів, біометричних параметрів, мобільних пристроїв, тимчасових кодів та мережевої інфраструктури. Ці системи дають змогу організаціям не тільки підвищити рівень безпеки, а й оптимізувати процеси управління персоналом, обліку робочого часу, аналітики відвідуваності та інших пов'язаних із доступом завдань.

До основних компонентів типових систем електронного доступу належать: контролери доступу, ідентифікатори (картки, брелоки, мобільні додатки, біометричні ознаки), зчитувачі, механізми замикання (електромагнітні або електромеханічні замки), програмне забезпечення для адміністрування системи, а також інтерфейси інтеграції з іншими системами підприємства (відеоспостереження, сигналізація, ERP, CRM тощо).

Серед технологічних трендів останніх років можна відзначити активну інтеграцію хмарних сервісів, що дозволяють керувати системою доступу віддалено з будь-якого пристрою з інтернет-з’єднанням, використання мобільних

ідентифікаторів (доступ через смартфон), підтримку тимчасових та одноразових кодів для гостей, а також застосування штучного інтелекту для виявлення аномальної поведінки користувачів.

**Класифікація розумних замків за принципами дії.** Сучасні розумні замки є важливою частиною систем «розумного дому» та широко застосовуються для забезпечення зручного й безпечного доступу до приміщень без звичайних ключів. В основі кожного різновиду «розумного замка» лежить принцип керування замком через цифрові технології, а не механічний вплив. Умовно всі такі пристрої можна класифікувати за типом каналу зв’язку, методом ідентифікації користувача, а також за архітектурою взаємодії з іншими компонентами системи.

**Bluetooth-замки.** Bluetooth є одним із найбільш поширених методів з'єднання розумного замка до мобільного пристрою. Завдяки малому енергоспоживанню (зокрема у версіях BLE), такі замки можуть працювати від батарейок тривалий час.

Ключова риса: замок автоматично з'єднується з телефоном користувача, коли він знаходиться в зоні дії сигналу (зазвичай до 10 метрів). Активація розблокування може бути як ручною (через застосунок), так і автоматичною (GeoUnlock).

Переваги:

– Відсутність потреби в Інтернеті;

– Порівняно висока безпека за рахунок належного шифрування.

Недоліки:

– Обмежена зона дії;

– Неможливість віддаленого керування без додаткового шлюзу.

**Wi-Fi-замки.** Wi-Fi-моделі дозволяють керувати замком через Інтернет з будь-якої точки світу. Така система зазвичай передбачає стале підключення до домашньої мережі Wi-Fi і серверної частини (хмарного сервісу чи власного сервера).

Переваги:

– Можливість віддаленого управління.

– Інтеграція з системами розумного дому (Google Home, Alexa, Apple HomeKit).

Недоліки:

– Високе споживання енергії — батарея зношується швидше.

– Залежність від стабільності Інтернет-з'єднання.

– Потенційні вразливості при неналежному шифруванні або вразливому API.

**NFC-замки.** Технологія Near Field Communication дозволяє відімкнути замок за допомогою близького дотику з авторизованим пристроєм (наприклад, смартфоном або NFC-карткою). Цей спосіб часто використовується в офісних будівлях чи в готельних замках.

Переваги:

– Швидке і безконтактне відмикання.

– Висока точність автентифікації.

Недоліки:

– Не всі смартфони підтримують NFC.

– Неможливість віддаленого доступу без додаткових способів.

– Вразливість до атак через клонування карток при низькому рівні шифрування.

**Біометричні замки.** У біометричних smart lock застосовується відбиток пальця, сканування обличчя або навіть розпізнавання голосу. Це один із найбільш зручних способів, адже користувачеві не треба мати при собі додаткові способи автентифікації.

Переваги:

– Високий рівень безпеки.

– Швидкий доступ без зайвих дій.

– Не потребує смартфона.

Недоліки:

– Більша вартість.

– Проблеми в роботі при пошкодженні сенсора або забрудненні (наприклад, після дощу).

– Вірогідність хибного спрацьовування або відмови.

**PIN-кодові замки.** Замки з цифровою клавіатурою дозволяють ввести певний шифр для відчинення дверей. Такі замки є розповсюдженим та зрозумілим способом,особливо в готелях та в інших приміщеннях.

Переваги:

– Простота у використанні.

– Можливість задавати тимчасові або індивідуальні шифри.

Недоліки:

– Коди можуть бути відгадані.

– Потреба ручного введення (не зручно для людей із вадами зору).

**Гібридні замки.** Деякі системи поєднують декілька способів доступу: наприклад, Bluetooth + біометрія, або NFC + PIN-код. Це дозволяє гнучко конфігурувати сценарії доступу для різних юзерів.

Такі рішення часто застосовуються в офісах, де працівники мають різні права доступу або в домівках з декількома мешканцями. Наприклад, адміністратор може відкривати замок через додаток, а дитина — за допомогою PIN-коду або мітки.

Ринок розумних замків активно розвивається, пропонуючи споживачам різні рішення з функціональністю, зосередженою на безпеці, зручності та інтеграції з іншими пристроями розумного дому. У цьому розділі розглянуто особливості, переваги та недоліки популярних комерційних систем від таких брендів, як August, Nuki, Yale, Xiaomi, Ultraloq, Samsung, Tedee та інші.

**August Smart Lock.** August Smart Lock — один з найпопулярніших товарів на ринку розумних замків в США. Його головна ідея полягає у збереженні механізму звичайного замка зовні, водночас оснащуючи внутрішню

частину дверей цифровим блоком керування. Завдяки цьому монтаж не потребує змін в конструкції дверей, що особливо практично для орендованих помешкань або офісів. Встановлення займає всього декілька хвилин, не вимагаючи професійного монтажу.

Керування замком здійснюється через Bluetooth або Wi-Fi. Підключення до мережі Wi-Fi реалізується за допомогою окремого пристрою — August Wi-Fi Bridge, який дозволяє користувачам дистанційно управляти замком, навіть перебуваючи в іншій країні. За допомогою мобільного додатку можна відкривати й закривати двері, отримувати сповіщення про кожен вхід та вихід, а також встановлювати тимчасові ключі для гостей, кур'єрів чиробітників.

August підтримує автоматичне замикання та відмикання, орієнтуючись на геолокацію смартфону власника. Це особливо корисно, коли користувач повертається додому із зайнятими руками — двері відчиняться автоматично. Безпека даних гарантується шифруванням AES 128-bit та TLS, а також двофакторною автентифікацією.

Замок сумісний із провідними голосовими асистентами — Amazon Alexa, Google Assistant та Apple HomeKit. Крім того, August інтегрується з Airbnb, дозволяючи хостам автоматично видавати цифрові ключі гостям на час бронювання. Завдяки великій кількості схвальних відгуків і визнанню на ринку, August Smart Lock вважається одним з еталонних прикладів сучасного смарт-безпеки.

**Nuki Smart Lock.** Nuki — австрійська марка, що здобула популярність у Європі завдяки своїй відкритості, легкості встановлення та широким функціональним можливостям. Розумні замки Nuki встановлюються без потреби заміни механізму — просто монтуються поверх вже наявного циліндра з внутрішнього боку дверей. Користувач при цьому має змогу зберегти можливість відкриття звичайним ключем ззовні.

Система підтримує Bluetooth та Wi-Fi (через Nuki Bridge), щодаємогуяк локальне, так і віддалене керування. За допомогою мобільного застосунку Nuki

користувач може відмикати/замикати двері, створювати тимчасові ключі, задавати розклади доступу, а також переглядати детальний лог усіх дій — хто й коли заходив чи виходив. Це дозволяє використовувати Nuki не тільки в приватних будинках, а й у готелях або офісах.

Однією з фішок є Auto Unlock — двері автоматично відчиняються при наближенні власника. Крім того, замок сумісний із Airbnb, IFTTT, Apple HomeKit, Amazon Alexa, Google Assistant, а також має відкритий API для розробників. Компанія регулярно оновлює програмне забезпечення та підтримує зворотний зв'язок із користувачами через відкриті форуми та спільноти.

З точки зору безпеки, Nuki застосовує технології банківського рівня: end-to-end шифрування, двофакторну автентифікацію та постійний моніторинг на предмет вразливостей. Завдяки великому акценту на простоті, безпеці й гнучкості, Nuki є одним з найкращих рішень для європейських користувачів.

**Yale Linus / Assure.** Yale Linus — один із найдавніших брендів вобласті замків, збільш ніж 180-річною історією. У сьогоденному світі фірма активно інтегрується у цифрову екосистему безпеки. Завдяки партнерству з August, Yale випустила лінійку розумних замків Linus (для європейського ринку) та Assure (для американського). Ці рішення поєднують надійність класичного механізму з новаторськими цифровими функціями.

Yale Linus — це накладний модуль, що ставиться на внутрішню частину дверей і підтримує управління через Bluetooth або Wi-Fi (через Yale Connect Bridge). Assure серії є більш комплексними: повноцінна заміна замка з вмонтованою PIN-панеллю, можливістю застосування карток доступу, біометрії, смартфона або механічного ключа.

Керування доступом здійснюється через додаток Yale Access: тут можна видавати віртуальні ключі, встановлювати графіки, отримувати сповіщення про вхід і вихід, а також контролювати статус дверей (відчинено/зачинено). Замки сумісні з системами Amazon Alexa, Apple HomeKit, Google Assistant, SmartThings, а також підтримують інтеграцію з Airbnb та іншими сервісами.

Особливу увагу бренд приділяє фізичній безпеці — використовуються замки сертифікації ANSI Grade 1/2, які пройшли суворі тести на злам та зносостійкість. Рішення від Yale — це синтез традиційної надійності з новітніми цифровими технологіями.

**Xiaomi Smart Door Lock Series.** Xiaomi активно просуває новації в секторі розумного дому, і її серія Smart Door Lock — одна з найзатребуваніших у категорії бюджетних, але функціонально насичених смарт-замків. Фірма пропонує декілька моделей, зокрема Xiaomi Smart Door Lock E, Xiaomi Push-Pull Pro та Xiaomi Face Recognition Smart Door Lock.

Основною перевагою Xiaomi є інтеграція зі смарт-екосистемою Mi Home. Замки підтримують шість або більше способів автентифікації: відбитки пальців (з використанням 3D-оптичного сенсора нового покоління), PIN-код, NFC-картки,Bluetooth-ідентифікацію, мобільний додаток та навіть розпізнавання обличчя в топових моделях. Наприклад, Xiaomi Push-Pull Pro обладнано AI-камерою з інфрачервоним сканером, що дозволяє здійснювати біометричну ідентифікацію користувача за обличчям навіть у темряві.

Велика увага приділена безпеці. Xiaomi використовує захищений чип для зберігання біометричних даних, шифрування Bluetooth-з'єднання (як правило, через BLE), а також спеціальний механізм захисту від підбору паролів. Замок автоматично блокується після кількох невдалих спроб введення паролю, а користувач отримує миттєве сповіщення в мобільному застосунку.

Також, Xiaomi замки дозволяють легко генерувати одноразові коди для гостей, інтегруються з камерами та відео домофонами, а енергія зберігається на батареях, яких вистачає до року роботи. Часто передбачено порт USB Type-C для екстреної підзарядки. Завдяки доброму співвідношенню ціна/функції, вони стали популярними не лише в Китаї, а й на європейському ринку.

**Ultraloq by U-tec.** Ultraloq — це преміальні смарт-замки американського бренду U-tec, що спеціалізується на створенні гібридних систем доступу. Найпопулярніші варіанти — Ultraloq U-Bolt Pro, Ultraloq UL3 BT та Ultraloq U-

Bolt WiFi. Вони вирізняються високою модульністю та підтримкою до шести методів автентифікації: PIN-код, відбиток пальця, смартфон через Bluetooth/Wi-Fi, фізичний ключ, голосовий доступ через Alexa/Google Assistant та тимчасовий код.

Ultraloq U-Bolt Pro підтримує 360° зчитування відбитків пальців, що забезпечує швидке й надійне розпізнавання за 0.3 секунди з точністю до 99,8%. Дані зберігаються в локальній зашифрованій пам'яті, що підвищує захист від віддалених атак. Крім того, пристрій підтримує Auto-Lock (автоматичне блокування після виходу з дому) та Auto-Unlock (розпізнає телефон власника при наближенні до дверей).

Wi-Fi модель підтримує повноцінний віддалений контроль через хмарний сервіс U-tec Cloud. Можна надсилати коди доступу віддалено, переглядати журнали відчинення замка та керувати декількома замками в одному додатку. Також реалізовано інтеграцію з IFTTT, SmartThings, Amazon Alexa.

Модулі легко монтуються, підходять для більшості американських стандартів дверей і мають ступінь захисту IP65 (захист від пилу й води). Працюють від 4 AA-батарей, що забезпечують близько 8000 циклів відмикання.

**Samsung Smart Door Locks.** Samsung пропонує широкий модельний ряд розумних замків, зокрема SHP-DP609, SHP-DR708, SHP-DH538ітаке інше. Ці прилади орієнтовані на преміум-сегмент і вирізняються елегантним дизайном, високим рівнем безпеки та функціональністю.

Більшість моделей підтримують автентифікацію за допомогою PIN-коду, RFID-карти, відбитків пальців (у моделях з біометрією), мобільного застосунку через Bluetooth/Wi-Fi, а також механічного ключа. Наприклад, SHP-DP609 оснащений 5 способами автентифікації та має хмарну інтеграцію з SmartThings, що дає змогу керувати замком віддалено через смартфон.

Окрему увагу приділено безпеці: є функції Anti-Panic Exit, Fire Detection Sensor, Intrusion Alarm, котрі захищають не тільки від злому, а й від внутрішніх небезпек, як-от пожежа. Samsung реалізує технологію Random Security Code, коли

перед введенням PIN-коду необхідно натиснути два довільних числа, щоб сховати справжній код від сторонніх очей.

Завдяки вбудованому Wi-Fi (у моделях Pro) можна в режимі реального часу отримувати сповіщення, переглядати журнал подій, надсилати тимчасові ключі гостям. Живлення зазвичай забезпечують 8 батарей типу AA, з автономністю до 12 місяців.

Моделі Samsung мають сертифікати безпеки класу Grade 1 (ANSI/BHMA), що свідчить про найвищий рівень механічної та електронної безпеки.

**Tedee Smart Lock.** Tedee — це польський бренд, який виготовляє високо технологічні смарт-замки для європейського ринку. Флагманський виріб— Tedee Lock Pro — відрізняється компактним формфактором, якісним металевим корпусом та повною підтримкою віддаленого керування.

Tedee Lock інтегрується з Apple HomeKit, Google Home, Amazon Alexa, а також підтримує стандарт Matter (через Bridge), що робить його одним з найперспективніших рішень у сфері взаємодії. Управління здійснюється через мобільний додаток Tedee (iOS/Android) або через веб-інтерфейс. Замок має функції Auto-Unlock, Auto-Lock, журнал подій, віддалене надання доступу, Push-сповіщення.

Tedee встановлюється поверх наявного євро профілю циліндра (наприклад, GERDA або M&C), і це дозволяє його швидко змонтувати без заміни всієї замкової системи. Він використовує Bluetooth LE для зв’язку з телефоном і Wi-Fi через окремий Tedee Bridge для віддаленого доступу.

Щодо безпеки — Tedee використовує 256-бітне шифрування, сертифікований механізм зберігання ключів, регулярні OTA-оновлення, і навіть проходив незалежне тестування німецькою лабораторією AV-TEST. Батареї вистачає до 6 місяців ,заряджання відбувається через магнітний конектор USB-C без демонтажу пристрою.

