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

# Бакалаврська кваліфікаційна робота на тему:

«Захищений мобільний застосунок для зберігання нотаток»

Виконав: студент 4 курсу групи 1БС-21б
спеціальності 125 Кібербезпека
Вадим ПАЛЬЧИК

Керівник: к. т. н., доцент каф. ЗІ
Володимир ГАРНАГА
«13» червня 2025 р.

Рецензент: к. т. н. доц. каф. ПЗ
Володимир МАЙДАНЮК
«13» червня 2025 р.

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

Вінниця ВНТУ – 2025 року

Міністерство освіти і науки України
Вінницький національний технічний університет

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

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

**ЗАВДАННЯ**
**НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**
Вадиму ПАЛЬЧИКУ

1. Тема роботи: «Захищений мобільний застосунок для зберігання нотаток» керівник роботи: Володимир ГАРНАГА, к. т. н., доцент кафедри ЗІ, затверджені наказом ректора ВНТУ від 20 березня 2025 року за № 97.
2. Строк подання студентом роботи 13 червня 2025 р.
3. Вихідні дані до роботи:
   - галузь застосування: захист нотаток у мобільному додатку;
   - метод захисту: використання методу Zero trust architecture на всіх етапах розробки додатку;
   - середовище для реалізації підсистеми: мобільні операційні системи Android та IOS.
4. Зміст текстової частини: Вступ. Аналіз методів та засобів захисту даних у мобільних застосунках. Проєктування застосунку на основі підходів Zero trust architecture. Програмна реалізація та тестування мобільного застосунку з використанням принципів Zero trust. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Схема атаки «Людина посередині» (MITM), схема периметрової моделі безпеки, п'ять основних принципів Zero trust architecture, загальна схема взаємодії з застосунком, загальна схема архітектури додатку, схема автентифікації в застосунку, схема шифрування даних, схема розшифрування даних, схема перевірки пристрою перед входом в застосунок.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| 1 | Володимир ГАРНАГА, к.т.н., доц. каф.ЗІ | 24.03.25 | 10.06.25 |
| 2 | Володимир ГАРНАГА, к.т.н., доц. каф.ЗІ | 25.04.25 | 10.06.25 |
| 3 | Володимир ГАРНАГА, к.т.н., доц. каф.ЗІ | 30.05.25 | 10.06.25 |

7. Дата видачі завдання 21 березня 2025 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів бакалаврської кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз методів та засобів захисту даних у мобільних застосунках | 24.03.25 – 30.03.25 | |
| 2 | Аналіз джерел за напрямком БКР | 31.03.25 – 13.04.25 | |
| 3 | Розробка архітектури програмного застосунку, реалізація, тестування | 14.04.25 – 04.05.25 | |
| 4 | Оформлення пояснювальної записки | 05.05.25 – 29.05.25 | |
| 5 | Попередній захист БКР | 30.05.25 – 02.06.25 | |
| 6 | Виправлення зауважень, підготовка ілюстративного матеріалу | 04.06.25 – 07.06.25 | |
| 7 | Перевірка на наявність текстових запозичень | 08.06.25 – 10.06.25 | |
| 8 | Представлення БКР до захисту, рецензування | 11.06.25 – 20.06.25 | |

Студент _____ Вадим ПАЛЬЧИК

Керівник роботи _____ Володимир ГАРНАГА

# АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 71 сторінки формату А4, містить 19 рисунків та 7 таблиць. Список використаних джерел налічує 31 найменування.

Робота присвячена підвищенню безпеки персональних даних у мобільних застосунках шляхом впровадження принципів Zero Trust Architecture. Проведено аналіз сучасних загроз, методів шифрування, автентифікації та контролю середовища. Обґрунтовано архітектурні рішення для захисту даних. Розроблено мобільний застосунок з підтримкою підтвердження email, симетричним шифруванням ключа, що генерується динамічно, та перевіркою середовища. Проведено тестування функціоналу й перевірку безпечності бібліотек.

Ключові слова: конфіденційні дані, мобільний застосунок, шифрування, алгоритм aes, firebase, автентифікація, zero trust architecture, cybersecurity, flutter, пароль.

# ABSTRACT

The bachelor's qualification work consists of 71 A4–sized pages, including 19 figures and 7 tables. The reference list contains 29 sources.

The work focuses on enhancing personal data security in mobile applications using Zero Trust Architecture principles. It analyzes modern threats, encryption, authentication, and environment control methods. Architectural decisions for data protection are substantiated. A mobile application was developed with email confirmation, dynamically generated symmetric encryption keys, and environment checks. The functionality was tested, and the security of used libraries was assessed.



Keywords: confidential data, mobile application, encryption, AES algorithm, Firebase, authentication, zero trust architecture, cybersecurity, Flutter, password.

**ЗМІСТ**

# ВСТУП

Сучасний світ дедалі більше залежить від мобільних технологій, які стали ключовим інструментом для зберігання, передавання та обробки персональних даних. Мобільні застосунки використовуються для обміну повідомленнями, керування фінансами, збереження конфіденційних записів тощо. Водночас зростає і кількість кіберзагроз, що націлені на компрометацію особистої інформації користувачів.

Найпоширенішими уразливостями мобільних застосунків залишаються несанкціонований доступ до даних, зберігання інформації у відкритому вигляді, використання небезпечних бібліотек під час розробки та недостатній контроль над середовищем виконання. У зв’язку з цим постає необхідність переходу від традиційних моделей безпеки до більш гнучких і стійких рішень, одним із яких є концепція Zero Trust Architecture (ZTA). Zero Trust передбачає, що жоден компонент або користувач не заслуговує на довіру за замовчуванням. Кожен доступ перевіряється, кожна дія – верифікується, кожне середовище – оцінюється. Такий підхід особливо актуальний для мобільних платформ, де пристрій може бути втрачений, зламаний або скомпрометований.

Метою роботи є підвищення захищеності мобільного застосунку для зберігання нотаток за рахунок використання принципів Zero Trust Architecture. Для досягнення поставленої мети у роботі необхідно виконати такі задачі:

- проаналізувати сучасні загрози та підходи до захисту даних;
- обґрунтувати вибір технологій та архітектурних рішень;
- реалізувати мобільний застосунок з підтримкою автентифікації, шифрування та перевірки середовища;
- виконати тестування та оцінку ефективності впроваджених механізмів.

Практична цінність роботи полягає в тому, що створене рішення демонструє можливість інтеграції сучасних підходів до безпеки в реальні мобільні продукти без втрати зручності для користувача.

# 1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ЗАХИСТУ ДАНИХ У МОБІЛЬНИХ ЗАСТОСУНКАХ

## 1.1 Сучасні загрози безпеки в мобільних застосунках

Мобільні застосунки посідають провідне місце в цифровій інфраструктурі сучасної людини, забезпечуючи доступ до фінансових сервісів, особистих повідомлень, хмарних сховищ, медичних та державних послуг. Водночас, їх поширеність і функціональна насиченість зумовлює зростання кількості загроз, пов'язаних із безпекою збереження та обробки даних. Актуальність захисту даних у мобільних додатках посилюється через високу частку чутливої інформації, яка ними обробляється, та зростання кількості кібератак на мобільні платформи [1].

### 1.1.1 Небезпека локального зберігання даних

Одним із найпоширеніших векторів атак є неналежне зберігання конфіденційної інформації на пристрої користувача. Наприклад, численні застосунки зберігають нотатки, токени автентифікації, ключі або паролі в незахищених файлових системах або в SQLite–базах без шифрування. Зловмисник, що має фізичний доступ до пристрою або отримав root–доступ, може без перешкод експортувати такі дані [2].

Особливу небезпеку становлять застосунки, які кешують облікові дані або сесійні токени у незашифрованому вигляді. Наприклад, збереження даних автентифікації у SharedPreferences без шифрування створює вразливість до інструментів реверс–інжинірингу або емуляції середовища.

### 1.1.2 Загрози, пов'язані з передачею даних

Невикористання шифрування або застосування застарілих протоколів передачі даних (HTTP, TLS 1.0/1.1) спричиняє значні ризики перехоплення чутливої інформації під час передачі між клієнтом і сервером. Класичним прикладом є атака типу «людина посередині» (MITM), яка реалізується через створення фальшивої точки доступу Wi–Fi, через яку проходить незашифрований трафік [3]. На рисунку 1.1 відображено графічне представлення атаки MITM, що

дозволяє наглядно зрозуміти, потенційну безпеку при використання застарілих протоколів передачі даних.

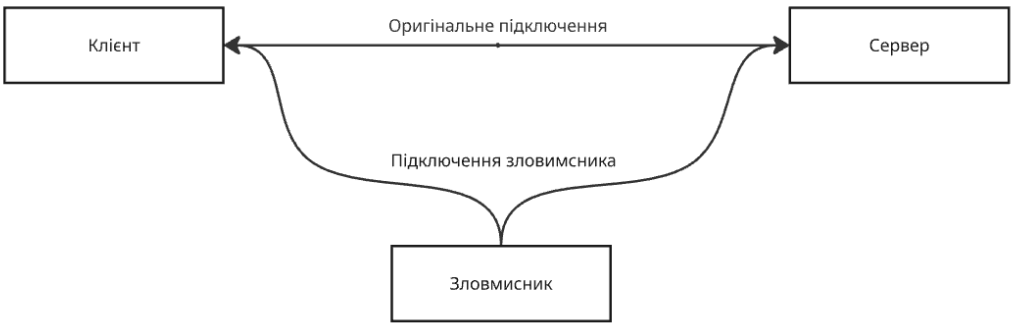



Рисунок 1.1 – Схема атаки «людина посередині» (MITM)

Окрему увагу заслуговує проблема відсутності SSL pinning, що дозволяє перехоплювати навіть зашифровані з’єднання за допомогою встановлення довіреного сертифіката вручну. Це є особливо небезпечним для застосунків, що працюють із фінансовою або медичною інформацією.

1.1.3 Уразливості в інтерфейсах прикладного програмування (API)

Мобільні застосунки часто взаємодіють з серверною частиною через REST або GraphQL API. У випадку неналежного контролю авторизації, відсутності перевірки прав доступу або реалізації відкритих кінцевих точок без обмеження кількості запитів (rate limiting), можливе проведення атак масового витоку інформації.

Прикладом є ситуація, коли API GET /user/{id} не перевіряє, чи має поточний користувач право на доступ до ресурсу, що призводить до повного витоку даних інших користувачів [4].

1.1.4 Соціальна інженерія та зловмисні SDK

Окрім технічних вразливостей, однією з головних загроз залишаються методи соціальної інженерії. Користувачів змушують інсталювати шкідливі застосунки, маскуючи їх під корисні або імітуючи функціонал популярних сервісів. Додатки часто запитують надлишкові дозволи, які не відповідають їхній основній функції, наприклад, доступ до SMS, мікрофона, місцезнаходження тощо [5].

Небезпеку також становлять сторонні SDK, що інтегруються у мобільні застосунки. Відомі випадки, коли рекламні або аналітичні SDK збирали конфіденційні дані користувача й передавали їх на сторонні сервери без згоди користувача.

### 1.1.5 OWASP Mobile Top 10

Організація OWASP (Open Web Application Security Project) регулярно оновлює перелік найкритичніших загроз безпеці мобільних застосунків. Остання версія, OWASP Mobile Top 10 – 2024, відображає сучасні виклики та тенденції у сфері мобільної безпеки [6]. У таблиці 1.1 наведено перелік найкритичніших вразливостей у мобільниxx застосунках.

Таблиця 1.1 – Перелік найкритичніших категорій вразливостей у мобільних застосунках

| № | Категорія | Короткий опис |
|---|---|---|
| M1 | Improper Credential Usage | Використання жорстко закодованих облікових даних, ненадійне зберігання або передача облікових даних. |
| M2 | Inadequate Supply Chain Security | Використання вразливих або скомпрометованих сторонніх бібліотек, SDK, інструментів. |
| M3 | Insecure Authentication/Authorization | Відсутність або неналежна реалізація механізмів автентифікації та авторизації. |
| M4 | Insufficient Input/Output Validation | Відсутність або недостатня перевірка даних, що вводяться або виводяться, що може призвести до атак. |
| M5 | Insecure Communication | Відсутність шифрування або використання застарілих протоколів для передачі даних. |
| M6 | Inadequate Privacy Controls | Відсутність або недостатній захист персональних даних користувачів. |
| M7 | Insufficient Binary Protections | Відсутність обфускації, захисту від реверс–інжинірингу та модифікації коду. |
| M8 | Security Misconfiguration | Неправильні або небезпечні налаштування безпеки, які можуть бути використані зловмисниками. |
| M9 | Insecure Data Storage | Зберігання чутливих даних у незашифрованому вигляді або у вразливих місцях. |
| M10 | Insufficient Cryptography | Використання слабких або застарілих криптографічних алгоритмів або неправильна їх реалізація. |

## 1.2 Традиційні методи захисту даних у мобільних додатках

Протягом останніх десятиліть традиційна модель забезпечення безпеки мобільних додатків ґрунтувалася на концепції периметрового захисту (perimeter–based security), яка передбачає наявність чіткої межі між "довіреною" внутрішньою інфраструктурою і "недовіреним" зовнішнім світом. У цій моделі всі компоненти всередині периметра вважаються безпечними, а захист зосереджується на контролі доступу ззовні. Незважаючи на те, що такий підхід широко використовувався у корпоративних системах, його застосування у мобільних платформах є обмеженим через розподілену природу мобільного середовища та відсутність єдиного центру керування довірою [7].

### 1.2.1 Захист даних: sandboxing та файлові політики

Мобільні операційні системи, зокрема Android та iOS, реалізують базовий рівень ізоляції застосунків через механізм sandboxing [8]. Кожен додаток виконується у власному середовищі та не має доступу до даних інших додатків без спеціальних дозволів. Android, наприклад, ізолює дані кожного пакета у власному каталозі, а iOS використовує контейнерну модель з окремими доступами до файлів, ключів і фото.

Додатково застосовуються такі механізми:

- keychain (iOS) – безпечне сховище для облікових даних;
- encryptedSharedPreferences (Android) – зберігання даних з використанням AES; [9].
- biometricPrompt або Face ID/Touch ID – контроль доступу до даних через біометрію.

Однак, на практиці, застосунки часто зберігають інформацію у відкритому вигляді або без криптографічного захисту. Наприклад, кешування нотаток або токенів у SQLite без шифрування все ще є поширеною практикою.

### 1.2.2 Захист передавання даних: TLS та сертифікати

Передавання даних між мобільним клієнтом і сервером традиційно захищається за допомогою протоколу TLS (Transport Layer Security). Для цього

використовуються стандартні HTTPS–з'єднання, де сертифікати перевіряються клієнтом автоматично. Проте багато застосунків відключають перевірку сертифіката (accept all SSL), не реалізують TLS pinning (зв'язування з конкретним сертифікатом), підтримують лише застарілі версії TLS (1.0, 1.1), які мають відомі уразливості [10].

Застосунок, який не перевіряє сертифікат, може легко стати жертвою атаки MITM, при якій шкідливий Wi–Fi–провайдер підмінює сертифікат і читає трафік.

1.2.3 Автентифікація та контроль доступу

Базова автентифікація в мобільних застосунках зазвичай реалізується через email/логін та пароль, рідше – через біометричні методи або одноразові коди. Для розширеного захисту іноді застосовується двофакторна автентифікація (2FA), зокрема, з отриманням SMS, через TOTP (Time–Based One–Time Passwords), та пуш–повідомлення [11].

Контроль доступу найчастіше реалізується у вигляді токенів (JWT, OAuth2), що зберігаються локально. Але при неналежному зберіганні токенів у SharedPreferences або Keychain без обмежень по часу/контексту доступу можливе захоплення сесії. В таблиці 1.2 відображено резулььтатпорівняння способів автентифікації.

Таблиця 1.2 – Результат порівняння способів автентифікації

| Спосіб автентифікації | Рівень захисту | Вразливість до атак | Потреба в користувацьких діях |
|---|---|---|---|
| Пароль | Низький | Брутфорс, фішинг | Висока |
| SMS–код | Середній | SIM swap, перехоплення | Середня |
| TOTP | Високий | Уразливість клієнта | Висока |
| Біометрія (Face ID, тощо) | Високий | Спуфінг, обхід через root | Низька |
| Push–based MFA | Дуже високий | Залежить від серверів | Низька |

Традиційні методи забезпечують базовий рівень захисту, однак їх ефективність обмежується тим, що вони часто передбачають "довіру за замовчуванням" до пристрою, мережі чи користувача після початкової автентифікації. У мобільному середовищі, де користувачі постійно змінюють мережі, геолокацію і пристрої, такий підхід виявляється недостатнім. Це створює передумови для впровадження новітніх концепцій, зокрема Zero Trust Architecture, що не допускає жодної довіри без перевірки – незалежно від місця доступу.

### 1.3 Недоліки периметрової моделі в умовах сучасних загроз

Традиційна периметрова модель безпеки, яка активно використовувалася в архітектурах корпоративних мереж і систем, вважає, що загрози надходять лише ззовні, тоді як внутрішнім суб’єктам можна довіряти. У контексті мобільних застосунків, де відсутнє чітке розмежування між внутрішнім і зовнішнім середовищем, така модель втрачає ефективність. З розвитком хмарних технологій, мультиплатформенного доступу, використання BYOD–пристроїв і підвищення складності атак, концепція периметру фактично втратила свою практичну цінність. На рисунку 1.2 зображено графічне відображення моделі perimeter based, що дозволяє ширше побачити потенційну небезпеку в такій моделі захисту.

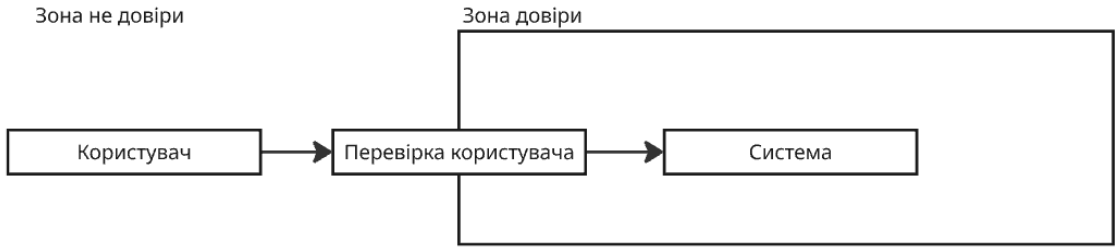



Рисунок 1.2 – Графічне зображення периметрової моделі безпеки

На рисунку 1.2 відображено дві області: «Зона довіри» та «Зона недовіри». Зона недовіри охоплює всі компоненти, що перебувають поза межами контрольованого середовища – зокрема користувача до моменту автентифікації. Після проходження перевірки (аутентифікації та авторизації) користувач потрапляє

у зону довіри, яка умовно вважається безпечною і в якій надається повний або частковий доступ до ресурсів системи [12].

Периметрова модель безпеки, на якій базується ця схема, передбачає, що всі внутрішні компоненти системи після входу вважаються довіреними, і лише на етапі входу реалізується перевірка. У цьому підході основна увага зосереджена на захисті периметру (тобто межі між недовіреним і довіреним середовищем), тоді як подальші дії користувача не супроводжуються додатковим моніторингом чи перевіркою контексту доступу.

1.3.1 Проблема довіри до пристрою

У периметровій моделі вся автентифікація виконується на етапі входу, після чого доступ до даних або функцій надається без подальших перевірок. Однак у мобільному середовищі користувач може:

- змінити геолокацію;
- підключитися до відкритої Wi–Fi мережі;
- використовувати скомпрометований або root–ований пристрій.

На основі таких даних потенційно можу відбутись наступні атака: користувач проходить автентифікацію в мобільному банкінгу в безпечному середовищі, але вже за кілька хвилин підключається до відкритої мережі, і токен залишається дійсним. Зловмисник перехоплює трафік або токен доступу без додаткової перевірки [13].

Така поведінка критично суперечить вимогам безпеки в сучасному середовищі, де кожен доступ має розглядатись як потенційно небезпечний.

У рамках класичної моделі користувач після автентифікації отримує повний доступ до дозволених функцій. Немає механізмів повторної перевірки на основі контексту (наприклад, зміни IP, геолокації або часу доби). Периметрова модель не підтримує динамічну адаптацію політик доступу, що створює значну уразливість.

1.3.2 Неможливість виявлення і реагування на загрози в реальному час

У традиційній моделі система не проводить повторну валідацію після входу користувача. Це створює «плоске поле довіри», де внутрішній користувач – навіть скомпрометований – має доступ без обмежень. Відсутність постійного

моніторингу, логування дій у реальному часі, а також аналітики поведінки користувачів (UEBA) робить систему вразливою до внутрішніх атак [14].

У випадку витоку токену або крадіжки пристрою зловмисник може здійснювати дії від імені користувача без жодного контролю. Периметрова модель не передбачає механізмів припинення сесії або повторного запиту автентифікації в критичних точках.

Враховуючи динамічний характер мобільного середовища, підвищення складності атак та необхідність адаптації до поведінкових і контекстуальних змін, периметрова модель безпеки більше не може вважатися достатньою. Вона не забезпечує актуальну перевірку довіри, ігнорує ризики, пов’язані з пристроями, мережами та поведінкою користувача після автентифікації. Це створює передумови для впровадження більш гнучкої, контекстно–орієнтованої та багаторівневої моделі.

