**Міністерство освіти і науки України**
**Луцький національний технічний університет**
**Факультет комп'ютерних та інформаційних технологій**
**Кафедра інженерії програмного забезпечення**

# КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

## РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ДОДАТКУ ДЛЯ ЗБЕРІГАННЯ ПАРОЛІВ ЗА ДОПОМОГОЮ KOTLIN

## DEVELOPMENT OF A CLIENT-SERVER APPLICATION FOR STORING PASSWORDS USING KOTLIN

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗ-42
**Пех Ростислав Миколайович**
_____
(підпис)

Керівник:
к.т.н., доцент
Гулієва Наталія Михайлівна
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«8» 06 2024 р.
к.т.н., доцент
Гарант освітньої програми:
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Н. Лящук
«2» 02 2024 р.

# ЗАВДАННЯ
## НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

*Пеху Ростиславу Миколайовичу*
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *Розробка клієнт-серверного додатку для зберігання паролів за допомогою Kotlin*

Керівник роботи: к.т.н., доцент, Гулієва Наталія Михайлівна
затверджені наказом закладу вищої освіти від *«30» грудня 2023 р. № 477/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«5» червня 2024 р.*
3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
*Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійністних параметрів проектованої системи, функціонально-структурна схема роботи об'єкта проектування, розробка моделі бази даних об'єкта проектування.*

5. Перелік графічного матеріалу: *12 рисунків.*

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| *Аналіз предметної області* | *Гулієва Н. М.* | | |
| *Специфікації вимог до розробленої системи* | *Гулієва Н. М.* | | |
| *Розробка об'єкта проєктування* | *Гулієва Н. М.* | | |
| *Нормоконтроль* | *Повстяна Ю. С.* | | |
| *Гарант ОП* | *Ліщина Н. М.* | | |
| *Показник запозичень тексту* | | 2,5% | |
| *Академічна доброчесність* | *Ящук А. А.* | | |

7. Дата видачі завдання *«2» лютого 2024 р.*

КАЛЕНДАРНИЙ ПЛАН

| № З/п | Назва етапів кваліфікаційної роботи бакалавра | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | *Огляд літературних джерел по темі кваліфікаційної роботи бакалавра* | *07.02.2024 р.* | вик. |
| 2 | *Аналіз проблеми розробки та впровадження об'єкту проектування* | *27.02.2024 р.* | вик. |
| 3 | *Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання* | *12.03.2024 р.* | вик. |
| 4 | *Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних* | *17.04.2024 р.* | вик. |
| 5 | *Практична реалізація об'єкта проектування та розробка бази даних* | *18.05.2024 р.* | вик. |
| 6 | *Тестування та налагодження об'єкта проектування* | *01.06.2024 р.* | вик. |
| 7 | *Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру* | *05.06.2024 р.* | вик. |
| | | | |
| | | | |

Здобувач вищої освіти _____ (підпис) *Пех Р. М.* (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) *Гулієва Н. М.* (прізвище, ініціали)

Міністерство освіти і науки України
Луцький національний технічний університет

# Рецензія
# на кваліфікаційну роботу

**Здобувач вищої освіти:** *Пех Ростислав Миколайович*

**Тема:** *«Розробка клієнт-серверного додатку для зберігання паролів за допомогою Kotlin».*

**Коротка характеристика кваліфікаційної роботи та прийнятих рішень:** *Кваліфікаційна робота бакалавра складається з трьох розділів, в яких проаналізовані проблематика та поставлені завдання за темою роботи, здійснено теоретичне дослідження та практичну реалізацію додатку для зберігання паролів , здійснено тестування розробленої системи*

**Самостійні розробки і пропозицій автора:** *Автор самостійно виконав розробку додатку для зберігання паролів*

**Практичне значення роботи** *полягає в створенні безпечного та зручного інструменту для управління паролями, який забезпечує надійне зберігання паролів та їх синхронізацію між пристроями користувача.*

**Недоліки:** *Істотних недоліків в роботі не виявлено.*

**Загальний висновок:** *Кваліфікаційна робота бакалавра здобувача освіти Пеха Ростислава Миколайовича присвячена розробці клієнт-серверного додатку для зберігання паролів за допомогою мови програмування Kotlin. У роботі проаналізовано проблематику та поставлені завдання, виконано теоретичне дослідження та практичну реалізацію додатку, а також проведено тестування розробленої системи.*

*Автор самостійно виконав розробку додатку, створивши безпечний та зручний інструмент для управління паролями. Додаток забезпечує надійне зберігання паролів та їх синхронізацію між пристроями користувача, що надає йому значне практичне значення.*

*Кваліфікаційна робота здобувача освіти Пеха Ростислава Миколайовича рекомендується до захисту екзаменаційній комісії та заслуговує позитивної оцінки.*

**Рецензент кваліфікаційної роботи:** *Багнюк Наталія Володимирівна, к.т.н., доцент кафедри комп'ютерної інженерії та кібербезпеки*
(прізвище, ім'я, по-батькові, посада)

(підпис)

«07» червня 2024 р.

Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

**ВІДГУК**
**КЕРІВНИКА НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Здобувач вищої освіти *Пех Ростислав Миколайович*
(прізвище, ім'я, по батькові)
група *ІПЗ-42*

**Тема кваліфікаційної роботи:**
*Розробка клієнт-серверного додатку для зберігання паролів за допомогою Kotlin*
**Керівник** *Гулієва Наталія Михайлівна*

**Актуальність теми** *У сучасному світі основним ресурсом є інформація. З кожним роком кількість онлайн-сервісів зростає, збільшуючи обсяг особистої інформації, яка вимагає надійного зберігання. Користувачі мають велику кількість облікових записів, які вимагають складних і унікальних паролів для забезпечення безпеки. Неправильне зберігання або використання однакових паролів на різних сайтах призводить до значних ризиків витоку особистих даних. Враховуючи це, розробка надійного менеджера паролів стає актуальним завданням для забезпечення захисту інформації.*

**Об'єкт дослідження** *Процеси зберігання та передачі паролів в умовах підвищеної потреби в інформаційній безпеці*

**Характеристика теоретичного рівня, наявності самостійних розробок і практичної значимості роботи:**
*Автор аналізує різні аналоги та їх функціональні особливості, обґрунтовує своє рішення щодо вибору технологій та алгоритмів вирішення поставленого завдання. Автор здійснив розробку додатку з використанням Kotlin. Зміст роботи відповідає обраній темі.*

**Зауваження та недоліки:** *Суттєвих недоліків в роботі не виявлено.*

**Загальний висновок**: *Кваліфікаційна робота бакалавра виконана на високому рівні. Робота виконана відповідно до вимог, логічно й послідовно описує хід виконання поставленого завдання. Пояснювальна записка відповідає стандартам до її оформлення. Робота може бути оцінена на високу оцінку*

**Керівник** _____ **(*Гулієва Н. М.*)**
(підпис) (прізвище, ім'я, по батькові)

«07» 06 2024 р.

## АНОТАЦІЯ

Пех Р. М. Розробка клієнт-серверного додатку для зберігання паролів за допомогою Kotlin. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2024.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків і пропозицій, списку використаних джерел та додатків.

У першому розділі здійснено аналіз сучасного стану проблеми розробки клієнт-серверних додатків і сформульовано завдання. В другому розділі визначено вимоги до розроблюваної системи, а також засоби, методи і алгоритми розробки. У третьому розділі розроблено клієнт-серверний додаток для зберігання паролів і здійснено його тестування. У висновках узагальнено інформацію, відображену у попередніх частинах. Результати розробки можуть бути застосовані в практичній діяльності для безпечного зберігання паролів.

Ключові слова: пароль, Kotlin, клієнт, сервер, захист даних.



## ABSTRACT

Pekh R. M. Development of a client-server application for storing passwords using Kotlin.

Manuscript. Bachelor's qualifying thesis of the educational program "Software Engineering". Lutsk National Technical University. Lutsk, 2024.

The bachelor's qualifying thesis consists of an introduction, three sections, conclusions and proposals, a list of used sources and annexes.

In the first chapter, an analysis of the current state of the problem of developing client-server applications was carried out and the task was formulated. The second chapter defines the requirements for the developed system, as well as means, methods and development algorithms. In the third chapter, a client-server application for storing passwords was developed and tested. The conclusions summarize the information presented in the previous parts. The development results can be applied in practical activities for safe storage of passwords.

Keywords: password, Kotlin, client, server, data protection.

**ЗМІСТ**

# ВСТУП

Актуальність теми. У сучасному світі основним ресурсом є інформація. З кожним роком кількість онлайн-сервісів зростає, збільшуючи обсяг особистої інформації, яка вимагає надійного зберігання. Користувачі мають велику кількість облікових записів, які вимагають складних і унікальних паролів для забезпечення безпеки. Неправильне зберігання або використання однакових паролів на різних сайтах призводить до значних ризиків витоку особистих даних. Враховуючи це, розробка надійного менеджера паролів стає актуальним завданням для забезпечення захисту інформації.

Метою роботи є розробка клієнт-серверного додатку менеджера паролів з підтримкою хмарних функцій, таких як синхронізація між пристроями і безпечний обмін паролями.

Реалізація мети вимагає вирішення наступних завдань:

– аналіз існуючих рішень для керування паролями та їх основних функцій;

– формулювання вимог до клієнт-серверного додатку менеджера паролів;

– вибір та вивчення алгоритмів шифрування та передачі даних;

– підбір архітектурного рішення та методів реалізації;

– проєктування інтерфейсу та взаємодії користувача з клієнтським додатком;

– розробка додатку менеджера паролів;

– тестування розробленого менеджера паролів та виправлення помилок.

Об’єктом дослідження є процеси зберігання та передачі паролів в умовах підвищеної потреби в інформаційній безпеці.

Предметом дослідження є методи і алгоритми забезпечення безпеки при зберіганні та передачі паролів, а також архітектура клієнт-серверного додатку для менеджера паролів.

Практична цінність розробки полягає в створенні безпечного та зручного інструменту для управління паролями, який забезпечує надійне зберігання паролів та їх синхронізацію між пристроями користувача. Це допоможе знизити ризики втрати особистих даних та підвищити загальний рівень безпеки користувачів в Інтернеті.

# РОЗДІЛ 1
# АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ КЕРУВАННЯ ПАРОЛЯМИ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

## 1.1 Аналіз сучасного стану проблеми

Інформація є ключовим ресурсом в сучасному світі. Більшість сервісів тепер пропонують свої послуги онлайн, що знімає потребу у фізичному відвідуванні численних організацій. Щороку кількість нових вебсайтів збільшується, що відповідно підвищує обсяг особистої інформації, яка потребує надійного зберігання. Сучасний інтернет-користувач має численні облікові записи для електронної пошти, роботи, соціальних мереж, месенджерів, інтернет-банків та магазинів. Кількість таких акаунтів постійно зростає, створюючи потребу в їх безпечному зберіганні та конфіденційності.

Зазвичай, для аутентифікації на вебсайтах використовуються логін і пароль. Цей метод є зручним для користувачів, але має багато вразливостей. Зі збільшенням кількості онлайн-сервісів зростає й кількість паролів, які користувач повинен запам’ятати. Часто користувачі змушені відновлювати паролі, що веде до використання однакових паролів на різних сайтах або зберігання їх у небезпечних місцях, таких як записки на папері або у нотатках телефону. Це значно підвищує ризик витоку особистої інформації. За даними HaveIBeenPwned, база даних скомпрометованих облікових записів містить понад 13,5 мільярдів записів [1].

Крім того, користувачам потрібно використовувати різні паролі та регулярно їх змінювати, створюючи складні комбінації цифр, символів і літер. Багато хто нехтує цими вимогами, що підтверджує дослідження NordPass [2], яке показує, що найпопулярніші паролі – це «123456», «admin» або «password» і вони є надзвичайно слабкими.

Для надійного зберігання складних паролів без потреби запам’ятовувати їх всі існують менеджери паролів. Це програми, які допомагають користувачам керувати своїми паролями. Паролі зберігаються у зашифрованому вигляді в

файлі або базі даних, доступ до яких можна отримати за допомогою майстер-пароля або біометричних даних. Менеджери паролів діляться на 3 основні типи: апаратне рішення, програмне забезпечення з локальним зберіганням і хмарне рішення.

Апаратне рішення – найбезпечніший варіант, оскільки апаратні пристрої можуть не мати доступу до Інтернету та використовують шифрування на рівні файлової системи. Використовуються переважно у виняткових випадках, таких як зберігання ключів від криптовалютних гаманців.

