ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
*(підпис)*

«___» _____ 2025 р.

# КВАЛІФІКАЦІЙНА РОБОТА

на тему

# РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ УПРАВЛІННЯ ТА ДИСТРИБУЦІЇ НОВИН НА ОСНОВІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

зі спеціальності 122 Комп’ютерні науки
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Оборожний Артем Віталійович
_____ «____»_____ 2025 р.
*(підпис)*

Науковий керівник к.пед.н., доцент, Кошова Оксана Петрівна
_____ «____»_____ 2025 р.
*(підпис)*

Рецензент

ПОЛТАВА 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка мобільного додатку для управління та дистрибуції новин на основі мікросервісної архітектури»
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Оборожний Артем Віталійович
Затверджена наказом ректора № 177-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___»_____ 2025 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки мобільних додатків.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
1.1. Мета та задачі дослідження
1.2. Визначення вимог до функціоналу мобільного додатку
1.3. Особливості використання децентралізованого шифрування
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Огляд існуючих платформ для дистрибуції новин
2.2. Порівняльний аналіз популярних рішень мобільних додатків для дистрибуції новин
2.3. Переваги та недоліки існуючих сервісів для дистрибуції новин
2.4. Використання децентралізованих систем у сучасних рішеннях безпеки та актуальні підходи до побудови API та архітектури мобільних додатків
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Опис обраних технологій та підходів для розробки мобільного додатку та API
3.2. Методологія розробки програмного забезпечення та побудова блок-схеми системи
3.3. Опис чистої архітектури та патерну MVVM
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Розробка алгоритму децентралізованого шифрування для забезпечення безпеки авторизації
4.2. Реалізація взаємодії класів мобільного додатку з використанням патерну MVVM та діаграма класів

4.3. Опис та реалізація інтерфейсу користувача і функціональних можливостей мобільного додатку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Перелік графічного матеріалу: 1 аркуш блок-схем, 20 ілюстрацій.
Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали, посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Постановка задачі | Кошова О.П. | | |
| Інформаційний огляд | Кошова О.П. | | |
| Теоретична частина | Кошова О.П. | | |
| Практична частина | Кошова О.П. | | |

Календарний графік виконання кваліфікаційної роботи

| Зміст роботи | Термін виконання | Фактичне виконання |
|---|---|---|
| Вступ | | |
| Вивчення методичних рекомендацій та стандартів та звіт керівнику | | |
| Постановка задачі | | |
| Інформаційний огляд джерел бібліотек та інтернету | | |
| Теоретична частина | | |
| Практична частина | | |
| Закінчення оформлення | | |
| Доповідь студента на кафедрі | | |
| Доробка (за необхідністю), рецензування | | |

Дата видачі завдання «__» _____ 2024 р.

Здобувач вищої освіти _____ Оборожний Артем Віталійович
(підпис)

Науковий керівник _____ к.пед.н., доц. Кошова О.П.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи
Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2025 р.
Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2024 р.

Погоджено
Науковий керівник _____
к.пед.н., Оксана КОШОВА
«____» _____ 2024 р.

План

дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Розробка мобільного додатку для управління та дистрибуції новин на основі мікросервісної архітектури»
Прізвище, ім'я, по батькові Оборожний Артем Віталійович

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
1.1. Мета та задачі дослідження
1.2. Визначення вимог до функціоналу мобільного додатку
1.3. Особливості використання децентралізованого шифрування
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Огляд існуючих платформ для дистрибуції новин
2.2. Порівняльний аналіз популярних рішень мобільних додатків для дистрибуції новин
2.3. Переваги та недоліки існуючих сервісів для дистрибуції новин
2.4. Використання децентралізованих систем у сучасних рішеннях безпеки та актуальні підходи до побудови API та архітектури мобільних додатків
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Опис обраних технологій та підходів для розробки мобільного додатку та API
3.2. Методологія розробки програмного забезпечення та побудова блок-схеми системи
3.3. Опис чистої архітектури та патерну MVVM
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Розробка алгоритму децентралізованого шифрування для забезпечення безпеки авторизації
4.2. Реалізація взаємодії класів мобільного додатку з використанням патерну MVVM та діаграма класів
4.3. Опис та реалізація інтерфейсу користувача і функціональних можливостей мобільного додатку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Здобувач вищої освіти _____ А. В. Оборожний

(підпис)

«____» _____ 2024 р.

# РЕФЕРАТ

Записка: 102 сторінки, 20 рисунків, 2 додатки, 20 літературних джерел.

Мета дипломної роботи є розробка програмного забезпечення мобільного додатку для управління та дистрибуції новин на основі мікросервісної архітектури.

Об'єкт розробки – мобільний додаток для управління та дистрибуції новин.

Методи дослідження – у процесі виконання роботи були застосовані сучасні методи проєктування та розробки мобільних додатків з використанням чистої архітектури та патерну MVVM (Model-View-ViewModel). Для реалізації функціоналу управління та дистрибуції новин було розроблено мікросервісне API, яке забезпечує взаємодію з базою даних та підтримує операції отримання, публікації та видалення новин.

Безпека авторизації користувачів та захист доступу до адміністративних функцій забезпечуються за допомогою децентралізованого шифрування та токенової автентифікації. Ключ авторизації перевіряється на термін дії перед виконанням кожного запиту, що підвищує рівень безпеки системи. Для поновлення ключа користувач повинен пройти повторну авторизацію, що запобігає несанкціонованому доступу.

У рамках дослідження було виконано наступні кроки:

Визначено ключові вимоги до функціоналу мобільного додатку для управління та дистрибуції новин, зокрема вимоги до безпеки, продуктивності та зручності використання.

Проведено аналіз існуючих платформ та мобільних додатків для дистрибуції новин, визначено їхні переваги та недоліки, що дозволило обґрунтувати вибір технологій та інструментів для розробки.

Описано та обґрунтовано вибір чистої архітектури та патерну MVVM для побудови мобільного додатку, що забезпечує розділення логіки додатку та його представлення, спрощує підтримку та розширення функціоналу.

Розроблено алгоритми та методологію створення додатку, зокрема побудовано блок-схеми системи, що відображають логіку роботи додатку та взаємодію його компонентів.

Створено API для взаємодії з базою даних, яке реалізує необхідні функції для отримання, публікації та видалення новин, а також обробку авторизації користувачів з використанням децентралізованого шифрування.

Інтегровано стороннє API для завантаження зображень, що дозволяє користувачам додавати фотографії до новин. Отримані посилання на зображення зберігаються в базі даних та відображаються в додатку.

Розроблено інтерфейс користувача та адміністратора, який забезпечує зручну навігацію по новинній стрічці з пагінаційною підгрузкою по 5 новин за запит, а також можливість публікації та видалення новин для авторизованих користувачів.

Представлено діаграми класів, use-case та блок-схеми, що ілюструють архітектуру додатку, взаємодію його компонентів та особливості програмного коду.

Проведено тестування мобільного додатку та API на відповідність вимогам, функціональність, безпеку та продуктивність, що підтвердило працездатність та надійність розробленого рішення.

Інтегровано децентралізоване шифрування в процес авторизації, що забезпечило високий рівень безпеки при взаємодії користувачів з системою та виконанні операцій через API.

ЗМІСТ

# ВСТУП

У сучасному світі інформаційних технологій та глобалізації оперативний доступ до актуальних новин є критично важливим для суспільства. Зростаючий обсяг інформації та необхідність її швидкої обробки вимагають ефективних засобів для управління та дистрибуції новин. Мобільні додатки стають одним із найпоширеніших інструментів для отримання інформації, оскільки вони забезпечують зручність, швидкість та персоналізацію контенту для користувачів.

Проте розробка мобільних додатків для новинних сервісів пов'язана з низкою викликів, серед яких забезпечення безпеки даних, ефективне управління контентом та інтеграція з різноманітними сервісами для збагачення функціоналу. Особливо актуальним є питання захисту інформації та безпечної авторизації користувачів, що вимагає застосування сучасних методів шифрування та автентифікації.

**Метою** даної роботи є розробка є розробка програмного забезпечення мобільного додатку для управління та дистрибуції новин на основі мікросервісної архітектури, який забезпечить зручний доступ до новинного контенту та можливість керування ним через інтуїтивно зрозумілий інтерфейс. Додаток повинен реалізовувати функції перегляду новинної стрічки з пагінацією, авторизації користувачів з використанням децентралізованого шифрування, а також можливість публікації та видалення новин через адміністративну панель.

**Об'єктом** дослідження є мобільний додаток для управління та дистрибуції новин, а **предметом** дослідження – процес розробки програмного забезпечення мобільного додатку з використанням мікросервісної архітектури та децентралізованого шифрування для забезпечення безпеки авторизації.

Для досягнення поставленої мети необхідно вирішити ряд завдань, зокрема:

Провести аналіз існуючих платформ та мобільних додатків для дистрибуції новин, визначити їхні переваги та недоліки.

Визначити ключові вимоги до функціоналу мобільного додатку, з особливим акцентом на безпеку та зручність використання.

Розробити архітектуру додатку з використанням чистої архітектури та патерну MVVM для забезпечення модульності та масштабованості.

Реалізувати API для взаємодії з базою даних, яке підтримує операції отримання, публікації та видалення новин.

Забезпечити безпечну авторизацію користувачів за допомогою децентралізованого шифрування та токенової автентифікації.

Інтегрувати стороннє API для завантаження зображень, що доповнить функціонал додатку та покращить користувацький досвід.

Провести тестування додатку на відповідність вимогам, функціональність та безпеку.

Методи дослідження включають сучасні підходи до розробки мобільних додатків, зокрема використання мікросервісної архітектури, патерну MVVM та децентралізованого шифрування. Для розробки програмного забезпечення використовувалися інструменти та середовища розробки, що відповідають сучасним стандартам індустрії.

Пояснювальна записка включає чотири основних розділи: постановка задачі, огляд сучасного стану проблеми, теоретична та практична частини. Структура роботи спрямована на логічне представлення матеріалу та всебічне розкриття теми дослідження. У ході роботи будуть розглянуті принципи побудови мобільних додатків для новинних сервісів, аналіз сучасних технологій та інструментів, що використовуються для розробки таких додатків, а також особливості забезпечення безпеки та ефективності системи.

За результатами дослідження планується отримати функціональний мобільний додаток, який може бути використаний як основа для подальшого розвитку та впровадження в реальні проекти з управління та дистрибуції новин.

# РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

## 1.1. Мета та задачі дослідження

Мета дослідження полягає в розробці мобільного додатку для управління та дистрибуції новин на основі мікросервісної архітектури, який забезпечить зручний, безпечний та ефективний доступ користувачів до актуального новинного контенту. Додаток повинен надати можливість перегляду новинної стрічки з пагінаційним підвантаженням, а також забезпечити авторизованим користувачам (адміністраторам) функції публікації, редагування та видалення новин через інтегровану адміністративну панель. Особлива увага приділяється впровадженню децентралізованого шифрування для забезпечення високого рівня безпеки при авторизації та взаємодії з API.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Аналіз сучасних рішень та технологій:

- Провести детальний огляд існуючих платформ та мобільних додатків для дистрибуції новин, визначити їхні переваги та недоліки.
- Проаналізувати сучасні підходи до побудови API та архітектури мобільних додатків, зокрема використання мікросервісної архітектури та патерну MVVM.
- Дослідити актуальні методи забезпечення безпеки в мобільних додатках, з особливим акцентом на децентралізоване шифрування та токенову автентифікацію.

2. Визначення вимог до мобільного додатку:

- Сформулювати функціональні вимоги до додатку, включаючи основні функції (перегляд новинної стрічки, перегляд деталей новини) та адміністративні функції (авторизація, публікація, редагування та видалення новин).

- Визначити нефункціональні вимоги, такі як зручність використання, продуктивність, безпека, масштабованість та сумісність з різними мобільними платформами.
- Розробити вимоги до безпеки додатку, враховуючи необхідність захисту персональних даних та контенту.

3. Проектування архітектури системи:

- Розробити архітектуру мобільного додатку з використанням чистої архітектури та патерну MVVM для забезпечення модульності та легкості підтримки.
- Спроектувати мікросервісне API для взаємодії з базою даних, що забезпечить необхідний функціонал для роботи додатку та відповідатиме вимогам безпеки.
- Розробити схеми бази даних та визначити структуру даних для зберігання новин та інформації про користувачів.

4. Розробка алгоритмів та механізмів безпеки:

- Розробити алгоритм децентралізованого шифрування для забезпечення безпечної авторизації користувачів та захисту даних при передачі через мережу.
- Реалізувати механізм перевірки терміну дії ключа авторизації перед виконанням запитів до API та механізм його поновлення шляхом повторної авторизації.
- Забезпечити захист від потенційних загроз безпеці, таких як несанкціонований доступ або перехоплення даних.

5. Реалізація мобільного додатку та API:

- Розробити інтерфейс користувача, який забезпечить зручний перегляд новинної стрічки з пагінацією по 5 новин за запит та можливість перегляду деталей новини.

- Реалізувати адміністративну панель для авторизованих користувачів, яка надасть можливість публікувати нові новини, завантажувати зображення, редагувати та видаляти існуючі новини.
- Інтегрувати стороннє API для завантаження зображень, яке повертає посилання на зображення для подальшого використання в додатку та зберігання в базі даних.

6. Тестування та оптимізація системи:

- Провести комплексне тестування мобільного додатку та API на функціональність, продуктивність та безпеку.
- Оптимізувати код та архітектуру додатку для забезпечення стабільної роботи та високої швидкодії.

7. Документування та підготовка матеріалів:

- Підготувати детальну технічну документацію, що описує архітектуру системи, алгоритми, API та інструкції з використання додатку.
- Розробити діаграми класів, блок-схеми та use-case діаграми, що ілюструють структуру та логіку роботи додатку.
- Створити інструкції для користувачів та адміністраторів щодо встановлення, налаштування та експлуатації додатку.

8. Аналіз результатів та перспективи розвитку:

- Проаналізувати результати розробки та визначити ступінь відповідності отриманого додатку поставленим вимогам та меті дослідження.
- Оцінити ефективність використання децентралізованого шифрування для авторизації та безпеки в контексті мобільних додатків.
- Визначити можливі напрямки подальшого розвитку та вдосконалення додатку, такі як впровадження додаткових функцій, розширення підтримки платформ або інтеграція з іншими сервісами.

9. Поширення результатів дослідження:

- Підготувати матеріали для публікації результатів дослідження на міжнародних наукових конференціях або в фахових виданнях.

- Представити розроблений додаток та отримані результати зацікавленим сторонам, включаючи потенційних користувачів та інвесторів.
- Виконання цих задач забезпечить створення сучасного, безпечного та зручного мобільного додатку для управління та дистрибуції новин, який відповідатиме вимогам користувачів та тенденціям розвитку інформаційних технологій. Розробка такого додатку сприятиме покращенню доступу до інформації, підвищенню ефективності управління новинним контентом та зміцненню безпеки даних у мобільних додатках.

## 1.2. Визначення вимог до функціоналу мобільного додатку

Для успішної реалізації мобільного додатку для управління та дистрибуції новин необхідно чітко визначити вимоги до його функціоналу. Ці вимоги враховують особливості завантаження новин, процес авторизації, роботу з адміністративною панеллю та інтеграцію зі сторонніми сервісами. Вимоги поділяються на функціональні та нефункціональні, які забезпечують ефективність, безпеку та зручність використання додатку.

Функціональні вимоги до розробки додатку передбачають кілька ключових аспектів. Основною функцією є завантаження та відображення новинної стрічки. Додаток повинен завантажувати новини при запуску, використовуючи патерн MVVM і ViewModel, що забезпечує незалежність даних від життєвого циклу фрагмента. Це дозволяє зберігати завантажені новини при повторному відкритті основного фрагмента. Реалізація завантаження новин здійснюється у двох режимах: повне завантаження під час першого запуску або оновлення, а також доповнене завантаження при прокручуванні стрічки, що дозволяє оптимізувати використання мережевих ресурсів.

Користувачі повинні мати можливість переглядати детальну інформацію про новину, включаючи заголовок, опис та зображення, з можливістю зручного переходу між списком новин і детальним переглядом. Окрім цього, додаток передбачає функцію зміни типу користувача. У налаштуваннях можна змінити статус із звичайного користувача на адміністратора за умови введення унікального UUID ключа. Процес авторизації включає перевірку ключа у базі даних, його прив’язку до Android ID пристрою користувача у форматі незворотного шифрування, а також отримання токена від сервера для зберігання в локальному сховищі. Ключ діє протягом 24 годин, після чого авторизацію необхідно поновлювати, що підвищує безпеку системи.

Адміністратор після авторизації отримує доступ до розширеного функціоналу, включаючи публікацію новин із можливістю додавання заголовків, опису та зображень, а також видалення існуючих новин із бази даних. Для завантаження зображень інтегрується стороннє API, яке забезпечує отримання посилань на зображення для збереження в базі даних.

Додаток взаємодіє з мікросервісним API для виконання запитів, зокрема отримання, публікації та видалення новин. API підтримує авторизацію через токени та забезпечує ефективність і мінімальні затримки. Перевірка авторизації перед адміністративними діями виконується локально, що зменшує навантаження на сервер. Локальне сховище використовується для збереження токенів та налаштувань, забезпечуючи швидкий доступ до даних і можливість автономної роботи.

Таким чином, реалізація функціональних вимог спрямована на забезпечення ефективності, зручності використання та високого рівня безпеки додатку.

Нефункціональні вимоги:

1. Безпека даних. Використання незворотного шифрування для зберігання ключів та Android ID забезпечує високий рівень безпеки.

Конфіденційні дані повинні бути захищені від несанкціонованого доступу як на пристрої користувача, так і на сервері.

2. Продуктивність та ефективність. Додаток повинен працювати швидко та стабільно, забезпечуючи миттєву реакцію на дії користувача. Використання патерну MVVM та оптимізація запитів до API сприяють високій продуктивності.

3. Зручність використання та інтуїтивний інтерфейс. Інтерфейс додатку повинен бути зрозумілим та легким у використанні для користувачів з різним рівнем технічної підготовки. Дизайн повинен бути сучасним та адаптивним до різних розмірів екранів.

4. Масштабованість та підтримка. Архітектура додатку повинна дозволяти легке додавання нового функціоналу та підтримку зростаючої кількості користувачів без погіршення якості роботи.