### 1.4 Регламентні документи та стандарти ZTA

Для побудови ефективної системи захисту на основі концепції Zero Trust Architecture (ZTA) важливо керуватись не лише теоретичними принципами, а й офіційними нормативними документами. Одним із ключових джерел, що регламентує основні положення та рекомендації щодо впровадження архітектури ZTA, є документ NIST Special Publication 800–207, опублікований Національним інститутом стандартів і технологій США (NIST).

У цьому документі Zero Trust визначається як концепція кібербезпеки, що передбачає відсутність довіри за замовчуванням до будь–якого суб’єкта або пристрою, незалежно від того, знаходиться він у внутрішній або зовнішній мережі. Кожен запит на доступ розглядається як потенційно небезпечний, а перевірка автентичності, авторизації та контексту проводиться постійно [15].

Згідно з NIST SP 800–207, Zero Trust Architecture ґрунтується на таких базових принципах:

– перевірка кожного доступу (Continuous verification);

– мінімізація привілеїв (Least privilege);
– аналіз контексту (Risk–based policies and context–aware decisions);
– сегментація середовища (Micro–segmentation);
– динамічне прийняття рішень на основі поведінкової аналітики.

На основі вищезазначених принципів Zero Trust Architecture сформовано таблицю 1.3, яка демонструє їхню відповідність до конкретних механізмів, що можливо реалізувати у мобільному застосунку.

Таблиця 1.3 – Можлива реалізація принципів Zero Trust Architecture у мобільному застосунку

| Принцип ZTA (NIST SP 800–207) | Суть | Реалізація у додатку |
|---|---|---|
| Continuous verification | Перевірка кожного запиту | Перевірка root, мережі, аномальних дій |
| Least privilege | Мінімум доступу | Користувачу доступні лише свої дані |
| Context–aware decisions | Доступ залежить від середовища | Тип мережі, контекст пристрою |
| Micro–segmentation | Ізоляція об’єктів | Шифрування даних |
| Behavioral analysis | Аналіз дій користувача | Виявлення дій, що потенційно небезпечні, повторний запит доступу |

Таблиця ілюструє, яким чином кожен принцип був адаптований до практичного середовища мобільного додатку з метою забезпечення цілісної та контекстно–залежної системи контролю доступу.

Важливим аспектом стандарту є також модель компонування системи ZTA, що включає:

– policy Engine (PE) – модуль, що ухвалює рішення про доступ;
– policy Enforcement Point (PEP) – точка, яка застосовує рішення;

– policy Administrator (PA) – компоненти, що відповідають за створення і поширення політик доступу;
– trust Algorithm – алгоритми оцінки довіри до запиту.

## 1.5 Концепція Zero Trust Architecture

Зростання кількості атак на мобільні додатки, є передумовою відмови від класичної периметрової моделі захисту та необхідністю забезпечення високої гнучкості й адаптивності в умовах багатоплатформеного середовища, з'явилась нова парадигма – Zero Trust Architecture (ZTA). Її ключова ідея – нікому не довіряй за замовчуванням, завжди перевіряй.

Zero Trust – це не конкретна технологія, а стратегія побудови захисту, в якій усі користувачі, пристрої, мережі та навіть внутрішні сервіси розглядаються як потенційно небезпечні. Таким чином, перевірка автентичності, авторизації, поведінки та відповідності політикам безпеки виконується на кожному рівні доступу, а не лише під час входу [16 – 17].

### 1.5.1 Основні принципи Zero Trust Architecture

Модель Zero Trust базується на принциповій відмові від довіри за замовчуванням у будь–яких взаємодіях між суб'єктами інформаційної системи. Всі запити до ресурсів, незалежно від джерела – чи то зовнішній користувач, внутрішній компонент або сервіс, – мають проходити перевірку перед наданням доступу. Кожен запит до інформаційного ресурсу повинен бути явно авторизований і підтверджений на основі контексту, до якого належать параметри середовища (геолокація, тип пристрою, мережа), ступінь довіри до пристрою, наявність актуальних засобів захисту, попередній шаблон поведінки користувача, а також оцінка ризику.

Важливим концептом є застосування принципу найменших привілеїв: кожному суб'єкту (користувачу, процесу, пристрою) надається лише той мінімальний обсяг доступу, який необхідний для виконання конкретного завдання, і лише на визначений час. Усі ресурси системи мають бути ізольованими один від

одного за допомогою мікросегментації, що дозволяє обмежити масштаб компрометації у випадку вторгнення.

Значну роль у Zero Trust відіграє механізм постійного моніторингу дій користувача та пристрою, що дозволяє виявляти аномальні дії в реальному часі. Наприклад, навіть після успішної автентифікації мобільний додаток може повторно ініціювати перевірку або заблокувати сесію, якщо користувач раптово змінює регіон доступу, використовує новий пристрій або підключається через підозрілу мережу. Таким чином, довіра до користувача в системі не є сталою, а формується динамічно, відповідно до зміни умов взаємодії [18].

На рисунку 1.3 відображено 5 основних принципів методу захисту Zero Trust.

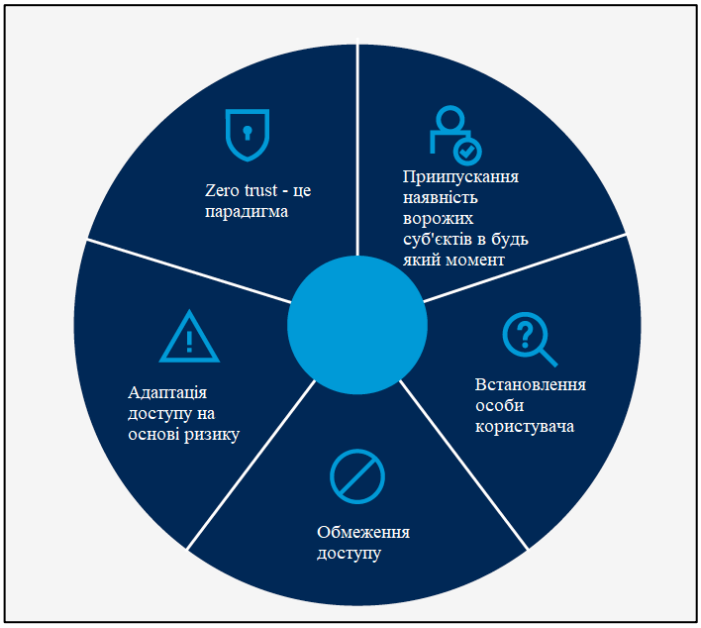



Рисунок 1.3 – П’ять основних принциипів Zero Trust Architecture

### 1.5.2 Архітектура Zero Trust у мобільних додатках

У реалізації моделі Zero Trust для мобільних застосунків критично важливо дотримуватись балансу між рівнем захисту, зручністю користувача та продуктивністю системи. Технічно впровадження концепції ZTA вимагає узгодженої роботи кількох ключових компонентів: системи автентифікації, модуля управління політиками доступу, засобів шифрування та контролю середовища виконання.

На етапі первинної автентифікації найоптимальнішим рішенням є використання багатофакторної автентифікації, при якій поряд із введенням логіна та пароля застосовується одноразовий код (наприклад, на основі протоколу TOTP) або пуш–підтвердження. З практичної точки зору, найменше навантаження створює TOTP, оскільки генерація коду виконується локально на пристрої і не потребує інтернет–з'єднання, на відміну від push–MFA, яка залежить від стабільного з'єднання з сервером.

Після проходження автентифікації доступ до функціоналу не повинен надаватися автоматично на весь час сесії. Архітектура має враховувати перевірку контексту доступу: зміна геолокації, мережі або часу доби може вважатися підозрілою подією, яка вимагає повторного запиту до системи автентифікації або тимчасового обмеження дій користувача. Цей контроль контексту реалізовується за допомогою модуля оцінки ризику на клієнтській стороні – зокрема, перевіряється, чи активовано VPN, чи змінено timezone, чи є root–доступ, і чи відповідає пристрій політиці безпеки (наприклад, відсутність root–доступу та наявність захисту екрану).

Ще одним критичним компонентом Zero Trust у мобільному застосунку є реалізація шифрування даних на стороні клієнта. Це особливо важливо для застосунків, які працюють із чутливою інформацією – наприклад, особистими нотатками. З міркувань продуктивності та простоти реалізації доцільним є використання симетричного шифрування (наприклад, AES–256), де ключ для шифрування генерується з пароля користувача за допомогою функції PBKDF2 або Argon2. Такий підхід дозволяє повністю уникнути зберігання ключів на сервері, що знижує ризики при можливій компрометації бекенду.

Архітектура повинна також включати логіку керування політиками доступу на стороні клієнта: наприклад, реалізацію тайм–ауту сесії, локального блокування при неактивності, а також розмежування прав доступу в межах функціоналу (читання, створення, редагування або видалення нотаток). Це дозволяє знизити потенційну шкоду навіть у випадку часткового компрометування акаунту або пристрою.

### 1.5.3 Переваги впровадження Zero Trust у мобільних застосунках

У порівнянні з традиційними моделями, впровадження Zero Trust у мобільному середовищі дає змогу досягти значно вищого рівня захисту даних без істотного погіршення користувацького досвіду. Постійна перевірка контексту доступу та використання шифрування на стороні клієнта дозволяють знизити залежність від безпеки мережі або серверної інфраструктури. Наявність незалежного контролю дій користувача в реальному часі унеможливлює приховане використання сесій зловмисником, навіть якщо той отримає валідні токени або пароль.

Завдяки тому, що в системі відсутнє поняття "довіри за замовчуванням", кожна дія оцінюється як потенційно небезпечна, що кардинально знижує ефективність класичних векторів атак. При цьому оптимізація модулів безпеки – зокрема, застосування двофакторної автентифікації з підтвердженям електронної пошти, PBKDF2 та симетричного шифрування – дозволяє досягти високого рівня ефективності при мінімальних затратах часу та обчислювальних ресурсів [19].

У результаті, модель Zero Trust стає не лише теоретичною відповіддю на новітні кіберзагрози, але й практичним, ефективним та відносно легко реалізовуваним підходом у межах мобільного застосунку, орієнтованого на зберігання чутливої інформації. Саме тому вона була обрана як концептуальна основа системи безпеки, яка реалізується у межах цієї бакалаврської кваліфікаційної роботи.

## 1.6 Порівняння Zero Trust та Периметрового підходу

Периметрова модель захисту (perimeter–based security), яка була стандартом у традиційних інформаційних системах, довгий час забезпечувала базову безпеку через створення логічного або фізичного кордону між довіреною та недовіреною зоною. Проте з еволюцією цифрових сервісів, розширенням мобільних середовищ, використанням хмарних рішень і відкритих API, ця модель стала втрачати свою

ефективність. Модель Zero Trust (ZTA) була розроблена як відповідь на ці обмеження.

1.6.1 Основні відмінності моделей

Головною відмінністю Zero Trust є відмова від апріорної довіри. Якщо у периметровій моделі після автентифікації користувач вважається довіреним і може вільно взаємодіяти з системою, то у ZTA кожен доступ потребує повторної, контекстно–залежної перевірки. Це суттєво змінює логіку побудови захищених сервісів, зокрема мобільних додатків.

У традиційній системі захист зосереджений переважно на вході в систему, тоді як у Zero Trust – на постійному контролі дій, зміни середовища (пристрою, мережі, поведінки користувача) та мікросегментації. Це дозволяє зменшити вплив компрометації на решту системи.

1.6.2 Приклад застосування в мобільному середовищі

Для мобільного застосунку, що зберігає приватні нотатки, використання периметрового підходу означало б, що після введення пароля користувач має повний доступ до даних. Якщо пристрій втрачається або скомпрометований, зловмисник отримує усю інформацію без обмежень. У Zero Trust–додатку дані шифруються на клієнті, кожна дія (наприклад, спроба видалення великого обсягу нотаток) тригерить контекстну перевірку. Навіть якщо токен автентифікації буде викрадений, він стане непридатним, якщо середовище доступу не відповідатиме очікуваному (новий IP, незнайомий пристрій, інша країна).

В таблиці 1.4 відображено результати порівняння підходів Zero Trust та Perimeter–based security. Та прораховано відповідні бали за плюси/недоліки використання такого підходу, якщо це перевага підходу, підхід отримує 1 бал, якщо недолік підходу, то – 0 балів.

Як показує порівняння, периметрова модель, хоч і проста у реалізації, але не відповідає вимогам безпеки у мобільному середовищі, через що отримала 2 бали.

Таблиця 1.4 – Результати порівняння підходяв Zero Trust та Perimeter–based security

| Ознака | Perimeter–based security | Zero Trust Architecture | Бали підходу Perimeter–based security | Бали підходу Zero Trust Architecture |
|---|---|---|---|---|
| Підхід до довіри | Один раз при вході | Жодної довіри без перевірки | 0 | 1 |
| Перевірка середовища доступу | Відсутня | Обов’язкова на кожному етапі | 0 | 1 |
| Захист після автентифікації | Відсутній | Динамічний, безперервний | 0 | 1 |
| Захист даних на клієнті | Опціональний | Принциповий (шифрування, MFA) | 1 | 1 |
| Виявлення аномальної поведінки | Нерелевантне | Обов’язкова частина | 0 | 1 |
| Контроль дій після входу | Мінімальний | Повний моніторинг сесії | 0 | 1 |
| Витрати на реалізацію | Нижчі | Вищі, але виправдані рівнем захисту | 1 | 0 |
| Сумарний бал | | | 2 | 6 |

Zero Trust забезпечує контрольований доступ, адаптивний захист, а головне – постійне підтвердження довіри до кожної взаємодії, що є критично важливим для роботи з конфіденційною інформацією, в наслідок порівняння отримала 6 балів, що відображає явну домінацію над підходом perimeter–based security [20].

## 1.7 Обґрунтування вибору підходу Zero Trust для захищеного мобільного застосунку

На основі попереднього аналізу можна дійти висновку, що традиційні методи захисту, особливо засновані на периметровій моделі, в умовах мобільного

середовища виявляються обмеженими та вразливими до актуальних кіберзагроз. Вони не забезпечують необхідного рівня захисту при втраті пристрою, компрометації токенів доступу або зміні контексту (наприклад, переходу на відкриту мережу чи вхід з нового пристрою). Саме тому в рамках цієї бакалаврської роботи для реалізації мобільного застосунку обрано архітектуру, побудовану на принципах Zero Trust.

### 1.7.1 Відповідність ZTA мобільному середовищу

Мобільні додатки функціонують у динамічному середовищі з постійною зміною мереж, пристроїв. У таких умовах класичні підходи до безпеки, побудовані навколо фіксованого периметру, не дають змоги забезпечити повноцінний контроль над діями користувача та станом середовища доступу. Zero Trust Architecture, навпаки, передбачає перевірку кожного запиту, незалежно від походження, з урахуванням множини факторів ризику.

У мобільному контексті це означає, що система має постійно верифікувати не лише особу користувача, а й стан пристрою, з якого здійснюється доступ, наявність VPN–з'єднання, поведінкові патерни та інші ознаки потенційного ризику. Це дозволяє виявляти підозрілу активність навіть у межах чинної сесії, що є критично важливим у випадку, коли зловмисник отримає частковий доступ до облікового запису.

Zero Trust також адаптивно реагує на ризик змін у середовищі: наприклад, при переході в іншу часову зону, спробі доступу з нового пристрою або після тривалої неактивності система не надає довіри автоматично, як це властиво класичним моделям, а запитує повторну автентифікацію або блокує сесію [21].

### 1.7.2 Переваги у сфері захисту конфіденційних даних

Основним завданням розроблюваного мобільного застосунку є збереження чутливої інформації – особистих нотаток користувача. Такий тип даних вимагає особливо ретельного захисту, оскільки будь–яке несанкціоноване втручання може призвести до порушення конфіденційності або репутаційних ризиків.

Zero Trust дозволяє запровадити гнучке управління доступом не лише на етапі входу в систему, а й у процесі взаємодії з даними. Наприклад, навіть якщо

користувач отримав доступ до застосунку, система може обмежити можливість видалення або експорту великої кількості даних, якщо ці дії не збігаються з типовим сценарієм використання. Це дозволяє істотно зменшити ймовірність зловживання, навіть у разі компрометації облікових даних.

Крім того, модель Zero Trust дозволяє уникнути залежності від довіри до конкретного пристрою або середовища, завдяки чому система зберігає свою стійкість навіть у випадку втрати смартфона або несанкціонованого доступу до файлів користувача.

1.7.3 Узагальнення обґрунтування

Ураховуючи все зазначене, можна зробити висновок, що Zero Trust Architecture є найбільш відповідною концепцією безпеки для мобільного застосунку, який обробляє конфіденційні нотатки. Вона враховує динамічну природу мобільного середовища, не покладається на постійне середовище довіри, забезпечує перевірку кожного доступу, а також дозволяє швидко реагувати на зміну поведінки користувача або параметрів пристрою. Такий підхід не лише підвищує загальний рівень інформаційної безпеки, а й забезпечує масштабованість рішення, що є важливим фактором у разі подальшого розвитку застосунку.

## 1.8 Висновки до розділу

У першому розділі бакалаврської роботи проведено комплексний аналіз сучасних методів та засобів захисту даних у мобільних застосунках. Показано, що зростання кількості мобільних загроз, зокрема пов’язаних з неналежним зберіганням інформації, передачею даних у незашифрованому вигляді, використанням скомпрометованих SDK та соціальною інженерією, потребує принципово нових підходів до організації безпеки. У якості сучасної альтернативи було розглянуто концепцію Zero Trust Architecture, яка забезпечує контроль кожного запиту незалежно від джерела та підтримує динамічну перевірку контексту взаємодії.

# 2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ НА ОСНОВІ ПІДХОДІВ ZERO TRUST ARCHITECTURE

У цьому розділі наведено процес проєктування захищеного мобільного застосунку для зберігання нотаток з урахуванням принципів Zero Trust Architecture. Розглянуто архітектурні рішення, функціональні вимоги, а також практичні засоби реалізації автентифікації, шифрування даних, перевірки середовища виконання та захисту від підозрілої активності користувача. Усі механізми спроєктовані таким чином, щоб забезпечити максимальний рівень довіри до дій користувача, пристрою та мережі відповідно до концепції «Не довіряй нікому, перевіряй усе».

Застосунок створено з урахуванням вимог до безпеки персональних даних у мобільному середовищі, де традиційні методи захисту виявляються недостатніми. Запропоновані підходи дозволяють знизити ризики компрометації навіть у разі втрати пристрою, перехоплення трафіку чи використання недовіреного середовища.

## 2.1 Цілі, функціональні вимоги та сценарії використання

Основною метою проєктування даного мобільного застосунку є забезпечення надійного зберігання нотаток користувача з реалізацією сучасного підходу Zero Trust Architecture. Додаток має гарантувати цілісність і конфіденційність даних незалежно від середовища запуску, а також забезпечити багаторівневу перевірку користувача і пристрою на всіх етапах взаємодії із системою.

До основних цілей додатку належать:

- безпечне створення, редагування, збереження та перегляд нотаток;
- забезпечення доступу до даних з кількох пристроїв лише для автентифікованого користувача;
- шифрування даних перед збереженням у хмарному середовищі;
- виявлення та блокування підозрілої активності або потенційно небезпечного середовища;
- мінімізація впливу захисних механізмів на зручність користувача.

На рисунку 2.1 представлено загальну схему взаємодії користувача із застосунком, яка демонструє основні етапи доступу та взаємодії з даними в межах концепції Zero Trust, що означає додаткова перевірка користувача, можлива після проходження користувачем першого процесу автентифікації та початку роботі з додатком [22].

Після автентифікації з підтвердженням поштою здійснюється перевірка середовища. Далі генерується ключ шифрування на основі пароля користувача, після чого відкривається доступ до нотаток. У зоні взаємодії передбачено можливість повторної перевірки у разі підозрілих дій. Такий підхід відповідає принципам Zero Trust – система не довіряє автоматично жодному етапу взаємодії.