Замок ідеально підходить для орендодавців, готелів, а також приватних осіб, яким потрібен елегантний та надійний смарт-замок з мінімальним

втручанням в інфраструктуру дверей.

## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

На етапі планування проєкту в першу чергу було визначено робоче середовище розробника. У якості інтегрованого середовища (компілятора й редактора) обрано PyCharm, бо воно забезпечує повноцінну підтримку Python-проєктів, інтеграцію з системою контролю версій Git, зручні інструменти налагодження веб-сервісів Flask і відлагодження мобільних застосунків Kivy, а також вбудовані засоби роботи з віртуальними середовищами та базами даних. Використання PyCharm дає змогу швидко налаштовувати простір, автоматично встановлювати потрібні пакети та мінімізувати кількість помилок завдяки підсвічуванню синтаксису й статичному аналізу коду.

Розробка системи для розумного замка є важливим та складним процесом, котрий потребує не лише інноваційних підходів,ай ретельного підбору технологій та інструментів для гарантування високої ефективності роботи системи. Вибір відповідних засобів розробки є критично важливим етапом, оскільки від цього залежить як якість самої системи, так і зручність її застосування кінцевим користувачем. У цьому контексті доречним є застосування Flask як серверного фреймворку та Kivy для створення мобільного додатку, що разом забезпечить належну функціональність і надійність системи.

Flask є потужним, але водночас легким і гнучким фреймворком для створення веб-додатків на Python. Він дозволяє швидко налаштувати сервер і створити API, через котрий мобільний додаток може взаємодіяти із сервером для виконання необхідних операцій, таких як перевірка доступу до замка, реєстрація користувачів, обробка даних про відвідування тощо. Flask дозволяє зосередитись на функціональності замка, мінімізуючи зайву складність при налаштуванні сервера. Це надзвичайно важливе для успішного розвитку системи, оскільки в

умовах швидкої зміни вимог та потреб, фреймворк має забезпечувати високу гнучкість і простоту в розробці.

Однією з основних причин вибору Flask для серверної частини є його здатність ефективно працювати з базами даних та опрацьовувати численні HTTP запити. У цьому проекті необхідно зберігати різноманітні дані, такі як історія доступів до замка, відомості про користувачів та їхні налаштування. Flask має вбудовані інструменти для інтеграції з популярними базами даних, зокрема SQLite, PostgreSQL або MySQL, що дозволяє створювати надійні та масштабовані сховища даних. Це також дає змогу підключати додаткові механізми автентифікації та авторизації, забезпечуючи високий рівень безпеки для користувачів системи. Вибір Flask забезпечує швидкість розробки завдяки його спрощеній архітектурі, а також можливість реалізації функцій, що потребують високої продуктивності.

Щодо вибору мобільного фреймворку, то Kivy є чудовим варіантом для розробки кросплатформних додатків. Це відкритий фреймворк для створення графічних інтерфейсів користувача на Python, який підтримує роботу на різних операційних системах, таких як Android, iOS, Windows, Linux та macOS. Це дає змогу створювати універсальні додатки, що працюють на широкому спектрі пристроїв, і забезпечують високу зручність для кінцевого користувача. У випадку розробки мобільного додатку для розумного замка Kivy дозволяє створити інтуїтивно зрозумілий та функціональний інтерфейс, зокрема інтерактивні елементи керування, кнопки для активації/деактивації замка, перегляд історії доступу, налаштування користувача та інші важливі функції.

Однією з ключових переваг Kivy є підтримка сенсорних екранів та інших ввідних пристроїв, що дозволяє створювати додатки з багатим інтерфейсом користувача, де кожен елемент інтерактивний та реагує на дії користувача. Це особливо важливим є для мобільних додатків, де простота взаємодії з користувачем є головною вимогою. Завдяки великій кількості вбудованих віджетів і можливості налаштовувати їх зовнішній вигляд та поведінку, можна

створити не тільки функціональний, алей естетично привабливий інтерфейс.

Вибір Python як основної мови програмування для проєктує логічним, оскільки ця мова володіє рядом переваг, таких як простота синтаксису, велика кількість бібліотек і фреймворків для різноманітних задач, а також активна спільнота розробників, яка постійно створює нові рішення. Python також є однією з найбільш популярних мов програмування для розробки веб-систем та мобільних додатків, що значно полегшує пошук розв'язань для специфічних проблем, таких як обробка даних, забезпечення безпеки та інтеграція з іншими системами. Окрім того, Python є відмінним вибором для створення багатофункціональних додатків завдяки своїй підтримці різних бібліотек для роботи з мережею, графічними інтерфейсами, базами даних та іншими компонентами.

### 2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, що призначене для керування системою розумного замка з можливістю дистанційного доступу через мобільний застосунок. У системі використовується серверна частина, реалізована з використанням Flask, та клієнтський мобільний додаток, створений за допомогою Kivy. Головною метою розробки є забезпечення зручного, надійного та безпечного управління електронним замком користувачем у режимі реального часу.

У процесі створення випускної кваліфікаційної праці необхідно виконати наступний обсяг задач:

а) здійснити аналіз наявних рішень у сфері "розумних замків" та систем доступу з керуванням через мобільні пристрої. Провести порівняння функціональних можливостей, переваг та недоліків таких рішень. Результати аналізу врахувати при розробці власної системи;

б) обґрунтувати вибір загальної архітектури системи. Розробити функціональну модель взаємодії серверної частини (Flask) з мобільним додатком (Kivy), схему передачі даних, обробки запитів, зберігання інформації про користувачів та події доступу. Передбачити можливість масштабування системи та забезпечення її безпеки;

в) реалізувати програмне забезпечення системи, яке включає створення API для обробки запитів на сервері, механізми аутентифікації, управління замком, а також розробку повнофункціонального інтерфейсу користувача у мобільному додатку. Розробити блок-схеми алгоритмів основних програмних модулів;

г) організувати інтерфейс мобільного додатку таким чином, щоб користувач мав змогу легко здійснювати всі потрібні дії: відкривати та закривати замок, переглядати історію доступів, налаштовувати параметри користувача. Передбачити повідомлення про помилки, некоректні дії та виняткові ситуації;

д) підготувати методичні рекомендації щодо впровадження системи у побутове або офісне середовище. Розробити інструкції щодо встановлення та використання, а також опис можливих сценаріїв роботи системи;

е) здійснити оцінку економічної доцільності впровадження системи розумного замка. Порівняти витрати на її реалізацію з аналогічними рішеннями на ринку. Провести попередній розрахунок вартості компонентів і розробки;

ж) сформувати базові заходи щодо безпечної експлуатації системи, зокрема при роботі з електроживленням, встановленні замка, використанні додатку в публічних мережах. Врахувати питання кібербезпеки та захисту особистих даних користувача;

з) підготувати висновки щодо реалізації проекту, ефективності створеного рішення, його функціональності, перспектив удосконалення та можливості інтеграції з іншими системами "розумного дому".

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

У процесі розробки цієї дипломної роботи було створено повнофункціональну систему, яка поєднує мобільний додаток, реалізований з використанням фреймворку Kivy, та серверну складову, розроблену на основі Flask. Головне призначення системи — гарантування безпечного доступу до електронного замка з можливістю автентифікації користувачів та впровадженням двофакторної автентифікації.

Система складається з трьох головних компонентів:

– Мобільний застосунок (написаний з використанням Kivy) — графічний інтерфейс користувача, що дає доступ до функцій входу, реєстрації та керування замком.

– Серверна частина (Flask) — обробляє запити від клієнтів, забезпечує зберігання та перевірку облікових даних користувача, а також втілює логіку двофакторної автентифікації.

– База даних (SQLite) — використовується для збереження облікових записів юзерів.

Користувач взаємодіє з мобільним застосунком, котрий у свою чергу надсилає HTTP-запити на сервер. Серверна частина обробляє ці запити, проводить перевірку даних, генерує відповіді та надсилає їх назад клієнтові. У разі успішного входу юзер дістає доступ до екрана керування замком.

**Проблематика**

У зв’язку з активним розвитком цифрових технологій, бездротових мереж і мобільних пристроїв, у буденному житті з’являється дедалі більше пристроїв, що поєднують фізичну інфраструктуру з інформаційними технологіями. Одним із прикладів таких рішень є системи керування доступом — так звані "розумні

замки", які дозволяють користувачеві дистанційно відмикати або замикати двері, контролювати історію входів, а також налаштовувати параметри доступу для інших осіб. Однак, незважаючи на потенційну зручність таких рішень, їхнє впровадження супроводжується низкою практичних і технічних проблем, які обумовлюють потребу створення нового, більш продуманого підходу до розробки подібних систем.

Найперше, що потребує уваги — це рівень безпеки та надійність автентифікації. Багато сучасних пристроїв мають вразливості, які дозволяють обійти захист через слабкі паролі, відсутність шифрування або використання небезпечних протоколів зв'язку. Такі замки часто не передбачають жодного механізму верифікації користувача окрім вводу коду, що значно полегшує доступ до системи для сторонніх осіб. В умовах, коли безпека житла чи офісу має вирішальне значення, це є неприпустимим. Саме тому важливо реалізувати систему автентифікації, яка забезпечить надійний захист, наприклад, через перевірку введених даних на боці сервера, передачу інформації в зашифрованому вигляді та можливість керування правами доступу з мобільного додатку.

Окрім безпеки, важливим чинником є зручність використання. Низка наявних на ринку пристроїв або надто складні для звичайного користувача, або обмежені у своїй функціональності. Скажімо, деякі системи мають лише веб-інтерфейс чи функціонують виключно через Bluetooth, що обмежує дистанцію доступу. Водночас, інтеграція з мобільним додатком дозволяє значно розширити функціональні можливості — користувач може взаємодіяти із замком на відстані, переглядати історію входів та змінювати налаштування через інтуїтивно зрозумілий інтерфейс. Саме це й було враховано при проєктуванні системи, описаної у цій дипломній роботі.

Система складається з двох основних частин — серверної частини, реалізованої на базі фреймворку Flask, та клієнтського мобільного додатку, розробленого за допомогою бібліотеки Kivy. Сервер виконує функції обробки запитів, перевірки автентичності користувача, зберігання інформації у базі даних

та передачі команд до пристрою замка. Мобільний додаток, у свою чергу, забезпечує зручний інтерфейс користувача, через який можна здійснювати вхід до системи, надсилати запити на відмикання замка, отримувати повідомлення про статус операцій та переглядати лог дій. Такий підхід дозволяє розподілити навантаження між клієнтом і сервером, підвищити безпеку та зробити систему масштабованою для подальшого розширення.

Ще однією перевагою створеного рішення є можливість гнучкого налаштування прав доступу. Наприклад, в майбутньому можна реалізувати рольову модель, яка дозволить надавати тимчасові права доступу іншим юзерам, що є корисним у випадках оренди житла або роботи з кількома працівниками в офісі. Також є змога інтегрувати додаткові рівні безпеки, зокрема двофакторну автентифікацію, розпізнавання обличчя, геолокацію та інші сучасні технології.

Система функціонує таким чином: користувач запускає мобільний додаток, вводить свої облікові дані, котрі передаються до серверної частини. Flask-сервер здійснює перевірку даних, звертаючись до бази даних. У разі успішної автентифікації сервер формує відповідь та відправляє її клієнту. Далі користувач має здатність відправляти запити на відмикання замка, які також проходять через сервер. Сервер, у свою чергу, надсилає відповідну команду до пристрою замка (через локальний інтерфейс, API чи інші канали передачі), після чого оновлює статус у базі даних і повідомляє користувача про результат операції.

Отже, система "розумного замка" з автентифікацією через мобільний додаток поєднує в собі гнучкість, зручність та високий рівень захисту. Вона розроблена з урахуванням актуальних потреб користувачів і можливостей сучасних технологій. Основна мета розробки — забезпечити безпечний та зручний доступ до приміщень з використанням мобільного пристрою та централізованого серверного контролю. Отримане рішення може бути застосоване як у побуті, так і в комерційних приміщеннях, з можливістю масштабування та доопрацювання під специфічні потреби юзера.

**Шляхи рішення завдання**

Щоб створити систему «розумного замка», що буде зручною і для користувача, і для подальшого вдосконалення, спершу потрібно було визначитись, як конкретно її реалізовувати. Завдання було нелегке, бо хотілося, аби все функціонувало стабільно, безпечно та ще й мало сучасний вигляд. Тому я вирішив розділити систему на дві основні частини: серверну й мобільну. Це дозволяє ліпше організувати роботу програми, а також легше оновлювати окремі елементи, якщо буде потрібно.

Почну з мобільного додатка. Зараз майже кожна людина постійно використовує телефон, тому логічно було зробити саме мобільний додаток, через який можна буде відмикати й замикати замок. Для цього я використав Kivy — це така бібліотека на Python, що дозволяє створювати програми, котрі можуть працювати і на Android, і на інших платформах. Мені це підійшло, бо я писав на Python, а ще Kivy досить проста у використанні, тож не треба було вивчати важкі мови для Android-розробки.

Мобільний додаток дозволяє користувачеві увійти в акаунт, надіслати запит на відчинення або замикання замка, а також отримати сповіщення,чи замок дійсно відчинився. Він простий у використанні, і я намагався зробити його максимально зрозумілим навіть для тих, хто не надто добре розуміється в техніці.

Друга частина — це сервер, котрий я написав на Flask. Це дуже легкий і зручний веб-фреймворк для Python. Він добре підходить для таких проєктів, бо дає змогу швидко створити API, через яке мобільний додаток буде обмінюватись даними із сервером. Наприклад, коли користувач натискає кнопку "Відкрити замок" у додатку, ця команда летить на сервер, і вже сервер вирішує, чи можливо це зробити, перевіряє логін, пароль, і надсилає команду замку.

Щоб зберігати облікові записи користувачів та історію операцій, я використав SQLite. Це така легка база даних, що зберігається без посередньо у вигляді файлу. Її не потрібно окремо встановлювати або запускати, і для цього проєкту її можливостей цілком вистачає. Якщо у майбутньому знадобиться щось

серйозніше, можна буде перейти на більш потужну базу, наприклад, PostgreSQL чи MySQL.

Ще важливий аспект — це автентифікація. Було надзвичайно важливо зробити так, щоб тільки дозволені користувачі могли керувати замком. Тому я реалізував авторизацію через логін та пароль. Ці дані передаються на сервер, де перевіряються, і лише якщо все співпадає, тоді виконується команда. У майбутньому можна буде додати більш надійний варіант авторизації, наприклад, через SMS-код або спеціальний токен.

Окрема частина — це комунікація із самим замком. Після того як сервер підтверджує, що користувач має право щось зробити, він надсилає сигнал на замок — наприклад, через мережу або по Bluetooth. У проєкті поки що реалізовано симуляцію, але в реальних умовах можна підключити мікроконтролер типу ESP32, який буде слухати команди з сервера і керувати мотором замка.

Я теж намагався будувати систему так, аби її легко було розширити. Скажімо, можна буде додати ще один замок, надати доступ кільком особам, зробити розклад доступу — все це можна втілити, просто трохи допрацювавши серверну частину та мобільний інтерфейс. Тобто, структура системи незамкнена,а така, що дозволяє додавати нові функції в майбутньому.

Отже, загалом я вирішив йти способом розділення системи на клієнт (мобільний додаток) і сервер, між котрими відбувається обмін через API. За для цього застосував Python, Flask і Kivy, бо ці інструменти прості, зручні та добре пасують для такого проєкту. Усе це дало змогу зробити робочу, зрозумілу та готову до подальшого розвитку систему.