5. Сумісність. Додаток повинен коректно працювати на різних версіях операційної системи Android та на різних пристроях, забезпечуючи однаковий користувацький досвід.

6. Надійність та стійкість до збоїв. Додаток повинен бути стабільним, з можливістю відновлення після збоїв без втрати даних або порушення роботи.

7. Конфіденційність персональних даних. Особиста інформація користувачів та адміністраторів не повинна бути доступна третім особам. Додаток повинен відповідати вимогам законодавства щодо захисту персональних даних.

8. Оптимізація мережевих ресурсів. Мінімізація кількості запитів до сервера за рахунок децентралізованої перевірки токенів та ефективного кешування даних.

9. Відповідність сучасним стандартам розробки. Додаток повинен бути розроблений з урахуванням кращих практик програмування, забезпечуючи чистий та підтримуваний код.

Таким чином, визначені вимоги забезпечать створення мобільного додатку, який задовольнить потреби користувачів у швидкому та зручному доступі до новин, а також надасть адміністраторам ефективні інструменти для управління контентом. Дотримання цих вимог гарантує високу якість додатку, його безпеку та надійність у використанні [13].

## 1.3. Особливості використання децентралізованого шифрування

Забезпечення безпеки в мобільних додатках є одним із ключових аспектів сучасної розробки програмного забезпечення, особливо коли мова йде про авторизацію користувачів та захист конфіденційних даних. У розробленому мобільному додатку для управління та дистрибуції новин впроваджено децентралізоване шифрування як основний механізм забезпечення безпеки авторизації та доступу до адміністративних функцій.

Децентралізоване шифрування в контексті даного додатку означає, що перевірка дійсності ключа авторизації здійснюється без постійного звернення до центрального сервера або бази даних. Це реалізується шляхом зберігання зашифрованого токена авторизації у локальному сховищі пристрою користувача та використання незворотного шифрування для прив’язки ключа до конкретного пристрою.

Процес авторизації адміністраторів починається з введення унікального UUID ключа, який генерується та зберігається в базі даних. Після введення ключа додаток здійснює одноразовий запит до сервера для перевірки його наявності та активності. Якщо ключ не був активований раніше, сервер асоціює його з унікальним Android ID пристрою користувача, використовуючи незворотне шифрування. Це забезпечує, що ключ не може бути використаний на іншому пристрої, навіть якщо він стане відомим третім особам.

Сервер повертає зашифрований токен авторизації, який зберігається у локальному сховищі додатку. Кожен токен має обмежений термін дії — 24 години. Перед виконанням адміністративних операцій, таких як публікація або видалення новин, додаток здійснює локальну перевірку дійсності токена, включаючи перевірку терміну його дії. Якщо токен прострочений, користувач повинен повторно пройти процес авторизації, використовуючи той самий UUID ключ.

Використання децентралізованого шифрування має кілька важливих переваг:

1. Зменшення навантаження на сервер та базу даних. Оскільки перевірка токена здійснюється локально, немає потреби у постійних запитах до сервера. Це оптимізує роботу додатку та знижує вимоги до серверних ресурсів.

2. Підвищення рівня безпеки. Прив’язка ключа до конкретного пристрою та використання незворотного шифрування унеможливлює несанкціоноване використання ключів на інших пристроях. Навіть якщо ключ буде перехоплено, його не можна буде застосувати без відповідного Android ID.

3 Гнучкість у керуванні доступом. Адміністратори можуть легко відкликати доступ, видаляючи ключ з бази даних. Це негайно блокує можливість авторизації для відповідного ключа, забезпечуючи контроль над доступом до адміністративних функцій.

4 Захист від атак на мережу. Оскільки токен зберігається локально і не передається через мережу під час кожної операції, знижується ризик його перехоплення або компрометації через мережеві атаки.

5 У процесі публікації новин децентралізоване шифрування також відіграє важливу роль. Перед тим як надіслати запит на власний мікросервісний API для збереження новини, додаток перевіряє дійсність токена авторизації локально. Це гарантує, що лише авторизовані користувачі можуть здійснювати адміністративні дії.

6 Крім того, під час завантаження зображень для новин використовується сторонній API, який повертає посилання на зображення. Це рішення дозволяє зменшити обсяг даних, що передаються через власний сервер, та уникнути необхідності зберігання великих файлів. Посилання на зображення додається до новини та зберігається у базі даних, що забезпечує ефективне управління медіа-контентом.

Важливо зазначити, що децентралізоване шифрування доповнює загальну архітектуру додатку, побудовану на основі патерну MVVM та мікросервісної архітектури. Воно сприяє підвищенню масштабованості та надійності системи, оскільки зменшує залежність від центрального сервера для перевірки авторизації.

Таким чином, використання децентралізованого шифрування в мобільному додатку для управління та дистрибуції новин дозволяє забезпечити високий рівень безпеки, ефективність роботи та зручність для користувачів. Це рішення поєднує переваги сучасних криптографічних методів з гнучкістю та масштабованістю мікросервісної архітектури, відповідаючи вимогам сучасних стандартів розробки мобільних додатків.

# РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ

## 2.1. Огляд існуючих платформ для дистрибуції новин

На сьогодні на ринку існує велика кількість платформ та інструментів для дистрибуції новин, які пропонують різноманітний функціонал, форматування та можливості персоналізації. Більшість таких сервісів орієнтовані на користувачів, які прагнуть отримувати актуальні новини з мінімальними зусиллями та без необхідності спеціальних знань. До популярних платформ для дистрибуції новин відносяться онлайн-агрегатори, такі як Google News, Apple News, Flipboard, а також соціальні мережі та месенджери, які все більше використовуються для поширення новинного контенту.

Основні функції, які зазвичай пропонують подібні платформи:

- Персоналізація контенту: більшість платформ надають користувачам можливість налаштовувати теми та джерела, які їх цікавлять. Це дозволяє отримувати новини, що відповідають індивідуальним інтересам та вподобанням.
- Інтеграція з соціальними мережами: багато сервісів дозволяють користувачам ділитися новинами в соціальних мережах, слідкувати за активністю друзів та отримувати рекомендації на основі соціальних зв’язків.
- Мультимедійний контент: підтримка різних форматів, таких як текст, зображення, відео та аудіо, робить споживання новин більш зручним та цікавим.
- Пуш-повідомлення: оперативне інформування про важливі події та новини за допомогою сповіщень на мобільних пристроях.

Огляд кожної з цих платформ показує, що їх основна перевага полягає в можливості швидкого та зручного доступу до актуальної інформації без необхідності відвідувати безліч окремих веб-сайтів. Проте більшість існуючих рішень не мають достатнього рівня інтерактивності та персоналізації, що ускладнює процес індивідуального налаштування стрічки новин під конкретні

потреби користувача.

Іншим типом рішень для дистрибуції новин є месенджери та соціальні мережі, які наразі набувають все більшої популярності. Вони надають більше можливостей для інтерактивної взаємодії з контентом, допомагаючи користувачам обговорювати новини, ділитися думками та отримувати інформацію в зручному для них форматі через діалог. На відміну від класичних платформ, вони орієнтовані на безпосереднє спілкування та можуть адаптувати контент на основі поведінки користувача.

Цей огляд демонструє потребу в удосконаленні існуючих рішень, особливо в напрямку інтерактивності, безпеки та персоналізації, що може бути досягнуто за допомогою мобільного додатку для управління та дистрибуції новин, який автоматизує процес збору, обробки та організації новинного контенту для створення оптимізованого інформаційного потоку.

Сучасний ринок пропонує безліч інструментів для дистрибуції новин, які спрощують процес отримання актуальної інформації. Нижче наведено огляд деяких популярних сервісів:

1) Google News - це онлайн-платформа, що надає користувачам можливість отримувати новини з різних джерел за допомогою зручного агрегатора. Сервіс пропонує персоналізовану стрічку новин, яка формується на основі інтересів користувача, його географічного розташування та історії пошукових запитів. Google News також включає функції розподілу новин за категоріями, такими як політика, спорт, технології, що дозволяє легко знаходити інформацію за тематикою.

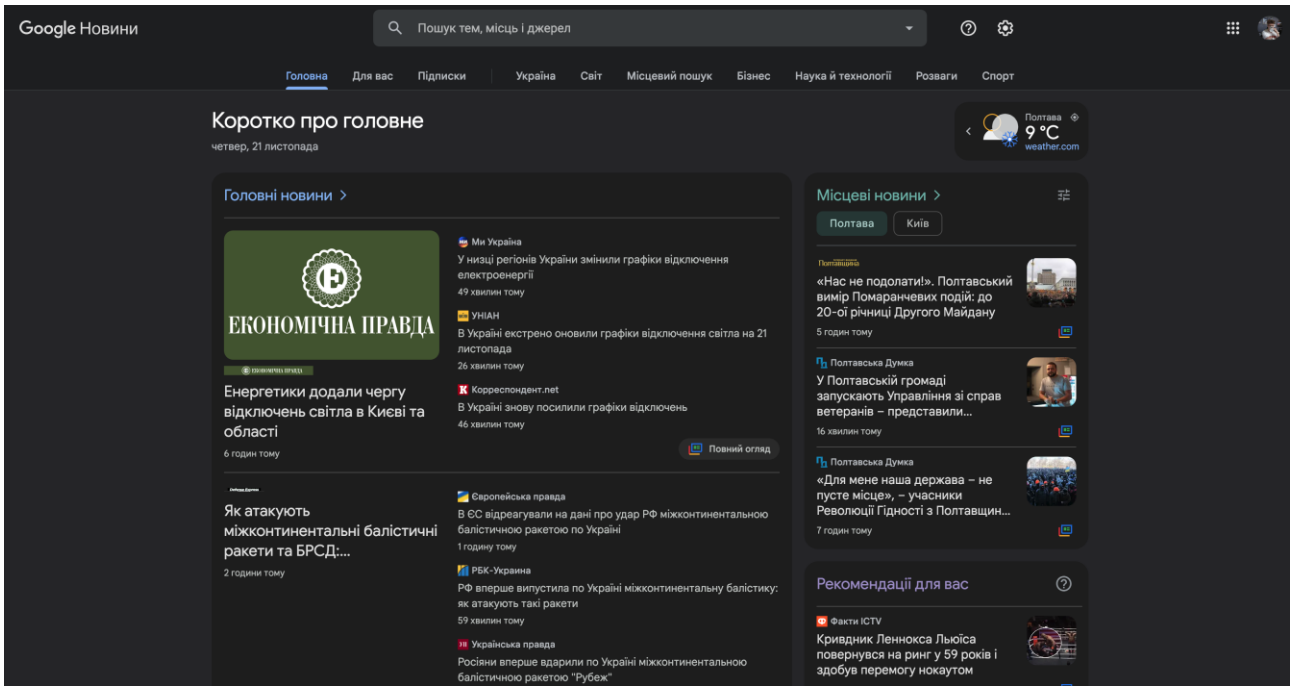


Рис. 2.1 – Платформа Google News

Крім того, платформа використовує технології штучного інтелекту для автоматизованого підбору контенту, що полегшує процес отримання релевантних новин для користувачів без додаткових зусиль. Це особливо корисно для тих, хто хоче бути в курсі останніх подій у певній сфері, оскільки сервіс підтримує багатомовність та локалізацію.

Google News також пропонує функції адаптації контенту під уподобання користувача, що підвищує шанси отримання цікавих та важливих новин. Платформа має інтуїтивно зрозумілий інтерфейс, що робить процес споживання новин простим та зручним для користувачів. Відгуки користувачів свідчать про високу якість сервісу та зручність його використання.

2) Apple News є онлайн-платформою, яка надає користувачам пристроїв Apple можливість отримувати персоналізовані новини з різних джерел. Основною перевагою сервісу є його інтеграція з екосистемою Apple, що забезпечує зручний доступ до новин на iPhone, iPad та Mac.

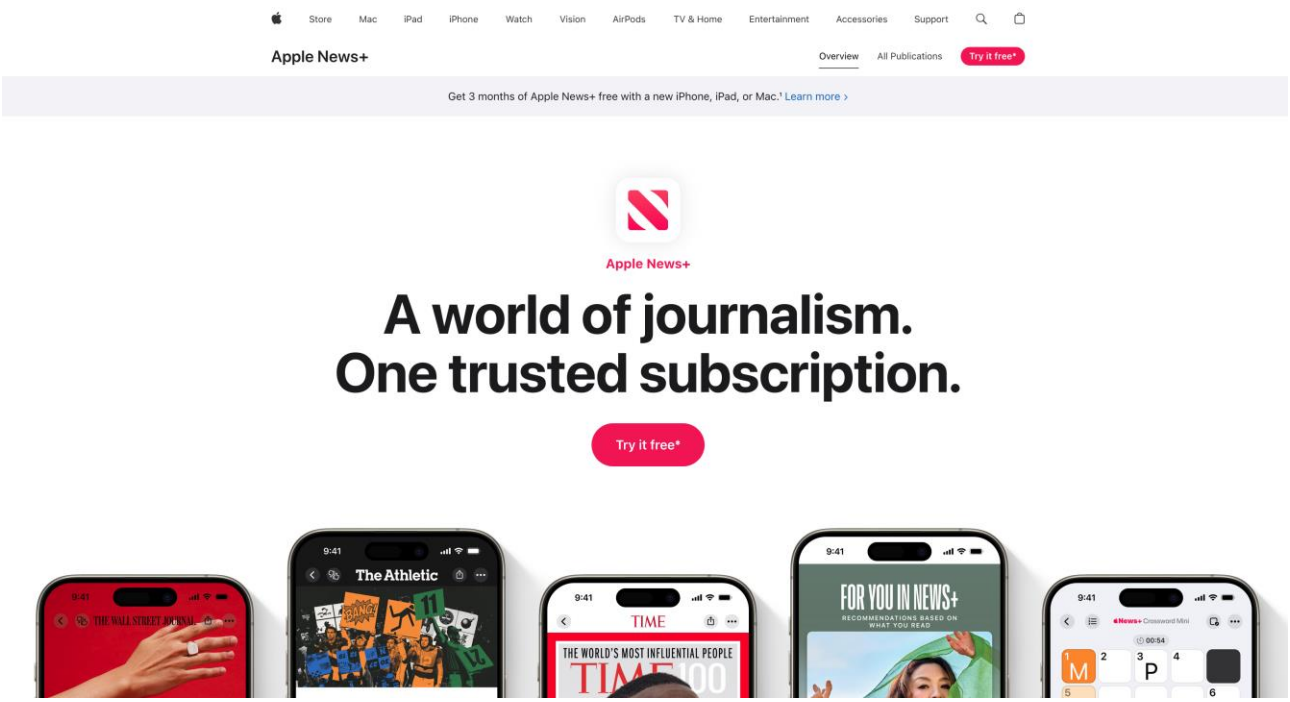


Рис. 2.2 – Платформа Apple News

Apple News пропонує персоналізовану стрічку новин, яка формується на основі вподобань користувача та його взаємодії з контентом. Сервіс також надає можливість підписки на преміум-контент через Apple News+, що відкриває доступ до журналів та ексклюзивних статей.

Окрім того, платформа підтримує функцію "Today View", яка відображає найважливіші новини дня, що дозволяє користувачам швидко ознайомитися з головними подіями. Проте, незважаючи на свої переваги, Apple News обмежений для користувачів пристроїв Apple та не доступний на інших платформах.

3) Flipboard - це онлайн-платформа та мобільний додаток, який перетворює новини з різних джерел у персоналізований цифровий журнал. Користувачі можуть обирати теми, які їх цікавлять, і Flipboard автоматично збирає та форматує відповідний контент у вигляді красивих та зручних для читання статей.

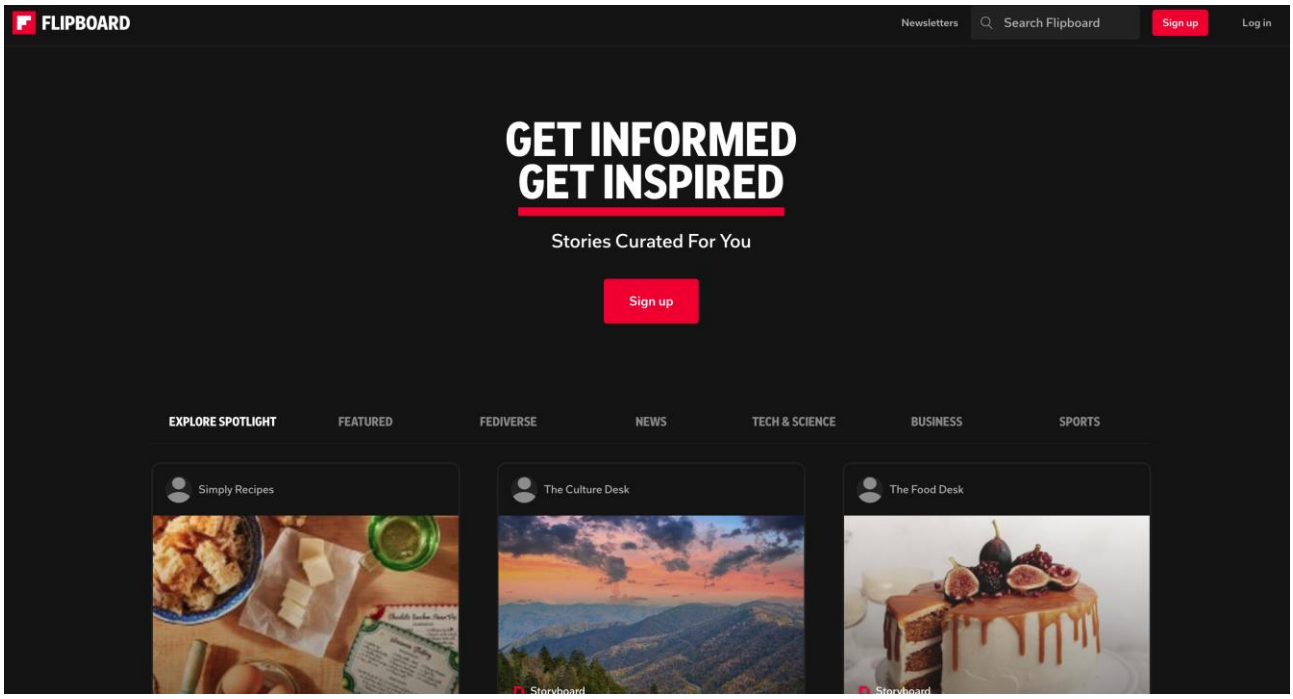


Рис. 2.3 – Платформа Flipdoard

Однією з основних переваг Flipboard є його візуальна привабливість та можливість інтеграції з соціальними мережами. Користувачі можуть додавати свої акаунти Facebook, Twitter та інших мереж, що дозволяє отримувати новини та оновлення від друзів та підписок у єдиному місці.