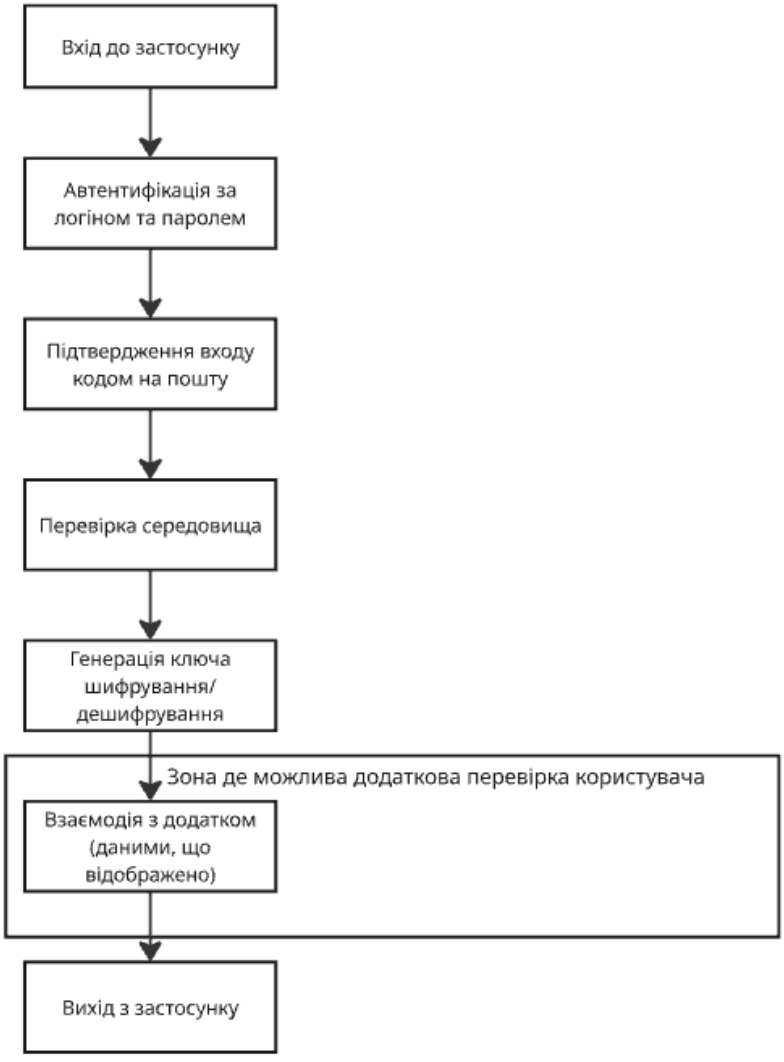



Рисунок 2.1 – Загальна схема взаємодії з застосунком

З точку зору функціоналності, додаток реалізовує основні вимоги наведені в таблиці 2.1

Таблиця 2.1 – Основні вимоги до функціональності

| Назва вимоги | Опис функціоналу |
|---|---|
| Автентифікація | Вхід за email і паролем з підтвердженням коду, надісланого на електронну пошту |
| Перевірка середовища | Виявлення root, емуляторів, шкідливих мереж |
| Шифрування | Локальне шифрування кожної нотатки перед збереженням у Firestore |
| Підозріла поведінка | Виявлення масових змін, видалень, спроб злому |
| Блокування візуального доступу | Заборона скріншотів, запису екрану, автозавершення сесії |
| Синхронізація | Відновлення нотаток із Firestore при вході з іншого пристрою |

## 2.2 Архітектура додатку та її відповідність принципам Zero Trust

Мобільний застосунок повинен реалізовувати архітектуру типу клієнт–хмара з локальним шифруванням даних і збереженням у хмарному сховищі Firebase Firestore. При цьому жодна частина системи не вважається «довіреною» – відповідно до принципів Zero Trust кожен етап перевіряється незалежно.

Основою в побудові архітектури додатку повинні складати такі компоненти:

- інтерфейс користувача (Flutter): забезпечує взаємодію з системою;
- firebase Authentication: автентифікація користувача за email і паролем з підтвердженням коду з електронної пошти;
- локальний криптомодуль: генерує ключ шифрування на основі пароля користувача (з використанням PBKDF2);
- firestore: база даних для збереження зашифрованих нотаток;

– zero Trust middleware: перевіряє середовище перед доступом до даних (наявність root–доступу, тип мережі, емулятор тощо).

Усі компоненти архітектури взаємодіють таким чином, щоб забезпечити мінімальний рівень довіри на кожному етапі. Наприклад, навіть після успішного входу до облікового запису, користувач не отримає доступу до нотаток, якщо його пристрій виявиться скомпрометованим (root–доступ, запуск в емуляторі тощо). Шифрування реалізується повністю на стороні клієнта, а ключ шифрування обчислюється щоразу на основі введеного пароля, що виключає його зберігання в пам’яті або на сервері.

Перед відображенням нотаток система здійснює остаточну перевірку середовища, після чого відбувається дешифрування даних, що були отримані з Firestore. Такий підхід дозволяє реалізувати не лише безпеку доступу, а й цілісність та конфіденційність даних, незалежно від стану хмарної інфраструктури [23].

На рисунку 2.2 відображено загальну архітектуру додатку.


Рисунок 2.2 – Загальна архітектура додатку

На рисунку наведено загальну архітектуру мобільного застосунку, яка побудована за принципом рівневого поділу відповідальності. У верхній частині знаходиться рівень користувацького інтерфейсу, що забезпечує взаємодію з користувачем та відображення вмісту – його реалізовано за допомогою фреймворку Flutter. Наступним рівнем є бізнес–логіка, яка відповідає за обробку дій користувача, зокрема перевірку середовища, запуск механізмів шифрування, обробку аномальних ситуацій тощо. Всі правила доступу та реакції на поведінку користувача реалізуються саме в цій частині архітекруті додатку.

Рівень моделі даних визначає структуру об’єктів, які використовуються в системі: нотатки, ключі, проміжні токени автентифікації. Тут відбувається серіалізація/десеріалізація, а також підготовка даних до шифрування або дешифрування. В основі архітектури – інфраструктурний рівень, що включає сервіси Firebase: Firestore для зберігання зашифрованих даних, Authentication для входу в обліковий запис, а також допоміжні сервіси (відправка коду підтвердження, FlutterSecureStorage для збереження тимчасової інформації тощо).

Такий рівневий підхід дозволяє реалізувати гнучку й масштабовану архітектуру, яка легко піддається розширенню, тестуванню й оновленню без порушення загальної структури. Крім того, саме така побудова відповідає вимогам Zero Trust: на кожному з рівнів реалізовані власні перевірки та обмеження, які унеможливлюють неконтрольований доступ до даних.

### 2.3 Автентифікація з двофакторною перевіркою

Процес автентифікації в додатку реалізовано у два етапи. Спочатку користувач вводить логін (email) і пароль, які перевіряються через сервіс Firebase Authentication. Якщо дані вірні, система генерує одноразовий код підтвердження й надсилає його на вказану електронну пошту. Для завершення входу користувач має ввести цей код у додатку. Лише після коректного введення обох компонентів система переходить до перевірки середовища та ініціалізації сеансу.

Такий підхід відповідає моделі Zero Trust, оскільки передбачає перевірку не лише облікових даних, а й додаткового каналу підтвердження – зокрема, контролю за доступом до поштової скриньки. На відміну від статичних паролів, код, надісланий на пошту, є одноразовим і має обмежений термін дії (3 хвилини), що значно знижує ризик його перехоплення або зловживання.

Обраний спосіб автентифікації є надійним завдяки кільком чинникам. По–перше, він базується на двох різних каналах автентифікації – знанні (пароль) і володінні (доступ до поштової скриньки). По–друге, всі перевірки виконуються на захищеній стороні Firebase, що мінімізує ризик маніпуляцій із боку клієнта. По–третє, у додатку реалізовано механізм захисту від перебору облікових даних: після п’яти невдалих спроб входу обліковий запис блокується на 5 хвилин, що унеможливлює атаки типу brute–force, оскільки потрібно дуже багато часу для перебору паролів [24].

Таким чином, двофакторна автентифікація, реалізована через email–код із часовим обмеженням, дозволяє досягти балансу між зручністю користування, складністю реалізації та рівнем захисту необхідним для роботи з конфіденційними даними у мобільному середовищі.

### 2.4 Шифрування даних перед зберіганням

З метою захисту змісту нотаток, що зберігаються у хмарному середовищі Firebase Firestore, у додатку реалізовано симетричне шифрування на клієнтській стороні. Це означає, що весь текст нотатки шифрується до моменту збереження у базі, а сервер не має жодного доступу до відкритих даних, що забезпечує безпеку даних навіть в випадку взлому бази даних.

Для шифрування використано алгоритм AES (Advanced Encryption Standard) у режимі GCM (Galois/Counter Mode), що забезпечує не лише конфіденційність, а й цілісність даних за допомогою вбудованого механізму автентифікації (authentication tag). Такий вибір зумовлений низкою переваг AES–GCM у мобільному середовищі:

- висока продуктивність навіть на пристроях зі середньою обчислювальною потужністю;
- підтримка апаратного прискорення на більшості сучасних процесорів;
- стійкість до відомих криптоаналітичних атак;
- наявність широкої підтримки у Flutter–бібліотеках (encrypt, pointycastle тощо).

Ключ для шифрування не зберігається на пристрої або у хмарі – натомість, він щоразу виводиться динамічно на основі логіна та пароля користувача за допомогою механізму PBKDF2 (Password–Based Key Derivation Function 2). Це дозволяє отримати криптографічно стійкий ключ заданої довжини (наприклад, 256 біт) із введених користувачем даних, доповнених випадковою сіллю, яка зберігається локально.

Додатковим аргументом на користь AES–GCM є те, що цей алгоритм офіційно затверджений у якості симетричного блочного шифру стандартом NIST (Національний інститут стандартів і технологій США) і рекомендований до використання в урядових системах, що обробляють конфіденційну інформацію. Його перевагою є підтримка режиму AEAD (Authenticated Encryption with Associated Data), що одночасно забезпечує конфіденційність, цілісність і автентичність даних – тобто навіть незначна зміна в зашифрованому вмісті викличе помилку при розшифруванні [25].

Для мобільного середовища це особливо важливо: AES–GCM може працювати апаратно за допомогою сучасних ARM–процесорів, що значно прискорює шифрування/дешифрування й знижує навантаження на акумулятор. У Flutter–екосистемі існують перевірені бібліотеки, які надають просту інтеграцію AES–GCM без втрати безпеки – зокрема encrypt, flutter_secure_storage, cryptography тощо.

Алгоритм шифрування AES–GCM було обрано на основі порівняльного аналізу альтернативних алгоритмів, згідно таблиці 2.2

Таблиця 2.2 – Результат порівняння алгоритмів шифрування

| Алгоритм | Тип | Продуктивність | Стійкість | Підтрика на мобільних пристроях | Примітки |
|---|---|---|---|---|---|
| AES–GCM | Симетричний AEAD | Висока | Висока | + | Обраний варіант |
| ChaCha20–Poly1305 | Симетричний AEAD | Висока | Висока | + | Стабільна альтернатива AES, але рідше підтримується апаратно |
| RSA–2048 | Асиметричний | Низька | Висока | – | Не підходить для великих обсягів даних, складне керування ключами |
| ECC | Асиметричний | Середня | Висока | – | Переважно використовується для ключової угоди, а не для шифрування великих даних |

## 2.5 Перевірка безпечності середовища виконання

У межах реалізації концепції Zero Trust Architecture мобільний застосунок виконує комплексну перевірку безпечності середовища кожного разу, коли користувач ініціює взаємодію із системою. Навіть після успішної автентифікації не передбачається жодної автоматичної довіри – усе середовище розглядається як потенційно ворожа зона. Такий підхід дозволяє виявляти загрози не лише на етапі входу, а й у процесі використання додатку.

Перевірки середовища виконання реалізовані у вигляді модулів, що працюють локально у клієнтській частині додатку. Вони активуються при кожному запуску застосунку, а також після виконання ризикованих дій – наприклад,

масового редагування або видалення нотаток. У цьому розділі розглянуто три ключові напрями перевірки.

2.5.1 Виявлення ознак компрометації пристрою

Значна частина атак на мобільні застосунки ґрунтується на тому, що користувач виконує їх у скомпрометованому середовищі. Для цього у додатку реалізовано виявлення root–доступу (Android) або jailbreak (iOS), що є ознаками втрати цілісності ОС. Застосунок виявляє наявність шкідливих або нестандартних системних компонентів (наприклад, файли su, директорії magisk, підозрілі процеси тощо), а також перевіряє індикатори емуляторів і віртуальних середовищ. У разі виявлення компрометації додаток блокує доступ до нотаток або дозволяє лише обмежений режим перегляду без можливості редагування.

2.5.2 Аналіз мережевого середовища

Ще одним аспектом безпеки є надійність мережі, через яку здійснюється доступ до сервісів. Застосунок перевіряє тип мережі (Wi–Fi, мобільна, VPN), визначає, чи є мережа відкритою або потенційно небезпечною. Виявлення підключення до відкритих точок доступу Wi–Fi без шифрування може спричинити повторну автентифікацію або запуск додаткової перевірки середовища.

2.5.3 Оцінка поведінки користувача та аномалій

Для забезпечення динамічного реагування на потенційні загрози додаток постійно аналізує поведінку користувача. Зокрема, фіксується кількість нотаток, які створюються, змінюються або видаляються у короткий проміжок часу. Якщо така активність виходить за межі звичайної поведінки, система може викликати додаткову перевірку середовища, обмежити доступ або вимагати повторного введення пароля.

Також перевіряється коректність системного часу та синхронізація з сервером – спроби змінити час або дату на пристрої можуть бути свідченням втручання з метою обману механізмів шифрування або TTL токенів.

### 2.6 Захист логіки та взаємодії з інтерфейсом

Захист логіки додатку та його інтерфейсної взаємодії є не менш важливими елементами безпеки, ніж автентифікація чи шифрування даних. У межах реалізації архітектури Zero Trust важливо гарантувати, що користувач не зможе обійти визначені правила поведінки в застосунку або змінити спосіб взаємодії з конфіденційною інформацією через інтерфейс чи зловживання API.

Одним із базових рішень стала заборона зняття скріншотів та запису екрана. У мобільному застосунку використано спеціальні налаштування безпеки на рівні Android (FLAG_SECURE) та аналогічні API на iOS. Це унеможливлює спроби збереження або перехоплення чутливої інформації з екрана під час використання. Такий підхід є обов’язковим для додатків, що працюють з приватними нотатками, фінансовими чи медичними даними [26].

Для захисту логіки роботи реалізовано обмеження на масові операції. Це дає змогу виявляти автоматизовані сценарії або дії після компрометації акаунту. Реалізовано контроль подій не лише за кількістю, а й за частотою – наприклад, три видалення нотаток менш ніж за 10 секунд активують обмеження.

Також додаток не дозволяє редагування в зашифрованому вигляді без повторної перевірки. Перед кожною критичною дією (видалення, перезапис або масове оновлення) застосунок повторно ініціює перевірку середовища та автентифікацію ключа шифрування.

Крім того, інтерфейс реалізовано таким чином, щоб мінімізувати ризик випадкових натискань або доступу до функцій, які потребують підвищеного рівня довіри. Наприклад, функція «видалити всі нотатки» відсутня взагалі.

Інтерфейсна логіка побудована на принципах контекстної адаптації: функції та кнопки з’являються тільки після успішного проходження відповідних перевірок. Наприклад, доступ до кнопки «редагувати» з’являється лише після локального дешифрування нотатки з правильним ключем.

Усі ці рішення спрямовані на реалізацію принципу мінімального доступу та мінімального інтерфейсу, що є центральними у концепції Zero Trust. В результаті,

навіть якщо зловмисник отримає тимчасовий доступ до пристрою, можливості взаємодії з критичним функціоналом будуть обмежені як на рівні інтерфейсу, так і на рівні логіки.

### 2.7 Перевірка безпечності бібліотек, що використовуватимуться

У рамках реалізації мобільного застосунку, побудованого за принципами Zero Trust Architecture, особливу увагу приділяється вибору сторонніх бібліотек, що використовуються для забезпечення шифрування, збереження даних та загальної логіки безпеки. Відповідно до концепції «нульової довіри», жодна бібліотека не вважається безпечною за замовчуванням – кожна повинна проходити перевірку за чітко визначеними критеріями.

До основних критеріїв за якими необхідно виконати тестуванння бібліотеки належить:

- наявність відкритого вихідного коду;
- активність спільноти та регулярні оновлення;
- для бібліотек, що реалізують шифруання відповідність стандартам криптографії;
- відсутність мережевої активності або прихованих запитів;
- публічна документація та прозора історія змін;
- наявність попередніх результатів аудиту або відгуків фахівців.

Для перевірки безпечності бібліотек та виявлення можливих вразливостей планується використання спеціалізованих інструментів автоматичного аналізу, а саме: Snyk – платформа для перевірки залежностей на наявність відомих вразливостей (CVEs); OSS Security Scanner – інструмент для аналізу open–source бібліотек у проєктах на Dart/Flutter; Aikido Security – сервіс для безперервного моніторингу безпеки з репозиторіїв GitHub/GitLab, який виявляє CVE, секрети, шкідливі бібліотеки [27].

Попередній відбір бібліотек здійснюється з урахуванням зазначених критеріїв, тоді як фактичне тестування бібліотек на наявність вразливостей та

відповідність вимогам безпеки буде проведено у наступному розділі, присвяченому практичній реалізації та тестуванню системи.

### 2.8 Висновки до розділу

У цьому розділі було розроблено проєктну модель захищеного мобільного застосунку для зберігання нотаток, побудовану відповідно до принципів Zero Trust Architecture. Було визначено ключові цілі захисту інформації, обґрунтовано вибір архітектури клієнт–хмара з локальним шифруванням та проведено детальне обґрунтування кожного з реалізованих механізмів безпеки.

Реалізована багаторівнева автентифікація з підтвердженням через email дозволяє мінімізувати ризик несанкціонованого доступу, а шифрування даних за допомогою AES–GCM із ключем, що обчислюється з пароля, забезпечує конфіденційність навіть у разі втрати пристрою або компрометації сховища.

Особливу увагу приділено перевірці середовища виконання, що дозволяє формувати динамічний профіль довіри користувача. Застосунок активно перевіряє наявність root-доступу, аналізує мережу та виявляє аномальну поведінку, що відповідає принципу постійної недовіри до будь–якого елемента системи.

Крім того, реалізовано захист логіки додатку та взаємодії з інтерфейсом – блокування скріншотів, обмеження на масові дії, динамічне управління доступом до функцій.

Всі описані механізми працюють у сукупності, формуючи цілісну модель безпеки, яка враховує обмеження мобільного середовища та загрози сучасної цифрової взаємодії. Запропоноване рішення є масштабованим, легко підтримується та відповідає актуальним рекомендаціям провідних галузевих стандартів.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ З ВИКОРИСТАННЯМ ПРИНЦИПІВ ZERO TRUST

У цьому розділі наведено опис програмної реалізації мобільного застосунку, створеного з урахуванням принципів Zero Trust. Деталізовано процес впровадження механізмів автентифікації, шифрування, перевірки середовища та контролю доступу до конфіденційних даних. Представлено фрагменти коду, що демонструють роботу ключових компонентів системи захисту, а також схеми, які ілюструють логіку перевірок і взаємодії елементів системи. Особливу увагу приділено практичному застосуванню моделі нульової довіри в контексті мобільного середовища. У завершальній частині наведено результати тестування функціональних модулів та аналіз використаних бібліотек з точки зору безпеки.

## 3.1 Загальні положення реалізації

Мобільний застосунок було реалізовано з використанням кросплатформеного фреймворку Flutter, який базується на мові програмування Dart. Даний вибір зумовлено необхідністю створення захищеного додатку, що працює на різних мобільних платформах – зокрема Android та iOS, а також надає гнучкий інтерфейс розробки з високою швидкістю оновлення інтерфейсу та коду. Flutter дозволяє інтегрувати криптографічні бібліотеки, сервіси захисту даних і реалізовувати принципи Zero Trust Architecture без обмежень з боку середовища виконання.

Розробка здійснювалася у середовищі Visual Studio Code з розширеннями Dart/Flutter, Android SDK та емулятором Android Studio. Для тестування використовувалися як віртуальні, так і фізичні пристрої. Застосунок використовує сучасний підхід до організації коду, у якому чітко розмежовано відповідальність компонентів. Усі ключові функціональні частини реалізовані як окремі модулі або сервіси, що забезпечує простоту підтримки та зручність перевірки безпеки кожного з них.

Архітектура застосунку побудована за принципом поділу відповідальностей між модулями. Всі компоненти згруповано за функціональним призначенням:

- інтерфейс користувача реалізований у вигляді окремих екранів взаємодії з користувачем: додавання нотатки, перегляд, вхід, підтвердження дій;
- сервіси безпеки включають перевірку середовища виконання (root–доступ, мережа), генерацію ключів шифрування, шифрування і розшифрування вмісту;
- менеджери даних відповідають за створення, оновлення, збереження і видалення нотаток, які зберігаються у хмарній базі даних Firebase Firestore;
- логіка автентифікації охоплює вхід через email і пароль, валідацію пароля при критичних діях, а також генерацію ключа на основі пароля;
- керування станом застосунку реалізовано через Provider і обробник життєвого циклу AppLifecycleObserver, що забезпечує реакцію на блокування екрана, зміну активності чи інші події.

На відміну від підходу з локальними базами, в даному випадку усі нотатки зберігаються у Firestore лише у зашифрованому вигляді. Це дозволяє реалізувати принцип «нульової довіри» до зовнішніх сервісів – навіть у разі компрометації облікового запису Firebase вміст нотаток залишиться недоступним без ключа, який зберігається лише на пристрої користувача.

Уся логіка шифрування побудована відповідно до моделі локального формування ключа. Після введення пароля та логіну користувачем, система застосовує алгоритм PBKDF2 для генерації симетричного ключа, який надалі використовується для шифрування і розшифрування вмісту нотаток. Весь вміст, що зберігається у Firestore, попередньо обробляється криптографічними функціями, тому жоден елемент не зберігається у відкритому вигляді [28].

Ключ шифрування не передається на сервер і не зберігається в хмарі – він тимчасово доступний у пам’яті пристрою тільки після локальної авторизації. Сіль, необхідна для виведення ключа, створюється під час реєстрації користувача і зберігається окремо у Firebase. Таким чином, кожен користувач має унікальний криптографічний контекст.

Сіль створюється на етапі реєстрації користувача й зберігається у Firestore у відкритому вигляді. Це не становить загрози безпеці, оскільки сіль сама по собі не є секретом. Її основне завдання – забезпечити унікальність ключа навіть при однакових паролях у різних користувачів. Такий підхід відповідає криптографічним рекомендаціям OWASP та NIST.

Усі операції шифрування й розшифрування виконуються на пристрої. Перед тим як зберегти або прочитати нотатку, застосунок викликає сервіс note_crypto_service.dart, який застосовує алгоритм AES–GCM для обробки тексту. Цей режим шифрування забезпечує не лише конфіденційність, але й цілісність даних, оскільки включає автентифікацію повідомлення. Для кожного шифрування генерується новий IV (ініціалізаційний вектор), що зберігається разом із зашифрованим вмістом [29].

