Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____ Іван ПАРХОМЕНКО
«____» червня 2023 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)
спеціальність 125 Кібербезпека
(код і назва спеціальності)
освітній ступень бакалавр
освітня програма Кібербезпека
(назва освітньо-професійної програми)
на тему: Методи автентифікації користувачів у мобільних додатках

Виконавець: студент IV курсу, групи КБ-43

_____ **Богдан ОЛІЙНИК**
(підпис) (ім'я, прізвище)

	Ім'я, прізвище	**Підпис**
Керівник	Інна МИХАЛЬЧУК	

Нормоконтроль	Андрій БІГДАН	

Київ 2023

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Сергій ТОЛЮПА
«24» жовтня 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-43** (група) **Олійнику Богдану Володимировичу** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Методи автентифікації користувачів у мобільних додатках

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №3 від 20.10.2022 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Сучасні методи автентифікації у мобільних додатках

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно ознайомитись із сучасними методами автентифікації та їх типовими алгоритмами, вразливостями з боку безпеки даних, розробити рекомендації щодо застосування цих методів у мобільних додатках

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Застосування наданих рекомендацій при проектуванні мобільних додатків значно підвищить рівень безпеки та зручність користування

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 24 жовтня 2022 року

Завдання видала _____ Інна МИХАЛЬЧУК
(підпис) (ім'я, прізвище)

Завдання прийняв до виконання _____ Богдан ОЛІЙНИК
(підпис) (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	24.10.2022 – 03.11.2022	*виконано*
2	Аналіз літератури	04.11.2022 – 30.11.2022	*виконано*
3	Обґрунтування вибору методів дослідження	01.12.2022 – 22.12.2022	*виконано*
4	Концепція застосовування алгоритмів автентифікації	23.12.2022 – 10.01.2023	*виконано*
5	Аналіз проблем інформаційної безпеки при автентифікації в мобільних додатках	11.01.2023 – 21.02.2023	*виконано*
6	Дослідження методів автентифікації та рівня безпеки	22.02.2023 – 30.03.2023	*виконано*
7	Вироблення рекомендацій щодо вибору методу автентифікації, що використовуються в мобільних додатках	31.03.2023 – 15.05.2023	*виконано*
8	Оформлення пояснювальної записки	16.05.2023 – 24.05.2023	*виконано*
9	Підготовка до захисту кваліфікаційної роботи	25.05.2023 – 12.06.2023	*виконано*

Завдання видала _____ Інна МИХАЛЬЧУК
(підпис) (ім'я, прізвище)

Завдання прийняв до виконання _____ Богдан ОЛІЙНИК
(підпис) (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 12 червня 2023 року

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел, додатків, має 55 сторінок основного тексту та 3 таблиці. Список використаних джерел містить 36 найменувань і займає 3 сторінки.

Метою роботи є розробка рекомендацій щодо використання сучасних методів автентифікації у мобільних додатках.

Об'єктом дослідження є процес автентифікації користувачів у мобільних додатках.

Предметом дослідження є сучасні технології автентифікації користувачів у мобільних додатках з урахуванням вимог безпеки.

Методи дослідження кваліфікаційної роботи:

- аналіз наукової літератури та відкритих джерел;
- експериментальні дослідження;
- компаративний аналіз.

Практичною цінністю є розробка рекомендацій до застосування розглянутих в даній роботі методів автентифікації у мобільних додатках з урахуванням рівня безпеки та зручності користування додатком.

Ключові слова: мобільні додатки, автентифікація користувачів, методи автентифікації, безпека даних, багатофакторна автентифікація, Android-застосунок, приватність, кібербезпека.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

BaaS	–	Backend-as-a-Service
NSD	–	Network Service Discovery
QR	–	Quick Response Code
2FA	–	Two-Factor Authentication
ПЗ	–	Програмне забезпечення
IDE	–	Integrated Development Environment
MVVM	–	Model-View-ViewModel
PIN	–	Personal Identification Numbers
API	–	Application Programming Interface
UI	–	User Interface
SSL	–	Content delivery network
CDN	–	Secure Sockets Layer

ВСТУП

Від початку XXI століття, людство спостерігало значний прогрес у розвитку інформаційних технологій, особливо у сфері мобільних пристроїв. Завдяки широкому поширенню смартфонів, сьогодні виникає необхідність ретельного вивчення методів автентифікації користувачів у мобільних додатках, що враховують вимоги до безпеки. За останні десятиліття, кількість користувачів мобільних пристроїв стрімко зросла, підкреслюючи актуальність цієї проблеми.

Це свідчить про те, що все більше людей починають володіти різними пристроями, зокрема смартфонами, які стають все доступнішими для широких верств населення. Така ситуація вимагає ретельного дослідження методів автентифікації користувачів у мобільних додатках, які забезпечують надійний захист від несанкціонованого доступу та відповідають вимогам безпеки.

Завдяки стрімкому росту користувачів мобільних застосунків та все більшому застосуванню смартфонів у повсякденному житті, важливість використання надійних методів автентифікації у мобільних додатках набуває нових обертів. Забезпечення безпеки особистої інформації та захист від несанкціонованого доступу відіграють ключову роль у сучасному цифровому світі.

Тож **актуальність роботи** полягає в тому, щоб розробити рекомендації щодо застосування методів автентифікації у системах з урахуванням в останніх вимог до безпеки для забезпечення як стійкості мобільних сервісів та захисту особистої інформації, так і забезпечення зручності, що може призвести до зростання довіри користувачів і стійкого розвитку галузі.

Метою роботи є розробка рекомендацій щодо використання сучасних методів автентифікації у мобільних додатках.

Для досягнення зазначеної мети кваліфікаційної роботи поставлено наступні **завдання**:

- проаналізувати літературу та джерела, що стосуються сучасних методів автентифікації користувачів у мобільних додатках, зокрема стандарти, дослідження та приклади з реального життя;
- вивчити сучасні підходи та технології, які використовуються для автентифікації користувачів в мобільних додатках, з акцентом на їх надійність та зручність для користувачів;
- створити Android-застосунок, який включатиме декілька різних методів автентифікації для порівняльного аналізу та випробувань їхньої ефективності та надійності;
- на основі отриманих даних та результатів досліджень розробити рекомендації щодо використання сучасних методів автентифікації у мобільних додатках, враховуючи специфіку різних галузей та потреби користувачів.

Об'єктом дослідження є процес автентифікації користувачів у мобільних додатках.

Предметом дослідження є сучасні технології автентифікації користувачів у мобільних додатках з урахуванням вимог безпеки.

Методи дослідження:

- аналіз наукової літератури та відкритих джерел;
- експериментальні дослідження;
- компаративний аналіз.

Практична цінність роботи полягає в наступному:

- проведенні аналізу методів автентифікації та доцільність використання тих чи інших способів автентифікації у системах з різними вимогами безпеки;
- розробка рекомендацій щодо застосування розглянутих в даній роботі методів автентифікації в системах з урахуванням в останніх вимог до безпеки та створення Android додатка, який включатиме декілька різних методів автентифікації для порівняльного аналізу та випробувань їхньої ефективності та надійності.

РОЗДІЛ 1.
ОГЛЯД МЕТОДІВ АВТЕНТИФІКАЦІЇ У МОБІЛЬНИХ ДОДАТКАХ

1.1. Безпека у мобільних додатках

Сьогодні мобільні додатки широко використовуються для виконання різноманітних завдань, починаючи від спілкування у соціальних мережах і закінчуючи керуванням банківськими рахунками, роботою з документами та управління "розумним" будинком. Це зумовлює необхідність розробки та впровадження методів автентифікації, які б дозволили користувачам впевнено захищати свої дані та приватність.

Останнім часом особливої уваги набуває двофакторна автентифікація, яка передбачає використання двох незалежних факторів для перевірки особи користувача. Це можуть бути, наприклад, щось, що відомо тільки користувачеві (пароль або пін-код) та щось, що належить користувачеві (смартфон або спеціальний пристрій для генерації одноразових паролів), або щось, що характерно для фізіології користувача (відбиток пальця, малюнок сітківки ока чи голосовий зліпок). Такий підхід значно знижує ризик компрометації акаунту, оскільки зловмисникам стає важче отримати доступ до всіх необхідних факторів автентифікації одночасно. Відповідно, вірогідність успішної атаки на особисті дані користувачів значно знижується.

Біометричні методи автентифікації, такі як розпізнавання обличчя або відбитків пальців, також стають все більш популярними завдяки їхній зручності та високому рівню безпеки. Завдяки сучасним технологіям, ці методи забезпечують надійний рівень автентифікації та допомагають знижувати ризик викрадення паролів або інших методів ідентифікації.

Враховуючи поширення мобільних застосунків та важливість забезпечення конфіденційності, важливо розуміти, що використання надійних методів автентифікації є ключовим аспектом для розробників мобільних застосунків. Вони не

лише повинні бути простими у використанні та доступними для користувачів, але й відповідати сучасним вимогам безпеки. В результаті, забезпечення надійної автентифікації стає важливим завданням для забезпечення приватності та безпеки користувачів.

У сучасному світі, де все прискорюється, і інформаційні потоки миттєво розповсюджуються через платформи, такі як TikTok, Instagram Reels, та інші соціальні медіа, виникає неабияка потреба у максимально швидких, безпечних та зручних для користувачів методах автентифікації.

Розробники мобільних застосунків повинні враховувати вимоги сучасних користувачів, зосереджуючись на надійності та зручності автентифікаційних методів. Це означає, що користувачі мають легко та швидко отримувати доступ до своїх облікових записів без необхідності проходити складні процедури входу [1].

Цього можна досягти за допомогою комбінації традиційних методів автентифікації, таких як паролі, та сучасних біометричних рішень, які дозволяють швидко розпізнати особу користувача та надати йому доступ до своїх даних.

Останні дослідження показують, що застосування біометричних методів автентифікації, таких як розпізнавання обличчя та відбитків пальців, зростає з кожним роком. Згідно з прогнозами Juniper Research, кількість мобільних застосунків з біометричною автентифікацією збільшиться до 2024 року до 1,4 мільярда, порівняно з 429 мільйонами у 2018 році [2].

Наприклад, система автентифікації Google Authenticator дозволяє користувачам легко та швидко автентифікуватись, використовуючи двофакторну автентифікацію, яка поєднує щось, що знає користувач (пароль), та щось, що має користувач (смартфон).

Враховуючи стрімкий розвиток технологій та популярність мобільних застосунків, актуальність дослідження методів автентифікації користувачів у мобільних додатках не може бути переоціненою.

Оскільки все більше людей звертаються до застосунків для особистого та професійного використання, забезпечення безпеки особистих даних в останніх стає важливим завданням. Важливо зазначити, що зловмисники також адаптуються до

цього тренду, розробляючи нові способи атак та компрометації облікових записів користувачів. Дослідження методів автентифікації дає можливість виявити слабкі місця в чинних системах та розробити нові, більш надійні та зручні методи, які враховують різні аспекти користувачів, їх потреби та сучасні тенденції.

Однофакторна автентифікація полягає в використанні лише одного методу перевірки для доступу до системи чи облікового запису користувача. Це може бути, наприклад, пароль чи PIN-код.

Однофакторні методи автентифікації мають свої переваги та недоліки. Однією з найбільших переваг є їх простота та доступність. Крім того, вони є досить швидкими та легкими у використанні. Однак, недоліком є те, що вони не є надійними та легко підроблюються. Дані методи стали широко використовуватися у кібербезпеці та відомі своїми застосуваннями в онлайн-банкінгу, електронній пошті та інших онлайн-сервісах [3].

Однак, у зв'язку зі дедалі більшими загрозами кібератак та шахрайством, вони можуть бути надійнішими, якщо використовувати їх в поєднанні з іншими методами автентифікації, такими як двофакторна або багатофакторна автентифікація. Також захист можна покращити якщо поєднувати їх з іншими технологіями, такими як шифрування даних, приватні мережі та брандмауери.