Програмне забезпечення з локальним зберіганням передбачає, що зашифрований файл зберігається на локальному диску і використовується лише на одному пристрої. Відсутність синхронізації між пристроями створює незручності для користувачів, які мають кілька пристроїв.

Хмарне рішення передбачає, що зашифрована база даних зберігається на сервері та синхронізується між пристроями користувача. Хоча це може здаватися небезпечним, використання сучасного шифрування гарантує, що несанкціонований доступ до даних неможливий. Майстер-пароль не зберігається на сервері та ніколи не передається за межі клієнтського пристрою.

Для розробки ефективного клієнт-серверного додатку для зберігання паролів важливо вивчити існуючі рішення, їх функціональність, переваги та недоліки. Це дозволить зрозуміти найкращі практики, а також визначити області для вдосконалення, а також забезпечити конкурентоспроможність нового додатку.

На сьогоднішній день існує велика кількість рішень для зберігання паролів. Розглянемо деякі популярні менеджери паролів, що доступні для операційної системи Android [3].

На запит «менеджер паролів» Google Play [4] пропонує такі популярні рішення як LastPass, 1Password, Bitwarden, Password Safe and Manager, Dashlane та інші (рис. 1.1).

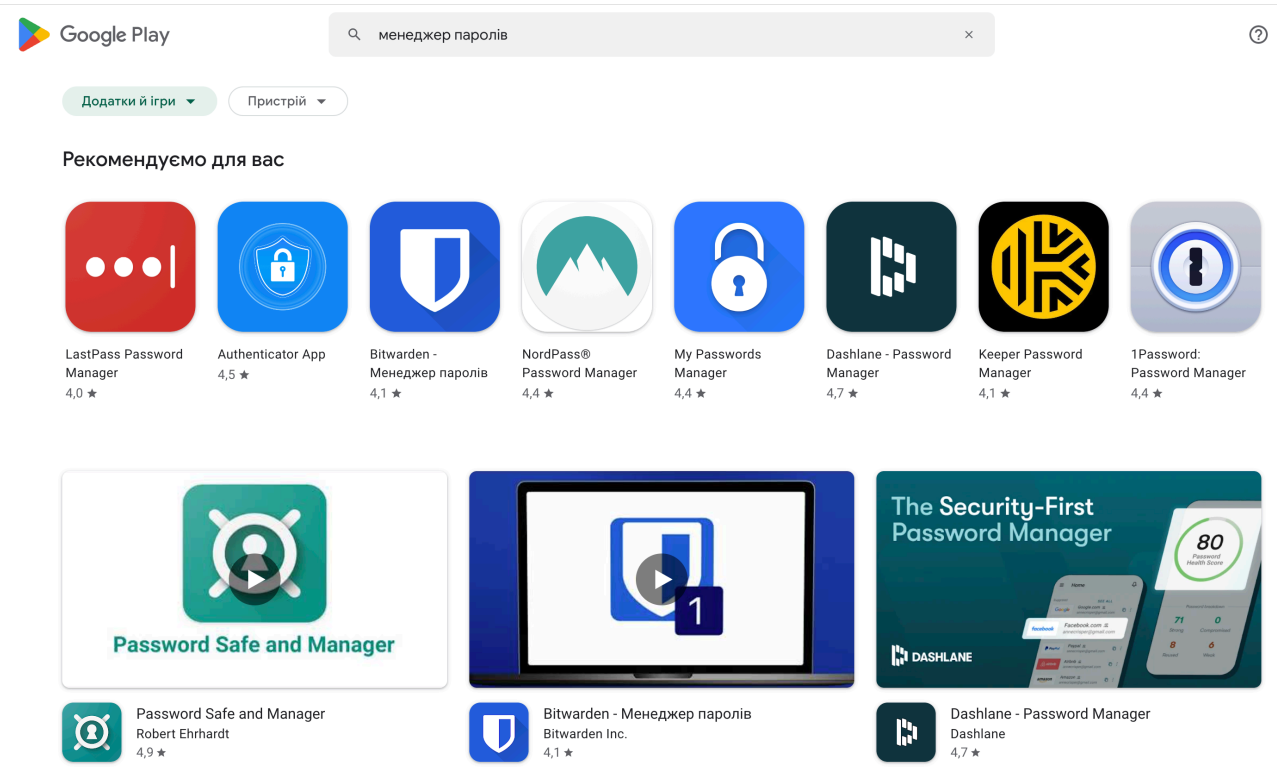


Рисунок 1.1 – Менеджери паролів, що пропонуються у Google Play

Password Safe and Manager [5] забезпечує надійне зберігання та управління всіма даними в зашифрованому вигляді, що гарантує їх безпеку. Користувачу потрібно запам’ятати лише один головний пароль. Цей менеджер паролів дозволяє легко керувати та відстежувати всі дані. Використовується потужне 256-бітне шифрування AES для захисту даних.

Password Safe не підключається до Інтернету. Цей менеджер паролів працює лише в автономному режимі, тому функція автоматичної синхронізації відсутня через відсутність доступу до Інтернету.

Для обміну сховищами можна створити резервну копію бази даних на сховищах типу Dropbox та імпортувати її на інший пристрій за допомогою вбудованої функції експорту та імпорту.

Основні можливості Password Manager:

– безпечне зберігання та управління паролями, пін-кодами, обліковими записами та даними доступу;

– класифікація даних у Password Safe;

– доступ через єдиний головний пароль;
– генератор паролів для створення надійних паролів;
– резервне копіювання та відновлення зашифрованої бази даних;
– зручне налаштування інтерфейсу користувача;
– статистика;
– автоматичне очищення буфера обміну (з деякими обмеженнями на певних пристроях);
– віджети для генерування паролів;
– автоматичне резервне копіювання;
– імпорт та експорт у форматі CSV;
– індикатор надійності пароля;
– відсутність непотрібних прав для Android;
– підтримка Wear OS.

Також пропонуються додаткові фнкції у платній версії програми, зокрема додавання зображень до записів, призначення та організація власних полів введення, біометричний вхід (відбиток пальця, розпізнавання обличчя тощо), архівування записів, експорт у PDF, друк та інші.

Інтуїтивний дизайн допомагає легко та ефективно керувати даними. Безпеку гарантує використання 256-бітного шифрування AES.

Дані додатку повністю зашифровані, тому у випадку втрати головного пароля відновити дані неможливо.

LastPass [6] – це менеджер паролів, який захищає паролі та особисту інформацію у зашифрованому сховищі. Під час відвідування додатків та вебсайтів, LastPass автоматично заповнює облікові дані для входу. За допомогою сховища LastPass можна зберігати паролі та логіни, створювати профілі для покупок онлайн, генерувати надійні паролі, безпечно зберігати особисту інформацію у нотатках тощо. Все, що для цього потрібно зробити, це запам’ятати свій майстер-пароль, і LastPass автоматично заповнить дані для входу у браузер або додаток.

LastPass не має доступу до зашифрованих даних користувачів, тому тільки користувач маєте до них доступ. Сховище користувача захищене 256-бітним AES-шифруванням.

Понад 30 мільйонів користувачів та понад 85 тис компаній використовують LastPass. LastPass отримав високу оцінку від PCWorld, Inc., PCMag, ITProPortal, LaptopMag, TechRadar, U.S. News & World Report, NPR, TODAY, TechCrunch, CIO та інших.

LastPass використовує функцію доступності Android, щоб забезпечити безперебійне заповнення даних для входу у додатки та вебсайти, навіть на старих версіях Android, які не підтримують автозаповнення.

Bitwarden [7] – це простий і безпечний спосіб зберігання всіх паролів користувача та їх зручна синхронізація між всіма пристроями.

Bitwarden також зберігає всі дані користувача у зашифрованому сховищі, яке синхронізується на всіх ваших пристроях. Дані повністю шифруються ще до того, як покинуть пристрій користувача, тому тільки користувач має до них доступ. Дані захищені алгоритмами AES-256 та PBKDF2 SHA-256.

Bitwarden – це програмне забезпечення з відкритим кодом. Весь код Bitwarden доступний на GitHub, де будь-хто може його переглянути, провести аудит або взяти участь у розробці.

Dashlane [8] – це передовий менеджер паролів, який відрізняється простотою використання та високим рівнем безпеки. Він допомагає як окремим користувачам, так і компаніям захищати свої паролі, платіжну інформацію та особисті дані, забезпечуючи доступ до них з будь-якого місця за допомогою найкращих засобів захисту.

Сховище користувача автоматично синхронізується за допомогою Dashline на всіх пристроях, що дозволяє легко отримувати доступ, генерувати або безпечно ділитися паролями. Dashlane також дозволяє організувати сховище в персоналізовані колекції для зручного пошуку та фільтрації паролів.

Такі функції, як автозаповнення, полегшують введення паролів та платіжної інформації онлайн. Крім того, функція моніторингу Dark Web від Dashlane стежить за інтернет-простором, щоб попередити вас про будь-які підозрілі активності.

Код додатку Dashline для Android та iOS є загальнодоступним, що дозволяє кожному перевірити його та зрозуміти, як працює Dashlane.

Поряд з управлінням паролями Dashlane активно працює над впровадженням безпарольних технологій, таких як підтримка ключів доступу та безпарольний вхід, залишаючись на передовій галузі.

Dashlane має нагороди та визнання, такі як:

– найкращий менеджер паролів за надійністю (Forbes Advisor);
– вибір редакції (PC World);
– лідер серед менеджерів паролів (G2: весна 2023).

Додаток Dashlane для Android використовує AccessibilityService API для забезпечення можливості автозаповнення на пристроях з Android 8 та Android 9.

1Password: Password Manager [9] – це ще один менеджер паролів. 1Password з 2006 року допомагає користувачам забути про необхідність запам’ятовувати паролі. За версією The New York Times [10], «1Password пропонує найкраще поєднання функцій, сумісності, безпеки та простоти використання» серед менеджерів паролів. Йому довіряють мільйони людей і понад 150 тис. компаній.

1Password пропонує вбудований генератор паролів для створення складних паролів, функцію автозаповнення, яка автоматично вводить імена користувачів та збережені паролі під час входу на веб-сайти чи в додатки та інші функції.

1Password також може генерувати та автоматично заповнювати одноразові двофакторні коди для служб, які підтримують 2FA, що усуває потребу у використанні окремого додатка для аутентифікації.

Результати аналізу популярних менеджерів паролів зведені до таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика популярних менеджерів паролів для Android

| Характеристика | Менеджер паролів | | | | |
|---|---|---|---|---|---|
| | astPass | 1Password | Bitwarden | Password Safe and Manager | Dashlane |
| Загальне шифрування | AES-256 | AES-256 | AES-256 | AES-256 | AES-256 |
| Генератор паролів | Так | Так | Так | Так | Так |
| Двофакторна аутентифікація | Так | Так | Так | Так (PRO версія) | Так |
| Автоматичне заповнення | Так | Так | Так | Ні | Так |
| Синхронізація пристроїв | Так | Так | Так | Ні (автономний режим) | Так |
| Резервне копіювання | Так | Так | Так | Так | Так |
| Підтримка біометрії | Так | Так | Так | Так (PRO версія) | Так |
| Автономний режим | Ні | Ні | Так | Так | Ні |
| Зберігання захищених нотаток | Так | Так | Так | Так | Так |
| Поділитися паролями | Так | Так | Так | Так | Так |
| Модель безпеки | Архітектура з нульовим знанням | Архітектура з нульовим знанням | Архітектура з нульовим знанням | Працює в автономному режимі, без доступу до Інтернету | Архітектура з нульовим знанням |
| Додаткові функції | Моніторинг darkweb, інструменти для бізнесу | Підтримка ключів доступу, зберігання медичних записів, захищені нотатки | Підтримка ключів доступу, відкритий код, можливість самостійного розгортання | Wear OS App, індикатор надійності пароля, імпорт/експорт в CSV, архівні записи (PRO версія) | Моніторинг темного веб, автозаповнення, організація паролів в колекції |

В результаті аналізу можна виділити основні функції менеджерів паролів. Розглянемо їх далі.

Генерація паролів. Ця функція дозволяє генерувати скадні і надійні паролі для нових облікових записів. Також може бути реалізовано налаштування параметрів генерації (довжина пароля, включення спеціальних символів, цифр тощо).