### 3.2 Розробка структурної схеми

Для кращого розуміння принципу функціонування такої системи, доцільно розглянути її структурну схему, яка умовно розділяється на дві основні частини:

клієнтську та серверну. Клієнтська частина — це все те, з чим безпосередньо взаємодіє користувач. Це графічний інтерфейс, клавіші, поля введення, сповіщення, а також механізм обробки дій користувача. Завдання цієї частини — зробити процес управління замком максимально доступним, зрозумілим і комфортним. Вона має бути адаптована під різні пристрої, мати приємний візуальний вигляд і швидко реагувати на дії.

На початку роботи з системою користувач відкриває застосунок і потрапляє на початкову сторінку, де запропоновано вибрати між входом та реєстрацією. Якщо це новий користувач, то він обирає реєстрацію, вводить необхідні дані, такі як логін та пароль. Ці дані зберігаються на сервері для подальшої ідентифікації. У випадку, якщо користувач вже зареєстрований, він переходить до авторизації, вводить свій логін і пароль. У разі правильного введення даних, система допускає його до головного вікна програми. Якщо ж введені дані є неправильними, система повідомляє про помилку, та користувач може повторити спробу. Також реалізовано ліміт на кількість спроб входу, не більше п’яти, після чого доступ тимчасово блокується на 30 секунд — це підвищує безпеку.

Після вдалого входу користувач потрапляє до основного інтерфейсу управління. У найпростішому вигляді це сторінка з однією головною кнопкою “відчинити замок”. При натисканні цієї кнопки утворюється запит, котрий відсилається до серверної частини. На цьому етапі відбувається обмін інформацією між клієнтським додатком і сервером. Важливо, щоб цей процес був захищений — з використанням шифрування або токенів доступу, адже йдеться про управління реальним фізичним пристроєм.

Серверна частина приймає запит, перевіряє його правдивість. Це означає, що сервер мусить переконатися у тому, що запит справді надійшов від авторизованого користувача, який має право керувати замком. Якщо всі перевірки пройдено успішно, сервер формує спеціальну команду, яка відсилається на апаратний пристрій — сам замок або контролер, що ним

управляє. Відповідно до отриманого сигналу, замок виконує дію — відчиняється або зачиняється.

Наголосити необхідно, що власне сервер бере на себе ключове навантаження в плані логіки, безпеки й контролю. Він повинен фіксувати всі події, зберігати відомості про дату й час доступу, вести журнал спроб входу, а також забезпечувати надійну автентифікацію користувачів. Завдяки такій організації, система стає не тільки зручною, але й безпечною.

У реалізації цього проєкту клієнтська частина створена за допомогою фреймворку Kivy, що є універсальним інструментом для побудови графічного інтерфейсу у Python. Він дає змогу легко створювати багато віконні додатки з різними елементами управління: кнопками, текстовими полями, лейблами, повідомленнями та інше. Завдяки цьому створено інтерфейс, котрий зручно адаптується під різні пристрої й забезпечує плавну взаємодію з користувачем.

Стосовно серверної частини, вона реалізована на основі Flask — мікро фреймворку для Python, що чудово підходить для створення веб-серверів. Flask дозволяє легко налаштовувати обробку HTTP-запитів, управляти сесіями користувачів, здійснювати перевірку автентичності, а також відповідати на запити клієнтів. Його гнучкість та простота роблять його відмінним рішенням для невеликих, але функціонально насичених проєктів. У нашій системі він відповідає за усі процеси: збереження та перевірку даних, обробку логіки дій, а також взаємодію з фізичним замком.

Фізичний пристрій, котрий виконує роль замка, у даному випадку може бути реалізований на базі реле або сервоприводу, підключеного до контролера типу Arduino або Raspberry Pi. Саме цей пристрій отримує команду з сервера у вигляді сигналу, після чого змінює своє положення (наприклад, обертає серводвигун), що призводить до фізичного відкриття або закриття замка. При цьому дуже важливо, щоб зв’язок між сервером і замком був надійним та швидким, щоб уникнути затримок або збоїв у роботі.

Отже, вся система діє за таким алгоритмом: користувач запускає застосунок – вибирає вхід або реєстрацію – вводить відомості – система перевіряє дані – при вдалій перевірці користувач дістає доступ до головного інтерфейсу – натискає кнопку для управління замком – формується запит до сервера – сервер перевіряє запит – відправляє команду замку – замок виконує операцію. Цей процес може тривати менше секунди, проте включає у себе функціонування багатьох компонентів, які мусять бути чітко синхронізовані.

Підсумовуючи, розроблена система демонструє високий рівень інтеграції між програмним забезпеченням та фізичними приладами. Вона може бути використана в побуті, офісах, готелях чи інших об’єктах, де потрібен безпечний та зручний доступ. Крім того, така архітектура є гнучкою до змін — у майбутньому її легко можна розширити, додавши нові функції: сповіщення, біометричну авторизацію, управління голосом чи інтеграцію з іншими системами безпеки. Весь потенціал такої системи залежить лише від фантазії і потреб користувачів, а також від бажання розробників удосконалювати її далі.

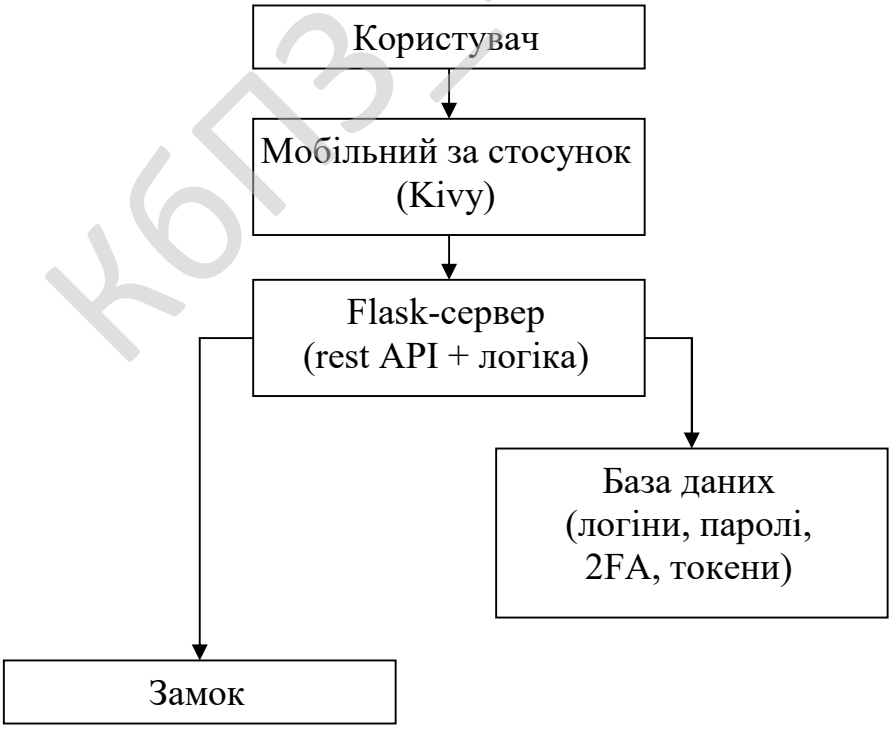



Рисунок 3.1 – Структурна схема роботи системи

Ось структурна схема системи віддаленого керування замком, представлена поетапно та по пунктах:

1. Користувач:

– Користувач є головним учасником системи, який взаємодіє з мобільним додатком для керування замком.

– Для авторизації користувач вводить свій логін, пароль та, запотреби, одноразовий код підтвердження (OTP), що створюється системою для забезпечення двофакторної автентифікації (2FA).

2. Мобільний додаток (Kivy):

– Мобільний додаток є головним інтерфейсом для користувача. Він дає змогу реєструватися в системі, виконувати авторизацію, налаштовувати 2FA, а також управляти замком (відчиняти або замикати його).

– Після введення даних користувачем додаток надсилає HTTP-запити до сервера, що можуть бути запитами на реєстрацію, авторизацію або виконання команд для управління замком.

3. Flask-сервер:

– Сервер, втілений на основі Flask, відіграє роль ключової логіки для обробки запитів користувача. Він відповідає за автентифікацію, перевірку коректності введених даних (логін, пароль), а також за виконання процедури двофакторної автентифікації.

– Сервер теж здійснює взаємодію з базою даних задля збереження даних користувачів, паролів, секретів для 2FA та інших необхідних даних.

– Сервер приймає запити від мобільного застосунку через REST API (наприклад, запити на реєстрацію, авторизацію, налаштування 2FA та управління замком).

4. База даних:

– База даних призначена для зберігання відомостей про користувачів, їхні паролі, секрети для двофакторної автентифікації та токени для сесій. Вона є важливим складником для забезпечення цілісності даних та ефективного

керування доступом до системи.

– Доступ до бази даних реалізується через сервер, який отримує необхідні дані для авторизації користувачів та підтвердження їх прав доступу.

5. Емулятор замка / пристрій:

– Цей елемент відповідає за фізичне виконання наказу на замок: відчинення або замикання. У справжній системі замок може бути з'єднаний з сервером через спеціальне устаткування (наприклад, через Wi-Fi чи Bluetooth), проте в межах цієї системи він представлений як емуляційний пристрій.

– Емулятор замка отримує команди на керування замком, отримані від сервера, та реалізує відповідну операцію, наприклад, відмикає чи замикає замок.

Ця структура ілюструє головні фази взаємодії користувача з системою, починаючи з мобільного додатку до серверної обробки запитів та керування фізичними приладами. Усі елементи системи діють узгоджено, щоб забезпечити стабільний доступ і контроль над замком, збільшуючи ступінь безпеки завдяки двофакторній автентифікації.

### 3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.




Рисунок 3.2 – Функціональна схема роботи системи

Функціональна схема показує загальну логіку роботи програмного комплексу, створеного для гарантування безпечної авторизації користувачів з використанням двофакторної автентифікації (2FA) та функціоналу автоматичного

блокування сесії користувача (LockScreen). Вона є важливим елементом пояснювальної записки, бо відображає основні програмні процеси, що втілюються під час взаємодії користувача з клієнтською частиною програми та сервером.

Робота системи стартує із запуску клієнтського застосунку. У цей момент відбувається ініціалізація інтерфейсу, створення з'єднання до серверної частини, перевірка конекту та готовності компонентів. Якщо з’єднання із сервером налагодити не вдалося, користувач отримує відповідне сповіщення, і програма завершує роботу або переходить в режим повторної спроби підлкючення.

Після вдалої ініціалізації, користувач бачить основне вікно з комірками для введення логіна та пароля. Після натискання кнопки «Вхід», введені дані передаються на сервер через зашифрований канал. На сервері відбувається перевірка відповідності облікових даних записам у базі.

У випадку хибного введення, користувач сповіщається про помилку, та дається можливість повторної спроби. У разі кількох підряд неправильних спроб входу — обліковий запис може бути тимчасово заблокований, аби запобігти атакам перебором паролів.

У випадку вдалої перевірки логіна і пароля, сервер створює одноразовий код автентифікації (OTP — One Time Password) та відправляє його користувачу через додатковий канал (мобільний додаток-генератор кодів Google Authenticator).

Користувач вводить отриманий код у відповідне поле. Код передається на сервер, де відбувається його перевірка. Якщо код вірний і в нього не вийшов час дії — користувача успішно автентифіковано, і він переходить до основного інтерфейсу програми.

У разі некоректного коду або перевищення дозволеної кількості спроб — користувач перенаправляється на початковий етап входу або блокується.

Після проходження автентифікації, користувач одержує доступ до головного інтерфейсу програми. Тут він може працювати з функціоналом згідно з

наданими правами доступу. Програма відстежує активність користувача — рух миші, натискання клавіш та таке інше.

У разі виявлення бездіяльності протягом визначеного часу (наприклад, 5 хвилин), система автоматично активує режим блокування (LockScreen). Для продовження роботи користувач мусить повторно пройти автентифікацію, але без потреби введення логіна — досить повторного введення пароля або 2FA-коду.

Сеанс користувача може бути завершений одним способом:

– Користувачем власноруч, через вибір функції «Вихід».

У будь-якому випадку, програма завершує з'єднання з сервером, очищає тимчасові дані (наприклад, токени), зупиняє активні потоки (threads) та завершує свою роботу.

Розроблена функціональна схема дозволяє детально відстежувати логіку взаємодії користувача із системою, демонструючи ключові етапи: ініціалізація, автентифікація, обробка 2FA, блокування сеансу та завершення роботи. Такий підхід гарантує високий рівень захисту, оберігає дані користувача, забезпечує стійкість до типових атак (підбір паролів, несанкціонований доступ) та відповідає сучасним вимогам до безпечних програмних систем.

### 3.4 Розробка діаграми процесів

Діаграма містить основні етапи, починаючи з запуску застосунку і закінчуючи виконанням операцій над замком та фіксацією відомостей про події.

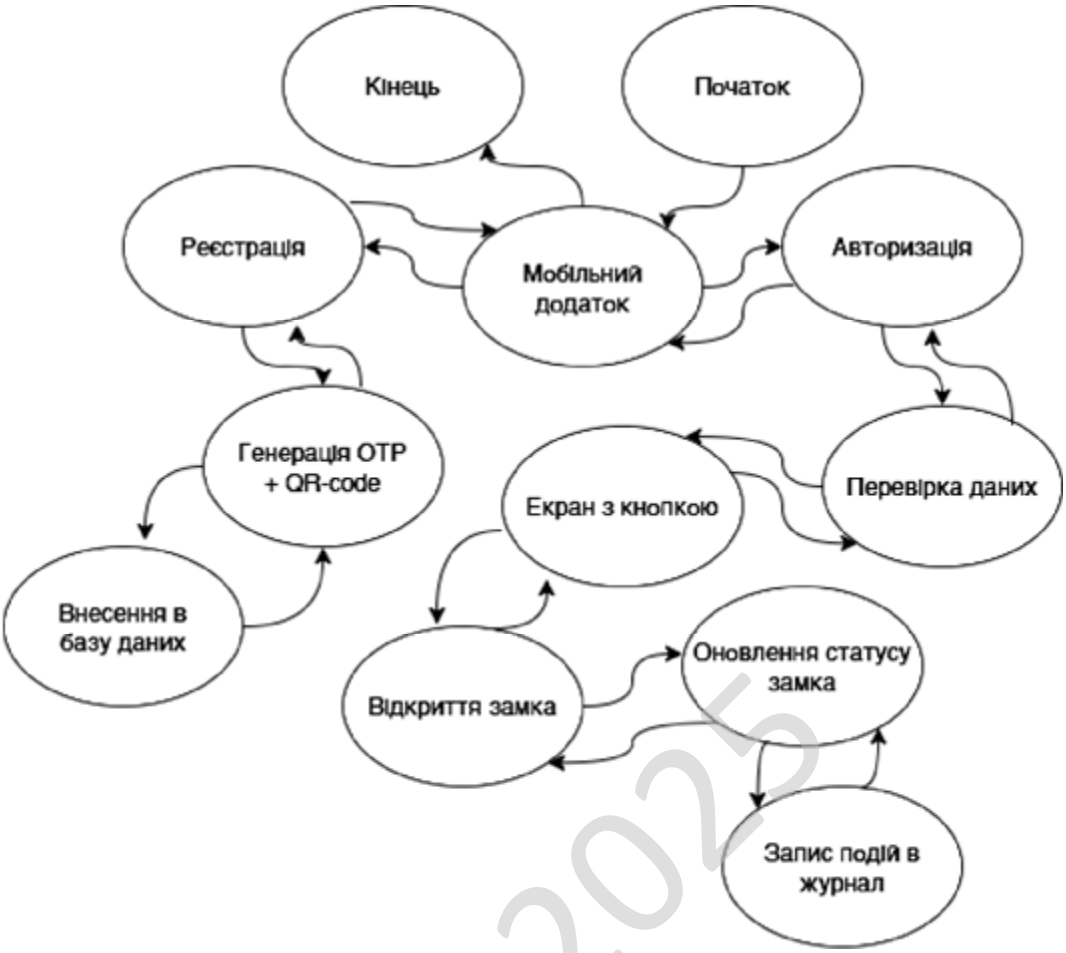



Рисунок 3.3 – Діаграма функцій

На першому етапі користувач має змогу пройти реєстрацію, що передбачає створення облікового запису, генерацію одноразового коду (OTP) та збереження відповідних відомостей у базі даних. Далі передбачено можливість авторизації з використанням логіну, паролю та OTP-коду у випадку активованої двофакторної автентифікації.

