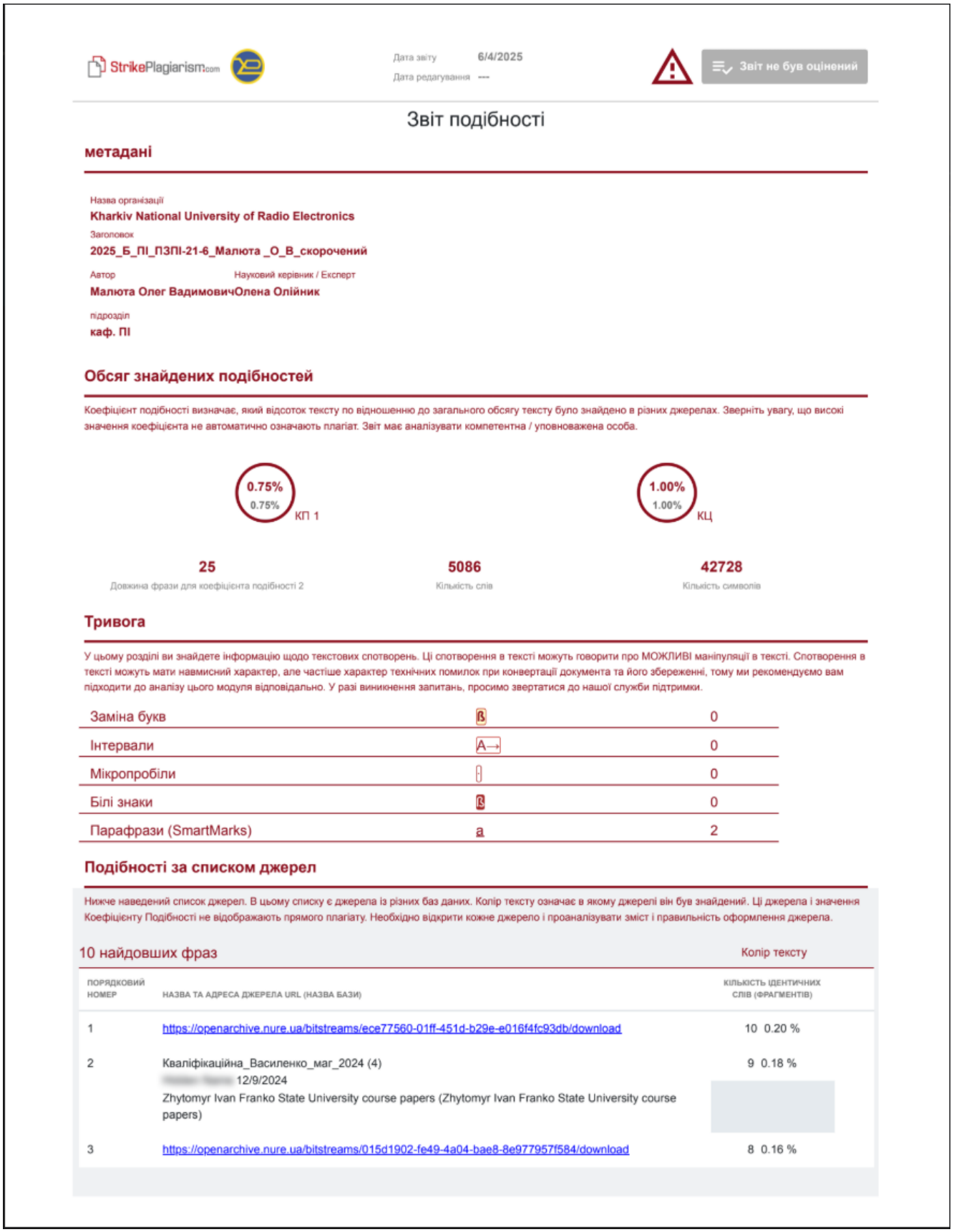
ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 6/4/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_Б_ПІ_ПЗПІ-21-6_Малюта _О_В_скорочений
Автор Науковий керівник / Експерт
Малюта Олег ВадимовичОлена Олійник
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

0.75% 0.75% КП 1

1.00% 1.00% КЦ

25 Довжина фрази для коефіцієнта подібності 2
5086 Кількість слів
42728 Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	0
Інтервали	A→	0
Мікропробіли	¶	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	2

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download	10 0.20 %
2	Кваліфікаційна_Василенко_маг_2024 (4) [redacted] 12/9/2024 Zhytomyr Ivan Franko State University course papers (Zhytomyr Ivan Franko State University course papers)	9 0.18 %
3	https://openarchive.nure.ua/bitstreams/015d1902-fe49-4a04-bae8-8e977957f584/download	8 0.16 %

4	Кваліфікаційна_Василенко_маг_2024 (4) [redacted] 12/9/2024 Zhytomyr Ivan Franko State University course papers (Zhytomyr Ivan Franko State University course papers)	6 0.12 %
5	Кваліфікаційна_Василенко_маг_2024 (4) [redacted] 12/9/2024 Zhytomyr Ivan Franko State University course papers (Zhytomyr Ivan Franko State University course papers)	5 0.10 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.39 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Кваліфікаційна_Василенко_маг_2024 (4) [redacted] 12/9/2024 Zhytomyr Ivan Franko State University course papers (Zhytomyr Ivan Franko State University course papers)	20 (3) 0.39 %

з Інтернету (0.35 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://openarchive.nure.ua/bitstreams/ece77560-01ff-451d-b29e-e016f4fc93db/download	10 (1) 0.20 %
2	https://openarchive.nure.ua/bitstreams/015d1902-fe49-4a04-bae8-8e977957f584/download	8 (1) 0.16 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

1
ВСТУП
Сучасний ринок мобільних додатків пропонує широкий спектр рішень для організації повсякденного життя, включаючи управління гардеробом. Однак багато з існуючих рішень мають обмежену функціональність, потребують постійного підключення до Інтернету або не забезпечують достатньо гнучких інструментів для аналізу гардеробу. Це обумовлює актуальність розробки нового мобільного додатку, який би поєднував зручність використання, широкий функціонал та можливість роботи в офлайн-режимі.
Метою даної роботи є створення мобільного додатку, призначеного для управління гардеробом, який дозволить користувачам: додавати та редагувати елементи одягу з детальним описом (категорія, сезонність, колір, матеріал тощо); генерувати комбінації одягу на основі вподобань користувача, погодних умов, призначення та стилю; аналізувати статистику використання гардеробу (наприклад, кількість речей за категоріями, середній вік елементів, розподіл за матеріалами); працювати з додатком без підключення до Інтернету (у гостьовому режимі).
Основним завданням розробки додатку включає створення інтуїтивного інтерфейсу для зручного управління елементами гардеробу, реалізацію розумних алгоритмів автоматичного підбору образів, розробку детальної статистики використання гардеробу, забезпечення стабільної офлайн-роботи та оптимізацію продуктивності для різноманітних пристроїв.

ДОДАТОК Б

Слайди презентації

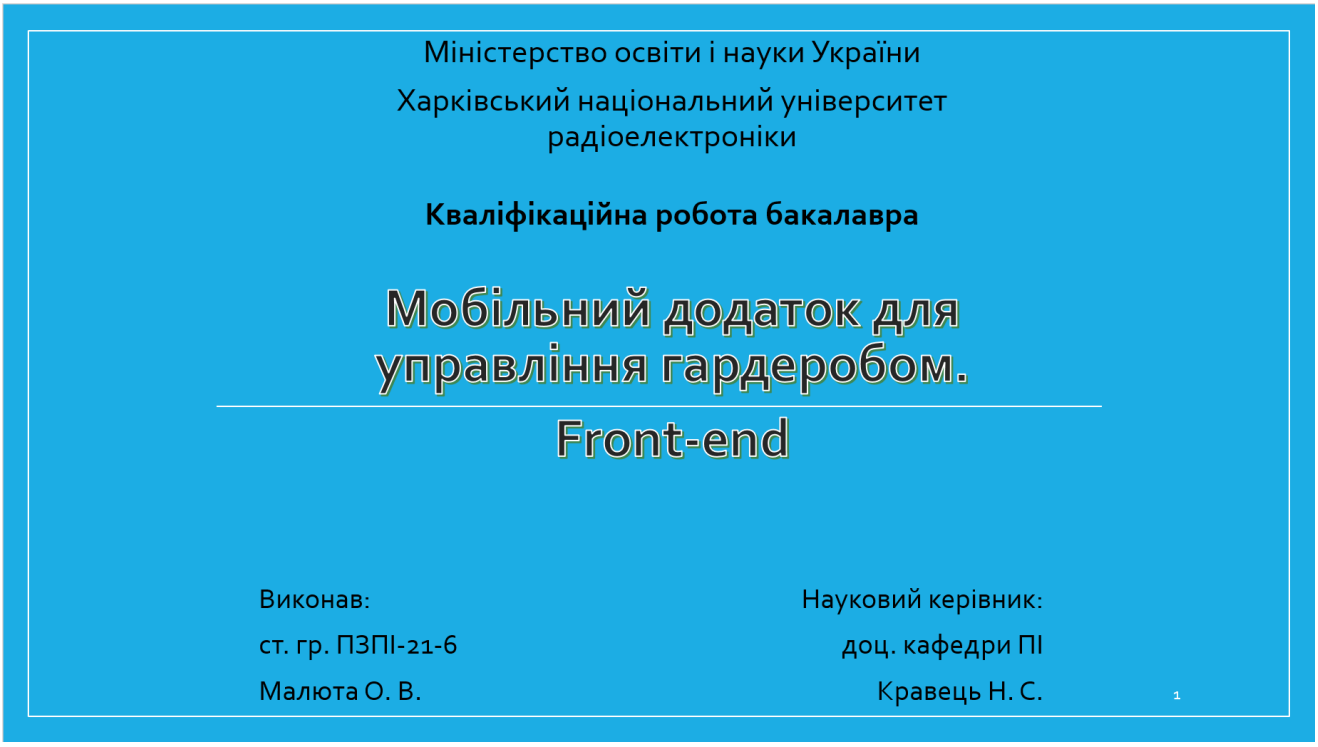


Рисунок Б.1 – Слайд презентації №1 (рисунок виконано самостійно)