Автентифікація користувача. Двофакторна аутентифікація (2FA) користувача є важливою функцією для додаткового рівня безпеки. Підтримка біометричних методів аутентифікації (відбиток пальця, розпізнавання обличчя) також є сучасним підходом, що забезпечує зручний і надійний спосіб захисту даних.

Автоматичне заповнення полів для входу на веб-сайти та в додатки реалізується через підтримку браузерних розширень, що забезпечує спрощення входу на веб-сторінки.

Зберігання паролів. Менеджери паролів забезпечують безпечне зберігання паролів у зашифрованому вигляді. Сучасні менеджери паролів дозволяють хмарну синхронізацію паролів між різними пристроями користувача. Менеджери паролів використовують потужні алгоритми шифрування, такі як AES-256.

Серед інших методів, покликаних забезпечити високий рівень захисту є автоматичне блокування додатку після певного періоду неактивності, перевірка безпеки паролів та рекомендації щодо їх оновлення, виявлення зламаних облікових записів, оповіщення користувача про потенційні загрози.

## 1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Після аналізу існуючих рішень для керування паролями та їх основних функцій, було поставлено такі завдання:

– сформулювати вимоги до клієнт-серверного додатку менеджера паролів, визначити можливості і функціональність, які повинні бути реалізовані в новому

продукті;

– здійснити вибір та вивчення алгоритмів шифрування та передачі даних, зокрема криптографічні алгоритми для захисту даних, протокол безпечної передачі даних між клієнтом та сервером;

– підібрати архітектурне рішення та методи його реалізації, зокрема розробити архітектуру клієнт-серверного додатку і обрати технології для реалізації серверної частини (бекенду) та клієнтського додатку, розробити стуктуру бази даних, спроєктувати сховище облікових записів та структуру даних для зв’язку з сервером, здійснити вибір системи управління базою даних для зберігання зашифрованих даних.

– здійснити проєктування інтерфейсу та взаємодії користувача з клієнтським додатком, дизайн інтерфейсу користувача для платформи Android і опис сценаріїв взаємодії користувача з додатком;

– здійснити розробку додатку менеджера паролів, а саме: програмну реалізацію клієнтської частини додатку на Android, а також серверної частини;

– виконати функціональне та безпекове тестування розробленого менеджера паролів, виправлення помилок та оптимізацію продуктивності.

## Висновки до розділу 1

Було здійснено аналіз сучасного стану питання. Вивчення існуючих рішень показує, що сучасні менеджери паролів надають широкий спектр функцій для забезпечення безпеки та зручності користувачів.

Основними завданнями при розробці нового додатку повинні стати забезпечення високої безпеки даних, інтуїтивний інтерфейс та підтримка хмарної синхронізації. Використання новітніх алгоритмів шифрування та забезпечення зручного користування є ключовими факторами для успішного проекту.

# РОЗДІЛ 2
# СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

## 2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

На основі аналізу існуючих рішень для зберігання паролів визначені наступні вимоги до клієнт-серверного додатку для зберігання паролів.

Функціональні вимоги:

– реєстрація та автентифікація користувачів, підтримка майстер-пароля та біометричних методів автентифікації;

– паролі повинні зберігатися у зашифрованому вигляді у базі даних, до якої доступ має лише авторизований користувач;

– користувач має можливість створювати складні паролі з вибором типу (цифри, літери, спеціальні символи) та довжини пароля;

– автоматичне заповнення полів для входу на веб-сайти та у додатки;

– підтримка хмарної синхронізації паролів між різними пристроями користувача;

– можливість безпечного обміну паролями між користувачами.

Нефункціональні вимоги:

– використання сучасних алгоритмів шифрування (наприклад, AES-256) для забезпечення конфіденційності даних;

– додаток має бути швидким та не впливати на продуктивність пристроїв користувача;

– система повинна підтримувати збільшення кількості користувачів без втрати продуктивності;

– забезпечення безперебійної роботи сервера та швидкого відновлення після збоїв;

– інтуїтивно зрозумілий інтерфейс та позитивний користувацький досвід.

Технічні вимоги. Клієнтська частина являє собою мобільний додаток Android. Серверна частина передбачає використання надійного серверного рішення з підтримкою REST API для комунікації з клієнтом. Для зберігання зашифрованих паролів передбачається використання бази даних.

Клієнтський додаток створений для платформи Android з використанням мови Kotlin, яка є однією з найпоширеніших і найпідтримуваніших в Android-розробці. Далі розглянуто деталі архітектури та інструменти, що використовуються в реалізації програми.

Архітектура додатку розроблена відповідно до принципів SOLID і концепцій чистої архітектури, викладених Робертом Мартіном у його книзі «Clean Architecture: A Craftsman's Guide to Software Structure and Design» [11]. Ця архітектура добре зарекомендувала себе в мобільній розробці та застосовується в багатьох проєктах. SOLID – це акронім, що представляє п'ять принципів об'єктно-орієнтованого програмування, також сформульованих Робертом Мартіном:

– принцип єдиної відповідальності: кожен клас повинен виконувати лише одне конкретне завдання, що спрощує внесення змін у майбутньому, оскільки класи, які виконують кілька завдань, мають взаємозалежні частини, і зміни в одному завданні можуть вплинути на інші;

– принцип відкритості/закритості: програмні компоненти повинні бути відкритими для розширення, але закритими для модифікації. Це означає, що компоненти повинні розроблятися так, щоб їх можна було розширити, додаючи новий код, без зміни існуючого;

– принцип заміщення Ліскова: підкласи повинні бути замінниками своїх суперкласів, тобто підкласи мають коректно реалізовувати поведінку батьківських класів;

– принцип сегрегації інтерфейсів: інтерфейси повинні описувати мінімальний набір функціоналу, необхідний клієнту. Це знижує складність супроводу та розвитку додатку, оскільки прості інтерфейси легше реалізовувати та модифікувати;

– принцип інверсії залежностей: компоненти верхнього рівня не повинні залежати від компонентів нижнього рівня. Залежність повинна бути від абстракцій, а не від конкретних реалізацій. Абстракції не повинні залежати від деталей, а деталі повинні залежати від абстракцій.

Рисунок 2.1 ілюструє основну ідею чистої архітектури – нашарування та інверсію залежностей.

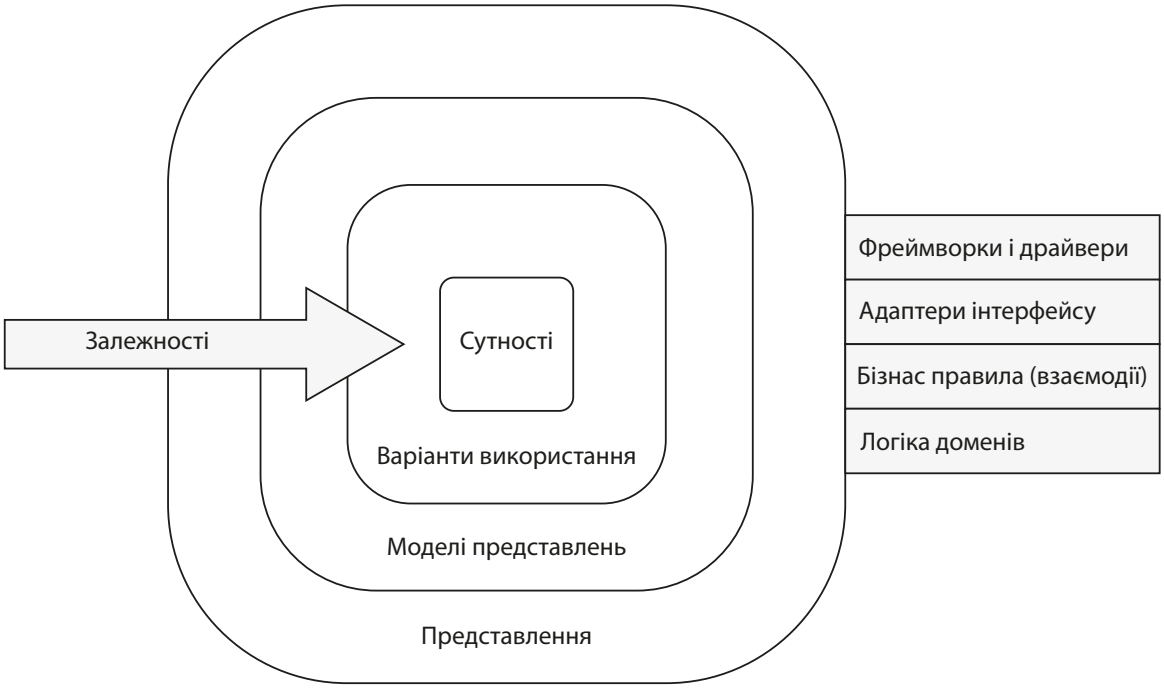


Рисунок 2.1 – Чиста архітектура мобільних додатків

Концентричні кола зображують різні рівні програми, кожен з яких виконує певні завдання. Зовнішній рівень представляє деталі реалізації, тоді як внутрішній охоплює бізнес-логіку та моделі. Тобто, чим ближче шар до центру, тим більше в ньому логіки високого рівня. Стрілка показує напрямок залежностей компонентів у архітектурі: зовнішні шари залежать лише від внутрішніх, що гарантує відсутність тісного зв’язку між компонентами. Це забезпечує тестованість коду та незалежність від користувацького інтерфейсу, фреймворків і зовнішніх сервісів.

В цілому, шари чистої архітектури в Android-додатку можна поділити на наступні:

– рівень даних: отримує та зберігає дані;

– доменний рівень: обробляє бізнес-логіку та дані;

– презентаційний рівень: відповідає за інтерфейс користувача та роботу з Android Framework.

Важливо, щоб кожен шар використовував окремі моделі даних для забезпечення незалежності між ними. Наприклад, якщо змінюється модель даних на сервері, це впливає лише на модель рівня даних, тоді як логіка домену та рівня представлення залишається незмінною, оскільки працює з власними моделями.

Кожен з трьох основних шарів має свої підрівні. У додатку реалізована структура компонентів (шарів), яка показана на рисунку 2.2., яка включає рівень даних, доменний рівень і презентаційний рівень.

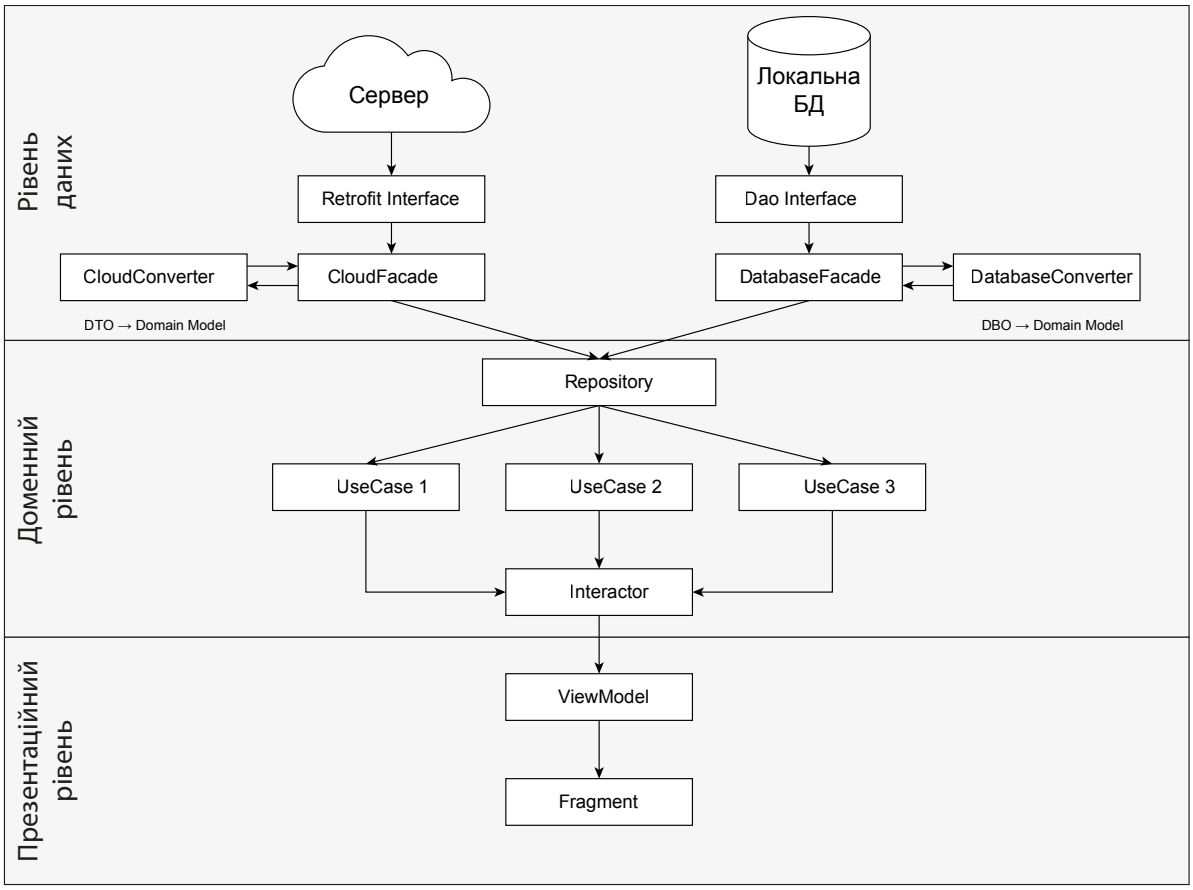


Рисунок 2.2 – Архітектура розроблюваного мобільного додатку менеджера паролів

Діаграма варіантів використання Android-додатку показана на рисунку 2.3.