1.2. Огляд найпоширеніших методів автентифікації

Використання імені користувача та пароля [4] – цей метод полягає в тому, що користувачі вводять унікальне ім'я користувача та пароль, що асоціюється з його обліковим записом. Це один з найбільш поширених та доступних методів автентифікації, однак він має певні недоліки. Наприклад, користувачі можуть вибрати слабкі паролі або використовувати однакові паролі для різних сервісів, що збільшує ризик злому.

Крім того, зловмисники можуть використовувати атаку перебору паролів, а також атаки соціальної інженерії, такі як підміна, фішинг та інше, щоб отримати доступ до чужих облікових записів.

PIN-коди (Personal Identification Numbers) – це короткі набори чисел, які користувачі вводять для перевірки своєї особистості. Вони є простішою альтернативою паролям та широко використовуються для доступу до банкоматів, мобільних пристроїв та інших систем. Однак PIN-коди мають свої власні обмеження, такі як менший розмір та відсутність використання спеціальних символів, що може знизити рівень безпеки порівняно з використанням паролів [5].

Персональні питання, такі як "Ім’я першого домашнього улюбленця" або "Назва вашої улюбленої книги", можуть використовуватися як форма однофакторної автентифікації. Користувачі відповідають на ці питання під час реєстрації та використовують їх для відновлення доступу до своїх облікових записів. Цей метод також має певні обмеження, оскільки зловмисники можуть отримати доступ до такої інформації шляхом шахрайства, злому, або ж, якщо відповіді на питання є занадто очевидними [6].

Автентифікація за допомогою QR-коду передбачає генерацію унікального QR-коду на сервері або ж локально на пристрої, який користувач сканує за допомогою свого мобільного пристрою. Цей метод дозволяє підтвердити особистість користувача без необхідності введення паролів або інших даних. QR-коди можуть бути використані для швидкої та зручної автентифікації, але також можуть стати мішенню для зловмисників, якщо коди будуть перехоплені або підроблені. Тому важливо використовувати безпечні протоколи передачі даних та забезпечувати належний захист сервера або пристрою, на якому генеруються QR-коди [7].

Android NSD дозволяє мобільним додаткам виявляти й спілкуватися з іншими пристроями в спільній WI-FI мережі. Автентифікація через Android NSD може використовуватися як альтернативу традиційним методам автентифікації та як додатковий метод автентифікації до уже наявного, оскільки вона забезпечує підтвердження особистості користувача на основі розташування та мережевих параметрів [8].

Та як і будь-яка технологія, NSD також має декілька недоліків.

1) Залежність від локальної мережі: NSD працює тільки в межах однієї локальної мережі.

2) Безпека: Через відкритість технології NSD до виявлення сервісів може виникнути проблема безпеки. Несанкціоновані пристрої можуть спробувати використовувати NSD для виявлення і використання сервісів без дозволу при наявності встановленого застосунку на якому цей сервіс працює, або ж його злому [9].

Попри згадані вище недоліки, технологія NSD продовжує бути цінним інструментом для побудови зв'язку між пристроями в локальній мережі, особливо у контексті розробки застосунків для системи Android.

Отже, враховуючи переваги та недоліки різних методів автентифікації, потрібно обирати найбільш слушний метод відповідно до вимог безпеки та потреб користувачів. Важливо зазначити, що не потрібно забувати застосовувати додаткові заходи безпеки, такі як надійні методи шифрування даних, оновлення програмного забезпечення до актуальної версії та забезпечення безпеки серверів або пристроїв, для зменшення ризиків, пов'язаних з однофакторною автентифікацією.

Висновки за розділом 1

Автентифікація є важливим процесом, який забезпечує безпеку облікових записів користувачів в мережі. Вона допомагає встановити довіру між користувачами та системами, зменшуючи ризик несанкціонованого доступу до чужих даних.

Однофакторна автентифікація, хоча і є широко використовуваною та доступною, має ряд недоліків, включаючи вразливість до атак перебору паролів, атак соціальної інженерії та інших видів злому. Це обумовлює необхідність пошуку додаткових заходів захисту та використання більш надійних методів автентифікації.

Різні методи однофакторної автентифікації, такі як використання імені користувача та пароля, PIN-кодів, персональних питань та автентифікації за

допомогою QR-коду, мають свої переваги та недоліки. Вибір найбільш слушного методу залежить від специфіки системи та потреб користувачів.

Автентифікація через Android NSD пропонує новий підхід до процесу автентифікації, базований на виявленні сервісів в мережі. Хоча цей метод має свої обмеження, він може стати ефективним доповненням до традиційних методів автентифікації.

Важливо враховувати, що незалежно від вибраного методу автентифікації, застосування додаткових заходів безпеки, таких як шифрування даних та захист серверів, є критично важливим для забезпечення надійного захисту користувачів.

Усі розглянуті методи потребують подальшого дослідження та вдосконалення, особливо з огляду на постійний розвиток технологій та постійно наростальні загрози кібербезпеки.

РОЗДІЛ 2.
АНАЛІЗ ТА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА АЛГОРИТМІВ АВТЕНТИФІКАЦІЇ

2.1. Network Service Discovery: використання та безпека в автентифікації

Network Service Discovery (NSD) став одним з ключових компонентів, що дозволяють пристроям в мережі взаємодіяти між собою. Цей механізм має в собі автоматичне виявлення служб в мережі, що значно спрощує процес налаштування з'єднань між пристроями. У контексті Android, NSD використовується для розкриття та пошуку служб на однаковому мережевому просторі імен.

Застосунки, що використовують NSD, можуть оголосити свої служби на мережі, що дозволяє іншим пристроям або застосункам виявити ці служби та взаємодіяти з ними. Це особливо корисно для застосунків, що вимагають спілкування з іншими пристроями в мережі, що забезпечує ефективну взаємодію між різними пристроями та службами.

Android NSD API дозволяє розробникам включати NSD функціональність у свої додатки. Це може бути використано для виявлення доступних служб в мережі або для оголошення своїх власних служб. Процес містить створення NsdServiceInfo об'єкта, який містить інформацію про службу, яку потрібно виявити або оголосити. Потім розробник може використовувати NsdManager, щоб зареєструвати, відкрити або виявити служби в мережі [10].

Хоча NSD має багато переваг, існують деякі питання безпеки, які варто розглянути. NSD не надає вбудованого механізму шифрування, тому всі передані дані можуть бути перехоплені та прочитані. Крім того, NSD не має вбудованих механізмів захисту від несанкціонованого доступу, тому служби, що використовують NSD, можуть бути вразливими до атак.

Іншим питанням безпеки є можливість атаки "man-in-the-middle", де зловмисник може перехопити комунікацію між двома пристроями, що

використовують NSD для взаємодії. Це може призвести до витоку конфіденційної інформації або несанкціонованого доступу до служб.

Використання NSD в контексті автентифікації на Android вимагає використання додаткових методів захисту для розв'язання цих питань безпеки. Одним із таких методів є використання шифрування для захисту даних, що передаються через NSD.

Крім того, служби можуть впровадити механізми аутентифікації для перевірки пристроїв, що намагаються з’єднатися з ними. Це може містити використання паролів, токенів аутентифікації або інших механізмів безпеки для перевірки того, чи має пристрій право доступу до служби.

NSD є потужним інструментом для виявлення і взаємодії зі службами в мережі в контексті Android. Однак, його використання в контексті автентифікації вимагає врахування ряду питань безпеки, включаючи несанкціонований доступ і можливість перехоплення даних. Важливо впровадити відповідні механізми захисту для забезпечення безпечного використання NSD в автентифікації на Android [11].

2.2. QR-коди: принцип роботи, використання та безпека в автентифікації

QR-коди – це двовимірні штрих-коди, що здатні зберігати велику кількість інформації. QR-коди можуть бути прочитані за допомогою камери смартфона, що робить їх зручним інструментом для використання в мобільних додатках, включаючи автентифікацію в системі Android [12].

QR-коди працюють за допомогою кодування даних у двовимірну матрицю. Ця матриця може бути розшифрована за допомогою спеціального сканера QR-кодів, який зазвичай є стандартною особливістю більшості смартфонів.

Коли користувач сканує QR-код за допомогою камери свого смартфона, застосунок для читання QR-кодів перетворює зображення назад на дані, що можуть бути використані для різних цілей. В контексті автентифікації, ці дані зазвичай містять інформацію, що ідентифікує користувача або пристрій.

У сфері автентифікації QR-коди можуть використовуватися для надання безпечного та ефективного методу входу в систему. Наприклад, застосунок може

генерувати унікальний QR-код для користувача при реєстрації. Користувач потім може сканувати цей код за допомогою іншого пристрою для автоматичного входу в систему без потреби вводити автентифікаційні дані.

Попри свою зручність, QR-коди також мають ряд питань безпеки, які варто враховувати. Через відсутність вбудованих механізмів шифрування, дані, збережені в QR-коді, можуть бути легко прочитані. Це означає, що якщо QR-код, який використовується для автентифікації, було перехоплено, зловмисник міг би отримати доступ до інформації користувача якщо не були залучені додаткові дані для перевірки користувача.

Також QR-коди можуть бути використані для фішингових атак. Наприклад, зловмисник може створити шкідливий QR-код, який, коли його сканують, перенаправляє користувача на шкідливий вебсайт або завантажує шкідливе ПЗ на їх пристрій.

Є різні способи захисту QR-кодів від цих загроз. Одним з них є використання шифрування для захисту даних, збережених в QR-коді. Це означає, що навіть якщо код перехоплюється, дані не можуть бути прочитані без відповідного ключа.

Іншим методом є використання двофакторної аутентифікації (2FA), де користувач повинен підтвердити свій вхід, використовуючи ще один метод, наприклад, SMS-підтвердження або застосунок автентифікатор, прикладами таких застосунків є Google Authenticator, Microsoft Authenticator. В деяких випадках сам застосунок може служити як автентифікатор і мати вбудовані механізми, прикладом таких застосунків є Facebook, Google та Gmail сповіщення на які буде приходити, якщо алгоритми компанії помітять, що користувач робить автентифікацію з підозрілого місця розташування, пристрою і т.д [13].

QR-коди пропонують ефективний спосіб автентифікації для мобільних застосунків, включаючи ті, що працюють на платформі Android. Однак, необхідно впроваджувати відповідні механізми захисту, щоб забезпечити, що QR-коди використовуються безпечно та ефективно.

2.3. Social Login: переваги, недоліки та безпека в автентифікації

Social Login, також відомий як соціальна автентифікація — це метод автентифікації, який дозволяє користувачам входити в систему, використовуючи свої облікові записи соціальних мереж. Для Android застосунків це означає, що користувачі можуть використовувати свої облікові записи Google, Facebook, Twitter або інших соціальних мереж для входу в систему, замість того, щоб створювати новий обліковий запис [14].

Однією з головних переваг Social Login є зручність для користувачів. Замість того, щоб запам'ятовувати численні автентифікаційні дані, користувачам просто потрібно використовувати свій обліковий запис соціальної мережі. Це також зменшує кількість кроків, потрібних для реєстрації та входу в систему, відповідно покращує досвід користування додатком.

Проте, Social Login також має свої недоліки. Цей метод створює залежність від третьої сторони — провайдера соціальних мереж. Якщо обліковий запис користувача в соціальній мережі стає недоступним, це може призвести до того, що користувач не зможе входити в систему.

Щодо безпеки, Social Login створює додаткові ризики. Якщо обліковий запис соціальної мережі було зламано, зловмисник міг би отримати доступ до всіх застосунків, які використовують Social Login через цей обліковий запис. Важливо використовувати додаткові механізми безпеки при використанні Social Login, які можуть мати с собі двофакторну аутентифікацію, політику безпечних паролів та періодичне оновлення доступу до застосунків.

Social Login є зручним способом автентифікації для користувачів Android. Втім, важливо враховувати потенційні ризики безпеки та вживати необхідних заходів для їх зменшення.

Хоча Social Login може полегшити процес входу в систему, необхідно також забезпечити, щоб дані користувачів залишалися захищеними.