Проте, незважаючи на свої переваги, Flipboard може бути перенасиченим інформацією для деяких користувачів, що ускладнює процес вибору дійсно важливих новин. Додатково, сервіс іноді може дублювати контент, якщо одні й ті самі новини публікуються в різних джерелах.

4) Microsoft News - надає користувачам доступ до новин з провідних світових та локальних медіа. Сервіс інтегрований з Windows та доступний на мобільних пристроях. Microsoft News пропонує персоналізовану стрічку новин, яка формується на основі інтересів користувача та його взаємодії з контентом.

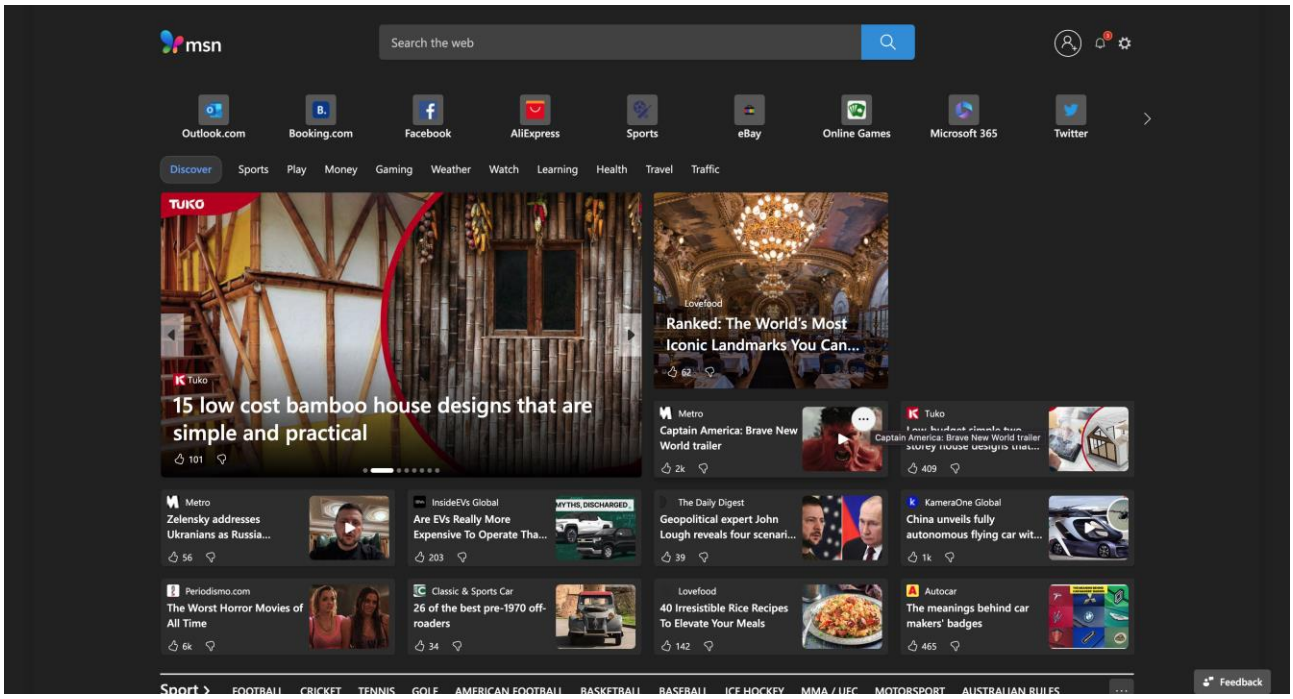


Рис. 2.4 – Платформа Microsoft News

Сервіс також підтримує темну тему оформлення та надає можливість отримувати сповіщення про важливі події. Проте, як і інші платформи, Microsoft News може включати рекламу та спонсорований контент, що може бути недоліком для деяких користувачів.

5) SmartNews є платформою, яка використовує штучний інтелект для аналізу мільйонів статей та виявлення найважливіших новин. Сервіс надає користувачам персоналізовану стрічку новин та підтримує офлайн-режим читання.

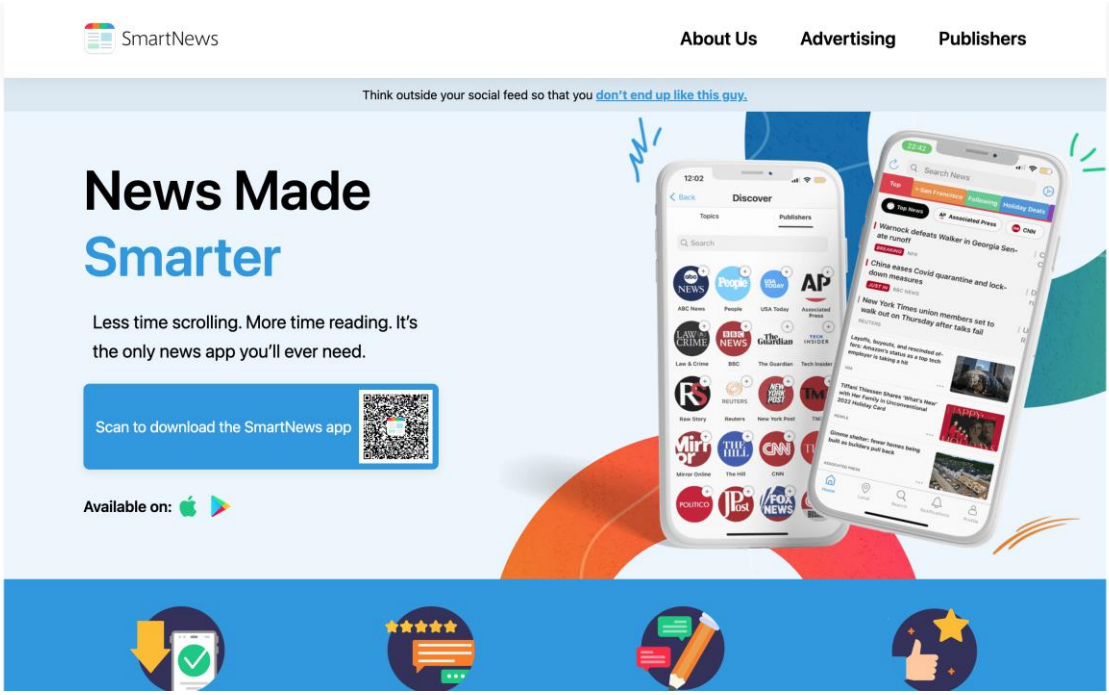


Рис. 2.5 – Платформа SmartNews

Однією з ключових особливостей SmartNews є його здатність виявляти тенденції та популярні теми, що дозволяє користувачам бути в курсі найактуальніших подій. Інтерфейс додатку простий та інтуїтивно зрозумілий, що робить його зручним для щоденного використання.

Проте SmartNews може мати обмежені можливості налаштування персоналізації порівняно з іншими сервісами, а також включати рекламу в безкоштовній версії.

6) Feedly - це RSS-агрегатор, який дозволяє користувачам створювати власну стрічку новин з улюблених джерел. Сервіс пропонує можливість організації підписок за категоріями та інтегрується з різними додатками та сервісами, такими як Evernote, Pocket та інші.

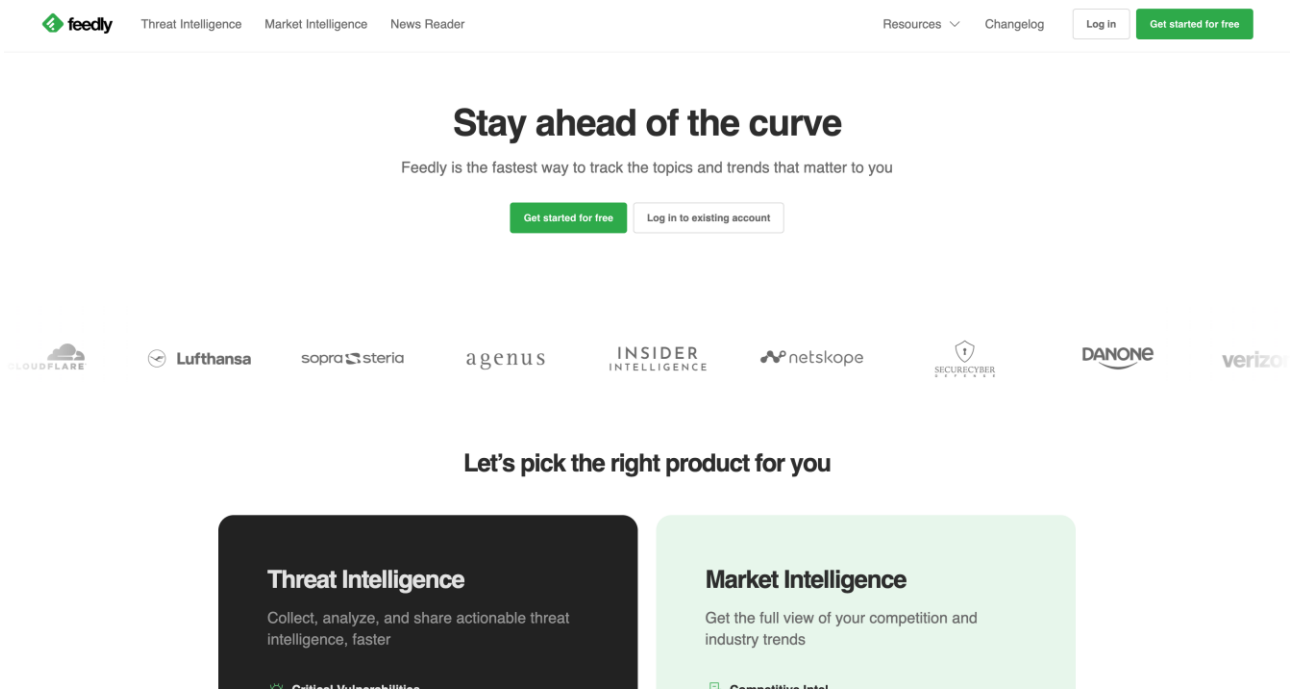


Рис. 2.6 – Платформа Feedly

Feedly є популярним серед професіоналів та ентузіастів, які потребують доступу до спеціалізованого контенту. Платформа надає повний контроль над джерелами та дозволяє уникнути надлишку непотрібної інформації.

Проте, для новачків Feedly може здатися складним у налаштуванні та використанні, а деякі розширені функції доступні лише у платній версії.

7) Reddit - це платформа соціальних новин та обговорень, де користувачі можуть публікувати, коментувати та голосувати за контент. Reddit складається з численних спільнот, відомих як субреддіти, які охоплюють практично будь-яку тему.

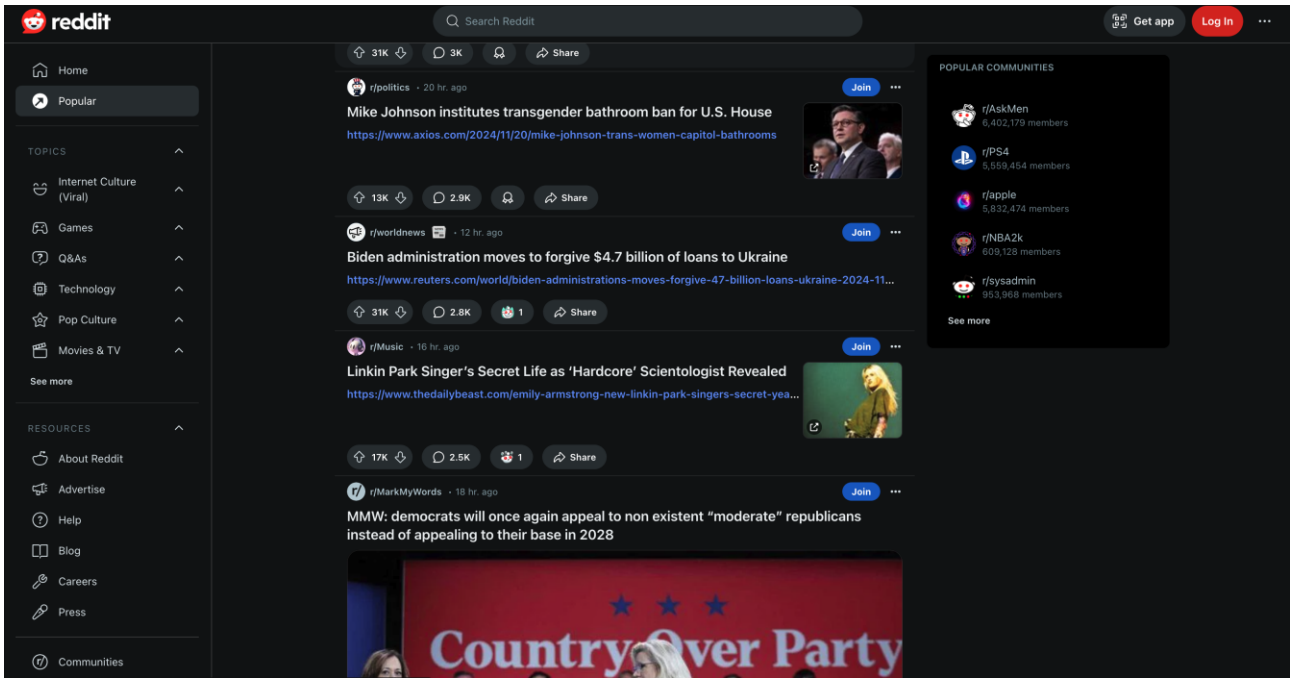


Рис. 2.7 – Платформа Reddit

Однією з переваг Reddit є активна спільнота та можливість отримувати інформацію з перших рук. Користувачі можуть задавати питання експертам, брати участь у обговореннях та отримувати різноманітні точки зору.

Проте, відсутність професійної модерації та можливість поширення недостовірної інформації є суттєвими недоліками платформи. Крім того, інтерфейс Reddit може бути незручним для нових користувачів.

8) Medium є платформою для публікації статей та блогів, де автори можуть ділитися своїми думками, дослідженнями та історіями. Medium пропонує користувачам персоналізовану стрічку контенту на основі їхніх інтересів та вподобань.

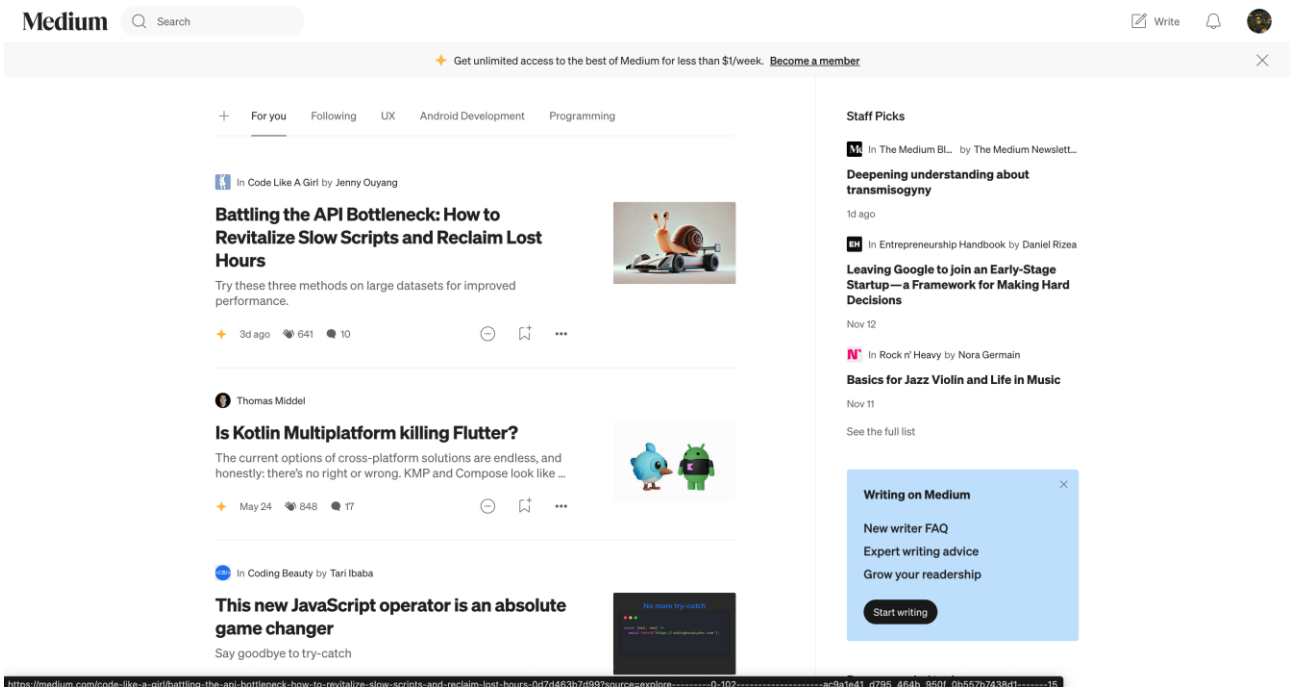


Рис. 2.8 – Платформа Medium

Платформа відома високою якістю контенту та можливістю взаємодії з авторами через коментарі та аплодисменти. Medium також надає можливість підписки на платний контент через програму Membership.

Проте, Medium не є традиційною новинною платформою і менш підходить для отримання оперативних новин. Більшість публікацій є аналітичними або особистими історіями, що може не задовольняти потреби користувачів, які шукають швидких оновлень.

9) Inshorts - це мобільний додаток, який надає короткі новини, стискаючи основну інформацію до 60 слів. Сервіс орієнтований на користувачів, які бажають швидко ознайомитися з головними подіями дня.