Рисунок 2.3 – UML діаграма варіантів використання

На рисунку 2.4 показана UML-діаграма послідовності, що описує процес входу користувача в систему та перегляду паролів.

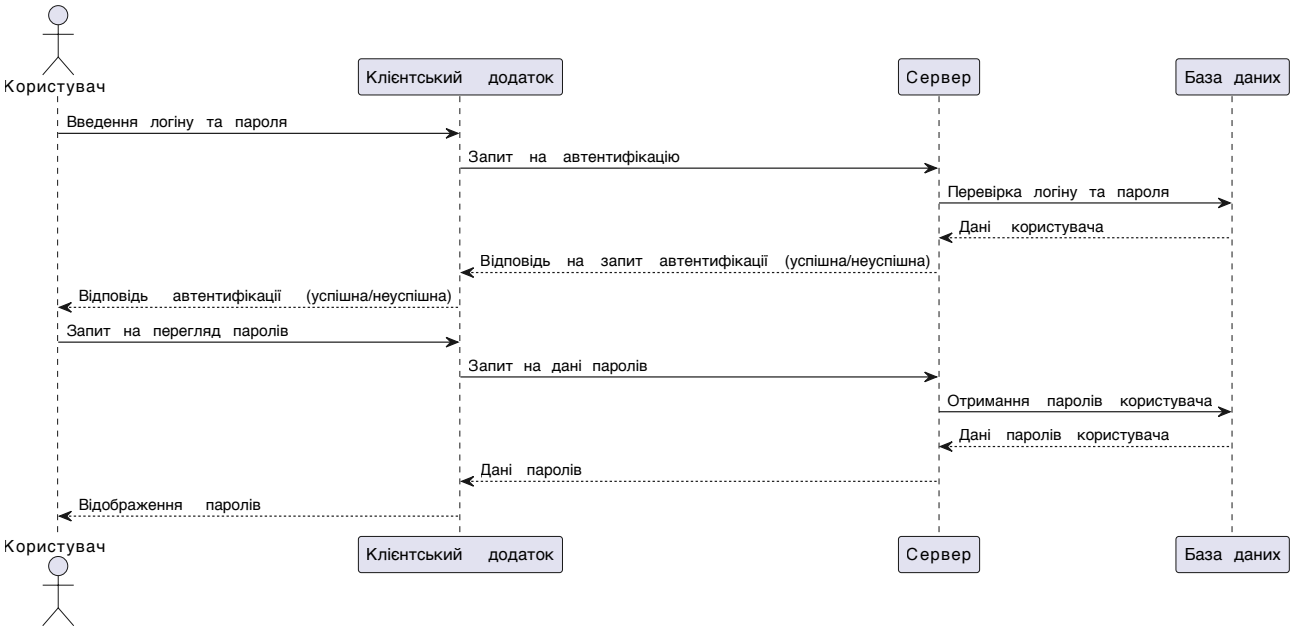


Рисунок 2.4 – UML діаграма послідовності

Користувач вводить логін та пароль у клієнтському додатку, який надсилає запит на автентифікацію до сервера. Сервер перевіряє логін і пароль у базі даних і повертає відповідь клієнтському додатку, який повідомляє користувача про успіх чи невдачу автентифікації. Якщо автентифікація успішна, користувач запитує перегляд паролів, клієнтський додаток надсилає запит на сервер, сервер отримує паролі з бази даних і повертає їх клієнтському додатку для відображення користувачу.

## 2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для криптографічних додатків використання передових рішень є критично важливим, оскільки вони виправляють відомі помилки та вразливості. Для забезпечення максимальної безпеки зберігання даних у Android-додатку планується використовувати систему Android KeyStore [12], з додаванням класів для взаємодії з KeyStore, які були введені у версії API 23.

Android Keystore дозволяє зберігати криптографічні ключі в системному контейнері, ускладнюючи їх вилучення з пристрою. Після створення ключа у сховищі, його можна використовувати для криптографічних операцій і не можна прочитати безпосередньо, оскільки ключові дані не передаються виконуваному процесу програми. Всі криптографічні операції делегуються системному процесу, що називається Keystore. Деякі нові пристрої з API 28 або вище можуть також мати апаратний модуль StrongBox Keymaster, який надає додаткові апаратні механізми для перевірки цілісності системи та авторизації процесів.

Необхідність використання мінімальної версії API 23 (Android 6.0), яка була випущена у 2015 році, обґрунтовується тим, що за даними Google, на даний момент 84,9 % пристроїв працюють під управлінням Android 6.0 і вище. Таким чином, додаток буде доступний для більшості користувачів Android без шкоди для безпеки.

KeyStore використовується для шифрування SharedPreferences, який є інструментом системи Android для зберігання пар ключ-значення у внутрішній пам’яті програми. Оскільки ключі не можна записати або прочитати у сховищі ключів, він використовується для операцій шифрування даних у SharedPreferences. Логіка роботи з KeyStore знаходиться в класі EncryptedSharedPreferences.

EncryptedSharedPreferences використовується у двох випадках.

Перший випадок: повторна авторизація користувача після одного успішного входу в систему. У цьому випадку ключ авторизації записується в EncryptedSharedPreferences, щоб подальші спроби входу не робили потенційно довгих запитів до мережі, а натомість миттєво порівнювали його зі збереженим ключем.

Другий випадок: вхід за відбитком пальця, у цьому випадку ключ шифрування даних і ключ авторизації зберігаються в EncryptedSharedPreferences для використання після успішної перевірки відбитком пальця.

Для доступу до особистих даних користувача використовується майстер-пароль, який залишається відомим лише йому. Цей пароль використовується для генерації симетричного 256-бітного ключа AES, який використовується для шифрування та розшифрування особистих даних на пристрої користувача. Алгоритм Advanced Encryption Standard (AES) використовується в режимі CBC з 256-бітним ключем і є визнаним стандартом криптографічної безпеки.

Незважаючи на стійкість AES, вгадати пароль можна за допомогою простого методу перебору. Для уникнення цього застосовується стандарт PBKDF2 [13], який використовує псевдовипадкову функцію на основі хешу (PRF), сіль (випадковий набір байтів) та багаторазові обчислення для створення похідного ключа.

Однак алгоритм Argon2 [14] може бути альтернативою для PBKDF2. Розроблений у 2015 році, Argon2 спеціально призначений для ефективного ускладнення перебору на спеціальних пристроях. Його параметри можуть бути налаштовані, включаючи обсяг використовуваної пам’яті, що дозволяє

обмежити продуктивність під час паралельних ітерацій. Реалізація і параметри Argon2 були взяті з відкритої реалізації захищеного месенджера Signal, надійність якого підтверджена численними дослідженнями.

Додавання солі в пароль зменшує можливість використання заздалегідь обчислених хешів (райдужних таблиць).

Розроблюване рішення забезпечує обмін даними між користувачами таким чином, що лише відправник і одержувач мають доступ до переданої інформації. Для реалізації цього використовується асиметричне шифрування. Під час реєстрації створюється пара унікальних ключів (публічний і приватний). Публічний ключ прив’язується до профілю користувача і передається на сервер, тоді як приватний ключ залишається на стороні клієнта. Користувачу не потрібно запам’ятовувати або записувати цей ключ, оскільки приватний ключ шифрується разом з основним сховищем і передається на сервер у зашифрованому вигляді. Генерація та обмін ключами здійснюється за допомогою протоколу еліптичної кривої Діффі-Хеллмана (ECDH) з використанням еліптичної кривої secp256r1, рекомендованої Групою стандартів ефективної криптографії (SECG) [15]. Ця крива підтримується стандартною реалізацією Android OpenSSL.

Рішення використовує протокол передачі даних HTTPS з підтримкою TLS 1.2, сучасний протокол з доведеною криптографічною стійкістю, який використовується більш, ніж на половині інтернет-ресурсів.

Процес підключення клієнта до сервера можна описати наступними кроками:

– клієнт і сервер обмінюються інформацією про підтримувані алгоритми і вибирають найкращий, в даному випадку це буде TLS 1.2 або вище;

– сервер надсилає свій сертифікат;

– клієнт отримує сертифікат сервера та перевіряє його через довірений центр сертифікації (ЦС);

– клієнт шифрує випадковий ключ публічним ключем сервера і відправляє його на сервер;

– сервер розшифровує отриманий ключ, після чого він використовується як симетричний ключ для шифрування всього каналу зв’язку;

Таким чином, якщо зловмисник спробує перехопити або змінити канал зв’язку, йому доведеться замінити сертифікат сервера, що відразу буде виявлено клієнтом, і з’єднання не буде встановлено.

Для розробки клієнтської частини менеджера паролів буде використовуватися Kotlin для Android [16].

Схема роботи з захищеними даними і ключами шифрування показана на рисунку 2.5.

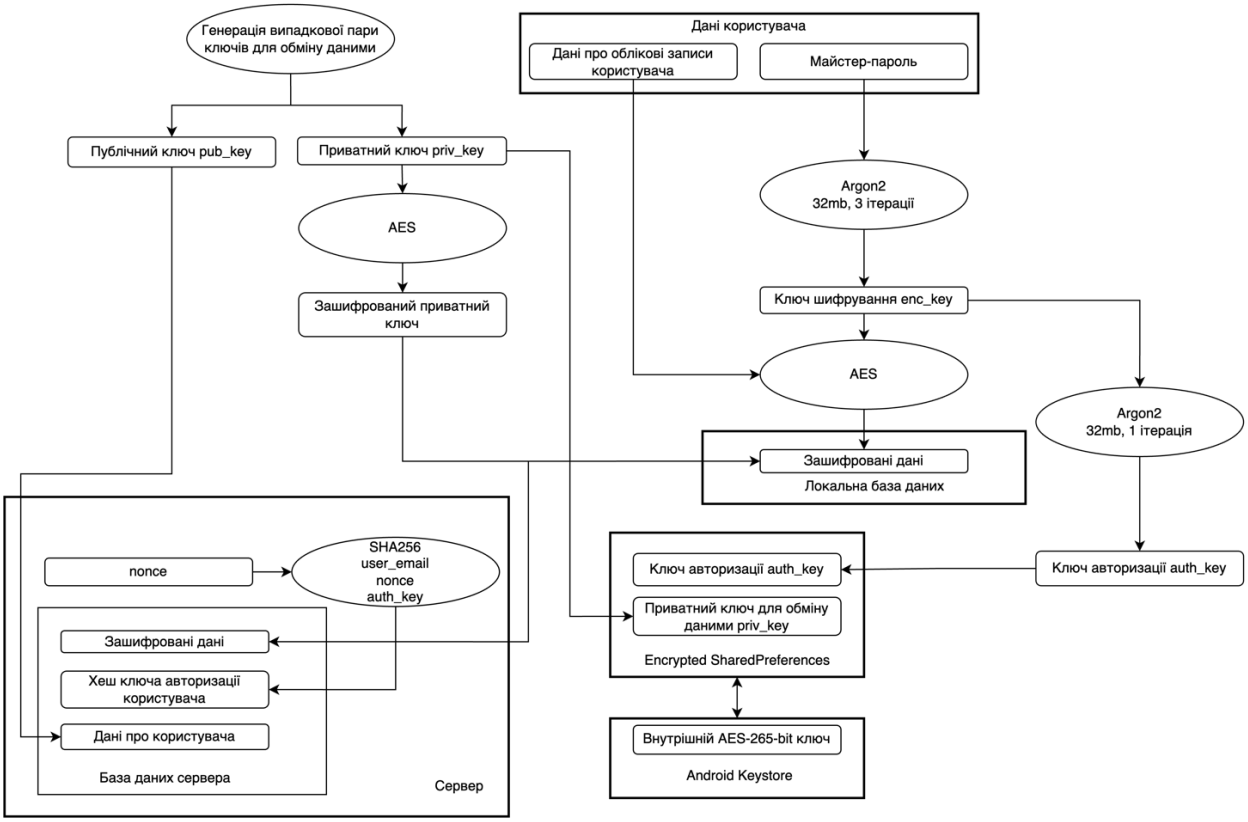


Рисунок 2.5 – Архітектура безпеки

Дані, які зберігаються на сервері, зашифровані майстер-паролями користувачів, відповідно навіть у разі отримання доступу до чужих даних, зловмисник не зможе їх використати. Проте, аутентифікація користувачів все одно необхідна, щоб кожен міг отримувати доступ тільки до власних даних.