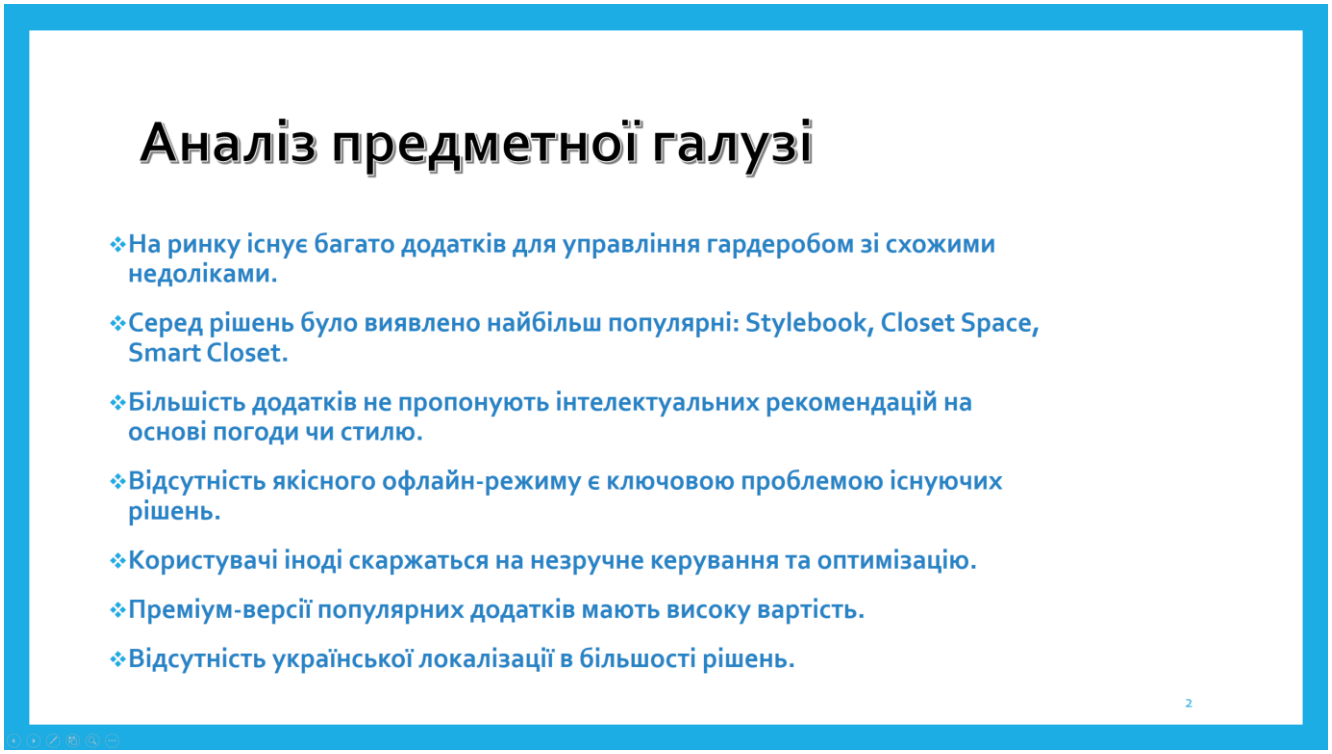


Рисунок Б.2 – Слайд презентації №2 (рисунок виконано самостійно)

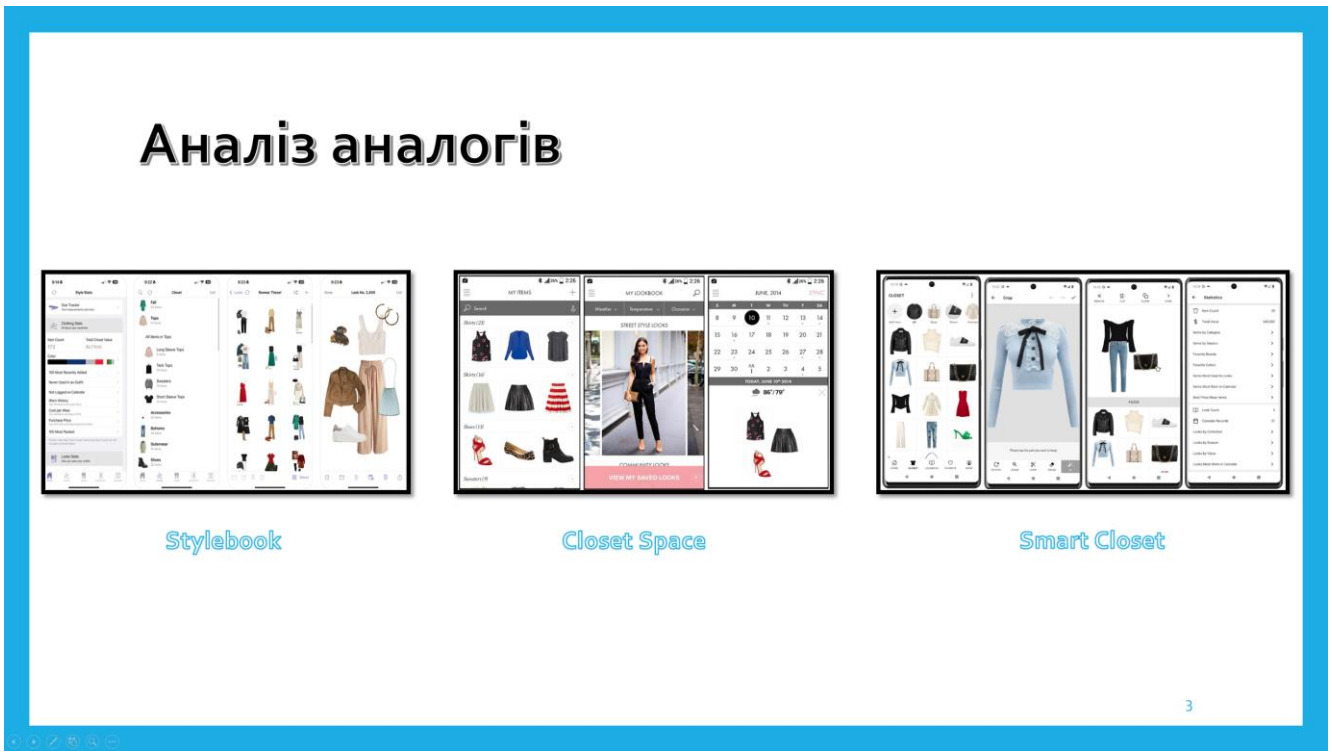


Рисунок Б.3 – Слайд презентації №3 (рисунок виконано самостійно)

Додаток	Переваги	Недоліки
Stylebook	Детальна каталогізація одягу	Немає автоматичних рекомендацій
	Календар носіння	Обмежена безкоштовна версія
Closet Space	Простий інтерфейс	Не враховує погоду
	Синхронізація між пристроями	Обмежена підтримка мов
Smart Closet	Використання ШІ для підбору образів	Висока ціна преміум-версії
	Детальна статистика	Високі вимоги до пристрою

Рисунок Б.4 – Слайд презентації №4 (рисунок виконано самостійно)

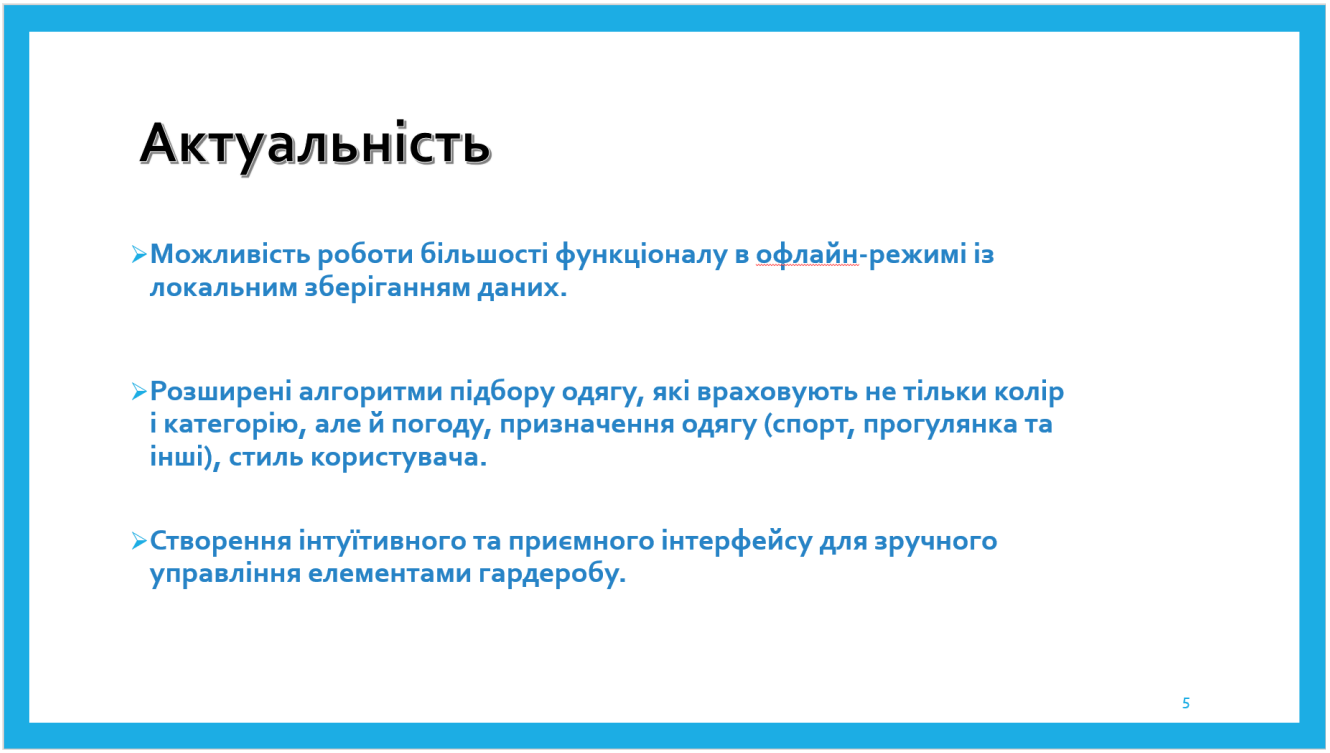


Рисунок Б.5 – Слайд презентації №5 (рисунок виконано самостійно)

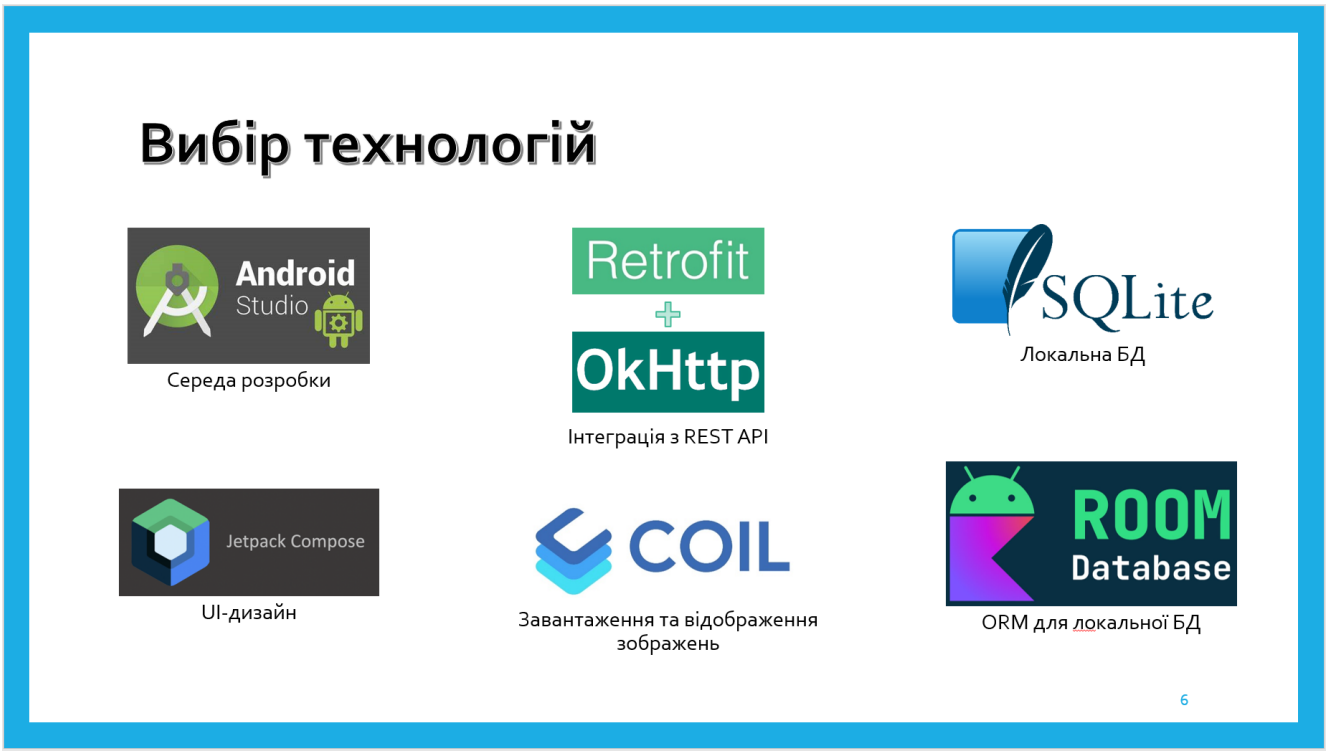


Рисунок Б.6 – Слайд презентації №6 (рисунок виконано самостійно)