Після успішного входу користувач потрапляє на основний екран програми, де може переглядати стан замка та ініціювати його відмикання або замикання. Кожна така дія надсилається до серверної частини, де обробляється, після чого статус замка оновлюється у клієнтському інтерфейсі. Додатково, для підвищення безпеки, реалізовано функцію автоматичного блокування замка через визначений проміжок часу.

Кожна дія, включно зі спробами входу та керування замком, супроводжується логуванням: події записуються до бази даних для подальшого аналізу та аудиту.

Отже, ця діаграма відображає загальний сценарій користування мобільним додатком, з акцентом на безпечну і зручну взаємодію між користувачем, додатком та фізичним пристроєм замка.

# 4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

## 4.1 Блок-схеми та опис алгоритмів функціонування системи

Основна програма є клієнтською складовою системи управління розумним замком та втілена мовою програмування Python з використанням фреймворку Kivy. Вона забезпечує графічний інтерфейс користувача та виконує ключові функції взаємодії з сервером, враховуючи автентифікацію, авторизацію, керування пристроєм, підтримку двофакторної автентифікації (2FA), а також обробку повідомлень та помилок.

Після запуску застосунку відбувається ініціалізація графічного інтерфейсу. Користувач потрапляє на головний екран, де вводить логін та пароль. У випадку нового користувача передбачена змога переходу на екран реєстрації, де вводяться нові облікові дані. Після введення інформації та натискання кнопки “Login” або “Register” відповідно, програма формує запит до API серверної частини системи. Усі дані передаються у форматі JSON.

У разі вдалого входу, сервер повертає токен доступу (JWT), котрий на далі використовується для авторизації при всіх запитах до сервера. Якщо користувач має активовану 2FA, після введення логіну й паролю він повинен додатково ввести одноразовий код підтвердження з мобільного додатку,наприклад, Google Authenticator. Цей код перевіряється на сервері. Якщо код вірний, користувач переходить до основного екрану програми, де йому стає доступним керування розумним замком.

Інтерфейс керування включає кнопки для відмикання й замикання замка, а також можливість виходу з облікового запису. Усі дії, що стосуються замка, супроводжуються запитом до серверної частини, при цьому токен користувача

додається до заголовків запиту для підтвердження його автентичності. Система обробляє відповіді сервера й відповідно змінює інтерфейс, показуючи поточний стан замка.

З міркувань безпеки реалізовано автоматичне блокування замка через певний проміжок часу після відмикання. Це зроблено за допомогою механізму таймера: після натискання кнопки “Відкрити” запускається зворотний відлік, після завершення якого клієнтська програма самостійно надсилає запит на замикання замка. Це дозволяє уникнути ситуацій, коли користувач відкрив замок, але забув його зачинити.

Програма також має захист від brute-force атак. У разі п’яти послідовних невдалих спроб входу наступні спроби блокуються на 10 секунд. Користувач інформується про помилки або неправильне введення за допомогою спливаючих вікон повідомлень. Усі критичні дії програми супроводжуються перевірками введених даних, і в разі невідповідності або виникнення помилок у з’єднанні з сервером, користувач отримає відповідне сповіщення.

Таким чином, клієнтська частина системи розумного замка виконує функцію інтерфейсу між користувачем і сервером, забезпечуючи зручне керування пристроєм, надійну авторизацію, підвищений рівень безпеки та стабільну взаємодію з серверною логікою. Уся взаємодія з користувачем побудована з урахуванням простоти використання та ефективного реагування на дії, що дозволяє застосунку працювати інтуїтивно зрозуміло і надійно в умовах реального часу.

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

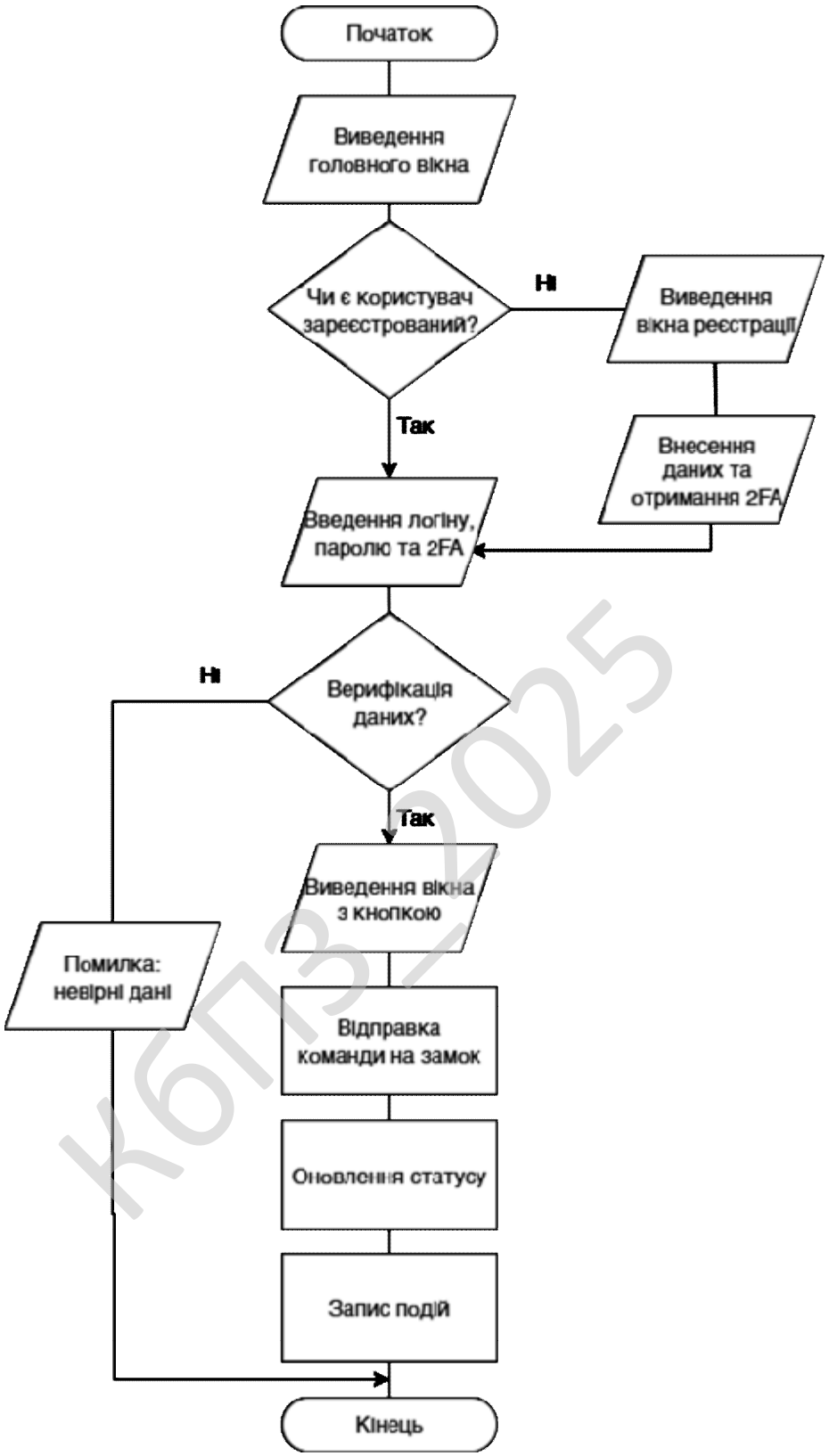



Рисунок 4.1 – Блок-схема основної програми

**login()** у LoginScreen відповідає за аутентифікацію. Якщо юзер не заблокований, функція збирає логін, пароль і, за потреби, одноразовий 2FA-код, надсилає запит POST на /login і аналізує відповідь сервера. У разі успіху зберігає JWT-токен, перемикає додаток на екран керування замком, показує pop-up «Successfully logged in!» і скидає лічильники помилкових спроб. Якщо бек-енд запитує 2FA, показує відповідне застереження. При невірних даних збільшує лічильник спроб; після п’яти помилок викликає десятисекундне блокування, під час якого кнопка входу вимкнена й відлічується таймер. З’єднання з сервером, що не вдалося, теж показується pop-up-ом.

**Лістинг функції авторизації:**

```
def login(self):
    # Перевірка блокування
    if self.is_locked:
        self.show_error_popup("Too many failed attempts. Please wait.")
        return

    # Збір даних з текстових полів
    username = self.ids.username.text
    password = self.ids.password.text
    otp_code = self.ids.otp_code.text

    # Формування тіла запиту
    payload = {
        "username": username,
        "password": password,
    }

    if otp_code:
        payload["2fa_code"] = otp_code

    try:
        # Відправка POST-запиту до серверу
        response = requests.post("http://127.0.0.1:5000/login", json=payload)

        if response.status_code == 200:
```

```
            # Успішний вхід
            token = response.json().get("token")
            self.manager.current = "lock_screen"
            self.manager.get_screen("lock_screen").set_token(token)
            self.show_success_popup("Successfully logged in!")
            self.login_attempts = 0
            self.ids.lock_label.text = ""
        elif response.status_code == 401 and response.json().get("message") ==
"2FA code required":
            self.show_error_popup("2FA code required. Please enter the code from
your authenticator app.")
        else:
            # Помилка входу
            self.login_attempts += 1
            attempts_left = 5 - self.login_attempts
            if self.login_attempts >= 5:
                self.start_lockout()
            else:
                self.show_error_popup(f"Invalid credentials or 2FA code. Attempts
left: {attempts_left}")
    except requests.exceptions.RequestException:
        self.show_error_popup("Server connection error.")
```

Функція **register()** у RegisterScreen втілює швидку реєстрацію з миттєвим підключенням двофакторної аутентифікації. Насамперед відбувається POST на /register. Якщо користувача зараховано, функція автоматично виконує логін, дістає JWT і запитує «/2fa/setup», передаючи токен у заголовку. Сервер повертає PNG-зображення QR-коду; воно показується у вікні, аби новоспечений користувач одразу додав запис у Google Authenticator. Будь-які негаразди — зайнятий логін, помилка під час логіну чи недоступність QR-коду — виводяться через pop-up помилки.

**Лістинг функції реєстрації:**

```
def register(self):
    username = self.ids.reg_username.text
    password = self.ids.reg_password.text
```

```
    try:
        # Крок 1: надсилання реєстраційних даних
        response = requests.post("http://127.0.0.1:5000/register", json={
            "username": username,
            "password": password
        })

        if response.status_code == 201:
            # Крок 2: логін для отримання токена
            login_resp = requests.post("http://127.0.0.1:5000/login", json={
                "username": username,
                "password": password
            })

            if login_resp.status_code == 200:
                token = login_resp.json().get("token")

                # Крок 3: запит QR-коду для 2FA
                headers = {"Authorization": f"Bearer {token}"}
                qr_resp = requests.get("http://127.0.0.1:5000/2fa/setup",
headers=headers)

                if qr_resp.status_code == 200:
                    # Крок 4: відображення QR-коду
                    image_data = qr_resp.content
                    data = BytesIO(image_data)
                    img = CoreImage(data, ext='png')

                    popup = Popup(title="Scan this QR with your authenticator",
                                  content=Image(texture=img.texture),
                                  size_hint=(None, None), size=(300, 300))
                    popup.open()
                else:
                    self.show_error_popup("Failed to get 2FA QR code.")
            else:
                self.show_error_popup("Login failed after registration.")
        else:
            self.show_error_popup("Username already exists.")
    except requests.exceptions.RequestException:
        self.show_error_popup("Server error during registration.")
```

**lock_action()** у LockScreen надсилає «lock» або «unlock» на бек-енд. За відсутністю токена користувачеві пропонується увійти повторно. В іншому випадку формується запит POST на /lock з JWT-авторизацією. Коли сервер відповідає вдало, текст з відповіді відображається і на екрані, і в pop-up «Success». Якщо двері відмикаються, запускається таймер автозамикання на 10 секунд; при замиканні таймер відміняється. Помилки (статус ≠ 200) чи проблеми з мережею теж повідомляються користувачеві через pop-up.

**Лістинг функції замку:**

```
def lock_action(self, action):
    if not self.token:
        self.show_error_popup("No valid token. Please log in again.")
        return

    headers = {"Authorization": f"Bearer {self.token}"}
    payload = {"action": action}

    try:
        response = requests.post("http://127.0.0.1:5000/lock", json=payload,
headers=headers)
        if response.status_code == 200:
            msg = response.json().get("message")
            self.update_status_label(msg)
            self.show_success_popup(msg)

            # Якщо двері відкриваються — активувати авто-блок
            if action == "unlock":
                self.schedule_auto_lock()
            elif action == "lock":
                self.cancel_auto_lock()
        else:
            self.show_error_popup(response.json().get("message"))
    except requests.exceptions.RequestException:
        self.show_error_popup("Server connection error.")
```

На рисунку 4.2 зображено блок-схему алгоритму роботи підпрограми пошуку.

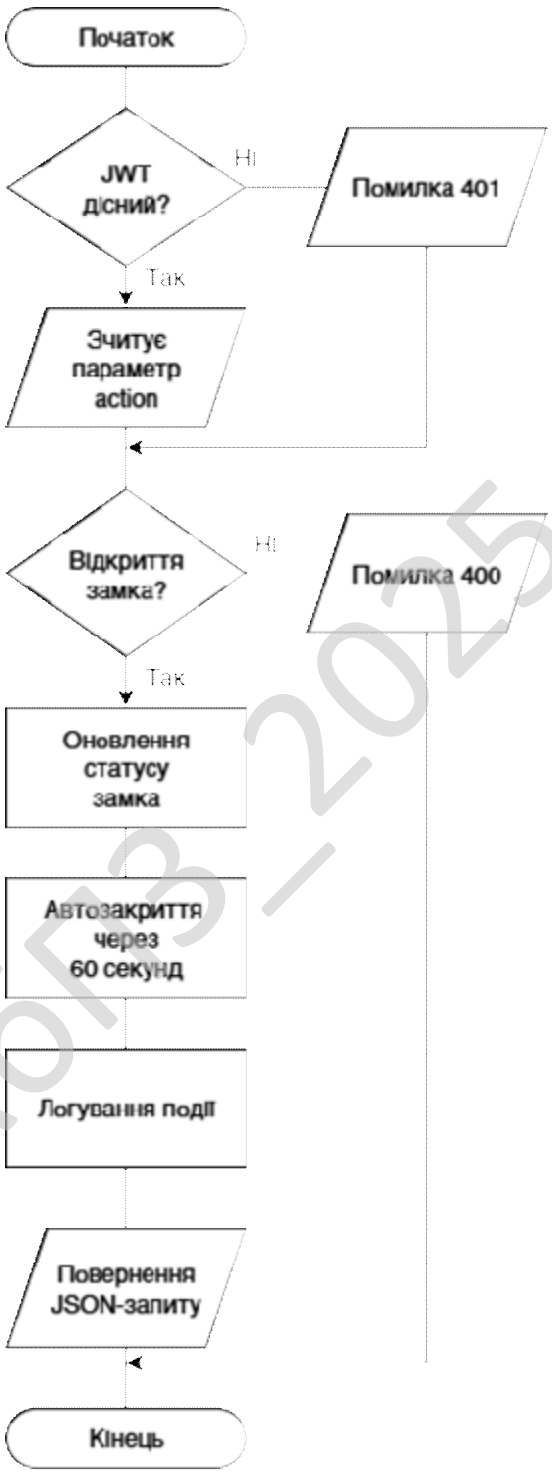





Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми пошуку

Підпрограма «Управління станом замка» обслуговує запит POST /lock, що надходить з мобільного застосунку. Перш ніж виконати будь-які дії, декоратор `@jwt_required()` перевіряє валідність токена: розбирає заголовок Authorization, засвідчує підпис JWT, контролює термін дії та витягує ім'я користувача. Якщо автентифікація не проходить, сервер одразу повертає 401, і логіка підпрограми навіть не запускається.

