Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

**КВАЛІФІКАЦІЙНА РОБОТА**

**на здобуття ступеня освіти бакалавра**

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка мобільного новинного додатку

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-20-1

____________ /М. О. Яблочкін /

Керівник
кваліфікаційної роботи ______________ / І. О. Доценко /

Завідувач кафедри ______________ / А. М. Стрюк /

Кривий Ріг

2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
__________ А. М. Стрюк
«___» _________ 2024 р.

**ЗАВДАННЯ**

**на кваліфікаційну роботу**

студенту групи ІПЗ-20-1 Яблочкіну Максиму Олександровичу

1. Тема: Розробка мобільного новинного додатку затверджено наказом по КНУ № 275с від «15» квітня 2024 р.
2. Термін подання студентом закінченої роботи: «01» червня 2024р.
3. Вихідні дані по роботі: створити зручний мобільний застосунок для отримання новин, що забезпечує персоналізований контент, актуальні сповіщення та безпечний доступ.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): проаналізувати методи розробки новинних мобільних додатків, обрати спосіб завантаження та відображення новин, спроектувати архітектуру, розробити інтерфейс та програмну реалізацію, реалізувати повідомлення, забезпечити безпеку, провести тестування та оцінити ефективність.
5. Перелік ілюстративного матеріалу: діаграми аналізу аналогів та методів розробки, схеми архітектури та дизайну додатку, скріншоти реалізації, ілюстрації повідомлень та аутентифікації, результати тестування та графіки ефективності.

Календарний план:

| № | Найменування етапів кваліфікаційної роботи | Термін виконання етапів роботи |
|---|---|---|
| 1 | Перегляд літератури та аналіз існуючих методів | 16.04.2024 – 18.04.2024 |
| 2 | Проведення аналізу існуючих аналогів | 19.04.2024 – 25.04.2024 |
| 3 | Підготовка матеріалів до першого розділу | 26.04.2024 – 15.05.2024 |
| 4 | Підготовка матеріалів до другого розділу | 16.05.2024 – 22.05.2024 |
| 5 | Вибір методу проектування системи | 23.05.2024 – 06.06.2024 |
| 6 | Підготовка матеріалів до третього розділу | 07.05.2024 – 13.05.2024 |
| 7 | Остаточні доробки та виправлення | 14.05.2024 – 20.05.2024 |
| 8 | Розробка програмної частини | 21.05.2024 – 27.05.2024 |
| 9 | Тестування та налагодження | 28.05.2024 – 29.05.2024 |
| 10 | Оформлення пояснювальної записки | 29.05.2024 – 01.06.2024 |

Дата видачі завдання: «15» квітня 2024 р.

Студент ________________ / М. О. Яблочкін /

Керівник роботи ________________ / І. О. Доценко /

# РЕФЕРАТ

Пояснювальна записка: 72 с., 2 табл., 36 рис., 1 дод., 12 джерел

Метою даної кваліфікаційної роботи є створення мобільного додатка для отримання та перегляду новин на мобільних пристроях.

У роботі проведено аналіз існуючих новинних додатків для мобільних платформ, включаючи аналіз їх функціональності, дизайну та користувацького досвіду. На основі цього аналізу визначено основні вимоги до розроблюваного додатка.

Розроблено концепцію новинного додатка, включаючи архітектуру додатка, його функціональні можливості та інтерфейс користувача. Особлива увага приділялася зручності та швидкості отримання новин.

Виконано програмну реалізацію додатку поки тільки для платформи Android з використанням сучасних технологій розробки мобільних додатків.

Завершальним етапом роботи стане тестування розробленого додатка, включаючи функціональне тестування, тестування продуктивності та тестування на реальних пристроях.

Основні висновки та результати роботи будуть представлені у вигляді публікації тез на конференції MobileConf 2024.

# ABSTRACT

Explanatory Note: 48 pages, 2 tables, 4 figures, 2 appendices, 16 references

The goal of this qualification work is to create a mobile application for receiving and viewing news on mobile devices.

The work includes an analysis of existing news applications for mobile platforms, including an evaluation of their functionality, design, and user experience. Based on this analysis, the main requirements for the developed application were determined.

The concept of the news application was developed, including the application architecture, its functional capabilities, and user interface. Particular attention was paid to the convenience and speed of news retrieval.

The software implementation of the developed application was carried out for the Android platform using modern mobile application development technologies.

The final stage of the work will involve testing the developed application, including functional testing, performance testing, and testing on real devices.

The main conclusions and results of the work will be presented in the form of thesis publication at the MobileTech 2024 conference.

# ЗМІСТ

ВСТУП .......... 7
1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФ ІКАЦІЙНОЇ РОБОТИ .......... 8
1.1 Аналіз аналогів .......... 8
1.2 Аналіз методів вирішення проблеми .......... 12
1.3 Цілі та завдання кваліфікаційної роботи .......... 14
1.4 Вимоги до програмного забезпечення .......... 15
1.4.1 Функціональні вимоги .......... 15
1.4.2 Нефункціональні вимоги .......... 16
1.5 Реалізація алгоритмів .......... 17
1.6 Впровадження програмного забезпечення .......... 18
2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .......... 20
2.1 Вибір технологій рішення проблеми .......... 20
2.2 Вибір мови і IDE .......... 22
2.3 Вибір дизайн системи користувальницького інтерфейсу та його розробка .......... 24
2.4 Розробка та докладний опис функціональної схеми програми .......... 32
3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .......... 40
3.1 Розробка інтерфейсу програми .......... 40
3.2 Розробка бази даних .......... 48
3.3 Тестування користувацького інтерфейсу .......... 49
3.4 Тестування програмного забезпечення .......... 55
ВИСНОВКИ .......... 57
ПЕРЕЛІК ПОСИЛАНЬ .......... 58
Додаток NewsThreads – Код програми .......... 59

# ВСТУП

У сучасному інформаційному суспільстві, де мобільні додатки стали основним джерелом новин, виникає необхідність дослідження та розробки інноваційних рішень, що відповідають зростаючим вимогам користувачів до персоналізації контенту та безпеки.

Зростаюча популярність мобільних пристроїв та попит на якісні новинні додатки спонукають до пошуку нових підходів та технологій у цій сфері. Сучасний користувач очікує від новинного додатку не лише оперативності та різноманітності контенту, але й персоналізованого підходу, зручного інтерфейсу та високої продуктивності.

Мета цієї кваліфікаційної роботи полягає у створенні мобільного новинного додатку з розширеними функціональними можливостями, що забезпечує індивідуальний підхід до кожного користувача та гарантує надійний захист його даних.

Об'єктом дослідження є мобільні новинні додатки як комплексний програмний продукт, що охоплює широкий спектр аспектів: від функціональних можливостей та архітектурних рішень до вибору технологій розробки, принципів персоналізації контенту та забезпечення безпеки даних користувачів.

Завданням роботи є проведення комплексного аналізу існуючих рішень, вибір оптимального технологічного стеку, проектування архітектури та інтерфейсу додатку, реалізація алгоритмів персоналізації контенту, впровадження механізмів безпечної аутентифікації та авторизації користувачів, а також тестування та оцінка ефективності розробленого продукту.

# 1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФ ІКАЦІЙНОЇ РОБОТИ

## 1.1 Аналіз аналогів

У сучасному інформаційному просторі мобільні додатки відіграють ключову роль у споживанні новин. Як студент, я активно використовую такі додатки для отримання актуальної інформації. З метою визначення оптимального інструменту для своїх потреб, я провів аналіз популярних новинних додатків, зосередившись на таких аспектах:

- функціональність: наявність базових та розширених функцій, зручність використання;
- інтерфейс користувача (UI/UX): інтуїтивність, естетична привабливість, адаптивність до різних пристроїв;
- персоналізація: можливості налаштування стрічки новин під індивідуальні інтереси, наявність рекомендаційних систем;
- джерела новин: різноманітність та авторитетність джерел, можливість додавання користувацьких джерел;
- соціальні функції: інструменти для коментування, обговорення та поширення новин;
- технічні характеристики: швидкість роботи, стабільність, рівень споживання ресурсів пристрою.

Огляд аналогів:

- Google News (див. рис. 1.1):

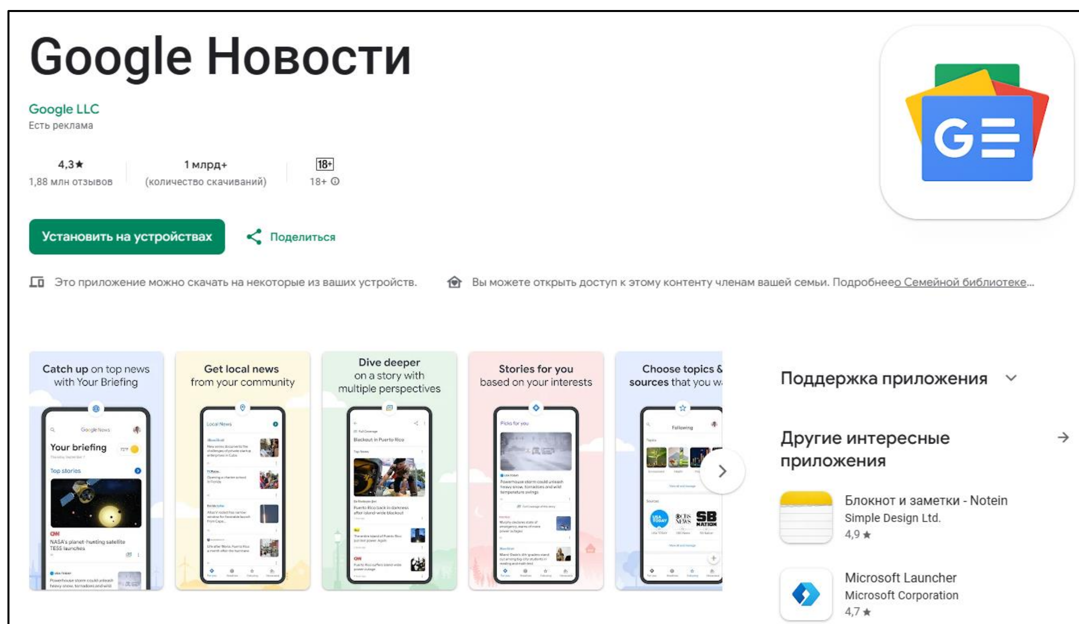


Рисунок 1.1 – Застосунок “Google News”

1) переваги: широкий спектр новин з різноманітних джерел, високий рівень персоналізації стрічки новин, зручний пошук, можливість збереження статей для подальшого читання;
2) недоліки: не завжди висока якість джерел новин, обмежені можливості для соціальної взаємодії (коментування, поширення).

- Beloud (див. рис. 1.2):

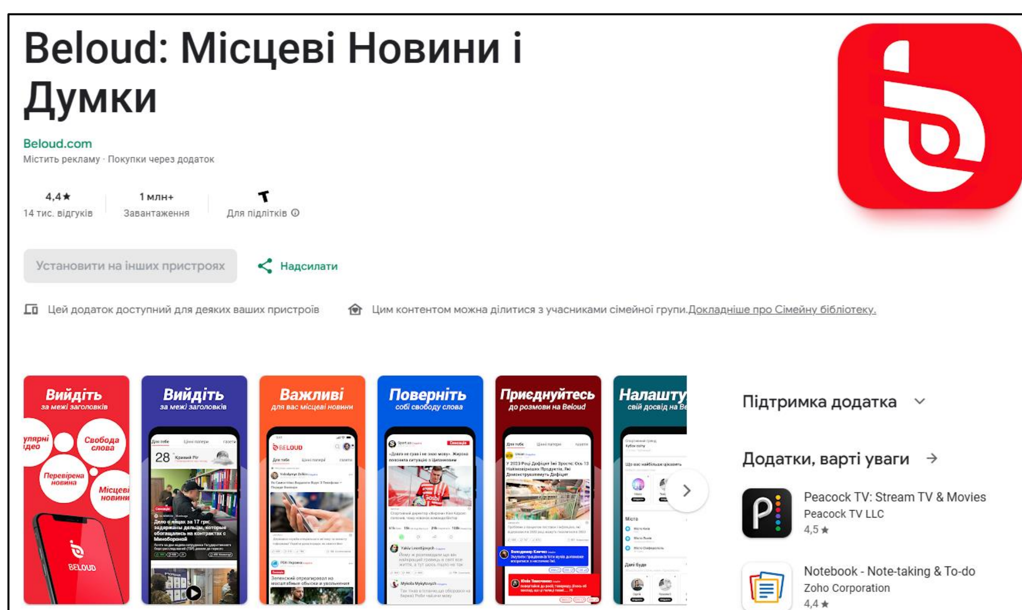


Рисунок 1.2 – Застосунок “Beloud”

1) переваги: акцент на українські локальні та регіональні новини, можливість додавання власних джерел новин;
2) недоліки: обмежені можливості персоналізації, відсутність соціальних функцій, дизайн інтерфейсу потребує вдосконалення.

- NYTimes (див. рис. 1.3):

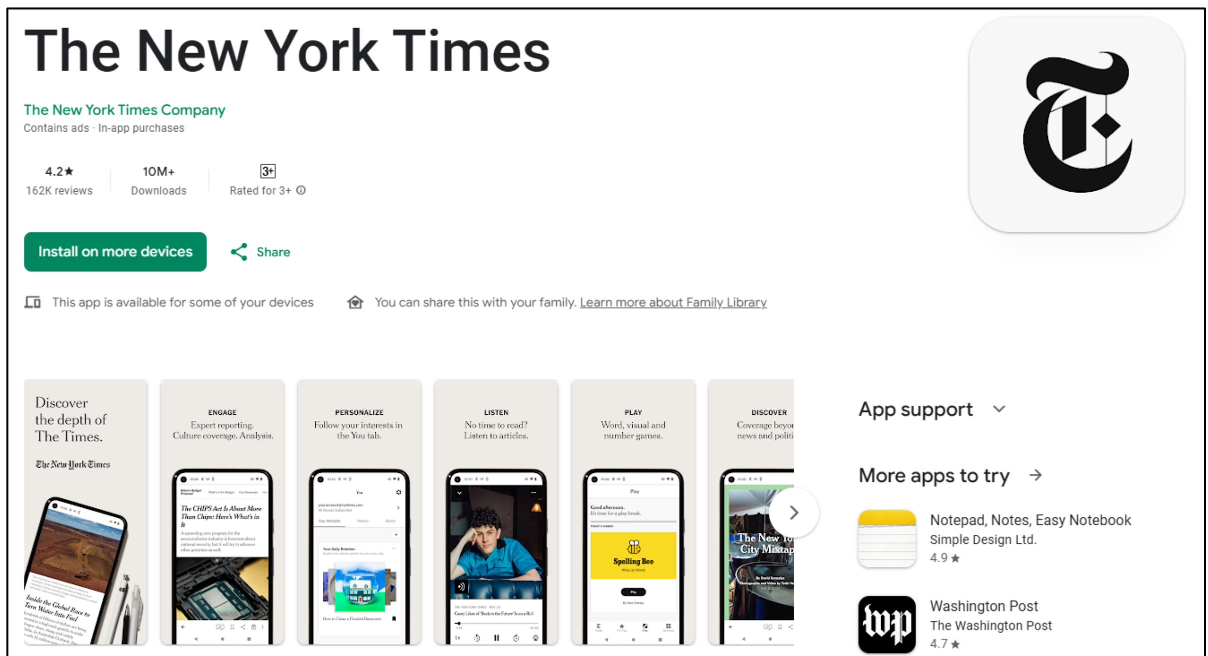


Рисунок 1.3 – Застосунок “The New York Times”

1) переваги: високоякісні статті від авторитетного видання The New York Times, можливість читання офлайн та прослуховування аудіо-версій статей, стильний та зручний інтерфейс;
2) недоліки: обмежене джерело новин (тільки The New York Times), обмежені можливості персоналізації та соціальної взаємодії.

- Google Discovery (див. рис. 1.4):

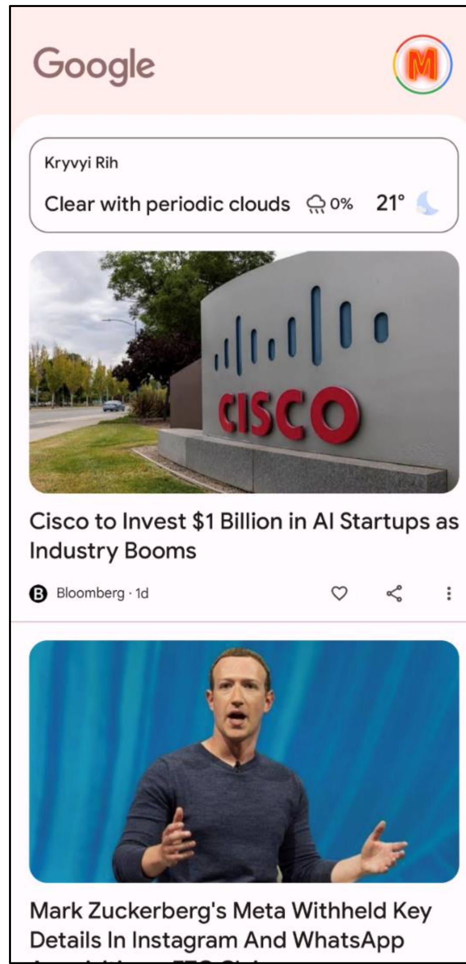


Рисунок 1.4 – Стрічка новин “Google Discovery”

1) переваги: стрічка новин формується на основі інтересів користувача, можливість збереження статей, інтеграція з іншими сервісами Google, візуально привабливий інтерфейс;
2) недоліки: якість джерел новин може варіюватися, відсутність соціальних функцій, інтерфейс іноді перевантажений інформацією.