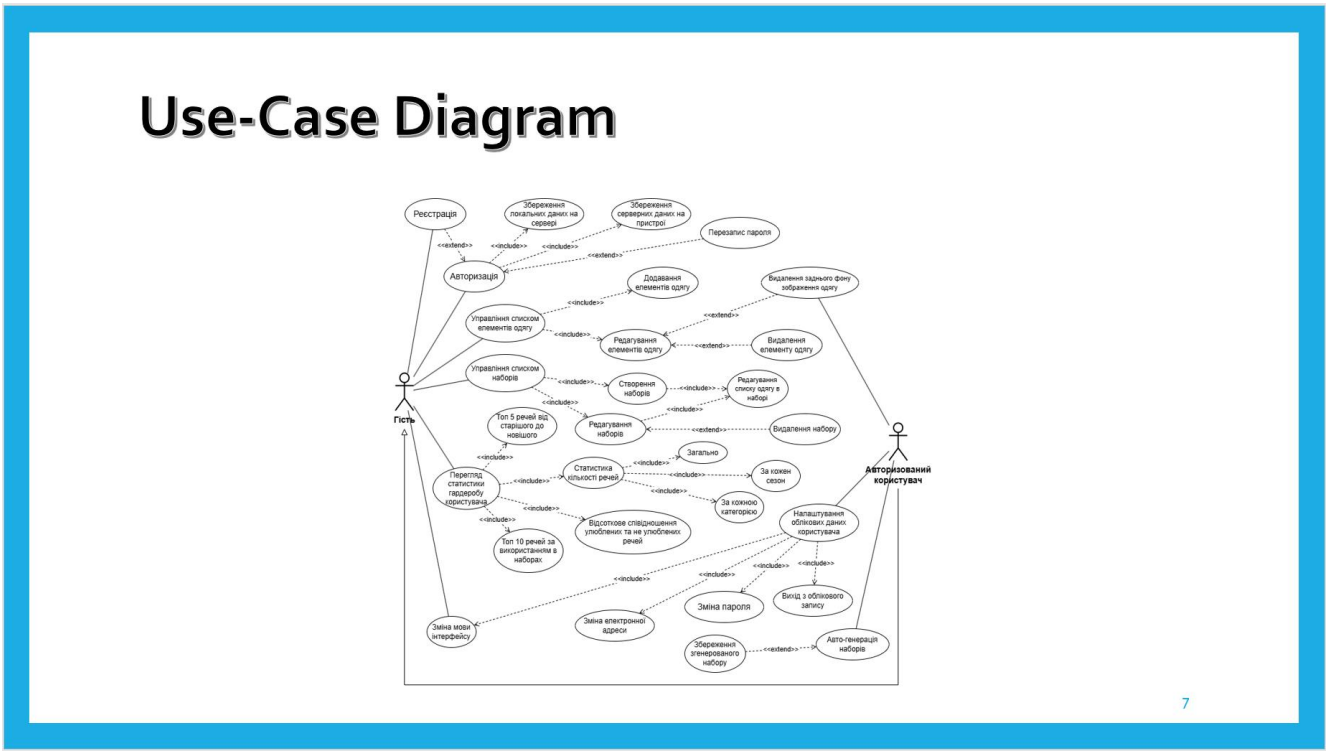


Рисунок Б.7 – Слайд презентації №7 (рисунок виконано самостійно)

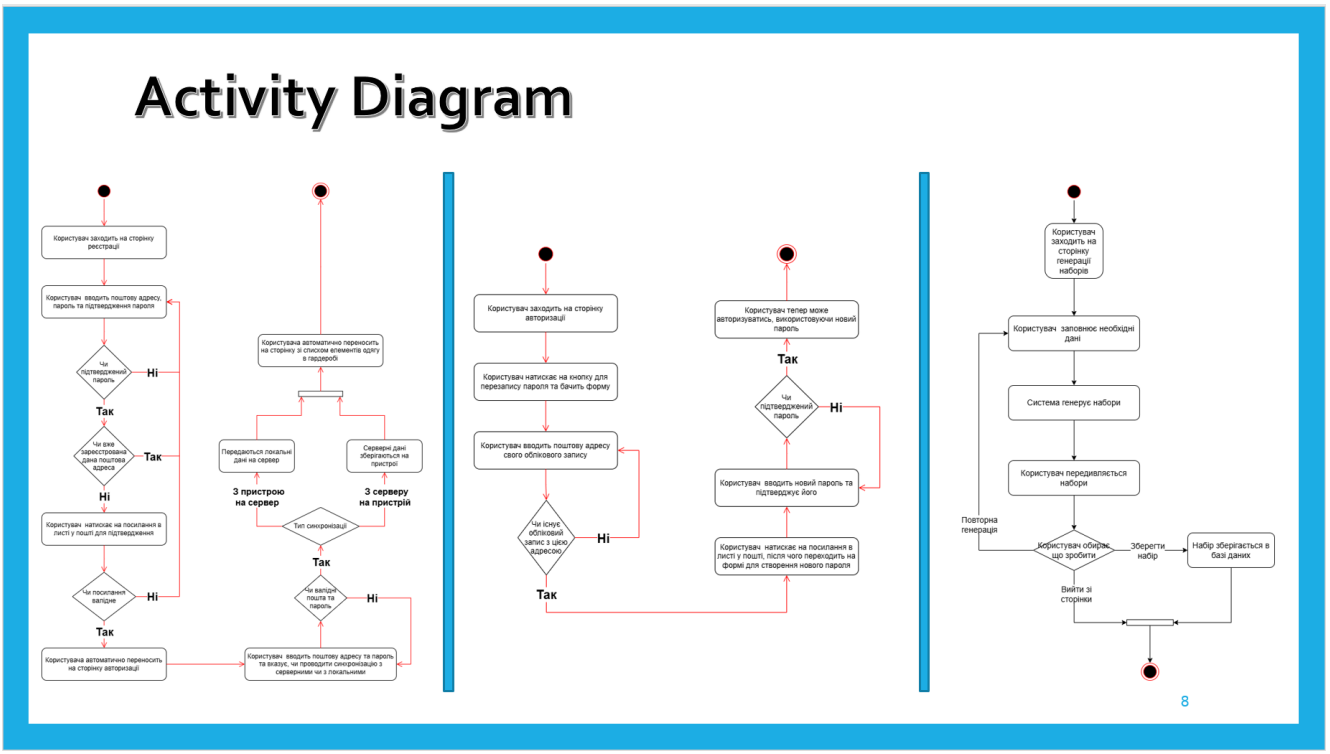


Рисунок Б.8 – Слайд презентації №8 (рисунок виконано самостійно)

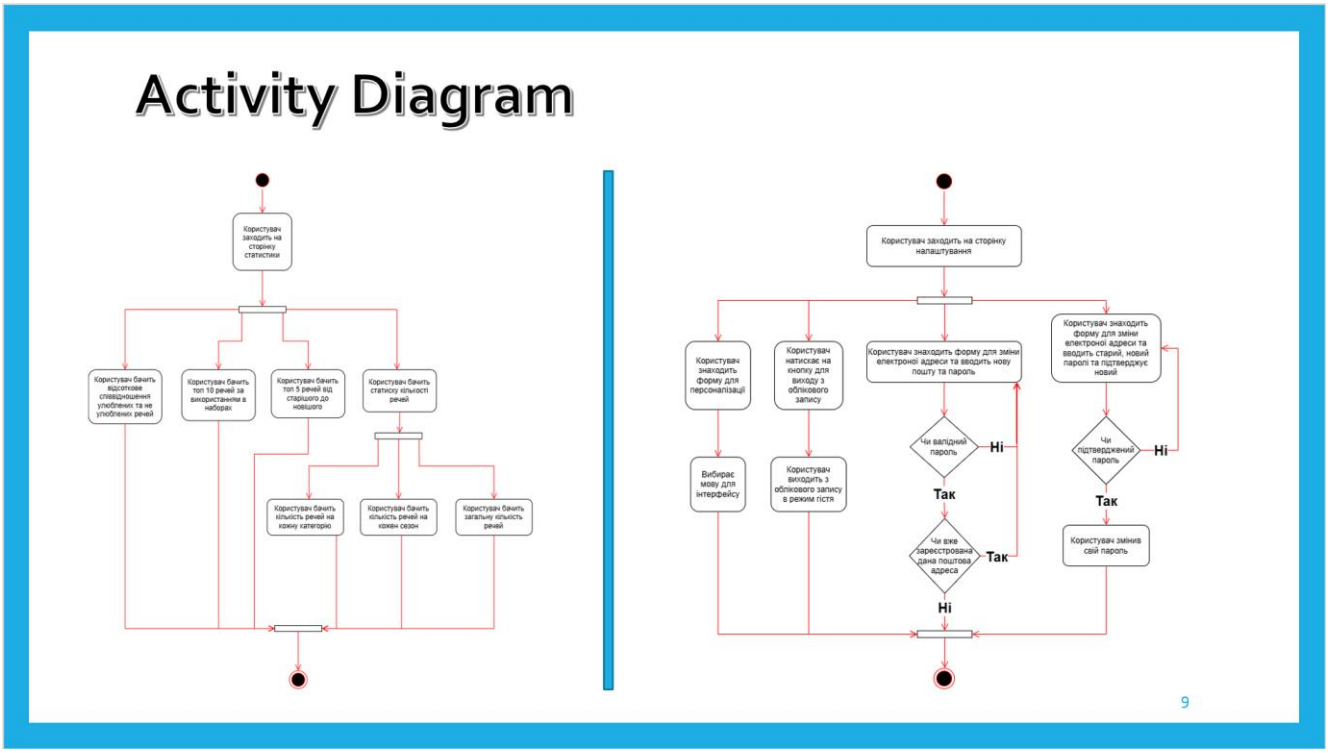


Рисунок Б.9 – Слайд презентації №9 (рисунок виконано самостійно)