2.4. Порівняльна характеристика популярних методів автентифікації

Переваги Social Login:

1) Швидкість і зручність: Користувачі можуть швидко і легко ввійти в систему, використовуючи вже чинні облікові записи в соціальних мережах, що зменшує кількість дій при реєстрації або вході.

2) Зменшення ризику забутого пароля: Оскільки користувачі використовують облікові записи соціальних мереж, вони менш схильні забувати свої паролі.

Недоліки:

1) Залежність від третьої сторони: Якщо соціальна мережа, яку використовується для входу, зазнає збою, це може вплинути на здатність користувачів увійти в систему.

2) Ризики з приватністю: Користувачі можуть бути незадоволені тим, що застосунок отримує доступ до деяких даних їхнього профілю в соціальній мережі.

Переваги Network Service Discovery:

1) Доступ до локальних сервісів: NSD дозволяє застосунку виявляти сервіси, які надають інші пристрої в локальній мережі.

2) Підтримка peer-to-peer застосувань: NSD корисний для різних застосувань peer-to-peer, таких як обмін файлами.

Недоліки:

1) Проблеми з надійністю: Як вказано в деяких обговореннях на StackOverflow, можуть виникати проблеми з надійністю та стабільністю NSD.

2) Потреба в локальній мережі: NSD працює тільки в межах локальної мережі, що обмежує його використання в інтернеті.

Переваги QR кодів.

1) Простота використання: QR-коди легкі для сканування та використання, що робить їх доступними для більшої кількості користувачів.

2) Багатофункціональність: QR-коди можуть використовуватися для різноманітних цілей, включаючи автентифікацію, надання інформації про продукт, і навіть для з'єднання з Wi-Fi.

Недоліки:

1) Потреба в камері: Користувачам потрібна камера для сканування QR-кодів, що може бути неможливим для деяких пристроїв.

2) Ризики безпеки: Якщо QR-код веде на шкідливий вебсайт, це може призвести до витоку даних користувача.

Порівняльна характеристика представлених вище методів автентифікації наведена в таблиці 2.1.

Таблиця 2.1

Порівняльна характеристика методів автентифікації

	Social Login	NSD	QR-код
Переваги	Швидкість і зручність: Користувачі можуть швидко ввійти в систему використовуючи вже наявні облікові записи в соціальних мережах; Зменшення ризику забутого пароля.	Доступ до локальних сервісів; Підтримка peer-to-peer застосувань.	Простота використання; Багатофункціональність.
Недоліки	Залежність від третьої сторони, проблеми з доступом до соціальної мережі можуть вплинути на доступ до системи; Ризики з приватністю.	Проблеми з стабільністю; Потреба в мережі; Обмеження використання в інтернеті.	Потреба в камері; Ризики безпеки при переході на шкідливий вебсайт.

Підсумовуючи, вибір методу автентифікації залежить від конкретних потреб користувачів. Social Login забезпечує зручність і швидкість, але може мати проблеми з приватністю. NSD дозволяє виявляти сервіси в локальній мережі, але може бути ненадійним і обмеженим у використанні. QR-коди легкі в використанні та гнучкі, але вимагають камери й можуть мати ризики безпеки.

Висновки за розділом 2

На основі аналізу різних методів автентифікації, які були розглянуті в другому розділі, можна зробити наступні висновки:

Social Login є потужним інструментом, що полегшує процес автентифікації для користувачів, дозволяючи їм використовувати облікові записи соціальних мереж для входу в застосунок. Це не тільки покращує користувацький досвід, зменшуючи кількість потрібних дій для входу, але і знижує ризик забутого пароля. Однак, цей метод має і свої недоліки, зокрема ризики безпеки та залежність від сторонніх сервісів.

Network Service Discovery (NSD) пропонує унікальну можливість виявлення і взаємодії з сервісами в локальній мережі. Це може бути особливо корисним для розробки peer-to-peer застосувань, таких як обмін файлами або з’єднання з іншими пристроями без участі користувача. Проте, NSD має обмеження в тому, що працює лише в межах однієї локальної мережі, і може мати проблеми з надійністю.

QR-коди використовуються в широкому спектрі сценаріїв, включаючи автентифікацію. Вони прості в використанні та доступні широкому колу користувачів. Однак, для їх використання потрібна камера, і існує ризик безпеки, пов'язаний зі шкідливими QR-кодами.

Загалом, при виборі методу автентифікації для застосунку важливо враховувати специфіку, потреби користувачів, а також потенційні ризики. Жоден з методів не є ідеальним для всіх сценаріїв, тому важливо поєднувати різні методи та вживати заходів для забезпечення безпеки користувачів.

РОЗДІЛ 3.
КОМПОНЕНТИ ANDROID ДОДАТКА

3.1. Android Studio та архітектура MVVM

Для створення додатка в даній роботі використовується потужне інтегроване середовище розробки (IDE) під назвою Android Studio.

Android Studio є основним інструментом для розробки Android застосунків, забезпечуючи розширені можливості та зручні інструменти для розробників. За допомогою Android Studio розробник зможе створити візуально привабливий та функціональний інтерфейс. Інтегрована розробка на основі XML дозволяє створювати та редагувати елементи інтерфейсу у зручний спосіб, використовуючи візуальний редактор [15].

Одним із ключових аспектів розробки даного проєкту є використання технології Material Design версії 3. Material Design є сучасним дизайн-стандартом, розробленим компанією Google, який надає чіткі та зрозумілі принципи та рекомендації щодо створення інтерфейсів. Використання Material Design дозволяє створити стильний та однорідний вигляд додатка, з урахуванням сучасних трендів та кращих практик. Material Design 3 містити нові елементи дизайну, вбудовані компоненти для створення інтуїтивно зрозумілого UI, покращену анімацію та переходи між екранами, а також має підтримку тем та стилів, що дозволяє адаптувати інтерфейс до різних вимог та налаштувань користувачів [16].

Для організації коду та забезпечення модульності й підтримки розширення функціональності, в даному проєктові буде використовуватись архітектурний шаблон MVVM (Model-View-ViewModel) [17].

Архітектура Model-View-ViewModel це модель проєктування, яка дозволяє розділити логіку в застосунку на три основні компоненти:

1) Модель (Model) – це компонент, що відповідає за доступ до даних. Модель може включати різні джерела даних, наприклад вебсервіси або локальні бази

даних, і вона відповідає за отримання, збереження та надання даних для інших компонентів.

2) Вид (View) – це компонент, який відповідає за відображення даних користувачу. В Android, вид може бути представлений активностями, фрагментами або іншими компонентами інтерфейсу.

3) Вид-модель (ViewModel) – це міст між Видом та Моделлю. ViewModel отримує дані від Моделі та передає їх до Виду. ViewModel не має знання про конкретні деталі Виду, і це допомагає забезпечити модульність та можливість повторного використання коду.

Таке розділення дозволяє забезпечити чітку відокремленість між компонентами та полегшує тестування та обслуговування коду.

При розробці додатка будуть використовуватись наступні компоненти системи Android:

Fragment – це компонент інтерфейсу користувача в Android, який представляє частину екрана користувача. Для кращого розуміння можна уявити, що головний екран який бачить користувач — це книга, а фрагменти — це різні розділи цієї книги. Кожен розділ має свою власну тему та історію, і вони разом складають цілісну книгу. Фрагменти в Android працюють подібним чином. Вони допомагають організувати код і інтерфейс користувача, розбиваючи їх на логічні частини.

ViewModel – це компонент в архітектурі MVVM (Model-View-ViewModel), який містить дані та бізнес-логіку, необхідну для представлення фрагмента (або виду, в цьому контексті). ViewModel не знає про конкретний вид, з яким він працює. Він просто надає дані та обробляє бізнес-логіку. Це створює розрив між інтерфейсом користувача та бізнес-логікою, що полегшує тестування і підтримку коду.

Тож, якщо підсумувати та коротко описати інформацію про кожен компонент:

- Fragment створюється, щоб розділити інтерфейс користувача на логічні частини. Це зроблено для того, щоб код був більш організованим і зрозумілим. Наприклад, ми можемо мати один фрагмент для входу користувача, інший для відображення списку продуктів, і так далі.

- ViewModel створюється, щоб відокремити бізнес-логіку від інтерфейсу користувача. Він допомагає забезпечити, щоб фрагмент (або вид) залишався легким для системи та не навантажував її, щоб інші компоненти відповідали тільки за відображення інформації користувачеві, а не за обробку важкої бізнес-логіки.
- Activity становити собою важливу компоненту, яка представляє одне вікно, в якому користувач може взаємодіяти для виконання певних дій. Кожен екран в Android- застосунку відповідає одній активності. Отже, активність служить рамками, в які поміщається користувацький інтерфейс додатка. Це вікно може заповнити екран, але може бути й меншим, і може бути вставлено в інше вікно, що створює інша активність. В один момент часу на екрані може відображатися лише одна активність. Коли користувач переходить з одного екрана на інший, поточна активність призупиняється, а нова активність стає активною.

Всі активності в Android є підкласами класу Activity. Кожна активність має життєвий цикл, що включає різні стани, такі як створена, почата, повторно запущена, призупинена та знищена. Android запускає і знищує активності відповідно до потреб користувача та системи, що робить управління життєвим циклом активності важливою частиною розробки Android-додатка.

У контексті розробки за допомогою архітектури MVVM, активності зазвичай відповідають за реагування на дії користувача, такі як натискання на кнопки, та за оновлення користувацького інтерфейсу відповідно до даних, які надає ViewModel. Однак логіка обробки даних, така як з’єднання з базою даних або виконання мережевих запитів, повинна бути відокремлена в ViewModel, щоб забезпечити кращу модульність, можливість повторного використання та тестування коду.

Крім того, активності в Android відповідають за збереження та відновлення стану користувацького інтерфейсу. Це означає, що якщо активність була призупинена або знищена системою (наприклад, під час обертання екрана або коли користувач перейшов до іншого застосунку).

Все це є частиною системи Android, тому важливо дотримуватись правил користування компонентами згідно документації від Google.

3.2. Мова програмування Kotlin

Розробка функціональності проєкту буде здійснюватися мовою програмування Kotlin, яка є офіційною рекомендованою мовою для розробки Android застосунків [18].

Використання Kotlin дозволяє писати чистий та ефективний код, а також ця мова програмування володіє розширеними можливостями, що сприяють покращенню продуктивності та надійності додатка, основні переваги цієї мови наведені в таблиці 3.1.

Таблиця 3.1

Основні переваги мови програмування Kotlin

Нульові вказівники за умовчанням	Kotlin використовує систему типів, яка автоматично знижує ризик зустрічі з вказівниками на null, що може спричинити виключення виконання.
Висока сумісність з Java	Kotlin сумісний з Java, що означає, що розробник має змогу використовувати всі екосистеми Java, а також перетворювати Java код на Kotlin і навпаки.
Концентрація на безпеці	Kotlin включає ряд особливостей, що сприяють безпеці коду, включаючи перевірку на нульові вказівники та незмінність за замовчуванням.
Короткість коду	Kotlin має багато “синтаксичного цукру”, який допомагає писати коротший код без втрати читабельності та розуміння.
Підтримка Android	Kotlin є офіційно підтримуваною мовою розробки для Android, що робить його ідеальним вибором для розробників Android.

Kotlin – це функціональна об'єктноорієнтована мова програмування зі статичним написанням, що дозволяє компілювати для віртуальної машини Java та JavaScript. Він розроблений командою програмістів з JetBrains, редактором IntelliJ

IDEA, інтегрованого середовища розробки для Java і на якій базується Android Studio, офіційна IDE для розробки програм для Android.

Google реалізує всю свою роботу з підтримки Kotlin в Android Studio. Але поза цим аспектом Kotlin має багато інших переваг.

За даними Google:

“Котлін виразний, стислий, розширюваний, потужний і приємний для читання та письма, а також має цікаві функції безпеки з точки зору дозволеності та незмінності, які відповідають Вашим інвестиціям у створення програм за замовчуванням безпечними та ефективними”.

“Розробка Android все частіше буде здійснюватися на Kotlin”.