Оскільки шифрування відбувається до надсилання даних у Firebase, застосунок повністю дотримується принципу нульового довір’я до зовнішніх систем. Firebase виступає лише як носій зашифрованих об’єктів, не маючи змоги проаналізувати або зламати їхній вміст. Усі критичні дії, як–от перегляд, редагування або видалення нотатки, можливі лише після локального розблокування даних, що знижує ризики навіть у випадку компрометації облікового запису користувача.

Ключовим елементом реалізації принципів Zero Trust є контроль середовища, в якому функціонує мобільний застосунок. У межах проєкту реалізовано декілька перевірок, що дозволяють виявити ризиковані умови та своєчасно блокувати функціональність. Це відповідає принципу "нульового довір’я", який передбачає недовіру до будь–якого пристрою або мережі за замовчуванням.

Модуль root_checker_service.dart аналізує характеристики системи: перевіряється наявність root–доступу на Android, jailbreak на iOS, використання емуляторів або тестових ключів. Ці ознаки є індикаторами скомпрометованого середовища. У разі виявлення небезпечних умов доступ до зашифрованих даних автоматично блокується, а інтерфейс виводить повідомлення про неможливість

використання програми в поточному стані. Це унеможливлює запуск застосунку на зламаному пристрої навіть при наявності правильного пароля.

Додатково реалізовано перевірку стану мережі через network_checker_service.dart. При кожному запуску застосунок аналізує тип підключення: якщо виявлено незашифроване Wi–Fi–з'єднання або відсутність доступу до інтернету, взаємодія з Firebase припиняється, а користувач отримує повідомлення про небезпеку. Таким чином, унеможливлюється передача метаданих у незахищеному вигляді.

Окрему роль відіграє контроль життєвого циклу застосунку. У файлі app_lifecycle_observer.dart реалізовано логіку автоматичного блокування після неактивності: при переході у фоновий режим або при блокуванні екрана застосунок втрачає доступ до ключа розшифрування. При повторному вході користувач змушений знову ввести пароль для поновлення сесії. Це запобігає несанкціонованому доступу до даних у разі тимчасової втрати контролю над пристроєм.

Реалізація таких механізмів створює багаторівневу систему перевірок, що охоплює як внутрішній стан пристрою, так і зовнішні умови. Завдяки цьому забезпечується дотримання одного з базових постулатів Zero Trust – постійна перевірка середовища та мінімізація довіри до будь–яких компонентів системи.

### 3.2 Реалізація автентифікації

У мобільному застосунку реалізовано автентифікацію з використанням хмарного сервісу Firebase Authentication, яка поєднується з локальним криптографічним захистом. Такий підхід дозволяє централізовано зберігати облікові записи користувачів, забезпечуючи при цьому повну локальну обробку конфіденційної інформації відповідно до принципів Zero Trust.

Під час реєстрації користувач вводить адресу електронної пошти та пароль. Для створення облікового запису використовується функція Firebase:

```
await FirebaseAuth.instance
.createUserWithEmailAndPassword(email: email, password: password);
```

Після успішного створення облікового запису в Firebase Authentication користувач автоматично отримує унікальний ідентифікатор uid. Після цього на вказану електронну адресу надсилається лист із посиланням для підтвердження:

```
await userCredential.user?.sendEmailVerification();
```

Паралельно з цим створюється унікальна **сіль**, яка буде використана під час генерації симетричного ключа шифрування. Сіль генерується локально та зберігається у Firestore:

```
await _userDataService.generateAndStoreSaltForUser(
  email: email,
  password: password,
  uid: uid,
);
```

Завдяки цьому жоден ключ шифрування не зберігається в хмарі, що повністю відповідає принципу «мінімальної довіри до середовища». Навіть при повному доступі до бази Firestore зловмисник не зможе розшифрувати дані без знаходження вірного паролю, підбору методу шифрування та виведення ключа.

Після підтвердження email користувач може виконати вхід. При цьому застосунок перевіряє:

- чи існує обліковий запис;
- чи підтверджено адресу електронної пошти;
- правильність облікових даних.

Після підтвердження email користувач отримує можливість увійти до системи. Вхід відбувається за допомогою функції signInWithEmailAndPassword():

```
final userCredential = await FirebaseAuth.instance
  .signInWithEmailAndPassword(email: email, password: password);
```

Далі перевіряється, чи підтверджено електронну адресу:

```
if (!user.emailVerified) {
  await FirebaseAuth.instance.signOut();
  _showMessage('Підтвердіть email перед входом.');
  return;
}
```

Якщо email підтверджено, застосунок отримує унікальний uid, завантажує з Firestore сіль, яка була згенерована при реєстрації, і формує симетричний ключ:

```
final salt = await _userDataService.fetchSaltForUser(uid);
final derivedKey = await _userDataService.encryptionService
  .deriveKeyFromCredentials(
    login: email,
    password: password,
    salt: salt,
  );
```

Цей ключ зберігається лише в оперативній пам’яті, а не в базі даних. Він використовується для подальшого розшифрування даних користувача, які зберігаються у Firestore лише у зашифрованому вигляді. Завершальний крок логіну – перенаправлення на головний екран:

```
Navigator.of(context).pushReplacement(
  MaterialPageRoute(builder: (_) => const HomeScreen()));
```

Усі ці дії реалізуються на клієнтському боці, без передачі пароля або ключів до серверу, що забезпечує повну відповідність принципу «довіра лише після перевірки» в Zero Trust Architecture.

У контексті Zero Trust ключовим є принцип повної недовіри до будь–якого суб’єкта доти, поки не буде доведено протилежне. У реалізованому застосунку це забезпечується кількома механізмами, що стосуються обробки помилок і обмеження доступу.

Одним із них є обмеження кількості спроб входу. Якщо користувач вводить неправильний пароль п’ять разів поспіль, подальші спроби блокуються на 30 секунд. Це відповідає принципу безперервна перевірка (continuous verification), оскільки після кожної невдалої спроби система переглядає статус користувача і ухвалює нове рішення щодо доступу:

```
if (_failedAttempts >= 5) {
  _blockedUntil = DateTime.now().add(const Duration(seconds: 30));
  _showMessage('Забагато спроб. Блокування на 30 секунд.');
```

Рішення щодо блокування приймається на клієнтському боці, без звернень до зовнішнього сервера. Це підсилює локальну автономність системи автентифікації, що є важливою частиною моделі ZTA, згідно з NIST SP 800–207.

Ще один важливий механізм – перевірка стану облікового запису. Після входу за email і паролем застосунок додатково перевіряє, чи підтвердив користувач

електронну пошту. Без підтвердження навіть успішний логін скасовується, а сесія не ініціалізується:

```
if (!user.emailVerified) {
  await FirebaseAuth.instance.signOut();
  _showMessage('Підтвердіть email перед входом.');
  return;
}
```

Цей підхід унеможливлює доступ до даних без доказу контролю над email–адресою, що узгоджується з концепцією explicit verification of every access request.

Крім цього, симетричний ключ шифрування, отриманий через PBKDF2, не зберігається у жодному сховищі. Його існування обмежене лише часом роботи застосунку, що унеможливлює повторне використання або витік через кеш–пам’ять.

Відсутність «довіреного стану» навіть у разі фізичного доступу до пристрою повністю відповідає філософії ZTA, де кожен сеанс і кожна дія потребують верифікації.

Крім перевірки самої наявності акаунта, важливою умовою допуску є саме підтвердження права власності на обліковий запис. Такий контроль реалізовано через механізм обов’язкової верифікації електронної пошти, без якої автентифікація вважається неповною. Це не лише знижує ймовірність автоматизованої реєстрації або атак методом підбору, а й гарантує, що жоден обліковий запис не може бути використаний без участі його власника. Таким чином, система реалізує суворе відокремлення авторизаційного рівня від шифрувального, і лише при повному проходженні перевірки допускає ініціалізацію сесії розшифрування.

Подано рисунок 3.1, що ілюструє повну логіку автентифікації користувача в мобільному застосунку відповідно до принципів Zero Trust Architecture. У схемі відображено послідовність перевірок та умов переходу до додатку.

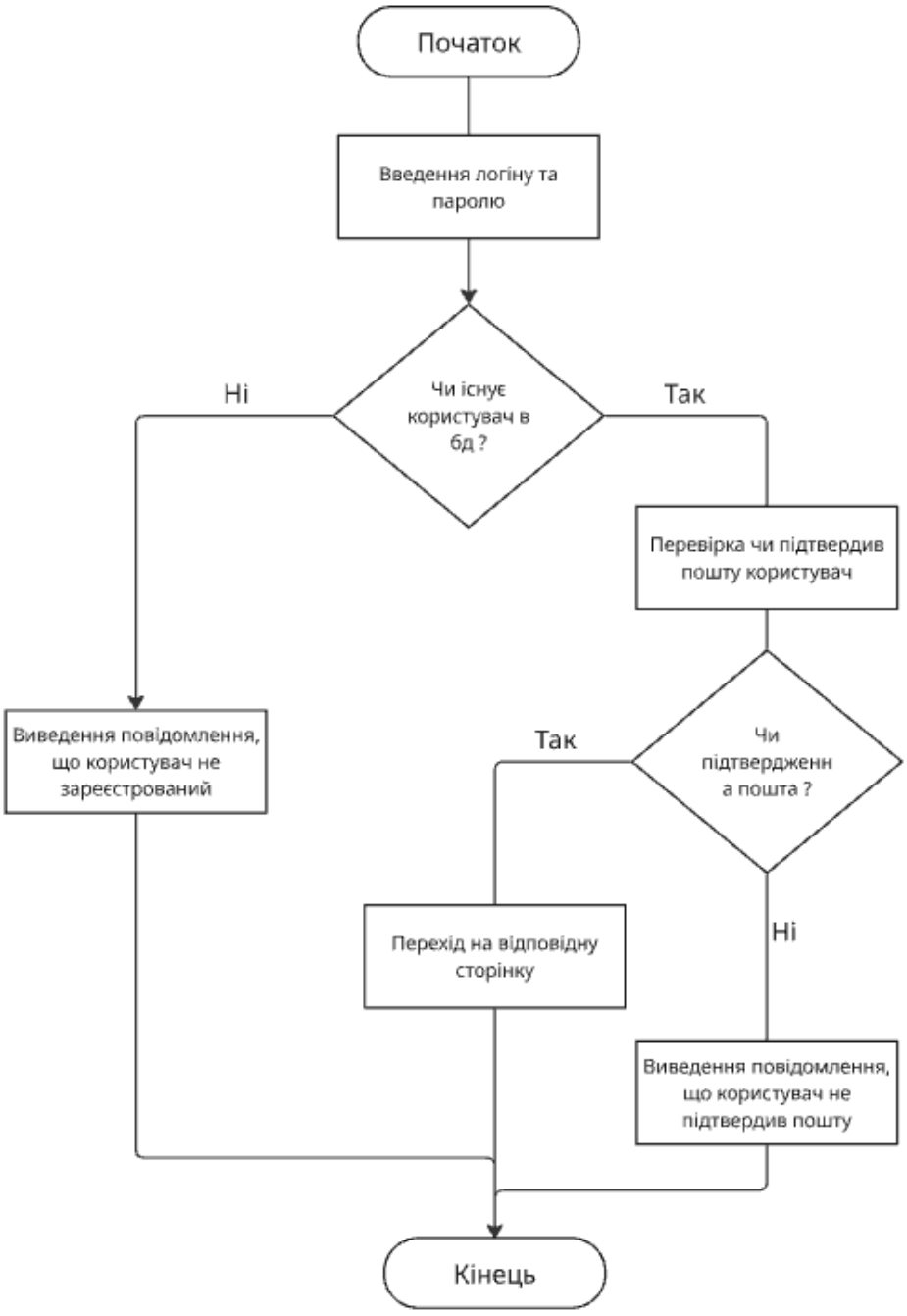



Рисунок 3.1 – Схема автентифікації в застосунку

Реалізований процес забезпечує гарантовану відмову доступу у разі невідповідності будь–якому з етапів перевірки. Таким чином досягається ізоляція та контроль, що відповідає базовим положенням Zero Trust.

## 3.3 Криптографічний захист даних

Основним механізмом захисту в мобільному застосунку є клієнтське шифрування вмісту перед збереженням у Firebase Firestore. Усі нотатки користувача зашифровані на пристрої за допомогою алгоритму AES у режимі GCM, а симетричний ключ, що використовується, генерується локально з пароля через PBKDF2. Такий підхід повністю виключає можливість доступу до вмісту нотаток навіть з боку Firebase або розробника без введення правильного пароля користувачем.

### 3.3.1 Генерація ключа шифрування

Генерація ключа реалізована в класі EncryptionService методом deriveKeyFromCredentials. Ключ створюється з поєднання логіна та пароля користувача із сіллю, збереженою у Firestore під час реєстрації:

```
final combined = utf8.encode('$login:$password');
final key = await pbkdf2.deriveKey(
  secretKey: SecretKey(combined),
  nonce: salt,
);
```

Сіль – це масив із 16 випадкових байтів, який робить кожен виведений ключ унікальним навіть для однакових паролів. Вона не є секретною і зберігається у Firestore відкрито, що відповідає криптографічним рекомендаціям OWASP.

### 3.3.2 Шифрування даних

Шифрування вмісту нотаток відбувається на клієнтському боці, до моменту надсилання даних у Firestore. Це відповідає принципу Zero Trust: система не довіряє хмарній інфраструктурі і зберігає лише вже зашифровані об'єкти.

Для шифрування використовується алгоритм AES у режимі GCM, реалізований засобами пакета cryptography. У кожному виклику шифрування генерується новий вектор ініціалізації (IV), що забезпечує криптостійкість навіть при однаковому вмісті:

```
final algorithm = AesGcm.with256bits();
final nonce = algorithm.newNonce(); // унікальний IV
final secretBox = await algorithm.encrypt(
  utf8.encode(plainText),
  secretKey: key,
```

```
    nonce: nonce,
);
```

Результат шифрування містить:

– cipherText – зашифрований вміст;
– nonce (IV) – необхідний для подальшого дешифрування;
– mac – код автентичності повідомлення.

Ці три параметри перетворюються у Base64 і зберігаються у Firestore у вигляді JSON–структури.

На рисунку 3.2 зображено повний процес шифрування нотатки, починаючи з введення користувачем тексту, завершуючи збереженням зашифрованих компонентів у Firestore.

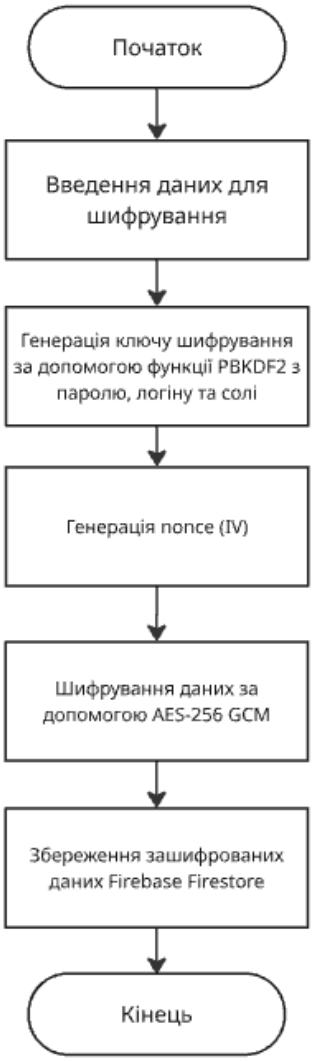



Рисунок 3.2 – Схема шифрування даних

### 3.3.3 Розшифрування даних при доступі

Після успішної автентифікації користувач отримує доступ до зашифрованих нотаток, які зберігаються у Firebase Firestore. Для кожної нотатки застосунок витягує три компоненти:

- cipherText – зашифрований вміст нотатки;
- nonce (IV) – ініціалізаційний вектор;
- mac – код автентичності.

Усі ці дані зберігаються у форматі Base64. Перший крок – їх завантаження з бази даних та декодування:

```
final doc = await FirebaseFirestore.instance
  .collection('notes')
  .doc(noteId)
  .get();

final cipherText = doc['cipherText'].;
final nonce = doc['nonce'].;
final mac = doc['mac'].;
```

Після цього виконується побудова об’єкта SecretBox, який включає зашифровані дані, IV і MAC:

```
final secretBox = SecretBox(
  base64Decode(cipherText),
  nonce: base64Decode(nonce),
  mac: Mac(base64Decode(mac)),
);
```

Далі викликається дешифрування з перевіркою MAC за допомогою AES–GCM:

```
final decryptedBytes = await AesGcm.with256bits().decrypt(
  secretBox,
  secretKey: key,
);
```

Результат перетворюється в рядок:

```
final noteContent = utf8.decode(decryptedBytes);
```

Для гарантії безпеки весь процес обгортається у блок обробки помилок:

```
try {
  final decryptedBytes = await AesGcm.with256bits().decrypt(
    secretBox,
    secretKey: key,
  );
  final noteContent = utf8.decode(decryptedBytes);
  print('Нотатка: $noteContent');
} catch (e) {
  print('Помилка розшифрування: ${e.toString()}');
```

```
    showMessage('Неможливо розшифрувати. Ймовірно, пароль невірний.');
  }
```

Цей процес реалізує локальне контрольоване розшифрування, яке відповідає принципу Zero Trust – навіть після входу користувач не отримує доступ до даних без проходження всіх перевірок і побудови правильного ключа. Усі дешифрувальні операції виконуються виключно на боці пристрою, що виключає можливість втручання сторонніх сервісів.

На рисунку 3.3 відображено повну послідовність дій під час розшифрування вмісту нотатки.


Рисунок 3.3 – Схема розшифрування даних

Таким чином, навіть у випадку втрати пристрою або компрометації хмари, дані користувача залишаються захищеними.

### 3.4 Контроль середовища виконання

Одним із ключових принципів Zero Trust Architecture, відповідно до документа NIST SP 800–207, є постійна перевірка контексту виконання та недовіра до пристрою за замовчуванням. Навіть якщо користувач пройшов автентифікацію, система не повинна надавати доступ до захищених ресурсів, поки не буде впевненості у безпечності середовища. У мобільному застосунку цей підхід реалізовано через модулі перевірки компрометації пристрою та стану мережі.

3.4.1 Перевірка пристрою

Перевірка пристрою виконується у класі RootCheckerService, який використовує бібліотеку device_info_plus. При кожному запуску застосунку визначається тип платформи (Android або iOS), після чого перевіряються відповідні характеристики пристрою. Для Android перевіряється:

- чи є пристрій фізичним (або емульованим);
- чи присутні ознаки root–доступу (наприклад, test–keys, generic бренди або google_sdk моделі);
- чи запущено пристрій у режимі розробника.

```
final isEmulator = !androidInfo.isPhysicalDevice;
final isRooted = tags.contains('test-keys') ||
    brand.contains('generic') ||
    model.contains('google_sdk');
```

У випадку виявлення хоча б одного з критеріїв компрометації функціонал додатку блокується ще до етапу автентифікації. Це дозволяє запобігти несанкціонованому доступу з потенційно скомпрометованих пристроїв, навіть якщо користувач володіє правильним паролем.

Для iOS реалізовано перевірку на емуляцію (симулятор), яка також виключає доступ:

```
if (Platform.isIOS) {
  final isEmulator = !iosInfo.isPhysicalDevice;
```

```
  return isEmulator;
}
```

Окрім перевірки платформи, також застосунок може бути розширений підтримкою перевірки jailbreak через сторонні бібліотеки, якщо цього потребують політики безпеки. Усе це відповідає принципу динамічної оцінки ризиків та контекстного контролю доступу.

На рисунку 3.4, поданій нижче, візуалізовано послідовність перевірок середовища виконання. Схема включає логіку розгалуження за платформами, аналіз ознак компрометації та прийняття рішення – дозволити чи заборонити доступ до застосунку.

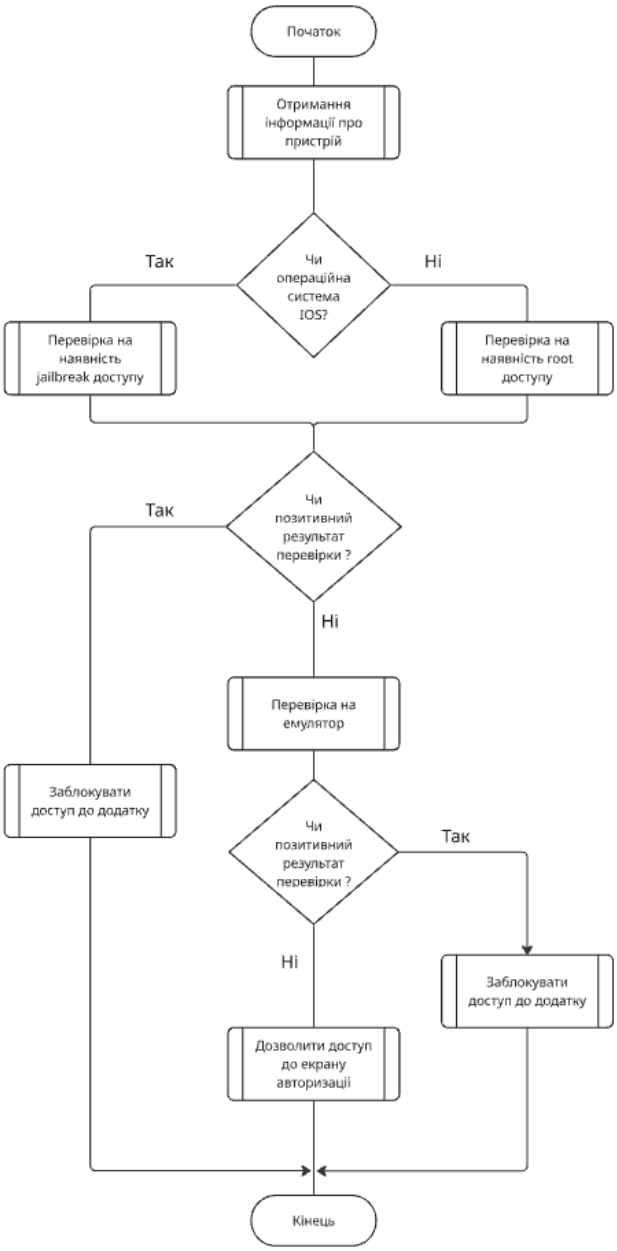



Рисунок 3.4 – Логіка перевірки пристрою перед входом в застосунок

### 3.4.2 Перевірка мережі

Ще одним важливим аспектом контролю середовища є перевірка мережевого з’єднання, через яке застосунок потенційно може передавати або отримувати дані. У межах концепції Zero Trust будь–яке мережеве підключення вважається ненадійним, доки не буде доведено зворотне. У застосунку це реалізовано через клас NetworkCheckerService, що використовує бібліотеки connectivity_plus та network_info_plus.