Висновки та рекомендації:

Аналіз аналогів виявив, що кожен з них має свої сильні та слабкі сторони. Для створення власного мобільного новинного додатку варто врахувати наступні рекомендації:

- широкий вибір джерел новин: забезпечити користувачам доступ до різноманітних джерел новин, як це реалізовано в Google News та Google Discovery;

- якісний контент: надавати пріоритет авторитетним та перевіреним джерелам новин, як це робить NYTimes;
- фокус на локальні новини: врахувати важливість локальних новин для користувачів, як це реалізовано в Beloud;
- зручний та привабливий інтерфейс: забезпечити інтуїтивну навігацію та естетичний дизайн, подібний до NYTimes;
- розширені можливості персоналізації: надати користувачам інструменти для налаштування стрічки новин під свої інтереси, як це реалізовано в Google News та Google Discovery;
- унікальні функції: впровадити додаткові функції, які виділять додаток серед конкурентів (наприклад, додавання власних джерел новин, аудіо-версії статей, інтеграція з соціальними мережами).

## 1.2 Аналіз методів вирішення проблеми

У процесі розробки мобільного новинного додатку необхідно вирішити ряд ключових завдань, пов'язаних з отриманням, обробкою, представленням новин, забезпеченням персоналізації та безпеки. Розглянемо основні методи та технології, що можуть бути використані для вирішення цих проблем.

- Отримання новин:
  1) використання готових API (наприклад, News API, Google News API) дозволяє отримувати новини з різних джерел в структурованому форматі, що спрощує їх обробку та відображення. Однак, такі API можуть мати обмеження щодо кількості запитів та доступних джерел.
- Обробка та зберігання новин:
  1) room – це бібліотека, що спрощує роботу з SQLite у Android-додатках. Вона надає абстракцію над SQLite, дозволяючи працювати з даними у вигляді об'єктів та використовувати зручніші запити;
  2) використання кешу (наприклад, в оперативній пам'яті або на диску) дозволяє зменшити кількість запитів до сервера та прискорити

завантаження новин. Однак, кеш потребує періодичного оновлення для забезпечення актуальності новин.

- Відображення новин:
  1) для відображення списків новин можна використовувати RecyclerView (Android) / UITableView (iOS), які залишаються актуальними, особливо у проєктах без Jetpack Compose [1], або LazyColumn (Jetpack Compose), що підходить для різних платформ, ефективно відображає великі списки, покращує продуктивність та має додаткові можливості (роздільники, анімації, sticky headers).
- Персоналізація:
  1) фільтрація за категоріями;
  2) надання користувачам можливості вибирати цікаві їм категорії новин та відображати лише релевантний контент;
  3) рекомендаційні системи;
  4) використання алгоритмів машинного навчання для аналізу інтересів користувача та формування персоналізованих рекомендацій новин.
- Безпечна аутентифікація та авторизація:
  1) google OAuth – це сервіс аутентифікації від Google, що дозволяє користувачам входити у додатки та веб-сайти, використовуючи свої облікові записи Google. Це спрощує процес реєстрації та входу для користувачів, оскільки їм не потрібно створювати нові облікові записи та запам'ятовувати нові паролі [2];
  2) google Passkeys (див. рис. 1.5) – це нова технологія аутентифікації, що дозволяє користувачам входити у додатки та веб-сайти без використання паролів. Замість паролів використовуються біометричні дані (наприклад, відбиток пальця або розпізнавання обличчя) або спеціальні ключі безпеки [3].

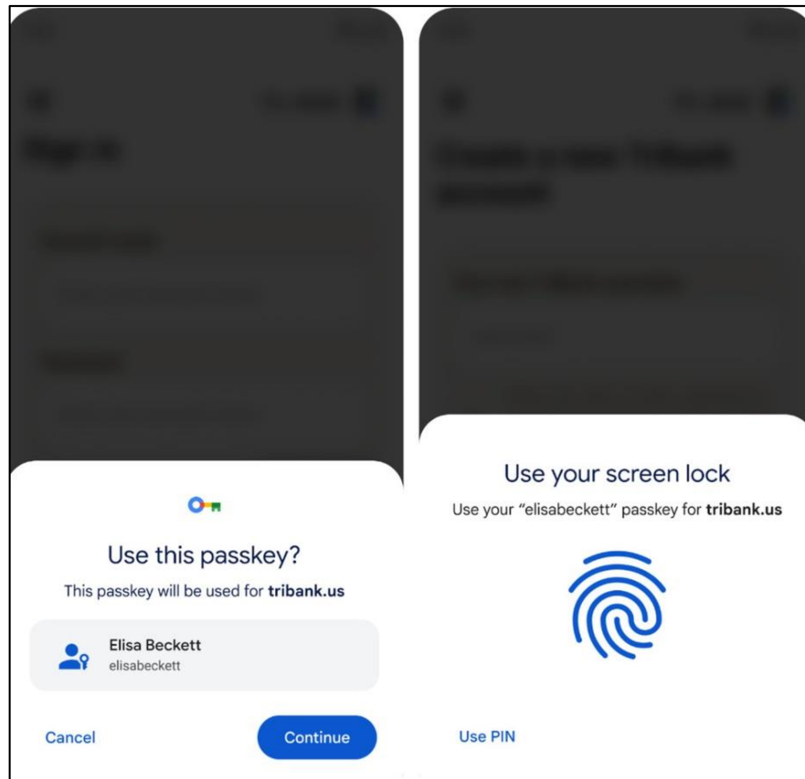


Рисунок 1.5 – Приклад з авторизацією "Google Passkeys"

### 1.3 Цілі та завдання кваліфікаційної роботи

У сучасному інформаційному суспільстві, де потік новин зростає з кожним днем, мобільні додатки стали основним інструментом для отримання актуальної інформації. Вони надають користувачам можливість бути в курсі подій у будь-який час та в будь-якому місці, забезпечуючи доступ до новин з різних джерел, персоналізацію контенту та зручну взаємодію.

Метою даної кваліфікаційної роботи є створення мобільного додатка для отримання та перегляду новин на мобільних пристроях.

Об'єктом дослідження є процес розробки мобільного новинного додатку, що відповідає сучасним вимогам та потребам користувачів. Це включає в себе не лише технічні аспекти створення програмного продукту, але й аналіз ринку, вивчення конкурентів, визначення цільової аудиторії та її потреб. Дослідження охоплює весь життєвий цикл додатку: від ідеї та проєктування до розробки, тестування, впровадження та подальшої підтримки.

Предметом розробки виступає безпосередньо мобільний новинний додаток який включає функціональні можливості, дизайн інтерфейсу, архітектурні рішення, технологічний стек та принципи взаємодії з користувачем, що забезпечують ефективність та зручність використання.

Актуальність дослідження зумовлена зростаючою популярністю мобільних пристроїв та попитом на якісні новинні додатки. Користувачі очікують від таких додатків не лише оперативного доступу до новин, але й персоналізованого контенту, зручної навігації, привабливого дизайну та високої швидкодії.

В рамках дослідження буде проведено детальний аналіз існуючих новинних додатків, таких як Google News, Beloud, NYTimes та Google Discovery. Це дозволить виявити сильні та слабкі сторони конкурентів, визначити найкращі практики та унікальні особливості, які можуть бути використані при розробці власного додатку.

Особлива увага буде приділена вибору оптимальних технологій та інструментів розробки. Сучасні мобільні платформи, такі як Android та iOS, пропонують широкий спектр можливостей для створення потужних та функціональних додатків. Використання мови програмування Kotlin, декларативного UI-фреймворку Jetpack Compose, архітектурних патернів MVVM та Clean Architecture, а також спеціалізованих бібліотек, таких як Hilt, Retrofit, Room та Coil, дозволить створити додаток, що відповідає найвищим стандартам якості та продуктивності.

## 1.4 Вимоги до програмного забезпечення

### 1.4.1 Функціональні вимоги

Додаток повинен забезпечувати агрегацію новин з різноманітних джерел, включаючи авторитетні вітчизняні та міжнародні видання, а також спеціалізовані ресурси різних тематик. Користувачі повинні мати можливість легко знаходити потрібну інформацію завдяки інтуїтивній навігації та ефективному пошуку за ключовими словами, категоріями, датою публікації

або джерелом. Важливою функцією є персоналізація контенту, що дозволяє користувачам налаштовувати стрічку новин під свої інтереси шляхом вибору тематичних категорій, збереження уподобань щодо джерел та використання рекомендаційних алгоритмів. Для забезпечення безперервного доступу до новин, додаток повинен підтримувати офлайн-режим, дозволяючи читати збережені новини навіть за відсутності інтернет-з'єднання. Інтеграція з соціальними мережами дозволить користувачам ділитися цікавими новинами з друзями та підписниками, сприяючи розширенню аудиторії додатку. Оперативне інформування про важливі новини за допомогою push-повідомлень підвищить залученість користувачів та стимулюватиме регулярне використання додатку.

### 1.4.2 Нефункціональні вимоги

Додаток повинен забезпечувати високу продуктивність та швидке завантаження на різних мобільних пристроях, включаючи моделі з обмеженими ресурсами. Це вимагає оптимізації процесів завантаження даних, обробки зображень та рендерингу інтерфейсу, а також використання кешування даних та зображень, асинхронного завантаження контенту та адаптивного дизайну.

Надійність та відмовостійкість є важливими аспектами, тому додаток повинен бути стійким до збоїв та помилок, забезпечуючи безперебійну роботу навіть в умовах нестабільного мережевого з'єднання. Це досягається шляхом ретельного тестування, обробки виняткових ситуацій та використання резервних механізмів.

Захист персональних даних користувачів є пріоритетом. Необхідно використовувати надійні методи аутентифікації (наприклад, Google Auth, Passkeys), шифрування даних при передачі та зберігання паролів у хешованому вигляді. Крім того, важливо захистити додаток від потенційних вразливостей, таких як SQL-ін'єкції та XSS-атаки.

Інтерфейс додатку повинен бути інтуїтивно зрозумілим та простим у навігації, використовуючи звичні елементи управління, чітку структуру та зрозумілі підказки. Це забезпечить позитивний користувацький досвід та зручність взаємодії з додатком.

Додаток повинен коректно працювати на різних мобільних пристроях з різними версіями операційних систем (Android та iOS). Це вимагає використання адаптивного дизайну, ретельного тестування на різних пристроях та врахування особливостей кожної платформи.

### 1.5 Реалізація алгоритмів

Алгоритм отримання новин забезпечує ефективну взаємодію з зовнішніми джерелами інформації. За допомогою News API, додаток отримує структуровані дані у форматі JSON, що містять заголовки, описи, зображення та інші метадані новин з різноманітних джерел. Далі, ці дані обробляються та зберігаються у локальній базі даних Room, що використовує SQLite для забезпечення офлайн-доступу та кешування, що дозволяє скоротити час завантаження новин при повторному відкритті додатку та зменшити навантаження на мережу.

Алгоритм відображення новин відповідає за візуальне представлення новин користувачу. Завдяки використанню LazyColumn з Jetpack Compose, додаток ефективно відображає великі списки новин, динамічно завантажуючи лише ті елементи, які знаходяться на екрані. Кожна новина представлена у вигляді картки, що містить заголовок, короткий опис та зображення, яке завантажується асинхронно за допомогою бібліотеки Coil, що запобігає блокуванню інтерфейсу під час завантаження зображень.

Алгоритм персоналізації дозволяє адаптувати контент додатку до інтересів кожного користувача. Він складається з двох основних компонентів: фільтрації новин за категоріями, обраними користувачем, та рекомендаційної системи. Рекомендаційна система аналізує історію переглядів користувача, враховуючи такі фактори, як час перегляду, кількість відкритих статей певної

категорії, взаємодію з контентом (лайки, коментарі). На основі зібраних даних система формує персоналізовані рекомендації, пропонуючи користувачу новини, які можуть його зацікавити.

Алгоритм аутентифікації та авторизації забезпечує безпечний доступ до додатку та захист користувацьких даних. Завдяки інтеграції з Firebase Authentication, користувачі можуть входити в додаток за допомогою своїх облікових записів Google або використовувати новітню технологію Google Passkeys, що дозволяє здійснювати вхід без паролів, використовуючи біометричні дані або спеціальні ключі безпеки. Для авторизації запитів до API новинних агрегаторів використовуються JSON Web Tokens (JWT), що гарантує захист від несанкціонованого доступу до даних.

Алгоритм надсилання повідомлень дозволяє інформувати користувачів про нові статті, що відповідають їхнім інтересам. Використання Firebase Cloud Messaging забезпечує надійну та ефективну доставку push-повідомлень на пристрої користувачів. Користувачі мають можливість налаштовувати частоту та типи повідомлень, що дозволяє персоналізувати процес отримання новин.

### 1.6 Впровадження програмного забезпечення

Підготовка до публікації: Першим кроком є ретельна підготовка додатку до розміщення в офіційних магазинах мобільних додатків – Google Play Market та App Store. Цей процес включає створення облікового запису розробника, оформлення інформативної сторінки додатку з детальним описом, якісними скріншотами та привабливою іконою. Крім того, важливо визначити оптимальну цінову політику, враховуючи можливість як безкоштовного розповсюдження з вбудованою рекламою, так і платної підписки з розширеним функціоналом. Після проходження процедури модерації додаток стає доступним для завантаження користувачами.

Маркетингова стратегія: Для ефективного просування додатку та залучення цільової аудиторії розробляється комплексна маркетингова

стратегія. Вона може включати різноманітні канали комунікації: таргетовану рекламу в соціальних мережах, контекстну рекламу, публікації в профільних онлайн-виданнях та блогах, а також участь у спеціалізованих заходах та конференціях. Важливим аспектом є також пошукова оптимізація (SEO) сторінки додатку в магазинах, що забезпечить його високу видимість у результатах пошуку за релевантними ключовими словами.

Аналіз та підтримка: Після запуску додатку важливо здійснювати постійний моніторинг його використання та збір зворотного зв'язку від користувачів. Це дозволить виявити потенційні проблеми, отримати цінні ідеї для покращення та забезпечити безперервний розвиток продукту. Для аналізу ключових метрик використання (кількість активних користувачів, час сеансів, кількість переглядів новин тощо) використовуються інструменти аналітики, такі як Firebase Analytics, Google Analytics, Flurry.

Регулярні оновлення з новими функціями, виправленням помилок та оптимізацією продуктивності сприяють підтримці зацікавленості аудиторії та забезпечують довгостроковий успіх додатку на конкурентному ринку.

# 2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Вибір технологій рішення проблеми

Для успішної реалізації мобільного новинного додатку необхідно обрати оптимальний стек технологій, який забезпечить високу продуктивність, зручність розробки та підтримки, а також можливість масштабування у майбутньому.

- Мова програмування та UI-фреймворк:
  1) kotlin: сучасна, лаконічна та безпечна мова програмування, що є офіційною та основною мовою для розробки Android-додатків а також використовується для розробки під різні платформи;
  2) jetpack Compose: декларативний UI-фреймворк від Google, що дозволяє створювати інтерфейси користувача у більш простий та інтуїтивно зрозумілий спосіб, ніж традиційні підходи на основі XML.
- Архітектура:
  1) mvvm (Model-View-ViewModel): архітектурний патерн, що дозволяє розділити логіку представлення даних (View), логіку роботи з даними (Model) та логіку взаємодії між ними (ViewModel). Це забезпечує кращу структуру коду, легкість тестування та підтримки;
  2) clean Architecture: архітектурний підхід, що дозволяє розділити додаток на незалежні шари, що відповідають за різні аспекти роботи (презентація, бізнес-логіка, доступ до даних). Це забезпечує високу гнучкість, масштабованість та можливість повторного використання коду.
- Бібліотеки та інструменти:
  1) hilt: бібліотека для впровадження залежностей (dependency injection), що спрощує управління життєвим циклом об'єктів та їх взаємодією [4];

2) retrofit: бібліотека для роботи з мережею, що дозволяє легко взаємодіяти з API новинних агрегаторів [5];
3) room: бібліотека для роботи з локальною базою даних SQLite, що спрощує зберігання та отримання новин [6];
4) coil: бібліотека для завантаження та кешування зображень, що забезпечує ефективне відображення зображень новин [7];
5) kotlin Coroutines: фреймворк для асинхронного програмування, що дозволяє спростити роботу з мережевими запитами та операціями з базою даних [8];
6) firebase Authentication [9]: сервіс від Google для аутентифікації користувачів, що підтримує Google Auth та інші методи аутентифікації;
7) firebase Cloud Messaging [9]: сервіс від Google для надсилання push-повідомлень, що дозволяє інформувати користувачів про нові статті.