“Для Kotlin спочатку буде доступно багато нових API та функцій Jetpack. Якщо ви починаєте новий проєкт, вам слід писати його на Kotlin”.

3.2. Kotlin Coroutines

Корутини в Kotlin є важливим елементом асинхронного програмування, що використовуються для спрощення написання багатопотокового коду. Вони вводять концепцію суспендуючих функцій, які дозволяють виконувати довготривалі або блокувальні операції без блокування потоку виконання, в якому вони запущені [19].

Асинхронне програмування – це парадигма програмування, яка використовується для ефективного управління завданнями, що можуть бути виконані незалежно одне від одного. По своїй суті, корутини Kotlin дозволяють розробникам писати асинхронний код так само просто, як і синхронний, використовуючи ключове слово suspend для вказівки на те, що виконання функції може бути призупинено і відновлено пізніше. На відміну від традиційних механізмів багато поточності, таких як потоки та виконавці, корутини мають низьку вартість створення та виконання, що дозволяє використовувати їх у великій кількості без занепаду продуктивності. Це робить їх ідеальним рішенням для виконання великого числа асинхронних завдань, таких як мережеві запити, взаємодія з базами даних, читання та запис файлів, та ін [20].

У контексті розробки Android-застосунків корутини часто використовуються для виконання довготривалих або важких завдань на фоновому потоці, щоб не блокувати основний потік інтерфейсу користувача. Наприклад, вони можуть використовуватися для завантаження даних з мережі, виконання важкого обчислювального завдання або отримання даних з бази даних, все це може зайняти деякий час і викликати блокування інтерфейсу користувача, якщо воно виконується на основному потоці.

У своєму загальному вигляді, корутини можуть бути створені з допомогою функцій launch або async, які належати до класу CoroutineScope. Ці функції запускають нову корутину та повертають Job або Deferred відповідно, що дозволяє контролювати життєвий цикл корутини та отримувати результат її виконання.

Загалом, корутини в Kotlin дозволяють створювати ефективний, простий у розумінні та легко підтримуваний асинхронний код, що робить їх важливою складовою сучасної розробки Android-застосунків. Використовуючи концепції, що були розглянуті, можна зрозуміти, як глибоко вкорінені Kotlin Coroutines в розробці сучасних Android-застосунків. Ці асинхронні конструкції не тільки спрощують код, але й значно підвищують продуктивність додатка, забезпечуючи плавну роботу користувачів з графічним інтерфейсом.

У контексті розглянутого проєкту, корутини були використані для генерації QR-кодів. Вони дозволяють перемістити важку роботу в фоновий потік, попереджаючи при цьому блокування основного потоку інтерфейсу користувача. Така асинхронна обробка забезпечує швидку відповідь на дії користувача та високу продуктивність додатка. Корутини також можуть бути використані для асинхронного доступу до мережі та баз даних, якщо ці задачі виникнуть. Вони дозволяють виконувати велику кількість асинхронних завдань паралельно, забезпечуючи при цьому високу ефективність.

Таким чином, Kotlin Coroutines є важливим інструментом для створення ефективних Android-застосунків.

Також вони можуть застосовуватись для виконання важких операцій, виконання яких треба уникати в у головному потоці без використання корутин.

3.3. Взаємодія з платформою Google Firebase

Google Firebase – це масштабна платформа розробки, яка надає різноманітні сервіси та функції, що полегшують розробку застосунків, зокрема Android-застосунків. Вона включає такі сервіси, як база даних в реальному часі, аутентифікація, хмарне сховище, хостинг та багато інших. Firebase також інтегрований з іншими продуктами Google, що дозволяє розробникам використовувати ці сервіси безпосередньо у своїх додатках [21].

Одним з ключових аспектів, що робить Firebase ідеальним рішенням для нашого Android-застосунку, є його можливості аутентифікації. Firebase Authentication дозволяє нам інтегрувати різні методи входу в розроблений застосунок, включаючи Google Sign-In. Це забезпечує застосунку безпечну систему аутентифікації, яка використовує вже наявні облікові записи Google користувачів, що робить швидкий та простий вхід.

Firebase – це одне з BaaS-рішень (Backend as a Service), яке дає розробнику масу можливостей. Це і сервер, і база даних, і хостинг, і аутентифікація в одній платформі.

Так, Firebase Realtime Database надає розробникам API, який синхронізує дані додатки між клієнтами та зберігає їх в хмарному сховищі. Застосунок підключається до бази даних через WebSocket, який відповідає за синхронізацію даних протягом усього сеансу.

Firebase має доволі великий список інструментів для допомоги при розробці будь-якого типу застосунків, зокрема ось деякі з них:

- Firebase Authentication – Цей сервіс дозволяє легко та безпечно автентифікувати користувачів. Firebase Authentication підтримує різні методи автентифікації, включаючи вхід за допомогою електронної пошти та пароля, автентифікацію за допомогою Google, Facebook, Twitter та GitHub, а також автентифікацію за допомогою телефонного номера [22].
- Firebase Firestore – Це хмарна NoSQL база даних, що дозволяє синхронізувати дані між користувачами в реальному часі. Firestore підтримує

автоматичне масштабування, прості запити та транзакції, розподіл індексації даних та інше [23].

- Firebase Cloud Messaging – Це сервіс, який дозволяє легко надсилати повідомлення на різні платформи, такі як Android, iOS, вебсайт. Це можуть бути як миттєві повідомлення, так і повідомлення, які доставляються в той час, коли вони найбільш корисні для користувача [24].
- Firebase Storage – Цей сервіс надає можливість зберігати та обмінюватися великими файлами користувачів, такими як зображення або відео. Firebase Storage автоматично масштабується, забезпечуючи простоту і зручність роботи з мультимедійними файлами [25].
- Firebase Hosting – Надає швидкі та надійні статичні хостинг-сервіси для вебсайту. Він підтримує SSL за замовчуванням та забезпечує глобальне розповсюдження через CDN [26].
- Firebase Analytics – Це безплатний інструмент для аналізу користувацької активності, який дозволяє вимірювати та аналізувати ключові показники ефективності додатка. Він дозволяє відстежувати події, створювати аудиторії та налаштовувати конверсії [27].
- Firebase Crashlytics – Це ефективний інструмент для відстеження, пріоритетності та розв'язання проблем зі збоями в програмному забезпеченні. Він допомагає розробникам вчасно виявляти та виправляти проблеми, що впливають на користувачів додатка [28].

Використання Firebase може значно спростити розробку застосунків, оскільки при його використанні розробник може сконцентруватися на створенні досвіду користувачів, а не на побудові власних backend платформ.

Для створення даного проєкту у Firebase були виконані наступні кроки:

Крок перший: перехід на вебсайт Firebase і вхід в обліковий запис Google – Перш за все, потрібно перейти на офіційний вебсайт Firebase – https://console.firebase.google.com/. Тут необхідно натиснути на кнопку “Увійти” в правому верхньому куті екрану і ввести дані свого облікового запису Google.

Крок другий: натискання на кнопку “Додати проєкт” та введення назви проєкт — Після входу в обліковий запис, стане видимою кнопка “Додати проєкт”. Після натискання на неї з‘явиться вікно, в якому потрібно ввести назву нового проєкт. Ця назва використовується для ідентифікації проєкт у Firebase і може бути будь-якою, вікно створення назви зображено на рисунку 3.1.

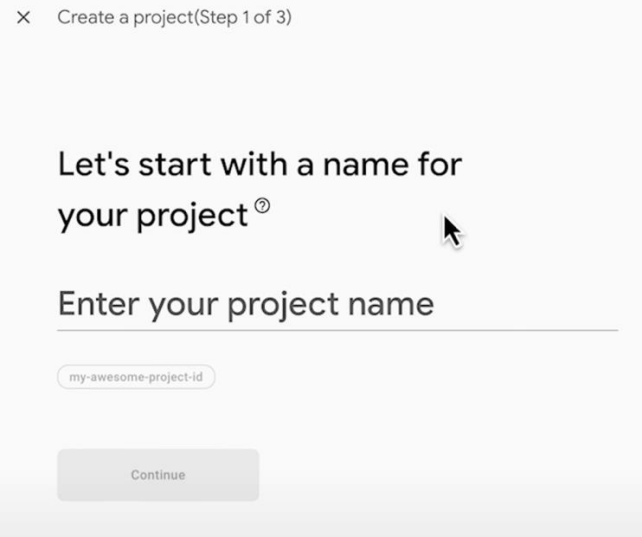

Рисунок 3.1 – Вікно створення назви проєкту на платформі Firebase

Крок третій: вибір регіону для свого проєкту та активація Google Analytics для нього – Під час створення нового проєкту, Firebase пропонує вибрати регіон для даних проєкту. Вибір регіону важливий, оскільки він може вплинути на швидкість відгуку сервера. Крім того, є можливість активувати Google Analytics для свого проєкту, що дозволить відстежувати використання додатка користувачами. Вікно підключення аналітики представлене на рисунку. 3.2.

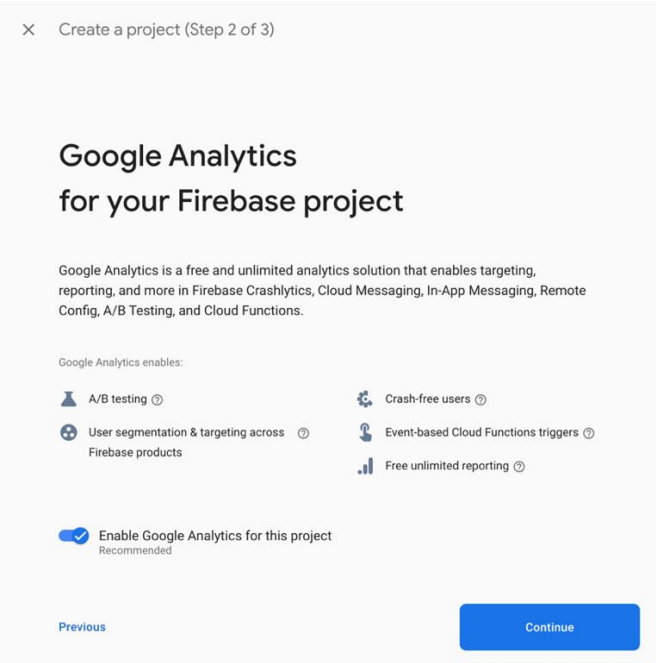

Рисунок 3.2 – Вікно підключення сервісу Google analytics

Крок четвертий: прив’язування додатка до проєкту Firebase відбувається за введенням інформації про пакет Android та назву застосунку – На наступному етапі потрібно додати Android-застосунок до новоствореного проєкту Firebase. Це здійснюється шляхом введення імені пакета додатка (який знаходиться в файлі Manifest застосунку) та назви застосунку. Вікно прив’язування до проєкту можна побачити на рисунку 3.3.