У реалізованому рішенні використовується принцип Digest-авторизації. Ключ авторизації генерується з майстер-пароля разом із ключем шифрування бази даних шляхом додаткового обчислення ще одного раунду Argon2. Як майстер-пароль, так і ключ шифрування бази даних не можуть бути відновлені з отриманого ключа. Цей ключ авторизації передається на сервер при кожному запиті через захищений канал HTTPS.

Під час реєстрації цей ключ надсилається на сервер разом з іншими даними нового користувача. Сервер обчислює хеш ключа і логін користувача, після чого зберігає його в базі даних.

Обчислення хешу потрібне для того, щоб у разі несанкціонованого доступу до бази даних зловмисник не зміг видавати себе за користувача, оскільки для цього потрібен ключ авторизації, який не зберігається на сервері, і неможливо відновити його з хешу.

Хеш розраховується за таким принципом: SHA256(user_email:сіль:auth_key).

При зверненні до сервера клієнт відправляє ключ авторизації, а сервер перевіряє його, обчислюючи і порівнюючи хеш зі значенням, збереженим в базі даних під час реєстрації користувача.

Всі дані, що передаються між клієнтом і сервером, шифруються. Це означає, що передати їх у вигляді тексту не вийде, оскільки зашифровані дані являють собою масив байтів. Традиційні формати передачі даних, такі як JSON, використовувати не можна. Після аналізу можливих рішень було обрано протокол ProtoBuf [17], який широко підтримується на різних платформах і має оптимізовану реалізацію серіалізації та десеріалізації даних для Android.

У загальному випадку формат являє собою закодовану послідовність полів, що складається з ключа і значення.

Ключ – це число, визначене для кожного поля повідомлення у файлі «proto».

У лістингу 2.1 наведено приклад повідомлення користувача protobuf.

Лістинг 2.1 - Опис структури Користувача Protobuf

```
syntax = "proto3";
package aam.proto;
message Usr {
  string userid = 1;
  string username = 2;
  string emailaddres = 3;
  bytes publ = 4;
  bytes priv = 5;
}
```

Кінець лістингу 2.1

Далі класи генеруються за допомогою компілятора protobuf (protoc) для використання в програмному коді, описані у вихідному proto-файлі (ліст. 2.2).

Лістинг 2.2 – Код, згенерований Java Builder для структури користувача

```
UserDto.UserData.newBuilder()
  .setUserid(userDbo._id.toByteArray().toByteString())
  .setUsername(userDbo.username)
  .setEmailaddres(userDbo.emailaddress)
  .setPubl(userDbo.publ.toByteString())
  .setPriv(userDbo.priv.toByteString())
  .build();
```

Кінець лістингу 2.2

Це дозволяє ефективно передавати та обробляти зашифровані дані між клієнтом і сервером, забезпечуючи їх безпечність та цілісність.

Серверний додаток реалізовано на мові програмування Kotlin з використанням веб-фреймворку Ktor [18], а роль веб-сервера виконує Netty [19].

Для зберігання даних було обрано MongoDB [20], оскільки серверу необхідно зберігати байтові масиви із зашифрованими даними користувачів. Ці дані можуть включати файли, які займають декілька мегабайтів дискового простору. Хоча реляційні бази даних теж можуть впоратися з цим завданням, вони не оптимізовані для таких цілей. Тому було вирішено використовувати NOSQL базу даних MongoDB, яка підтримує безпосереднє зберігання двійкових даних та дозволяє зберігати до 16 мегабайтів в одному записі. Це дозволяє

зберігати зашифровані файли безпосередньо в базі даних. Замість таблиць, як у SQL базах даних, MongoDB використовує колекції документів (записів), які є структурами у форматі BSON (Binary JavaScript Object Notation). BSON розширює формат JSON, додаючи підтримку для різноманітних нових форматів даних, таких як двійкові дані, різні формати дат та нульові поля.

## Висновки до розділу 2

У цьому розділі визначено вимоги, архітектурні принципи та засоби для створення клієнт-серверного додатку для зберігання паролів. Функціональні вимоги включають реєстрацію та автентифікацію користувачів, зберігання зашифрованих паролів, генерацію складних паролів, автоматичне заповнення полів, хмарну синхронізацію, безпечний обмін паролями, організацію паролів і підтримку імпорту/експорту. Нефункціональні вимоги зосереджуються на використанні сучасних алгоритмів шифрування, швидкості роботи, масштабованості, безперебійній роботі сервера та інтуїтивно зрозумілому інтерфейсі.

Технічні вимоги передбачають розробку клієнтської частини на Android з використанням Kotlin і серверної частини з підтримкою REST API. Архітектура побудована на принципах SOLID і чистої архітектури, що забезпечує гнучкість і масштабованість, з розподілом на рівні даних, домену та презентації, кожен з яких використовує окремі моделі даних.

Для забезпечення безпеки використовується Android Keystore для зберігання криптографічних ключів, AES-256 для шифрування даних і PBKDF2 або Argon2 для ускладнення перебору паролів. Асиметричне шифрування застосовується для безпечного обміну паролями, а HTTPS з TLS 1.2 або вище – для захищеного каналу зв’язку.

Протокол ProtoBuf використовується для серіалізації та десеріалізації даних. Серверна частина реалізована на Kotlin з використанням Ktor і Netty, а

MongoDB обрано для ефективного зберігання двійкових даних. Це забезпечує високий рівень безпеки, масштабованість та зручність використання додатку.

# РОЗДІЛ 3
# РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ДОДАТКУ МЕНЕДЖЕРА ПАРОЛІВ

## 3.1 Практична реалізація об’єкта проектування

Практична реалізація передбачає розробку клієнтського додатку на Android, а також серверної частини, які повинні між собою взаємодіяти.

На рівні даних інкапсульована взаємодія із сервером і локальною базою даних. Мережева комунікація визначається в інтерфейсі PassmanRest, реалізація якого забезпечується через бібліотеку Retrofit. Для зберігання локальних даних використовується SQLite разом із ORM-бібліотекою Room, робота з базою даних описується через інтерфейси Dao (Data Access Object), які реалізуються за допомогою Room.

Особливістю даної реалізації є наявність додаткового фасадного шару (Facade), де моделі з рівня даних перетворюються в доменні моделі. CloudFacade використовується для перетворення серверних моделей (Data Transfer Object, DTO) в доменні моделі, а DatabaseFacade – для перетворення моделей бази даних (DBOs – DataBase Object). Це зазвичай відбувається в репозиторіях (Repository), але в цьому додатку було вирішено розділити на фасади, щоб зменшити обсяг бізнес-логіки в репозиторіях для роботи із зашифрованими даними, оскільки в них було надто багато коду. Репозиторії працюють з уже перетвореними доменними моделями, тому доцільніше розглядати їх як частину доменного рівня.

Доменний рівень включає Repository, UseCase та Interactors. Репозиторії відповідають за логіку роботи з даними, таку як шифрування, кешування та вибір джерел даних. Класи UseCase інкапсулюють бізнес-логіку і відповідають методам у репозиторіях. Використання класів сценаріїв дозволяє відокремити презентаційний шар від репозиторіїв, що полегшує внесення змін до репозиторіїв без зміни логіки презентації. Сценарії використання схожі на будівельні блоки, з якими працює презентаційний шар.

Оскільки рівень представлення, тобто інтерфейс користувача, є клієнтом юзкейсів, можна одразу вказати, які дані потрібно отримати або обробити, і зазначити, що для цього потрібно підписатися на UI flow. Це здійснюється за допомогою бібліотеки RxJava [21].

Інтерактор виступає контейнером для варіантів використання і визначає залежності компонентів представлення від відповідних кейсів використання. Наприклад, DataInteractor описує сценарії роботи з даними, реалізовані за допомогою необхідних кейсів використання (ліст. 3.1).

Лістинг 3.1 – Приклад інтерактора для роботи з обліковими записами

```
// Інтерфейс для взаємодії з даними
interface DataInteractor {
    fun fetchEntry(entryId: ObjectId): Single<Entry>
    fun fetchSharedEntry(entryId: ObjectId):
Single<Entry>
    fun storeEntry(entry: Entry): Completable
    fun removeEntry(entryId: ObjectId): Completable
    fun removeSharedEntry(entryId: ObjectId): Completable
    fun getSharedReceivers(originId: ObjectId):
Single<List<SharedReceiver>>
    fun verifyPassword(password: CharArray):
Single<String>
}
// Реалізація інтерфейсу з використанням кейсів
class DataInteractorImpl(
    private val retrieveEntryUseCase:
RetrieveEntryUseCase,
    private val retrieveSharedEntryUseCase:
RetrieveSharedEntryUseCase,
    private val storeEntryUseCase: StoreEntryUseCase,
    private val removeEntryUseCase: RemoveEntryUseCase,
    private val removeSharedEntryUseCase:
RemoveSharedEntryUseCase,
    private val fetchSharedReceiversUseCase:
FetchSharedReceiversUseCase,
    private val verifyPasswordUseCase:
VerifyPasswordUseCase
) : DataInteractor { // Реалізація методів інтерфейсу
}
```

Кінець лістингу 3.1

Додаток використовує реактивний підхід до обробки даних, де дані представляються у вигляді потоку, на який можна підписатися для відстеження. Для цього використовується бібліотека RxJava. Реактивний підхід зручний для роботи з асинхронно отриманими даними, обробки помилок та виконання різних операцій завдяки широкому набору операторів для реактивних типів.

RxJava включає п’ять основних типів, що є обгортками над даними. У додатку переважно використовується тип Single, який зберігає або дані, або помилку, що виникла під час завантаження. Операції видалення та збереження використовують тип Completable, який може бути успішним або містити помилку.

Звичайний асинхронний код включає багато функцій зворотного виклику. Для послідовного виконання асинхронних операцій доводиться писати зворотні виклики один всередині іншого. Наприклад, для завантаження, розшифрування, індексації даних користувача та їх запису в локальну базу даних.

За допомогою RxJava можна створити ланцюжок обробки потоків даних, що полегшує читання коду, розуміння його логіки і підвищує стабільність.

У лістингу 3.2 наведено приклад реактивного коду в додатку. У цьому методі список облікових записів завантажується асинхронно, фільтрується за типом запису, робляться запити на сторонній сервер, що вимагає не більше одного запиту в секунду, і в кінці дані об’єднуються в один загальний список. З використанням RxJava такий код стає більш компактним, простим у реалізації та надійним, оскільки всі помилки оброблюються всередині Rx-типів.

Лістинг 3.2 – Приклад реактивного коду

```
override fun breachesForAllAccounts():
Single<List<AccountBreach>> =
    getEntryHeadersUseCase.source() // Отримання даних
про облікові записи
        .flatMapObservable { Observable.fromIterable(it)
} // Перетворення у потік даних
        .filter { it.type == ENTRY_TYPE_ACCOUNT } //
Фільтрація за типом запису
```

```
        .map { it.details } // Отримання деталей
        .distinct() // Вилучення дублікатів
        .delay(1000, TimeUnit.MILLISECONDS, false) //
Затримка в мілісекундах
        .flatMapSingle {
            getBreachesForAccountUseCase.source {
PARAM_KEY_ACCOUNT of it } // Отримання порушень для
кожного облікового запису
                .onErrorReturnItem(emptyList) // Обробка
помилки
        }
        .flatMapIterable { it } // Розгортання списку
порушень
        .toList() // Конвертація у список
```

Кінець лістингу 3.2

Презентаційний рівень додатку реалізований відповідно до архітектури MVVM (Model – View – ViewModel), рекомендованої Google для розробки Android-додатків (ліст. 3.3).