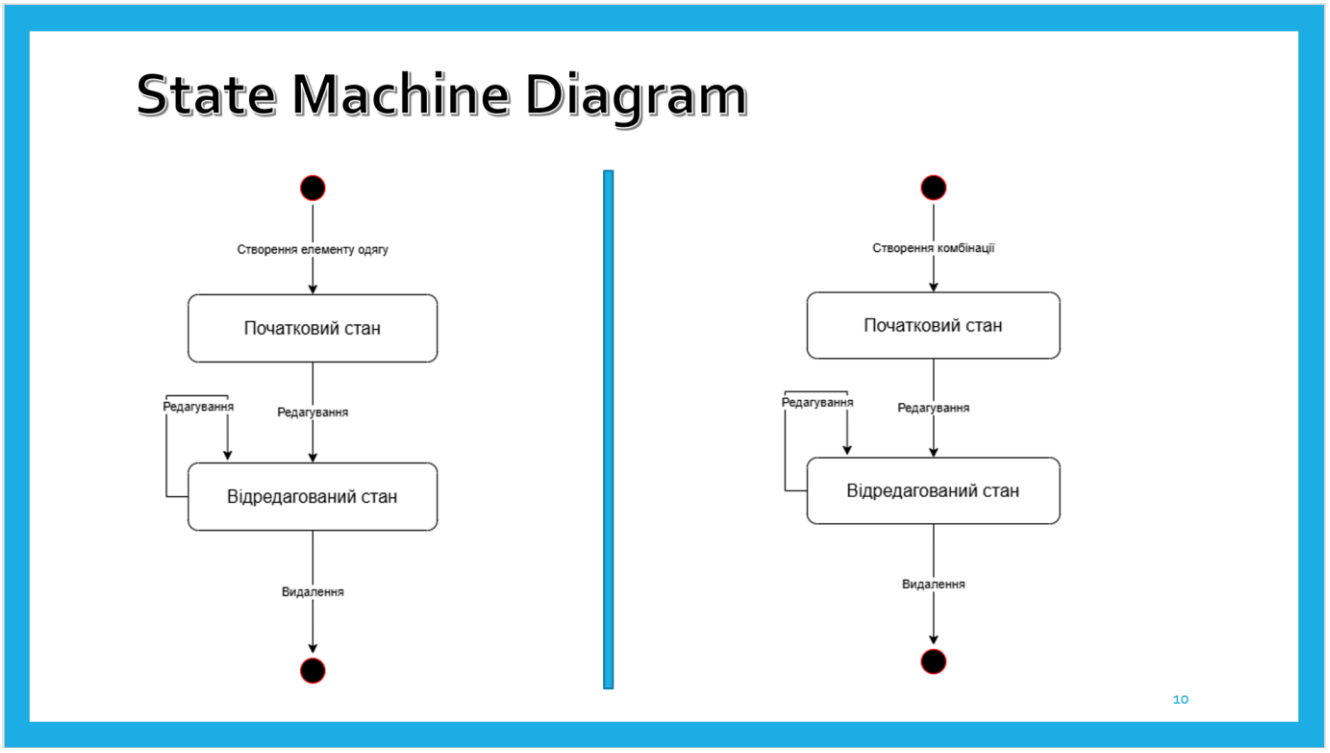


Рисунок Б.10 – Слайд презентації №10 (рисунок виконано самостійно)

Рисунок Б.11 – Слайд презентації №11 (рисунок виконано самостійно)

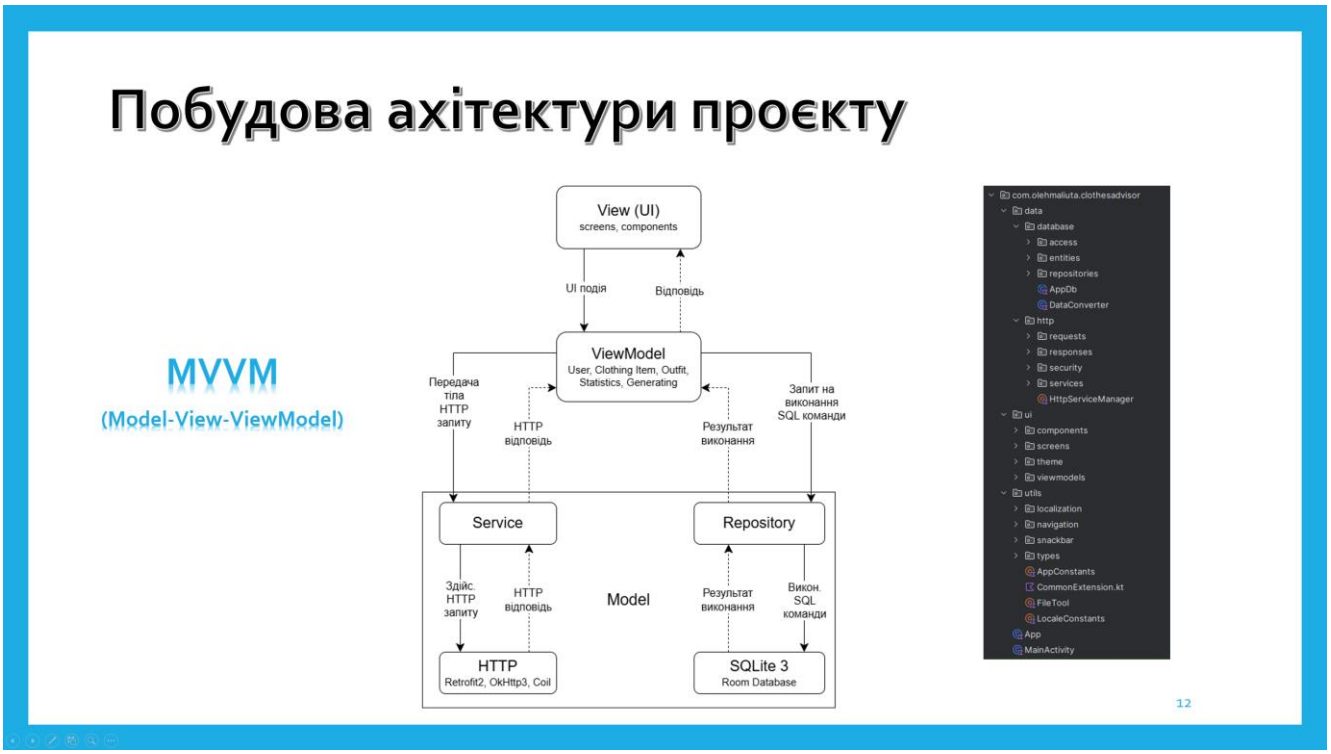


Рисунок Б.12 – Слайд презентації №12 (рисунок виконано самостійно)

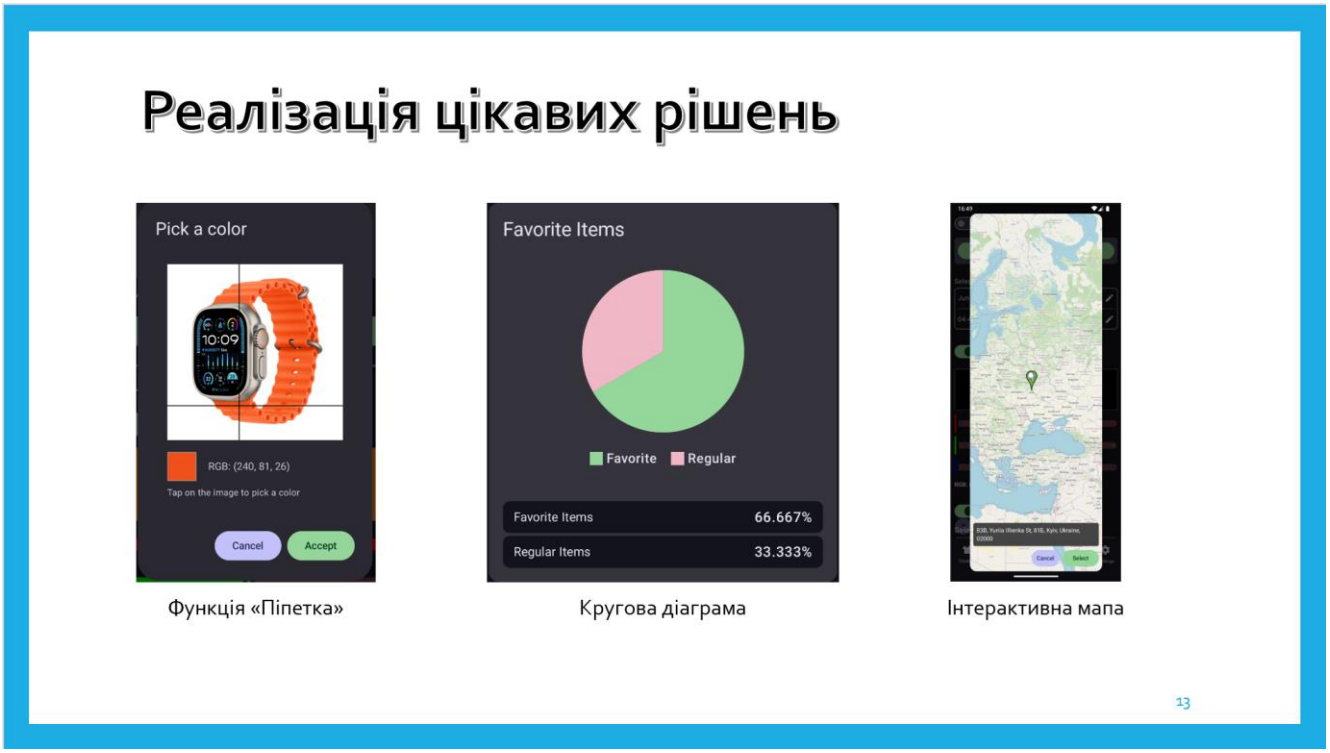


Рисунок Б.13 – Слайд презентації №13 (рисунок виконано самостійно)

Рисунок Б.14 – Слайд презентації №14 (рисунок виконано самостійно)

Рисунок Б.15 – Слайд презентації №15 (рисунок виконано самостійно)

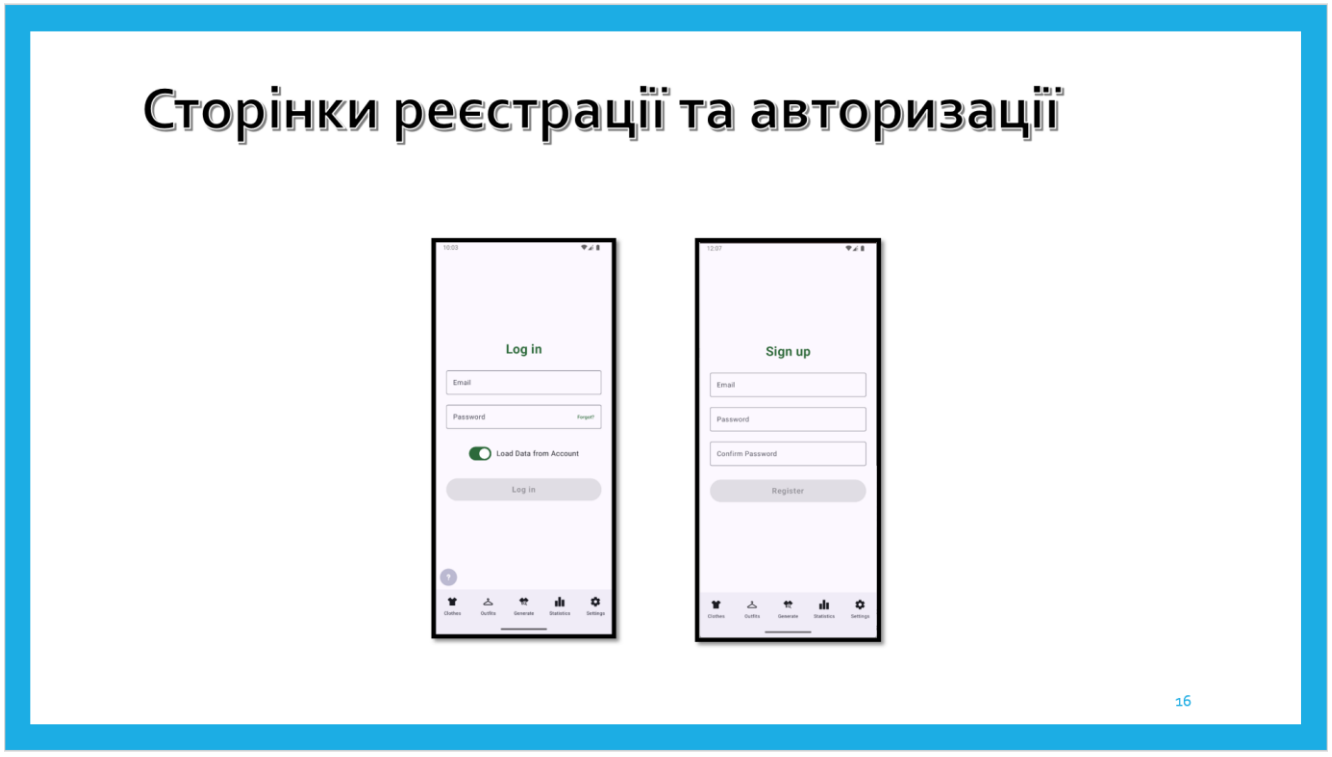


Рисунок Б.16 – Слайд презентації №16 (рисунок виконано самостійно)