- Додаткові бібліотеки:
  1) jetpack Navigation: бібліотека для організації навігації між екранами додатку;
  2) androidX: набір бібліотек, що розширюють стандартні можливості Android SDK та забезпечують зворотну сумісність;
  3) material Design 3 (Material You): набір рекомендацій щодо дизайну інтерфейсів від Google, що дозволяє створювати сучасні та зручні інтерфейси користувача.
- Обґрунтування вибору:
  1) kotlin та Jetpack Compose: сучасний та ефективний стек технологій для розробки Android-додатків, що забезпечує високу продуктивність та зручність розробки;
  2) mvvm та Clean Architecture: архітектурні підходи, що забезпечують хорошу структуру коду, легкість тестування та підтримки, а також високу гнучкість та масштабованість;

3) hilt, Retrofit, Room, Coil, Kotlin Coroutines: бібліотеки та інструменти, що спрощують роботу з мережею, базою даних, зображеннями та асинхронним програмуванням;
4) firebase Authentication та Firebase Cloud Messaging: сервіси від Google, що забезпечують безпечну аутентифікацію користувачів та надсилання push-повідомлень.

## 2.2 Вибір мови і IDE

Для розробки мобільного новинного додатку обрано сучасний та ефективний стек технологій, що забезпечує високу продуктивність, зручність розробки та підтримки, а також можливість масштабування у майбутньому.

Мова програмування Kotlin (див. рис. 2.1) [10] обрана завдяки своїй лаконічності, безпеці та повній сумісності з Java, що дозволяє використовувати існуючі бібліотеки та фреймворки, а також спрощує асинхронне програмування завдяки вбудованій підтримці корутин.

Рисунок 2.1 – Логотип мови “Kotlin”

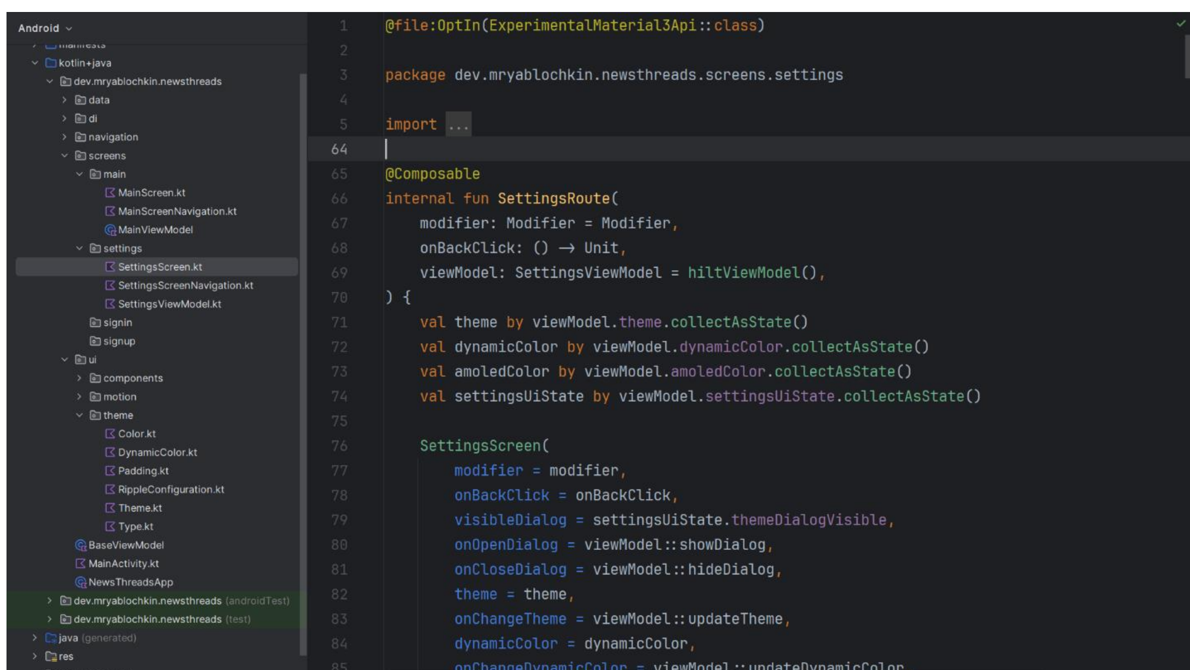


Рисунок 2.2 – Проєкт в IDE

Інтегроване середовище розробки Android Studio (див. рис. 2.3) [11] забезпечує потужний інструментарій для розробки, тестування та зневадження Android-додатків, включаючи інтелектуальний редактор коду, візуальний редактор макетів, вбудований емулятор Android, інструменти зневадження, інтеграцію з Gradle та підтримку Jetpack Compose.

Рисунок 2.3 – Логотип IDE “Android Studio”

Jetpack Compose (див. рис. 2.4), як декларативний UI-фреймворк, дозволяє створювати сучасні та інтерактивні інтерфейси користувача з меншими зусиллями, ніж традиційні підходи на основі XML.

Рисунок 2.4 – Логотип UI-фреймворку “Jetpack Compose”

Архітектурні патерни MVVM (Model-View-ViewModel) та Clean Architecture забезпечують чітку структуру коду, розділення відповідальності між компонентами, полегшують тестування та підтримку, а також сприяють масштабованості та гнучкості додатку.

Бібліотеки Hilt, Retrofit, Room та Coil спрощують роботу з впровадженням залежностей, мережевими запитами, локальною базою даних

та завантаженням зображень відповідно, забезпечуючи ефективну та надійну роботу додатку.

Firebase Authentication та Firebase Cloud Messaging використовуються для безпечної аутентифікації користувачів та надсилання push-повідомлень, що є важливими компонентами сучасного мобільного додатку.

### 2.3 Вибір дизайн системи користувальницького інтерфейсу та його розробка

Дизайн-система є важливим етапом у розробці, оскільки вона визначає візуальний стиль та взаємодію з користувацьким інтерфейсом. Дизайн-система – це набір правил, обмежень та принципів, що включає компоненти інтерфейсу, кольорову палітру, типографіку, іконки, анімації та інші елементи, а також правила їх використання. Вона забезпечує послідовність дизайну, прискорює розробку, полегшує масштабування та підтримку додатку.

При виборі дизайн-системи необхідно враховувати платформу (Android або iOS), візуальний стиль, функціональність, гнучкість та підтримку. Для розробки новинного додатку на Android з використанням Jetpack Compose найбільш підходящим вибором є Material Design, офіційна дизайн-система від Google. Вона пропонує широкий набір компонентів та шаблонів, що відповідають сучасним тенденціям дизайну, повністю інтегрується з Jetpack Compose та має велику спільноту підтримки.

Для ефективної роботи з Material Design (див. рис. 2.5) рекомендується вивчити її принципи та гайдлайни, використовувати готові компоненти Jetpack Compose та адаптувати дизайн-систему до конкретних потреб додатку. Це дозволить створити зручний, привабливий та функціональний інтерфейс, що забезпечить позитивний користувацький досвід.

Рисунок 2.5 – Дизайн система “Material You / Material Design 3” від Google

Material Design 3 (M3), також відомий як Material You, є останньою версією дизайн-системи з відкритим вихідним кодом від Google, що представляє собою еволюцію Material Design. M3 призначений для створення красивих, інтуїтивно зрозумілих і адаптивних продуктів на різних платформах, включаючи Android, iOS та веб.

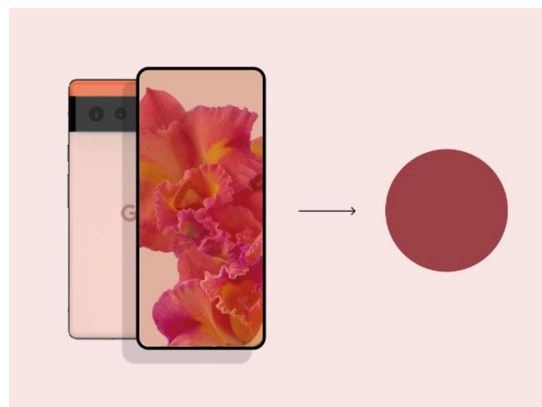

Рисунок 2.6 – Приклад кольору на основі шпалер

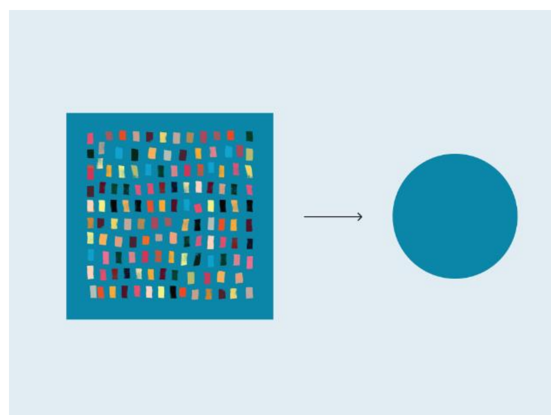

Рисунок 2.7 – Приклад кольору на основі вмісту контенту

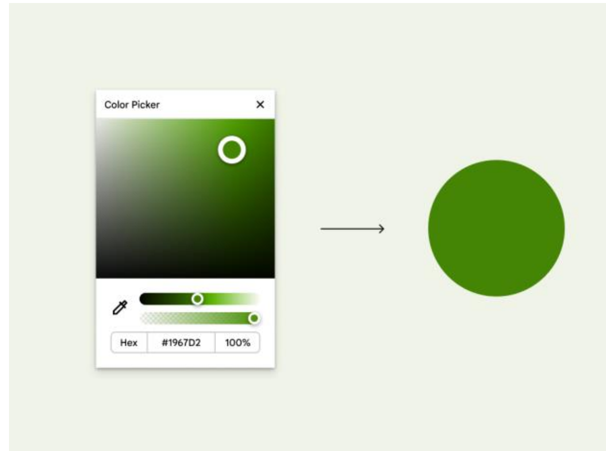


Рисунок 2.8 – Приклад вибраного кольору користувачем

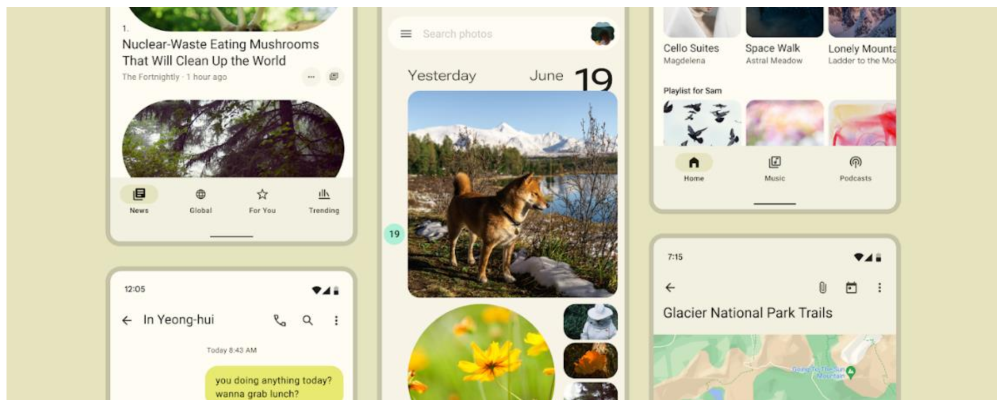


Рисунок 2.9 – Приклад динамічних кольорів у світлий темі

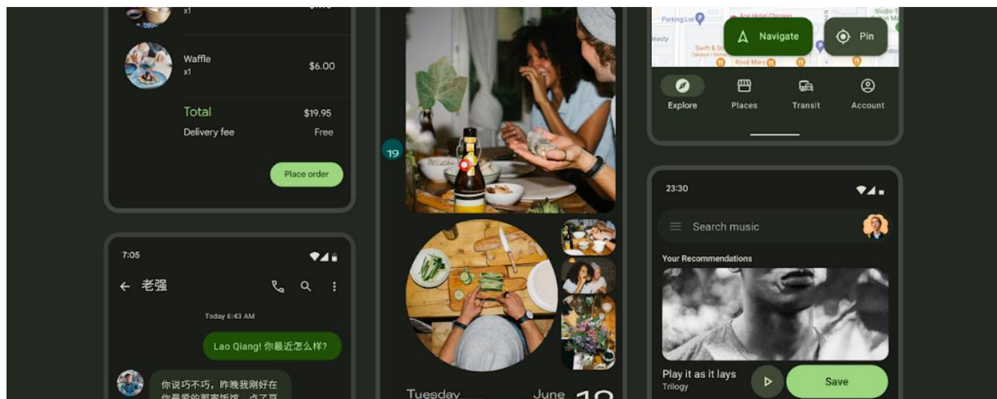


Рисунок 2.10 – Приклад динамічних кольорів у темній темі

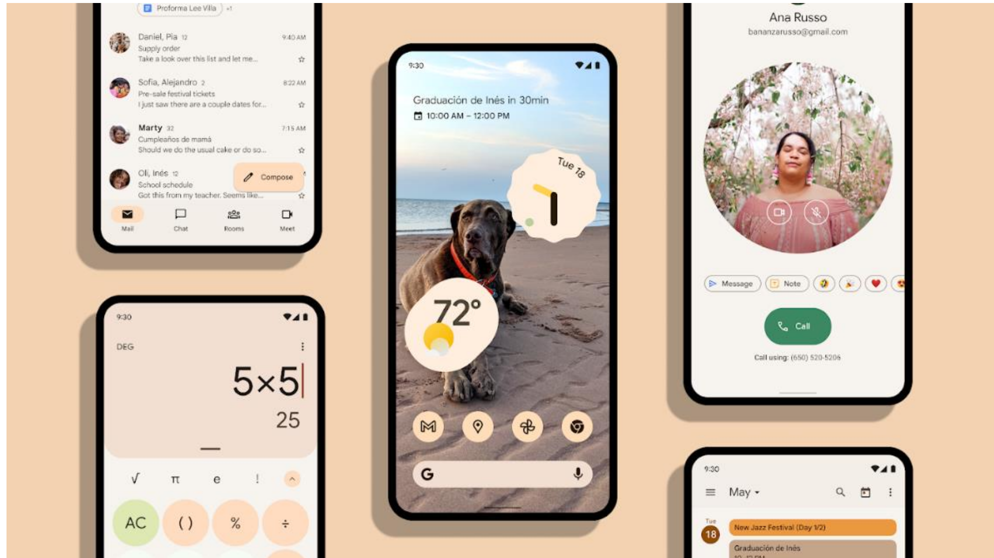


Рисунок 2.11 – Приклад динамічних кольорів на рівні всій системи

Основні особливості Material Design 3 включають динамічні кольори (див. рис. 2.6 – 2.11), що дозволяють створювати персоналізовані колірні палітри на основі шпалер користувача, оновлену систему компонентів з акцентом на більшій гнучкості та адаптивності, нову типографіку для кращої читабельності, більше анімацій та руху для більш живого інтерфейсу, а також повну інтеграцію з Jetpack Compose для спрощення розробки сучасних Android-додатків.

Використання Material Design 3 надає ряд переваг, включаючи сучасний та привабливий дизайн, що відповідає останнім тенденціям, персоналізацію для кожного користувача, покращену зручність використання завдяки оновленим компонентам, типографіці та анімаціям, гнучкість та адаптивність дизайну до різних пристроїв та розмірів екрану, а також простоту розробки завдяки інтеграції з Jetpack Compose (див. рис. 2.12).

Рисунок 2.12 – Адаптація додатку під різни екрани

Щоб почати використовувати Material Design 3 [12], рекомендується переглянути офіційну документацію, використовувати готові компоненти Jetpack Compose, експериментувати з динамічними кольорами та слідкувати за оновленнями.