Лістинг 3.3 – Приклад реалізації базового ViewModel

```
// Приклад реалізації базового ViewModel з життєвим
циклом
class BaseViewModel : ViewModel() {
    // Життєвий цикл ViewModel
    private val lifecycleEvents =
PublishSubject.create<Lifecycle.Event>()

    init {
        // Повідомлення про створення ViewModel
        lifecycleEvents.onNext(Lifecycle.Event.ON_CREATE)
    }

    override fun onCleared() {
        super.onCleared()
        // Повідомлення про знищення ViewModel

lifecycleEvents.onNext(Lifecycle.Event.ON_DESTROY)
    }
}
```

Кінець лістингу 3.3

ViewModel описує логіку взаємодії представлення з моделлю, тобто спостереження за змінами даних та реагування на них. У випадку Android-додатку, використання ViewModel дозволяє зберегти стан даних при зміні орієнтації екрану або інших подіях.

RxJava використовується для обробки подій кліків або введення тексту, що дозволяє підвищити стабільність додатку. Встановлення debounce на кнопку уникне неправильної обробки багаторазових натискань, що може виникнути при швидкому натисканні користувачем (ліст. 3.4).

Лістинг 3.4 – Використання RxJava для обробки натискань кнопок

```
buttonClicksObservable
    .throttleFirst(350, TimeUnit.MILLISECONDS
// debounce для уникнення багатократного натискання
    .subscribe { processButtonClick() }
```

Кінець лістингу 3.4

Таким чином, застосування RxJava у додатку дозволяє покращити його стабільність та відмовостійкість.

Додаток розроблено з урахуванням принципу інверсії залежностей. За цим підходом високорівневі компоненти не прив’язані до конкретних низькорівневих класів. Це спрощує заміну залежностей і сприяє тестуванню. Замість цього, залежність формується від абстракції – інтерфейсу, реалізованого залежністю. Залежності передаються через конструктор під час створення класу, що робить список необхідних компонентів очевидним. Управління залежностями і їх областю видимості виконується за допомогою бібліотеки DI (Dependency Injection) Koin (ліст. 3.5).

Лістинг 3.5 – Створення реєстру залежностей

```
open class BaseApp : Application {
    override fun onCreate() {
        super.onCreate()
        Timber.plant(Timber.DebugTree())
```

```
        startKoin {
            this: KoinApplication
            timberLogger(Level.DEBUG)
            androidContext(androidContext: this@BaseApp)
            modules(
                tasksModule, databaseModule, utilModule,
networkModule, viewModelModule, interactorModule,
facadeModule, repositoryModule, converterModule,
useCaseModule, prefModule
            )
        }
    }
}
```

Кінець лістингу 3.5

Модулі Koin – це списки BeanDefenition, які визначають залежність та її життєвий цикл. Наприклад, модуль для створення випадків використання показаний у лістингу 3.6.

Лістинг 3.6 – Модуль для створення випадків використання

```
val useCaseModule = module {
    factory { CreateUseCase(get()), get()) }
    factory { GetEntryHeadersUseCase(get(), get()) }
    factory { GetEntryUseCase(get(), get()) }
      // ...
}
```

Кінець лістингу 3.6

У додатку використовуються одиночні та фабричні BeanDefinition. Одиночні використовуються для створення синглтонів, тобто компоненти, які створюються один раз і використовуються всюди в додатку. Фабричні використовуються для створення нових екземплярів класу при кожному виклику, що дозволяє не тримати в пам’яті велику кількість компонентів одночасно, наприклад, сценарії використання.

Компоненти рівнів даних та домену представлені через інтерфейси, а не конкретні реалізації, що дозволяє на рівні BeanDefenition вказувати, яку

реалізацію надавати клієнтам. Наприклад, у розробці зручно використовувати прозору реалізацію класу, що відповідає за шифрування.

Навігація між екранами здійснюється за принципом Single Activity Application, де одна активність може включати кілька фрагментів. Активність відповідає за створення екранів і обробку дій користувача, тоді як фрагменти представляють вміст екранів.

Модулі Koin містять визначення для створення різних випадкових об’єктів.

Цей підхід до навігації принесе декілька переваг:

– загальна нижня панель AppBar анімується при переходах між фрагментами;

– більше можливостей для налаштування навігації і анімації;

– фрагменти займають менше пам’яті, що дозволяє швидше відображати екрани;

– можливість виконання лише одної дії на екрані і відображення декількох фрагментів одночасно.

Крім того, для зручної навігації між фрагментами використовується Navigation Framework. Він надає набір компонентів для керування переміщеннями між екранами, анімацією та передачею параметрів.

Navigation також інтегрується з ViewModel, дозволяючи створювати спільні ViewModel для фрагментів у графі. Це полегшує роботу з даними між різними фрагментами і покращує безпеку додатка.

Використання Navigation і Koin спрощує розробку додатка і робить його більш безпечним і ефективним.

У додатку існує алгоритм, який визначає надійність пароля. По-перше, він призначає LEN_WEIGHT одиниць кожному символу у паролі. Потім відбуваються різні перевірки, щоб виявити фактори, що можуть вплинути на надійність пароля. Кожна успішна перевірка призводить до віднімання або додавання певної кількості одиниць.

Наразі реалізовані такі перевірки:

– перевірка наявності великих літер. До загальної кількості додається UPPER_WEIGHT за кожен символ, що не є великою літерою;

– перевірка наявності символів нижнього регістру. Аналогічно до попередньої перевірки, але з LOWER_WEIGHT;

– перевірка наявності цифр. Аналогічно до попередньої перевірки, але з DIGIT_WEIGHT;

– перевірка наявності інших символів. Аналогічно до попередньої перевірки, але з SYMBOL_WEIGHT;

– перевірка, чи пароль складається тільки з малих літер;

– перевірка, чи пароль складається тільки з великих літер;

– перевірка, чи пароль складається тільки з цифр;

– перевірка максимальної довжини послідовності однакових символів;

– перевірка наявності пароля в списку 10 тис. найпопулярніших паролів.

Якщо оцінка пароля становить 300 і більше одиниць, пароль вважається надійним. Ця оцінка використовується в ProgressBar для відображення надійності пароля.

Додаток може функціонувати навіть без підключення до Інтернету. Ідея полягає в тому, що спочатку дані записуються в базу даних, а потім, коли з’являється доступ до мережі, вони автоматично відправляються на сервер. Для цього було прийнято декілька рішень щодо реалізації цієї функціональності.

Для ідентифікації кожного запису використовується 12-байтовий ідентифікатор об’єкта. Цей ідентифікатор складається з:

– 4 байтів часової позначки UNIX (часу в секундах з 1 січня 1970 року);

– 3 байтів ідентифікатора пристрою;

– 2 байтів ідентифікатора процесу;

– 3 байтів послідовного лічильника.

У додатку весь алгоритм формування цього ідентифікатора реалізовано у класі ObjectIdGenerator.

Використання такого ідентифікатора має декілька переваг:

– немає потреби в конвертації ідентифікаторів або зберіганні кількох різних для додатку і сервера. MongoDB, яка використовується як серверна база даних, використовує цей формат ідентифікаторів за замовчуванням;

– структура ObjectId розроблена з урахуванням можливості генерації унікальних ідентифікаторів на різних пристроях незалежно один від одного. Тому можна згенерувати ObjectId навіть на пристрої, який не має доступу до мережі, і впевнено використовувати його пізніше для синхронізації з сервером.

Для відправки даних на сервер та управління синхронізацією використовується бібліотека WorkManager. Ця бібліотека дає можливість створювати відкладені завдання, які будуть виконуватися при відповідних умовах, навіть після перезавантаження пристрою. У нашому випадку основною умовою запуску завдання є наявність Інтернет-з’єднання.

Для уникнення конфліктів даних під час синхронізації, клас OperationManager використовує чергу завдань синхронізації. Він перевіряє необхідність додавання нового завдання на основі списку завдань, які вже були додані. Для зберігання цього списку використовується локальна база даних.

Після того, як Інтернет-з’єднання стає доступним, WorkManager запускає SyncWorker, який отримує список завдань з OperationManager і відправляє їх на сервер. При успішній синхронізації дані видаляються з локальної бази даних.

Цей підхід дозволяє ефективно керувати синхронізацією даних між додатком і сервером, забезпечуючи їхню надійність та унікальність.

Основною складністю при розробці інтерфейсу було створення екрану редагування запису (рис. 3.1). Цей екран повинен бути гнучким і підтримувати будь-який набір полів різних типів у записі, з можливістю додавання, видалення та зміни їх. Для досягнення цього, екран динамічно формується на основі даних з EntryData.

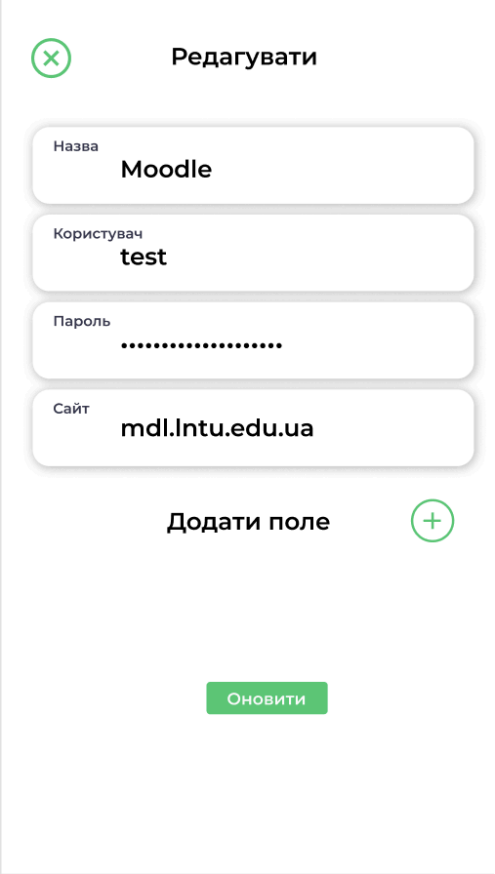


Рисунок 3.1 – Екран редагування запису

Однак загальна реалізація списку (RecyclerView) передбачає однорідний тип розмітки елементів. У нашому випадку потрібно підтримувати більше, ніж п’ять різних типів елементів у списку, кожен з яких має власну логіку кліків та введення даних. Для цього ми використовуємо бібліотеку Groupie, яка пропонує альтернативний підхід до оформлення списків.

Groupie дозволяє створювати самостійні блоки (Items), що містять інформацію про розмітку та логіку кожного елемента. Таким чином, кожен елемент у списку представлений своїм класом-нащадком класу Item, де перевизначаються методи, що повертають ID розмітки елемента та заповнення цієї розмітки.

За допомогою такого підходу значно спрощується реалізація навіть складних елементів з багатьма текстовими полями та кнопками. Кожен елемент має свій власний клас, що успадковується від класу Item, і в ньому визначаються методи для ID розмітки елемента і його заповнення.

### 3.2 Розробка бази даних

Програма використовує SQLite для локального зберігання даних користувачів через бібліотеку Room, частина Android Architecture Components від Google. Room дозволяє описувати операції з базою даних у інтерфейсі DAO (Data Access Objects), а його реалізація генерується автоматично.

Також, в базі даних додатка зберігаються ще 5 таблиць (рис. 3.2):

– EncryptedEntryDbo, яка містить записи користувачів, дані яких шифруються;

– EntryTagRef, яка відображає зв'язки між записами та тегами;

– EncryptedSharedEntryDbo, яка містить інформацію про спільні записи, надані або надіслані іншим користувачам;

– UserDbo, що містить інформацію про користувачів, які отримали або надали доступ до записів;

– EntryHeaderDbo, що містить індексовану інформацію про записи користувача для швидкого відображення та пошуку.