На першому етапі виконується перевірка типу підключення:

```
final connectivityResult = await Connectivity().checkConnectivity();

if (connectivityResult == ConnectivityResult.none) {
  return 'Немає з’єднання з Інтернетом';
}
```

У разі відсутності мережі застосунок виводить повідомлення про недоступність сервісу та блокує подальші дії. Якщо пристрій підключено до Wi–Fi, система здійснює додаткову перевірку на потенційну небезпеку:

```
final info = NetworkInfo();
final ssid = await info.getWifiName();
final bssid = await info.getWifiBSSID();
final ip = await info.getWifiIP();
```

Якщо значення SSID, BSSID або IP є null або '<unknown ssid>', система визначає з’єднання як ненадійне:

```
if (ssid == null || ssid == '<unknown ssid>' || bssid == null || ip == null)
{
  return 'Підключено до потенційно небезпечної Wi-Fi мережі';
}
```

Це дозволяє ідентифікувати публічні або неавтентифіковані мережі, в яких можливий перехват трафіку або атаки типу MITM (man–in–the–middle). У таких випадках застосунок:

- відмовляється від спроби синхронізації;
- не завантажує дані з Firestore;
- повністю блокувати доступ до застосунку.

Такий механізм перевірки забезпечує контекстно–залежне прийняття рішення (context–aware access), що є основою сучасної Zero Trust моделі. Рішення

ухвалюється локально – без участі сервера – що відповідає принципу мінімізації довіри до будь–яких зовнішніх компонентів.

### 3.5 Створення, зберігання, доступ і керування даними

Процес роботи з зашифрованими даними у застосунку реалізовано через Firebase Firestore, де зберігаються лише зашифровані об’єкти. Усі операції над ними – створення, перегляд, редагування чи видалення – виконуються після перевірки контексту: наявності ключа та, у випадку критичних дій, повторного підтвердження пароля.

Нова нотатка створюється після введення тексту на екрані AddNoteScreen. Перед збереженням виконується шифрування:

```
final encrypted = await noteCryptoService.encryptNote(
  noteText,
  encryptionKeyProvider.key!,
);
```

Об’єкт зберігається у підколекції users/{uid}/notes:

```
await FirebaseFirestore.instance
  .collection('users')
  .doc(userId)
  .collection('notes')
  .add(encryptedNote.toMap());
```

Після входу в систему та переходу на головний екран (HomeScreen) здійснюється завантаження списку нотаток. Далі кожен заголовок розшифровується окремо:

```
final title = await _encryptionService.decrypt(
  cipherText: note.cipherTitle,
  nonce: note.nonceTitle,
  mac: note.macTitle,
  key: key,
);
```

Ключ береться з EncryptionKeyProvider, і якщо його не існує (наприклад, у разі втрати сесії), доступ до жодного елементу не буде надано. У такому випадку користувач отримає повідомлення «Ключ не знайдено», що запобігає роботі без авторизації.

Цей підхід забезпечує ізоляцію даних у хмарі: усі зашифровані елементи не мають значення без ключа, який існує тільки локально в додатку і формується на основі email + пароль + сіль.

Операції видалення нотаток вимагають окремої перевірки користувача через повторне введення пароля. Це реалізує принцип явна перевірка кожного запиту на доступ (explicit verification of every access request), рекомендований у Zero Trust Architecture. Видалення виконується лише після повторної автентифікації через Firebase.

Після натискання кнопки видалення нотатки на головному екрані викликається метод _confirmAndDeleteNote():

```
final confirmed = await showDialog<bool>(
  context: context,
  builder: (_) => ConfirmPasswordDialog(
    onConfirm: (password) async {
      final user = FirebaseAuth.instance.currentUser;
      final credential = EmailAuthProvider.credential(
        email: user!.email!,
        password: password,
      );
```

У цьому фрагменті відкривається модальне вікно ConfirmPasswordDialog, яке вимагає повторного введення пароля. Введений пароль передається у зворотний виклик onConfirm. Потім відбувається отримання поточного користувача з Firebase та побудова об’єкта AuthCredential для повторної автентифікації.

```
      await user.reauthenticateWithCredential(credential);
      await FirebaseFirestore.instance
        .collection('users')
        .doc(user.uid)
        .collection('notes')
        .doc(note.id)
        .delete();
    },
  ),
);
```

Після успішного введення пароля виконується повторна автентифікація за допомогою методу reauthenticateWithCredential(). Лише після цього система виконує операцію видалення нотатки з підколекції notes. Якщо автентифікація не проходить, виняток перехоплюється та дія скасовується. Цей підхід дозволяє

надійно захистити дані від несанкціонованого втручання, навіть у межах активної сесії.

Крім видалення окремої нотатки, користувач може ініціювати масове видалення всіх своїх записів. Ця операція також потребує обов'язкової повторної перевірки через ConfirmPasswordDialog, щоб запобігти випадковому або несанкціонованому знищенню даних.

```
final notesRef = FirebaseFirestore.instance
    .collection('users')
    .doc(user.uid)
    .collection('notes');
final notes = await notesRef.get();
for (final doc in notes.docs) {
  await doc.reference.delete();
```

Після проходження повторної автентифікації система завантажує всі нотатки користувача з Firestore та видаляє їх у циклі. У разі скасування або помилки жодна з нотаток не буде втрачена.

Таким чином, реалізація створення, зберігання, доступу та керування даними в мобільному застосунку не тільки забезпечує криптографічну безпеку, а й дотримується ключових принципів архітектури з нульовим рівнем довіри на кожному кроці взаємодії.

### 3.6 Інтерфейс користувача в умовах загроз

У мобільному застосунку реалізовано низку захисних механізмів на рівні інтерфейсу користувача, які дозволяють ефективно реагувати на потенційно небезпечні дії. Такий підхід ґрунтується на принципі явна перевірка кожного запиту на доступ (explicit verification of every access request), що є ключовим для Zero Trust Architecture. Особливу увагу приділено моментам, коли користувач ініціює зміну або видалення зашифрованих даних.

Операції видалення окремої або всіх нотаток вимагають введення пароля, навіть якщо користувач уже автентифікований у системі. Це забезпечує додатковий бар'єр безпеки, особливо у випадках, коли пристрій тимчасово залишено без нагляду.

На рисунку 3.5 зображено логіку взаємодії інтерфейсу з користувачем під час ініціювання дій. У разі виявлення критичної дії система запускає діалог перевірки пароля. Доступ дозволяється лише після підтвердження особи.

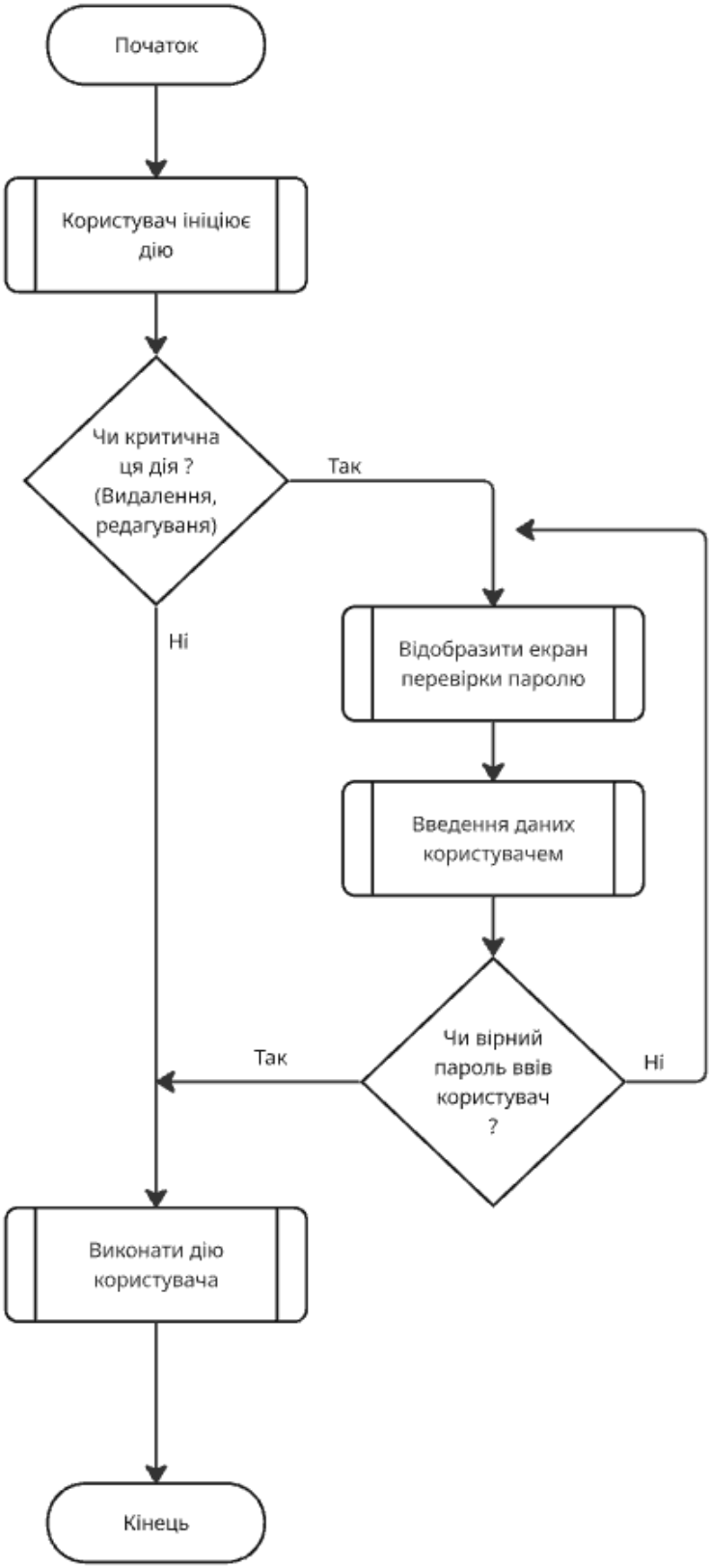



Рисунок 3.5 – Перевірка пароля перед виконанням критичної дії

Такий рівень контролю над діями користувача є особливо важливим для мобільного середовища, де пристрій може опинитися в чужих руках у розблокованому стані. В умовах Zero Trust навіть сам факт активної сесії не вважається достатнім доказом авторизованого доступу до критичних функцій. Тому будь–яке редагування або видалення даних потребує повторного підтвердження особи через введення чинного пароля. Це мінімізує ризик несанкціонованої модифікації інформації та запобігає зловживанню у випадках, коли автентифікаційна сесія ще не завершена, але довіра до пристрою вже не може бути гарантованою.

Нижче наведено ключовий фрагмент діалогу підтвердження пароля:

```
AlertDialog(
  title: const Text('Підтвердження паролем'),
  content: Column(
    mainAxisSize: MainAxisSize.min,
    children: [
      TextField(
        controller: _controller,
        obscureText: true,
        decoration: const InputDecoration(labelText: 'Пароль'),
      ),
```

Цей фрагмент формує діалогове вікно з заголовком та полем введення пароля. Використовується TextField з obscureText: true, що приховує введення, забезпечуючи конфіденційність. Контролер _controller дає змогу зчитати значення введеного пароля, яке потім буде використано для повторної автентифікації користувача.

```
      if (_error != null)
        Padding(
          padding: const EdgeInsets.only(top: 8.0),
          child: Text(
            _error!,
            style: const TextStyle(color: Colors.red),
          ),
        ),
```

Якщо пароль введено неправильно, в змінну _error записується текст помилки. Далі на екрані відображається повідомлення червоного кольору. Це дозволяє користувачу зрозуміти, що введене значення неправильне, і не дає завершити критичну дію. Таким чином реалізовано простий, але ефективний захист на рівні інтерфейсу.

У застосунку використовується симетричне шифрування AES–GCM, де ключ формується динамічно на основі логіна, пароля та солі. Цей ключ існує лише в оперативній пам’яті під час сесії. Якщо з будь–якої причини ключ відсутній (наприклад, після перезапуску застосунку без повторного логіну), інтерфейс не дозволяє продовжити роботу. Замість порожнього списку або помилок, користувач бачить чітке повідомлення про відсутність ключа:

```
body: key == null
  ? const Center(child: Text('Ключ не знайдено'))
  : FutureBuilder<List<EncryptedNote>>(
      future: _notesFuture,
      builder: (context, snapshot) {
        ...
      },
    ),
```

Це рішення унеможливлює доступ до зашифрованих даних без наявності ключа, навіть якщо Firestore вже повернув дані. Таким чином, інтерфейс підтримує модель «відсутності довіри» до поточного стану сесії, якщо не виконано повну перевірку доступу.

Щоб запобігти атакам перебору паролів (brute force), застосунок відстежує кількість помилок автентифікації. Після п’ятої помилки відбувається тимчасове блокування входу – користувач бачить відповідне повідомлення з таймером.

```
if (_failedAttempts >= 5) {
  _blockedUntil = DateTime.now().add(const Duration(seconds: 30));
  _showMessage('Забагато спроб. Блокування на 30 секунд.');
}
```

У цей період усі нові спроби входу будуть відхилятися. Користувач побачить повідомлення про те, скільки часу залишилось до розблокування:

```
if (_blockedUntil != null && DateTime.now().isBefore(_blockedUntil!)) {
  final remaining = _blockedUntil!.difference(DateTime.now()).inSeconds;
  _showMessage('Аккаунт тимчасово заблоковано. Спробуйте через $remaining
сек.');
  return;
}
```

Такий інтерфейс не тільки підвищує рівень безпеки, але й забезпечує прозорість для користувача: замість раптового відключення або мовчазної відмови система чітко пояснює, що відбувається і чому.

### 3.7 Тестування безпечності бібліотек

Мобільні застосунки, що працюють із чутливими даними, використовують численні сторонні бібліотеки, кожна з яких потенційно може становити ризик. Відповідно до принципів Zero Trust, не варто довіряти навіть бібліотекам за замовчуванням – їх необхідно перевіряти на предмет підтримки, актуальності, наявності відомих вразливостей та безпечних налаштувань за замовчуванням.

У межах роботи було виконано перевірку безпечності бібліотек, що використовуються у проєкті, за такими критеріями:

- відкритість та підтримка (GitHub / офіційні джерела);
- частота оновлень;
- наявність критичних CVE (Common Vulnerabilities and Exposures);
- відповідність найкращим практикам безпечної розробки.

У проєкті було використано низку Flutter–бібліотек, які виконують критичні функції, пов’язані із зберіганням, шифруванням та передачею даних. Серед них:

- cryptography – бібліотека для реалізації симетричного шифрування AES–GCM та алгоритму PBKDF2;
- flutter_secure_storage – сховище для зберігання ключа шифрування в оперативній пам’яті;
- firebase_auth – автентифікація користувачів з підтвердженням через email;
- cloud_firestore – хмарна база даних для зберігання зашифрованих нотаток;
- network_info_plus, connectivity_plus – перевірка середовища виконання, мережевого підключення та виявлення ознак емулятора чи рутованого пристрою;
- provider – передача та оновлення симетричного ключа в межах сесії.

Усі ці бібліотеки мають прямий або опосередкований вплив на захист персональних даних. Тому їх було перевірено за допомогою спеціалізованих засобів, зокрема Aikido Security.

Цей інструмент дозволяє своєчасно виявляти критичні вразливості у сторонніх залежностях, а також надає рекомендації щодо оновлення або заміни

небезпечних компонентів. Завдяки інтеграції з системою контролю версій забезпечується постійний аудит безпеки, що особливо важливо для захисту персональних даних у мобільних застосунках.

Для виконання автоматизованої перевірки залежностей проєкт було попередньо розміщено у приватному репозиторії на платформі GitHub. Після цього було підключено репозиторій до сервісу Aikido Security, який підтримує аналіз Flutter–застосунків і виконує моніторинг відомих вразливостей, ліцензійних обмежень, витоків конфіденційних даних та інших факторів ризику.

На рисунку 3.6 відображено результат сканування.

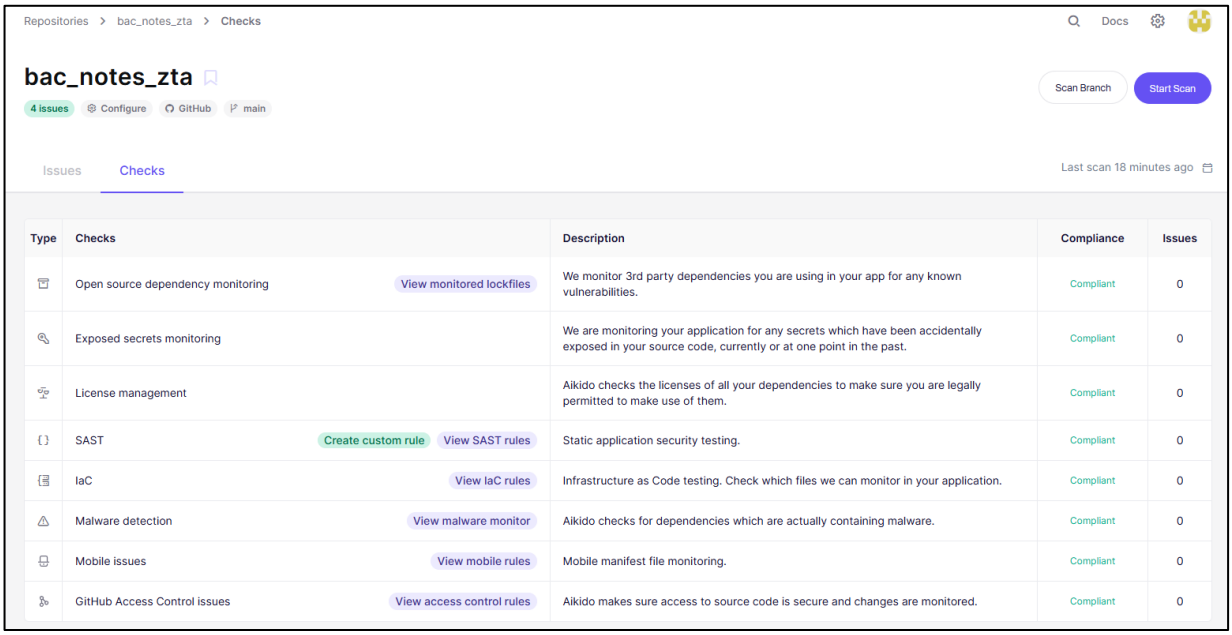



Рисунок 3.6 – Загальний звіт перевірки проєкту в Aikido Security

Після завершення сканування було отримано зведений звіт, який засвідчив, що жодних критичних або високоризикованих вразливостей у проєкті не виявлено. Всі перевірки отримали статус Compliant, що свідчить про відповідність використовуваних бібліотек найкращим практикам безпечної розробки.

На основі результатів автоматичного аналізу було проведено додатковий ручний аудит, під час якого ретельно перевірялися всі виявлені залежності. Особливу увагу приділено бібліотекам, що взаємодіють із персональними даними користувачів або мають доступ до мережевих ресурсів. Це дозволило підтвердити відсутність прихованих або недостатньо задокументованих вразливостей, а також

упевнитися у надійності механізмів шифрування, автентифікації та збереження даних, що використовуються у застосунку.

Окремо було переглянуто список усіх бібліотек, що були виявлені у файлі pubspec.lock. Для кожної з них вказано версію, відповідність стандартам безпеки та відсутність відомих вразливостей. Зокрема, бібліотеки cryptography, firebase_auth, cloud_firestore, flutter_secure_storage були перевірені й підтверджені як безпечні до використання на час перевірки.

На рисунку 3.7 зазначено результат перевірки окремо по бібліотекам.