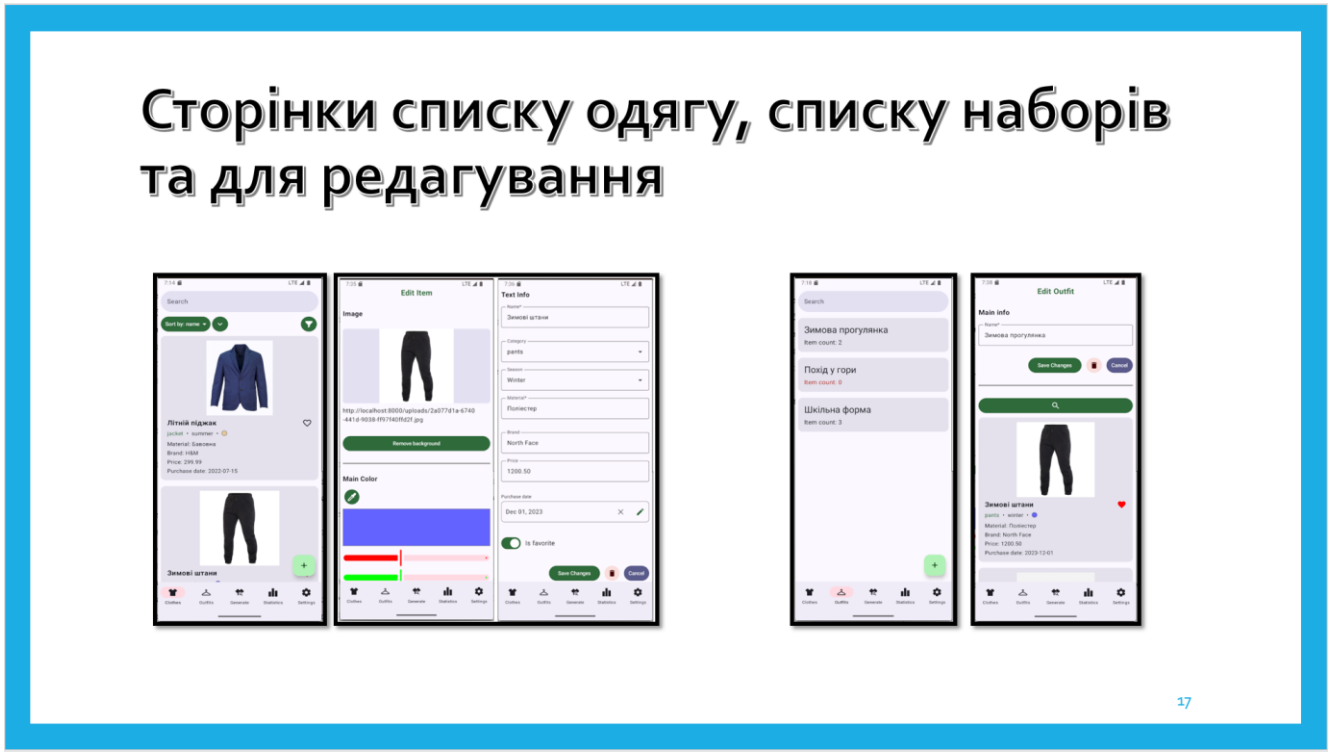


Рисунок Б.17 – Слайд презентації №17 (рисунок виконано самостійно)

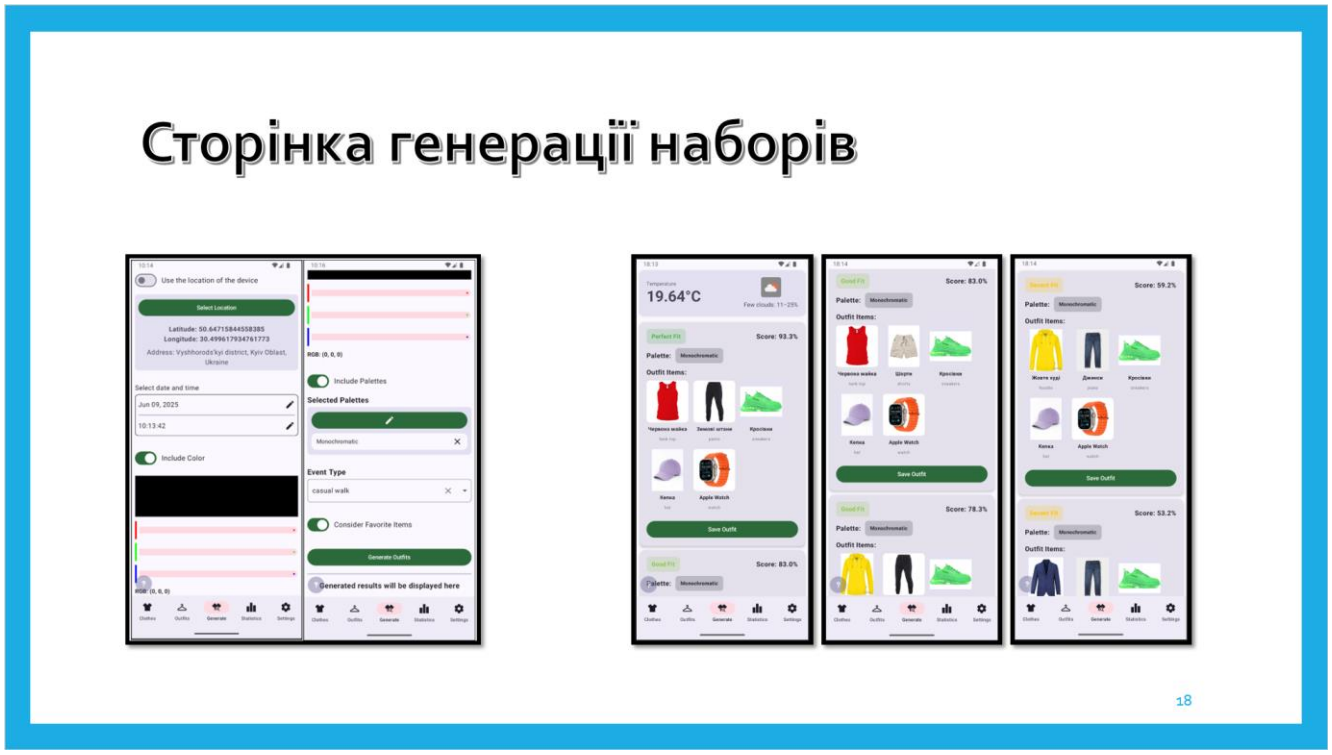


Рисунок Б.18 – Слайд презентації №18 (рисунок виконано самостійно)

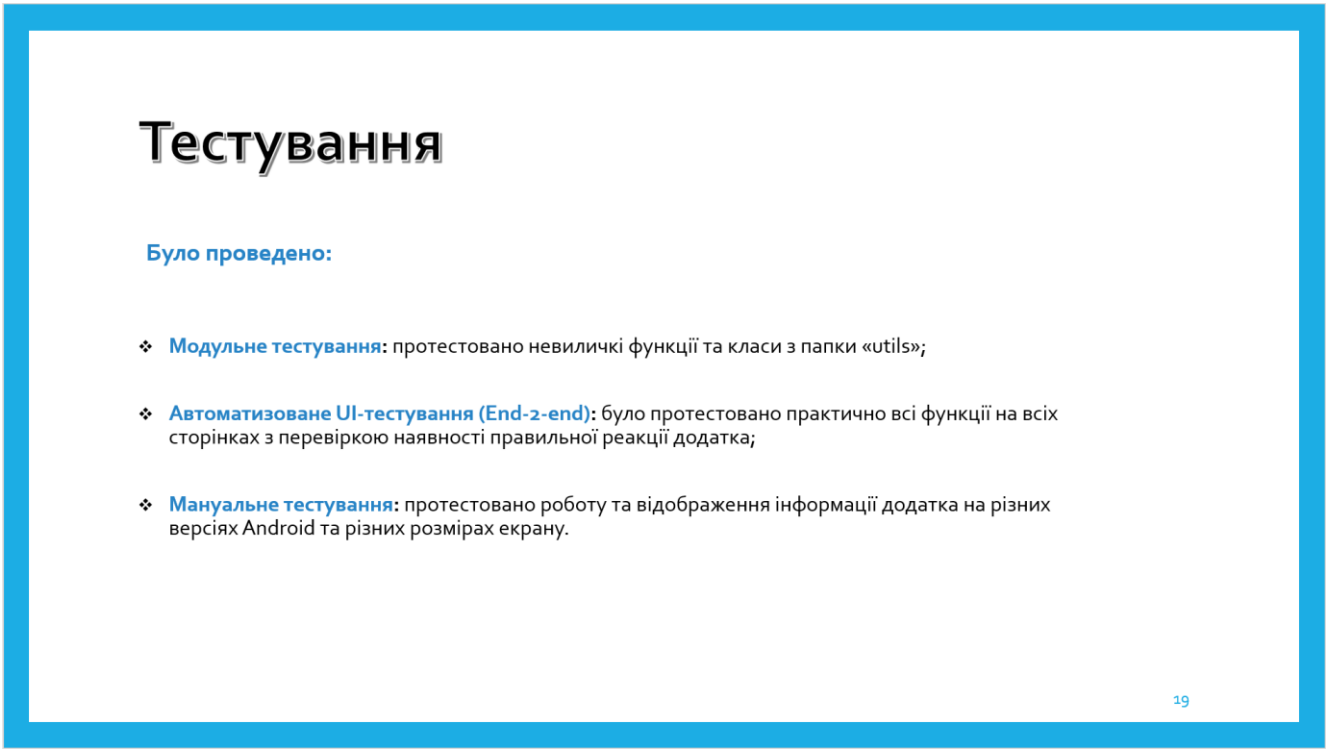


Рисунок Б.19 – Слайд презентації №19 (рисунок виконано самостійно)