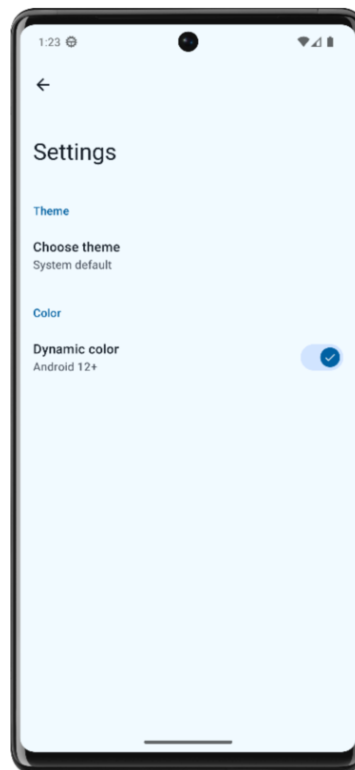


Рисунок 2.13 – Налаштування теми додатку

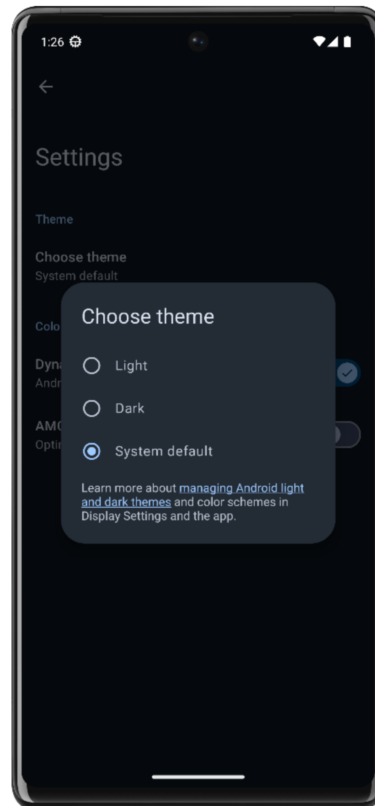


Рисунок 2.14 – Налаштування світлого, темного, або як в системі режиму

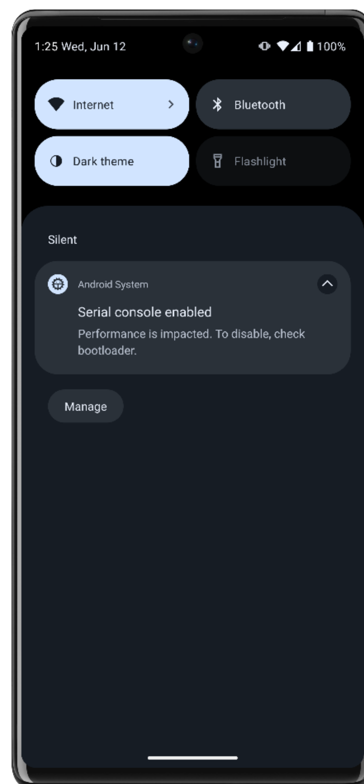


Рисунок 2.15 – Перемикання світлого або темного режиму зі шторки управління

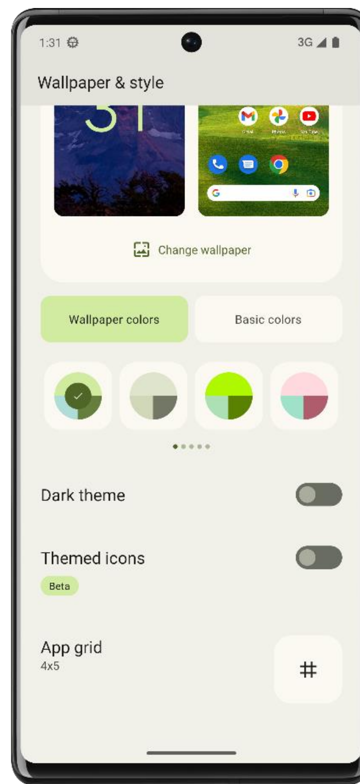


Рисунок 2.16 – Системні налаштування

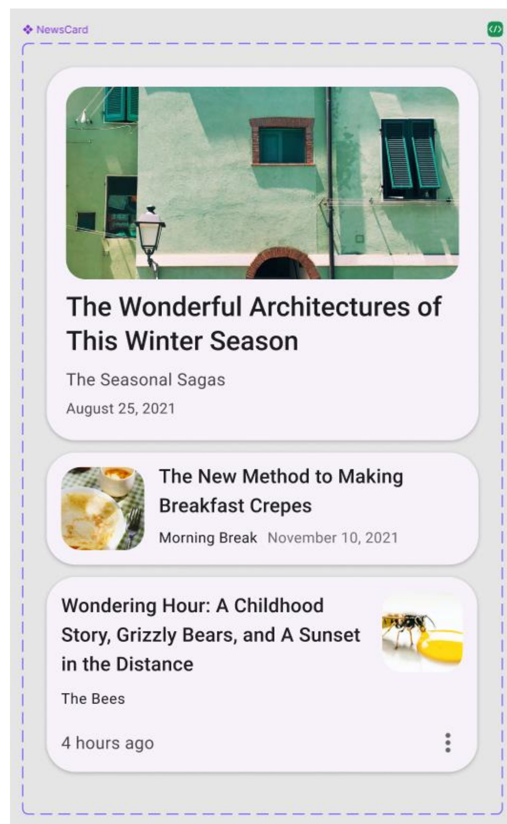


Рисунок 2.17 – Дизайн карток

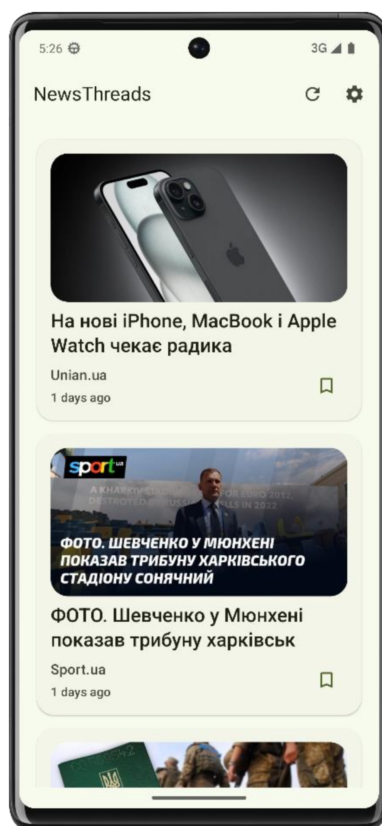


Рисунок 2.18 –Екран додатку у світлій темі

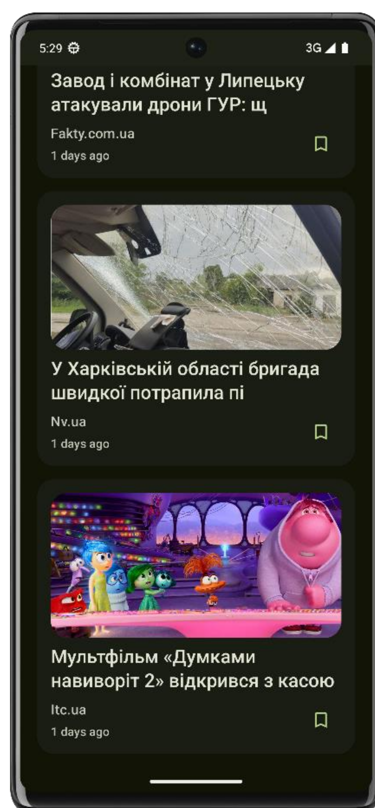


Рисунок 2.19 – Екран додатку у темній темі

Перша картка займає найбільше місця та має найбільше зображення. Заголовок "The Wonderful Architectures of This Winter Season" розміщений внизу картки, під зображенням. Під заголовком знаходиться назва джерела "The Seasonal Sagas" та дата публікації "August 25, 2021".

Друга картка менша за розміром та розташована під першою. Зображення тут займає менше місця, а заголовок "The New Method to Making Breakfast Crepes" розташований зверху. Під ним знаходиться назва джерела "Morning Break" та дата публікації "November 10, 2021".

Третя картка найменша та розташована під другою. Вона не містить зображення, а лише заголовок "Wondering Hour: A Childhood Story, Grizzly Bears, and A Sunset in the Distance", назву джерела "The Bees" та інформацію про час публікації "4 hours ago". У правому нижньому кутку картки знаходиться іконка з трьома крапками, на яку можна визначити додаткові опції.

## 2.4 Розробка та докладний опис функціональної схеми програми

Користувач: це головний актор в нашій системі (див. рис. 2.18). Користувач взаємодіє з мобільним додатком для отримання та перегляду новин. Він має можливість виконувати наступні дії:

Прецеденти:

- реєстрація: користувач має можливість створити новий обліковий запис у системі. Для цього він вводить свої персональні дані (ім'я, електронну пошту, пароль та інші необхідні дані). Після успішної реєстрації, користувач може входити в систему;
- вхід: користувач, який вже має обліковий запис, може увійти в систему, ввівши свої облікові дані (електронну пошту та пароль). Це дозволяє йому отримати доступ до персоналізованих налаштувань та функцій додатку;
- перегляд новин: користувач може переглядати новини, які доступні в додатку. Новини можуть бути представлені у вигляді списку або у

форматі стрічки новин. Користувач може обирати конкретні новини для детального перегляду;

- персоналізація контенту: користувач може налаштовувати свій додаток для відображення новин за його інтересами. Це включає вибір категорій новин (політика, спорт, технології тощо), джерел новин та інших параметрів персоналізації;
- отримання миттєвих повідомлень: користувач може отримувати миттєві повідомлення (push notifications) про нові статті, важливі події або новини, які відповідають його інтересам. Це забезпечує актуальність інформації та своєчасне інформування користувача;
- збереження у вибране: користувач може зберігати улюблені новини або статті у спеціальний розділ додатку «Вибране». Це дозволяє швидко знаходити та повертатися до важливих новин у будь-який час.

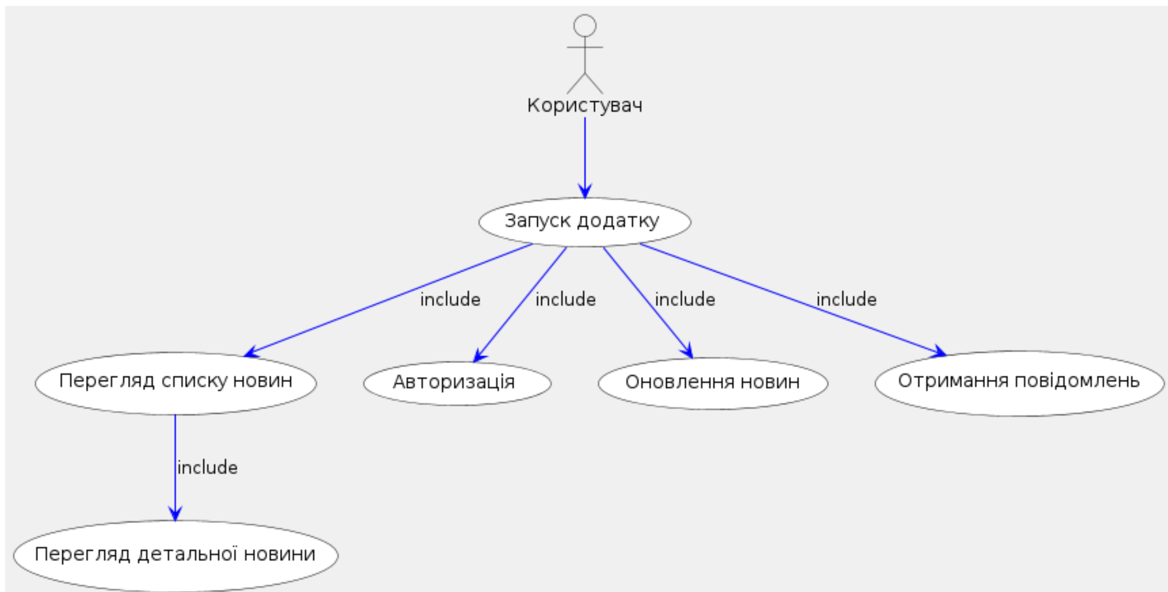


Рисунок 2.20 – Діаграма прецедентів

Функціональна діаграма відображає основні функціональні компоненти системи та взаємодію між ними (див. рис. 2.19):

- інтерфейс користувача (UI): основний компонент, з яким взаємодіє користувач. Він відповідає за відображення новин, налаштування

контенту, обробку реєстрації та входу, а також за отримання повідомлень;

- управління користувачами: відповідає за реєстрацію нових користувачів, вхід в систему, збереження та оновлення інформації про користувачів у базі даних;
- управління новинами: відповідає за отримання новин із зовнішніх джерел, їх збереження у базі даних, та відображення новин користувачам через інтерфейс користувача;
- персоналізація контенту: відповідає за налаштування контенту відповідно до вподобань користувача. Цей компонент зберігає персоналізовані налаштування в базі даних та забезпечує їх застосування при відображенні новин;
- система повідомлень: відповідає за відправлення миттєвих повідомлень користувачам. Повідомлення можуть включати нові статті, важливі новини та інші сповіщення;
- база даних: центральний компонент, який зберігає всю інформацію про користувачів, новини, персоналізацію контенту та повідомлення. Інші компоненти взаємодіють з базою даних для отримання та зберігання необхідних даних.

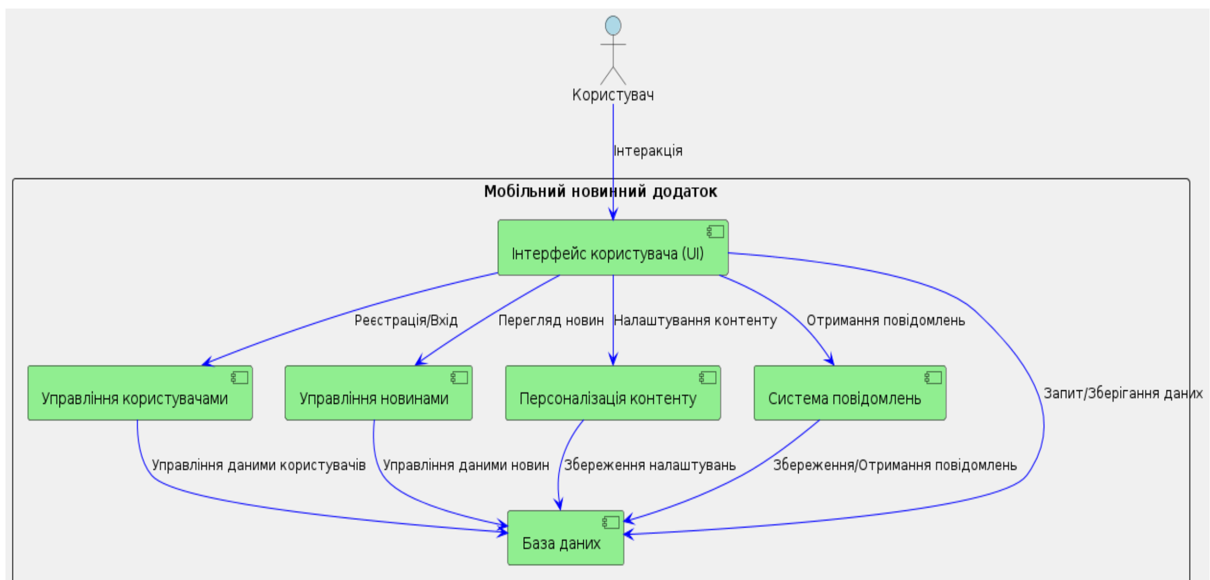


Рисунок 2.21 – Функціональна діаграма

Діаграма компонентів (Аналіз функціональних можливостей новинних додатків) (див. рис. 2.20) наочно ілюструє ключові аспекти чотирьох популярних мобільних застосунків для споживання новин: Google News, Beloud, The New York Times та Google Discovery. Користувач, як центральний актор, взаємодіє з кожним із цих додатків, використовуючи їх функціонал для отримання та обробки новин.

Google News виділяється своєю здатністю агрегувати новини з різноманітних джерел, забезпечуючи користувачеві широкий спектр інформації. Крім того, додаток має потужний компонент персоналізації контенту, який аналізує поведінку користувача та його інтереси, формуючи персоналізовану стрічку новин. Інтерфейс користувача Google News відрізняється простотою та інтуїтивністю, а також наявністю функції миттєвих повідомлень для оперативного інформування про важливі події.

Beloud, український додаток, акцентує увагу на локальному контенті та надає можливість додавати власні джерела новин. Він також пропонує функції соціальної взаємодії, дозволяючи користувачам коментувати та ділитися новинами. Однак, можливості персоналізації в Beloud обмежені, а інтерфейс користувача може потребувати деяких покращень.

The New York Times вирізняється наданням доступу до високоякісного контенту від авторитетного видання. Додаток пропонує платну підписку для доступу до преміум-контенту, а також деякі можливості персоналізації. Інтерфейс користувача відзначається стильним та зручним дизайном.

Google Discovery, подібно до Google News, агрегує новини з різних джерел та використовує алгоритми машинного навчання для персоналізації контенту. Інтерфейс Google Discovery візуально привабливий, але іноді може бути перевантажений інформацією. Додаток також має функцію миттєвих повідомлень для інформування користувача про новий контент.