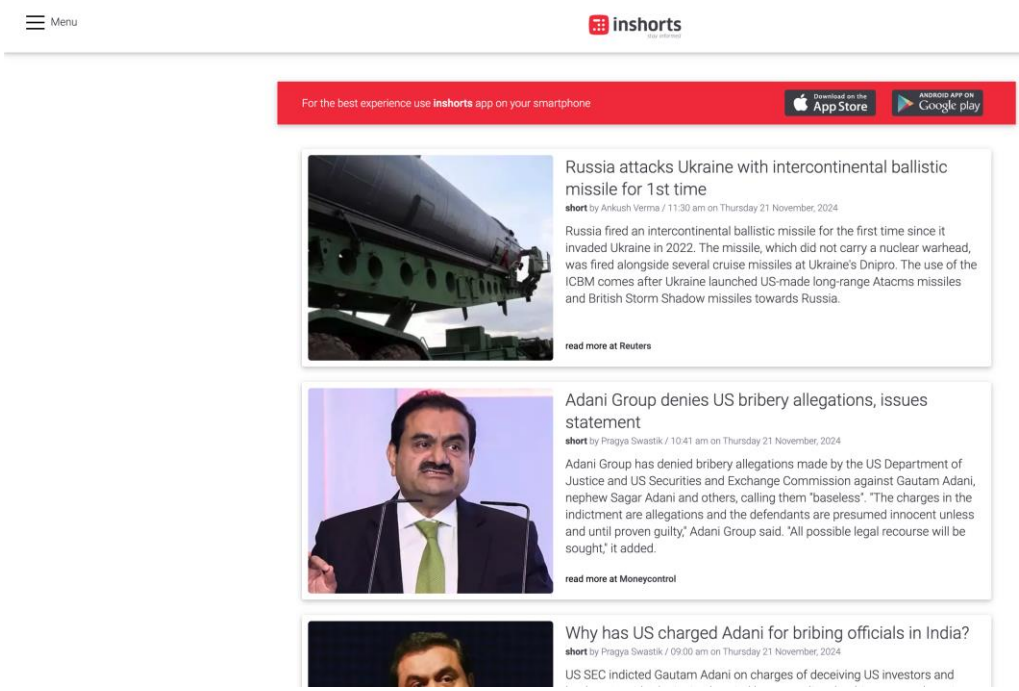


Рис. 2.9 – Платформа Inshorts

Inshorts підтримує різні категорії новин та надає можливість персоналізації стрічки. Проте, такий формат може не забезпечувати достатньої глибини інформації для користувачів, які бажають детального аналізу подій.

10) Telegram є одним із найпопулярніших месенджерів у світі, який набув значної популярності як платформа для дистрибуції новин. Завдяки своїй функціональності, безпеці та зручності використання, Telegram став важливим каналом для медіа, журналістів та організацій, які прагнуть оперативно донести інформацію до своєї аудиторії.

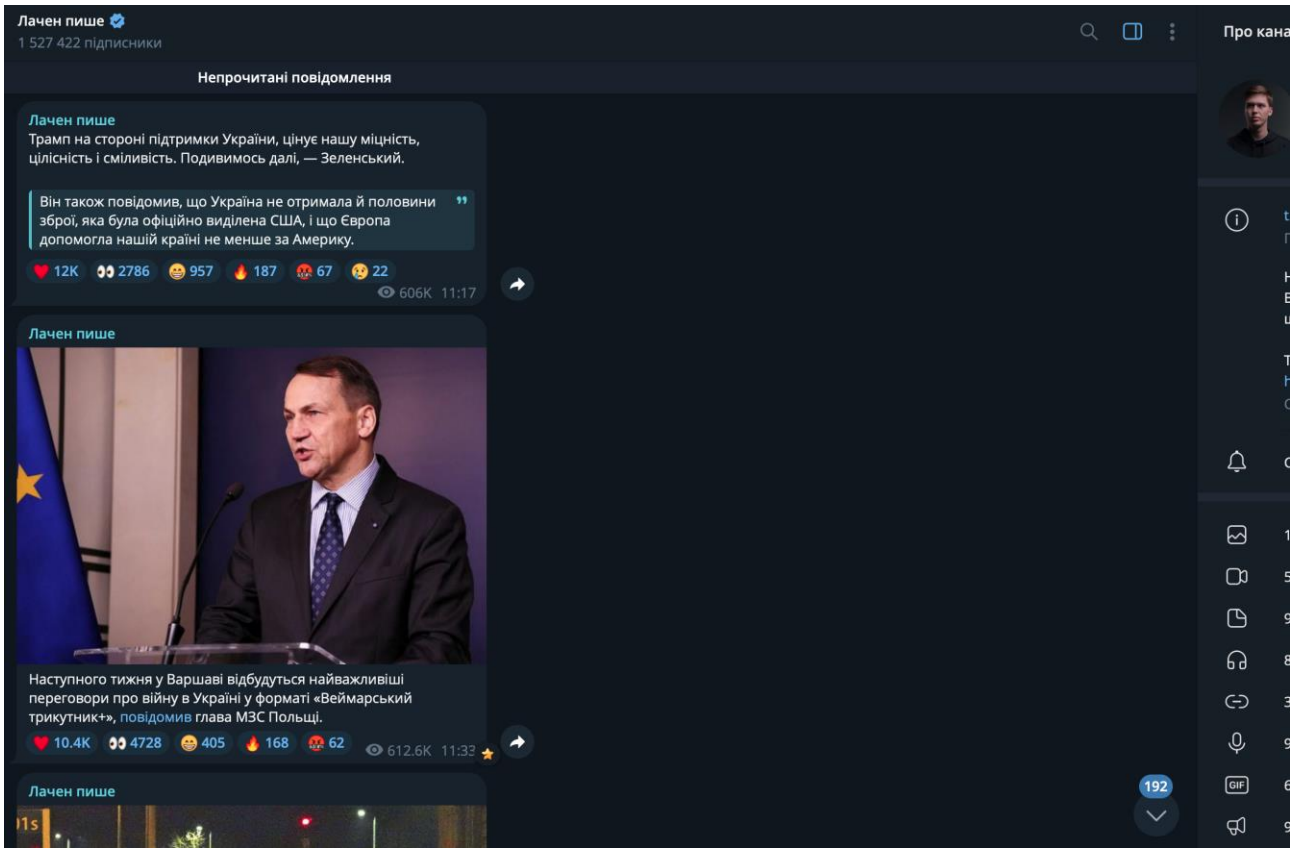


Рис. 2.10 – Платформа Telegram

Telegram надає користувачам можливість підписуватися на канали, які можуть бути як публічними, так і приватними. Канали дозволяють адміністраторам публікувати повідомлення, що автоматично доставляються всім підписникам.

Це робить Telegram ефективним інструментом для поширення новин, оголошень та оновлень. Багато відомих медіа-організацій, таких як BBC News, The New York Times, Українська Правда та інші, мають офіційні канали в Telegram, де вони публікують останні новини та аналітичні матеріали.

Однією з ключових переваг Telegram є висока швидкість передачі даних та миттєві сповіщення, що дозволяє користувачам отримувати новини в режимі реального часу. Крім того, платформа підтримує мультимедійний контент, включаючи зображення, відео, аудіо та документи, що робить подачу інформації більш насиченою та інтерактивною.

Telegram також підтримує ботів, які можуть автоматизувати процес

доставки новин та взаємодії з користувачами. Наприклад, новинні боти можуть надсилати персоналізовані повідомлення на основі інтересів користувача, відповідати на запити або навіть пропонувати інтерактивні опитування. Це відкриває широкі можливості для індивідуалізації контенту та підвищення залученості аудиторії.

Переваги використання Telegram для дистрибуції новин:

- Широке охоплення аудиторії: Telegram має мільйони користувачів по всьому світу, що дозволяє медіа-організаціям розширити свою аудиторію.
- Безпека та приватність: Платформа забезпечує високий рівень шифрування та конфіденційності, що важливо для користувачів та організацій.
- Відсутність реклами: На відміну від деяких соціальних мереж, Telegram не містить сторонньої реклами, що покращує користувацький досвід.

Однак Telegram має і певні недоліки в контексті дистрибуції новин:

- Відсутність алгоритмічної персоналізації: Платформа не пропонує складних алгоритмів для підбору контенту на основі поведінки користувача, що може обмежувати релевантність отримуваних новин.
- Ризик поширення дезінформації: Відсутність суворої модерації контенту може призводити до поширення фейкових новин та недостовірної інформації.
- Обмежені аналітичні інструменти: Організації мають обмежений доступ до статистики та аналітики щодо взаємодії користувачів з контентом, що ускладнює оцінку ефективності.
- Залежність від зовнішніх серверів: Хоча Telegram декларує безпеку, сервери платформи знаходяться за межами України, що може викликати питання щодо контролю над даними.

Приклади успішного використання Telegram у дистрибуції новин:

- Канал "Українська Правда" регулярно публікує актуальні новини, аналітику та інтерв'ю, охоплюючи широку українську аудиторію.
- Канал "BBC News Україна" надає українською мовою міжнародні та

локальні новини, що дозволяє користувачам бути в курсі світових подій.

- Незалежні журналістські проекти використовують Telegram для поширення розслідувань та ексклюзивних матеріалів, оминаючи цензуру та обмеження інших платформ.

Крім офіційних каналів, Telegram також підтримує групові чати, де користувачі можуть обговорювати новини, ділитися думками та брати участь у дискусіях. Це сприяє формуванню спільнот навколо певних тем або медіа-джерел.

У контексті розвитку технологій, Telegram продовжує впроваджувати нові функції, які можуть бути корисними для дистрибуції новин. Наприклад, відкладені публікації, опитування та реакції на повідомлення розширюють можливості взаємодії з аудиторією.

Загалом, Telegram є потужним інструментом для дистрибуції новин завдяки своїй функціональності, безпеці та широкому охопленню аудиторії. Однак для максимально ефективного використання платформи важливо враховувати її обмеження та виклики, пов'язані з модерацією контенту та аналітикою. Це підкреслює необхідність у поєднанні Telegram з іншими інструментами та платформами, а також у розробці нових рішень, які б забезпечили більш інтерактивний, персоналізований та безпечний досвід для користувачів.

Підсумовуючи, існує велика кількість платформ та інструментів для дистрибуції новин, кожен з яких має свої переваги та недоліки. Більшість сервісів пропонують персоналізацію контенту, інтеграцію з соціальними мережами та мультимедійний контент. Проте, недоліками можуть бути перенасиченість інформацією, реклама, обмежені можливості персоналізації або платний доступ до преміум-контенту.

Це підкреслює необхідність розробки нових рішень, які б поєднували переваги існуючих платформ та мінімізували їхні недоліки. Мобільний додаток для управління та дистрибуції новин з використанням сучасних технологій,

таких як мікросервісна архітектура та децентралізоване шифрування, може стати таким рішенням. Він забезпечить користувачам ефективний, безпечний та зручний доступ до новинного контенту, а також надасть можливості для інтерактивної взаємодії та персоналізації.

## 2.2. Порівняльний аналіз популярних рішень мобільних додатків для дистрибуції новин

У попередньому розділі було детально розглянуто існуючі платформи для дистрибуції новин, їхні функціональні можливості та особливості. У цьому розділі проведемо порівняльний аналіз найбільш популярних мобільних додатків для дистрибуції новин, щоб виявити їхні переваги та недоліки, а також визначити можливості для подальшого вдосконалення. Основна мета аналізу полягає в оцінці додатків за ключовими критеріями, які є важливими для користувачів та розробників.

Основні критерії порівняння:

- Функціональність - набір доступних функцій, включаючи персоналізацію контенту, підтримку мультимедіа, можливості офлайн-доступу та інтеграцію з іншими сервісами.
- Зручність використання - інтуїтивність інтерфейсу, простота навігації та загальний користувацький досвід.
- Персоналізація - рівень налаштування контенту відповідно до інтересів користувача.
- Безпека та приватність - захист даних користувачів, використання шифрування та політика конфіденційності.
- Достовірність контенту - надійність джерел новин та наявність механізмів перевірки інформації.

- Монетизація - наявність реклами, платних підписок або інших моделей монетизації.
- Платформи та доступність - підтримка різних операційних систем та пристроїв.

Табл. 2.1

**Порівняльна таблиця мобільних додатків для дистрибуції новин**

| Критерії | Google News | Apple News | Flipboard | Microsoft News | SmartNews | Telegram | Reddit | Inshorts |
|---|---|---|---|---|---|---|---|---|
| Функціональність | Висока | Висока | Висока | Висока | Середня | Середня | Висока | Низька |
| Зручність використання | Висока | Висока | Висока | Висока | Висока | Висока | Середня | Висока |
| Персоналізація | Висока | Висока | Висока | Середня | Середня | Низька | Висока | Низька |
| Безпека та приватність | Висока | Висока | Висока | Висока | Висока | Середня | Середня | Висока |
| Достовірність контенту | Висока | Висока | Залежить від джерел | Висока | Середня | Низька | Залежить від спільноти | Середня |
| Монетизація | Безкоштовно | Безкоштовно/ Платно | Безкоштовно | Безкоштовно | Безкоштовно | Безкоштовно | Безкоштовно/ Платно | Безкоштовно |

Google News пропонує широкий спектр функціональних можливостей, включаючи персоналізовану стрічку новин, підтримку мультимедійного контенту та офлайн-доступ. Завдяки складним алгоритмам машинного навчання, сервіс забезпечує високу персоналізацію контенту. Інтерфейс додатку є інтуїтивно зрозумілим, що сприяє позитивному користувацькому досвіду. Безпека та приватність даних підтримуються на високому рівні, відповідно до політики Google. Достовірність контенту також є високою, оскільки сервіс співпрацює з перевіреними джерелами. Монетизація здійснюється через рекламу, але вона не є надмірною та не заважає користуванню.

Apple News схожий за функціональністю на Google News, але доступний лише на пристроях Apple. Сервіс відрізняється високою якістю контенту та безпекою даних користувачів. Преміум-підписка Apple News+ надає доступ до

ексклюзивних матеріалів, що може бути перевагою для користувачів, які шукають глибокий аналітичний контент. Персоналізація контенту реалізована на високому рівні, а інтерфейс додатку відповідає загальним стандартам дизайну Apple.

Flipboard вирізняється своїм візуально привабливим інтерфейсом та можливістю створення персоналізованих журналів. Функціональність додатку включає інтеграцію з соціальними мережами, що дозволяє користувачам додавати контент з різних платформ. Персоналізація контенту є високою, оскільки користувачі можуть самостійно обирати теми та джерела. Проте достовірність контенту залежить від обраних джерел, і може виникати ризик отримання недостовірної інформації.

Microsoft News надає користувачам доступ до новин з перевірених джерел, забезпечуючи високу достовірність контенту. Функціональність додатку включає персоналізацію стрічки новин, але можливості налаштування є менш гнучкими порівняно з Google News або Flipboard. Інтерфейс є зручним та інтуїтивно зрозумілим. Монетизація здійснюється через рекламу, яка може бути присутньою у додатку.

SmartNews фокусується на швидкому доставленні найважливіших новин дня. Функціональність додатку є середньою, оскільки можливості персоналізації обмежені. Проте додаток вирізняється швидким завантаженням контенту та можливістю офлайн-доступу. Достовірність контенту є середньою, оскільки джерела можуть бути різноманітними. Інтерфейс є простим та зручним для швидкого ознайомлення з новинами.

Telegram використовується як платформа для дистрибуції новин через канали та боти. Функціональність додатку в цьому контексті є обмеженою, оскільки основний фокус месенджера — це обмін повідомленнями. Персоналізація контенту залежить від вибору каналів користувачем. Безпека та приватність знаходяться на середньому рівні, хоча Telegram і пропонує шифрування. Достовірність контенту може бути низькою, оскільки відсутня

централізована модерація та контроль за джерелами інформації.

Reddit є соціальною платформою, де користувачі можуть ділитися новинами та обговорювати їх у тематичних спільнотах. Функціональність додатку є високою, включаючи можливості персоналізації шляхом підписки на субреддіти. Проте достовірність контенту залежить від спільноти та може варіюватися. Інтерфейс може бути складним для новачків, але надає широкі можливості для взаємодії.

Inshorts пропонує короткі новини у стислому форматі, що зручно для швидкого ознайомлення з подіями. Функціональність додатку є низькою, оскільки можливості персоналізації та взаємодії обмежені. Достовірність контенту є середньою, а інтерфейс простим та зручним для користувачів, які шукають швидкий доступ до новин.

Отже, проведений порівняльний аналіз показує, що жоден з існуючих мобільних додатків не поєднує в собі всі необхідні характеристики, які б повністю задовольняли потреби користувачів у дистрибуції новин. Найбільш збалансованими є Google News та Apple News, які пропонують високу функціональність, зручність використання, персоналізацію та безпеку. Проте вони мають недоліки, пов'язані з формуванням "інформаційних бульбашок" та обмеженнями доступності.

Flipboard та Microsoft News також пропонують якісний контент та зручний інтерфейс, але можуть поступатися в можливостях персоналізації та контролю за достовірністю інформації. SmartNews та Inshorts підходять для користувачів, які цінують швидкість отримання новин, але обмежені в функціональності та персоналізації.

Telegram та Reddit надають можливості для взаємодії та обговорення новин, але стикаються з проблемами достовірності контенту та безпеки даних. Відсутність централізованої модерації може призводити до поширення недостовірної інформації.

## 2.3. Переваги та недоліки існуючих сервісів для дистрибуції новин

Аналізуючи існуючі сервіси для управління та дистрибуції новин, важливо виділити ключові аспекти щодо використаного програмного забезпечення та технологій, які впливають на їхню функціональність, зручність та адаптивність. Сучасні платформи для дистрибуції новин використовують широкий спектр технологій, включаючи веб-фреймворки, мобільні платформи, алгоритми штучного інтелекту та машинного навчання, а також засоби для забезпечення безпеки та приватності користувачів.

Веб-фреймворки та мобільні платформи, такі як React Native, Flutter, Angular та Vue.js, дозволяють розробникам створювати кросплатформні додатки з інтуїтивно зрозумілим інтерфейсом. Ці технології забезпечують швидку розробку, легку масштабованість та підтримку на різних пристроях, включаючи смартфони, планшети та веб-браузери. Використання таких фреймворків сприяє створенню додатків з високою продуктивністю та привабливим дизайном.

Алгоритми штучного інтелекту та машинного навчання широко застосовуються для персоналізації контенту та рекомендаційних систем. Сервіси, такі як Google News та Apple News, використовують складні алгоритми для аналізу поведінки користувачів, їхніх вподобань та історії взаємодії з контентом. Це дозволяє надавати користувачам більш релевантний та цікавий контент, підвищуючи їхню залученість.