**Лістинг функції перевірки JWT:**

```
# ---- головна підпрограма керування замком ------------------------
@app.route("/lock", methods=["POST"])
@jwt_required()
def lock_control():
global lock_status
current_identity = get_jwt_identity()
data = request.json

if "action" in data and data["action"] in ["lock", "unlock"]:
# оновлення стану
lock_status["locked"] = (data["action"] == "lock")
log_action(current_identity, f"Lock {data['action']}")

# запуск автозамикання після unlock
if data["action"] == "unlock":
threading.Thread(target=auto_lock, daemon=True).start()

return jsonify({"message": f"Lock is now "
f"{'locked' if lock_status['locked'] else 'unlocked'}"})

# якщо action невірний
log_action(current_identity, f"Failed action: {data.get('action',
'unknown')}")
return jsonify({"message": "Invalid action"}), 400
```

Після успішної перевірки JSON-тіло запиту аналізується на наявність ключа action. Єдині допустимі значення — «lock» або «unlock»; будь-яке інше трактується як помилка 400.

**Лістинг функції валідації:**

```
        data = request.json
        якщо "action" не в даних або data["action"] не в ["lock", "unlock"]:
        log_action(current_identity,                f"Невдала                дія:
{data.get('action','unknown')}")
        return jsonify({"message": "Недійсна дія"}), 400
```

Як тільки команда визнана коректною, у журналі аудиту з’являється запис про намір користувача змінити стан замка. Далі глобальний об’єкт lock_status оновлюється: якщо надійшла команда «lock», замок у словнику позначається закритим, якщо «unlock» — відчиненим. У реальній установці на цьому етапі формується й надсилається апаратна команда на модуль Bluetooth або Wi-Fi, що безпосередньо керує електромеханічним приводом.

### 4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення є одним із важливих аспектів при створенні системи контролю доступу з застосуванням двофакторної автентифікації. Для забезпечення надійного захисту від неправомірного доступу та забезпечення безперебійної роботи системи були впроваджені сучасні механізми автентифікації, авторизації та контролю доступу, а також проведене комплексне автоматизоване тестування функціональності.

Основою безпеки системи є багаторівнева модель автентифікації користувачів. Першим етапом є реєстрація користувача з унікальним ідентифікатором — логіном, а також надійним паролем. Для підвищення безпеки зберігання паролів здійснюється за допомогою хешування з використанням криптографічно стійких алгоритмів. Отже, навіть у випадку несанкціонованого доступу до бази даних паролі залишаються захищеними.

Після реєстрації для входу в систему юзер проходить процедуру логіну, під час якої здійснюється перевірка відповідності введених облікових даних

збереженим у базі. У випадку успішної перевірки генерується спеціальний токен у форматі JWT, що використовується для подальшої автентифікації користувача в рамках поточної сесії. Токен має обмежений час дії, що додатково підвищує безпеку, оскільки виключає можливість тривалого використання викраденого токена.

Наступним рівнем безпеки є реалізація двофакторної автентифікації (2FA). Вона передбачає, що після введення логіна і пароля користувача має підтвердити свою особу за допомогою одноразового коду, згенерованого мобільним додатком автентифікатора. Для цього кожному користувачу при першому вході генерується унікальний секретний ключ, який можна встановити в додатку у вигляді QR-коду. Такий підхід значно ускладнює зловмисникам доступ до системи, навіть якщо їм відомий пароль.

Для контролю доступу до певних ресурсів та операцій в системі передбачено механізми авторизації на основі ролей і прав користувачів. Зокрема, керування замком, що є головним апаратним елементом системи, доступне лише користувачам з відповідними повноваженнями. Такий підхід дозволяє гнучко керувати рівнями доступу та запобігати виконанню критичних операцій неавторизованими користувачами.

Для перевірки коректності реалізації описаних механізмів безпеки та функціональності системи було розроблено набір автоматизованих тестів. Вони реалізовані на базі фреймворку pytest і охоплюють всі основні сценарії роботи:

– Реєстрація;
– Логін;
– Двофакторна автентифікація;
– Управління станом замка;
– Обробка помилкових ситуацій;

Виконані тести засвідчили вірність роботи усіх функцій. Тести реєстрації перевіряли здатність створення нового користувача та коректність збереження його облікових даних у базі. Тести логіну доводили, що користувач має змогу

отримати дійсний токен доступу при введенні правильних даних. Відповідні тести двофакторної автентифікації демонстрували вірну генерацію та верифікацію одноразового коду, а також обмежували доступ при відсутності або неправильності 2FA-коду.

Додатково були розроблені тести, котрі імітували намагання несанкціонованого доступу: входу з неправильним паролем, використання застарілих токенів, пропуск етапу 2FA. Усі ці випадки правильно оброблялися системою, що свідчить про високий рівень захищеності.

Застосування таких комплексних способів захисту і автоматизованого контролю дозволило гарантувати надійність розробленої системи і відповідає сучасним вимогам безпеки інформаційних систем. Використання JWT-токенів і двофакторної автентифікації значно зменшує ризики зламу облікових записів, а ролевий розподіл доступу мінімізує можливості несанкціонованих операцій.

Отже, розроблене програмне забезпечення має високий рівень захисту, що дозволяє використовувати систему в умовах, які вимагають надійної безпеки, наприклад у спорудах із обмеженим доступом або на підприємствах із підвищеними вимогами до контролю.

Додатково, впровадження автоматизованих тестів полегшує підтримку і подальший розвиток системи, оскільки дозволяє оперативно перевіряти коректність роботи при внесенні змін чи доповнень. Це значно збільшує стабільність і якість продукту в цілому.

Перспективами подальшого розвитку захисту є впровадження механізмів моніторингу та логування спроб входу й операцій, інтеграція з корпоративними системами автентифікації (Google Authenicator), а також розширення системи ролей для більш точного контролю доступу.

# 5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Впровадження системи «розумний замок» у виробничу експлуатацію передбачає комплексний та поетапний підхід. У процесі інтеграції подібного рішення необхідно брати до уваги цілий спектр технічних, організаційних та інформаційно-безпекових аспектів, котрі забезпечать надійну та стабільну роботу як на рівні окремого користувача, так і в масштабі великої інфраструктури.

Системні потреби:

– Операційна система: Windows 10/Windows 11, Linux;

– Python: версія 3.9 чи свіжіша;

– Пакунки: Flask, Flask-JWT-Extended, Flask-Bcrypt, Flask-SQLAlchemy, Flask-Migrate, pyotp, qrcode, Pillow;

– База даних: SQLite (у вбудованому вигляді, дозволяється замінити на PostgreSQL у виробничому середовищі);

– Веб-сервер: Gunicorn / uWSGI у зв'язці з Nginx (для продакшн середовища);

Одним із перших етапів впровадження є аналіз і підготовка технічної інфраструктури. Сюди входить створення надійного мережевого середовища з захищеним каналом зв’язку між сервером і клієнтськими пристроями, налаштування серверного середовища для обробки запитів, встановлення необхідного програмного забезпечення та формування бази даних. Для тестових цілей була використана легка база даних SQLite, котра дозволила оперативно протестувати систему у середовищі розробки.

Ініціалізація бази даних здійснюється за допомогою Flask-Migrate:

```
flask db init
flask db migrate -m "Initial migration."
flask db upgrade
```

Для доступу до системи користувачі застосовують веб-інтерфейс із формою входу, яка містить поля для введення логіна та пароля, а також підтримує двофакторну автентифікацію (2FA) для збільшення безпеки. Користувачі, які ще не зареєстровані, можуть скористатися кнопкою реєстрації, щоб створити новий акаунт. Після введення облікових даних та проходження 2FA, користувач отримує доступ до функцій керування замком і перегляду журналу подій.

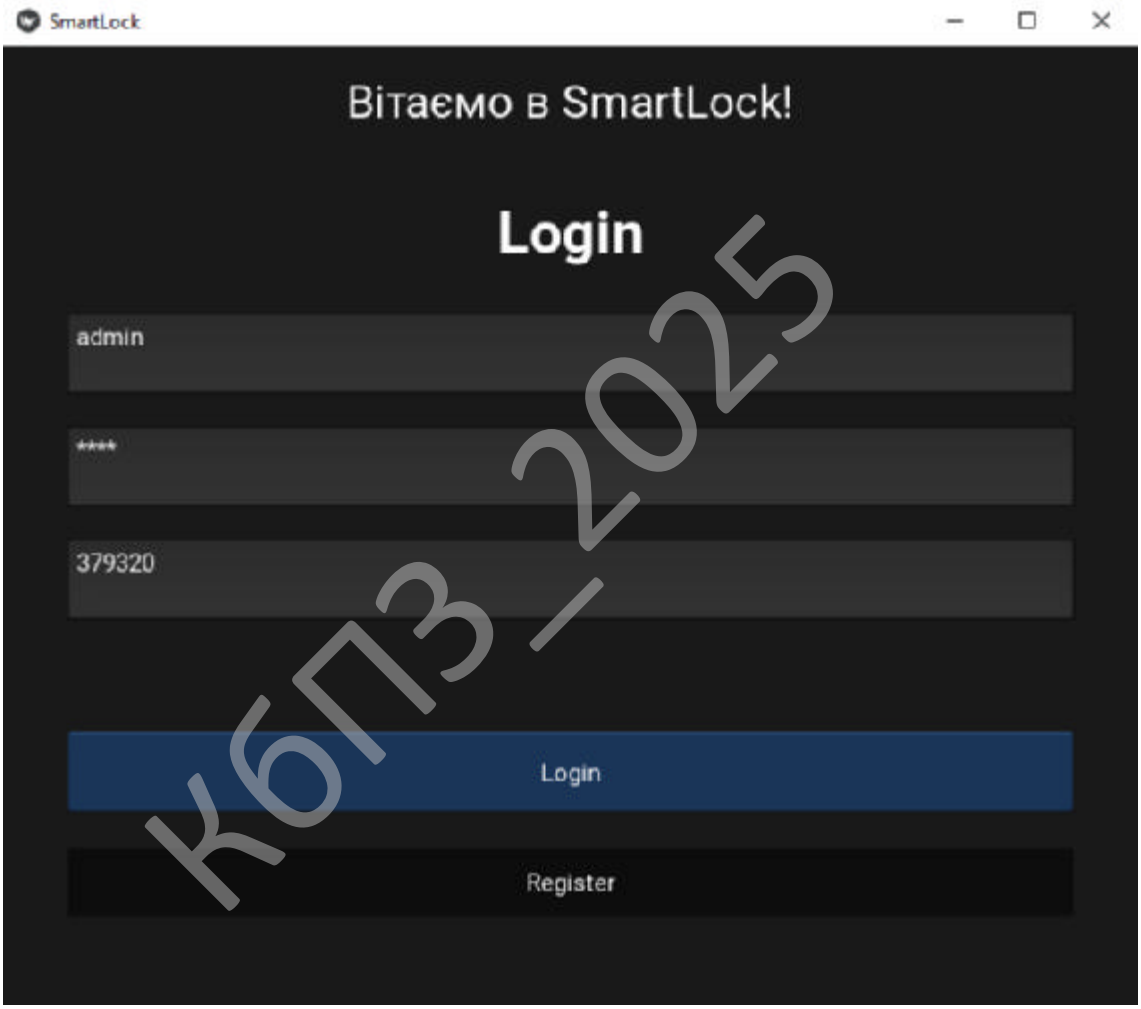





Рисунок 5.1 – Головна сторінка

З технічного боку, серверна частина виконана з використанням Python-фреймворку Flask, що дозволяє створювати REST API швидко й ефективно. Інтерфейс прикладного програмування (API) забезпечує взаємодію клієнтів із

сервером через надсилання HTTP-запитів та отримання відповідей у форматі JSON. Безпека взаємодії гарантується за допомогою токенної аутентифікації, реалізованої через JSON Web Tokens (JWT), а також хешуванням паролів користувачів із використанням алгоритму bcrypt. Ці засоби захищають систему від несанкціонованого доступу, навіть у випадку компрометації каналу зв'язку.

Усі базові параметри зосереджені у Flask-конфігурації:

`JWT_SECRET_KEY`: таємний ключ для генерування токенів доступу

`SQLALCHEMY_DATABASE_URI`: шлях до сховища даних

`JWT_ACCESS_TOKEN_EXPIRES`: тривалість життя токена (1 година)

`BCRYPT_LOG_ROUNDS`: складність хешування паролів (12 раундів)

Особливу увагу було приділено впровадженню двофакторної автентифікації (2FA), яка значно підвищує безпеку облікових записів користувачів. Після реєстрації користувачеві пропонується відсканувати QR-код, який містить його унікальний секретний ключ. Цей ключ зберігається в базі даних у зашифрованому вигляді та застосовується для синхронізації з мобільним додатком Google Authenticator, котрий генерує одноразові коди доступу кожні 30 секунд. У результаті, навіть якщо зловмисник дізнається пароль користувача, без TOTP-коду йому не вдасться авторизуватися у системі.


Рисунок 5.2 - Інтерфейс реєстрації користувача та QR-код

Наступним важливим елементом системи є керування станом замка. Для цього в API було створено особливий ендпойнт `/lock`, що приймає запити на зміну стану пристрою. Ці запити дозволяють або замкнути, або відімкнути замок залежно від поточного стану. Щоб уникнути ситуацій, коли користувач забув повторно замкнути замок, було впроваджено функцію автоматичного замикання через 60 секунд після відмикання. Такий підхід дозволяє не тільки підвищити рівень безпеки, а й мінімізувати людський фактор.