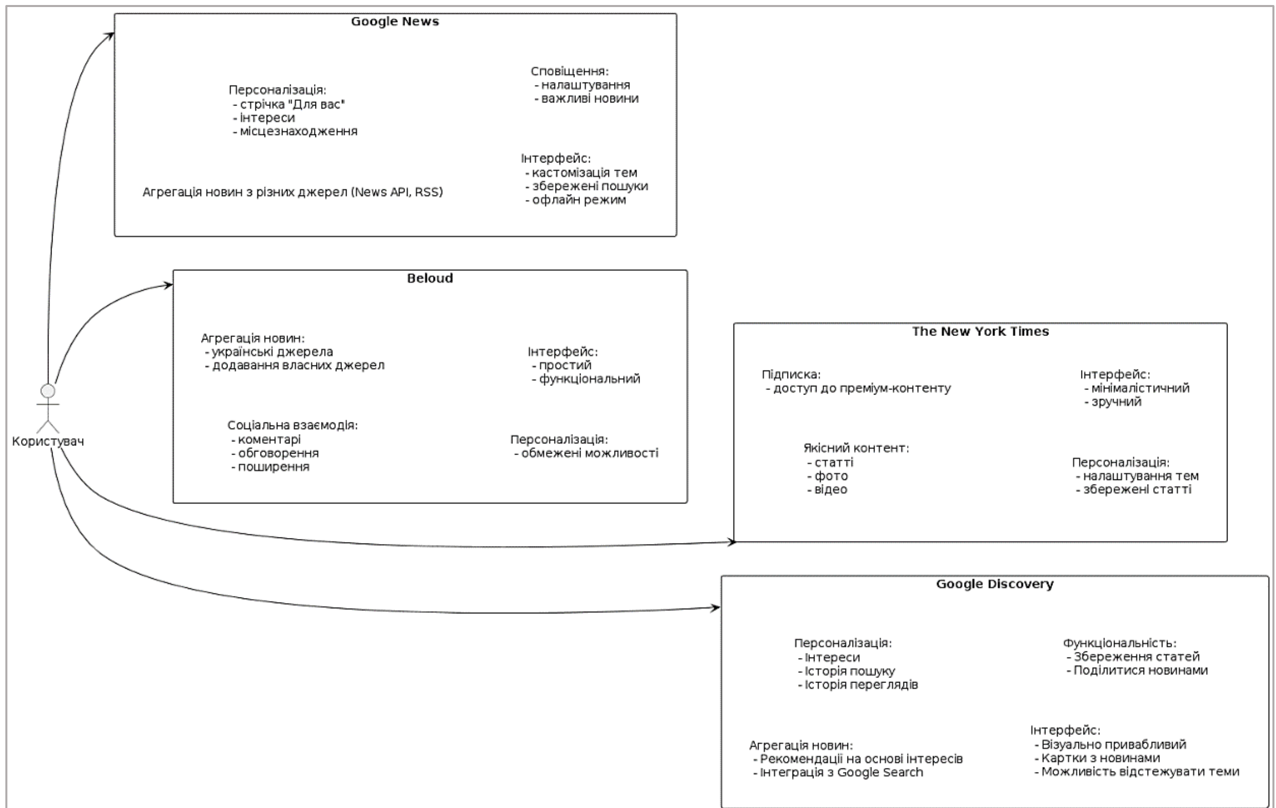


Рисунок 2.22 – Діаграма аналізу аналогів

View Layer (рівень інтерфейсу користувача):

- mainActivity: головна активність, яка містить навігацію між екраном новин та екраном входу;
- newsListScreen: екран, який відображає список новин;
- newsDetailScreen: екран, який відображає детальну інформацію про обрану новину;
- loginScreen: екран для авторизації користувача.

ViewModel Layer (рівень між view та model):

- newsListViewModel: відповідає за управління даними та логікою для NewsListScreen;
- newsDetailViewModel: відповідає за управління даними та логікою для NewsDetailScreen;
- loginViewModel: відповідає за управління даними та логікою для LoginScreen.

Model Layer (рівень в який може входить модель даних, база даних, репозиторій і т.п.):

- newsRepository: репозиторій, який керує доступом до новин (отримання з API та локального сховища);
- authRepository: репозиторій, який керує авторизацією користувача за допомогою Google OAuth та Firebase.

Data Layer (рівень даних):

- newsApiService: api-сервіс для отримання новин через HTTP запити;
- roomDatabase: локальна база даних для зберігання новин;
- googleOAuthService: сервіс авторизації через Google OAuth;
- firebaseService: сервіс авторизації та інших функцій через Firebase.

Взаємодія слоїв архітектури:

- view взаємодіє лише з viewmodel;
- viewmodel взаємодіє з model;
- model взаємодіє з data.

Технології та бібліотеки:

- kotlin: мова програмування для розробки всіх компонентів додатку;
- jetpack Compose: декларативний UI-фреймворк для створення інтерфейсу користувача;
- hilt: бібліотека для впровадження залежностей;
- retrofit: бібліотека для роботи з мережею та взаємодії з News API;
- room: бібліотека для роботи з локальною базою даних;
- coil: бібліотека для завантаження та кешування зображень;
- kotlin Coroutines: фреймворк для асинхронного програмування;
- firebase Authentication: сервіс для аутентифікації користувачів;
- firebase Cloud Messaging: сервіс для надсилання push-повідомлень;
- android Studio: інтегроване середовище розробки;
- material Design 3: дизайн-система для створення сучасного інтерфейсу користувача.

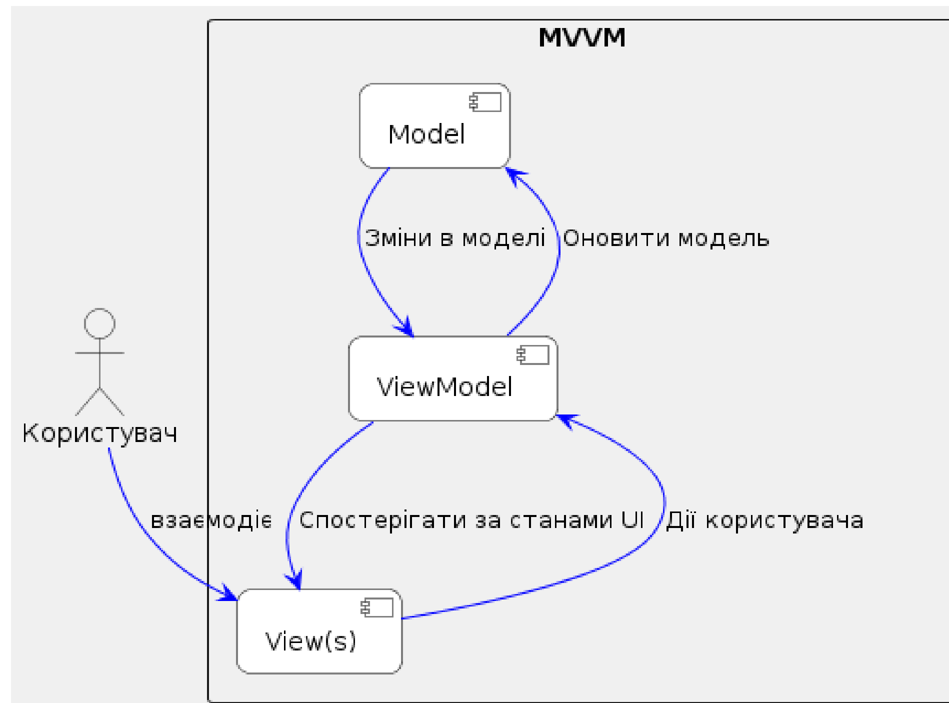


Рисунок 2.23 – Приклад MVVM архітектури адаптованої для Android та мультиплатформи

Опис архітектури (див. рис. 2.21):

- view: екрани додатку взаємодіють з ViewModel, яка надає дані та виконує бізнес-логіку;
- viewModel: viewmodel звертається до репозиторіїв для отримання даних з віддалених (NewsApiService) та локальних (RoomDatabase) джерел;
- model: репозиторії інкапсулюють всю логіку доступу до даних, використовуючи сервіси для авторизації та отримання новин.

Використання Kotlin Coroutines дозволяє обробляти асинхронні операції з даними ефективно та зручно. Hilt спрощує впровадження залежностей у різних компонентах додатку, забезпечуючи модульність та гнучкість. Retrofit надає зручний спосіб взаємодії з HTTP API, а Coil полегшує роботу з зображеннями у UI. Jetpack Compose та Material Design 3 забезпечують сучасний та привабливий інтерфейс користувача.

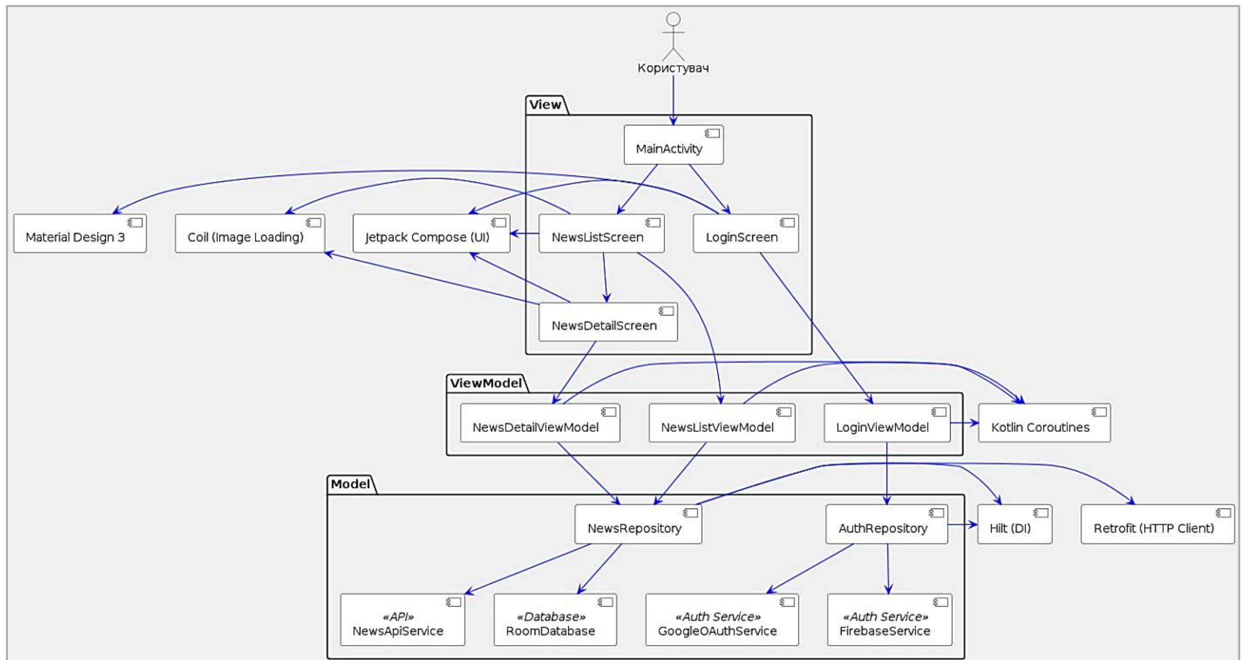


Рисунок 2.24 – Діаграма схеми архітектури

# 3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Розробка інтерфейсу програми

Наведений нижче у додатках код демонструє функціональну реалізацію екрану налаштувань для Android-додатку, використовуючи архітектурний підхід MVVM (Model-View-ViewModel) та Jetpack Compose для побудови користувацького інтерфейсу. Цей підхід забезпечує чітке розділення обов'язків між різними частинами програми, покращуючи підтримуваність, тестованість та розширюваність коду.

Код складається з двох основних функцій-компонентів: SettingsRoute і SettingsScreen. Функція SettingsRoute є точкою входу для екрану налаштувань. Вона отримує необхідні параметри, включаючи viewModel для роботи з даними налаштувань та функцію onBackClick для обробки повернення до попереднього екрану. SettingsRoute ініціалізує збирання станів теми, динамічних кольорів та AMOLED кольорів за допомогою collectAsState з viewModel. Потім ці стани передаються у компонент SettingsScreen, який відповідає за відображення користувацького інтерфейсу налаштувань. Крім того, SettingsRoute передає функції для відкриття та закриття діалогового вікна, а також для оновлення налаштувань теми та кольорів.

SettingsScreen відповідає за побудову користувацького інтерфейсу налаштувань. Він використовує компонент Scaffold, який забезпечує базову структуру інтерфейсу з верхньою панеллю (TopAppBar) та прокручуваним вмістом. На верхній панелі знаходиться кнопка повернення, яка дозволяє користувачу повернутися до попереднього екрану. Основний вміст екрану налаштувань містить кілька налаштувань, таких як вибір теми, увімкнення або вимкнення динамічних кольорів та AMOLED режиму. Для кожного налаштування використовуються спеціальні компоненти Setting та SettingsSwitch, які забезпечують відповідний інтерфейс.

Коли користувач натискає на налаштування теми, відкривається діалогове вікно ChooseThemeDialog, яке дозволяє користувачеві вибрати одну з доступних тем (світлу, темну або системну). Це діалогове вікно забезпечує інтуїтивний вибір теми за допомогою радіокнопок, які відображають поточний стан вибраної теми.

Основна логіка взаємодії полягає в наступному: коли користувач змінює будь-яке налаштування (наприклад, тему або кольори), відповідна функція ViewModel оновлює стан налаштувань. Ці зміни автоматично відображаються у користувацькому інтерфейсі завдяки двосторонній зв'язці між ViewModel і View за допомогою колекцій стану (State).

Такий підхід має кілька переваг: чіткий розподіл обов’язків між компонентами, що відповідають за різні аспекти роботи програми (робота з даними, логіка взаємодії, відображення інтерфейсу), полегшує тестування, оскільки ViewModel містить усю бізнес-логіку та не залежить від реалізації інтерфейсу. Завдяки чіткій архітектурі та використанню сучасних інструментів, таких як Jetpack Compose, підтримка та розширення функціональності програми є більш структурованим та зрозумілим процесом.

У наведеному коді нижче у додатках реалізовано компонент Setting, який використовується для відображення налаштувань в інтерфейсі користувача в Android додатку, створеному за допомогою Jetpack Compose. Цей компонент забезпечує універсальність і гнучкість у відображенні різних типів налаштувань, що дозволяє легко адаптувати його для різних випадків використання.

Компонент Setting приймає кілька параметрів: nameSetting, primaryText, secondaryText, trailingContent і onClick. Параметр nameSetting використовується для відображення назви налаштування, яка стилізована за допомогою кольору та типографіки MaterialTheme. Це дозволяє користувачам легко ідентифікувати конкретне налаштування серед інших.

Основна частина компонента реалізована за допомогою елемента ListItem, який підтримує функціональність клікабельності. Параметр

primaryText використовується для відображення основного тексту налаштування, а secondaryText для додаткового тексту, що пояснює або розширює основну інформацію. Ці текстові елементи також стилізовані за допомогою типографіки MaterialTheme, що забезпечує єдиний вигляд у всьому додатку.

Параметр trailingContent дозволяє додавати додатковий контент, який буде розташований праворуч від основного тексту. Це може бути, наприклад, перемикач або інший інтерактивний елемент, який надає користувачеві можливість взаємодіяти з налаштуванням безпосередньо в списку.

Функція onClick надає можливість визначити дію, яка виконується при натисканні на елемент налаштування. Це може бути відкриття діалогового вікна або перехід до іншого екрану з деталями налаштувань.

Наведений код нижче у додатках реалізує компонент HyperlinkText2, який дозволяє відображати текст з інтерактивними гіперпосиланнями в інтерфейсі користувача Android додатку, створеному за допомогою Jetpack Compose. Цей компонент надає гнучкість у форматуванні тексту та стилізації гіперпосилань, забезпечуючи при цьому зручність у використанні.

Компонент HyperlinkText2 приймає кілька параметрів, що дозволяють налаштувати зовнішній вигляд і поведінку тексту та гіперпосилань. Основні параметри включають: текст (text), список текстів для посилань (linkText) та відповідні URL-адреси (urls). Додаткові параметри, такі як textColor, linkTextColor, fontSize, fontWeight та linkTextDecoration, дозволяють налаштувати кольори, розмір шрифту, товщину шрифту та декорації тексту відповідно до вимог дизайну.

Внутрішня реалізація компонента використовує buildAnnotatedString для побудови анотованого рядка, що містить стилізований текст і гіперпосилання. Ця функція додає стилі до всього тексту, а також окремо до кожного гіперпосилання, забезпечуючи їх виділення кольором та підкресленням. Крім того, для кожного гіперпосилання додається анотація з відповідним URL, що робить їх інтерактивними.

Компонент BasicText використовується для відображення анотованого рядка з заданими стилями. Це забезпечує можливість легко включити компонент HyperlinkText2 у будь-яке місце інтерфейсу користувача, використовуючи параметр modifier для налаштування його розташування та зовнішнього вигляду.

Такий підхід до реалізації компонента HyperlinkText2 дозволяє створювати текстові елементи з гіперпосиланнями, які виглядають естетично привабливо та інтерактивно, що підвищує зручність користування додатком і забезпечує кращу взаємодію з користувачем.

Клас SettingsViewModel представляє собою важливий елемент архітектури MVVM у додатку на Android, побудованому за допомогою бібліотеки Hilt для впровадження залежностей. Він відповідає за керування станом інтерфейсу користувача та взаємодію з репозиторієм налаштувань користувача (UserSettingsRepository).

Анотація @HiltViewModel використовується для позначення класу ViewModel, що дозволяє Hilt автоматично створювати і надавати екземпляри цього класу з усіма необхідними залежностями. Конструктор SettingsViewModel приймає екземпляр UserSettingsRepository, що забезпечує доступ до даних налаштувань користувача.