Засоби забезпечення безпеки та приватності, включаючи шифрування даних, аутентифікацію користувачів та контроль доступу, є критичними для захисту особистої інформації користувачів. Сервіси, які приділяють увагу безпеці, такі як Apple News, мають перевагу в довірі користувачів.

Переваги існуючих сервісів для управління та дистрибуції новин:

1. Зручний та інтуїтивний інтерфейс: Більшість платформ мають

простий у використанні інтерфейс, що робить їх доступними для широкого кола користувачів незалежно від їхнього технічного досвіду.

2. Персоналізація контенту: Алгоритми машинного навчання дозволяють надавати користувачам контент, який відповідає їхнім інтересам, що підвищує релевантність та залученість.

3. Швидкий доступ до актуальних новин: Сервіси забезпечують оперативне оновлення інформації, дозволяючи користувачам бути в курсі останніх подій у режимі реального часу.

4. Підтримка мультимедійного контенту: Можливість перегляду не лише текстових новин, але й фото, відео та аудіоматеріалів, що робить споживання інформації більш різноманітним та цікавим.

5. Інтеграція з соціальними мережами: Користувачі можуть легко ділитися новинами з друзями та підписниками, що сприяє поширенню інформації та взаємодії.

6. Офлайн-доступ: Деякі додатки дозволяють зберігати новини для читання без підключення до інтернету, що зручно для користувачів з обмеженим доступом до мережі.

Недоліки існуючих сервісів для управління та дистрибуції новин:

1. "Інформаційні бульбашки": Персоналізація контенту може призводити до того, що користувачі отримують лише ті новини, які підтверджують їхні погляди, обмежуючи різноманітність інформації та сприяючи поляризації думок.

2. Проблеми з достовірністю контенту: Відсутність ефективних механізмів модерації та перевірки фактів може призводити до поширення фейкових новин та дезінформації, особливо на платформах з користувацьким контентом, таких як Reddit або Telegram.

3. Питання приватності та безпеки даних: Збір великої кількості особистих даних для персоналізації може викликати занепокоєння щодо конфіденційності. Деякі платформи можуть передавати дані третім сторонам або

не забезпечувати належний рівень захисту.

4. Реклама та монетизація: Надмірна кількість реклами в додатках може негативно впливати на користувацький досвід. Деякі сервіси обмежують доступ до контенту без платної підписки, що може бути незручним для користувачів.

5. Обмеження в персоналізації: Не всі платформи надають достатні можливості для налаштування контенту під індивідуальні потреби користувача. Це може призводити до отримання нерелевантної або нецікавої інформації.

6. Технічні обмеження: Деякі додатки можуть вимагати постійного підключення до інтернету, споживати багато ресурсів пристрою або мати проблеми з сумісністю на різних платформах.

7. Залежність від алгоритмів: Надмірне покладання на алгоритми може призводити до непередбачуваних результатів у формуванні новинної стрічки, оскільки алгоритми можуть помилятися або бути упередженими.

Вплив технологій та інструментів на сервіси:

- Використання AI та ML: Хоча штучний інтелект покращує персоналізацію, він також може сприяти формуванню інформаційних бульбашок та упередженості. Алгоритми можуть підсилювати вже існуючі вподобання користувачів, не пропонуючи альтернативних точок зору.
- Веб та мобільні фреймворки: Сучасні фреймворки полегшують розробку кросплатформних додатків, але можуть мати проблеми з оптимізацією продуктивності. Крім того, складність цих технологій вимагає висококваліфікованих розробників, що може збільшувати вартість розробки та підтримки.
- Безпека та приватність: Недостатня увага до безпеки може призвести до витоку даних або несанкціонованого доступу. Використання децентралізованого шифрування та інших сучасних технологій може підвищити рівень захисту, але вимагає додаткових ресурсів та експертизи.

Потреби користувачів, які не повністю задовольняються існуючими сервісами:

- Більший контроль над контентом: Користувачі бажають мати можливість більш тонко налаштовувати джерела та теми, від яких вони отримують новини, а також фільтрувати небажаний контент.
- Прозорість алгоритмів: Зростає потреба в розумінні того, як формуються рекомендації та чому користувачам пропонуються ті чи інші новини.
- Достовірність та перевірка фактів: Користувачі хочуть бути впевненими в надійності отриманої інформації та мати доступ до інструментів, які допомагають виявляти фейкові новини.
- Захист приватності: Занепокоєння щодо збору та використання особистих даних спонукає користувачів шукати сервіси, які мінімізують збір інформації та забезпечують високий рівень конфіденційності.

Отже, існуючі сервіси для управління та дистрибуції новин мають значні переваги, але також стикаються з низкою проблем, які обмежують їхню ефективність та задоволення потреб користувачів. Для створення більш досконалого рішення необхідно врахувати наступні аспекти:

- Розробка інтуїтивно зрозумілого інтерфейсу, який забезпечить легкий доступ до функцій персоналізації та управління контентом.
- Впровадження прозорих алгоритмів персоналізації, які надаватимуть користувачам можливість розуміти та контролювати процес формування новинної стрічки.
- Підвищення рівня безпеки та приватності, використовуючи сучасні методи шифрування та мінімізуючи збір особистих даних.
- Забезпечення достовірності контенту шляхом співпраці з перевіреними джерелами та впровадження механізмів перевірки фактів.
- Надання інструментів для боротьби з інформаційним перенасиченням, таких як можливість фільтрувати або групувати новини за пріоритетом або темами.
- Підтримка кросплатформності та оптимізації продуктивності, щоб

додаток був доступним та швидким на різних пристроях та операційних системах.

- Врахування зворотного зв'язку від користувачів для постійного вдосконалення сервісу та адаптації до змінних потреб аудиторії.

Розуміння переваг та недоліків існуючих сервісів для управління та дистрибуції новин є ключовим для розробки нового мобільного додатку, який зможе задовольнити сучасні вимоги користувачів. Поєднання передових технологій з акцентом на безпеку, персоналізацію та зручність використання дозволить створити конкурентоспроможний продукт, який вирішить існуючі проблеми та запропонує інноваційні рішення в сфері дистрибуції новин.

## 2.4. Використання децентралізованих систем у сучасних рішеннях безпеки та актуальні підходи до побудови API та архітектури мобільних додатків

У сучасному світі, де обсяги даних зростають експоненційно, питання безпеки та приватності інформації набувають критичного значення. Традиційні централізовані системи зберігання та обробки даних стикаються з викликами, пов'язаними з масштабованістю, вразливістю до атак та залежністю від єдиних точок відмови. У зв'язку з цим, децентралізовані системи стають все більш популярними як ефективні рішення для підвищення безпеки та стійкості інформаційних систем.

Блокчейн та розподілені реєстри.

Блокчейн — це розподілена база даних, яка зберігає інформацію у вигляді послідовного ланцюжка блоків. Кожен блок містить дані транзакцій та

криптографічний хеш попереднього блоку, що забезпечує незмінність та цілісність даних.

Прозорість у блокчейн-системах забезпечується тим, що всі учасники мережі мають доступ до єдиної версії даних, що сприяє підвищенню довіри та контролю з боку користувачів.

Невід'ємною характеристикою блокчейну є незмінність даних, оскільки інформацію, яка вже була додана до ланцюга блоків, неможливо змінити без досягнення консенсусу більшості учасників мережі. Крім того, децентралізація системи, яка характеризується відсутністю центрального керуючого органу, мінімізує ризики зловживань і зменшує ймовірність здійснення атак завдяки розподіленій структурі управління.

Застосування блокчейну у сфері безпеки охоплює різні аспекти, зокрема аутентифікацію та управління ідентифікацією. У цьому контексті блокчейн сприяє розробці децентралізованих систем ідентифікації (DID), що дозволяють користувачам контролювати свої персональні дані без залучення посередників, підвищуючи рівень конфіденційності та безпеки. Крім того, блокчейн відіграє важливу роль у захисті даних пристроїв Інтернету речей (IoT), забезпечуючи безпечну взаємодію між пристроями та запобігаючи можливості несанкціонованого доступу чи маніпуляцій з інформацією.

Пірингові мережі (P2P).

Пірингові мережі базуються на прямій взаємодії між вузлами без центрального сервера. Це забезпечує можливість ефективного розподілу навантаження, оскільки кожен вузол мережі бере участь у процесах передачі та обробки даних, що дозволяє оптимізувати роботу системи. Окрім цього, підвищується стійкість до відмов завдяки відсутності єдиної точки відмови, що значно зменшує вразливість мережі до збоїв і атак.

Приклади застосування технології P2P у сфері безпеки включають захищене розповсюдження файлів у файлообмінних мережах, де обмін даними здійснюється без залучення центрального сервера, що підвищує стійкість до

збоїв та зловживань. Крім того, децентралізовані VPN-системи забезпечують високий рівень приватності та анонімності користувачів під час доступу до інтернету, мінімізуючи ризики перехоплення даних чи стеження.

Криптографія та шифрування.

Криптографія є основою для забезпечення конфіденційності та цілісності даних у децентралізованих системах.

Сучасні методи криптографічного захисту даних охоплюють асиметричне шифрування, яке базується на використанні пар відкритого та закритого ключів для забезпечення конфіденційності та захисту інформації. Важливою складовою є також застосування хеш-функцій, які гарантують цілісність даних шляхом створення унікальних цифрових відбитків, що дозволяє виявляти будь-які зміни в інформації.

Криптографія відіграє ключову роль у забезпеченні безпеки завдяки таким механізмам, як цифрові підписи, які підтверджують автентичність та незмінність повідомлень, забезпечуючи надійний захист від підробок. Крім того, енд-ту-енд шифрування є ефективним методом захисту комунікацій, оскільки воно унеможливлює несанкціонований доступ до переданих даних навіть на рівні постачальників послуг.

Децентралізовані ідентифікаційні системи (DID).

DID дозволяють користувачам контролювати свої цифрові ідентичності без необхідності довіряти централізованим організаціям.

Переваги застосування сучасних технологій включають підвищений рівень приватності, оскільки користувачі отримують можливість самостійно контролювати, які дані надавати та з ким їх ділитися. Крім того, забезпечується вищий рівень безпеки завдяки зниженню ризику компрометації даних, що часто виникає через атаки на центральні сервери.

Серед прикладів практичного застосування можна виокремити управління доступом, що дозволяє контролювати, хто має право отримувати доступ до конкретної інформації чи ресурсів, забезпечуючи ефективний захист даних.

Також важливу роль відіграють електронні підписи та транзакції, які дозволяють безпечно підтверджувати операції без залучення посередників, мінімізуючи ризики шахрайства та несанкціонованого втручання.

Актуальні підходи до побудови API.

Сучасні додатки все частіше взаємодіють з серверами через API (Application Programming Interface). Розвиток мобільних та веб-додатків вимагає ефективних, масштабованих та безпечних API.

Ось деякі з актуальних підходів:

RESTful API

REST (Representational State Transfer) — це архітектурний стиль, який використовує стандартизовані HTTP методи (GET, POST, PUT, DELETE) для взаємодії між клієнтом та сервером.

Переваги RESTful API полягають у його простоті та зручності використання завдяки застосуванню протоколу HTTP та стандартних методів, таких як GET, POST, PUT і DELETE, що робить взаємодію з ресурсами інтуїтивно зрозумілою та ефективною. Крім того, RESTful архітектура забезпечує високу масштабованість, дозволяючи обробляти значну кількість запитів і підтримувати роботу навіть при зростанні навантаження. Важливою особливістю є також можливість кешування відповідей, що сприяє підвищенню продуктивності та зниженню затримок при повторному зверненні до тих самих ресурсів.

GraphQL

GraphQL — це мова запитів для API, розроблена Facebook, яка дозволяє клієнтам запитувати саме ті дані, які їм потрібні, і нічого зайвого.

Переваги сучасних підходів до роботи з API включають ефективність у передачі даних, що досягається за рахунок зменшення обсягу переданих даних та кількості запитів, необхідних для отримання потрібної інформації. Гнучкість системи проявляється в можливості клієнта самостійно визначати структуру відповіді відповідно до власних потреб, що значно підвищує зручність і

оптимізацію взаємодії. Додатковою перевагою є використання єдиної кінцевої точки, через яку відбуваються всі запити за допомогою одного URL, що спрощує архітектуру та управління запитами до сервера.

gRPC.

gRPC - це фреймворк з відкритим кодом, розроблений Google, який використовує протоколи буферів (Protocol Buffers) для серіалізації даних. Основними перевагами сучасних технологій передачі даних є висока продуктивність, що досягається завдяки використанню бінарного протоколу, який забезпечує швидку та ефективну передачу інформації з мінімальними затримками. Крім того, підтримка різних мов програмування дозволяє автоматично генерувати код для різних платформ, забезпечуючи сумісність і зручність інтеграції у багатоплатформних середовищах. Важливою особливістю є бі-дирекційний стримінг, який підтримує асинхронний обмін даними в реальному часі, що є особливо актуальним для додатків з високими вимогами до швидкодії та інтерактивності.

Мікросервісна архітектура

Мікросервіси - це підхід до побудови програмного забезпечення як набору незалежних сервісів, які взаємодіють між собою через легкі механізми, часто через HTTP/REST або повідомлення [3].

Переваги мікросервісної архітектури включають високу масштабованість, оскільки кожен сервіс може масштабуватися незалежно від інших, що дозволяє ефективно розподіляти ресурси відповідно до навантаження. Незалежність розробки є ще однією важливою особливістю, адже команди можуть працювати над різними сервісами паралельно, що прискорює процес розробки та впровадження нових функцій. Крім того, мікросервісна архітектура забезпечує стійкість до відмов, оскільки збій одного сервісу не впливає на роботу решти системи, що підвищує загальну надійність.

Водночас, така архітектура має певні виклики. Зокрема, ускладнюється управління системою через необхідність оркестрації великої кількості сервісів,

що вимагає використання додаткових інструментів для їх координації. Тестування та відлагодження також стають більш складними через труднощі у відстеженні помилок у розподіленому середовищі, де взаємодія між сервісами може створювати непередбачувані ситуації.

API Gateway.

API Gateway діє як єдина точка входу для всіх клієнтських запитів до мікросервісів. Основні функції систем управління запитами включають маршрутизацію, яка забезпечує направлення запитів до відповідних сервісів відповідно до заданих правил і логіки. Важливим аспектом є забезпечення безпеки, що охоплює процеси аутентифікації, авторизації та обмеження доступу до ресурсів, таким чином захищаючи дані та сервіси від несанкціонованого використання. Крім того, функція трансформації даних дозволяє змінювати формат запитів та відповідей, забезпечуючи сумісність між різними системами та спрощуючи взаємодію в гетерогенних середовищах.

Актуальні підходи до архітектури мобільних додатків.

Розробка мобільних додатків вимагає застосування сучасних архітектурних патернів та технологій, які забезпечують високу продуктивність, зручність використання та легкість підтримки.

1. Архітектурні патерни.

MVC (Model-View-Controller)

Model: Дані та бізнес-логіка.

View: Відображення інформації користувачу.

Controller: Обробка подій та взаємодія між Model та View.

MVP (Model-View-Presenter)

Presenter: Посередник між Model та View, який містить логіку відображення.

Переваги: Полегшує тестування та розділяє відповідальності.

MVVM (Model-View-ViewModel)

ViewModel: Забезпечує дані для View та обробляє користувацькі дії.

Переваги: Підтримка двостороннього зв'язку між View та ViewModel, що спрощує синхронізацію стану.

2. Clean Architecture.

Принципи:

Інверсія залежностей: Вищі рівні не залежать від нижчих.

Розділення на шари: Презентація, домен, дані.

Переваги: Гнучкість, легкість у підтримці та масштабуванні.

3. Технології та інструменти.

Кросплатформна розробка.

React Native: Використовує JavaScript та дозволяє створювати додатки для iOS та Android.

Flutter: Розроблений Google, використовує мову Dart, забезпечує високу продуктивність та нативний інтерфейс.

Переваги:

Зменшення витрат: Один код для обох платформ.

Прискорення розробки: Швидке прототипування та оновлення.

Асинхронне програмування.

Coroutines (Kotlin): Полегшують роботу з асинхронними операціями, такими як мережеві запити або робота з базою даних.

Promises/Futures: Упорядковують асинхронні операції та обробку результатів.

Розширення та ін'єкція залежностей.

Dependency Injection: Патерн, який спрощує управління залежностями між компонентами.

Dagger/Hilt (Android): Бібліотеки для впровадження залежностей, які покращують модульність та тестованість коду.

Реактивне програмування.

RxJava/RxSwift: Дозволяють обробляти потоки даних та подій асинхронно.

Переваги:

Обробка великих обсягів даних: Ефективне управління потоками.

Легкість управління станом: Спрощує обробку складних взаємодій.

4. Безпека та приватність.

Шифрування даних: Захист локально збережених даних та переданих по мережі.

Аутентифікація та авторизація: Використання безпечних протоколів (OAuth2, JWT) для захисту доступу.

Відповідність стандартам: Дотримання GDPR та інших регуляторних вимог щодо приватності.

5. Сучасні тенденції та виклики.

Інтеграція штучного інтелекту.

Персоналізація: Використання машинного навчання для надання користувачам релевантного контенту.

Обробка природної мови: Покращення взаємодії через голосові команди та чат-боти.

Інтернет речей (IoT).

Взаємодія з пристроями: Розробка додатків, які можуть керувати та отримувати дані від IoT-пристроїв.

Безпека: Захист від несанкціонованого доступу до пристроїв та даних.

Хмарні технології та серверлесс-архітектура.

Серверлесс-обчислення: Запуск коду без управління серверами, що спрощує масштабування та знижує витрати.

Функції як сервіс (FaaS): Виконання невеликих функцій у відповідь на події.

Сучасні рішення в галузі безпеки та розробки мобільних додатків активно використовують децентралізовані системи та передові технології для забезпечення високого рівня безпеки, продуктивності та зручності використання. Децентралізація дозволяє підвищити стійкість систем до атак, забезпечити

конфіденційність даних та надати користувачам більше контролю над інформацією.

У сфері розробки API та мобільних додатків актуальними є підходи, що сприяють гнучкості, масштабованості та швидкій розробці. Використання сучасних архітектурних патернів, кросплатформних інструментів та реактивного програмування дозволяє створювати додатки, які відповідають високим вимогам користувачів та ринку.