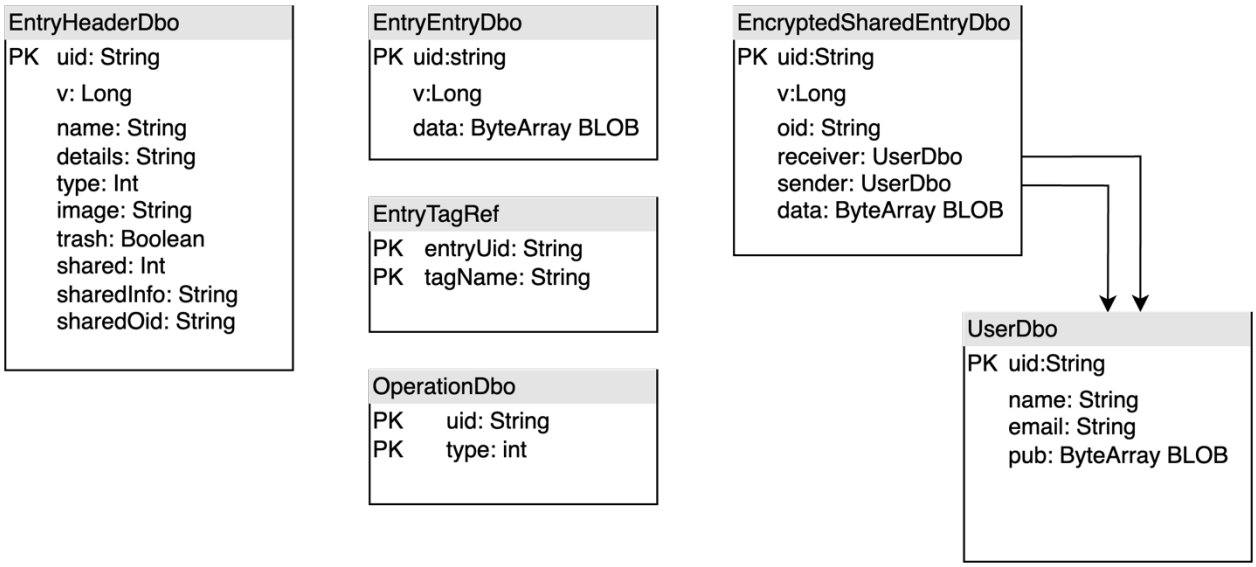


Рисунок 3.2 – Таблиці бази даних

Ці дані індексуються під час завантаження нових записів або після зміни на пристрої. Під час індексації дані розшифровуються, індекси записуються у базу даних разом із зашифрованими записами, що дозволяє миттєво показувати список записів користувача та швидко їх фільтрувати.

Дані в розшифрованому вигляді представлені класами EntryData та EntryField. EntryData містить заголовок, тип, список тегів, прапорець про наявність у кошику, а також список полів EntryField. Поля EntryField містять інформацію про ідентифікатор, назву, тип і вміст поля (рис. 3.3).

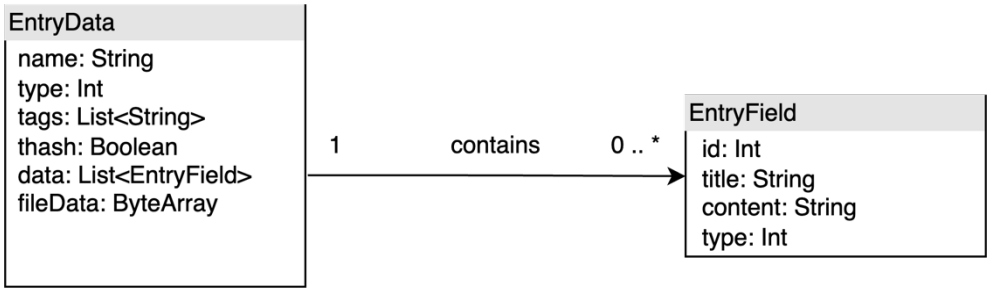


Рисунок 3.3 – Таблиці даних в розшифрованому вигляді

Для серіалізації класу EntryData використовується JSON, але масив байтів fileData зберігається окремо через його несумісність із JSON. Під час десеріалізації спершу зчитується розмір серіалізованих даних, після чого вони десеріалізуються, а залишок байтів, якщо є, записується в поле fileData.

### 3.3 Реалізація взаємодії з мережею

Робота з мережею здійснюється через бібліотеку Retrofit [22]. Її особливість полягає у тому, що методи API описуються безпосередньо у інтерфейсі з додатковою деталізацією запиту в анотаціях (ліст. 3.7).

Лістинг 3.7 – Взаємодія з мережею за допомогою Retrofit

```
interface PassmanRest {
    // Реєстрація нового користувача
```

```
        @POST("user/reg")
        Fun   register(@Body   userDto:   UserDto.User):
Completable
        // Авторизація користувача за електронною поштою
та ключем авторизації
        @GET("user/auth")
        fun    auth(@Query("email")    email:    String,
@Query("auth") auth: String): Single<UserDto.UserData>
        // Отримання публічного ключа користувача за його
електронною поштою
        @GET("user/pub")
        fun     getUserPubData(@Query("email")     email:
String): Single<UserDto.UserDataPub>
        // Збереження нових даних
        @POST("data")
        fun            saveData(@Body            dataDto:
EncDataDto.EntryList): Completable
        // ...
    }
```

Кінець лістингу 3.7

Реалізація цього інтерфейсу створюється автоматично під час виконання завдяки механізму відображення в Java, зокрема за допомогою Java Proxy. Запити конвертуються та відправляються за допомогою бібліотеки OkHttp, яка є стандартом в розробці під Android.

### 3.4 Розробка серверного додатку

Серверний додаток реалізує програму на мові Kotlin за допомогою веб-фреймворку Ktor, що працює на сервері Netty. Для зберігання даних вибрано базу даних MongoDB, оскільки головним завданням сервера є зберігання зашифрованих байтових масивів даних користувачів, які можуть включати файли розміром кілька мегабайт. MongoDB відмінно підходить для цієї мети, оскільки вона підтримує негайне зберігання двійкових даних, здатних зберігати до 16 мегабайт в одному записі.

Основна відмінність MongoDB полягає в тому, що вона використовує колекції документів замість таблиць, як у реляційних базах даних. Документи в MongoDB представлені у форматі BSON (Binary JavaScript Object Notation), яке є розширенням формату JSON, що додає підтримку великої кількості нових форматів даних, таких як двійкові дані, різні формати дат, нульові поля.

У базі даних існує п’ять колекцій:

– data: зберігає зашифровані дані користувача разом з ідентифікатором власника та часом оновлення у форматі UNIX Timestamp;

– data_removal: містить записи, які були видалені з колекції даних. Це потрібно для сповіщення клієнтських програм про видалення даних на сервері;

– shared: містить інформацію про дані, які пересилаються між користувачами, включаючи ID вихідного запису, ID відправника та ID одержувача;

– shared_removal: аналогічно data_removal, але для спільного доступу, де до shared_removal додаються два записи для відправника та одержувача;

– user: містить дані користувача, такі як публічний ключ, закритий ключ, хеш ключа авторизації та випадкова сіль для забезпечення безпеки (рис. 3.4).

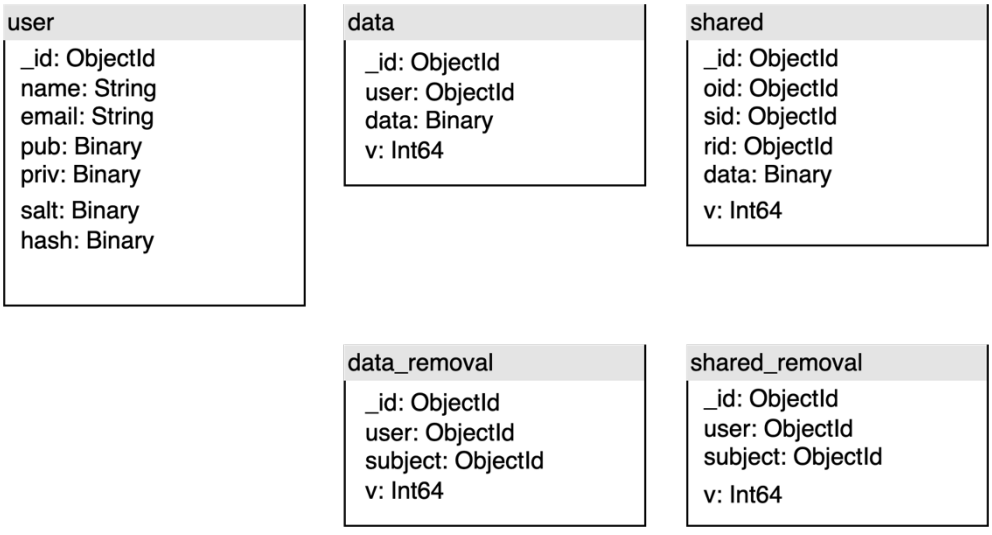


Рисунок 3.4 – Структура даних в базі даних

Архітектура серверного додатку складається з трьох рівнів:

– репозиторій: це модуль, що відповідає за взаємодію з базою даних. Він використовує MongoCollection з Java MongoDB Driver для доступу до колекцій MongoDB;

– сервіс: він залежить від репозиторію і конвертерів. Його основна функція - конвертувати дані між форматами MongoDB документів та DTO (Data Transfer Object) у форматі ProtoBuf, що передаються від та до клієнта;

– маршрут: це модуль, що містить описи API-методів. Він приймає запити від клієнтів, перевіряє їх валідність та авторизацію, а потім передає їх до відповідного сервісу для обробки (рис. 3.5).

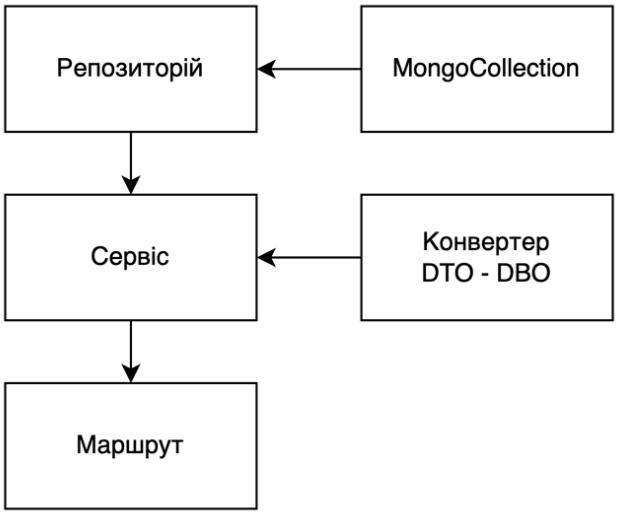


Рисунок 3.5 – Архітектура серверних додатків.

Для управління залежностями використовується бібліотека Koin DI, яка має розширення для інтеграції з Ktor, аналогічно до клієнтського додатку.

Клієнтські додатки взаємодіють з API сервера через HTTPS, що додатково шифрує дані для захисту від можливої підміни. SSL-сертифікат, необхідний для шифрування, знаходиться в ресурсах серверної програми.

Сервер має 3 маршрути для запитів, пов’язаних з даними користувача, звичайними записами та даними, що передаються між користувачами.

Методи API були розроблені для оптимізації фонової синхронізації на клієнтському пристрої. Запити DELETE і POST підтримують передачу списків об’єктів або ідентифікаторів, що дозволяє ефективно обробляти декілька змін.

Для авторизації використовується HTTP Header Token, що містить ідентифікатор користувача та ключ авторизації. Сервер перевіряє та авторизує користувача на основі цих даних.

### 3.5 Тестування та налагодження інформаційно-комп’ютерної системи

На рисунку 3.6 показано вигляд мобільного додатку, після запуску якого потрібно пройти автентифікацію. Якщо автентифікація успішна, то доступний список збережених паролів з можливістю додавання нового запису або редагування існуючих.