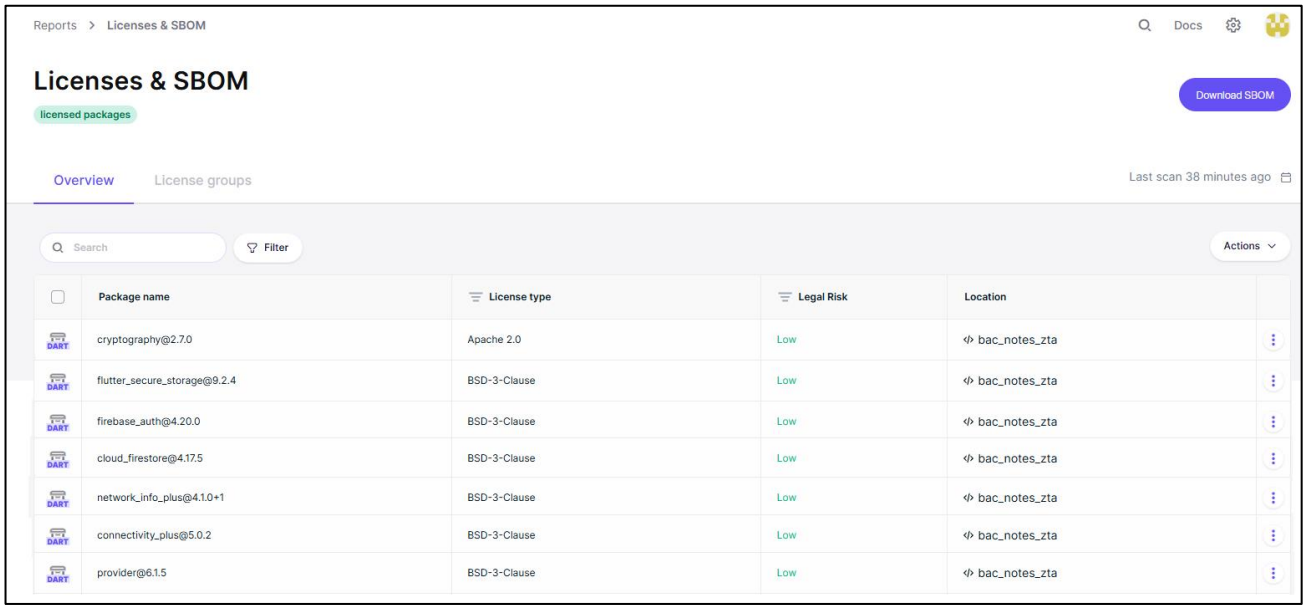



Рисунок 3.7 – Стан безпечності бібліотек згідно зі скануванням Aikido

На рисунку подано перелік бібліотек, які були сканування. Для кожної залежності вказано тип ліцензії, ризик (усі – Low) та локалізацію у проєкті.

Отримані результати підтверджують, що використані бібліотеки є актуальними, активно підтримуються спільнотою та не містять відомих критичних вразливостей. Аналіз із використанням сервісу Aikido Security засвідчив відповідність залежностей сучасним вимогам до безпечної розробки, що є ключовим у контексті реалізації принципів Zero Trust Architecture. Таким чином, зовнішні компоненти програмного забезпечення не становлять додаткового ризику для захисту персональних даних у межах реалізованого застосунку.

### 3.8 Тестування додатку

З метою підтвердження працездатності реалізованого функціоналу та відповідності застосунку принципам Zero Trust Architecture було проведено серію тестувань. Перевірці підлягали як основні функції застосунку (реєстрація, автентифікація, робота з нотатками), так і поведінка системи у випадку потенційних загроз: критичних дій користувача, небезпечного середовища, несанкціонованого доступу до даних.

На першому етапі тестування перевірялась коректна робота функцій автентифікації. Реєстрація нового користувача відбувається із введенням електронної адреси та пароля, після чого на email надсилається лист для підтвердження. Без підтвердження облікового запису користувач не зможе увійти в систему, що відповідає принципу "явна перевірка кожного запиту".

На рисунку 3.8 відображено екран реєстрації та авторизації користувача.

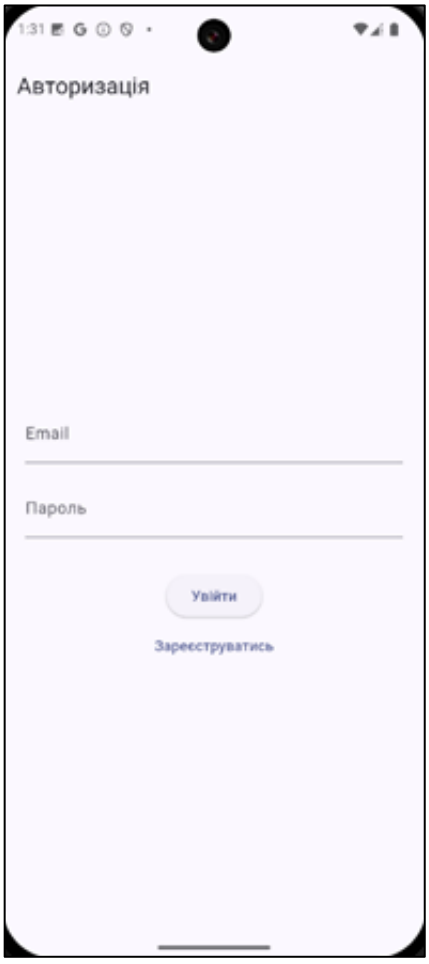
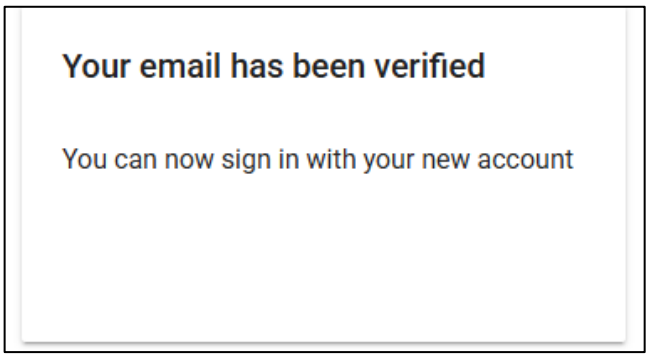



Рисунок 3.8 – Екран реєстрації та авторизації користувача

Після введення користувачем логіну та паролю, та натисканню кнопки «Зареєструватись», відбувається реєстрація користувача, та надсиланню користувачу повідомлення на пошту з посиланням для підтвердження електронної пошти. На рисунку 3.9 відображено повідомлення з посиланням для підтвердження електронної пошти.


Рисунок 3.9 – Лист з посиланням для підтвердження електронної пошти

Для підтвердження електронної пошти необхідно перейти за посиланням, що було вказано в листі, після переходження буде відображено повідомленя, з інформацією про те, що електронну пошту було успішно підтверджено, на рисунку 3.10 відображено вигляд цього повідомлення.




Рисуок 3.10 – Вигляд повідомлення про успішне підтвердження електронної пошти

Після підтвердження електронної пошти користувач може здійснити вхід у систему. У разі введення правильних облікових даних відбувається ініціалізація ключа шифрування та відкривається головний екран, що відображено на рисунку 3.11.

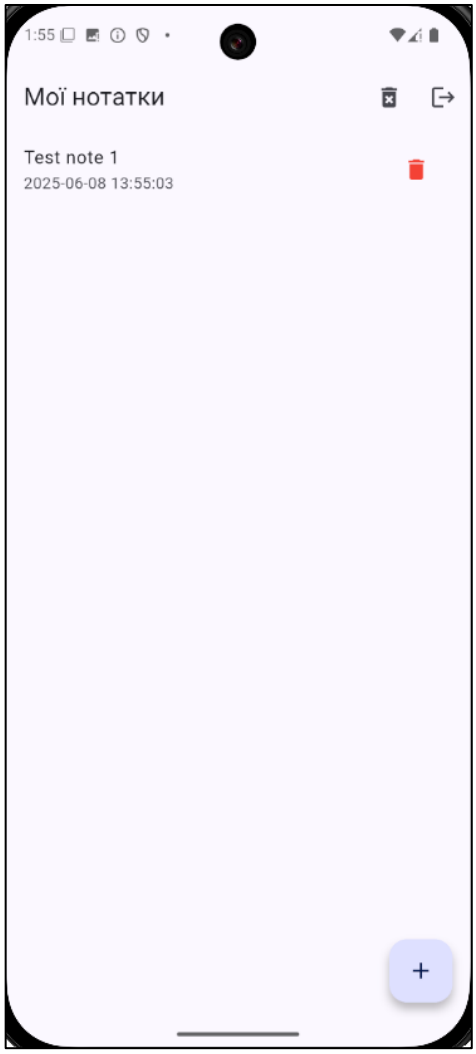



Рисунок 3.11 – Вигляд головного екрану

На головному екрані відображається список нотаток користувача, якщо вони містяться. Кожна нотатка зберігається в зашифрованому вигляді у Firestore та дешифрується на боці клієнта в момент відображення. На головному екрані відображаються лише Теми нотаток, для відображення повного змісту нотатки, необхідно перейти до самої нотатки.

У межах тестування було перевірено реакцію інтерфейсу у ситуаціях, що моделюють потенційні загрози – зокрема, неправильні дії користувача або порушення цілісності сесії.

Одним із прикладів є ситуація, коли користувач намагається видалити нотатку. У такому випадку, навіть якщо автентифікація вже виконана, система вимагає повторного введення пароля. Це дозволяє запобігти несанкціонованим діям, наприклад, у випадку, коли пристрій залишено у розблокованому стані.

На рисунку 3.12 відображено запит на повторне введення паролю.

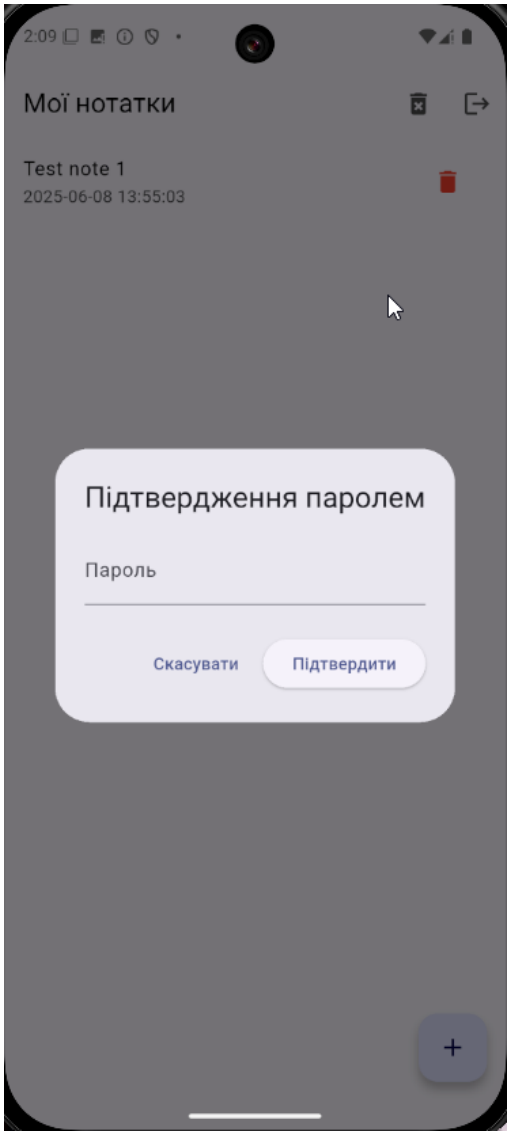



Рисунок 3.12 – Повторна перевірка паролем

Якщо введено неправильний пароль, користувач бачить попередження про помилку, а сама дія не виконується. Це реалізує принцип “fail securely” – дія зупиняється без розкриття деталей. У разі правильного введення пароля нотатка видаляється з бази даних і зникає зі списку.

Аналогічний механізм додаткової перевірки реалізовано і для операцій редагування нотатки. Перед тим як дозволити змінити вміст зашифрованих даних, інтерфейс виводить діалогове вікно із запитом повторного введення пароля. Це дозволяє переконатись, що критичну зміну ініціює саме власник облікового запису, а не сторонній користувач, який отримав фізичний доступ до пристрою. Такий

підхід повністю відповідає принципам нульового рівня довіри та забезпечує контроль доступу навіть до дій, які не пов’язані з передачею даних за межі системи.

Застосунок реалізує також перевірку середовища виконання, яка активується під час запуску. Основна мета – виявлення ознак небезпечного середовища, таких як запуск на емуляторі, наявність root–доступу або підключення до невідомої Wi–Fi мережі.

Для Android перевірка виконується через device_info_plus і Platform.isAndroid, де аналізується тип пристрою, виробник, модель і тег прошивки. Якщо модель пристрою містить ознаки generic або google_sdk, або тег має значення test–keys, система розпізнає середовище як потенційно скомпрометоване та блокує доступ до додатку, виводячи повідомлення про те, що доступ було заборонено як відобрежно на рисунку 3.13.


Рисунок 3.13 – Екран заборони доступу до додатку та повідомлення про скомпрометоване середовище виконання

Ці перевірки дозволяють знизити ризики, пов’язані з можливістю перехоплення або модифікації даних, а також несанкціонованим доступом до функціоналу застосунку в неконтрольованому середовищі. Визначення таких умов виконується локально на пристрої до встановлення з’єднання з серверною частиною, що дає змогу запобігти потенційним атакам ще до початку обробки чутливої інформації.

Аналогічно, перевірка мережі дозволяє виявити ситуації, коли пристрій підключено до невідомої або підозрілої Wi–Fi мережі. Для цього використовуються пакети connectivity_plus та network_info_plus, що дозволяють перевірити SSID, BSSID та IP–адресу (рис. 3.14).


Рисунок 3.14 – Екран заборони доступу до додатку та попередження про ризиковане мережеве підключення

Таким чином, реалізовані механізми автентифікації, контролю дій користувача та перевірки середовища виконання підтвердили свою ефективність у межах проведеного тестування.

# ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було успішно реалізовано захищений мобільний застосунок для зберігання нотаток, що базується на принципах Zero Trust Architecture. Застосунок дозволив реалізувати основну мету дослідження – підвищення рівня безпеки персональних даних у мобільних середовищах за рахунок перевірених засобів автентифікації, контролю середовища, криптографічного захисту та обмеження доступу до зашифрованих даних.

У першому розділі було здійснено аналітичний огляд сучасних підходів до захисту мобільних застосунків. Проаналізовано основні архітектурні принципи, пов’язані з Zero Trust, та нормативні документи, зокрема NIST SP 800–207. Обґрунтовано доцільність реалізації постійної перевірки запитів доступу, мінімізації привілеїв, аналізу контексту та перевірки середовища виконання.

У другому розділі виконано проєктування архітектури мобільного застосунку. Обрано Firebase для реалізації автентифікації та зберігання даних, а для шифрування – алгоритм AES–GCM із динамічним формуванням ключа на основі логіна, пароля та солі через PBKDF2. Передбачено захист критичних операцій через повторне введення пароля, що унеможливлює несанкціонований доступ навіть у межах авторизованої сесії.

У третьому розділі описано процес програмної реалізації, що включає автентифікацію з підтвердженням електронної пошти, шифрування нотаток, захист середовища виконання, обробку помилок і виведення відповідних повідомлень користувачу. Проведено тестування безпечності використаних бібліотек за допомогою інструменту Aikido, результати якого підтвердили відсутність критичних вразливостей. Завдяки сценаріям тестування продемонстровано стабільність та відповідність застосунку принципам Zero Trust, зокрема через обмеження доступу до зашифрованих даних у разі втрати ключа, контроль повторної автентифікації для видалення та редагування записів, а також блокування після багаторазових помилок входу.

Таким чином, усі поставлені завдання було вирішено. Розроблений мобільний застосунок демонструє ефективну інтеграцію сучасних механізмів захисту інформації в умовах мобільного середовища та підтверджує практичну доцільність впровадження моделі Zero Trust Architecture у сфері захисту персональних даних.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. П’ять основних загроз безпеки, з якими стикаються користувачі смартфонів. URL: https://bitdefender.ua/blog/pyat-osnovnikh-zagroz-bezpeki-z-yakimi-stikayutsya-koristuvachi-smartfoniv/ (дата звернення: 24.03.2025).
2. M9: Insecure Data Storage. OWASP Foundation. URL: https://owasp.org/www-project-mobile-top-10/2023-risks/m9-insecure-data-storage (accessed: 25.03.2025).
3. Enforce HTTPS with TLS (SSL/TLS Security). Man-in-the-Middle Attacks in Android Development: How to secure your app. URL: https://proandroiddev.com/man-in-the-middle-mitm-attacks-in-android-development-how-to-secure-your-app-062d1fdb3a7e (accessed: 27.03.2025).
4. What is API7:2019 Security Misconfiguration? Insecure API Endpoint /api/user/:id. URL: https://www.indusface.com/blog/owasp-api-2019-security-misconfiguration/ (accessed: 27.03.2025).
5. What is social engineering uses psychological manipulation to trick users into making security mistakes. URL: https://www.imperva.com/learn/application-security/social-engineering-attack/ (accessed: 28.03.2025)
6. OWASP Mobile Top 10 – 2024. URL: https://owasp.org/www-project-mobile-top-10/ (accessed: 28.03.2025).
7. Perimeter Security vs Zero Trust: It's time to make the move. URL: https://www.techtarget.com/searchsecurity/tip/Perimeter-security-vs-zero-trust-Its-time-to-make-the-move (accessed: 29.03.2025).
8. Android application sandbox and secure data storage mechanisms. URL: https://developer.android.com/privacy-and-security/security-tips (accessed: 29.03.2025).
9. Secure Your Android App’s Data with EncryptedSharedPreferences. URL: https://medium.com/@moraneus/secure-your-android-apps-data-with-encryptedsharedpreferences-7091da539f5e (accessed: 02.04.2025).

10. TLS протокол — що це таке та як він захищає ваші дані в Інтернеті. URL: https://cityhost.ua/uk/blog/scho-take-protokol-tls-yak-vin-pracyu-ta-vid-chogo-zahischa.html (дата звернення: 05.04.2025).
11. Що таке двофакторна автентифікація, і як вона працює? URL: https://cip.gov.ua/ua/faqs/sho-take-dvofaktorna-avtentifikaciya-i-yak-vona-pracyuye (дата звернення: 08.04.2025)
12. Zero Trust Security vs Traditional Perimeter-Based Models. URL: https://blog.scalefusion.com/zero-trust-vs-traditional-security/ (accessed: 09.04.2025).
13. Криючкова Л. С., Ворохоб М. Г. Сучасні перспективи застосування концепції Zero Trust при формуванні політики інформаційної безпеки підприємства // Кібербезпека: освіта, наука, техніка. 2023. № 3(19). С. 227–235. URL: https://csecurity.kubg.edu.ua/index.php/journal/article/view/508 (дата звернення: 10.04.2025).
14. Securing the Perimeterless: Dive Deep into Zero Trust Architecture with Continuous Authentication. URL: https://www.authgear.com/post/securing-the-perimeterless-dive-deep-into-zero-trust-architecture-with-continuous-authentication (accessed: 11.04.2025).
15. National Institute of Standards and Technology. Zero Trust Architecture : NIST SP 800–207 / NIST. — Gaithersburg, MD. — 2020. — 52 с. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf (accessed: 11.04.2025).
16. Zero Trust Architecture – SentinelOne. URL: https://www.sentinelone.com/cybersecurity-101/identity-security/zero-trust-architecture/ (accessed: 11.04.2025).
17. Що таке архітектура нульової довіри? — Українською. URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-zero-trust-architecture (дата звернення: 15.04.2025).

18. Що таке модель нульової довіри Zero Trust. URL: https://gigatrans.ua/ua/news/chto-takoe-model-nulevogo-doveriya-zero-trust-i-zachem-ona-nuzhna (дата звернення: 17.04.2025).
19. A guide to Zero Trust for your mobile apps. URL: https://promon.io/resources/knowledge-center/zero-trust-mobile-app-security (accessed: 18.04.2025).
20. Perimeter Security vs Zero Trust: Paving the Way for Cybersecurity Transformation. URL: https://www.tufin.com/blog/perimeter-security-vs-zero-trust-cybersecurity-transformation (accessed: 20.04.2025).
21. Zero Trust: The new frontier in Mobile app security. URL: https://build38.com/blog/technology/zero-trust-the-new-frontier-in-mobile-app-security (accessed: 22.04.2025).
22. A Secure Mobile Apps with Zero Trust Architecture: A Complete Guide. URL: https://www.hakunamatatatech.com/our-resources/blog/zero-trust-security-model (accessed: 25.04.2025).
23. How to Implement Zero Trust for Mobile Apps. URL: https://approov.io/knowledge/the-role-of-approov-in-zero-trust (accessed: 28.04.2025).
24. Add multi-factor authentication to your web app. URL: https://firebase.google.com/docs/auth/web/multi-factor (accessed: 29.04.2025).
25. Understanding AES Encryption and AES-GCM Mode: An In-depth Exploration Using. URL: https://medium.com/@pravallikayakkala123/understanding-aes-encryption-and-aes-gcm-mode-an-in-depth-exploration-using-java-e03be85a3faa (accessed: 02.05.2025).
26. Android Developers. FLAG_SECURE. URL: https://developer.android.com/reference/android/view/WindowManager.LayoutParams#FLAG_SECURE (accessed: 02.05.2025).
27. Open Source Vulnerability Scanner for GitHub, GitLab and Bitbucket. URL: https://www.aikido.dev/use-cases/vulnerability-management (accessed: 05.05.2025).