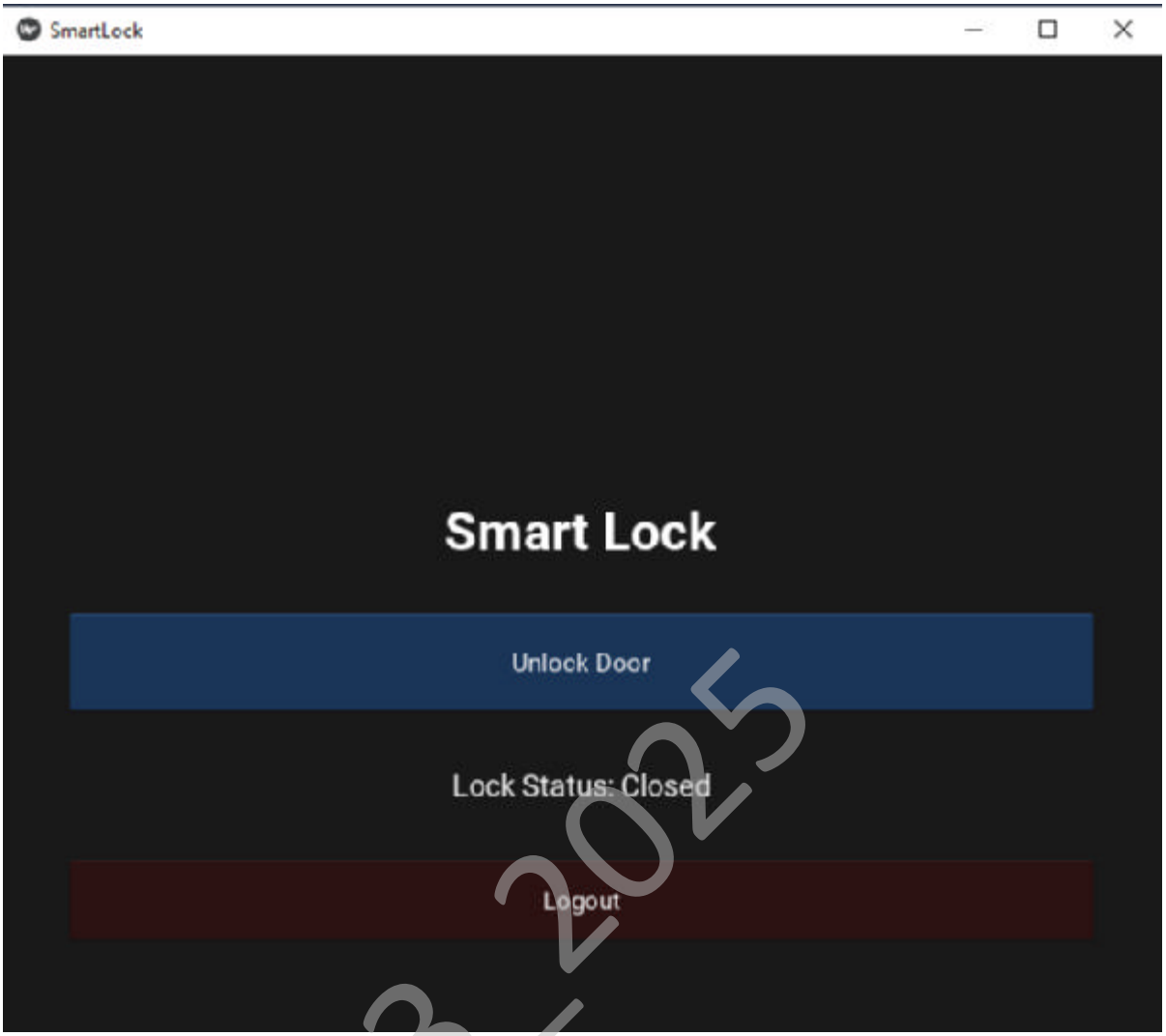
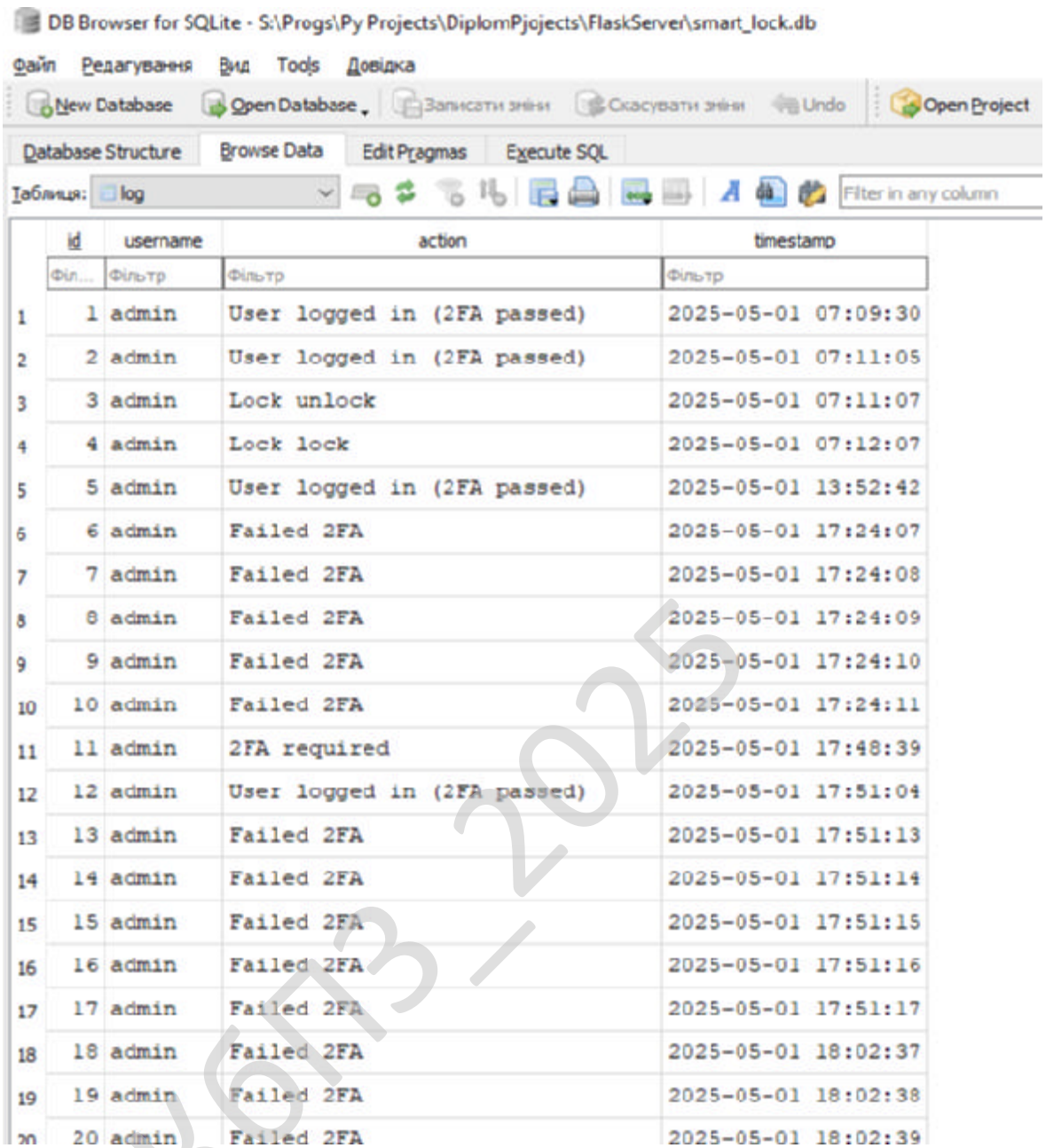



Рисунок 5.3 – Кнопка відкриття замка



Важливим аспектом системи є зберігання журналу подій (логування). Усі операції користувачів, як спроби входу, зміна стану замка, помилки автентифікації або неправильні коди, записуються в логах. Ця інформація зберігається на сервері й доступна тільки адміністраторам, що дозволяє проводити аудит безпеки та аналізувати потенційні загрози або підозрілу поведінку.




| | id | username | action | timestamp |
|---|---|---|---|---|
| | Філ... | Фільтр | Фільтр | Фільтр |
| 1 | 1 | admin | User logged in (2FA passed) | 2025-05-01 07:09:30 |
| 2 | 2 | admin | User logged in (2FA passed) | 2025-05-01 07:11:05 |
| 3 | 3 | admin | Lock unlock | 2025-05-01 07:11:07 |
| 4 | 4 | admin | Lock lock | 2025-05-01 07:12:07 |
| 5 | 5 | admin | User logged in (2FA passed) | 2025-05-01 13:52:42 |
| 6 | 6 | admin | Failed 2FA | 2025-05-01 17:24:07 |
| 7 | 7 | admin | Failed 2FA | 2025-05-01 17:24:08 |
| 8 | 8 | admin | Failed 2FA | 2025-05-01 17:24:09 |
| 9 | 9 | admin | Failed 2FA | 2025-05-01 17:24:10 |
| 10 | 10 | admin | Failed 2FA | 2025-05-01 17:24:11 |
| 11 | 11 | admin | 2FA required | 2025-05-01 17:48:39 |
| 12 | 12 | admin | User logged in (2FA passed) | 2025-05-01 17:51:04 |
| 13 | 13 | admin | Failed 2FA | 2025-05-01 17:51:13 |
| 14 | 14 | admin | Failed 2FA | 2025-05-01 17:51:14 |
| 15 | 15 | admin | Failed 2FA | 2025-05-01 17:51:15 |
| 16 | 16 | admin | Failed 2FA | 2025-05-01 17:51:16 |
| 17 | 17 | admin | Failed 2FA | 2025-05-01 17:51:17 |
| 18 | 18 | admin | Failed 2FA | 2025-05-01 18:02:37 |
| 19 | 19 | admin | Failed 2FA | 2025-05-01 18:02:38 |
| 20 | 20 | admin | Failed 2FA | 2025-05-01 18:02:39 |



Рисунок 5.4 – Журнал подій

Щоб уникнути атак типу brute-force, було введено метод тимчасового блокування користувача після кількох невдалих спроб входу. Це рішення ґрунтується на зберіганні кількості спроб входу та часу останньої спроби, що дає змогу динамічно блокувати аккаунт на певний час, якщо порушено визначений ліміт.

# 6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, розроблене в межах випускної кваліфікаційної роботи першого (бакалаврського) рівня за темою «Система “розумний замок” з автентифікацією відомостей через мобільний додаток», призначене для збільшення фізичної та інформаційної безпеки об’єктів за рахунок поєднання електромеханічного замка та захищеного клієнт-серверного рішення.

У сьогоденних умовах на українському ринку представлено обмежену кількість вітчизняних комплексних систем «розумних» замків із повноцінною двофакторною автентифікацією та централізованим журналюванням подій, через те запропоноване рішення має практичну новизну й актуальність.

В процесі праці визначено такі задачі:

– Здійснено огляд наявних IoT-рішень для управління електронними замками та проаналізовано їх моделі безпеки.

– Спроєктовано клієнт-серверну архітектуру, що включає мобільний додаток, REST-API на Flask та електронний замок з модулем Bluetooth/Wi-Fi.

– Зреалізовано серверну частину з підтримкою реєстрації, JWT-автентифікації, двофакторної TOTP-перевірки, ролей користувачів та ведення журналу дій.

– Розроблено алгоритм автоматичного замикання після тайм-ауту й механізм тимчасового блокування облікового запису при багаторазових невдалих спробах входу.

– Створено зручний веб-інтерфейс для входу (логін, пароль, 2FA, реєстрація) та головний екран управління замком з відображенням останніх подій.

Основні переваги розробленого ПЗ:

– Безпека: застосування Bcrypt для хешування паролів, JWT-токенів для сесій та двофакторної автентифікації на основі TOTP.

– Надійність: механізм автоматичного запуску сервера як systemd-служби та резервне копіювання бази даних гарантують безперебійну роботу.

– Масштабованість: об’єктно-орієнтований підхід у сукупності з Flask-Migrate дозволяє легко розширювати функціонал та переходити з SQLite на PostgreSQL у продакшні.

– Зручність: інтуїтивний інтерфейс входу та управління замком не потребує від користувача особливих знань.

Програмна частина написана мовою високого рівня Python 3.9+ з використанням популярних open-source бібліотек, що зменшує строки розробки та скорочує витрати на підтримку. ПЗ працює під керуванням багатозадачних ОС Windows 10/11 чи Linux, що спрощує його впровадження у різних організаціях.

Загалом створена система підтверджує коректність ухвалених технічних рішень, відповідає вимогам технічного завдання та має потенціал для подальшого розвитку (підтримка NFC-ключів, інтеграція з “розумним” будинком, розширена аналітика журналів тощо).

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. ДСТУ ISO/IEC 27001:2015. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги.

2. ДСТУ ISO/IEC 27002:2019. Код практик з управління інформаційною безпекою.

3. ДСТУ ISO/IEC 29192-2:2017. Легковагова криптографія. Блокові шифри.

4. ДСТУ 1337-2003. Засоби криптографічного захисту інформації. Вимоги до формування ключів.

5. ДСТУ 4145-2002. Інформаційні технології. Криптографічний захист. Алгоритм цифрового підпису на еліптичних кривих.

6. ДСТУ 28147:2009. Системи оброблення інформації. Криптографічний захист. Блочний шифр.

7. ДСТУ ISO/IEC 24760-1:2019. Управління ідентичностями та ідентифікаційна безпека.

8. Закон України «Про інформаційну безпеку та кіберзахист», № 1498-IX, 15.07.2021.

9. Національна стратегія кібербезпеки України на 2021-2025 рр. – Указ Президента №96/2021.

10. Методичні рекомендації Держспецзв'язку щодо застосування двофакторної автентифікації, 2023.

11. Ковальчук О.М. «Безпека IoT-пристроїв: український контекст». – Журнал «Захист інформації», 2023, № 2, с. 17-25.

12. Шевченко В.О. «Захист Bluetooth-каналів у системах "розумний дім"». – Вісник НТУУ «КПІ». Серія «ІПС», 2022, № 71, с. 45-52.

13. Гуменюк С.І. «Використання JWT у веб-сервісах на Python». – Системи обробки інформації, 2021, вип. 167, с. 88-94.

14. Мельник І.В. «Розробка безпечних REST-API на Flask». – Матеріали конференції «IT-Sec Ukraine 2022», Київ, 2022, с. 101-106.

15. Бондаренко Д.О. «Порівняння SQLite та PostgreSQL у вбудованих системах». – Електроніка та системи керування, 2020, № 4, с. 59-64.

16. Яковенко Є.П. «Застосування TOTP-аутентифікації в мобільних додатках». – Інформаційні технології та безпека, 2023, № 3, с. 72-79.

17. Приходько Р.С. «Автоматичне блокування облікових записів при brute-force-атаках». – Вісник ХНУРЕ, 2021, № 1, с. 110-116.

18. Гайдук А.В. «Система журналювання подій у хмарних сервісах». – Вчені записки ТНУ ім. В. І. Вернадського. Технічні науки, 2022, т. 33, № 2, с. 138-145.

19. Кравченко Д.С. «Моніторинг Flask-застосунків за допомогою Prometheus». – Тези конф. «DevOps Ukraine», 2022.

20. Олійник Н.П. «Методи шифрування команд для електронних замків». – Журнал «Кібербезпека в Україні», 2021, № 5, с. 30-36.

21. Литвиненко А.М. «Огляд протоколів безпечного парування BLE-пристроїв». – Радіоелектроніка та інформатика, 2023, № 2, с. 53-60.

22. Тарасюк І.В. «Практика використання systemd для Python-сервісів». – Linux-Ukr Digest, 2020, № 12, с. 11-16.

23. Ємельянов К.О. «Розробка мобільних застосунків для керування smart-lock». – Труди ХНУРЕ, 2021, № 4, с. 77-84.

24. Дьяков С.В. «Тестування навантаження IoT-сервісів методом Locust». – Проблеми програмування, 2022, № 4, с. 41-48.

25. Кулик М.Б. «Використання AES-128 в мікроконтролерах STM32». – Сучасні інформаційні системи, 2020, № 2, с. 65-71.

26. Федоренко Г.В. «Архітектура MQTT для систем доступу». – Матеріали семінару «IoT-UA 2021», Львів, 2021.

27. Білан О.Ю. «Оцінка ризиків у системах фізичного доступу». – Захист інформації, 2022, № 1, с. 55-62.

28. Панченко П.В. «Використання QR-коду для Provisioning TOTP». – Міжнар. журн. «Комп’ютерні науки та інформаційні технології», 2023, т. 11, № 3, с. 23-29.

29. Поліщук О.В. «Скрипти резервного копіювання SQLite у Linux». – Open Source Digest, 2021, № 9, с. 18-22.

30. Виговський Б.М. «HTTPS-конфігурація Nginx для малих підприємств». – Dev-UA Magazine, 2020, № 6, с. 44-49.