Клас SettingsViewModel успадковується від BaseViewModel, що може містити загальну логіку для всіх ViewModel у додатку. Основна функціональність SettingsViewModel зосереджена навколо керування станом інтерфейсу користувача, який зберігається в об'єкті SettingsUiState.

Для оновлення теми користувача використовується метод updateTheme, який запускає корутину у межах viewModelScope і викликає відповідний метод репозиторію для оновлення теми. Аналогічно, методи updateDynamicColor і updateAmoledColor використовуються для оновлення відповідних налаштувань кольору.

Методи showDialog і hideDialog відповідають за відображення та приховування діалогового вікна теми. Вони змінюють стан _settingsUiState за

допомогою функції update, що створює новий екземпляр SettingsUiState з оновленим значенням властивості themeDialogVisible.

Клас SettingsViewModel забезпечує чітке розділення логіки керування станом і представленням даних у відповідності до архітектури MVVM. Це дозволяє легко тестувати логіку керування станом і робить код більш підтримуваним та масштабованим.

Наведений нижче код у додатках реалізує інтерфейс UserSettingsRepository, який відповідає за керування налаштуваннями користувача в мобільному новинному додатку. Цей репозиторій використовує SharedPreferences як механізм зберігання даних, зберігаючи налаштування у вигляді пар ключ-значення та надаючи інтерфейс для читання та запису цих значень.

Інтерфейс UserSettingsRepository визначає три властивості для зберігання налаштувань: theme (вибір теми додатку), dynamicColor (увімкнення/вимкнення динамічних кольорів) та amoledColor (увімкнення/вимкнення AMOLED кольорів). Кожна з цих властивостей має відповідний StateFlow, що дозволяє спостерігати за змінами налаштувань та реагувати на них іншим компонентам додатку, таким як ViewModel або елементи UI.

Клас UserSettingsRepositoryImpl реалізує інтерфейс UserSettingsRepository, використовуючи SharedPreferences для постійного зберігання даних та делегуючи читання та запис значень налаштувань спеціалізованим класам-делегатам: ThemePreferenceDelegate, DynamicColorPreferenceDelegate та AmoledColorPreferenceDelegate. Ці делегати відповідають за отримання значень з SharedPreferences, оновлення відповідних об'єктів StateFlow та збереження змін назад у SharedPreferences.

Функції updateTheme, updateDynamicColor та updateAmoledColor використовуються для оновлення значень налаштувань. Вони встановлюють нові значення для властивостей та оновлюють відповідні об'єкти StateFlow,

забезпечуючи синхронізацію між даними в репозиторії та станами у ViewModel.

Модуль SettingsModule відіграє важливу роль у забезпеченні впровадження залежностей (dependency injection) за допомогою Hilt у мобільному новинному додатку. Він чітко визначає, як Hilt повинен надавати екземпляри залежностей, пов'язуючи інтерфейс UserSettingsRepository з його конкретною реалізацією UserSettingsRepositoryImpl.

Аннотація @Module вказує, що цей клас є Hilt-модулем, відповідальним за визначення правил надання залежностей. Аннотація @InstallIn(SingletonComponent::class) визначає, що залежності, надані цим модулем, будуть синглтонами, тобто єдиними екземплярами на весь життєвий цикл додатку.

Абстрактна функція bindUserSettings, позначена аннотаціями @Binds та @Singleton, повідомляє Hilt, що при запиті екземпляра UserSettingsRepository слід надати екземпляр UserSettingsRepositoryImpl. Це забезпечує централізоване керування залежностями, дозволяючи легко змінювати реалізацію репозиторію без необхідності модифікації коду в місцях його використання.

Такий підхід значно спрощує процес тестування, оскільки під час тестів можна легко замінити реальну реалізацію репозиторію на мок-об'єкт, що ізолює тестовану логіку. Крім того, використання Hilt зменшує кількість шаблонного коду, оскільки розробникам не потрібно вручну створювати екземпляри UserSettingsRepository у кожному місці, де він потрібен.

```
@Module
@InstallIn(SingletonComponent::class)
abstract class SettingsModule {
    @Binds
    @Singleton
    abstract fun bindUserSettings(
        userSettingsRepositoryImpl: UserSettingsRepositoryImpl,
    ): UserSettingsRepository
}
```

Наведений код визначає sealed class під назвою ScreenRoutes, який представляє маршрути екранів у мобільному додатку. Sealed class в Kotlin – це клас, який дозволяє обмежити ієрархію наслідування, тобто всі його підкласи повинні бути оголошені в тому ж файлі, що і сам sealed class.

Кожен об'єкт даних (data object) успадковується від ScreenRoutes та містить єдину властивість route, яка є рядком, що представляє унікальний ідентифікатор маршруту. Це дозволяє використовувати ці значення в навігації між екранами додатку, наприклад, з використанням бібліотеки Jetpack Compose Navigation.

```
sealed class ScreenRoutes(val route: String) {
    data object MainScreen : ScreenRoutes("MainScreen")
    data object SettingsScreen : ScreenRoutes("SettingsScreen")
}
```

Наведений код представляє собою базову структуру Android-додатку "NewsThreadsApp", що використовує бібліотеку Hilt для впровадження залежностей. Клас NewsThreadsApp, позначений анотацією @HiltAndroidApp, успадковується від базового класу Application та слугує точкою входу для ініціалізації компонентів додатку.

Маніфест Android (AndroidManifest.xml) визначає основні параметри додатку, такі як його назва, іконка, підтримка мов з письмом справа наліво та тема оформлення. Також він оголошує головну активність MainActivity, яка буде запущена при першому відкритті додатку. Атрибут android:exported="true" дозволяє іншим додаткам запускати цю активність, а тег <intent-filter> визначає, що активність може бути запущена як головний екран додатку.

При запуску додатку система Android створює об'єкт класу NewsThreadsApp, який ініціалізує компоненти додатку, зокрема Hilt. Після цього запускається головна активність MainActivity, що відповідає за відображення інтерфейсу користувача та обробку взаємодії з користувачем.

```
@HiltAndroidApp
```

```
class NewsThreadsApp : Application()
```

Наведений фрагмент коду демонструє реалізацію головної активності (MainActivity) мобільного новинного додатку, використовуючи Jetpack Compose та архітектурний підхід MVVM (Model-View-ViewModel). Завдяки анотації @AndroidEntryPoint, Hilt забезпечує автоматичне впровадження залежностей, таких як UserSettingsRepository, в активність, що сприяє чистоті та модульності архітектури коду.

Для отримання екземпляра MainViewModel, який відповідає за управління станом UI екрану та взаємодію з UserSettingsRepository, використовується делегат viewModels. Такий підхід відповідає патерну MVVM, забезпечуючи чітке розділення між логікою інтерфейсу користувача та обробкою даних.

У методі onCreate відбувається налаштування інтерфейсу користувача за допомогою Jetpack Compose. Спочатку вмикається відображення від краю до краю екрану для більш захоплюючого досвіду користувача. Потім, використовуючи collectAsState, збираються поточні налаштування теми, динамічних кольорів та AMOLED-кольорів з MainViewModel. Ці значення використовуються для конфігурації компонуємої функції NewsThreadsTheme, що забезпечує відповідність візуального вигляду додатку вподобанням користувача.

DisposableEffect застосовується для динамічного налаштування стилів системних панелей (верхньої та нижньої) на основі вибраної теми. Це забезпечує коректне відображення панелей як у світлій, так і в темній темі.

Компонуєма функція NewsThreadsNavHost відповідає за керування навігацією між різними екранами додатку, ймовірно, використовуючи бібліотеку Jetpack Compose Navigation.

Допоміжна компонуєма функція useDarkTheme визначає, чи слід використовувати темну тему на основі налаштувань користувача. Вона повертає true, якщо вибрано темну тему або системна тема встановлена в темний режим, і false в інших випадках.

### 3.2 Розробка бази даних

Створення бази даних є ключовим етапом у розробці мобільного новинного додатку, адже вона забезпечує надійне зберігання та доступ до новин навіть у режимі офлайн. Для досягнення цієї мети було обрано бібліотеку Room, яка входить до складу архітектурних компонентів Android Jetpack та пропонує зручний інтерфейс для роботи з базами даних на основі SQLite.

Першим кроком у процесі розробки бази даних було визначення сутностей, що відображають структуру даних новин. У нашому випадку, ми створили дві сутності: Article та Source. Сутність Article включає поля для зберігання інформації про окрему новину: унікальний ідентифікатор (URL), автора, заголовок, короткий опис, URL зображення, дату публікації та інші деталі. Сутність Source зберігає інформацію про джерело новини: ідентифікатор та назву.

Наступним кроком було створення DAO (Data Access Object) – інтерфейсів, що визначають методи доступу до даних у базі. Для сутності Article було створено ArticleDao, який забезпечує методи для вставки списку новин (insertArticles), отримання всіх новин (getAllArticles) та видалення всіх новин (deleteAllArticles). Аналогічно, для сутності Source було створено SourceDao з методами для роботи з джерелами новин.

Далі, ми створили клас NewsDatabase, який успадковується від RoomDatabase та об'єднує всі сутності та DAO. Цей клас визначає версію схеми бази даних і надає абстрактні методи для отримання об'єктів DAO.

Для взаємодії з базою даних у додатку використовується репозиторій NewsRepository. Він відповідає за отримання новин з News API, їх збереження у локальній базі даних та надання доступу до збережених новин іншим компонентам додатку. Метод fetchAndCacheArticles() реалізує логіку отримання новин з API, їх парсинг та збереження у базі даних.

Важливо зазначити, що для забезпечення ефективності та чуйності інтерфейсу користувача, всі операції з базою даних виконуються асинхронно з використанням Kotlin Coroutines. Це дозволяє уникнути блокування головного потоку.

Таблиця 3.1 – Приклад таблиці бази даних “Source”

| № | id | name |
|---|---|---|
| 1 | 1 | CNN |
| 2 | 2 | BBC News |
| 3 | 3 | Reuters |

Таблиця 3.2 – Приклад таблиці бази даних “Article”

| № | url | author | title | description | urlToImage | publishedAt | content | sourceId | sourceName |
|---|---|---|---|---|---|---|---|---|---|
| 1 | https://cnn.com/article1 | John Doe | Breaking News | A major event has occurred | https://cnn.com/image1.jpg | 2024-06-12T10:00:00Z | Full article content here... | 1 | CNN |
| 2 | https://bbc.com/article2 | Jane Smith | Global News Update | Updates from around the world | https://bbc.com/image2.jpg | 2024-06-11T15:30:00Z | Detailed content for global news update... | 2 | BBC News |
| 3 | https://reuters.com/article3 | Michael Johnson | Market Report | Stock markets have fluctuated | https://reuters.com/image3.jpg | 2024-06-10T08:45:00Z | In-depth analysis of stock market fluctuations... | 3 | Reuters |
| 4 | https://cnn.com/article4 | Anna Brown | Technology Today | New advancements in technology | https://cnn.com/image4.jpg | 2024-06-09T11:00:00Z | Exploration of the latest technological advancements... | 4 | CNN |
| 5 | https://bbc.com/article5 | Richard Davis | Health News | Latest updates on health | https://bbc.com/image5.jpg | 2024-06-08T13:20:00Z | Comprehensive coverage of recent health news... | 5 | BBC News |

### 3.3 Тестування користувацького інтерфейсу

Аннотація “@Preview” є потужним інструментом у Jetpack Compose, який дозволяє розробникам візуалізувати компоненти користувацького інтерфейсу (UI) безпосередньо в Android Studio, не потребуючи запуску емулятора чи фізичного пристрою. Це значно прискорює та спрощує процес

розробки та тестування UI, оскільки зміни в коді миттєво відображаються у вікні попереднього перегляду.

Застосування "@Preview" в тестуванні UI є особливо корисним, оскільки дозволяє розробникам перевіряти різні стани компонентів, їх реакцію на зміни даних та взаємодію з користувачем. Наприклад, можна створити кілька превью з різними параметрами вхідних даних, щоб переконатися, що компонент відображається коректно у всіх можливих сценаріях. Також, використовуючи "@PreviewParameterProvider", можна автоматично генерувати кілька превью з різними наборами даних, що полегшує тестування компонентів, які залежать від зовнішніх даних.

```
@Preview
@Composable
fun PreviewChooseThemeDialogLight() {
    NewsThreadsTheme(darkTheme = false, dynamicColor = false,
amoledColor = false) {
        ChooseThemeDialog(
            visibleDialog = true,
            onCloseDialog = { },
            onChangeTheme = { },
            theme = ThemeSettings.System
        )
    }
}
```

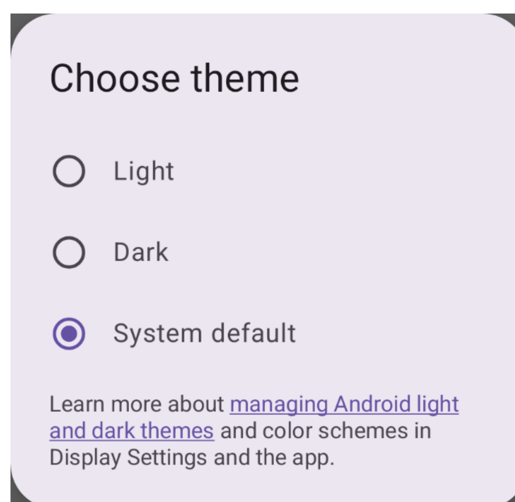


Рисунок 3.1 – Приклад Preview функції для ChooseThemeDialog

```
@Preview
```

```
@Composable
fun PreviewSettingsScreenDynamicLight() {
    NewsThreadsTheme(darkTheme = false, dynamicColor = true,
amoledColor = false) {
        SettingsScreen(
            onBackClick = { },
            visibleDialog = false,
            onOpenDialog = { },
            onCloseDialog = { },
            theme = ThemeSettings.System,
            onChangeTheme = { },
            dynamicColor = true,
            onChangeDynamicColor = { },
            amoledColor = false,
            onChangeAmoledColor = { }
        )
    }
}
```

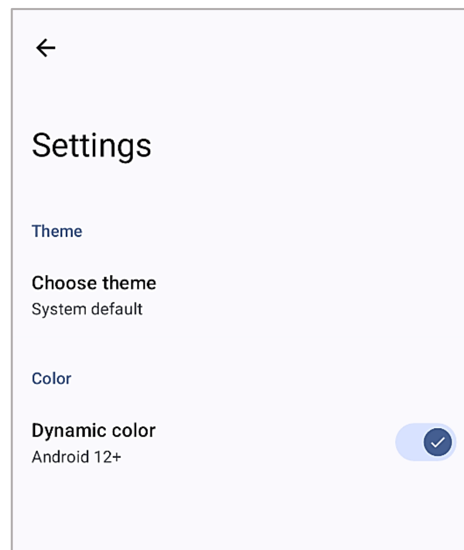


Рисунок 3.2 – Приклад Preview функції для SettingsScreen

```
@Preview
@Composable
fun PreviewSettingsTopAppBar() {
    NewsThreadsTheme {
        SettingsTopAppBar(onBackClick = { })
    }
}
```

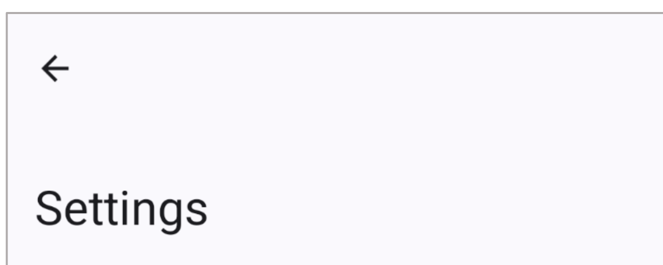


Рисунок 3.3 – Приклад Preview функції для SettingsTopAppBar

**Class com.google.samples.apps.nowinandroid.feature.foryou.ForYouScreenshotTests**

all > com.google.samples.apps.nowinandroid.feature.foryou > ForYouScreenshotTests

| 8 | 1 | 0 | 0.562s |
|---|---|---|---|
| tests | failures | skipped | duration |

**87%** successful

**Failed tests** Tests

**ForYouScreenPopulatedAndLoading**

**Reference Image** **Actual Image** **Diff Image**

Android Basics with Compose

We released the first two units of Android Basics with Compose, our first free course that teaches Android Development with Jetpack Compose to anyone; you do not need any prior programming experience other than basic computer literacy to get started. You'll learn the fundamentals of programming in Kotlin while building Android apps using Jetpack Compose, Android's modern toolkit that simplifies and accelerates native UI development. These two units are just the beginning; more will be coming soon. Check out Android Basics with Compose to get started on your Android development journey

Android Basics with Compose

We released the first two units of Android Basics with Compose, our first free course that teaches Android Development with Jetpack Compose to anyone; you do not need any prior programming experience other than basic computer literacy to get started. You'll learn the fundamentals of programming in Kotlin while building Android apps using Jetpack Compose, Android's modern toolkit that simplifies and accelerates native UI development. These two units are just the beginning; more will be coming soon. Check out Android Basics with Compose to get started on your Android development journey

Generated by Gradle 8.6 at May 10, 2024, 3:36:26 PM

Рисунок 3.4 – Приклад звіту HTML

Compose Preview Screenshot Testing – це новий експериментальний інструмент, який дозволяє автоматизувати тестування UI для компонуємих превью в Android Studio. Цей інструмент генерує опорні зображення з компонуємих превью, а потім порівнює їх зі скрінами UI пізніше. Це допомагає виявити регресії в UI.