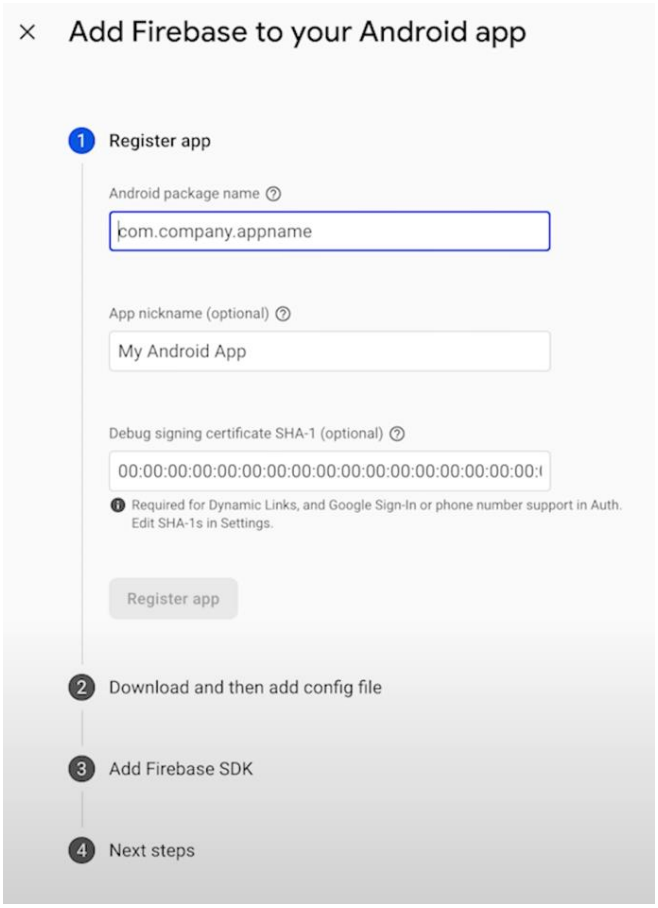

Рисунок 3.3 – Прив’язування застосунку до платформи Firebase

Крок п’ятий: завантаження файлу конфігурації google-services.json та його переміщення в директорію app/ проєкт в Android Studio – Після додавання файлу в директорію додатка, Firebase згенерує файл конфігурації з назвою google-services.json, який можна завантажити із платформи, вікно завантаження якого наведено на рисунку 3.4. Цей файл містить всю необхідну інформацію для інтеграції Android-додатка з проєктом Firebase. Це і буде наступним кроком, виконуємо завантаження цього файлу та переміщення його в директорію app/ проєкту AuthGuard.

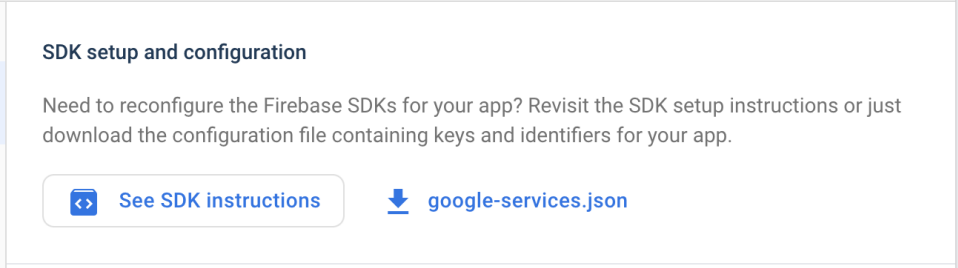

Рисунок 3.4 – Вікно завантаження файлу конфігурації google-services.json

Крок 6: додавання залежностей Firebase SDK до файлу build.gradle проєкту Android Studio – Останнім кроком є додавання залежностей Firebase SDK до файлу build.gradle на рівні проєкту і модулю [29].

Firebase SDK надає бібліотеки, які допомагають інтегрувати різні сервіси Firebase, такі як аутентифікація, бази даних, зберігання файлів і інше, в Android-застосунок. Необхідні залежності наведено у таблиці 3.2.

Таблиця 3.2

Залежності файлу build.gradle

apply plugin: 'com.google.gms.google-services'
implementation("com.google.firebase:firebase-auth:22.0.0")
implementation("com.google.android.gms:play-services-auth:20.5.0")
implementation("com.google.firebase:firebase-analytics-ktx")
implementation("com.google.android.gms:play-services-vision:20.1.3")

Після успішного виконання початкових кроків із встановлення та інтеграції Firebase у проєкт тепер можна починати активно використовувати різноманітні сервіси, які ця платформа може надати. Це охоплює велику кількість служб та можливостей, які дозволяють значно розширити функціональність додатка, від аналізу даних до управління користувачами та інше.

Особлива увага була приділена автентифікації користувачів. Використовуючи Firebase Authentication, було реалізовано можливість входу в систему за допомогою Google Sign in, що дозволяє користувачам легко та безпечно увійти в застосунок, використовуючи свої чинні облікові записи Google, що суттєво спрощує процес реєстрації та автентифікації.

Додатково, використовуючи Firebase маємо можливість використовувати інші висококласні служби, такі як Firebase Analytics. Це дозволяє отримувати важливі відомості про поведінку наших користувачів, що допомагає нам зрозуміти, як вони взаємодіють з додатком, і зробити висновки для його поліпшення.

Висновки за розділом 3

В рамках третього розділу було виконано значну роботу по аналізу і використанню ключових компонентів Android-додатка, що показало їхню взаємну взаємодію та важливість в створенні стабільних та продуктивних застосунків.

Були досліджені компоненти активності та фрагменти, їх роль та взаємодія в Android-додатках. Активності є ключовим компонентом інтерфейсу користувача Android, вони керують вікнами, в яких користувач взаємодіє з додатком. Фрагменти є частинами активностей і їх можна повторно використовувати в різних активностях. Можна зрозуміти, як вони допомагають створювати адаптивні та гнучкі інтерфейси користувача.

Зосередившись на архітектурі MVVM, було вивчено і реалізовано кожний з її компонентів. Було пояснена, чому модель відповідає за бізнес-логіку, в той час, як вигляд відповідає за інтерфейс користувача.

Firebase став цінним доповненням до проєкту, він надає можливості хмарного зберігання, автентифікації та аналітики. Було вивчено процес інтеграції Firebase в Android-застосунок, починаючи від завантаження файлу конфігурації google-services.json та переміщення його в директорію проєкт в Android Studio, до додавання залежностей Firebase SDK у файл конфігурації build.gradle.

Третій розділ кваліфікаційної роботи був успішним виконанням вивчення та практичної реалізації ключових компонентів Android-додатка, архітектури MVVM і Firebase. Можна відзначити успішне інтегрування наведених концепцій і технологій в кваліфікаційному проєкті.

РОЗДІЛ 4.
РЕАЛІЗАЦІЯ АЛГОРИТМІВ NSD, GOOGLE SIGN IN ТА QR-CODE

4.1. Розробка алгоритму Google Sign in та UI для користувача

При розробці інтерфейсу користувача основна увага була зосереджена на створенні зручної навігації, інтуїтивно зрозумілих елементах управління та привабливого вигляду, що забезпечить позитивний досвід взаємодії з додатком AuthGuard.

Загалом, використання Android Studio, XML та технології Material Design 5 дозволило розробити інтерфейс користувача, який буде сучасним, естетичним і функціональним, забезпечуючи зручну та правильну взаємодію з додатком.

Отже, першим кроком перед реалізацією бізнес-логіки алгоритмів, мені потрібно створити головний екран, який буде включати такі елементи як кнопки натиснувши на які користувач зможе обрати метод автентифікації, який він захоче використати в застосунку, на рисунку 4.1 зображено перший екран який буде бачити користувач коли відкриє застосунок:

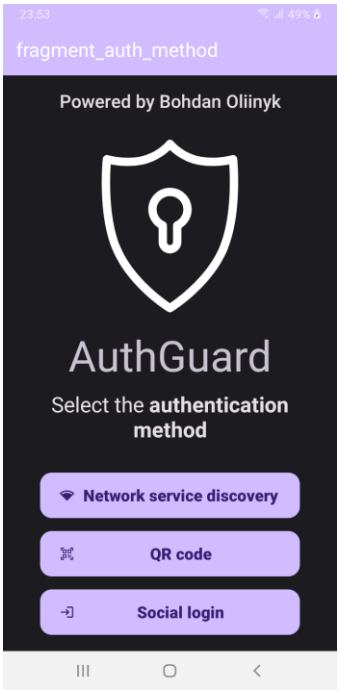

Рисунок 4.1 – Головний екран додатка

Для реалізації Google Sign in в проєкті були використані два класи: SocialLoginFragment і SocialLoginViewModel. Кожен з цих класів виконує важливу роль у роботі даної технології. SocialLoginFragment представлений на рисунку 4.2.

Рисунок 4.2 – Вигляд екрана Social login

Цей клас становить собою фрагмент, що відповідає за відображення інтерфейсу Google Sign in на екрані. Він містить ряд методів та полів, які визначають його поведінку:

- googleSignInClient: Клієнт Google Sign In, який використовується для авторизації користувачів.
- resultLauncher: Змінна для запуску ActivityResultLauncher, яка обробляє результат спроби входу користувача.
- onCreateView(): Цей метод відповідає за створення виду фрагмента та прив'язку даних.

- onViewCreated(): В цьому методі відбувається ініціалізація Google Sign in клієнта та установка слухачів на кнопки входу та виходу.
- onStart(): Цей метод перевіряє, чи користувач вже увійшов, при запуску фрагмента.
- onClick(): Цей метод відповідає за обробку натискань по кнопках входу та виходу.
- onGoogleSignInClicked() та onGoogleSignOutClicked(): Ці два методи використовуються для реалізації входу та виходу відповідно.

SocialLoginViewModel – клас є спадкоємцем від абстрактного класу ViewModel в архітектурі MVVM і служить мостом між фрагментом (View) та моделлю даних. Він містить наступні важливі елементи:

- _isGoogleSignInAvailable: LiveData, яка містить інформацію про наявність акаунта Google Sign in.
- init(): Цей метод викликається для ініціалізації ViewModel.
- checkIfUserAlreadySignedIn(): Цей метод перевіряє, чи є вже користувач увійшов в Google Sign in.
- onGoogleSignInClicked() та onGoogleSignOutClicked(): Ці два методи викликаються при натисканні на відповідні кнопки на екрані фрагмента.

За допомогою представлених двох класів, Google Sign In імплементується в проєкт, що дозволяє кінцевим користувачам увійти в застосунок за допомогою свого облікового запису Google [30].

Детальний код реалізації представлений у Додатку Б.

4.2. Розробка UI для алгоритму Network Service Discovery

При переході з основного екрана додатка, навігація додатка перенаправить користувача до виду, який буде демонструвати можливість обрати: або пристрій-сервер, або пристрій-клієнт через те, що того потребує технологія NSD та зображений на рисунку 4.3.

Рисунок 4.3 – Екран вибору ролі пристрою

Далі я продемонструю короткий лістинг xml-коду, що реалізує розмітку даних деяких елементів екрана який зображений на рисунку 4.4 та рисунку 4.5.

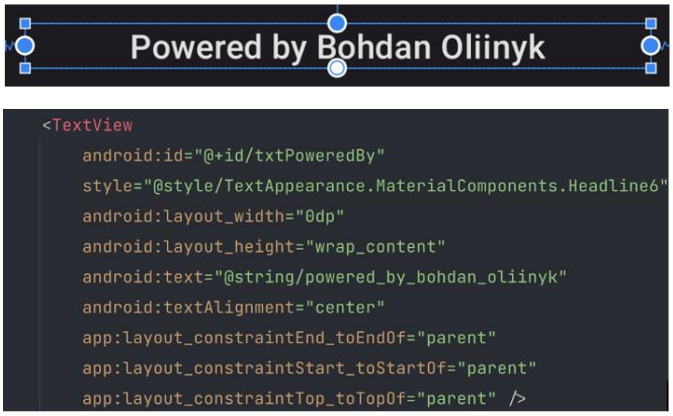

Рисунок 4.4 – Титульна назва головного екрану додатка

Рисунок 4.5 – Розмітка кнопок екрана NsdChooseRoleFragment

При натисканні на кнопки: “I want to be client” рисунок 4.6 або “I want to be server” рисунок 4.7, користувач побачить відповідний екран:

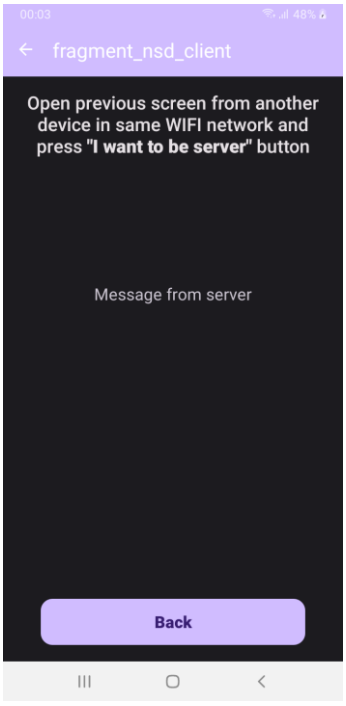

Рисунок 4.6 – Екран «клієнта»

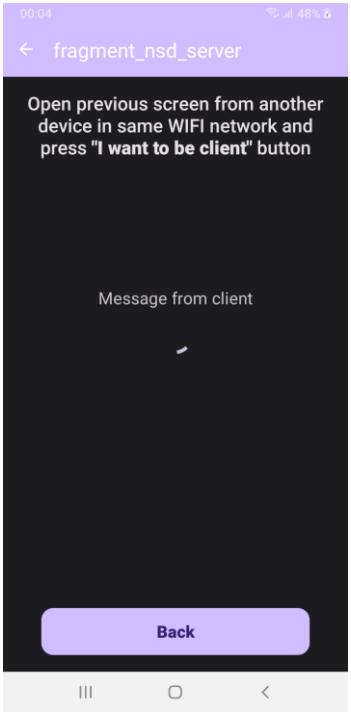

Рисунок 4.7 – Екран «сервера»»

Для демонстрації можливостей алгоритму може бути використано один екран, який буде відкритий з двох пристроїв одночасно. Відповідно до рекомендацій дизайну Material 5 та для зручності демонстрації його роботи, запобіганню помилок - було прийнято рішення щодо реалізації додатка за допомогою двох екранів.