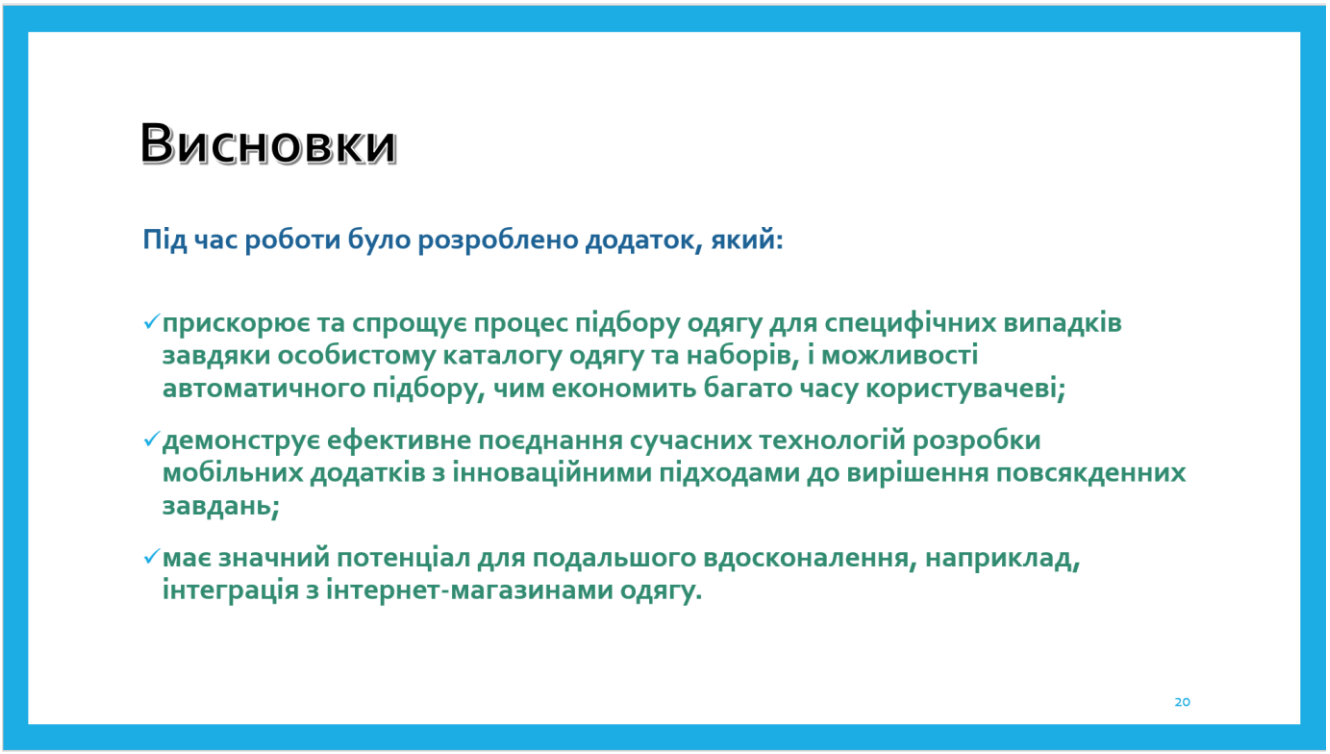


Рисунок Б.20 – Слайд презентації №20 (рисунок виконано самостійно)

ДОДАТОК В

Специфікація

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр денної форми навчання
Кафедра програмної інженерії

СПЕЦІФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мобільного додатку для управління гардеробом

Студент гр. ПЗПІ-21-6 _____ Малюта О. В.

Студент гр. ПЗПІ-21-6 _____ Замковий А. Г.

Харків
2025

1 ВСТУП

1.1 Огляд продукту

Мобільний додаток для управління гардеробом призначений для платформи Android і надає користувачам інструменти для каталогізації одягу, створення наборів, отримання персоналізованих рекомендацій та аналізу статистики використання гардеробу. Додаток підтримує офлайн-режим та синхронізацію даних між пристроями.

1.2 Мета

Метою розробки є створення зручного інструменту для організації гардеробу, який автоматизує процес підбору одягу, надає детальну статистику і більша частина якого може працювати без інтернет-з'єднання.

1.3 Межі

Додаток призначений для ОС Android (версія 7.0 та вище);

Такі функції як реєстрація, авторизація, синхронізація, генерація наборів, та деякі інші вимагають підключення до інтернету.

1.4 Посилання

1. Android — OpenStreetMap Wiki [Електронний ресурс] – URL: https://wiki.openstreetmap.org/wiki/Android (дата звернення: 15.04.2025)
2. Jetpack Compose UI App Development Toolkit - Android Developers [Електронний ресурс] – URL: https://developer.android.com/compose (дата звернення: 06.04.2025)
3. Coroutine Image Loader [Електронний ресурс] – URL: https://coil-kt.github.io/coil/ (дата звернення: 10.04.2025)
4. MVVM (Model View ViewModel) Architecture Pattern in Android |

GeeksforGeeks [Електронний ресурс] – URL: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/ (дата звернення: 08.04.2025)
5. Python 3.12 documentation. Python documentation. URL: https://docs.python.org/3/ (date of access: 10.04.2025).

1.5 Означення та абревіатури

API – Application Programming Interface
DB – Data Base
HTTP – Hyper-Text Transfer Protocol
MVVM – Model-View-ViewModel
REST – Representational State Transfer
RGB – Red Green Blue
SDK – Software Development Kit
UI – User Interface

2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи продукту

Аналізуючи поточний ринок та існуючі аналоги, додаток має дані перспективи, що роблять його більш актуальним:

- Можливість роботи більшості функціоналу в офлайн-режимі із локальним зберіганням даних.
- Розширені алгоритми підбору одягу, які враховують не тільки колір і категорію, але й погоду, призначення одягу (спорт, прогулянка та інші), стиль користувача.
- Створення інтуїтивного та приємного інтерфейсу для зручного управління елементами гардеробу.

2.2 Функції продукту

Мобільний додаток для управління гардеробом має наступні функціональні можливості:

- авторизація та реєстрація користувачів:
  - вхід у систему у ролі гостя з локальним зберіганням даних;
  - реєстрація за електронною адресою для збереження даних у обліковому записі;
  - авторизація з можливістю синхронізації даних між пристроями та сервером, де користувач буде вибирати, за якими даними (локальними або серверними) буде відбуватись синхронізація;
  - перезапис пароля користувача.
- управління гардеробом:
  - додавання елементів одягу з фотографуванням та збереженням зображень;

- редагування характеристик одягу: назва, категорія (головний убір, верхній одяг, взуття тощо), сезонність (зима, весна, літо, осінь, всесезонний), основний колір (формат RGB), матеріал (бавовна, вовна, поліестер тощо), додаткові параметри: бренд, дата придбання, вартість (за бажанням);
- видалення елементів одягу;
- можливість видалення заднього фону зображень для одягу з гардеробу;
- можливість позначання улюблених елементів одягу для покращення результату генерації комбінацій одягу.

- генерація наборів (комбінацій) одягу:
  - автоматична генерація наборів на основі: стилю та кольорової гами, погодних умов та призначення (робота, відпочинок тощо), переваг користувача (кольорова палітра та інше);
  - ручне створення наборів (комбінацій) з можливістю вибору елементів гардеробу;
  - редагування збережених наборів;
  - видалення збережених наборів.
- статистика:
  - статистика кількості речей (загально, для кожного сезону, для кожної категорії);
  - статистика за віком речей (топ 5 речей від старішого до новішого);
  - статистика використання одягу в наборах (топ 10 речей);
  - відсоткове співвідношення улюблених та не улюблених елементів одягу.
- особисті налаштування:
  - зміна мови інтерфейсу, вибираючи між англійською та українською мовами;

- зміна електронної пошти користувача;
- зміна пароля користувача.

2.3 Характеристики користувачів

Цільова аудиторія додатка охоплює широке коло людей з різними потребами та звичками:

- студенти та молоді професіонали (18–30 років) – активні користувачі смартфонів, які цінують простоту та функціональність. Вони часто стикаються з нестачею часу по вранці та потребують швидкого підбору одягу для навчання, роботи або зустрічей з друзями. Для них важливі такі функції, як швидке створення комплектів, синхронізація з розкладом і рекомендації на основі погоди;
- офісні працівники (25–45 років) – користувачі, які дотримуються дрес-коду або хочуть виглядати презентабельно. Вони потребують інструменту для планування образів на тиждень, аналізу того, які речі найчастіше використовуються, а які взагалі лежать без діла. Також їм буде корисна функція створення "капсульних гардеробів" для роботи;
- модні ентузіасти (будь-якого віку) – люди, які слідкують за трендами і люблять експериментувати зі стилем. Для них важливі такі функції, як збереження ідей образів, можливість ділитися своїми комплектами в соціальних мережах та отримувати інспірацію на основі власного гардеробу;
- подорожуючі – користувачі, які часто змінюють кліматичні умови і потребують швидкого підбору одягу під конкретну подорож. Для них буде особливо корисна функція автоматичного підбору речей з урахуванням прогнозу погоди в пункті призначення;
- батьки сімейств – зайняті люди, які керують не лише своїм гардеробом, але й одягом дітей. Вони оцінять можливість створення окремих профілів для

кожного члена сім'ї, нагадування про сезонну зміну одягу та поради щодо комплектів для різних подій.

2.4 Загальні обмеження

Програмна система має такі обмеження:

- підтримка додатка тільки на Android 7.0 та вище;
- обмеження на розмір зображень для елементів гардеробу (5 МБ);
- обмеження по кількості елементів одягу (100) та наборів (50) на одного користувача;
- необхідність підключення до інтернету для реєстрації, авторизації, синхронізації, генерації наборів, видалення фону зображення одягу, можливості зміни електронної пошти та пароля.

2.5 Припущення й залежності

Розробка та функціонування додатка "ClothesAdvisor" базуються на наступних ключових припущеннях і мають такі залежності.

Пристрій користувача має мінімальні апаратні вимоги:

- Android 7.0 (API 24) або новішої версії;
- камера для фотографування та додавання фотографій одягу.

Для повноцінної роботи окремих функцій необхідно:

- доступ до інтернету для синхронізації даних між пристроями;
- дозвіл на використання геолокації для автоматичного підбору наборів з урахуванням погоди;
- дозвіл на доступ до сховища для доступу до фотографій.

Система залежить від:

- сервісів Google Play для роботи на Android-пристроях;
- зовнішніх погодних API для функції рекомендацій;

– зовнішніх API [1], які відповідають за відображення інтерактивної мапи для вибору локації;
– безпечних HTTPS-з'єднань для роботи з серверною частиною.

Продуктивність додатка може залежати від:

– швидкості інтернет-з'єднання для синхронізації;
– апаратних можливостей пристрою;
– доступу до GPS для визначення локації.

Бізнес-припущення, в яких передбачається, що:

– користувач має базові навички роботи з мобільними додатками;
– додаток буде використовуватись персонально, а не на спільних пристроях.

Важливо враховувати, що зміна будь-якого з цих припущень може вимагати модифікації архітектури або функціоналу додатка.