28. How to implement end-to-end encryption using PBKDF in Flutter. URL: https://medium.com/@rllmuk/how-to-implement-end-to-end-encryption-using-pbkdf-in-flutter-efc79a946e82 (accessed: 06.05.2025).
29. Consistent Data Encryption in Android, iOS, and Flutter Apps with AES. URL: https://dev.to/canopassoftware/consistent-data-encryption-in-android-ios-and-flutter-apps-with-aes-54m6 (accessed: 10.05.2025).
30. Гарнага В. А. Використання підходів Zero Trust Architecture при розробці мобільних додатків. Матеріали конференції *Молодь в науці: дослідження, проблеми, перспективи ВНТУ*. Вінниця: ВНТУ, 2025. 4 с. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25380 (дата звернення: 21.03.2025).
31. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 125 «Кібербезпека» освітньо-професійні програми «Безпека інформаційних і комунікаційних систем» та «Кібербезпека критичних систем» / Уклад. В. А. Каплун, О. П. Войтович. Вінниця: ВНТУ, 2023. 61 с.

# ДОДАТКИ

# Додаток А

## Код програми

### Main.dart

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'providers/encryption_key_provider.dart';
import 'screens/auth_screen.dart';
import 'services/app_lifecycle_observer.dart';
import 'services/network_checker_service.dart';
import 'services/root_checker_service.dart';

void main() async {
WidgetsFlutterBinding.ensureInitialized();
await Firebase.initializeApp();

final networkStatus = await NetworkCheckerService.checkNetworkStatus();
debugPrint(' Network Status: $networkStatus');
final isCompromised = await RootCheckerService.isDeviceCompromised();

runApp(
MultiProvider(
providers: [
ChangeNotifierProvider(create: (_) => EncryptionKeyProvider()),
].,
child: MyApp(
isCompromised: isCompromised, // або виклик root checker
networkMessage: networkStatus,
),
),
);
}

class MyApp extends StatelessWidget {
final bool isCompromised;
final String networkMessage;

const MyApp({
super.key,
required this.isCompromised,
required this.networkMessage,
});

@override
Widget build(BuildContext context) {
return MaterialApp(
debugShowCheckedModeBanner: false,
title: 'Secure Notes',
theme: ThemeData(
useMaterial3: true,
colorScheme: ColorScheme.fromSeed(seedColor: Colors.indigo),
),
builder: (context, child) {
return AppLifecycleObserver(
child: Builder(
builder: (innerContext) {
// Блокування доступу якщо немає інтернету
if (networkMessage == 'Немає з’єднання з Інтернетом') {
return const NoInternetScreen();
```

```
}

// Якщо все ок — рендеримо екран додатку
return child!;
},
),
);
},
home: isCompromised ? const RootBlockedScreen() : const AuthScreen(),
);
}
}

/// Екран, який з’являється якщо немає інтернету
class NoInternetScreen extends StatelessWidget {
const NoInternetScreen({super.key});

@override
Widget build(BuildContext context) {
return const Scaffold(
body: Center(
child: Padding(
padding: EdgeInsets.all(24.0),
child: Text(
'Немає з’єднання з Інтернетом.\nПеревірте підключення та спробуйте ще раз.',
textAlign: TextAlign.center,
style: TextStyle(fontSize: 18),
),
),
),
);
}
}

class RootBlockedScreen extends StatelessWidget {
const RootBlockedScreen({super.key});

@override
Widget build(BuildContext context) {
return const Scaffold(
body: Center(
child: Padding(
padding: EdgeInsets.all(24.0),
child: Text(
'Доступ заборонено.\n Оскільки було виявлено небезпечну мережу',
textAlign: TextAlign.center,
style: TextStyle(fontSize: 18),
),
),
),
);
}
}
```

## auth_screen.dart

```
import 'package:bac_notes_zta/screens/home_screen.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import '../providers/encryption_key_provider.dart';
import '../services/user_data_service.dart';
```

```
class AuthScreen extends StatefulWidget {
  const AuthScreen({super.key});
  @override
  State<AuthScreen> createState() => _AuthScreenState();
}

class _AuthScreenState extends State<AuthScreen> {
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  final _userDataService = UserDataService();
  late final EncryptionKeyProvider _keyProvider;
  int _failedAttempts = 0;
  DateTime? _blockedUntil;

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    _keyProvider = Provider.of<EncryptionKeyProvider>(context, listen: false);
  }

  void _showMessage(String message) {
    if (!mounted) return;
    ScaffoldMessenger.of(
      context,
    ).showSnackBar(SnackBar(content: Text(message)));
  }

  Future<void> _handleLogin() async {
    final email = _emailController.text.trim();
    final password = _passwordController.text;

    if (email.isEmpty || password.isEmpty) {
      _showMessage('Заповніть поля email та пароль');
      return;
    }

    if (_blockedUntil != null && DateTime.now().isBefore(_blockedUntil!)) {
      final remaining = _blockedUntil!.difference(DateTime.now()).inSeconds;
      _showMessage(
        'Аккаунт тимчасово заблоковано. Спробуйте через $remaining сек.',
      );
      return;
    }

    try {
      final userCredential = await FirebaseAuth.instance
          .signInWithEmailAndPassword(email: email, password: password);

      final user = userCredential.user;

      if (user == null) {
        _showMessage('Помилка: користувача не знайдено.');
        return;
      }

      if (!user.emailVerified) {
        await FirebaseAuth.instance.signOut();
        _showMessage('Підтвердіть email перед входом.');
        return;
      }
      if (!user.emailVerified) {
        await FirebaseAuth.instance.signOut();
        _showMessage('Підтвердіть email перед входом.');
        return;
```

```
    }

    // ↓ Додаємо: витягуємо сіль, виводимо ключ, зберігаємо в Provider
    final uid = user.uid;
    final salt = await _userDataService.fetchSaltForUser(uid);
    final derivedKey = await _userDataService.encryptionService
        .deriveKeyFromCredentials(
          login: email,
          password: password,
          salt: salt,
        );
    _keyProvider.setKey(derivedKey);

    //  Перехід до домашнього екрану
    if (!mounted) return;
    Navigator.of(
      context,
    ).pushReplacement(MaterialPageRoute(builder: (_) => const HomeScreen()));

    //  Email підтверджено — вхід успішний
    if (!mounted) return;
    Navigator.of(
      context,
    ).pushReplacement(MaterialPageRoute(builder: (_) => const HomeScreen()));
  } on FirebaseAuthException catch (e) {
    if (e.code == 'user-not-found') {
      _showMessage('Акаунту не існує. Будь ласка, зареєструйтесь.');
    } else if (e.code == 'invalid-credential') {
      _failedAttempts++;
      if (_failedAttempts >= 5) {
        _blockedUntil = DateTime.now().add(const Duration(seconds: 30));
        _showMessage('Забагато спроб. Блокування на 30 секунд.');
      } else {
        _showMessage(
          'Невірний логін або пароль. Спроба $_failedAttempts з 5.',
        );
      }
    } else {
      _showMessage('Помилка: ${e.message}');
    }
  }
}

Future<void> _handleRegister() async {
  final email = _emailController.text.trim();
  final password = _passwordController.text;

  if (email.isEmpty || password.isEmpty) {
    _showMessage('Заповніть поля емаіл та пароль');
    return;
  }

  if (_blockedUntil != null && DateTime.now().isBefore(_blockedUntil!)) {
    final remaining = _blockedUntil!.difference(DateTime.now()).inSeconds;
    _showMessage(
      'Аккаунт тимчасово заблоковано. Спробуйте через $remaining сек.',
    );
    return;
  }

  try {
    final userCredential = await FirebaseAuth.instance
        .createUserWithEmailAndPassword(email: email, password: password);
```

```
      final uid = userCredential.user!.uid;

      // Зберегти сіль користувача
      await _userDataService.generateAndStoreSaltForUser(
        email: email,
        password: password,
        uid: uid,
      );

      await userCredential.user?.sendEmailVerification();
      _showMessage('Реєстрація успішна. Підтвердіть email.');
    } on FirebaseAuthException catch (e) {
      _showMessage('Помилка: ${e.message}');
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Авторизація')),
      body: Padding(
        padding: const EdgeInsets.all(24.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            TextField(
              controller: _emailController,
              decoration: const InputDecoration(labelText: 'Email'),
              keyboardType: TextInputType.emailAddress,
            ),
            const SizedBox(height: 16),
            TextField(
              controller: _passwordController,
              decoration: const InputDecoration(labelText: 'Пароль'),
              obscureText: true,
            ),
            const SizedBox(height: 32),
            ElevatedButton(
              onPressed: _handleLogin,
              child: const Text('Увійти'),
            ),
            TextButton(
              onPressed: _handleRegister,
              child: const Text('Зареєструватись'),
            ),
          ].,
        ),
      ),
    );
  }
}
```

## confirm_password_dialog.dart

```
import 'package:flutter/material.dart';

class ConfirmPasswordDialog extends StatefulWidget {
  final Function(String password) onConfirm;

  const ConfirmPasswordDialog({super.key, required this.onConfirm});

  @override
  State<ConfirmPasswordDialog> createState() => _ConfirmPasswordDialogState();
```

```
}

class _ConfirmPasswordDialogState extends State<ConfirmPasswordDialog> {
  final _controller = TextEditingController();
  bool _isLoading = false;
  String? _error;

  void _submit() async {
    final password = _controller.text.trim();
    if (password.isEmpty) return;

    setState(() {
      _isLoading = true;
      _error = null;
    });

    try {
      await widget.onConfirm(password);
      if (context.mounted) Navigator.of(context).pop(true);
    } catch (e) {
      setState(() {
        _error = ' Невірний пароль.';
        _isLoading = false;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return AlertDialog(
      title: const Text('Підтвердження паролем'),
      content: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          TextField(
            controller: _controller,
            obscureText: true,
            decoration: const InputDecoration(labelText: 'Пароль'),
          ),
          if (_error != null)
            Padding(
              padding: const EdgeInsets.only(top: 8.0),
              child: Text(_error!, style: const TextStyle(color: Colors.red)),
            ),
        ].,
      ),
      actions: [
        if (_isLoading)
          const Padding(
            padding: EdgeInsets.symmetric(horizontal: 16),
            child: CircularProgressIndicator(),
          )
        else ...[
          TextButton(
            onPressed: () => Navigator.of(context).pop(false),
            child: const Text('Скасувати'),
          ),
          ElevatedButton(
            onPressed: _submit,
            child: const Text('Підтвердити'),
          ),
        ].,
      ].,
    );
```

```
  }
}
```

## home_screen.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:cryptography/cryptography.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import '../../models/encrypted_note_model.dart';
import '../../providers/encryption_key_provider.dart';
import '../../screens/add_note_screen.dart';
import '../services/encryption_service.dart';
import 'confirm_password_dialog.dart';
import 'note_detail_screen.dart';

class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final _encryptionService = EncryptionService();
  Future<List<EncryptedNote>>? _notesFuture;

  @override
  void initState() {
    super.initState();
    _notesFuture = _fetchNotes();
  }

  Future<List<EncryptedNote>> _fetchNotes() async {
    final user = FirebaseAuth.instance.currentUser;
    final key = Provider.of<EncryptionKeyProvider>(context, listen: false).key;

    if (user == null || key == null) {
      return [].;
    }

    final snapshot = await FirebaseFirestore.instance
        .collection('users')
        .doc(user.uid)
        .collection('notes')
        .orderBy('createdAt', descending: true)
        .get();

    return snapshot.docs.map((doc) {
      return EncryptedNote.fromMap(doc.id, doc.data());
    }).toList();
  }

  Future<void> _confirmAndDeleteNote(EncryptedNote note) async {
    final confirmed = await showDialog<bool>(
      context: context,
      builder: (_) => ConfirmPasswordDialog(
        onConfirm: (password) async {
          final user = FirebaseAuth.instance.currentUser;
          final credential = EmailAuthProvider.credential(
            email: user!.email!,
            password: password,
```

```
        );
        await user.reauthenticateWithCredential(
          credential,
        ); // ❗ кидатиме помилку при неправильному паролі

        // Видалення з Firestore
        await FirebaseFirestore.instance
            .collection('users')
            .doc(user.uid)
            .collection('notes')
            .doc(note.id)
            .delete();
      },
    ),
  );

  if (confirmed == true) {
    setState(() {
      _notesFuture = _fetchNotes(); // оновити список
    });
  }
}

Future<void> _confirmAndDeleteAllNotes() async {
  final confirmed = await showDialog<bool>(
    context: context,
    builder: (_) => ConfirmPasswordDialog(
      onConfirm: (password) async {
        final user = FirebaseAuth.instance.currentUser;
        final credential = EmailAuthProvider.credential(
          email: user!.email!,
          password: password,
        );
        await user.reauthenticateWithCredential(credential);

        final notesRef = FirebaseFirestore.instance
            .collection('users')
            .doc(user.uid)
            .collection('notes');

        final notes = await notesRef.get();
        for (final doc in notes.docs) {
          await doc.reference.delete();
        }
      },
    ),
  );

  if (confirmed == true) {
    setState(() {
      _notesFuture = _fetchNotes(); // оновлюємо список
    });
  }
}

Future<String> _decryptTitle(EncryptedNote note, SecretKey key) async {
  try {
    return await _encryptionService.decrypt(
      cipherText: note.cipherTitle,
      nonce: note.nonceTitle,
      mac: note.macTitle,
      key: key,
    );
```

```
  } catch (_) {
    return '[Помилка дешифрування].';
  }
}

void _logout() async {
  await FirebaseAuth.instance.signOut();
  if (!mounted) return;
  Navigator.of(context).pushReplacementNamed('/');
}

@override
Widget build(BuildContext context) {
  final key = Provider.of<EncryptionKeyProvider>(context).key;

  return Scaffold(
    appBar: AppBar(
      title: const Text('Мої нотатки'),
      actions: [
        IconButton(
          icon: const Icon(Icons.delete_forever),
          tooltip: 'Видалити всі нотатки',
          onPressed: _confirmAndDeleteAllNotes,
        ),
        IconButton(
          icon: const Icon(Icons.logout),
          tooltip: 'Вийти',
          onPressed: _logout,
        ),
      ].,
    ),
    body: key == null
        ? const Center(child: Text('Ключ не знайдено'))
        : FutureBuilder<List<EncryptedNote>>(
            future: _notesFuture,
            builder: (context, snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return const Center(child: CircularProgressIndicator());
              }

              if (!snapshot.hasData || snapshot.data!.isEmpty) {
                return const Center(child: Text('Немає нотаток'));
              }
              final notes = snapshot.data!;
              return ListView.builder(
                itemCount: notes.length,
                itemBuilder: (context, index) {
                  final note = notes[index].;
                  return FutureBuilder<String>(
                    future: _decryptTitle(note, key),
                    builder: (context, titleSnapshot) {
                      final title = titleSnapshot.data ?? '[...].';
                      return ListTile(
                        title: Text(title),
                        subtitle: Text(
                          '${note.createdAt.toLocal()}'.split('.')[0].,
                        ),
                        trailing: IconButton(
                          icon: const Icon(Icons.delete, color: Colors.red),
                          tooltip: 'Видалити нотатку',
                          onPressed: () => _confirmAndDeleteNote(note),
                        ),
                        onTap: () {
                          Navigator.push(
```

```
                                context,
                                MaterialPageRoute(
                                  builder: (_) => NoteDetailScreen(note: note),
                                ),
                              );
                            },
                          );
                        },
                      );
                    },
                  );
                },
              ),
      floatingActionButton: FloatingActionButton(
        onPressed: () async {
          await Navigator.push(
            context,
            MaterialPageRoute(builder: (_) => const AddNoteScreen()),
          );
          // Оновлюємо список нотаток після повернення
          setState(() {
            _notesFuture = _fetchNotes();
          });
        },
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

## encrypted_note_model.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';

class EncryptedNote {
  final String id;
  late final String cipherTitle;
  late final String cipherContent;
  late final String nonceTitle;
  late final String macTitle;
  late final String nonceContent;
  late final String macContent;
  final DateTime createdAt;

  EncryptedNote({
    required this.id,
    required this.cipherTitle,
    required this.cipherContent,
    required this.nonceTitle,
    required this.macTitle,
    required this.nonceContent,
    required this.macContent,
    required this.createdAt,
  });

  factory EncryptedNote.fromMap(String id, Map<String, dynamic> data) {
    return EncryptedNote(
      id: id,
      cipherTitle: data['title'].,
      cipherContent: data['content'].,
      nonceTitle: data['nonceTitle'].,
      macTitle: data['macTitle'].,
      nonceContent: data['nonceContent'].,
      macContent: data['macContent'].,
```

```
      createdAt: (data['createdAt']. as Timestamp).toDate(),
    );
  }

  Map<String, dynamic> toMap() {
    return {
      'title': cipherTitle,
      'content': cipherContent,
      'nonceTitle': nonceTitle,
      'macTitle': macTitle,
      'nonceContent': nonceContent,
      'macContent': macContent,
      'createdAt': createdAt,
    };
  }
}
```

## encryption_key_provider.dart

```
import 'package:flutter/material.dart';
import 'package:cryptography/cryptography.dart';

class EncryptionKeyProvider with ChangeNotifier {
  SecretKey? _key;

  SecretKey? get key => _key;

  void setKey(SecretKey key) {
    _key = key;
    notifyListeners();
  }

  void clearKey() {
    _key = null;
    notifyListeners();
  }
}
```

**Додаток Б**

# ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Захищений мобільний застосунок для зберігання нотаток

Автор роботи: Пальчик Вадим Андрійович

Тип роботи: бакалаврська кваліфікаційна робота

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

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism **0,3** %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

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

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

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

Експертна комісія:

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

Гарант освітньої програми «Безпека інформаційних та комунікаційних систем» к. т. н., доц. Леонід КУПЕРШТЕЙН

Особа, відповідальна за перевірку Валентина КАПЛУН

З висновком експертної комісії ознайомлений(-на)

Керівник Володимир ГАРНАГА

Здобувач Вадим ПАЛЬЧИК

**Додаток В**

# ІЛЮСТРАТИВНА ЧАСТИНА

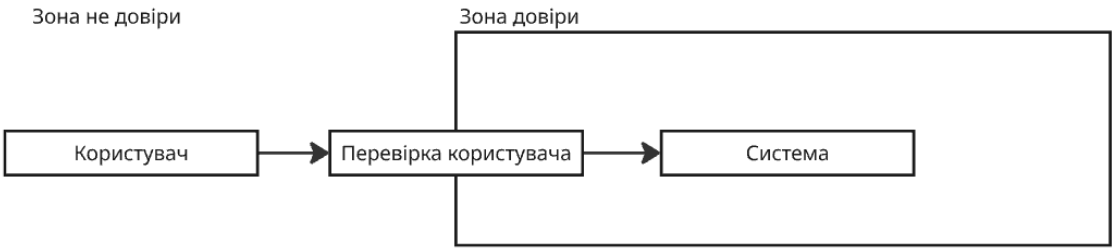
## ЗАХИЩЕНИЙ МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ЗБЕРІГАННЯ НОТАТОК

Виконав: студент 4 курсу групи 1БС–21б
спеціальності 125 Кібербезпека
_____ Вадим ПАЛЬЧИК
13 червня 2025 р.

Керівник: к. т. н., доцент каф. ЗІ
_____ Володимир ГАРНАГА
13 червня 2025 р.