Перший крок: створення коду для отримання екземпляру NsdManager [31], який є системним сервісом для управління технологією (рисунок 4.8):

```
nsdManager = requireContext().getSystemService(Context.NSD_SERVICE) as NsdManager
```

Рисунок 4.8 – Отримання екземпляра NsdManager

Другий крок: створення коду об'єкта NsdServiceInfo (реалізація якого зображена на рисунок 4.9), що містить інформацію про сервіс, який потрібно оголосити та відкрити для виявлення в мережі.

```
val serviceInfo = NsdServiceInfo().apply { this: NsdServiceInfo
    serviceName = UniversalUtils.getDeviceBrandAndModel()
    serviceType = SERVICE_TYPE
    port = 12345
}
```

Рисунок 4.9 – Об’єкт NsdServiceInfo

Третій крок: налаштування слухачів подій [32] для реєстрації та скасування реєстрації сервісу з використанням технології NSD, код яких зображений на рисунку 4.10.

```
private fun setupListeners() {
    registrationListener = object : NsdManager.RegistrationListener {
        override fun onServiceRegistered(serviceInfo: NsdServiceInfo) {
            Log.d( tag: "NSD",  msg: "Service registered: ${serviceInfo.serviceName}")
        }

        override fun onRegistrationFailed(serviceInfo: NsdServiceInfo, errorCode: Int) {
            Log.e( tag: "NSD",  msg: "Registration failed with error code: $errorCode")
        }

        override fun onServiceUnregistered(serviceInfo: NsdServiceInfo) {
            Log.d( tag: "NSD",  msg: "Service unregistered: ${serviceInfo.serviceName}")
        }

        override fun onUnregistrationFailed(serviceInfo: NsdServiceInfo, errorCode: Int) {
            Log.e( tag: "NSD",  msg: "Unregistration failed with error code: $errorCode")
        }
    }
}
```

Рисунок 4.10 – Реєстрація слухачів

Функція setupListeners() створює об'єкт registrationListener, який реалізує інтерфейс NsdManager.RegistrationListener. Це дозволяє мені визначити обробника подій, які будуть спрацьовують при реєстрації та скасуванні реєстрації сервісу.

Четвертий крок: реєстрація сервісу з використанням технології за допомогою екземпляру NsdManager який показаний на рисунку 4.11.

```
nsdManager.registerService(serviceInfo, NsdManager.PROTOCOL_DNS_SD, registrationListener)
```

Рисунок 4.11 – Реєстрація сервісу

П’ятий крок: створення коду для сервера який виконує налаштування та запуск серверного сокета для отримання з’єднання з клієнтом, реалізація зображена на рисунку 4.12.

```
fun startServerSocket() {
    CoroutineScope(Dispatchers.IO).launch { this: CoroutineScope
        withContext(Dispatchers.IO) { this: CoroutineScope
            try {
                val serverSocket = ServerSocket(servicePort!!)
                val socket = serverSocket.accept()

                val input = BufferedReader(InputStreamReader(socket.getInputStream()))
                val output = PrintWriter(socket.getOutputStream(), autoFlush: true)

                output.println("Hello from server!")
                val receivedData = input.readLine()
                withContext(Dispatchers.Main) { this: CoroutineScope
                    _receivedMessage.postValue(receivedData)
                }
                Log.d( tag: "NSD", msg: "Received data from client: $receivedData")

                socket.close()
                serverSocket.close()
            } catch (e: IOException) {
                Log.e( tag: "NSD", msg: "Server socket error: ", e)
            }
        }
    }
}
```

Рисунок 4.12 – Код методу startServerSocket

На цьому етапі основна логіка алгоритму для пристрою-серверу уже закінчена і наступним кроком буде імплементація логіки для пристрою клієнту.

В режимі клієнта я також створюю об’єкт nsdManager аналогічно до того як було продемонстровано для пристрою-серверу, після цього роблю налаштування слухачів подій для клієнта. Функція setupListeners() створює два об'єкти слухачів: discoveryListener і resolveListener.

discoveryListener є об'єктом, який реалізує інтерфейс NsdManager.DiscoveryListener, цей слухач відповідає за обробку подій, пов'язаних з початком та завершенням виявлення сервісів.

Метод onDiscoveryStarted (regType: String) викликається, коли виявлення сервісів розпочинається. Виконується логування повідомлень, щодо початку виявлення сервісів.

Метод onServiceFound (serviceInfo: NsdServiceInfo) викликається, коли знайдено новий сервіс. Виконується логування повідомлення щодо знайденого сервісу і викликається метод nsdManager.resolveService(), який керує процесом подальшої роботи алгоритму.

Метод onServiceLost (serviceInfo: NsdServiceInfo) викликається, коли сервіс стає недоступний. Виконується логування повідомлення, щодо недоступного сервісу.

У методах onDiscoveryStopped (serviceType: String), onStartDiscoveryFailed (serviceType: String, errorCode: Int) та onStopDiscoveryFailed (serviceType: String, errorCode: Int) здійснюється логування повідомлень, щодо недоступних сервісів та виконуються відповідні дії для обробки помилки.

resolveListener є об'єктом, який реалізує інтерфейс NsdManager.ResolveListener: Метод onResolveFailed (serviceInfo: NsdServiceInfo, errorCode: Int) викликається, якщо виникла при спробі з’єднати пристрої, виконується логування повідомлення про помилку.

Важливо зазначити, що метод onServiceResolved (serviceInfo: NsdServiceInfo) є ключовим, оскільки при успішному отриманні інформації про сервіс - викликається метод viewModel.startClientSocket(), який встановлює з'єднання з сервісом (в нашому випадку з пристроєм-сервером).

Для демонстрації можливостей автентифікації за алгоритмом NSD, необхідні наступні кроки:

1) Підготовка двох Android-пристроїв: потрібно мати доступ до двох фізичних Android-пристроїв, які будуть використовуватись для демонстрації.

2) Налаштування Wi-Fi мережі: впевнитись, що обидва пристрої підключені до однієї Wi-Fi мережі, в якій вони можуть спілкуватись між собою.

3) Запуск застосунку: запустити розроблений застосунок на обох пристроях.

4) Пошук і вибір пристрою: за допомогою застосунку на кожному пристрої здійсніть пошук доступних пристроїв в мережі.

5) Автентифікація та з’єднання: застосунок перевіряє обидва пристрої на наявність правильних автентифікаційних даних та забезпечує безпечне з’єднання між ними.

В результаті цих кроків, демонстрація можливостей автентифікації алгоритму показує встановлення безпечного з’єднання через Wi-Fi мережу між ними та забезпечує обмін даними між ними, після обміну даними між пристроями буде передано/отримано тестове повідомлення «Hello from client» рисунку 4.13 та «Hello from server» рисунку 4.14 відповідно:

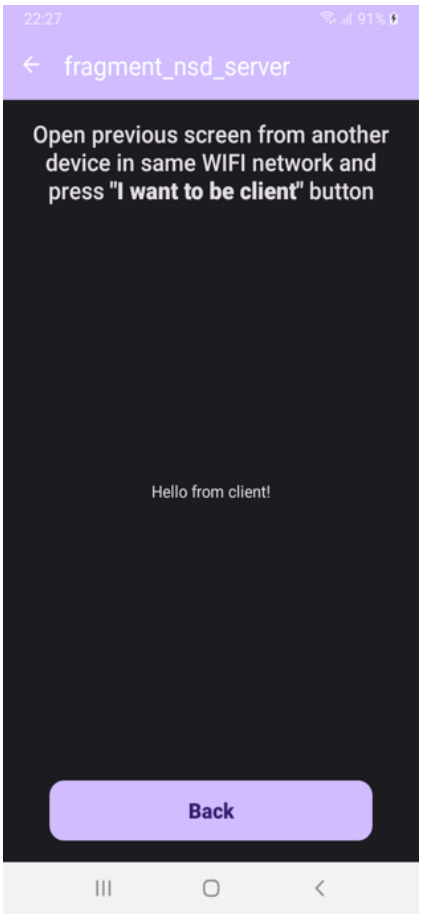

Рисунок 4.13 – Пристрій сервер

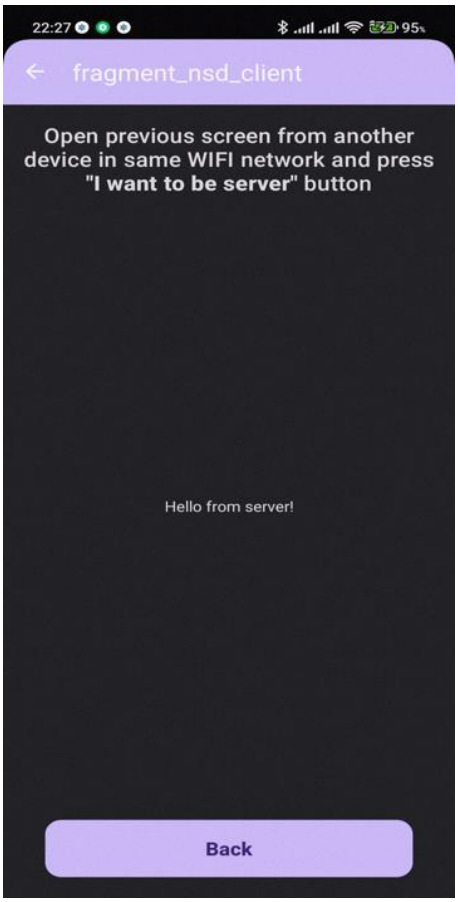

Рисунок 4.14 – Пристрій клієнт

Код реалізації логіки яка отримує від пристрою-сервера повідомлення представлений у Додатку Г.

4.3. Розробка UI для алгоритму QR-code

В даному підрозділі буде детально розглянуто процес розробки інтерфейсу користувача для алгоритму QR-коду додатка AuthGuard. Цей процес включає створення інтерфейсу користувача, який підтримує різні розміри екранів, а також відповідних обробників подій для керування взаємодією користувача з додатком.

Для реалізації інтерфейсу користувача було створено клас QrCodeGenerationFragment, який є фрагментом у представленому застосунку Android. Зовнішній вигляд екрана зображений на рисунку 4.15.

Рисунок 4.15 – Вигляд екрана QrCodeGenerationFragment

Основний інтерфейс користувача цього фрагменту включає кнопку для збереження QR-коду, область для відображення стану завантаження, і, звичайно, область для відображення самого QR-коду. Для зручності користувача, повідомлення збереження відображається за допомогою Snackbar.

Крім того, було додано кнопку для запуску сканування QR-коду, що містити роботу з дозволами на камеру [33]. У разі відсутності необхідних дозволів, користувач буде показано відповідне вікно з проханням надати ці дозволи, процес отримання яких зображений на рисунку 4.16.