3 КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до зовнішніх інтерфейсів

3.1.1 Інтерфейс користувача

Інтерфейс додатка побудований за допомогою бібліотеки Jetpack Comose [2] та повинен забезпечувати інтуїтивну взаємодію для користувачів з різним рівнем технічної підготовки. Головний екран містить чотири основні секції: список одягу, список наборів, генерація наборів та налаштування.

3.1.2 Апаратний інтерфейс

Додаток використовує стандартні апаратні можливості Android-пристроїв. Для роботи з фотографіями одягу необхідний доступ до галереї зображень. Функція автоматичної генерації наборів може використовувати локацію пристрою для подальшого використання в алгоритмі.

3.1.3 Програмний інтерфейс

API додатка реалізовано з використанням архітектурного стилю REST. Більшість запитів містить заголовки авторизації з JWT-токеном. Для роботи з зображеннями використовується бібліотека Coil [3], що дозволяє отримувати та відображати на екрані завантажені зображення з сервера або локального прострою. Документація API доступна у Swagger UI за захищеним HTTPS-з'єднанням.

3.1.4 Комунікаційний протокол

Основним протоколом обміну даними є HTTPS. Для передачі даних використовуються такі формати, як JSON, Multipart та FormURLEncoded з UTF-8 кодуванням.

3.1.5 Обмеження пам’яті

Фотографії зображень для завантаження у використанні для елементів одягу повинні мати розмір не більше 5 МБ. Один авторизований користувач або гість може мати максимум 100 елементів одягу та 50 наборів.

3.1.6 Операції

Система забезпечує атомарність операцій з базою даних через механізм транзакцій. Критичні операції (видалення наборів або одягу, дозвіл на геолокацію або фотографії) супроводжуються підтвердженням користувача.

3.1.7 Функції продукту

Ядро функціоналу включає три основні групи можливостей: управління гардеробом (додавання, редагування, категорізація, фільтрація), генерація наборів (ручний підбір, автоматичні рекомендації) та аналітика (статистика використання, рекомендації з оновлення гардеробу). Більшість функцій доступна у двох режимах - онлайн та офлайн.

3.1.8 Припущення й залежності

Система передбачає наявність стабільного інтернет-з’єднання для отримання актуальної інформації про погоду з API OpenWeather та роботи з сервером.

Передбачається, що зовнішній погодний сервіс надає точні, актуальні та стандартизовані погодні описи (наприклад, "clear sky", "light rain" тощо), які система інтерпретує згідно з внутрішньою таблицею відповідностей.

Користувач підтримує гардероб у системі в актуальному стані: додає, редагує або видаляє одяг відповідно до реального вмісту.

Система розробляється для роботи на сучасних пристроях Android , тому покладається на наявність підтримки сучасних веб-технологій або Android API.

Користувач приймає рекомендації одягу як допоміжні — остаточний вибір залежить від його вподобань.

Система залежить від сторонніх сервісів (погодне API, бібліотек тощо).

3.2 Властивості програмного продукту

Програмний продукт відрізняється гнучкістю налаштувань інтерфейсу, включаючи можливість вибору між компактним і розширеним відображенням елементів. Архітектура мобільного додатку MVVM [4] забезпечує чітке розділення логіки та відображення. Специфічною особливістю є алгоритм автоматичної генерації наборів, який враховує такі фактори, як погода, сезон, основний колір, кольорові палітри, призначення, та врахування улюблених речей.

3.3 Атрибути програмного продукту

3.3.1 Надійність

Система повинна функціонувати без збоїв протягом тривалого періоду часу, навіть при багаторазовому зверненні до погодного API або частій зміні користувацьких даних.

Застосовуються механізми обробки помилок для запобігання критичних збоїв при втраті з'єднання з мережею або при недоступності сторонніх сервісів.

У випадку непередбаченого завершення роботи або збою, система має забезпечити збереження введених даних (наприклад, оновлення гардеробу чи обрані налаштування).

При повторному запуску програми користувач продовжує з останнього стабільного стану.

Програма перевіряє правильність введених користувачем даних (наприклад текстові назви одягу, числові параметри температури), щоб уникнути некоректної обробки або краху.

3.3.2 Доступність

Офлайн-режим забезпечує доступ до всіх локально збережених даних без обмежень. При відсутності інтернет-з'єднання недоступні операції позначаються повідомленнями або повною візуальною відсутністю. Синхронізація відбувається у фоновому режимі при відновленні зв'язку.

3.3.3 Безпека

Система використовує захищений режим HTTPS для передачі даних до сервера та відправки відповіді. Для безпечного зберігання паролів користувачів використовується хешування алгоритмом bcrypt. Аутентифікація користувачів реалізована за допомогою токенів JWT. Після успішного входу система видає унікальний токен, який використовується для авторизації у всіх наступних запитах. Це забезпечує безпечний та контрольований доступ до персональних даних користувача без потреби повторного введення пароля. Токени мають обмежений термін дії та можуть бути відкликані у випадку втрати доступу або підозрілої активності.

3.3.4 Супроводжуваність

Кодова база мобільного додатку супроводжується модульними та автоматизованими і мануальними UI-тестами.

Усі REST API-запити мають документацію у форматі OpenAPI (Swagger), що спрощує підтримку, налагодження та інтеграцію з іншими клієнтами.

Сервер підтримує централізоване логування подій (зокрема помилок, авторизацій, запитів)

Основні компоненти серверної логіки покриті автоматизованими тестами та тестами навантаження, що дозволяє виявляти помилки до розгортання.

3.3.5 Переносимість

Система керування гардеробом повинна мати високий рівень переносимості, щоб забезпечити можливість її використання на різних платформах. Вимоги до переносимості включають:

Платформи: Система повинна працювати на операційних системах Android (версія 7.0 і вище). У майбутньому можлива адаптація для iOS та веб-платформи.

Мобільні пристрої: Інтерфейс користувача повинен коректно відображатися на екранах різних розмірів (від 4" до 7.5").

Переносимість даних: Повинна бути реалізована можливість експорту та імпорту даних гардеробу між клієнтом та сервером для резервного копіювання та перенесення між пристроями.

3.3.6 Продуктивність

Система повинна забезпечувати належну продуктивність для комфортного використання, зокрема:

Час запуску: Час запуску застосунку не повинен перевищувати 3 секунд на середньостатистичному пристрої (з 4 ГБ оперативної пам’яті).

Час відповіді: Час реакції на дії користувача (перегляд, редагування, додавання одягу) не повинен перевищувати 1 с, час відповіді від серверу не повинен перевищувати 3 с.

Масштабованість: Система повинна підтримувати зберігання 100 елементів одягу без суттєвого падіння продуктивності.

Оперативне споживання пам’яті: Додаток має раціонально використовувати ресурси, не перевищуючи 150 МБ оперативної пам’яті під час стандартної роботи.

3.4 Вимоги баз даних

Система повинна використовувати вбудовану базу даних для зберігання інформації про гардероб на локальному пристрої та на виділеному сервері. Основні вимоги до бази даних:

Має бути SQL сситемою з підтримкою ORM.

Операції CRUD: Повна підтримка операцій створення, читання, оновлення та видалення елементів.

Цілісність: Має бути реалізована підтримка зв’язків між сутностями (наприклад, багато до багатьох та один до багатьох).

Резервне копіювання: Повинна бути можливість експортувати базу даних у вигляді файлу, який може бути збережений.

3.5 Інші вимоги

Система повинна підтримувати інтернаціоналізацію з можливістю додавання нових мов без змін коду.

ДОДАТОК Г

Програмний код UI-компоненту для малювання кругової діаграми

```
@Composable
fun PieChart(
    data: Map<String, Float>,
    colors: List<Color>,
    modifier: Modifier = Modifier
) {
    val total = data.values.sum()
    var startAngle = -90f

    Column(
        modifier = modifier
            .fillMaxWidth()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Box(
            modifier = Modifier
                .size(200.dp)
                .padding(8.dp),
            contentAlignment = Alignment.Center
        ) {
            Canvas(modifier = Modifier.fillMaxSize()) {
                val diameter = size.minDimension
                val sweepAngles = data.values.map {
                      it * 360f / total
                 }

                sweepAngles.forEachIndexed { index, angle ->
                    drawArc(
                        color = colors[index % colors.size],
                        startAngle = startAngle,
                        sweepAngle = angle,
                        useCenter = true,
                        size = Size(diameter, diameter)
                    )
                    startAngle += angle
                }
            }
        }

        Row(
            modifier = Modifier
                .fillMaxWidth().padding(top = 8.dp),
            horizontalArrangement = Arrangement.Center
        ) {
            data.keys.forEachIndexed { index, label ->
                Row(
                    modifier = Modifier
                      .padding(horizontal = 8.dp, vertical = 4.dp),
                    verticalAlignment = Alignment.CenterVertically
                ) {
                    val boxSize = with(LocalDensity.current) {
                      15.sp.toDp()
```

```
                        }

                        Box(
                            modifier = Modifier
                                .size(boxSize)
                                .background(colors[index % colors.size])
                                .clip(CircleShape)
                        )
                        Spacer(modifier = Modifier.width(4.dp))
                        Text(
                            text = label,
                            fontSize = 15.sp,
                            style = MaterialTheme.typography.labelSmall
                        )
                    }
                }
            }
        }
    }
```