Загалом, інтеграція децентралізованих систем у поєднанні з актуальними підходами до розробки сприяє створенню інноваційних рішень, які відповідають сучасним викликам та забезпечують ефективність, безпеку та якість програмного забезпечення.

# РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

## 3.1. Опис обраних технологій та підходів для розробки мобільного додатку та API

При розробці мобільного додатку для дистрибуції новин та відповідного API було важливо забезпечити високу продуктивність, надійність, безпеку та зручність використання. Для досягнення цих цілей були обрані сучасні технології та підходи, які відповідають найкращим практикам розробки програмного забезпечення. Нижче детально описані обрані технології та їх роль у проєкті.

Hilt (Dagger Hilt).

Hilt є бібліотекою для впровадження залежностей (Dependency Injection) в Android-додатках, побудованою на основі Dagger. Вона спрощує процес управління залежностями між різними компонентами додатку, що підвищує модульність, тестованість та масштабованість коду. Завдяки Hilt, розробники можуть легко впроваджувати залежності у такі компоненти, як Activity, Fragment, ViewModel та інші, без необхідності писати багато шаблонного коду.

Hilt автоматизує створення та постачання об'єктів залежностей, дозволяючи уникнути ручного створення об'єктів та управління їх життєвим циклом. Це особливо важливо для великих додатків, де кількість залежностей може бути значною. Використання Hilt сприяє зменшенню зв'язаності між компонентами, що полегшує підтримку та розширення функціональності додатку [1].

AndroidX SwipeRefreshLayout.

SwipeRefreshLayout з пакету AndroidX забезпечує зручний спосіб реалізації функціоналу "потягни, щоб оновити" (Pull-to-Refresh) у додатку. Цей

компонент дозволяє користувачам оновлювати контент шляхом простого жесту проведення вниз, що є інтуїтивно зрозумілим та покращує взаємодію з додатком. Використання SwipeRefreshLayout підвищує зручність та швидкість доступу до актуальної інформації, що є критичним для додатку новин.

SwipeRefreshLayout легко інтегрується з існуючими компонентами інтерфейсу, такими як RecyclerView або ListView. Це дозволяє швидко додати функціонал оновлення без значних змін у структурі додатку. Крім того, компонент підтримує налаштування кольорів та анімацій, що дозволяє адаптувати його до дизайну додатку та покращити естетичний вигляд.

Lottie.

Lottie — це бібліотека, яка дозволяє відтворювати анімації, створені в Adobe After Effects та експортовані у форматі JSON за допомогою плагіна Bodymovin. Використання Lottie у додатку додає візуальної привабливості та робить інтерфейс більш динамічним. Анімації можуть використовуватися для відображення станів завантаження, підтвердження дій користувача, помилок або інших подій, що покращує загальний користувацький досвід.

Lottie підтримує складні анімації з високою якістю відображення без значного впливу на продуктивність додатку. Це досягається за рахунок використання апаратного прискорення та оптимізації процесу відтворення анімацій. Використання Lottie дозволяє уникнути необхідності працювати з великими файлами зображень або відео, зменшуючи розмір додатку та споживання ресурсів.

JWTDecode.

JWTDecode — це бібліотека для роботи з JSON Web Tokens (JWT) на клієнтській стороні. Використання JWT для автентифікації та авторизації користувачів є сучасним та безпечним підходом, який дозволяє передавати необхідну інформацію у зашифрованому вигляді. JWTDecode спрощує процес декодування та валідації токенів у мобільному додатку, забезпечуючи безпечний

доступ до захищених ресурсів [11].

Використання JWT дозволяє додатку самостійно перевіряти дійсність токену та його термін дії, що підвищує безпеку та зменшує навантаження на сервер, оскільки не потребує постійних запитів для перевірки автентифікації. Це особливо важливо для додатків, які повинні працювати швидко та ефективно навіть при обмеженому або нестабільному інтернет-з'єднанні.

AndroidX DataStore Preferences.

DataStore є сучасною альтернативою SharedPreferences для зберігання невеликої кількості даних у додатку [17]. Вона забезпечує асинхронне та безпечне зберігання налаштувань та конфігурацій користувача з використанням Kotlin Coroutines або RxJava. DataStore підтримує два варіанти: Preferences DataStore для зберігання пар ключ-значення та Proto DataStore для зберігання типізованих даних з використанням Protocol Buffers.

Використання DataStore покращує продуктивність додатку, оскільки всі операції виконуються асинхронно, запобігаючи блокуванню головного потоку. Це забезпечує плавність роботи інтерфейсу та швидкий доступ до даних. Крім того, DataStore є більш стійким до помилок та корупції даних, пропонує кращі можливості для міграції та оновлення схеми зберігання даних.

Retrofit2.

Retrofit2 — це потужна бібліотека для взаємодії з RESTful веб-сервісами, яка спрощує роботу з мережевими запитами. Вона дозволяє визначати інтерфейси API, автоматично перетворювати HTTP-запити у виклики методів та обробляти відповіді сервера [9]. Retrofit підтримує різні конвертери, такі як Gson або Moshi, для серіалізації та десеріалізації даних у форматі JSON.

Використання Retrofit робить код більш чистим та організованим, зменшує кількість шаблонного коду та полегшує обробку помилок. Підтримка асинхронних запитів з використанням Kotlin Coroutines або RxJava дозволяє

виконувати мережеві операції без блокування головного потоку, що підвищує продуктивність додатку та покращує користувацький досвід.

Gson та конвертер Gson.

Gson — це бібліотека від Google для серіалізації та десеріалізації об'єктів Java та Kotlin у формат JSON. У поєднанні з Retrofit, Gson автоматично перетворює JSON-відповіді сервера у відповідні моделі даних додатку та навпаки. Це спрощує роботу з даними та зменшує ризик помилок при ручному парсингу.

Gson підтримує складні структури даних, включаючи вкладені об'єкти, колекції та генерики. Це дозволяє легко працювати з різноманітними API та забезпечує гнучкість у розробці. Використання Gson також спрощує налаштування форматування дат, імен полів та інших аспектів серіалізації.

Picasso.

Picasso — це бібліотека для завантаження та відображення зображень у додатку. Вона автоматизує процес завантаження зображень з мережі, кешування їх у пам'яті та на диску, а також відображення у ImageView. Picasso оптимізує використання мережевих ресурсів, автоматично обробляє випадки відсутності зображень або помилок при завантаженні [15].

Використання Picasso підвищує продуктивність додатку та покращує користувацький досвід, оскільки зображення завантажуються асинхронно без блокування головного потоку. Бібліотека також підтримує різні трансформації зображень, такі як обрізка, зміна розміру та застосування фільтрів, що дозволяє адаптувати зображення під потреби інтерфейсу.

OkHttp.

OkHttp є потужним HTTP-клієнтом для Android та Java, який забезпечує ефективне управління мережевими з'єднаннями. Він підтримує сучасні

протоколи, такі як HTTP/2, забезпечує повторне використання з'єднань, кешування відповідей та інші оптимізації. OkHttp є основою для багатьох мережевих бібліотек, включаючи Retrofit.

Використання OkHttp у поєднанні з Retrofit дозволяє отримати переваги обох бібліотек: простоту та зручність використання Retrofit та продуктивність та гнучкість OkHttp. Це забезпечує швидке та надійне виконання мережевих запитів, що є критичним для додатку, який активно взаємодіє з сервером.

AndroidX Navigation Component.

Navigation Component з пакету AndroidX спрощує реалізацію навігації у додатку. Він надає єдиний спосіб визначення та керування навігацією між екранами, підтримує передачу даних між фрагментами та активностями, обробку кнопки "Назад" та інші аспекти навігації. Використання Navigation Component підвищує узгодженість та передбачуваність навігаційної логіки [10].

Navigation Component підтримує візуальне проєктування графа навігації за допомогою редактора у Android Studio, що полегшує розуміння структури додатку та спрощує його підтримку. Також він підтримує Deep Linking, що дозволяє відкривати певні екрани додатку ззовні, наприклад, з веб-посилань або інших додатків [7].

Технології та бібліотеки, використані у API.

Flask.

Flask є мікрофреймворком для мови Python, який використовується для розробки веб-додатків та API. Він забезпечує простий та гнучкий спосіб створення RESTful API з мінімальними накладними витратами. Flask дозволяє швидко визначати маршрути для обробки HTTP-запитів, працювати з різними форматами даних, інтегруватися з базами даних та іншими сервісами [6].

Використання Flask для створення API було обрано через його простоту, гнучкість та активну спільноту розробників [16]. Це дозволяє швидко розгорнути

серверну частину додатку, легко адаптувати її під змінні вимоги та інтегрувати необхідні розширення для додаткової функціональності.

Flask-Limiter.

Flask-Limiter є розширенням для Flask, яке дозволяє обмежувати кількість запитів до API з певної IP-адреси або за іншими критеріями. Це важливо для захисту сервера від надмірного навантаження, запобігання DDoS-атакам та зловживання ресурсами. Flask-Limiter підтримує різні стратегії обмеження, що дозволяє налаштовувати його відповідно до потреб додатку [12].

Використання Flask-Limiter підвищує безпеку та стабільність роботи серверної частини, забезпечує справедливий розподіл ресурсів та покращує загальну продуктивність системи. Це є важливим аспектом для додатків, які повинні обробляти велику кількість запитів та забезпечувати високу доступність сервісу.

PyMySQL.

PyMySQL є бібліотекою для підключення до бази даних MySQL з використанням мови Python. Вона дозволяє виконувати SQL-запити, отримувати та обробляти результати, керувати транзакціями та обробляти помилки. PyMySQL підтримує як синхронний, так і асинхронний режим роботи, що підвищує гнучкість у розробці серверної логіки [14].

Використання PyMySQL забезпечує надійний та ефективний доступ до бази даних, що є критичним для зберігання та отримання даних у додатку. MySQL як база даних була обрана через її продуктивність, масштабованість та широку підтримку в спільноті розробників. Це забезпечує стабільну роботу серверної частини та легкість у подальшому масштабуванні системи.

Обґрунтування вибору технологій та підходів.

Вибір зазначених технологій та бібліотек був обумовлений необхідністю

створення сучасного, надійного та ефективного додатку, який відповідає вимогам користувачів та сучасним стандартам розробки програмного забезпечення. Кожна з обраних технологій була ретельно проаналізована з точки зору її переваг, сумісності з іншими компонентами та здатності задовольнити потреби проєкту.

Використання Hilt для впровадження залежностей полегшує управління складними структурами коду, підвищує модульність та тестованість додатку. Це дозволяє швидко додавати нову функціональність та зменшує ризик помилок при розширенні коду.

Бібліотеки Retrofit, OkHttp та Gson забезпечують ефективну та безпечну взаємодію з сервером, спрощують роботу з мережевими запитами та обробкою даних. Це підвищує продуктивність додатку та покращує користувацький досвід за рахунок швидкого та надійного доступу до інформації.

Використання AndroidX-компонентів, таких як SwipeRefreshLayout та Navigation Component, підвищує зручність використання додатку, забезпечує відповідність сучасним вимогам до дизайну та взаємодії з користувачем. Це сприяє залученню користувачів та їх задоволеності додатком.

На серверній стороні, використання Flask та його розширень, таких як Flask-Limiter, забезпечує швидку розробку та розгортання API з необхідним рівнем безпеки та продуктивності. PyMySQL у поєднанні з MySQL як базою даних забезпечує стабільний та ефективний доступ до даних, що є критичним для успішної роботи додатку.

Загалом, обрані технології та підходи дозволяють створити комплексне рішення, яке забезпечує високу продуктивність, безпеку, масштабованість та зручність використання. Це сприяє успішному впровадженню проєкту, задоволенню потреб користувачів та створенню основи для подальшого розвитку та розширення функціональності додатку.

## 3.2. Методологія розробки програмного забезпечення та побудова блок-схеми системи

Методологія Waterfall, або водоспадна модель, є класичною моделлю управління проєктами, широко застосовуваною у сфері розробки програмного забезпечення. Ця модель передбачає лінійний та послідовний підхід, де кожен етап повинен бути завершений перед початком наступного [8].

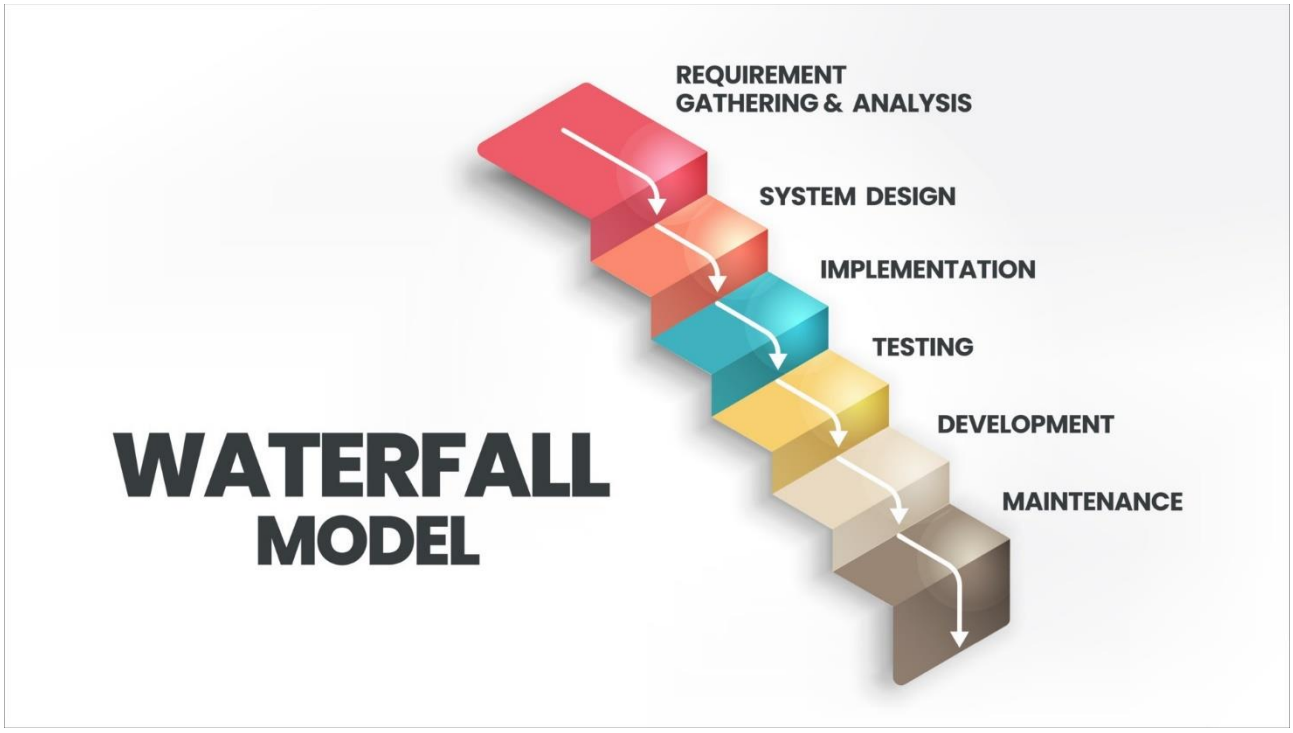


Рис. 3.1 Методологія розробки програмного забезпечення Waterfall

Використання Waterfall забезпечує структурованість процесу, чітке планування та контроль якості на кожному кроці, що особливо важливо при створенні складних систем, таких як мобільний додаток для дистрибуції новин.

На початковому етапі проводиться збір та аналіз вимог до додатку. Основною метою є визначення функціональних та нефункціональних вимог, необхідних для задоволення потреб користувачів та досягнення цілей проєкту. До ключових функціональних можливостей належать отримання та

відображення актуальних новин у зручному форматі, реалізація механізму автентифікації та авторизації користувачів з використанням токенів, можливість оновлення контенту шляхом жесту "потягни, щоб оновити", відображення зображень та анімацій для покращення користувацького досвіду, а також забезпечення безпеки даних та захист від несанкціонованого доступу. У цьому етапі залучаються зацікавлені сторони, такі як потенційні користувачі, замовники та експерти, для формування повного переліку вимог та очікувань щодо функціональності додатку.

Після визначення вимог розробляється технічний проєкт, що описує архітектуру системи, вибір технологій та інструментів, а також взаємодію між компонентами. Прийнято рішення використовувати чисту архітектуру (Clean Architecture) та патерн MVVM (Model-View-ViewModel) для забезпечення модульності, підтримуваності та масштабованості коду [2][5]. Обрано сучасні бібліотеки та фреймворки: Hilt для ін'єкції залежностей, Retrofit для мережевих запитів, Picasso для завантаження та кешування зображень, а також Flask для реалізації серверного API. Розроблено макети та прототипи екранів, визначено навігацію між ними з урахуванням принципів Material Design, щоб створити інтуїтивно зрозумілий та привабливий інтерфейс. Заплановано використання JWT (JSON Web Tokens) для безпечної автентифікації та авторизації користувачів, а також Flask-Limiter для обмеження кількості запитів до API.

На етапі реалізації здійснюється безпосередня розробка клієнтської та серверної частин додатку згідно з технічним проєктом. У клієнтській частині створено екрани додатку з використанням фрагментів та одного активності (Single Activity), впроваджено навігацію за допомогою Navigation Component. Використовуючи патерн MVVM, створено ViewModel для управління даними та логікою додатку, забезпечено взаємодію між інтерфейсом користувача та моделлю даних. Впроваджено Hilt для ін'єкції залежностей, Retrofit для виконання мережевих запитів, Picasso для завантаження зображень, а також Lottie для відтворення анімацій, що підвищує візуальну привабливість додатку.

У серверній частині створено RESTful API з використанням Flask, реалізовано ендпоїнти для отримання новин, авторизації користувачів та управління контентом. Налаштовано MySQL для зберігання даних про новини, користувачів та іншу необхідну інформацію. Впроваджено автентифікацію з використанням JWT, встановлено обмеження на кількість запитів за допомогою Flask-Limiter, реалізовано шифрування даних для підвищення безпеки.