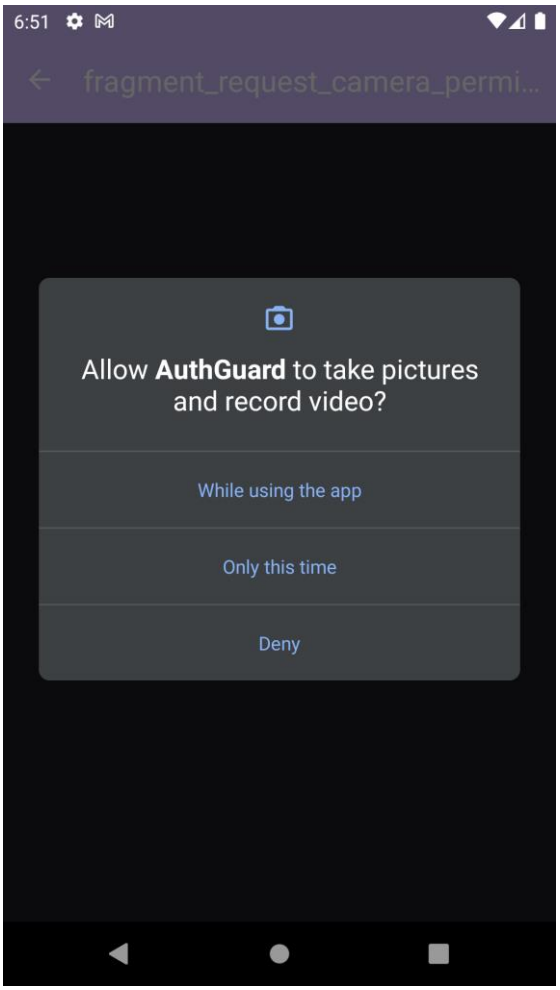

Рисунок 4.16 – Вигляд екрана на якому отримується дозвіл на камеру

На рисунку 4.17 та рисунку 4.18 я представлю короткий код розмітки структури основних елементів цього екрану, а саме елементи FrameLayout з ідентифікатором qrCodeLayout - це контейнер, який містить кілька елементів: ProgressBar, CardView та ImageView. FrameLayout призначений для показу одного елемента на екрані, але в цьому випадку він використовується для накладання елементів один на одного. ProgressBar відображає індикатор завантаження, поки QR-код генерується. CardView використовується для відображення QR-коду в візуально привабливих рамках ImageView.

```
<FrameLayout
    android:id="@+id/qrCodeLayout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <ProgressBar
        android:id="@+id/spinner"
        android:layout_width="96dp"
        android:layout_height="96dp"
        android:layout_gravity="center_horizontal|center_vertical"
        android:indeterminate="true" />

    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center_horizontal|center_vertical"
        app:cardCornerRadius="16dp">

        <ImageView
            android:id="@+id/imgQrCode"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:contentDescription="barcode"
            tools:src="@tools:sample/avatars" />

    </androidx.cardview.widget.CardView>
</FrameLayout>
```

Рисунок 4.17 – Розмітка місця зображення QR коду

```
<Button
    android:id="@+id/btnScanQrCode"
    style="@style/Widget.AuthGuardian.Button.Material"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal|bottom"
    android:text="@string/i_want_scan"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />
```

Рисунок 4.18 – Кнопка сканування QR коду

4.4. Розробка бізнес-логіки для алгоритму QR-code

Розробка бізнес-логіки для алгоритму QR-коду в проєкт є однієї із головних компонентів функціональності додатка. Ключовий аспект цього процесу – використання Kotlin Coroutine, асинхронної конструкції, яка покликана оптимізувати потік роботи з фоновими задачами. Важливо зазначити, що асинхронність корутин не означає складності або заплутаності коду – навпаки, вона надає можливість виконувати асинхронний код так, ніби він послідовний. Це призводить до значного полегшення читання та розуміння коду.

Почнемо з того, що створюється нова корутина в CoroutineScope. Запуск відбувається в потоці Dispatchers.IO, що є оптимальним варіантом для виконання завдань вводу/виводу, таких як мережеві операції чи читання/запис даних на диск. Щойно нова корутина запущена, стан завантаження (в нашому випадку, це _isLoading) встановлюється як "активний" (true). Це дозволяє надсилати сигнал користувачеві про те, що процес генерації QR-коду розпочато. Після цього вводиться затримка на одну секунду (1000 мілісекунд), яка симулює те, що процес створення QR-коду може зайняти певний час. Після цього визначається розмір QR-коду (512x512 пікселів) і формується вміст QR-коду. Цей вміст включає дані про Wi-Fi мережу, а саме SSID (ім'я мережі) і пароль. Ці дані отримуються з DataStoreRepository, що є джерелом даних [34].

Використовуючи клас QRCodeWriter, вміст QR-коду перетворюється в матрицю бітів. Кожен біт в цій матриці представляє конкретний піксель у QR-коді.

Короткий код алгоритму який асинхронно генерує QR-код зображений на рисунку 4.19.

```
private fun generateQrCode() {
    CoroutineScope(Dispatchers.IO).launch { this: CoroutineScope
        _isLoading.postValue( value: true)
        delay( timeMillis: 1000)
        val size = 512
        val qrCodeContent = "WIFI:S:${dataStoreRepository.getString(USER_SSID) ?: ssid}" +
                ";T:WPA;P:${dataStoreRepository.getString(USER_PASSWORD) ?: password};;"
        val bits = QRCodeWriter().encode(qrCodeContent, BarcodeFormat.QR_CODE, size, size)
        _qrCode.postValue(Bitmap.createBitmap(size, size, Bitmap.Config.RGB_565).also { it: Bitmap!
            for (x in 0 ≤ until < size) {
                for (y in 0 ≤ until < size) {
                    it.setPixel(x, y, if (bits[x, y]) Color.BLACK else Color.WHITE)
                }
            }
        })
        _isLoading.postValue( value: false)
    }
}
```

Рисунок 4.19 – Алгоритм генерації QR-коду

Далі виконується перетворення цієї матриці бітів на об'єкт Bitmap, який може бути відображений в UI. Якщо біт в матриці має значення true, відповідний піксель на Bitmap встановлюється як чорний. В іншому випадку, піксель буде білим.

Готовий Bitmap з QR-кодом публікується через LiveData _qrCode, що дозволяє UI негайно відреагувати на зміни та відобразити новий QR-код [35].

В кінці генерації QR-коду, стан завантаження (_isLoading) встановлюється в false і публікується. Це сигналізує про завершення процесу генерації QR-коду.

Таким чином, весь код, що описує алгоритм генерації та бізнес-логіку створення QR-кодів, є відображенням простоти та ефективності асинхронного програмування з використанням Kotlin Coroutine.

Повний код алгоритму генерації бізнес-логіки зображений у Додатку Д.

Висновки за розділом 4

У рамках 4-го розділу при розробці додатка AuthGuard було реалізовано ряд ключових технологій, кожна з яких сприяє покращенню взаємодії користувача з додатком, його зручності та ефективності.

У процесі розробки та імплементації алгоритму QR-коду для додатка AuthGuard було виявлено, що даний процес є багатогранним та містити ряд важливих етапів. Спочатку було ретельно продумано й розроблено інтерфейс користувача, при цьому було взято до уваги, що інтерфейс повинен підтримувати різні розміри екранів і забезпечувати користувачеві зручність взаємодії з додатком. Підхід до розробки інтерфейсу був з особливою увагою, через те, що від якості його виконання великою мірою залежить оцінка користувачем додатка в цілому.

Було створено клас QrCodeGenerationFragment, що відповідає за візуалізацію QR-коду. Даний клас включає кнопку для збереження QR-коду, область для відображення стану завантаження та область для відображення самого QR-коду. Така структура дозволяє користувачеві легко взаємодіяти з додатком і отримувати необхідну інформацію.

При розробці бізнес-логіки для алгоритму QR-коду було використано Kotlin Coroutine, який забезпечує асинхронне виконання завдань. За допомогою Kotlin Coroutine забезпечується ефективне виконання завдань, що відбувається в фоновому режимі, не навантажуючи основний потік виконання програми. Це особливо важливо при роботі з великими обсягами даних або з вимогами до високої швидкості реакції на дії користувача.

Алгоритм генерації QR-коду було реалізовано з урахуванням всіх потреб користувачів, зокрема щодо швидкості генерації та зручності використання. Використання Kotlin Coroutine дозволило значно спростити структуру коду, при цьому забезпечуючи високу швидкість і ефективність його виконання.

Таким чином, четвертий розділ присвячений детальному опису процесу розробки інтерфейсу користувача і бізнес-логіки для алгоритму QR-коду додатка AuthGuard. Описано основні складові інтерфейсу, їх роль і місце в системі, а також принципи організації асинхронного виконання завдань за допомогою Kotlin Coroutine. Це забезпечує гнучкість, ефективність і високу продуктивність додатка, що дозволяє йому успішно виконувати свої функції.

Також в даному розділі була впроваджена технологія Google Sign-In. Його використання в AuthGuard дає користувачам можливість авторизуватися в

застосунку, використовуючи свої вже наявні облікові записи Google, що є набагато зручнішим і безпечнішим, ніж створення нового облікового запису. Це знижує поріг входу для користувачів і збільшує швидкість авторизації, що, своєю чергою, поліпшує загальний досвід користувача. Використання API Google Sign-In також забезпечує додатковий рівень безпеки, оскільки Google забезпечує ряд заходів щодо захисту ідентифікаційних даних користувачів.

Ще однією технологією, що була впроваджена, є NSD. Вона використовується для автоматичного виявлення сервісів в локальній мережі без необхідності ручного введення IP-адреси. Використовуючи NSD, застосунок може виявляти та взаємодіяти з іншими пристроями в локальній мережі. Це робить застосунок більш гнучким та зручним для користувача, особливо в домашньому середовищі, де можуть бути різні пристрої, які підключаються до однієї та тієї ж мережі.

Таким чином, в рамках цього розділу, було виконано важливу роботу щодо реалізації Google Sign-In та Network Service Discovery в застосунку AuthGuard. Разом із розробкою і впровадженням алгоритму QR-коду ці технології формують основу функціональності застосунку, роблячи його не тільки потужним інструментом для захисту даних, але й зручним для кінцевого користувача.

ВИСНОВКИ

У даній кваліфікаційній роботі було досліджено методи автентифікації у мобільних додатках та проаналізовано характеристики їх алгоритмів. Для демонстрації переваг та недоліків створено мобільний застосунок, де було впроваджено найсучасніші підходи до створення користувацького інтерфейсу, побудову бізнес-логіки та забезпечення безпеки даних.

Автентифікація Google Sign-in була інтегрована з метою забезпечити надійну та безпечну автентифікацію користувачів. Ця технологія дозволила забезпечити безпеку персональних даних користувача, підвищуючи при цьому користувацький досвід і підтримуючи стандарти Google.

Network Service Discovery – це технологія, що автоматизує процес виявлення та підключення до доступних пристроїв в локальній мережі, дозволяє автентифікувати користувача без його прямої участі в процесі, а це, в свою чергу, збільшує ефективність та зручність користування застосунком.

Додатковим, та доволі зручним методом автентифікації, є алгоритм QR-коду, який використовує технологію Kotlin Coroutines. Він дозволяє провести автентифікацію без використання двох попередніх методів, але для цього потрібна наявність камери в мобільному пристрої.

При розробці інтерфейсу користувача, було враховано низку критеріїв: зручність, інтуїтивність, адаптивність до різних розмірів екранів і пристроїв, щоб забезпечити користувачам найкращий досвід використання додатка.

Отже, в ході реалізації було показано, що жоден із наведених методів повністю не задовольняє всіх вимог, що висуваються до сучасних мобільних застосунків (простота використання, багатофункціональність, потреба в камері та локальній мережі, залежність від 3-ої сторони). Тому, рекомендується застосувати зазначені методи комплексно, як такі, що доповнюють один одного.

Основні результати кваліфікаційної роботи пройшли апробацію у вигляді наукової статті (Scopus).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bohdan Oliinyk, Yurii Shcheblanin, Oleg Kurchenko, Toroshanko Oleksandr and Nataliia Korshun. Research of Authentication Methods in Mobile Applications. CPITS-2023: Cybersecurity Providing in Information and Telecommunication Systems, February 28, 2023, Kyiv, Ukraine, vol 3421. pp. 266-271.