Щоб використовувати інструмент, потрібно додати плагін до вашого проєкту та ввімкнути експериментальну властивість. Потім ви можете призначити компонуємі превью для використання в тестах скріншотів у новому наборі джерел.

Вимоги:

- android Gradle 8.5.0-beta01 або вище;
- kotlin 1.9.20 або вище.

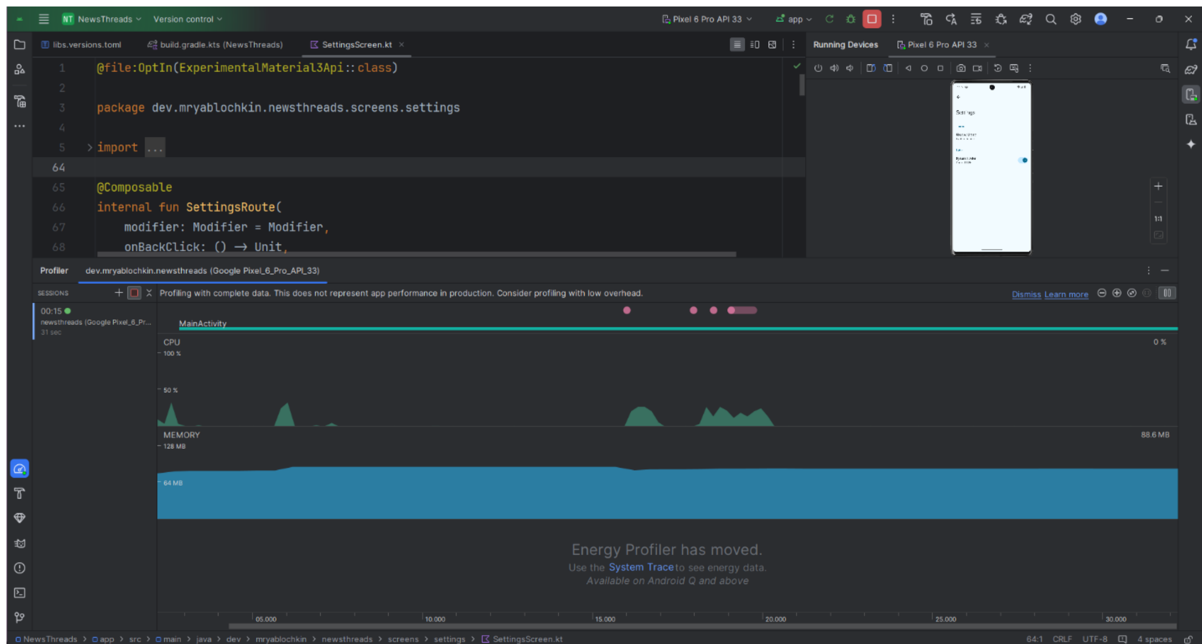


Рисунок 3.5 – Perfetto profiler

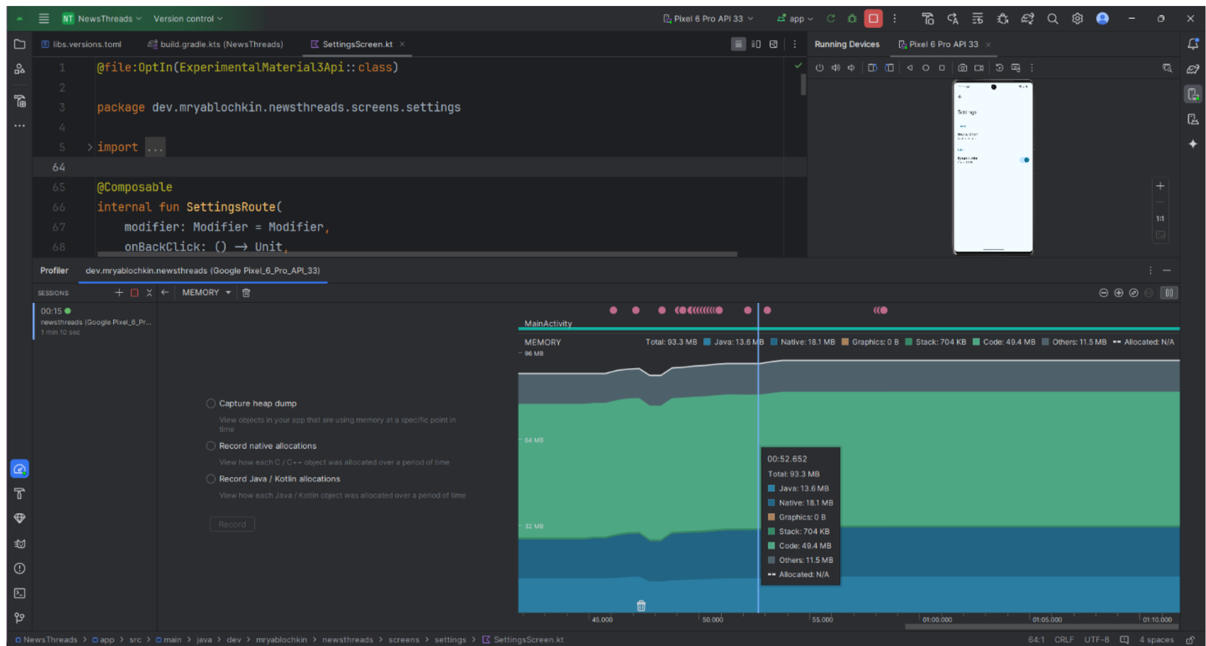


Рисунок 3.6 – Perfetto profiler у режимі MEMORY

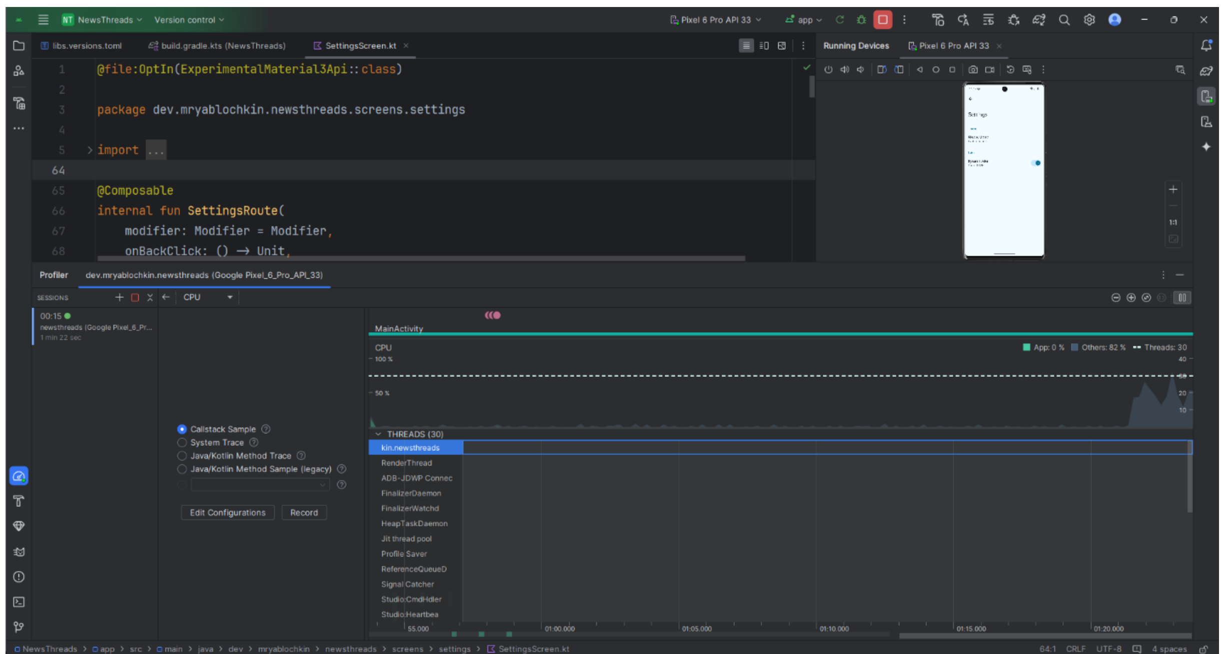


Рисунок 3.7 – Perfetto profiler у режимі CPU

Perfetto Profiler, інтегрований в Android Studio, є потужним інструментом для аналізу продуктивності додатків. Він дозволяє збирати детальні трейси системних подій, активності додатків та стану системи, надаючи розробникам глибоке розуміння поведінки їхнього програмного забезпечення.

У контексті тестування мобільного новинного додатку, Perfetto Profiler може бути використаний для:

- виявлення вузьких місць продуктивності: за допомогою Perfetto можна ідентифікувати ділянки коду, які споживають найбільше ресурсів (CPU, пам'ять, енергія) або займають найбільше часу на виконання. Це дозволяє зосередити зусилля оптимізації на найбільш критичних ділянках;
- вналіз взаємодії з мережею: perfetto дозволяє відстежувати мережеві запити, їх тривалість та обсяг переданих даних. Це допомагає виявити проблеми з мережевою взаємодією, такі як повільне завантаження контенту або надмірне споживання трафіку;
- оцінка плавності інтерфейсу: аналіз частоти кадрів та часу рендерингу дозволяє виявити причини «підвисання» інтерфейсу та забезпечити плавну анімацію та переходи між екранами;
- виявлення витоків пам'яті: perfetto може допомогти виявити об'єкти, які не звільняються з пам'яті після використання, що може призвести до витоків пам'яті та зниження продуктивності додатку.
- тестування під навантаженням: за допомогою Perfetto можна імітувати різні сценарії використання додатку з високим навантаженням, щоб оцінити його поведінку та виявити потенційні проблеми з продуктивністю в реальних умовах;
- порівняння результатів: perfetto дозволяє порівнювати трейси, отримані в різних умовах або після внесення змін до коду, що допомагає оцінити ефективність оптимізацій та виявити регресії в продуктивності.

### 3.4 Тестування програмного забезпечення

Тестування мобільного новинного додатку є критично важливим етапом розробки, оскільки воно дозволяє виявити та виправити помилки, забезпечити стабільну роботу та відповідність функціональним вимогам. Для

цього було проведено комплексний набір тестів, що охоплюють різні аспекти роботи додатку.

Модульне тестування було застосовано для перевірки коректності роботи окремих компонентів додатку, таких як класи моделі даних, репозиторії та ViewModel. Використовуючи фреймворк JUnit та бібліотеку Mockito, було створено тести, що імітують різні сценарії використання компонентів та перевіряють їх реакцію на вхідні дані та події.

Інтеграційне тестування дозволило перевірити взаємодію між різними компонентами додатку, такими як репозиторії, ViewModel та UI-компоненти. За допомогою Espresso та бібліотеки MockWebServer, було імітовано мережеві запити та перевірено коректність обробки отриманих даних, відображення новин на екрані та оновлення інтерфейсу при зміні даних.

Системне тестування проводилося на реальних пристроях та емуляторах з метою перевірки роботи додатку в цілому, включаючи завантаження новин, навігацію між екранами, аутентифікацію користувачів та надсилання повідомлень. Під час тестування було виявлено та виправлено ряд помилок, пов'язаних з відображенням новин, роботою з мережею та обробкою помилок.

# ВИСНОВКИ

У даній кваліфікаційній роботі було проведено комплексне дослідження та розробку мобільного новинного додатку з урахуванням сучасних вимог до персоналізації контенту та безпеки користувачів. Аналіз існуючих рішень дозволив виявити ключові тенденції та найкращі практики у цій сфері, а також визначити основні вимоги до розроблюваного додатку.

На основі отриманих даних було обрано оптимальний технологічний стек, що включає Kotlin, Jetpack Compose, MVVM та Clean Architecture, забезпечуючи високу продуктивність, модульність та легкість підтримки. Розроблена архітектура додатку забезпечує чітке розділення відповідальності між компонентами та ефективну взаємодію з різними джерелами даних.

Особлива увага була приділена реалізації персоналізації контенту, що дозволяє формувати унікальну стрічку новин для кожного користувача на основі його інтересів та уподобань. Також було впроваджено надійні механізми аутентифікації та авторизації користувачів, що гарантують безпеку їхніх даних.

Проведене тестування підтвердило коректність роботи додатку та його відповідність заявленим вимогам. Результати оцінки ефективності показали, що розроблений додаток забезпечує високу швидкість завантаження новин, зручну навігацію та інтуїтивно зрозумілий інтерфейс, що сприяє позитивному користувацькому досвіду.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Jetpack Compose: https://developer.android.com/develop/ui/compose
2. Google OAuth: https://developers.google.com/identity/protocols/oauth2
3. Google Passkeys: https://www.google.com/account/about/passkeys/
4. Hilt: https://developer.android.com/training/dependency-injection/hilt-android
5. Retrofit: https://square.github.io/retrofit/
6. Room: https://developer.android.com/training/data-storage/room
7. Coil: https://coil-kt.github.io/coil/compose/
8. Kotlin Coroutines: https://kotlinlang.org/docs/coroutines-overview.html
9. Firebase: https://firebase.google.com/
10. Kotlin: https://kotlinlang.org/
11. Android Studio: https://developer.android.com/studio
12. Material Design 3 / Material You: https://m3.material.io/