Після завершення розробки проводиться ретельне тестування додатку та серверної частини з метою виявлення та виправлення помилок, а також забезпечення відповідності вимогам. Перевірено всі функціональні можливості, включаючи завантаження новин, автентифікацію, оновлення контенту та взаємодію з користувачем. Оцінено зручність використання, коректність відображення елементів інтерфейсу на різних пристроях та екранах. Перевірено надійність механізмів автентифікації, захисту даних, стійкість до несанкціонованого доступу та атак. Оцінено швидкодію додатку, час завантаження контенту, оптимізацію використання ресурсів пристрою.

Після успішного тестування додаток готовий до розгортання та доступу користувачам. Підготовлено необхідні матеріали: іконки, скріншоти, опис, проведено процедуру публікації додатку в Google Play Market та інших платформах. Розгорнуто серверну частину на хостингу або в хмарному середовищі, забезпечено безперебійну роботу API та бази даних.

Після розгортання розпочинається етап обслуговування, який включає постійний моніторинг, підтримку та вдосконалення системи. Відбувається відстеження роботи додатку та сервера, аналіз журналів, виявлення та усунення потенційних проблем. Збираються відгуки користувачів, впроваджуються нові функції та поліпшення, оптимізується продуктивність та безпека. Надається допомога користувачам у вирішенні проблем, відповіді на запитання, оперативне реагування на критичні ситуації.

Архітектура системи складається з двох основних компонентів: клієнтської частини (мобільний додаток) та серверної частини (API та база

даних). Взаємодія між ними здійснюється через мережеві запити за допомогою протоколу HTTP(S). У клієнтській частині інтерфейс користувача відповідає за відображення інформації та взаємодію з користувачем, реалізований з використанням Material Design та компонентів AndroidX. Логіка додатку управляє даними та станом, взаємодіє з репозиторіями для отримання та зберігання даних. Репозиторії та сервіси відповідають за доступ до даних, виконання мережевих запитів, зберігання налаштувань та кешування. Мережевий шар реалізований з використанням Retrofit та OkHttp для виконання HTTP-запитів до серверного API [18].

Серверна частина включає API-сервер на базі Flask, який обробляє запити від клієнтського додатку, виконує бізнес-логіку, взаємодіє з базою даних. База даних MySQL використовується для зберігання даних про новини, користувачів, налаштувань та іншої інформації. Модулі безпеки включають механізми автентифікації та авторизації з використанням JWT, обмеження запитів за допомогою Flask-Limiter, шифрування даних для захисту інформації.

Блок-схема системи відображає основні компоненти та їх взаємодію під час роботи додатку. Користувач взаємодіє з мобільним додатком через інтерфейс, який передає дії до ViewModel для визначення необхідних операцій. ViewModel звертається до репозиторію для отримання або відправки даних. Репозиторій вирішує, чи дані доступні локально, чи потрібно виконати мережевий запит. Якщо необхідно, мережевий шар виконує запит до серверного API. Серверний API приймає запит, перевіряє автентичність за допомогою JWT, обробляє його та звертається до бази даних. Отримані дані передаються назад до мобільного додатку, де репозиторій отримує дані, передає їх до ViewModel, яка оновлює стан додатку. Інтерфейс користувача відображає оновлену інформацію користувачу.

У процесі взаємодії реалізовано механізми безпеки та автентифікації. При першому запуску або необхідності автентифікації додаток надсилає дані користувача до серверного API, який генерує та повертає JWT-токен. Мобільний

додаток зберігає токен з використанням DataStore, забезпечуючи безпечне зберігання. Подальші запити до серверного API містять токен у заголовках для перевірки автентичності.

Користувач може оновити новини за допомогою жесту "потягни, щоб оновити". ViewModel ініціює запит до репозиторію, який виконує мережевий запит до серверного API. Отримані новини зберігаються локально для доступу в офлайн-режимі та відображаються в інтерфейсі користувача.

Використання методології Waterfall при розробці мобільного додатку для дистрибуції новин дозволило чітко структурувати процес, забезпечити планування та контроль на кожному етапі. Детальний опис вимог, ретельне проєктування та послідовна реалізація сприяли створенню якісного та надійного продукту. Побудова блок-схеми системи допомогла візуалізувати взаємодію між компонентами та краще зрозуміти потоки даних, що було корисним під час розробки та тестування.

Завдяки обранню сучасних технологій та підходів вдалося досягти високої продуктивності, масштабованості та зручності використання додатку. Реалізація серверної частини на базі Flask з використанням JWT та Flask-Limiter забезпечила необхідний рівень безпеки та стабільності сервісу.

У підсумку, застосування методології Waterfall та продумана архітектура системи дозволили створити мобільний додаток, який відповідає сучасним вимогам та очікуванням користувачів, а також заклали основу для подальшого розвитку та вдосконалення продукту.

Діаграма демонструє архітектуру мобільного додатку та API, які взаємодіють для надання користувачеві функціональності отримання та керування новинами. Нижче представлено покроковий опис основних етапів, що охоплюють серверну та клієнтську частини системи.

1. Ініціалізація системи

   a. API: Сервер запускається та налаштовується для обробки вхідних запитів. Включає налаштування Flask, Flask-Limiter для обмеження запитів та визначення ендпоїнтів для роботи з базою даних.
   b. Додаток: На стороні клієнта завантажуються необхідні компоненти, налаштовується взаємодія з ViewModel, яка забезпечує збереження даних навіть при зміні життєвого циклу фрагментів.
2. Авторизація користувача
   a. При кожному запиті до серверного API перевіряється наявність токену автентифікації. Якщо токен відсутній або недійсний, сервер повертає помилку, і доступ до захищених функцій блокується.
   b. JWT (JSON Web Tokens) використовується для безпечного збереження автентифікаційних даних на клієнтській стороні.
3. Робота з новинами
   a. Завантаження новин: Клієнтський додаток надсилає запит до ендпоїнта /api/news. Сервер перевіряє автентифікацію, звертається до бази даних MySQL для отримання даних і повертає список новин у форматі JSON. ViewModel отримує відповідь і передає її до фрагмента для відображення.
   b. Публікація новин: У разі, якщо користувач працює в режимі адміністратора, він може надсилати запити до ендпоїнта /api/post. Сервер обробляє запит, зберігає дані в базі та повертає підтвердження.
4. Обмеження запитів
   a. Flask-Limiter контролює кількість запитів до API, забезпечуючи захист від надмірного навантаження.

Наприклад, для публікації новин діє обмеження "1 запит на хвилину", що запобігає спаму.

5. Режими користувача
    a. Користувач: Має доступ лише до перегляду новин. Використовує такі екрани: завантаження, перегляд стрічки, налаштування.
    b. Адміністратор: Окрім функцій звичайного користувача, може публікувати, редагувати або видаляти новини через відповідні ендпоїнти.
6. Робота з базою даних
    a. Серверна частина API використовує MySQL для зберігання інформації про новини та користувачів. Реалізовано операції INSERT, SELECT, DELETE, які забезпечують функціональність системи.
    b. Усі взаємодії з базою даних захищені, а дані шифруються перед зберіганням.
7. Обробка помилок
    a. На рівні API обробляються помилки, наприклад, відсутність параметрів у запитах або перевищення лімітів. Користувач отримує відповідні повідомлення про стан виконання операції.
8. Завершення сеансу
    a. При завершенні роботи система звільняє ресурси, видаляє тимчасові дані з клієнтської частини, зберігаючи тільки необхідну інформацію (наприклад, токен у DataStore).

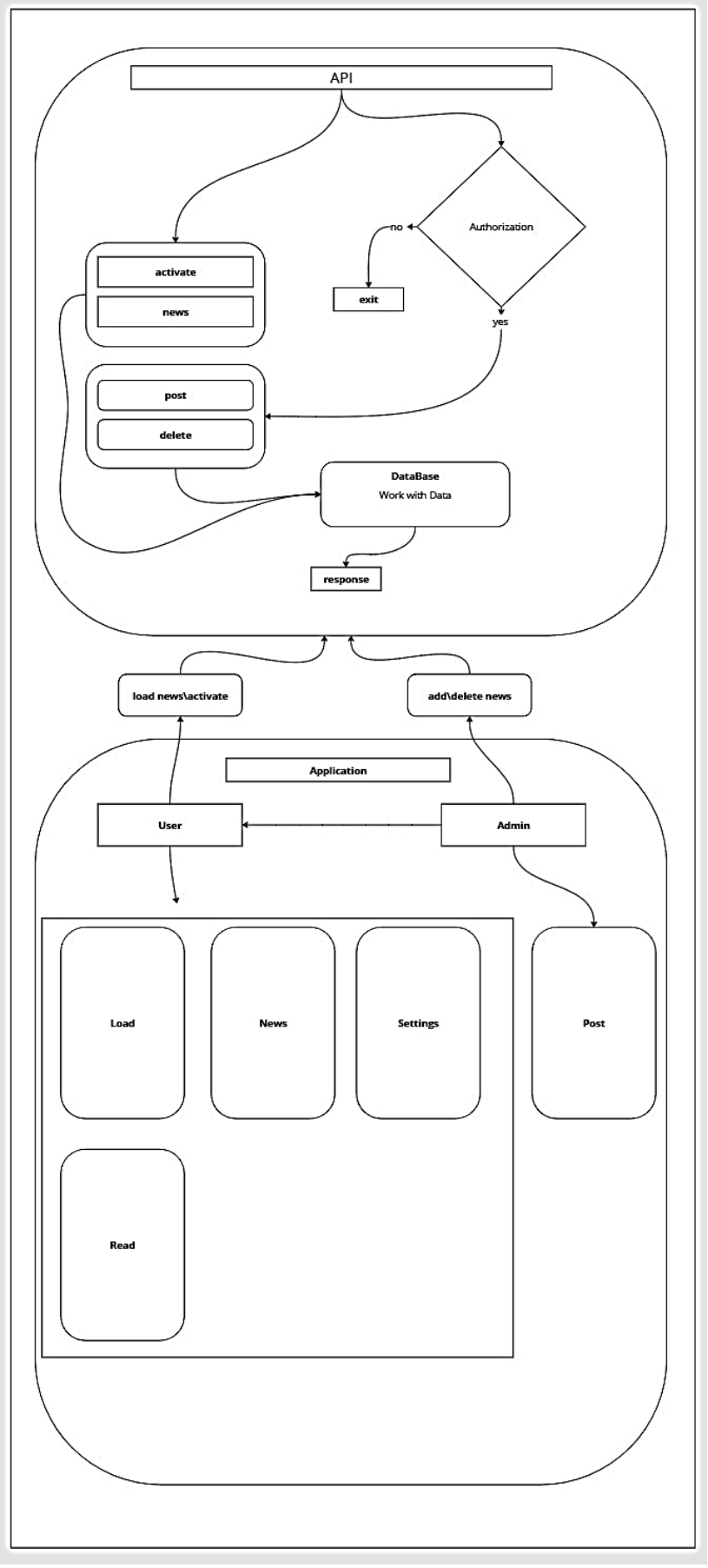


Рисунок 3. 2. Блок схема ПЗ розробленого додатку та API

## 3.3. Опис чистої архітектури та патерну MVVM

Сучасна розробка програмного забезпечення приділяє значну увагу архітектурним підходам, які сприяють створенню гнучких, масштабованих та підтримуваних додатків. Одним із таких підходів є Чиста архітектура (Clean Architecture), яка поєднує в собі найкращі практики розробки та принципи об'єктно-орієнтованого програмування. У поєднанні з архітектурним патерном MVVM (Model-View-ViewModel), цей підхід забезпечує чітку структуру коду, полегшує його підтримку та розширення функціональності.

Чиста архітектура була запропонована Робертом Мартіном і спрямована на створення систем із високою якістю коду, легкою підтримкою та розширюваністю. Основна ідея полягає в розділенні системи на шари з чітко визначеними відповідальностями та залежностями. Це дозволяє ізолювати бізнес-логіку від деталей реалізації, таких як бази даних, фреймворки та інтерфейси користувача.

У чистій архітектурі система поділяється на такі основні шари:

Доменний шар (Entities) - містить бізнес-логіку та об'єкти предметної області. Цей шар є незалежним від будь-яких зовнішніх ресурсів і може бути використаний повторно в різних контекстах.

Шар випадків використання (Use Cases) - описує специфічні сценарії використання системи та координує потік даних між рівнями. Він визначає, як дані зберігаються, відображаються та змінюються в системі.

Шар інтерфейсів (Interface Adapters) - відповідає за перетворення даних між внутрішніми та зовнішніми шарами. Це можуть бути адаптери для баз даних, веб-сервісів або інтерфейсу користувача.

Шар інфраструктури (Frameworks and Drivers) - містить зовнішні інструменти та фреймворки, такі як UI-фреймворки, бібліотеки доступу до даних та інші сторонні компоненти.

Важливим принципом чистої архітектури є те, що залежності в коді спрямовані лише всередину, до вищих шарів. Жоден код з внутрішнього шару не

повинен знати про код у зовнішніх шарах. Це забезпечує незалежність системи від зовнішніх деталей реалізації, що спрощує підтримку та тестування.

Архітектурний патерн MVVM (Model-View-ViewModel) спрямований на розділення відповідальностей між різними компонентами додатку, що спрощує розробку, тестування та підтримку коду. Він складається з трьох основних компонентів:

Model (Модель): містить дані та бізнес-логіку додатку. Це можуть бути об'єкти, які представляють реальні сутності, а також сервіси для доступу до даних.

View (Подання) - відповідає за відображення інформації користувачу та обробку його взаємодії з додатком. Вона отримує дані від ViewModel та відображає їх у зручному для користувача форматі.

ViewModel (Модель подання) - виступає посередником між Model та View. Вона перетворює дані з моделі у формат, придатний для відображення, та обробляє дії користувача, ініціюючи зміни в моделі.

Патерн MVVM дозволяє забезпечити двосторонній зв'язок між View та ViewModel, що спрощує синхронізацію даних та стану інтерфейсу користувача. Це особливо корисно в додатках з багатим інтерфейсом, де важливо швидко реагувати на зміни даних та дії користувача.

Поєднання чистої архітектури та патерну MVVM в мобільному додатку для дистрибуції новин дозволяє створити стійку, модульну та легко підтримувану структуру. Чиста архітектура забезпечує загальну організацію системи на рівні шарів та залежностей, тоді як MVVM фокусується на взаємодії між UI та бізнес-логікою.

У цьому контексті, модель новин, користувачів та інші бізнес-сутності розміщуються в доменному шарі. Шар випадків використання відповідає за виконання операцій, таких як отримання списку новин, авторизація користувача або збереження обраних статей. Інтерфейси для доступу до даних та взаємодії з API реалізуються в шарі інтерфейсів. Інфраструктурний шар містить реалізації

конкретних деталей, таких як запити до серверу через HTTP або збереження даних у локальній базі даних.

Використання MVVM в цьому підході дозволяє ViewModel взаємодіяти з шаром випадків використання, отримуючи необхідні дані та перетворюючи їх для View. View, в свою чергу, підписується на зміни в ViewModel та відображає дані користувачу. Це забезпечує чітке розділення між логікою відображення та бізнес-логікою, спрощуючи підтримку та тестування додатку.

Застосування цих підходів сприяє підвищенню якості коду та зменшенню складності системи. Вони дозволяють розробникам легше орієнтуватися в коді, швидко знаходити та виправляти помилки, а також додавати нову функціональність без значних змін у вже існуючому коді.

Важливо зазначити, що впровадження чистої архітектури та патерну MVVM вимагає дисципліни та узгодженості в команді розробників. Усі члени команди повинні дотримуватися прийнятих правил та стандартів, щоб зберегти цілісність архітектури. Це може вимагати додаткового навчання та обміну знаннями, але в довгостроковій перспективі такі інвестиції окупаються за рахунок полегшення підтримки та розширення додатку.

У підсумку, поєднання чистої архітектури та патерну MVVM є потужним інструментом для створення якісних мобільних додатків. Воно забезпечує чітку структуру коду, сприяє його масштабованості та полегшує підтримку, що є ключовими факторами успішного розвитку програмного забезпечення в сучасних умовах [19].

# РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

## 4.1. Розробка алгоритму децентралізованого шифрування для забезпечення безпеки авторизації

Безпека даних є критично важливим аспектом у будь-якому мобільному додатку, особливо якщо він працює з чутливою інформацією. У випадку нашого мобільного додатку для управління новинами було прийнято рішення використовувати децентралізований підхід до шифрування та валідації токенів для авторизації користувачів. Це дозволяє мінімізувати кількість запитів до сервера та підвищити продуктивність системи, зберігаючи високий рівень безпеки.

Децентралізований підхід базується на використанні JSON Web Token (JWT) як основного механізму для аутентифікації та авторизації. Основна ідея полягає в тому, що перевірка валідності токена здійснюється на стороні клієнта, що дозволяє зменшити навантаження на сервер і забезпечити більш швидку обробку запитів [21].

Механізм генерації токенів.

На сервері створюється токен, який містить інформацію про користувача (наприклад, унікальний ідентифікатор) та час дії. Для цього використовується бібліотека `jwt`. Створення токена відбувається за допомогою шифрування даних за алгоритмом H256 із використанням секретного ключа.

Процес генерації токена виглядає наступним чином:

```
import jwt
import datetime
```

```
SECRET_KEY = "your-secret-key"

def create_jwt(data, expires_in):
    payload = data.copy()
    payload['exp'] = datetime.datetime.utcnow() + datetime.timedelta(seconds=expires_in)
    return jwt.encode(payload, SECRET_KEY, algorithm='HS256')
```

Тут `payload` містить інформацію, яка буде зашифрована в токен. Наприклад, це може бути `{"user_id": 12345

}`, а `expires_in` вказує тривалість дії токена в секундах.

Процес перевірки токенів.

На сервері реалізовано функцію перевірки валідності токену. Ця функція декодує токен, перевіряє його цілісність та термін дії. Якщо токен недійсний або термін його дії закінчився, сервер повертає відповідну помилку. Код виглядає так:

```
def verify_jwt(token):
    try:
        return jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
    except jwt.ExpiredSignatureError:
        return None  # Токен прострочено
    except jwt.InvalidTokenError:
        return None  # Невалідний токен
```

Цей підхід гарантує, що навіть якщо токен буде викрадено, його використання обмежується встановленим терміном дії.

Валідація токену на клієнтській стороні.