# СХЕМА АТАКИ «ЛЮДИНА ПОСЕРЕДИНІ» (MITM)

# СХЕМА ПЕРИМЕТРОВОЇ МОДЕЛІ БЕЗПЕКИ



**П’ять основних принциипів Zero Trust Architecture.**

**Загальна схема взаємодії з застосунком**

**Загальна схема Архітектури додатку**

**Схема автентифікації в застосунку**

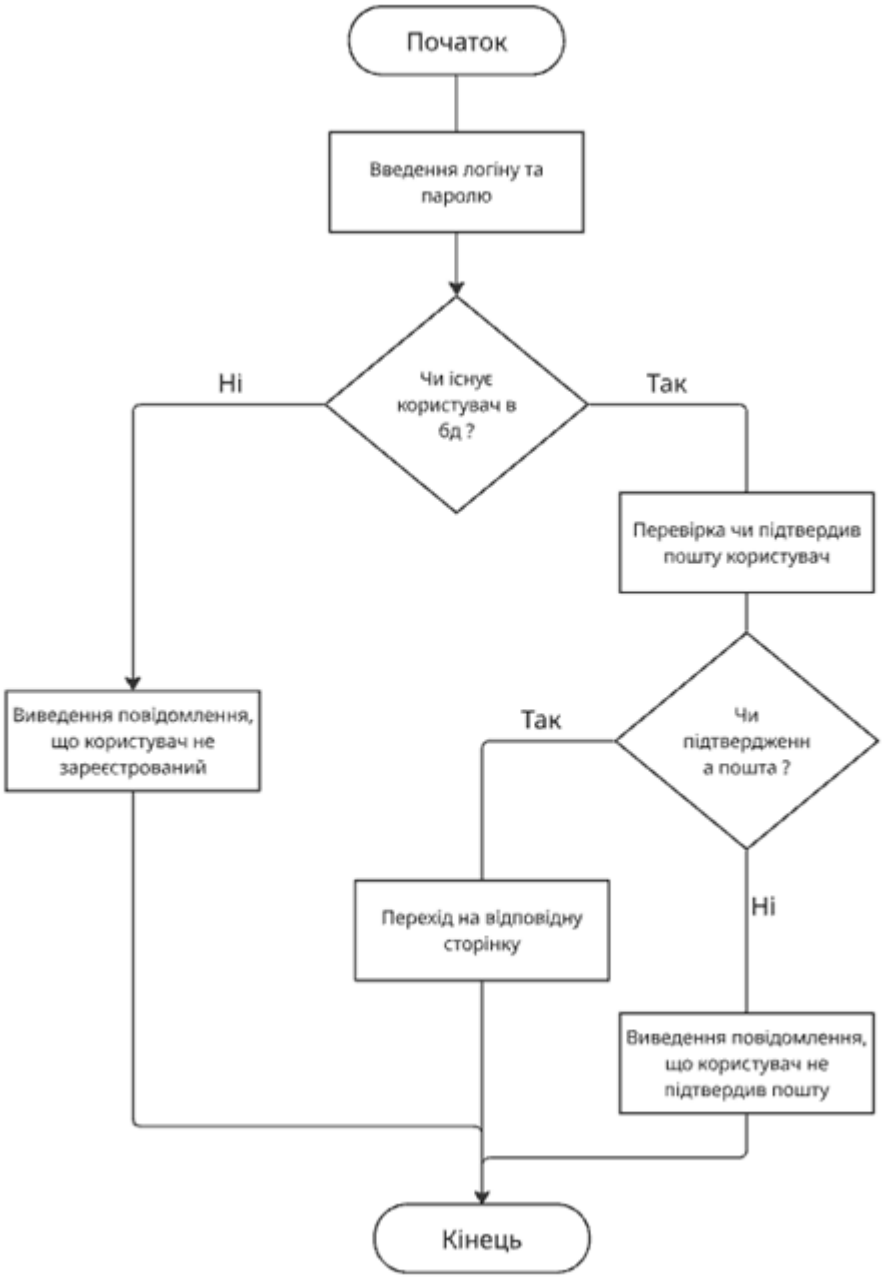


**Схема шифрування даних**

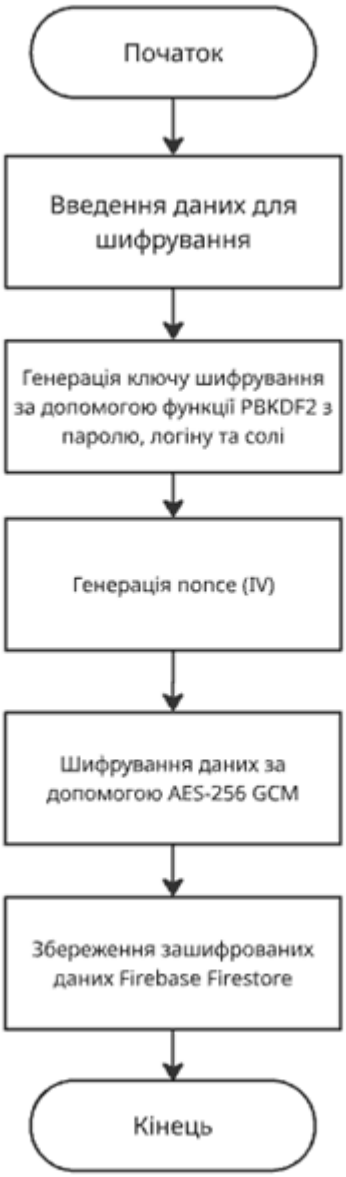
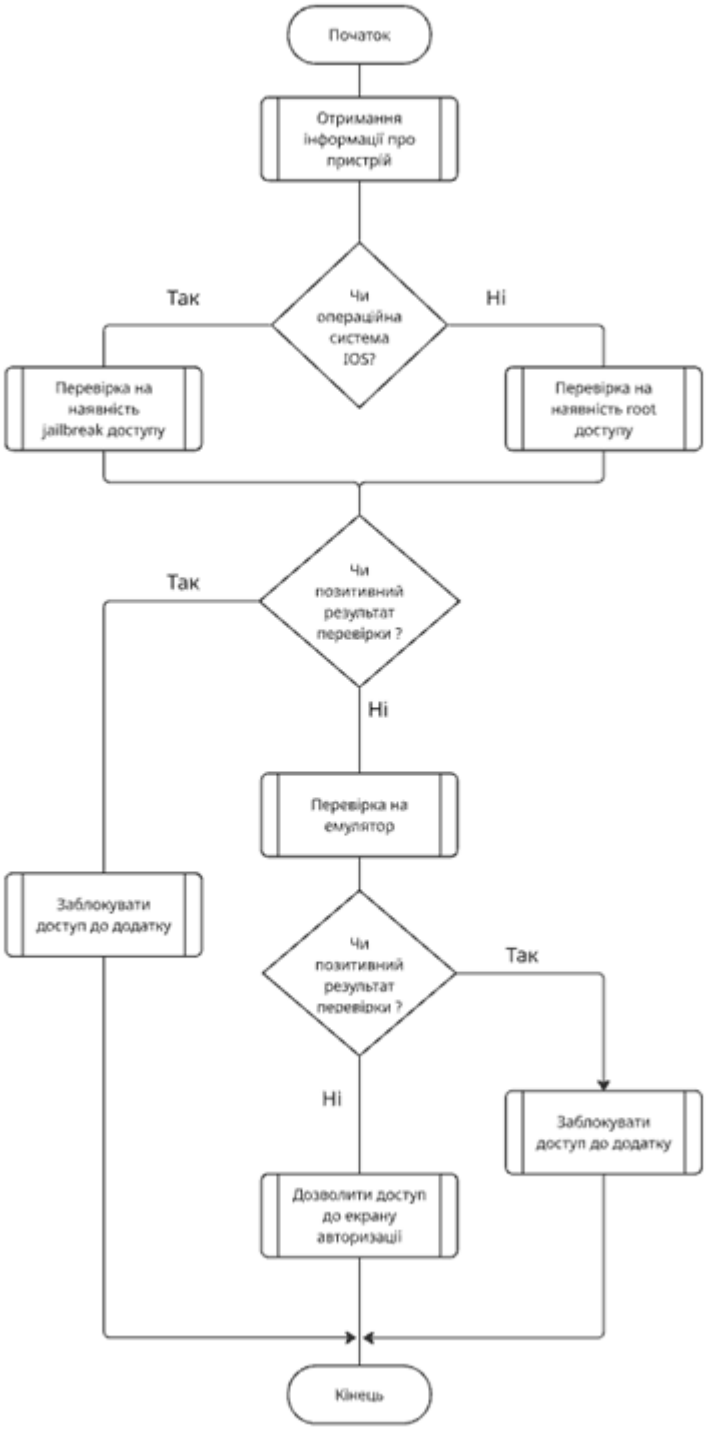
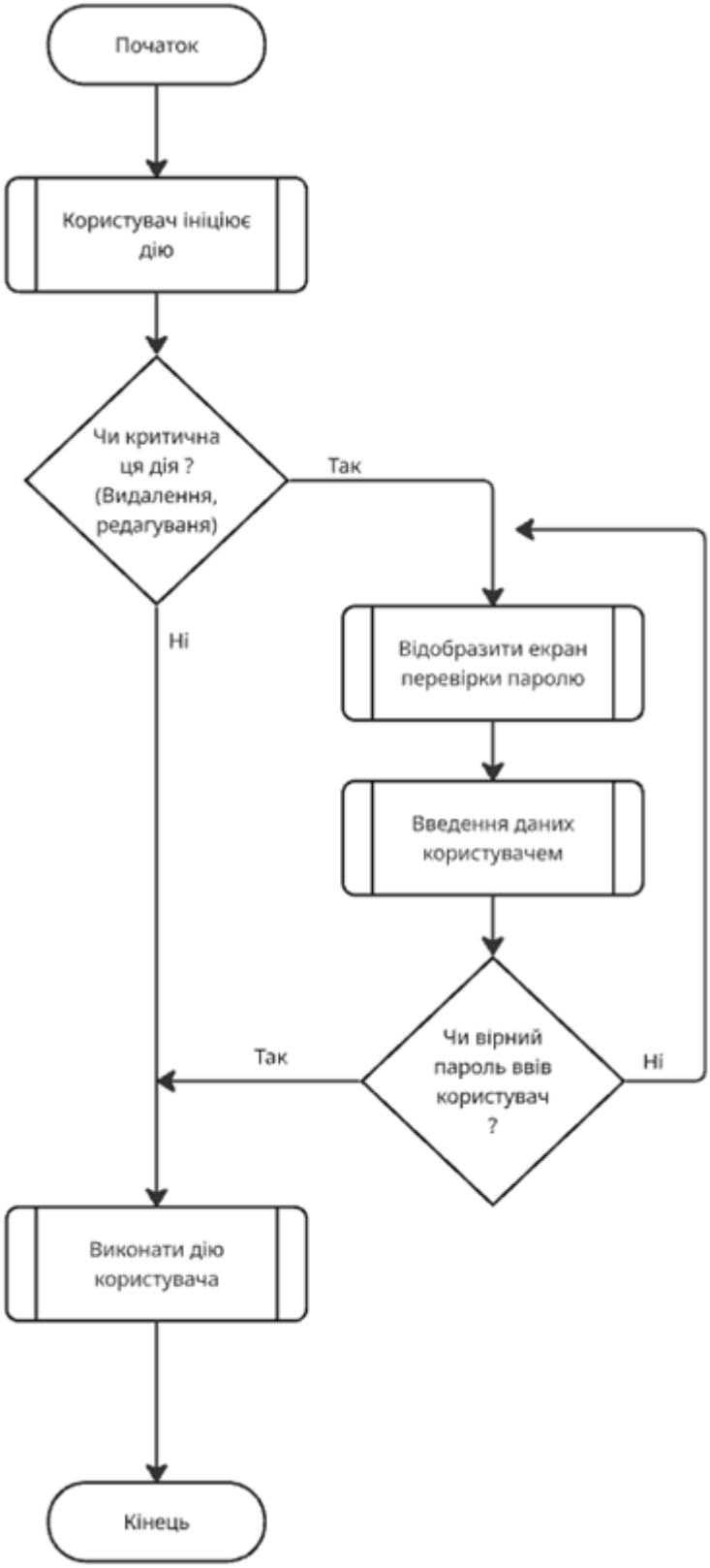


**Схема розшифрування даних**

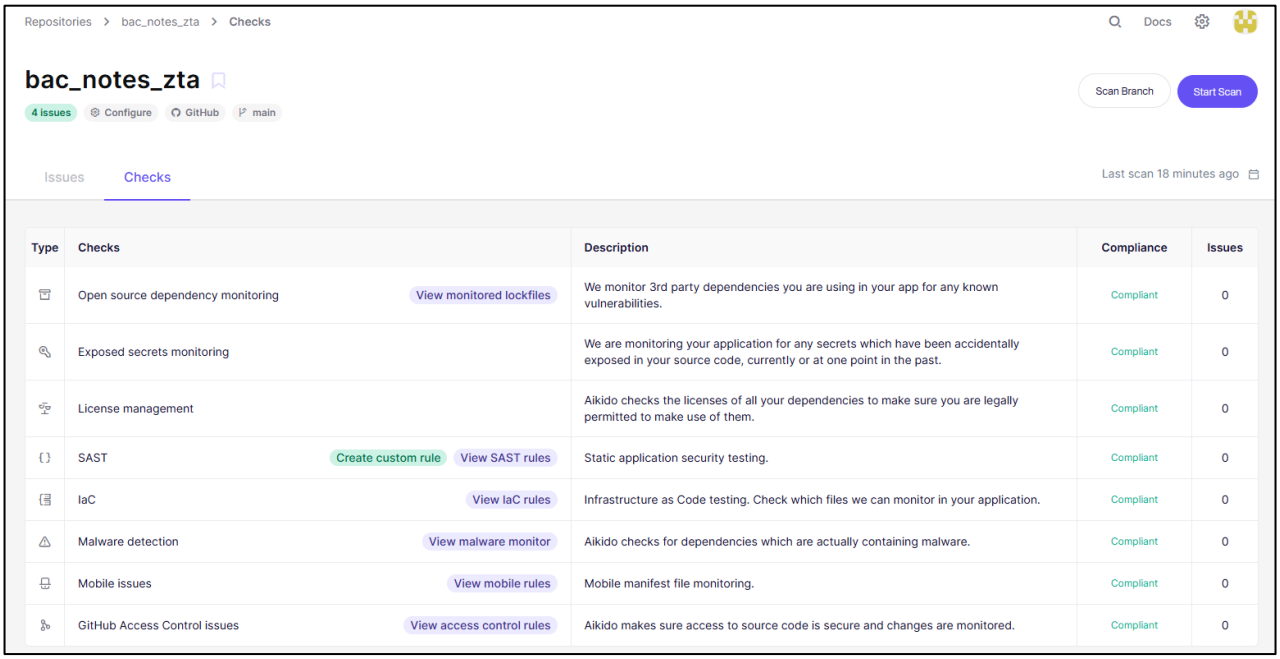
**Схема перевірки пристрою перед входом в застосунок**



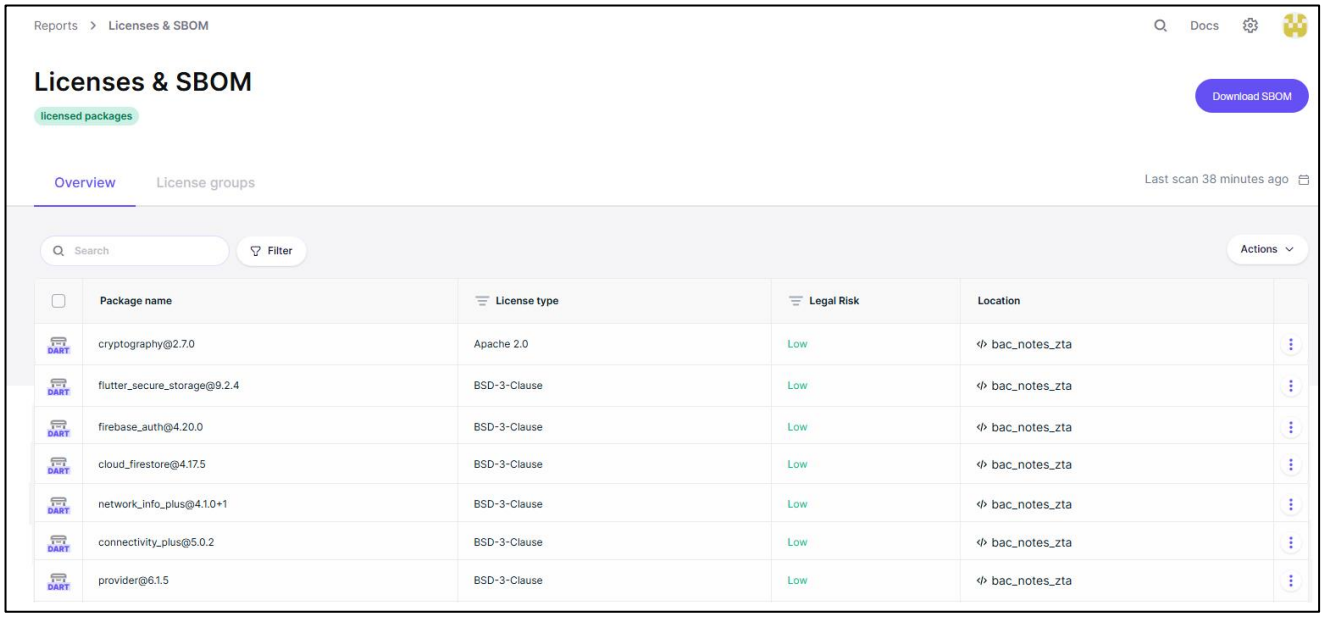
**Перевірка пароля перед виконанням критичної дії**



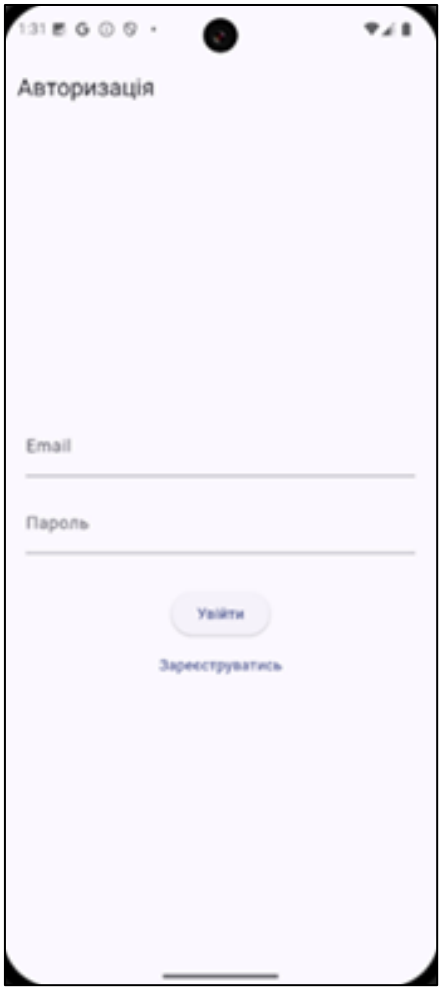
# РЕЗУЛЬТАТИ ТЕСТУВАННЯ ДОДАТКУ




Загальний звіт перевірки проєкту в Aikido Security

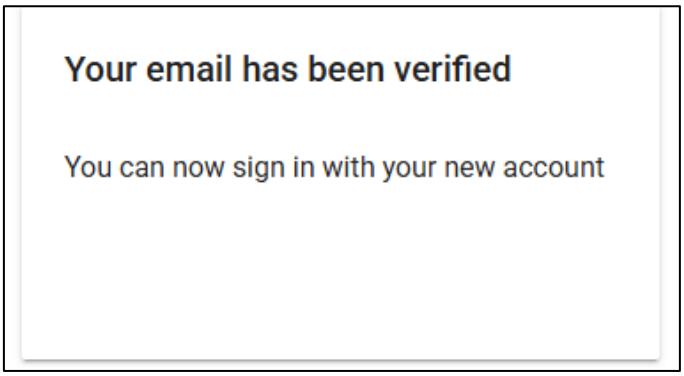



Стан безпечності бібліотек згідно зі скануванням Aikido

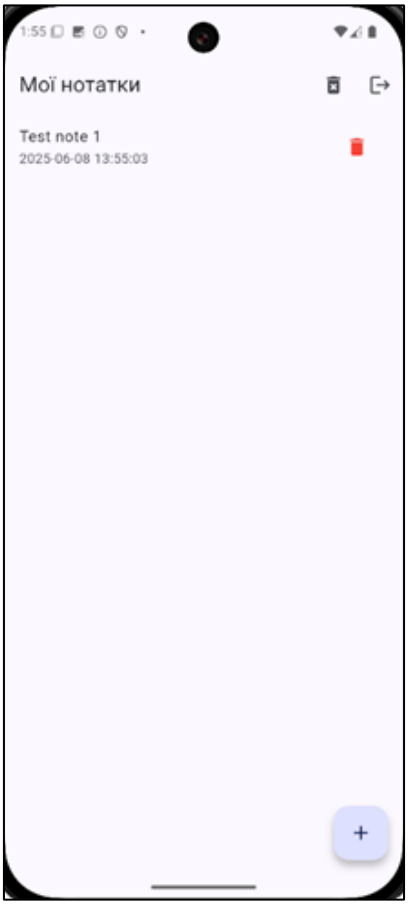



Екран реєстрації та авторизації користувача

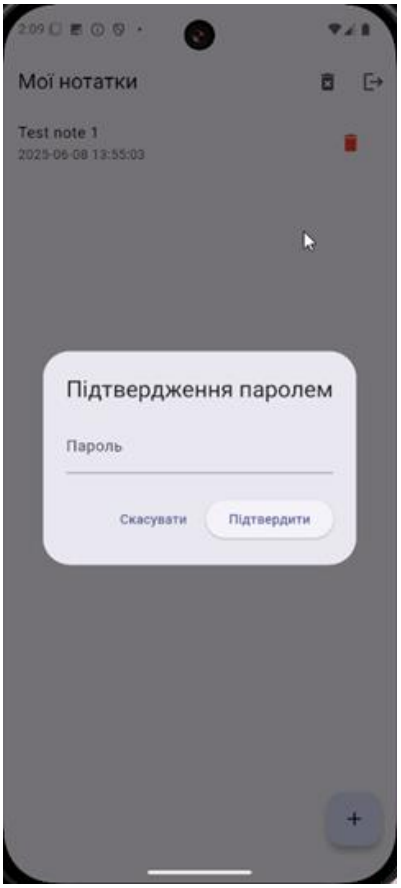

Лист з посиланням для підтвердження електронної пошти




Вигляд повідомлення про успішне підтвердження електронної пошти

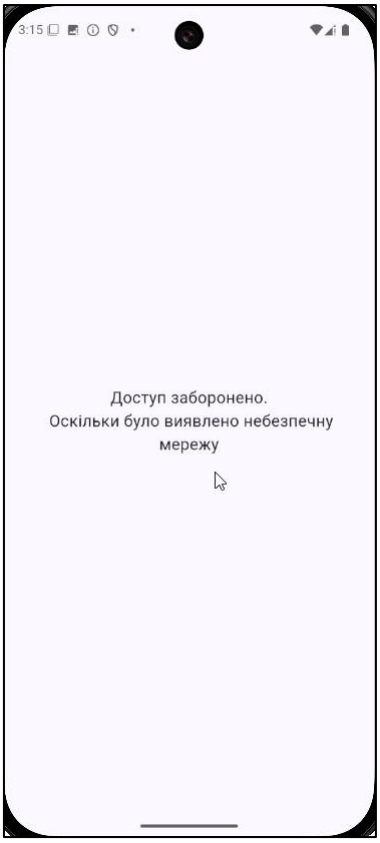



Вигляд головного екрану



Повторна перевірка паролем


Заборона доступу до додатку та повідомлення про скомпрометоване середовище виконання



Заборона доступу до додатку та попередження про ризиковане мережеве підключення.