# Додаток NewsThreads – Код програми

SettingsScreen.kt

```
@Composable
internal fun SettingsRoute(
    modifier: Modifier = Modifier,
    onBackClick: () -> Unit,
    viewModel: SettingsViewModel = hiltViewModel(),
) {
    val theme by viewModel.theme.collectAsState()
    val dynamicColor by viewModel.dynamicColor.collectAsState()
    val amoledColor by viewModel.amoledColor.collectAsState()
    val settingsUiState by
viewModel.settingsUiState.collectAsState()

    SettingsScreen(
        modifier = modifier,
        onBackClick = onBackClick,
        visibleDialog = settingsUiState.themeDialogVisible,
        onOpenDialog = viewModel::showDialog,
        onCloseDialog = viewModel::hideDialog,
        theme = theme,
        onChangeTheme = viewModel::updateTheme,
        dynamicColor = dynamicColor,
        onChangeDynamicColor = viewModel::updateDynamicColor,
        amoledColor = amoledColor,
        onChangeAmoledColor = viewModel::updateAmoledColor
    )
}

@Composable
private fun SettingsScreen(
    modifier: Modifier = Modifier,
    onBackClick: () -> Unit,
    visibleDialog: Boolean,
    onOpenDialog: () -> Unit,
    onCloseDialog: () -> Unit,
    theme: ThemeSettings,
    onChangeTheme: (ThemeSettings) -> Unit,
    dynamicColor: Boolean,
    onChangeDynamicColor: () -> Unit,
    amoledColor: Boolean,
    onChangeAmoledColor: () -> Unit,
) {
    val scrollBehavior =
TopAppBarDefaults.exitUntilCollapsedScrollBehavior()

    Scaffold(
        modifier = Modifier
            .nestedScroll(scrollBehavior.nestedScrollConnection)
```

```
            .then(modifier),
        topBar = { SettingsTopAppBar(onBackClick = onBackClick,
scrollBehavior = scrollBehavior) },
    ) { innerPadding ->
        Column(
            modifier = Modifier
                .padding(innerPadding)
                .verticalScroll(rememberScrollState())
        ) {
            val localHapticFeedback =
LocalHapticFeedback.current
            val darkTheme = useDarkTheme(theme)

            val currentNameTheme = when (theme) {
                ThemeSettings.Light -> themeNames[0]
                ThemeSettings.Dark -> themeNames[1]
                ThemeSettings.System -> themeNames[2]
            }

            Setting(
                nameSetting =
stringResource(R.string.name_setting_theme),
                primaryText =
stringResource(R.string.primary_text_theme),
                secondaryText = currentNameTheme,
                onClick = onOpenDialog
            )

            ChooseThemeDialog(
                visibleDialog = visibleDialog,
                onCloseDialog = onCloseDialog,
                onChangeTheme = onChangeTheme,
                theme = theme
            )

            if (DynamicColorsAvailable) {
                Setting(
                    nameSetting =
stringResource(R.string.name_setting_color),
                    primaryText =
stringResource(R.string.primary_text_color),
                    secondaryText =
stringResource(R.string.secondary_text_color),
                    onClick = {
                        onChangeDynamicColor()

localHapticFeedback.performHapticFeedback(HapticFeedbackType.Tex
tHandleMove)
                    },
                    trailingContent = {
                        SettingsSwitch(
                            checked = dynamicColor,
                            onCheckedChange = {
```

```
                                onChangeDynamicColor()

localHapticFeedback.performHapticFeedback(HapticFeedbackType.Tex
tHandleMove)
                            },
                            colors = SwitchColors
                        )
                    }
                )
            }

            AnimatedVisibility(darkTheme) {
                Setting(
                    primaryText =
stringResource(R.string.primary_text_amoled),
                    secondaryText =
stringResource(R.string.secondary_text_amoled),
                    onClick = {
                        onChangeAmoledColor()

localHapticFeedback.performHapticFeedback(HapticFeedbackType.Tex
tHandleMove)
                    },
                    trailingContent = {
                        SettingsSwitch(
                            checked = amoledColor,
                            onCheckedChange = {
                                onChangeAmoledColor()

localHapticFeedback.performHapticFeedback(HapticFeedbackType.Tex
tHandleMove)
                            },
                            colors = SwitchColors
                        )
                    }
                )
            }
        }
    }
}

@Composable
private fun SettingsTopAppBar(
    modifier: Modifier = Modifier,
    onBackClick: () -> Unit,
    scrollBehavior: TopAppBarScrollBehavior? = null,
) {
    LargeTopAppBar(
        modifier = modifier,
        scrollBehavior = scrollBehavior,
        title = { Text(text =
stringResource(R.string.title_on_settings_screen)) },
        navigationIcon = {
```

```
            IconButton(onClick = onBackClick) {
                Icon(imageVector =
Icons.AutoMirrored.Default.ArrowBack, contentDescription = null)
            }
        },
    )
}

@Composable
private fun SettingsSwitch(
    modifier: Modifier = Modifier,
    checked: Boolean,
    onCheckedChange: ((Boolean) -> Unit)?,
    enabled: Boolean = true,
    contentDescription: String? = null,
    colors: SwitchColors = SwitchDefaults.colors(),
    interactionSource: MutableInteractionSource? = null,
) {
    Switch(
        modifier = modifier,
        checked = checked,
        onCheckedChange = onCheckedChange,
        enabled = enabled,
        colors = colors,
        interactionSource = interactionSource,
        thumbContent = {
            Crossfade(
                targetState = checked,
                animationSpec =
tween(MotionDurationTokens.MotionDurationShort3),
                label = ""
            ) { targetState ->
                Icon(
                    imageVector = if (targetState)
Icons.Default.Check else Icons.Default.Close,
                    contentDescription = contentDescription,
                    modifier =
Modifier.size(SwitchDefaults.IconSize)
                )
            }
        }
    )
}

@Composable
private fun ChooseThemeDialog(
    visibleDialog: Boolean,
    onCloseDialog: () -> Unit,
    onChangeTheme: (ThemeSettings) -> Unit,
    theme: ThemeSettings,
) {
    if (visibleDialog) {
        DialogSwitchTheme(
```

```
            onDismissRequest = onCloseDialog,
            selectedTheme = theme,
            onThemeSelected = {
                onChangeTheme(it)
                onCloseDialog()
            }
        )
    }
}

@Composable
private fun DialogSwitchTheme(
    onDismissRequest: () -> Unit,
    selectedTheme: ThemeSettings,
    onThemeSelected: (ThemeSettings) -> Unit,
) {
    Dialog(onDismissRequest = onDismissRequest) {
        Surface(
            color = AlertDialogDefaults.containerColor,
            shape = AlertDialogDefaults.shape,
            tonalElevation = AlertDialogDefaults.TonalElevation,
            shadowElevation = 10.dp
        ) {
            Column(modifier =
Modifier.verticalScroll(rememberScrollState())) {
                Text(
                    modifier = Modifier.padding(TitlePadding),
                    text =
stringResource(R.string.primary_text_theme),
                    style =
MaterialTheme.typography.headlineSmall,
                    color = MaterialTheme.colorScheme.onSurface
                )
                RadioGroup(
                    modifier = Modifier.padding(ContentPadding),
                    items = ThemeItem.themeItems,
                    selected = selectedTheme.ordinal,
                    onItemSelected = {
onThemeSelected(ThemeSettings.fromOrdinal(it)) }
                )
                HyperlinkText2(
                    modifier = Modifier.padding(BottomPadding),
                    text =
stringResource(R.string.info_bottom_content),
                    linkText =
listOf(stringResource(R.string.link_text)),
                    urls = listOf(AdditionalInfoAboutTheme)
                )
            }
        }
    }
}
```

```
@Composable
private fun RadioGroup(
    modifier: Modifier = Modifier,
    items: List<ThemeItem>,
    selected: Int,
    onItemSelected: ((Int) -> Unit)? = null,
) {
    Column(
        modifier = Modifier
            .selectableGroup()
            .then(modifier)
    ) {
        items.forEach { item ->
            RadioGroupItem(
                item = item,
                selected = selected == item.id,
                onClick = { onItemSelected?.invoke(item.id) }
            )
        }
    }
}

@Composable
private fun RadioGroupItem(
    item: ThemeItem,
    selected: Boolean,
    onClick: ((Int) -> Unit)? = null,
) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .height(50.dp)
            .selectable(
                selected = selected,
                onClick = { onClick?.invoke(item.id) },
                role = Role.RadioButton
            ),
        verticalAlignment = Alignment.CenterVertically,
    ) {
        RadioButton(selected, null,
Modifier.padding(RadioButtonPadding))
        Spacer(modifier = Modifier.width(15.dp))
        Text(text = item.title, color =
MaterialTheme.colorScheme.onSurfaceVariant)
    }
}

data class ThemeItem(val id: Int, val title: String) {
    companion object {
        val themeItems: List<ThemeItem>
            @Composable
            get() = listOf(
```

```
                ThemeItem(id = ThemeSettings.Light.ordinal,
title = themeNames[0]),
                ThemeItem(id = ThemeSettings.Dark.ordinal, title
= themeNames[1]),
                ThemeItem(id = ThemeSettings.System.ordinal,
title = themeNames[2])
            )
    }
}

private val themeNames: Array<String>
    @Composable
    get() =
LocalContext.current.resources.getStringArray(R.array.secondary_
text_theme)

internal val SwitchColors
    @Composable
    get() = SwitchDefaults.colors(
        checkedThumbColor = MaterialTheme.colorScheme.primary,
        checkedTrackColor =
MaterialTheme.colorScheme.primaryContainer,
        uncheckedThumbColor =
MaterialTheme.colorScheme.secondary,
        uncheckedTrackColor =
MaterialTheme.colorScheme.secondaryContainer,
        checkedIconColor = MaterialTheme.colorScheme.onPrimary
    )

private val TitlePadding = PaddingValues(24.dp, 24.dp, 24.dp,
16.dp)
private val ContentPadding = PaddingValues(0.dp, 0.dp, 0.dp,
10.dp)
private val RadioButtonPadding = PaddingValues(24.dp, 0.dp,
0.dp, 0.dp)
private val BottomPadding = PaddingValues(24.dp, 0.dp, 24.dp,
24.dp)
private const val AdditionalInfoAboutTheme =
    "https://support.google.com/android/answer/9730472?hl=en"
```

Setting.kt

```
@Composable
internal fun Setting(
    nameSetting: String? = null,
    primaryText: String? = null,
    secondaryText: String? = null,
    trailingContent: @Composable (() -> Unit)? = null,
    onClick: (() -> Unit)? = null,
) {
    nameSetting?.let {
        Text(
```

```
            text = it,
            color = MaterialTheme.colorScheme.primary,
            style = MaterialTheme.typography.titleSmall,
            modifier = Modifier.padding(start = 16.dp, top =
25.dp, bottom = 8.dp)
        )
    }
    ListItem(
        modifier = Modifier.clickable { onClick?.invoke() },
        headlineContent = { primaryText?.let { Text(it, style =
MaterialTheme.typography.titleMedium)} },
        supportingContent = { secondaryText?.let { Text(it) } },
        trailingContent = trailingContent
    )
}
```

HyperlinkText.kt

```
@Composable
fun HyperlinkText2(
    modifier: Modifier = Modifier,
    text: String,
    linkText: List<String>,
    urls: List<String>,
    textColor: Color =
MaterialTheme.colorScheme.onSurfaceVariant,
    linkTextColor: Color = MaterialTheme.colorScheme.primary,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontWeight: FontWeight = FontWeight.Normal,
    linkTextDecoration: TextDecoration =
TextDecoration.Underline,
) {
    val annotatedString = buildAnnotatedString {
        append(text).apply { addStyle(SpanStyle(textColor), 0,
text.length) }
        linkText.forEachIndexed { index, link ->
            val startIndex = text.indexOf(link)
            val endIndex = startIndex + link.length
            addStyle(
                style = SpanStyle(color = linkTextColor,
textDecoration = linkTextDecoration),
                start = startIndex,
                end = endIndex
            )
            addLink(LinkAnnotation.Url(urls[index]), startIndex,
endIndex)
        }
        addStyle(SpanStyle(fontSize = fontSize, fontWeight =
fontWeight), 0, text.length)
    }

    BasicText(text = annotatedString, modifier = modifier)
```

```
}
```

HiltViewModel.kt

```
@HiltViewModel
class SettingsViewModel @Inject constructor(
    private val userSettingsRepository: UserSettingsRepository,
) : BaseViewModel(userSettingsRepository) {

    private val _settingsUiState =
MutableStateFlow(SettingsUiState())
    val settingsUiState = _settingsUiState.asStateFlow()

    fun updateTheme(themeSettings: ThemeSettings) {
        viewModelScope.launch {
            userSettingsRepository.updateTheme(themeSettings)
        }
    }

    fun updateDynamicColor() {
        viewModelScope.launch {
            userSettingsRepository.updateDynamicColor()
        }
    }

    fun updateAmoledColor() {
        viewModelScope.launch {
            userSettingsRepository.updateAmoledColor()
        }
    }

    fun showDialog() {
        _settingsUiState.update { it.copy(themeDialogVisible =
true) }
    }

    fun hideDialog() {
        _settingsUiState.update { it.copy(themeDialogVisible =
false) }
    }
}

data class SettingsUiState(
    val themeDialogVisible: Boolean = false
)
```

UserSettingsRepository.kt та UserSettingsRepositoryImpl.kt

```
interface UserSettingsRepository {
    var theme: ThemeSettings
    val themeStateFlow: StateFlow<ThemeSettings>
    var dynamicColor: Boolean
```

```
    val dynamicColorStateFlow: StateFlow<Boolean>
    var amoledColor: Boolean
    val amoledColorStateFlow: StateFlow<Boolean>

    suspend fun updateDynamicColor()
    suspend fun updateAmoledColor()
    suspend fun updateTheme(themeSettings: ThemeSettings)
}

class UserSettingsRepositoryImpl @Inject constructor(
    @ApplicationContext context: Context,
) : UserSettingsRepository {

    private val preferences: SharedPreferences =
context.getSharedPreferences(NAME_PREFERENCES, MODE)

    override var theme: ThemeSettings by
ThemePreferenceDelegate(THEME_NAME, THEME_DEFAULT_VALUE)
    override val themeStateFlow = MutableStateFlow(theme)

    override var dynamicColor: Boolean by
DynamicColorPreferenceDelegate(DYNAMIC_COLOR_NAME,
DYNAMIC_COLOR_VALUE)
    override val dynamicColorStateFlow =
MutableStateFlow(dynamicColor)

    override var amoledColor: Boolean by
AmoledColorPreferenceDelegate(AMOLED_COLOR_NAME,
AMOLED_COLOR_VALUE)
    override val amoledColorStateFlow =
MutableStateFlow(amoledColor)

    override suspend fun updateTheme(themeSettings:
ThemeSettings) {
        theme = themeSettings
    }

    override suspend fun updateDynamicColor() {
        dynamicColor = !dynamicColor
    }

    override suspend fun updateAmoledColor() {
        amoledColor = !amoledColor
    }

    inner class ThemePreferenceDelegate(
        private val name: String,
        private val defaultValue: ThemeSettings
    ) : ReadWriteProperty<Any?, ThemeSettings> {
        override fun getValue(thisRef: Any?, property:
KProperty<*>): ThemeSettings {
```

```
            return
ThemeSettings.fromOrdinal(preferences.getInt(name,
defaultValue.ordinal))
        }

        override fun setValue(thisRef: Any?, property:
KProperty<*>, value: ThemeSettings) {
            themeStateFlow.value = value
            preferences.edit {
                putInt(name, value.ordinal)
            }
        }
    }

    inner class DynamicColorPreferenceDelegate(
        private val name: String,
        private val defaultValue: Boolean
    ) : ReadWriteProperty<Any?, Boolean> {
        override fun getValue(thisRef: Any?, property:
KProperty<*>): Boolean {
            return preferences.getBoolean(name, defaultValue)
        }

        override fun setValue(thisRef: Any?, property:
KProperty<*>, value: Boolean) {
            dynamicColorStateFlow.value = value
            preferences.edit {
                putBoolean(name, value)
            }
        }
    }

    inner class AmoledColorPreferenceDelegate(
        private val name: String,
        private val defaultValue: Boolean
    ) : ReadWriteProperty<Any?, Boolean> {
        override fun getValue(thisRef: Any?, property:
KProperty<*>): Boolean {
            return preferences.getBoolean(name, defaultValue)
        }

        override fun setValue(thisRef: Any?, property:
KProperty<*>, value: Boolean) {
            amoledColorStateFlow.value = value
            preferences.edit {
                putBoolean(name, value)
            }
        }
    }

    companion object {
        const val NAME_PREFERENCES = "NewsThreadsPreferences"
        const val MODE = Context.MODE_PRIVATE
```

```
        const val THEME_NAME = "current_theme"
        val THEME_DEFAULT_VALUE = ThemeSettings.System
        const val DYNAMIC_COLOR_NAME = "dynamic_color"
        const val DYNAMIC_COLOR_VALUE = true
        const val AMOLED_COLOR_NAME = "amoled_color"
        const val AMOLED_COLOR_VALUE = false
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <application
        android:name=".NewsThreadsApp"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.NewsThreads"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.NewsThreads">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

MainActivity.kt

```
@AndroidEntryPoint
class MainActivity : ComponentActivity() {

    @Inject
    lateinit var userSettingsRepository: UserSettingsRepository

    private val viewModel by viewModels<MainViewModel>()
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            val theme by viewModel.theme.collectAsState()
            val dynamicColor by
viewModel.dynamicColor.collectAsState()
            val amoledColor by
viewModel.amoledColor.collectAsState()
            val darkTheme = useDarkTheme(theme)

            DisposableEffect(darkTheme) {
                enableEdgeToEdge(
                    SystemBarStyle.auto(Transparent,
Transparent) { darkTheme },
                    SystemBarStyle.auto(Transparent,
Transparent) { darkTheme },
                )
                onDispose {}
            }

            NewsThreadsTheme(darkTheme, dynamicColor,
amoledColor) {
                Surface(modifier = Modifier.fillMaxSize()) {
                    NewsThreadsNavHost()
                }
            }
        }
    }
}

@Composable
internal fun useDarkTheme(theme: ThemeSettings): Boolean {
    return when (theme) {
        ThemeSettings.Light -> false
        ThemeSettings.Dark -> true
        ThemeSettings.System -> isSystemInDarkTheme()
    }
}
```

Database.kt

```
@Entity(tableName = "articles")
data class Article(
    @PrimaryKey val url: String,
    val author: String?,
    val title: String?,
    val description: String?,
    val urlToImage: String?,
    val publishedAt: String?,
    val content: String?,
    val sourceId: String?,
```

```
    val sourceName: String?
)

@Entity(tableName = "sources")
data class Source(
    @PrimaryKey val id: String,
    val name: String
)

@Dao
interface ArticleDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertArticles(articles: List<Article>)

    @Query("SELECT * FROM articles")
    fun getAllArticles(): Flow<List<Article>>

    @Query("DELETE FROM articles")
    suspend fun deleteAllArticles()
}

@Dao
interface SourceDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertSources(sources: List<Source>)

    @Query("SELECT * FROM sources")
    fun getAllSources(): Flow<List<Source>>
}

@Database(entities = [Article::class, Source::class], version =
1)
abstract class NewsDatabase : RoomDatabase() {
    abstract fun articleDao(): ArticleDao
    abstract fun sourceDao(): SourceDao
}

class NewsRepository @Inject constructor(
    private val newsApi: NewsApi,
    private val newsDatabase: NewsDatabase
) {

    suspend fun fetchAndCacheArticles() {
        try {
            val response = newsApi.getTopHeadlines()
            if (response.isSuccessful) {
                response.body()?.let { newsResponse ->

newsDatabase.articleDao().insertArticles(newsResponse.articles)

newsDatabase.sourceDao().insertSources(newsResponse.articles.map
{ it.source })
                }
```

```
            } else {
                throw NetworkErrorException("Помилка мережевого
запиту: ${response.code()}")
            }
        } catch (e: HttpException) {
             throw NetworkErrorException("HTTP-помилка:
${e.code()}")
        } catch (e: IOException) {
            throw NetworkErrorException("Помилка вводу-виводу:
${e.message}")
        } catch (e: Exception) {
            throw RepositoryException("Невідома помилка:
${e.message}")
        }
    }
}
```