2. Juniper Research: Digital Therapeutics and Wellness App Users to Reach 1.4 Billion Globally by 2025, as Pandemic Accelerates Regulatory Acceptance [Електронний ресурс] – Режим доступу до ресурсу: https://www.businesswire.com/news/home/20201027005102/en/Juniper-Research-Digital-Therapeutics-and-Wellness-App-Users-to-Reach-1.4-Billion-Globally-by-2025-as-Pandemic-Accelerates-Regulatory-Acceptance.

3. The Advantages and Disadvantages of Single-Factor Authentication [Електронний ресурс] – Режим доступу до ресурсу: https://zappedia.com/single-factor-authentication/.

4. Improving Security with Multi-factor Authentication, Data Encryption, Private Networks and Firewalls [Електронний ресурс] – Режим доступу до ресурсу: https://www.defense.com/uk/blog/what-is-multi-factor-authentication/.

5. Usernames and Passwords: An Illusion of Security [Електронний ресурс] – Режим доступу до ресурсу: https://www.thinkcsc.com/usernames-and-passwords-an-illusion-of-security/.

6. Mobile Application Security: Definition, Practices, Benefits [Електронний ресурс] – Режим доступу до ресурсу: https://www.intellectsoft.net/blog/mobile-application-security-definition-practices-benefits/.

7. The Pros and Cons of Two-Factor Authentication Types and Methods [Електронний ресурс] – Режим доступу до ресурсу: https://www.makeuseof.com/tag/pros-cons-2fa-types-methods/.

8. Phillips B. Android Programming: The Big Nerd Ranch Guide, 3nd Edition / B. Phillips, K. Marsicano, C. Stewart., 2017. – 624 с. – (Big Nerd Ranch Guides).

9. Use network service discovery [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/training/connect-devices-wirelessly/nsd.

10. Connect Two Android Devices over WiFi using Network Service Disocvery [Електронний ресурс] – Режим доступу до ресурсу: https://jayrambhia.com/blog/android-wireless-connection-1.

11. Android NSD Overview [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/reference/android/net/nsd/package-summary.

12. Gunasekera S. Android Apps Security / Sheran Gunasekera., 2012.

13. Sirapat B. Authentication and Access Control: Practical Cryptography Methods and Tools / Boonkrong Sirapat., 2020. – 248 с.

14. Chestnykh D. Password Authentication for Web and Mobile Apps: The Developer's Guide To Building Secure User Authentication / Dmitry Chestnykh.. – 142 с.

15. The Executive’s Guide To Social Login, 2015.

16. Annuzzi J. ntroduction to Android Application Development: Android Essentials / J. Annuzzi, L. Darcey, C. Shane., 2013. – 672 с.

17. Material Design [Електронний ресурс] – Режим доступу до ресурсу: https://m3.material.io/.

18. Enterprise Application Patterns Using .NET MAUI [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/architecture/maui/.

19. Eckel B. Atomic Kotlin / B. Eckel, S. Isakova., 2021. – 636 с.

20. Moskała M. Kotlin Coroutines: Deep Dive (Kotlin for Developers) / Marcin Moskała., 2022. – 470 с.

21. Vasconcelos H. Asynchronous Android Programming Second Edition / Helder Vasconcelos., 2016. – 394 с. – (Revised edition; кн. 2).

22. Moroney L. The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform / Laurence Moroney., 2017. – 288 с.

23. Firebase Authentication [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/auth.

24. Cloud Firestore [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore.

25. Firebase Cloud Messaging [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/cloud-messaging.

26. Cloud Storage for Firebase [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/storage.

27. Firebase Hosting [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/hosting.

28. Get started with Google Analytics [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/analytics/get-started?platform=web.

29. Firebase Crashlytics [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/crashlytics.

30. Muschko B. Gradle in Action / B. Muschko, H. Dockter., 2014. – 480 с.

31. Google Auth Module [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/help/space/google-auth-module.html#google-auth-module-settings.

32. NsdManager [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/reference/android/net/nsd/NsdManager.

33. Smyth N. Android Studio Flamingo Essentials - Java Edition: Developing Android Apps Using Android Studio 2022.2.1 and Java / Neil Smyth., 2023. – 816 с.

34. Permissions on Android [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://developer.android.com/guide/topics/permissions/overview.

35. Saving Data on Android (Second Edition): Learn Jetpack DataStore, Room, Firebase & SQLite with Kotlin / [R. Tutorial Team, J. Bailey, D. Djermanović та ін.]., 2021. – 344 с.

36. Sharma P. LiveData Tutorial for Android: Deep Dive [Електронний ресурс] / Prateek Sharma. – 2020. – Режим доступу до ресурсу: https://www.kodeco.com/10391019-livedata-tutorial-for-android-deep-dive.

ДОДАТКИ

ДОДАТОК А
СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Тези наукових доповідей:

1. Bohdan Oliinyk, Yurii Shcheblanin, Oleg Kurchenko, Toroshanko Oleksandr and Nataliia Korshun. Research of Authentication Methods in Mobile Applications. CPITS-2023: Cybersecurity Providing in Information and Telecommunication Systems, February 28, 2023, Kyiv, Ukraine. SEUR-WS.org, vol 3421. pp. 266-271.

ДОДАТОК Б

Код алгоритму автентифікації Google Sign-in

```
class SocialLoginFragment: Fragment(), View.OnClickListener {
        private val viewModel: SocialLoginViewModel by viewModels()
        private var _binding: FragmentSocialLoginBinding? = null
        private val binding get() = _binding!!
        private var googleSignInClient: GoogleSignInClient? = null
        private var resultLauncher = registerForActivityResult (
ActivityResultContracts.StartActivityForResult() ) { result ->
        if (result.resultCode == Activity.RESULT_OK) {
                val task = GoogleSignIn.getSignedInAccountFromIntent(result.data)
                handleSignInResult(task)
                viewModel.checkIfUserAlreadySignedIn(requireContext())
        }
}
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View {
        _binding = FragmentSocialLoginBinding.inflate(inflater, container, false)
        return binding.root
}
override fun onStart() {
        super.onStart()
        viewModel.checkIfUserAlreadySignedIn(requireContext())
}
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        viewModel.init()
        val googleSignIn =
        GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN )
                .requestEmail()
                .build()
        googleSignInClient = GoogleSignIn.getClient(requireActivity(), googleSignIn)
        with(binding) {
                viewModel.isGoogleSignInAvailable.observe(viewLifecycleOwner) {
                isGoogleSignInAvailable ->
                if (isGoogleSignInAvailable) {
                with(googleSignInButton) {
                visibility = View.VISIBLE
                setSize(SignInButton.SIZE_ICON_ONLY)
        }
        googleSignOutButton.visibility = View.GONE
} else {
```

```
        googleSignInButton.visibility = View.GONE
        googleSignOutButton.visibility = View.VISIBLE
        }
}
        googleSignInButton.setOnClickListener(this@SocialLoginFragment)
        googleSignOutButton.setOnClickListener(this@SocialLoginFragment)
}
}
override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
}
override fun onClick(view: View?) {
        when (view?.id) {
                R.id.googleSignInButton -> { onGoogleSignInClicked() }
                R.id.googleSignOutButton -> { onGoogleSignOutClicked() }
        }
}
private fun onGoogleSignInClicked() {
        viewModel.onGoogleSignInClicked()
        val signInIntent: Intent = googleSignInClient!!.signInIntent
        resultLauncher.launch(signInIntent)
}
private fun onGoogleSignOutClicked() {
        viewModel.onGoogleSignOutClicked()
        googleSignInClient?.signOut()?.addOnCompleteListener(requireActivity()) {
        viewModel.checkIfUserAlreadySignedIn(requireContext())
}
}
private fun handleSignInResult(completedTask: Task<GoogleSignInAccount>) {
        try {
                NavigationUtils.navigate(findNavController(),
        SocialLoginFragmentDirections.actionSocialLoginFragmentToGoogleSignSuccessf
ulFragment())
        } catch (e: ApiException) {
                // The ApiException status code indicates the detailed failure reason.
                // Please refer to the [GoogleSignInStatusCodes] class reference for more
information.
                Snackbar.make(binding.root,"signInResult:failed code=" +
        e.statusCode,Snackbar.LENGTH_SHORT).show()
                }
        }
}
```

ДОДАТОК Г

Код отримання повідомлення від сервера

```
fun startClientSocket(serviceAddress: InetAddress, servicePort: Int) {
        _isLoading.postValue(true)
        viewModelScope.launch(Dispatchers.IO) {
        delay(1000) // only for testing
        try {
                val socket = Socket(serviceAddress, servicePort)
                val input = BufferedReader(InputStreamReader(socket.getInputStream()))
                val output = PrintWriter(socket.getOutputStream(), true)
                output.println("Hello from client!")
                val receivedData = input.readLine()
                withContext(Dispatchers.Main) {
                        _receivedMessage.postValue(receivedData)
                        _isLoading.postValue(false)
                }
        Log.d("NSD", "Received data from server: $receivedData")
        socket.close()
        } catch (e: IOException) {
                withContext(Dispatchers.Main) {
                _isLoading.postValue(false)
        }
        Log.e("NSD", "Client socket error: ", e)
                }
        }
}
```

ДОДАТОК Д

Код алгоритму генерації QR-code та бізнес-логіка

```
class QrCodeGenerationViewModel(
        application: Application,
        private val dataStoreRepository: DataStoreRepository
) : AndroidViewModel(application) {
// region LiveData
        private var _qrCode: MutableLiveData<Bitmap> = MutableLiveData()
        val qrCode: LiveData<Bitmap> = _qrCode
        private var _isLoading = MutableLiveData(false)
        val isLoading: LiveData<Boolean> = _isLoading
        private var _savedMessage: MutableLiveData<String> = MutableLiveData()
        val savedMessage: LiveData<String> = _savedMessage
// endregion
// region Data
        private var ssid: String? = null
        private var password: String? = null
        private var isInited: Boolean = false
// endregion
fun init(ssid: String?, password: String?): Boolean = when {
        isInited -> true
        else -> {
                this@QrCodeGenerationViewModel.ssid = ssid
                this@QrCodeGenerationViewModel.password = password
        sendEvent(AnalyticsEventScreen.QR_CODE_GENERATION_SCRN__VIEW.valu
e)
        if (ssid != null && password != null) {
                generateQrCode()
        }
        isInited = true
        true
        }
}
private fun generateQrCode() {
        CoroutineScope(Dispatchers.IO).launch {
                _isLoading.postValue(true)
                delay(1000) // only for demonstration
                val size = 512
                val qrCodeContent = "WIFI:S:${dataStoreRepository.getString(USER_SSID)
?: ssid};T:WPA;P:${dataStoreRepository.getString(USER_PASSWORD) ?: password};;"
val bits = QRCodeWriter().encode(qrCodeContent, BarcodeFormat.QR_CODE, size, size)
```

```
        _qrCode.postValue(Bitmap.createBitmap(size, size, Bitmap.Config.RGB_565).also
{
        for (x in 0 until size) {
                for (y in 0 until size) {
                        it.setPixel(x, y, if (bits[x, y]) Color.BLACK else Color.WHITE)
                }
        }
})
_isLoading.postValue(false)
        }
}
fun saveUserData() {
        viewModelScope.launch {
                if (dataStoreRepository.getString(USER_SSID) == null &&
                        dataStoreRepository.getString(USER_PASSWORD) == null) {
                        dataStoreRepository.apply {
                        putString(USER_SSID, ssid!!)
                        putString(USER_PASSWORD, password!!)
        }
_savedMessage.postValue(getApplication<Application>().getString(R.string.qr_code_sav
ed))
                }
        }
}
companion object {
        private const val USER_SSID = "USER_SSID"
        private const val USER_PASSWORD = "USER_PASSWORD"
        private val TAG = QrCodeGenerationViewModel::class.java.simpleName
        }
}
```