31. ISO/IEC 30107-3:2017. Biometric Presentation Attack Detection.

32. RFC 9395. JSON Web Algorithms (JWA), IETF, 2023.

33. ENISA. «Good Practices for Secure IoT Software Development», 2022.

34. Gartner. «Market Guide for Smart-Lock Solutions», 2023.

35. Schneier B. “Secrets and Lies: Digital Security in a Networked World.” – Wiley, 2022.

36. Kim S. «Usability of 2FA in Smart-Lock Apps». – IEEE Access, 2023, vol. 11, p. 44012-44020.

37. Rescorla E. «Transport Layer Security 1.3 Explained». – ACM Queue, 2019.

38. Song H. «Security in Smart Home IoT: Challenges». – IEEE Comms. Surveys, 2024.

39. Conteh A. «Adaptive Account Lockout Policies». – Information Security J., 2022.

40. Burns A. «SQLite vs PostgreSQL on Edge Devices». – Embedded DB Conf., 2021.

41. Reyna A. «BLE vs Wi-Fi Performance for Smart Locks». – IEEE Access, 2021.

42. Anderson R. “Security Engineering.” 3rd ed., Wiley, 2020.

43. Kettunen A. «Prometheus & Grafana for Python Microservices». – DevOps Digest, 2021.

44. Fielding R. “Architectural Styles of Network-Based Software.” – UC Irvine, 2000.

45. Tange A. «Flask-Migrate in Large Projects». – JOSS, 2019.

46. White G. “Practical Lock Security.” – Syngress, 2021.

47. Ali M. «Best Practices for HTTPS on Nginx». – SysAdmin Magazine, 2022.

48. Young J. “Design Patterns for Python Microservices.” – Manning, 2021.

49. Holz T. «Measuring Security Posture of IoT Devices». – ACM CCS, 2022.

50. Клименко С.О. «Керівні принципи безпечної розробки прошивок smart-lock». – Український журнал інформаційної безпеки, 2024, № 2, с. 9-15.

Додаток А
(обов'язковий)

**Технічне завдання**

**Зміст**

## 1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи "розумний замок" з автентифікацією даних через мобільний додаток.

## 2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 47-02 від 17.01.2025 року).

## 3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи "розумний замок" з автентифікацією даних через мобільний додаток.

## 4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

## 5 Технічні вимоги

### 5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи «розумного замка» з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

### 5.2 Показники призначення

Система повинна забезпечувати:

– системи "розумний замок" з автентифікацією даних через мобільний додаток

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

### 5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

### 5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

### 5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

### 5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

### 5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

### 5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

#### 5.8.1 Обладнання

Комп’ютер Intel® Core i3/ 8 Gb/ 128 Gb/ Intel HD Graphics або сумісні з ним.

#### 5.8.2 Мова програмування

Для реалізації програмної частини проєкту обрано Python у середовищі PyCharm.

### 5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

### 5.8.4 Вихідні дані

Робоча програма.

## 6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

## 7 Перелік документів, що розробляються

- Структурна схема системи
- Функціональна схема системи
- Діаграма процесів
- Блок-схема алгоритму роботи програми
- Пояснювальна записка

## 8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

## 9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист __.__.2025 р.

Додаток Б

(обов’язковий)

**Міністерство освіти і науки України**

**Центральноукраїнський національний технічний університет**

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Минайленко Р. М.

***Програмне забезпечення системи "розумний замок" з автентифікацією даних через мобільний додаток***

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 16

Літера: РП

Кропивницький – 2025 року

**Основна програма**

**Файл kivy_app.py основної програми**

```
# Імпортуємо необхідні модулі
import requests  # для HTTP-запитів до бекенду
from kivy.app import App  # базовий клас для додатку Kivy
from kivy.uix.screenmanager import ScreenManager, Screen  # керування екранами
from kivy.uix.label import Label  # текстові мітки
from kivy.uix.popup import Popup  # спливаючі вікна
from kivy.uix.boxlayout import BoxLayout  # вертикальні/горизонтальні контейнери
from kivy.clock import Clock  # таймери і планування
from kivy.uix.image import Image  # для зображень
from kivy.core.window import Window  # робота з вікном програми
from kivy.uix.button import Button  # кнопки
from kivy.uix.textinput import TextInput  # текстові поля
import qrcode  # створення QR-коду
from io import BytesIO  # буфер для зображень
from kivy.core.image import Image as CoreImage  # обробка зображень із байтів
import base64
import pyotp  # для роботи з 2FA (OTP)

# Екран логіну
class LoginScreen(Screen):
    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
        self.login_attempts = 0  # кількість спроб входу
        self.is_locked = False  # чи заблоковано користувача після помилок
        self.remaining_time = 0  # залишок часу до розблокування
        self.timer_event = None  # подія таймера

    def login(self):
        # Перевірка блокування
        if self.is_locked:
            self.show_error_popup("Too many failed attempts. Please wait.")
            return

        # Збір даних з текстових полів
        username = self.ids.username.text
        password = self.ids.password.text
        otp_code = self.ids.otp_code.text

        # Формування тіла запиту
        payload = {
            "username": username,
            "password": password,
        }

        if otp_code:
            payload["2fa_code"] = otp_code

        try:
            # Відправка POST-запиту до серверу
            response = requests.post("http://127.0.0.1:5000/login",
json=payload)

            if response.status_code == 200:
                # Успішний вхід
                token = response.json().get("token")
                self.manager.current = "lock_screen"
                self.manager.get_screen("lock_screen").set_token(token)
                self.show_success_popup("Successfully logged in!")
                self.login_attempts = 0
                self.ids.lock_label.text = ""
            elif response.status_code == 401 and response.json().get("message")
== "2FA code required":
                self.show_error_popup("2FA code required. Please enter the code
```

```
from your authenticator app.")
            else:
                # Помилка входу
                self.login_attempts += 1
                attempts_left = 5 - self.login_attempts
                if self.login_attempts >= 5:
                    self.start_lockout()
                else:
                    self.show_error_popup(f"Invalid credentials or 2FA code.
Attempts left: {attempts_left}")
        except requests.exceptions.RequestException:
            self.show_error_popup("Server connection error.")

    # Блокує можливість входу на 10 секунд після 5 помилок
    def start_lockout(self):
        self.is_locked = True
        self.remaining_time = 10
        self.ids.login.disabled = True
        self.ids.lock_label.text = f"Too many failed attempts. Try again in
{self.remaining_time} seconds."
        self.timer_event = Clock.schedule_interval(self.update_timer, 1)

    # Оновлює таймер блокування
    def update_timer(self, dt):
        self.remaining_time -= 1
        if self.remaining_time > 0:
            self.ids.lock_label.text = f"Too many failed attempts. Try again in
{self.remaining_time} seconds."
        else:
            self.timer_event.cancel()
            self.unlock_login()

    # Розблокування можливості входу
    def unlock_login(self):
        self.is_locked = False
        self.login_attempts = 0
        self.ids.login.disabled = False
        self.ids.lock_label.text = "You can try logging in again."
        Clock.schedule_once(self.clear_lock_label, 2)

    def clear_lock_label(self, dt):
        self.ids.lock_label.text = ""

    # Спливаючі вікна успіху та помилок
    def show_success_popup(self, message):
        popup = Popup(title='Success', content=Label(text=message),
size_hint=(None, None), size=(400, 200))
        popup.open()

    def show_error_popup(self, message):
        popup = Popup(title='Error', content=Label(text=message),
size_hint=(None, None), size=(400, 200))
        popup.open()


# Екран реєстрації нового користувача
class RegisterScreen(Screen):
    def register(self):
        username = self.ids.reg_username.text
        password = self.ids.reg_password.text

        try:
            # Крок 1: надсилання реєстраційних даних
            response = requests.post("http://127.0.0.1:5000/register", json={
                "username": username,
                "password": password
            })

            if response.status_code == 201:
```

```
                # Крок 2: логін для отримання токена
                login_resp = requests.post("http://127.0.0.1:5000/login", json={
                    "username": username,
                    "password": password
                })

                if login_resp.status_code == 200:
                    token = login_resp.json().get("token")

                    # Крок 3: запит QR-коду для 2FA
                    headers = {"Authorization": f"Bearer {token}"}
                    qr_resp = requests.get("http://127.0.0.1:5000/2fa/setup",
headers=headers)

                    if qr_resp.status_code == 200:
                        # Крок 4: відображення QR-коду
                        image_data = qr_resp.content
                        data = BytesIO(image_data)
                        img = CoreImage(data, ext='png')

                        popup = Popup(title="Scan this QR with your
authenticator",
                                      content=Image(texture=img.texture),
                                      size_hint=(None, None), size=(300, 300))
                        popup.open()
                    else:
                        self.show_error_popup("Failed to get 2FA QR code.")
                else:
                    self.show_error_popup("Login failed after registration.")
            else:
                self.show_error_popup("Username already exists.")
        except requests.exceptions.RequestException:
            self.show_error_popup("Server error during registration.")

    def show_error_popup(self, message):
        popup = Popup(title='Error', content=Label(text=message),
size_hint=(None, None), size=(400, 200))
        popup.open()

    def show_popup(self, title, message):
        popup = Popup(title=title, content=Label(text=message), size_hint=(None,
None), size=(400, 200))
        popup.open()


# Екран керування "розумним замком"
class LockScreen(Screen):
    def __init__(self, **kwargs):
        super(LockScreen, self).__init__(**kwargs)
        self.token = None
        self.auto_lock_event = None  # автоматичне блокування після відкриття

    def set_token(self, token):
        self.token = token

    def lock_action(self, action):
        if not self.token:
            self.show_error_popup("No valid token. Please log in again.")
            return

        headers = {"Authorization": f"Bearer {self.token}"}
        payload = {"action": action}

        try:
            response = requests.post("http://127.0.0.1:5000/lock", json=payload,
headers=headers)
            if response.status_code == 200:
                msg = response.json().get("message")
                self.update_status_label(msg)
```

```
                self.show_success_popup(msg)

                # Якщо двері відкриваються — активувати авто-блок
                if action == "unlock":
                    self.schedule_auto_lock()
                elif action == "lock":
                    self.cancel_auto_lock()
            else:
                self.show_error_popup(response.json().get("message"))
        except requests.exceptions.RequestException:
            self.show_error_popup("Server connection error.")

    def unlock_door(self):
        self.lock_action("unlock")

    def lock_door(self, *_):  # *_ — щоб можна було викликати з Clock
        self.lock_action("lock")

    def logout(self):
        self.token = None
        self.cancel_auto_lock()
        self.manager.current = "login_screen"

    def update_status_label(self, message):
        # Оновлює статус дверей на екрані
        if "unlock" in message.lower():
            self.ids.status.text = "Lock Status: Opened"
        elif "lock" in message.lower():
            self.ids.status.text = "Lock Status: Closed"

    def schedule_auto_lock(self):
        # Планує автоматичне блокування через 10 секунд
        if self.auto_lock_event:
            self.auto_lock_event.cancel()
        self.auto_lock_event = Clock.schedule_once(self.lock_door, 10)

    def cancel_auto_lock(self):
        if self.auto_lock_event:
            self.auto_lock_event.cancel()
            self.auto_lock_event = None

    def show_success_popup(self, message):
        popup = Popup(title='Success', content=Label(text=message),
size_hint=(None, None), size=(400, 200))
        popup.open()

    def show_error_popup(self, message):
        popup = Popup(title='Error', content=Label(text=message),
size_hint=(None, None), size=(400, 200))
        popup.open()


# Основний клас додатку
class SmartLockApp(App):
    def build(self):
        # Додає екрани до менеджера
        sm = ScreenManager()
        sm.add_widget(LoginScreen(name="login_screen"))
        sm.add_widget(LockScreen(name="lock_screen"))
        sm.add_widget(RegisterScreen(name='register_screen'))
        return sm


# Точка входу в програму
if __name__ == "__main__":
    SmartLockApp().run()
```

**Файл smartlockapp.kv – Дизайн інтерфейсу**

```
<LoginScreen>:
    BoxLayout:
        orientation: "vertical"
        padding: [40, 60, 40, 60]
        spacing: 20
        canvas.before:
            Color:
                rgba: 0.1, 0.1, 0.1, 1
            Rectangle:
                pos: self.pos
                size: self.size

        Label:
            text: "Вітаємо в SmartLock!"
            font_size: 28
            size_hint_y: None
            height: 60
            halign: "center"
            valign: "middle"
            text_size: self.size

        Label:
            text: "Login"
            font_size: 36
            color: 1, 1, 1, 1
            bold: True
            size_hint_y: None
            height: 60

        TextInput:
            id: username
            hint_text: "Username"
            background_color: 0.2, 0.2, 0.2, 1
            foreground_color: 1, 1, 1, 1
            cursor_color: 0.4, 0.6, 1, 1
            size_hint_y: None
            height: 50
            multiline: False

        TextInput:
            id: password
            hint_text: "Password"
            password: True
            background_color: 0.2, 0.2, 0.2, 1
            foreground_color: 1, 1, 1, 1
            cursor_color: 0.4, 0.6, 1, 1
            size_hint_y: None
            height: 50
            multiline: False

        TextInput:
            id: otp_code
            hint_text: "2FA Code (if enabled)"
            background_color: 0.2, 0.2, 0.2, 1
            foreground_color: 1, 1, 1, 1
            cursor_color: 0.4, 0.6, 1, 1
            size_hint_y: None
            height: 50
            multiline: False

        Label:
            id: lock_label
            text: ""
            color: (1, 0, 0, 1)  # червоний
            font_size: 16
```

```
            size_hint_y: None
            height: 30

        Button:
            id: login
            text: "Login"
            background_color: 0.3, 0.6, 1, 1
            color: 1, 1, 1, 1
            size_hint_y: None
            height: 50
            on_press: root.login()

        Button:
            text: "Register"
            background_color: 0.15, 0.15, 0.15, 1
            color: 1, 1, 1, 1
            size_hint_y: None
            height: 45
            on_press: app.root.current = "register_screen"

<RegisterScreen>:
    BoxLayout:
        orientation: 'vertical'
        padding: [40, 60, 40, 60]
        spacing: 20
        canvas.before:
            Color:
                rgba: 0.1, 0.1, 0.1, 1
            Rectangle:
                pos: self.pos
                size: self.size

        Label:
            text: "Register"
            font_size: 30
            color: 1, 1, 1, 1
            bold: True

        TextInput:
            id: reg_username
            hint_text: "Username"
            background_color: 0.2, 0.2, 0.2, 1
            foreground_color: 1, 1, 1, 1
            size_hint_y: None
            height: 50
            multiline: False

        TextInput:
            id: reg_password
            hint_text: "Password"
            password: True
            background_color: 0.2, 0.2, 0.2, 1
            foreground_color: 1, 1, 1, 1
            size_hint_y: None
            height: 50
            multiline: False

        Button:
            text: "Register"
            background_color: 0.3, 0.7, 0.3, 1
            color: 1, 1, 1, 1
            size_hint_y: None
            height: 50
            on_press: root.register()

        Button:
            text: "Back to Login"
            background_color: 0.15, 0.15, 0.15, 1
            color: 1, 1, 1, 1
```

```
            size_hint_y: None
            height: 45
            on_press: app.root.current = "login_screen"

<LockScreen>:
    BoxLayout:
        orientation: "vertical"
        padding: [40, 60, 40, 60]
        spacing: 25
        canvas.before:
            Color:
                rgba: 0.1, 0.1, 0.1, 1
            Rectangle:
                pos: self.pos
                size: self.size

        Label:
            text: "Smart Lock"
            font_size: 32
            color: 1, 1, 1, 1
            bold: True
            size_hint_y: None
            height: 50

        Button:
            text: "Unlock Door"
            background_color: 0.3, 0.6, 1, 1
            color: 1, 1, 1, 1
            size_hint_y: None
            height: 60
            on_press: root.unlock_door()

        Label:
            id: status
            text: "Lock Status: Closed"
            font_size: 18
            color: 1, 1, 1, 1
            size_hint_y: None
            height: 40

        Button:
            text: "Logout"
            background_color: 0.5, 0.2, 0.2, 1
            color: 1, 1, 1, 1
            size_hint_y: None
            height: 50
            on_press: root.logout()
```