ДОДАТОК Д

Програмний код UI-компоненту, який займається редагуванням кольорової схеми RGB та містить функцію «піпетка»

```
@Composable
fun ColorPicker(
    color: Color,
    imageUri: String?,
    onColorChange: (Color) -> Unit,
    modifier: Modifier = Modifier
) {
    val context = LocalContext.current

    var red by remember(color) { mutableFloatStateOf(color.red) }
    var green by remember(color) { mutableFloatStateOf(color.green) }
    var blue by remember(color) { mutableFloatStateOf(color.blue) }

    var isPipetteMenuOpen by remember { mutableStateOf(false) }
    var pipetteColor by remember { mutableStateOf(Color.Black) }
    var imageRequest by remember {
mutableStateOf<ImageRequest?>(null) }
    var bitmap by remember { mutableStateOf<Bitmap?>(null) }
    var imageSize by remember { mutableStateOf(IntSize.Zero) }
    var touchPosition by remember { mutableStateOf(Offset.Zero) }

    val updateColor: () -> Unit = {
        onColorChange(Color(red, green, blue))
    }

    LaunchedEffect(isPipetteMenuOpen) {
        if (isPipetteMenuOpen) {
            touchPosition = Offset(
                imageSize.width.toFloat() / 2,
                imageSize.height.toFloat() / 2)
        }
    }

    LaunchedEffect(imageUri) {
        if (imageUri != null) {
            val resultImageUrl = if (imageUri.startsWith("http"))
                imageUri.replace("://localhost", "://10.0.2.2") else
                imageUri
            val request = ImageRequest.Builder(context)
                .data(resultImageUrl)
                .memoryCachePolicy(CachePolicy.DISABLED)
                .diskCachePolicy(CachePolicy.DISABLED)
                .setHeader(
                    "Cache-Control",
                    "no-store, no-cache, must-revalidate")
                .setHeader("Pragma", "no-cache")
                .setHeader("Expires", "0")
                .allowHardware(false)
                .target { drawable ->
                    val bitmapDrawable = drawable as? BitmapDrawable
                    bitmap = bitmapDrawable?.bitmap
```

```
                        ?.copy(Bitmap.Config.ARGB_8888, true)
                }.build()
            val imageLoader = ImageLoader.Builder(context).build()
            imageLoader.execute(request)
        }
    }

    AcceptCancelDialog(
        isOpen = isPipetteMenuOpen,
        title = "Pick a color",
        onDismissRequest = { isPipetteMenuOpen = false },
        onAccept = {
            red = pipetteColor.red
            green = pipetteColor.green
            blue = pipetteColor.blue
            updateColor()
            isPipetteMenuOpen = false
        },
        acceptText = "Apply"
    ) {
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp)
        ) {
            if (imageUri != null) {
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .aspectRatio(1f)
                .background(MaterialTheme.colorScheme.surfaceVariant)
                        .pointerInput(Unit) {
                            detectTapGestures { offset ->
                                touchPosition = offset
                                bitmap?.let { bmp ->
                                    val x = (offset.x / size.width *
bmp.width)
                                        .toInt().coerceIn(0,
bmp.width - 1)
                                    val y = (offset.y / size.height *
bmp.height)
                                        .toInt().coerceIn(0,
bmp.height - 1)
                                    val pixel = bmp[x, y]
                                    pipetteColor = Color(
                                        red = android.graphics.Color
                                            .red(pixel) / 255f,
                                        green =
android.graphics.Color
                                            .green(pixel) / 255f,
                                        blue = android.graphics.Color
                                            .blue(pixel) / 255f
                                    )
                                }
                            }
                        }
                ) {
```

```
                    val resultImageUrl = if
(imageUri.startsWith("http"))
                        imageUri.replace("://localhost",
"://10.0.2.2") else
                        imageUri
                    val painter = rememberAsyncImagePainter(
                        model =
ImageRequest.Builder(LocalContext.current)
                            .data(resultImageUrl)
                            .memoryCachePolicy(CachePolicy.DISABLED)
                            .diskCachePolicy(CachePolicy.DISABLED)
                            .setHeader(
                                "Cache-Control",
                                "no-store, no-cache, must-
revalidate")
                            .setHeader("Pragma", "no-cache")
                            .setHeader("Expires", "0")
                            .build()
                    )

                    Image(
                        painter = painter,
                        contentDescription =
                           "Image to pick color from",
                        contentScale = ContentScale.FillBounds,
                        modifier = Modifier
                            .fillMaxSize()
                    )

                    Canvas(modifier = Modifier.fillMaxSize()) {
                        drawCircle(
                            color = pipetteColor,
                            center = touchPosition,
                            radius = 30f,
                            style = Stroke(width = 3f)
                        )
                        drawLine(
                            color = Color.Black,
                            start = Offset(touchPosition.x, 0f),
                            end = Offset(touchPosition.x,
size.height),
                            strokeWidth = 3f
                        )
                        drawLine(
                            color = Color.Black,
                            start = Offset(0f, touchPosition.y),
                            end = Offset(size.width,
touchPosition.y),
                            strokeWidth = 3f
                        )
                    }
                }

                Spacer(modifier = Modifier.height(16.dp))

                Row(
                    verticalAlignment = Alignment.CenterVertically,
                    modifier = Modifier.fillMaxWidth()
```

```
                ) {
                    Box(
                        modifier = Modifier
                            .size(40.dp)
                            .background(pipetteColor)
                            .border(1.dp,
MaterialTheme.colorScheme.outline)
                    )

                    Spacer(modifier = Modifier.width(16.dp))

                    Text(
                        text = "RGB: (${(pipetteColor.red *
255).toInt()}, " +
                                "${(pipetteColor.green *
255).toInt()}, " +
                                "${(pipetteColor.blue *
255).toInt()})",
                        style = MaterialTheme.typography.bodyMedium
                    )
                }

                Spacer(modifier = Modifier.height(8.dp))

                Text(
                    text = "Tap on the image to pick a color",
                    style = MaterialTheme.typography.bodySmall,
                    color =
MaterialTheme.colorScheme.onSurfaceVariant
                )
            } else {
                Text("No image available for color picking")
            }
        }
    }

    Column(modifier = modifier) {
        Row {
            if (imageUri != null) {
                IconButton(
                    colors = IconButtonDefaults.iconButtonColors(
                        containerColor =
MaterialTheme.colorScheme.primary,
                        contentColor =
MaterialTheme.colorScheme.onPrimary
                    ),
                    onClick = { isPipetteMenuOpen  = true },
                ) {
                    Icon(
                        painter =
painterResource(R.drawable.pipette),
                        contentDescription = null,
                        modifier = Modifier.size(22.dp),
                        tint = MaterialTheme.colorScheme.onPrimary
                    )
                }
            }
        }
```

```
Spacer(modifier = Modifier.height(8.dp))

Box(
    modifier = Modifier
        .fillMaxWidth()
        .height(100.dp)
        .background(Color(red, green, blue))
        .border(1.dp, MaterialTheme.colorScheme.outline)
)

Spacer(modifier = Modifier.height(8.dp))

Slider(
    value = red,
    onValueChange = { newValue ->
        red = newValue
        updateColor()
    },
    valueRange = 0f..1f,
    modifier = Modifier.fillMaxWidth(),
    colors = SliderDefaults.colors(
        thumbColor = Color.Red,
        activeTrackColor = Color.Red
    )
)

Spacer(modifier = Modifier.height(5.dp))

Slider(
    value = green,
    onValueChange = { newValue ->
        green = newValue
        updateColor()
    },
    valueRange = 0f..1f,
    modifier = Modifier.fillMaxWidth(),
    colors = SliderDefaults.colors(
        thumbColor = Color.Green,
        activeTrackColor = Color.Green
    )
)

Spacer(modifier = Modifier.height(5.dp))

Slider(
    value = blue,
    onValueChange = { newValue ->
        blue = newValue
        updateColor()
    },
    valueRange = 0f..1f,
    modifier = Modifier.fillMaxWidth(),
    colors = SliderDefaults.colors(
        thumbColor = Color.Blue,
        activeTrackColor = Color.Blue
    )
)
```

```
        Spacer(modifier = Modifier.height(8.dp))

        Text(
            text = "RGB: (" +
                    "${(red * 255).toInt()}, " +
                    "${(green * 255).toInt()}, " +
                    "${(blue * 255).toInt()})",
            fontWeight = FontWeight.Bold,
            style = MaterialTheme.typography.bodyMedium
        )
    }
}
```

ДОДАТОК Е

Програмний код UI-компоненту, який представляє собою меню з інтерактивною картою з можливістю вибору локації

```
@Composable
fun OsmLocationPickerDialog(
    isOpen: Boolean,
    initialLocation: GeoPoint = GeoPoint(0.0, 0.0),
    onLocationSelected: (GeoPoint, String?) -> Unit,
    onDismiss: () -> Unit
) {
    if (!isOpen) { return }

    val context = LocalContext.current
    var selectedLocation by remember {
mutableStateOf(initialLocation) }
    var address by remember { mutableStateOf<String?>(null) }
    var isLoading by remember { mutableStateOf(false) }

    LaunchedEffect(Unit) {
        Configuration.getInstance().userAgentValue =
context.packageName
    }

    val scope = rememberCoroutineScope()

    Dialog(onDismissRequest = onDismiss) {
        Surface(
            modifier = Modifier.fillMaxWidth(),
            shape = MaterialTheme.shapes.large,
            color = MaterialTheme.colorScheme.surface
        ) {
            Box(Modifier.fillMaxSize()) {
                OSMMapView(
                    context = context,
                    initialLocation = initialLocation,
                    onLocationSelected = { geoPoint ->
                        selectedLocation = geoPoint
                        scope.launch {
                            isLoading = true
                            address = getAddressFromGeoPoint(context,
geoPoint)
                            isLoading = false
                        }
                    }
                )
            }

            Column(
                verticalArrangement = Arrangement.Bottom,
                modifier = Modifier
                    .fillMaxSize()
                    .padding(8.dp)
            ) {
                address?.let {
```

```
                    Box(
                        modifier = Modifier
                            .background(
                                color = Color.DarkGray,
                                shape = RoundedCornerShape(5.dp)
                            )
                            .padding(vertical = 8.dp, horizontal =
8.dp)
                    ) {
                        Text(
                            text = it,
                            style =
MaterialTheme.typography.bodyMedium,
                            color = Color.White
                        )
                    }

                    Spacer(modifier = Modifier.height(5.dp))
                }

                Row(
                    modifier = Modifier.fillMaxWidth(),
                    horizontalArrangement = Arrangement.End
                ) {
                    TextButton(
                        onClick = onDismiss,
                        colors = ButtonDefaults.buttonColors(
                            containerColor =
MaterialTheme.colorScheme.tertiary,
                            contentColor =
MaterialTheme.colorScheme.onTertiary
                        )
                    ) {

Text(stringResource(R.string.osm_location_picker_dialog__cancel_butto
n))
                    }
                    Spacer(Modifier.width(8.dp))
                    Button(
                        onClick = {
                            scope.launch {
                                val finalAddress = address ?:
                                getAddressFromGeoPoint(context,
selectedLocation)
                                onLocationSelected(selectedLocation,
finalAddress)
                            }
                        },
                        enabled = !isLoading
                    ) {

Text(stringResource(R.string.osm_location_picker_dialog__select_butto
n))
                    }
                }
            }
        }
    }
```

```
}

@Composable
private fun OSMMapView(
    context: Context,
    initialLocation: GeoPoint,
    onLocationSelected: (GeoPoint) -> Unit
) {
    var mapView by remember { mutableStateOf<MapView?>(null) }

    AndroidView(
        factory = { ctx ->
            MapView(ctx).apply {
                setTileSource(TileSourceFactory.MAPNIK)
                setMultiTouchControls(true)
                setBuiltInZoomControls(false)
                controller.setZoom(4.0)
                controller.setCenter(initialLocation)

                val marker = Marker(this).apply {
                    position = initialLocation
                    setAnchor(Marker.ANCHOR_CENTER,
Marker.ANCHOR_BOTTOM)
                }

                overlays.add(MapEventsOverlay(object :
MapEventsReceiver {
                    override fun singleTapConfirmedHelper(p:
GeoPoint?): Boolean {
                        p?.let { tappedPoint ->
                            marker.position = tappedPoint
                            marker.setVisible(true)
                            onLocationSelected(tappedPoint)
                            controller.animateTo(tappedPoint)
                            invalidate()
                        }
                        return true
                    }

                    override fun longPressHelper(p: GeoPoint?):
Boolean = false
                }))

                overlays.add(marker)

                val myLocationOverlay =
MyLocationNewOverlay(GpsMyLocationProvider(context), this)
                myLocationOverlay.enableMyLocation()
                overlays.add(myLocationOverlay)

                mapView = this
            }
        },
        update = { view ->

view.overlays.filterIsInstance<Marker>().firstOrNull()?.position =
initialLocation
        }
```

```
    )

    DisposableEffect(Unit) {
        onDispose { mapView?.onDetach() }
    }
}

private fun getAddressFromGeoPoint(context: Context, geoPoint:
GeoPoint): String? {
    return try {
        val geocoder = Geocoder(context, Locale.getDefault())
        if (!Geocoder.isPresent()) return null

        val addresses = geocoder.getFromLocation(
            geoPoint.latitude,
            geoPoint.longitude,
            1
        ) ?: return null

        addresses.firstOrNull()?.let { address ->
            buildString {
                for (i in 0..address.maxAddressLineIndex) {
                    append(address.getAddressLine(i))
                    if (i != address.maxAddressLineIndex) append(",
")
                }
            }
        }
    } catch (_: Exception) {
        null
    }
}
```