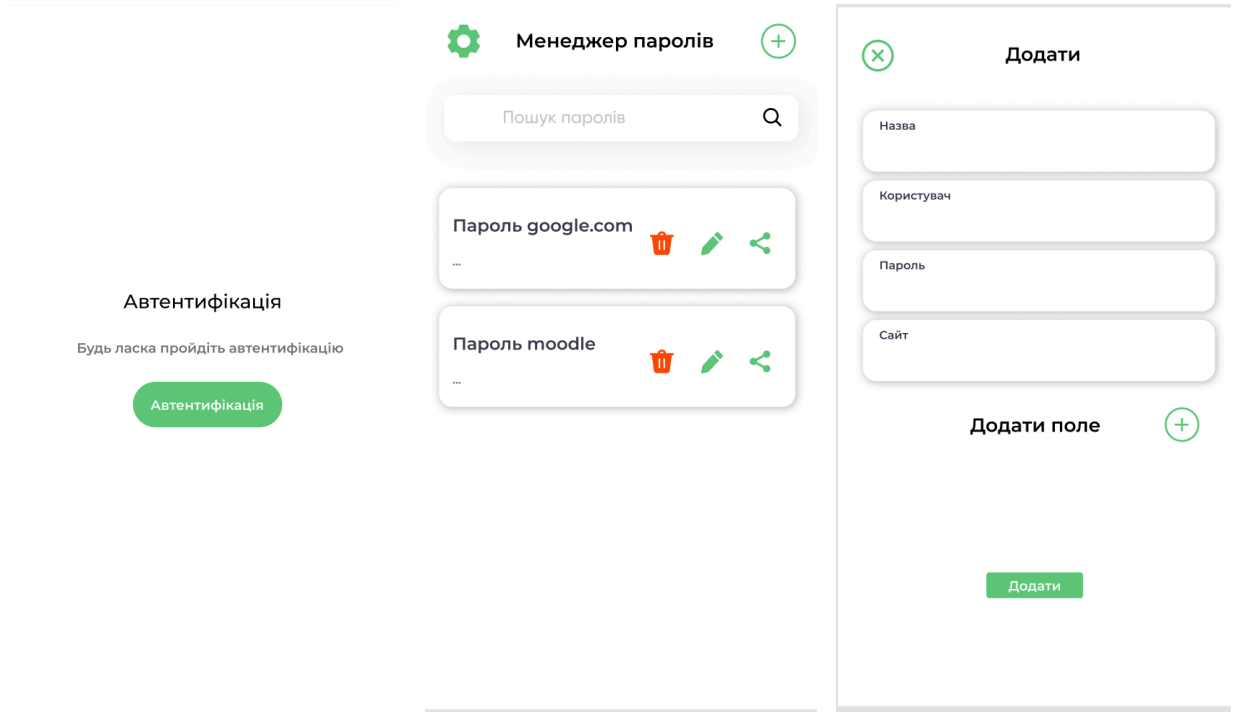


Рисунок 3.6 – Вигляд мобільного додатку менеджера паролів

Важливим етапом життєвого циклу є тестування. Тестування клієнт-серверного додатку для зберігання паролів за допомогою Kotlin включає кілька кроків, кожен з яких спрямований на забезпечення високої якості та надійності

продукту. Кроки, які можна застосувати до нашого проекту – це планування тестування, розробка тестових сценаріїв, створення тестових випадків, виконання тестування і аналіз результатів.

Проводилось функціональне, інтеграційне, системне тестування, тестування безпеки і продуктивності.

Перевірка безпеки передбачала перевірку надійності шифрування, захисту даних, автентифікації та авторизації користувачів.

Тестування продуктивності включала оцінку швидкості роботи додатку під різним навантаженням, перевірку ефективності.

Для автоматизації тестування використовувався інструмент JUnit – для тестування на Kotlin, Selenium – для тестування веб-інтерфейсів, а також Postman – для API тестування.

У лістингу 3.8 показано приклад автоматизації тестового сценарію на Kotlin.

Лістинг 3.8 – Автоматизація тестового сценарію

```
import org.junit.Test
import org.junit.Assert.*
import org.mockito.Mockito.*
class UserRegistrationTest {
    @Test
    fun testUserRegistrationAndPasswordSaving() {
// Встановлення: налаштування мокових залежностей та початкового стану
        val mockServer = mock(Server::class.java)
        val clientApp = ClientApp(mockServer)
        val testEmail = "testuser@example.com"
        val testPassword = "StrongPassword!123"
        val testService = "Example Service"
        val testServiceUser = "example_user"
        val testServicePassword = "ExamplePassword123"
// Дія: виконання кроків реєстрації та збереження пароля
        clientApp.openRegistrationScreen() // Відкрити екран реєстрації
        clientApp.registerUser("Тестовий Користувач", testEmail, testPassword, testPassword) // Реєстрація нового користувача
```

```
        // Перевірка: перевірка успішної реєстрації
        verify(mockServer).registerUser(testEmail,
testPassword) // Перевірка виклику методу реєстрації на
сервері
        assertTrue(clientApp.isRegistrationSuccessful())
// Переконатися, що реєстрація успішна
        // Дія: вхід у систему
        clientApp.login(testEmail, testPassword) // Вхід
користувача
        // Перевірка: перевірка успішного входу
        verify(mockServer).authenticateUser(testEmail,
testPassword) // Перевірка виклику методу автентифікації
на сервері
        assertTrue(clientApp.isLoginSuccessful()) //
Переконатися, що вхід успішний
        // Дія: збереження нового пароля
        clientApp.savePassword(testService,
testServiceUser, testServicePassword) // Збереження
нового пароля
        // Перевірка: перевірка, що пароль збережений
правильно
        verify(mockServer).savePassword(testService,
testServiceUser, testServicePassword) // Перевірка
виклику методу збереження пароля на сервері
        val savedPassword =
clientApp.getPassword(testService) // Отримання
збереженого пароля
        assertEquals(testServiceUser,
savedPassword.username) // Переконатися, що ім'я
користувача правильне
        assertEquals(testServicePassword,
savedPassword.password) // Переконатися, що пароль
правильний
    }
}
```

Кінець лістингу 3.8

Цей сценарій охоплює основні функціональні можливості додатку: реєстрація нового користувача, вхід до системи, збереження та перевірка збереження паролів. Автоматизація тестування на Kotlin дозволяє швидко

виконувати ці перевірки та виявляти можливі помилки на ранніх стадіях розробки.

### Висновки до розділу 3

Описано процес розробки клієнт-серверного додатку для управління паролями, включаючи створення клієнтського додатку для Android та серверної частини. Зосереджено увагу на розробці різних рівнів додатку, включаючи рівні даних, доменний та презентаційний. Основна робота з мережею здійснюється через інтерфейс PassmanRest з використанням бібліотеки Retrofit, а для зберігання даних локально використовується SQLite з ORM-бібліотекою Room. Особливість реалізації полягає у впровадженні фасадного шару для перетворення моделей між різними рівнями, що спрощує бізнес-логіку в репозиторіях.

Доменний рівень включає класи Repository, UseCase та Interactors, які відповідають за логіку роботи з даними, шифрування, кешування та вибір джерел даних. Презентаційний рівень реалізовано за архітектурою MVVM, що дозволяє розділити логіку взаємодії між представленням і моделлю, забезпечуючи стабільність і відмовостійкість додатку за допомогою реактивного підходу, реалізованого через RxJava.

У розробці бази даних використано SQLite для локального зберігання з допомогою бібліотеки Room, де дані представлені у вигляді таблиць та індексуються для швидкого доступу та пошуку. Взаємодія з мережею здійснюється через бібліотеку Retrofit, що забезпечує зручність опису методів API та їх реалізацію під час виконання.

Серверний додаток реалізовано на мові Kotlin з використанням фреймворку Ktor та бази даних MongoDB, яка зберігає зашифровані дані користувачів. Архітектура серверного додатку складається з трьох рівнів: Репозиторій, Сервіс та Маршрутизатор, що відповідають за взаємодію з базою даних, конвертацію даних та обробку запитів клієнтів відповідно.

Розробка додатку здійснена з урахуванням принципу інверсії залежностей, що забезпечує модульність та спрощує тестування. Для цього використано бібліотеку Koin для управління залежностями як у клієнтській, так і в серверній частині. Навігація в клієнтському додатку реалізована за принципом Single Activity Application з використанням Navigation Framework, що забезпечує гнучкість та зручність користування додатком.

Проведено тестування розробленого менеджера паролів. Для автоматизації тестування використовувався інструмент JUnit – для тестування на Kotlin, Selenium – для тестування веб-інтерфейсів, а також Postman – для API тестування.

# ВИСНОВКИ

Було здійснено детальний аналіз сучасних менеджерів паролів, таких як LastPass, 1Password, Bitwarden, Password Safe and Manager, та Dashlane. Виявлено їхні основні функціональні можливості, переваги та недоліки. Результати аналізу показали, що основними функціями таких додатків є генерація паролів, двофакторна автентифікація, автоматичне заповнення полів, хмарна синхронізація даних, резервне копіювання, підтримка біометрії та зберігання захищених нотаток.

Визначено основні вимоги до клієнт-серверного додатку, який включає функціональні вимоги (реєстрація та автентифікація користувачів, генерація паролів, автоматичне заповнення полів, хмарна синхронізація, безпечний обмін паролями, організація паролів) та нефункціональні вимоги (використання сучасних алгоритмів шифрування, висока продуктивність, масштабованість, безперебійна робота сервера та інтуїтивно зрозумілий інтерфейс).

Проведено вивчення та вибір криптографічних алгоритмів для забезпечення безпеки даних, таких як AES-256 для шифрування, PBKDF2 та Argon2 для ускладнення перебору паролів, а також асиметричне шифрування для безпечного обміну паролями. Для передачі даних між клієнтом і сервером використовується протокол HTTPS з підтримкою TLS 1.2.

Розроблено архітектуру клієнт-серверного додатку, яка базується на принципах SOLID та чистої архітектури. Використано мову програмування Kotlin для реалізації як клієнтської, так і серверної частини додатку. Для зберігання даних обрано MongoDB, що забезпечує ефективне зберігання двійкових даних. Використано бібліотеки Ktor та Netty для реалізації веб-сервера.

Здійснено проєктування інтерфейсу користувача для платформи Android з використанням Kotlin. Реалізовано дизайн інтерфейсу та опрацьовано сценарії взаємодії користувача з додатком, забезпечуючи інтуїтивно зрозумілий та зручний досвід користувача.

Виконано програмну реалізацію клієнтської частини додатку на Android та серверної частини, а також серверної частини з підтримкою REST API. Додаток забезпечує всі необхідні функції для управління паролями, їх зберігання та безпечного обміну між користувачами.

Проведено функціональне та безпекове тестування розробленого менеджера паролів. Виправлено виявлені помилки та здійснено оптимізацію продуктивності додатку, що дозволило досягти високого рівня надійності та безпеки.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. HaveIBeenPwned. URL: https://haveibeenpwned.com/ (дата звернення: 07.02.2024).

2. Nordpass. Top 200 Most Common Passwords. URL: https://nordpass.com/most-common-passwords-list/ (дата звернення: 07.02.2024).

3. Android. URL: https://www.android.com/ (дата звернення: 09.02.2024).

4. Google Play. URL: https://play.google.com/store/apps (дата звернення: 07.02.2024).

5. Password Safe and Manager. All your data encrypted in one place!URL: https://passwordsafe.app/#play (дата звернення: 14.02.2024).

6. LastPass. A password manager for everyone, everywhere. URL: https://www.lastpass.com/ (дата звернення: 23.02.2024).

7. BitWarden. The trusted password manager for you and your business. URL: https://bitwarden.com/ (дата звернення: 02.03.2024).

8. Dashlane. URL: https://www.dashlane.com/ (дата звернення: 02.03.2024).

9. 1Password. More than a password manager. URL: https://1password.com/ (дата звернення: 17.03.2024).

10. Austin, J. Our favorite password manager remembers all of your logins so you don’t have to. Wirecutter: Reviews for the Real World. URL: https://www.nytimes.com/wirecutter/reviews/1password-review/ (дата звернення: 17.03.2024).

11. Martin, R. C. Clean architecture: A Craftsman’s Guide to Software Structure and Design. Pearson Professional. 352 p. 2020.

12. Android Keystore System. URL: https://developer.android.com/training/articles/keystore (дата звернення: 19.03.2024).

13. PBKDF2. URL: https://uk.wikipedia.org/wiki/PBKDF2 (дата звернення: 19.03.2024).

14. Argon2. URL: https://uk.wikipedia.org/wiki/Argon2 (дата звернення: 25.03.2024).

15. Standards for Efficient Cryptography Group. URL: https://www.secg.org/SEC2-Ver-2.0.pdf (дата звернення: 26.03.2024).

16. https://kotlinlang.org/docs/android-overview.html (дата звернення: 02.04.2024).

17. Protocol Buffers Google Developers. URL: https://developers.google.com/protocol-buffers (дата звернення: 12.04.2024).

18. Ktor – Connected Applications with Kotlin. URL: https://ktor.io/ (дата звернення: 06.05.2024).

19. Netty Project. URL: https://netty.io/ (дата звернення: 09.05.2024).

20. MongoDB – The database for modern applications. URL: https://www.mongodb.com (дата звернення: 12.05.2024).

21. RxJava – Reactive Extensions for the JVM. URL: https://github.com/ReactiveX/RxJava (дата звернення: 14.05.2024).

22. Retrofit. A type-safe HTTP client for Android and Java. URL: https://square.github.io/retrofit/ (дата звернення: 14.05.2024).