**Файл main.py – Flask-сервер**

```
import pyotp
import qrcode
import io
import os
import threading
import time
from datetime import datetime, timedelta
from flask import Flask, request, jsonify, send_file
from flask_sqlalchemy import SQLAlchemy
from flask_jwt_extended import JWTManager, create_access_token, jwt_required,
get_jwt_identity
from flask_bcrypt import Bcrypt
from flask_migrate import Migrate
from PIL import Image

app = Flask(__name__)

# Налаштування бази даних SQLite, шлях до файлу бази
basedir = os.path.abspath(os.path.dirname(__file__))
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///" + os.path.join(basedir,
"smart_lock.db")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False  # Вимикаємо відстеження
змін для зменшення навантаження

# Налаштування безпеки: секретний ключ для JWT і час життя токена
app.config["JWT_SECRET_KEY"] = "super-secret-key"
app.config["JWT_ACCESS_TOKEN_EXPIRES"] = timedelta(hours=1)  # Токен дійсний 1
годину

# Кількість раундів bcrypt для хешування пароля (більше — більш безпечно, але
повільніше)
app.config["BCRYPT_LOG_ROUNDS"] = 12

# Ініціалізація розширень Flask
db = SQLAlchemy(app)          # ORM для роботи з базою даних
migrate = Migrate(app, db)    # Підтримка міграцій бази
bcrypt = Bcrypt(app)          # Хешування паролів
jwt = JWTManager(app)         # JWT авторизація

# Модель User — структура таблиці користувачів у БД
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)            # Унікальний
ідентифікатор
    username = db.Column(db.String(50), unique=True, nullable=False)  # Логін
користувача
    password = db.Column(db.String(100), nullable=False)     # Захешований
пароль
    role = db.Column(db.String(20), default="user")          # Роль користувача
(user/admin)
    two_factor_secret = db.Column(db.String(16))             # Секрет для 2FA
(TOTP)

    def __repr__(self):
        return f"User('{self.username}', '{self.role}')"

# Модель Log — зберігає логи дій користувачів (хто і що зробив, коли)
class Log(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50), nullable=False)      # Користувач, що
зробив дію
    action = db.Column(db.String(100), nullable=False)       # Опис дії
    timestamp = db.Column(db.DateTime, default=db.func.current_timestamp())  #
Час дії
```

```
# Функція для додавання запису у лог
def log_action(username, action):
    log_entry = Log(username=username, action=action)
    db.session.add(log_entry)
    db.session.commit()

# Словник для відстеження кількості невдалих спроб входу та блокування
користувача
login_attempts = {}  # Структура: {username: {"count": кількість спроб,
"blocked_until": час блокування}}

# Параметри блокування
BLOCK_THRESHOLD = 5                # Порогова кількість помилкових спроб
BLOCK_DURATION = timedelta(seconds=30)  # Час блокування після перевищення
порогу (30 секунд)

# Роут для налаштування двофакторної аутентифікації (2FA)
@app.route("/2fa/setup", methods=["GET"])
@jwt_required()  # Доступ лише для авторизованих користувачів
def setup_2fa():
    current_user = get_jwt_identity()  # Отримуємо ім'я поточного користувача
    user = User.query.filter_by(username=current_user).first()

    # Якщо секрет 2FA ще не створений — генеруємо новий
    if not user.two_factor_secret:
        user.two_factor_secret = pyotp.random_base32()
        db.session.commit()

    # Формуємо URI для додатку-генератора коду (Google Authenticator тощо)
    otp_uri = pyotp.totp.TOTP(user.two_factor_secret).provisioning_uri(
        name=user.username, issuer_name="SmartLockSystem"
    )

    # Генеруємо QR-код для цього URI
    img = qrcode.make(otp_uri)
    buffer = io.BytesIO()
    img.save(buffer, format='PNG')
    buffer.seek(0)

    # Відправляємо QR-код клієнту для сканування
    return send_file(buffer, mimetype='image/png')

# Роут для реєстрації нового користувача
@app.route('/register', methods=['POST'])
def register():
    username = request.json.get('username')
    password = request.json.get('password')

    # Перевірка, чи існує користувач з таким логіном
    if User.query.filter_by(username=username).first():
        return jsonify({'error': 'Username already exists'}), 400

    # Хешуємо пароль bcrypt
    hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')

    # Створюємо нового користувача у базі
    new_user = User(username=username, password=hashed_password, role="user")
    db.session.add(new_user)
    db.session.commit()

    return jsonify({'message': 'User registered successfully'}), 201

# Роут для логіну з перевіркою пароля, 2FA та блокуванням після невдалих спроб
@app.route("/login", methods=["POST"])
def login():
    data = request.json
    username = data.get("username")
    password = data.get("password")
    token_2fa = data.get("2fa_code")
```

```
    now = datetime.utcnow()

    # Перевіряємо, чи не заблокований користувач
    attempts = login_attempts.get(username, {"count": 0, "blocked_until": None})
    if attempts["blocked_until"] and now < attempts["blocked_until"]:
        remaining = int((attempts["blocked_until"] - now).total_seconds())
        return jsonify({"message": f"Too many failed attempts. Try again in
{remaining} seconds."}), 403

    user = User.query.filter_by(username=username).first()
    # Перевірка пароля
    if not user or not bcrypt.check_password_hash(user.password, password):
        log_action(username, "Failed login (wrong credentials)")
        # Оновлення лічильника спроб
        if username not in login_attempts:
            login_attempts[username] = {"count": 1, "blocked_until": None}
        else:
            login_attempts[username]["count"] += 1
            # Блокування при перевищенні порогу
            if login_attempts[username]["count"] >= BLOCK_THRESHOLD:
                login_attempts[username]["blocked_until"] = now + BLOCK_DURATION
        return jsonify({"message": "Invalid credentials"}), 401

    # Якщо ввімкнена 2FA — перевіряємо код
    if user.two_factor_secret:
        if not token_2fa:
            log_action(username, "2FA required")
            return jsonify({"message": "2FA code required"}), 401

        totp = pyotp.TOTP(user.two_factor_secret)
        if not totp.verify(token_2fa):
            log_action(username, "Failed 2FA")
            return jsonify({"message": "Invalid 2FA code"}), 401

    # Успішний вхід — скидаємо лічильник невдалих спроб
    login_attempts.pop(username, None)
    log_action(username, "User logged in (2FA passed)")

    # Генеруємо JWT токен доступу
    access_token = create_access_token(identity=user.username)
    return jsonify({"token": access_token})

# Глобальна змінна для зберігання статусу замка (True — заблоковано, False —
розблоковано)
lock_status = {"locked": True}

# Роут для керування замком (заблокувати/розблокувати)
@app.route("/lock", methods=["POST"])
@jwt_required()  # Тільки авторизовані користувачі
def lock_control():
    global lock_status
    current_identity = get_jwt_identity()
    data = request.json

    # Перевіряємо, чи вказано коректну дію
    if "action" in data and data["action"] in ["lock", "unlock"]:
        lock_status["locked"] = (data["action"] == "lock")
        log_action(current_identity, f"Lock {data['action']}")

        # Якщо розблоковано — запускаємо таймер для автоматичного повторного
блокування
        if data["action"] == "unlock":
            threading.Thread(target=auto_lock, daemon=True).start()

        return jsonify({"message": f"Lock is now {'locked' if
lock_status['locked'] else 'unlocked'}"})

    log_action(current_identity, f"Failed action: {data.get('action',
```

```
'unknown')}")
    return jsonify({"message": "Invalid action"}), 400

# Функція автоматичного блокування замка через певний час (за замовчуванням 60
секунд)
def auto_lock(delay=60):
    time.sleep(delay)
    with app.app_context():
        if not lock_status["locked"]:
            lock_status["locked"] = True
            log_action("system", "Lock auto-locked after timeout")

# Роут для отримання поточного статусу замка
@app.route("/lock/status", methods=["GET"])
@jwt_required()
def lock_status_check():
    return jsonify({"locked": lock_status["locked"]})

# Захищений роут, доступний лише авторизованим користувачам
@app.route('/protected', methods=['GET'])
@jwt_required()  # Вимагає JWT авторизацію
def protected():
    return jsonify({"message": "Access granted to protected resource!"})  #
Повертає підтвердження доступу

# Роут для видалення користувача (доступний лише адміністраторам)
@app.route("/user/delete", methods=["DELETE"])
@jwt_required()  # Вимагає JWT авторизацію
def delete_user():
    current_identity = get_jwt_identity()  # Отримуємо ім'я поточного
користувача з JWT
    current_user = User.query.filter_by(username=current_identity).first()  #
Завантажуємо дані користувача з БД

    # Перевірка, чи має користувач роль адміністратора
    if current_user.role != "admin":
        log_action(current_identity, "Unauthorized attempt to delete user")  #
Логування спроби
        return jsonify({"message": "Admin privileges required"}), 403  # Відмова
у доступі

    username_to_delete = request.json.get("username")  # Ім'я користувача для
видалення
    if not username_to_delete:
        return jsonify({"message": "No username provided"}), 400  # Помилка,
якщо ім'я не передано

    # Адміністратор не може видалити самого себе
    if username_to_delete == current_identity:
        return jsonify({"message": "Admin cannot delete themselves"}), 400

    # Пошук користувача, якого потрібно видалити
    user_to_delete = User.query.filter_by(username=username_to_delete).first()
    if not user_to_delete:
        log_action(current_identity, f"Failed user deletion: User
{username_to_delete} not found")  # Логування помилки
        return jsonify({"message": "User not found"}), 404

    # Видалення користувача з бази даних
    db.session.delete(user_to_delete)
    db.session.commit()
    log_action(current_identity, f"User {username_to_delete} deleted
successfully")  # Логування успішної операції

    return jsonify({"message": f"User {username_to_delete} deleted
successfully"}), 200  # Успішна відповідь

# Роут для отримання логів дій користувачів (доступний лише адміністраторам)
@app.route("/logs", methods=["GET"])
```

```
@jwt_required()  # Вимагає JWT авторизацію
def get_logs():
    current_identity = get_jwt_identity()  # Отримання імені поточного
користувача
    current_user = User.query.filter_by(username=current_identity).first()  #
Завантаження користувача

    # Перевірка, чи є користувач адміністратором
    if current_user.role != "admin":
        log_action(current_identity, "Unauthorized access to logs")  # Логування
спроби доступу
        return jsonify({"message": "Admin privileges required"}), 403  # Відмова
у доступі

    # Вибірка всіх логів у зворотному хронологічному порядку
    logs = Log.query.order_by(Log.timestamp.desc()).all()
    # Формування списку словників для відправки у відповідь
    log_list = [{
        "username": log.username,
        "action": log.action,
        "timestamp": log.timestamp.strftime("%Y-%m-%d %H:%M:%S")
    } for log in logs]

    log_action(current_identity, "Viewed logs")  # Логування перегляду логів
    return jsonify(log_list), 200  # Відповідь зі списком логів

# Запуск Flask додатку у режимі відладки
if __name__ == "__main__":
    app.run(debug=True)
```

**Файл test_api.py – Тестувальник API Flask-серверу**

```
import pytest
import json
from flask import Flask
from main import app, db, User  # Імпортуємо Flask-додаток і базу даних


@pytest.fixture
def client():
    with app.app_context():  # Ensures application context is available
        with app.test_client() as client:
            # Before each test, clear the database
            db.create_all()
            yield client
            # After each test, drop the database (or perform any cleanup)
            db.session.remove()
            db.drop_all()


# Тест реєстрації користувача
def test_register(client):
    response = client.post('/register', json={
        'username': 'testuser',
        'password': 'testpassword'
    })
    data = json.loads(response.data)
    assert response.status_code == 201
    assert data['message'] == 'User registered successfully'

    # Перевіряємо, чи користувач доданий у базу
    user = User.query.filter_by(username='testuser').first()
    assert user is not None


# Тест логіну користувача
def test_login(client):
    # Спочатку реєструємо користувача
    client.post('/register', json={
        'username': 'testuser',
        'password': 'testpassword'
    })

    # Тепер пробуємо увійти
    response = client.post('/login', json={
        'username': 'testuser',
        'password': 'testpassword'
    })
    data = json.loads(response.data)
    assert response.status_code == 200
    assert 'token' in data


# Тест 2FA
def test_2fa_setup(client):
    # Спочатку реєструємо користувача
    client.post('/register', json={
        'username': 'testuser',
        'password': 'testpassword'
    })

    # Логін
    login_response = client.post('/login', json={
        'username': 'testuser',
        'password': 'testpassword'
    })
    token = json.loads(login_response.data)['token']
```

```
    # Тепер запитуємо 2FA
    response = client.get('/2fa/setup', headers={
        'Authorization': f'Bearer {token}'
    })
    assert response.status_code == 200
    assert response.mimetype == 'image/png'


# Тест керування замком
def test_lock_control(client):
    # Спочатку реєструємо користувача
    client.post('/register', json={
        'username': 'testuser',
        'password': 'testpassword'
    })

    # Логін
    login_response = client.post('/login', json={
        'username': 'testuser',
        'password': 'testpassword'
    })
    token = json.loads(login_response.data)['token']

    # Перевіряємо початковий стан замка
    response = client.get('/lock/status', headers={
        'Authorization': f'Bearer {token}'
    })
    assert response.status_code == 200
    assert response.json['locked'] is True  # Замок має бути заблокованим

    # Розблоковуємо замок
    response = client.post('/lock', json={'action': 'unlock'}, headers={
        'Authorization': f'Bearer {token}'
    })
    assert response.status_code == 200
    assert 'Lock is now unlocked' in response.json['message']

    # Перевіряємо стан замка після розблокування
    response = client.get('/lock/status', headers={
        'Authorization': f'Bearer {token}'
    })
    assert response.status_code == 200
    assert response.json['locked'] is False  # Замок має бути розблокованим


# Тест для неправильного входу
def test_invalid_login(client):
    response = client.post('/login', json={
        'username': 'wronguser',
        'password': 'wrongpassword'
    })
    data = json.loads(response.data)
    assert response.status_code == 401
    assert data['message'] == 'Invalid credentials'


# Тест для спроби доступу без токена
def test_access_without_token(client):
    response = client.get('/protected')
    assert response.status_code == 401
    assert 'Missing Authorization Header' in response.data.decode()


# Тест для неправильного 2FA
def test_invalid_2fa(client):
    # Спочатку реєструємо користувача
    client.post('/register', json={
        'username': 'testuser',
```

```
            'password': 'testpassword'
        })

        # Логін
        login_response = client.post('/login', json={
            'username': 'testuser',
            'password': 'testpassword'
        })
        token = json.loads(login_response.data)['token']

        # Отримуємо 2FA секрет
        response = client.get('/2fa/setup', headers={
            'Authorization': f'Bearer {token}'
        })

        # Пробуємо увійти з неправильним 2FA кодом
        invalid_2fa_code = "123456"  # Невірний код
        response = client.post('/login', json={
            'username': 'testuser',
            'password': 'testpassword',
            '2fa_code': invalid_2fa_code
        })
        data = json.loads(response.data)
        assert response.status_code == 401
        assert data['message'] == 'Invalid 2FA code'


if __name__ == '__main__':
    pytest.main()
```