На стороні клієнта додаток перевіряє токен локально перед кожним запитом до сервера. Це дозволяє уникнути зайвих запитів і значно підвищує швидкість роботи. Для цього використовується бібліотека JWT, яка дозволяє розшифрувати токен і перевірити його термін дії. Алгоритм виглядає наступним чином:

```
class VerificationTokenUseCase @Inject constructor() {

    fun tokenVerification(token: String): Boolean {
        return try {
            val jwt = JWT(token)

            val expiresAt = jwt.expiresAt
            if (expiresAt != null && expiresAt.time > System.currentTimeMillis())
{
                true
            } else {
                throw Exception("Token has expired")
            }
        } catch (e: Exception) {
            Log.i("APP_LOG", "Invalid token: ${e.message}")
            false
        }
    }
}
```

Цей код декодує токен і перевіряє його термін дії. Якщо токен є дійсним, додаток дозволяє виконати операцію. Якщо ж ні, користувача перенаправляють на сторінку авторизації для отримання нового токена.

Захист від атаки "повторного відтворення" (Replay Attack).

Для додаткового захисту кожен токен містить інформацію про час створення та термін дії. Якщо користувач спробує повторно використати той самий токен після його закінчення, система це виявить. Це забезпечує захист від несанкціонованого доступу.

Реалізація на сервері обмеження частоти запитів.

На серверній стороні було реалізовано обмеження частоти запитів до ключових ендпоінтів за допомогою бібліотеки `flask_limiter`. Це дозволяє запобігти надмірному використанню ресурсів серверу та захищає від DoS-атак. Наприклад, кожен користувач може виконувати не більше 60 запитів на хвилину:

```
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address

app = Flask(__name__)

limiter = Limiter(
    get_remote_address,  # Визначення користувача за IP
    app=app,
)

@app.route("/api/news", methods=["GET"])
@limiter.limit("60 per minute")
```

```
def news():
    # Обробка запиту
    return {"status": "success", "data": [...]}
```

Децентралізований підхід до шифрування.

Основна ідея децентралізованого підходу полягає в тому, що клієнт самостійно здійснює базову перевірку токена перед кожним запитом. Це дозволяє зменшити кількість запитів до сервера, що знижує навантаження на інфраструктуру та покращує продуктивність. На стороні сервера перевірка виконується лише для ключових операцій, таких як публікація чи видалення новин.

Наприклад, під час операції публікації новини клієнт передає токен разом із даними. Сервер спочатку перевіряє валідність токена за допомогою `verify_jwt`, а потім виконує операцію лише у разі успішної перевірки.

Переваги використання децентралізованого підходу.

1. Підвищення продуктивності: Зменшення кількості запитів до сервера дозволяє знизити навантаження на серверну інфраструктуру та забезпечити швидку взаємодію користувача з додатком.

2. Безпека: Використання JWT із встановленим терміном дії та додатковими механізмами перевірки захищає систему від багатьох атак, зокрема Replay Attack.

3. Гнучкість: Децентралізований підхід дозволяє адаптуватися до різних сценаріїв використання, зберігаючи баланс між продуктивністю та безпекою.

Отже, запроваджений механізм децентралізованого шифрування та авторизації на основі JWT забезпечує високий рівень безпеки, ефективність і масштабованість системи. Реалізовані методи дозволяють не лише захищати додаток від атак, але й оптимізувати його роботу, надаючи користувачам

швидкий і надійний доступ до функціоналу [21].

## 4.2. Реалізація взаємодії класів мобільного додатку з використанням патерну MVVM та діаграма класів

Архітектура мобільного додатку, побудована за принципами Clean Architecture у поєднанні з патерном MVVM (Model-View-ViewModel), спрямована на створення гнучкої, тестованої та масштабованої структури коду. Вибір цієї архітектури обґрунтований необхідністю дотримання принципів SOLID та забезпечення слабкої зв’язаності компонентів, що дозволяє ефективно розділяти відповідальності між різними рівнями додатку.

На верхньому рівні архітектури реалізовано модель `Repository`, яка виступає абстракцією для отримання даних, незалежно від їхнього джерела. У конкретній реалізації використовується інтерфейс `DefaultRepository`, що містить декларацію основних методів для взаємодії з API, таких як завантаження новин (`loadNews`), створення новин (`postNews`) та видалення новин (`deleteNews`). Цей інтерфейс є ключовою частиною архітектури, оскільки дозволяє забезпечити слабку зв’язаність між рівнями бізнес-логіки та джерелом даних.

Реалізація інтерфейсу `DefaultRepository` здійснюється в класі `DefaultRepositoryImpl`. Цей клас використовує ін'єкцію залежностей для отримання доступу до API через об’єкт `DefaultAPI`, що спрощує його тестування та дозволяє замінювати джерела даних без змін у бізнес-логіці. Наприклад, метод `loadNews` реалізований таким чином, що викликає метод `news` з `DefaultAPI`, передаючи параметри, необхідні для запиту. Це дозволяє відокремити логіку роботи з даними від деталей їхньої реалізації.

```
override suspend fun loadNews(page: Int): Response<NewsResponseModel> {
    return defaultAPI.news(page)
}
```

Важливо зазначити, що всі методи класу `DefaultRepositoryImpl` працюють з даними на рівні `Response`, що забезпечує гнучкість у обробці успішних і невдалих запитів, а також можливість відлову винятків на рівні бізнес-логіки. Таким чином, `RepositoryImpl` слугує сполучною ланкою між API та бізнес-логікою, залишаючи всі деталі реалізації на нижчому рівні.

Для забезпечення модульності та відокремлення бізнес-логіки використовується шар `UseCase`. Наприклад, `LoadNewsUseCase` інкапсулює логіку завантаження новин, абстрагуючи її від деталей отримання даних. Завдяки цьому додаток залишається стійким до змін у джерелах даних, оскільки `UseCase` працює виключно з інтерфейсом `DefaultRepository`. У цьому класі реалізовано метод `execute`, що повертає `Flow`, дозволяючи легко інтегрувати асинхронну логіку та обробляти результати в реальному часі.

```
class LoadNewsUseCase @Inject constructor(private val defaultRepository: DefaultRepository) {
    suspend fun execute(page: Int): Flow<Result<Response<NewsResponseModel>>> = flow {
        try {
            val news = defaultRepository.loadNews(page)
            emit(Result.success(news))
        } catch (e: Exception) {
            emit(Result.failure(e))
        }
    }
}
```

У цьому прикладі видно, як `UseCase` обробляє винятки, щоб мінімізувати вплив помилок API на роботу додатку. Він також формує результати у вигляді об’єкта `Result`, що спрощує обробку результатів на рівні ViewModel.

ViewModel слугує посередником між шаром бізнес-логіки та інтерфейсом користувача. У випадку `LoadViewModel` реалізовано механізм роботи з новинами, який використовує `LoadNewsUseCase` для завантаження даних та зберігає їх у `StateFlow`. Завдяки цьому підхід MVVM дозволяє легко інтегрувати реактивну архітектуру в додаток, забезпечуючи автоматичне оновлення UI при зміні стану даних.

```
viewModelScope.launch(Dispatchers.IO) {
        _isLoading.value = true
        loadNewsUseCase.execute(currentPage).collect { result ->
            result.onSuccess {
                // Обробка успішного результату
            }.onFailure {
                // Обробка помилки
            }
            _isLoading.value = false
        }
    }
```

У цьому прикладі використовується `viewModelScope` для запуску корутин, що гарантує автоматичне завершення завдань при знищенні ViewModel. Таким чином, ViewModel зберігає дані для UI в реактивному вигляді, дозволяючи розробникам концентруватися на реалізації бізнес-логіки, не турбуючись про стан UI.

Однією з ключових переваг використання такої структури є її відповідність

принципам SOLID. Наприклад, принцип єдиної відповідальності реалізовано через чітке розділення відповідальності між рівнями додатку: Repository відповідає за джерело даних, UseCase — за бізнес-логіку, а ViewModel — за інтеграцію даних з UI. Завдяки цьому додаток залишається легким у підтримці та масштабуванні.

Використання патерну MVVM разом із Clean Architecture також забезпечує легкість у тестуванні. Оскільки ViewModel працює виключно з UseCase, його можна легко протестувати, замінивши UseCase на мок-об’єкт. Аналогічно, шар бізнес-логіки легко перевіряється окремо від UI, що спрощує виявлення та виправлення помилок.

Гнучкість і масштабованість цієї архітектури також забезпечуються завдяки слабкій зв’язаності компонентів. Наприклад, заміна джерела даних (API) не вплине на бізнес-логіку, оскільки весь функціонал інкапсульований у шарі Repository. Це дозволяє адаптувати додаток до нових вимог без ризику порушення роботи інших компонентів.

Узагальнюючи, структура класів мобільного додатку, побудована за принципами Clean Architecture та MVVM, забезпечує чіткий поділ відповідальності між рівнями додатку, гнучкість у налаштуванні, тестованість і масштабованість. Вона ідеально підходить для сучасної розробки, де критично важливо забезпечити стабільність коду, зручність у підтримці та можливість інтеграції нових функцій без значних змін у базовій архітектурі [20].

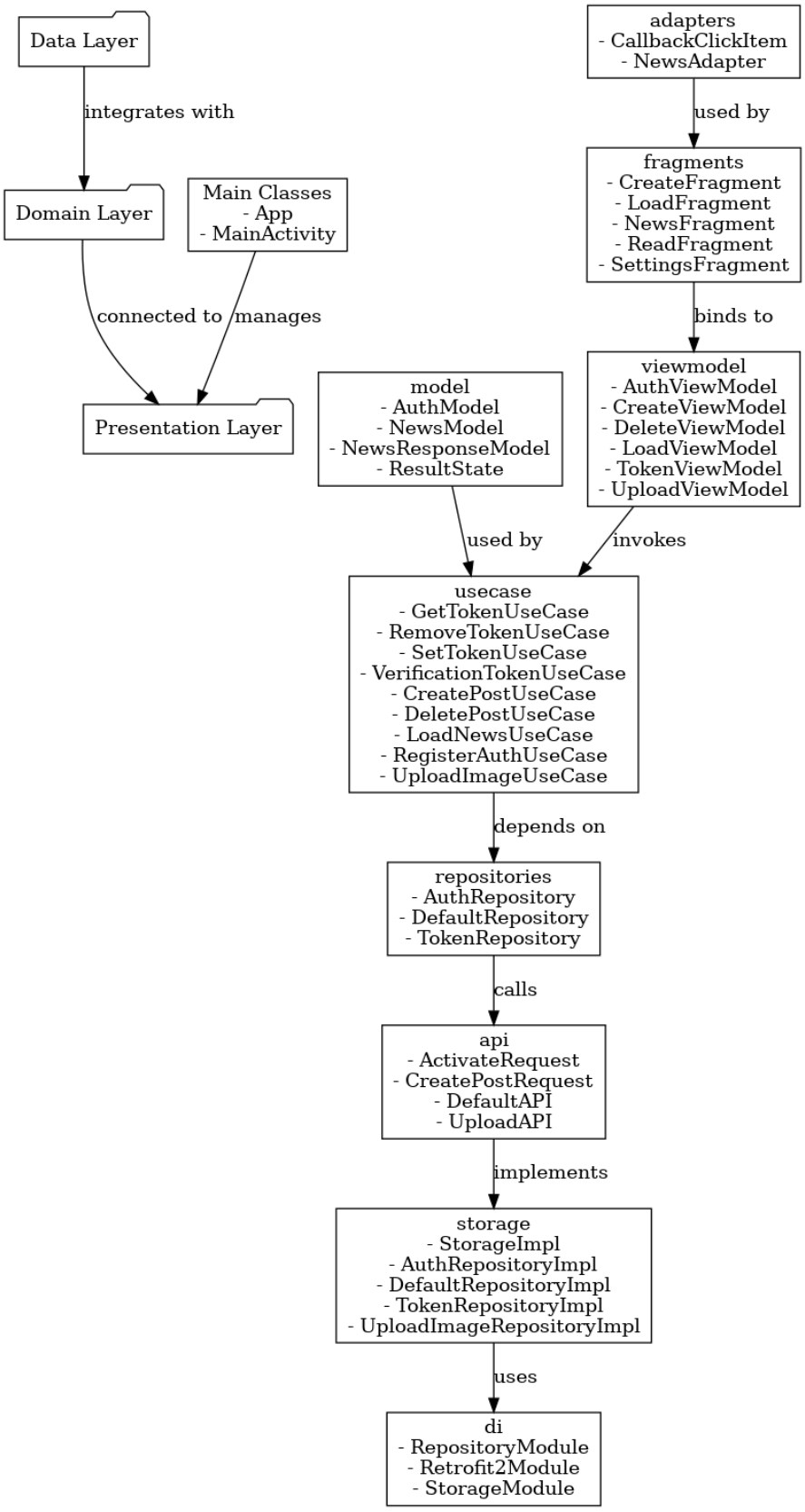


Рис. 4.1 – Діаграма класів

### 4.3. Опис та реалізація інтерфейсу користувача і функціональних можливостей мобільного додатку

На рисунку 4.2 зображено головний екран мобільного додатку, який виконує функцію агрегатора новин. Інтерфейс чітко організований, з акцентом на інформативність та доступність. Основним елементом цього екрану є перелік новин, який представлений у вигляді вертикального списку карток. Кожна картка містить зображення, заголовок новини, короткий текстовий опис, час прочитання та кнопку "Read". Така структура дозволяє користувачеві швидко оцінити релевантність новини та перейти до її детального перегляду [21].


Рис. 4.2 Головний екран додатку

Верхня частина екрану містить панель, де розташовано заголовок "Hot

news about Internet Technologies", а також іконка налаштувань у правому куті. Ця панель служить орієнтиром для користувача, вказуючи на тематику новинного контенту. Іконка налаштувань дозволяє перейти до персоналізації додатку, зокрема вибору категорій новин або управління акаунтом [21].

На рисунку 4.3 показано екран детального перегляду новини. Цей екран активується після натискання кнопки "Read" на головному екрані. Він містить великий заголовок новини, основний текст, а також зображення, яке служить візуальним доповненням до інформації. У верхній частині присутня кнопка "Назад", яка дозволяє користувачеві повернутися до списку новин. Цей екран створений для забезпечення зручного читання тексту завдяки великим шрифтам і продуманому розташуванню елементів [21].


Рис 4.3 Екран детального перегляду новини

На рисунку 4.4 представлено екран налаштувань, який включає функцію активації адміністративного доступу. У нижній частині екрана розташоване вікно введення UUID-токена та кнопка "Activate". Цей функціонал дозволяє обмеженому колу користувачів (адміністраторів) отримати доступ до розширених можливостей додатку, таких як модерація новин чи управління контентом. Для зручності взаємодії з цим екраном реалізовано плавний перехід від стану звичайного користувача до адміністративного режиму.

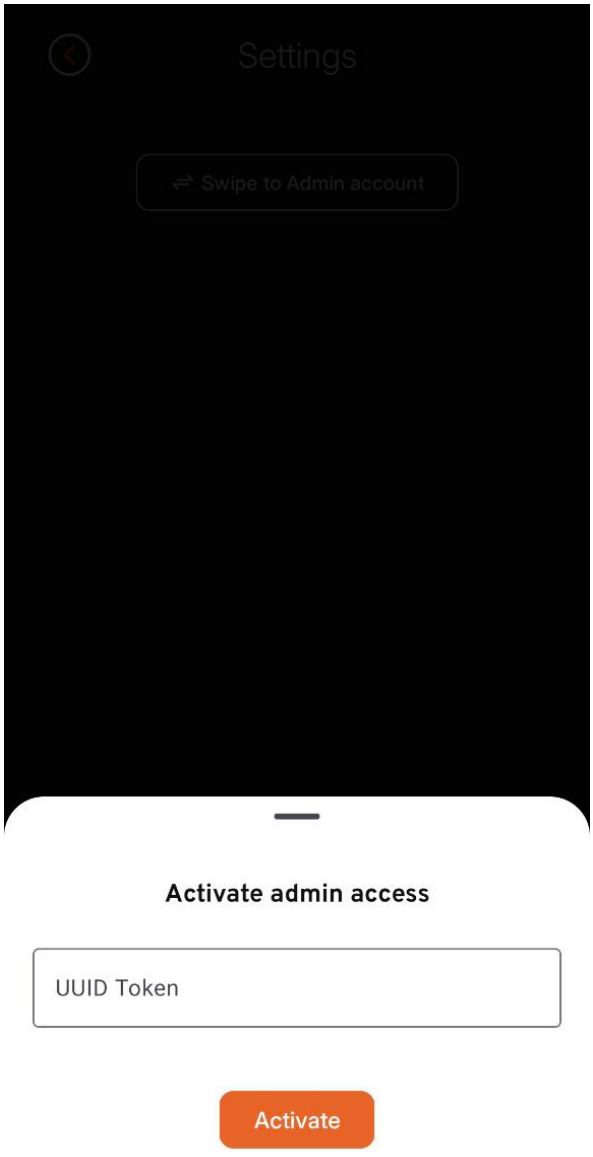


Рис 4.4 Екран налаштувань

Загальна логіка додатку забезпечує легкий перехід між екранами. Наприклад, користувач може перейти з головного екрану до перегляду новини або до налаштувань, а з налаштувань повернутися до основного функціоналу. Така організація дозволяє оптимізувати користувацький досвід і забезпечити логічну послідовність у виконанні задач.

Можливими варіантами розвитку функціоналу є впровадження персоналізованих рекомендацій новин на основі інтересів користувача, додавання темного режиму для комфортного використання в нічний час, а також інтеграція функції коментування чи оцінювання контенту. Це сприятиме підвищенню залученості користувачів і створенню спільноти навколо додатку.

Після переходу в режим адміністратора мобільний додаток надає розширений функціонал, що дозволяє управляти контентом. На екрані, де представлено інтерфейс зі списком новин. Додатково з'являється кнопка "Add News" (рисунок 4.5), яка дає змогу додавати нові записи до стрічки новин. Інші елементи залишаються незмінними, включаючи перегляд новин та функціонал для користувачів [21].


Рис 4.5 Оновлений екран з кнопкою “Add news” для адміністратора

З’являється новий екран, який демонструє форму для створення нового посту (рисунок 4.6). Інтерфейс забезпечує кілька текстових полів для введення

заголовка новини, опису та кнопки для завантаження зображення. Основна кнопка "Public" дозволяє підтвердити створення посту та відправити його на сервер. При цьому всі поля є обов’язковими для заповнення. У разі спроби публікації з порожніми полями з'являються повідомлення про помилки (рисунок 4.7).

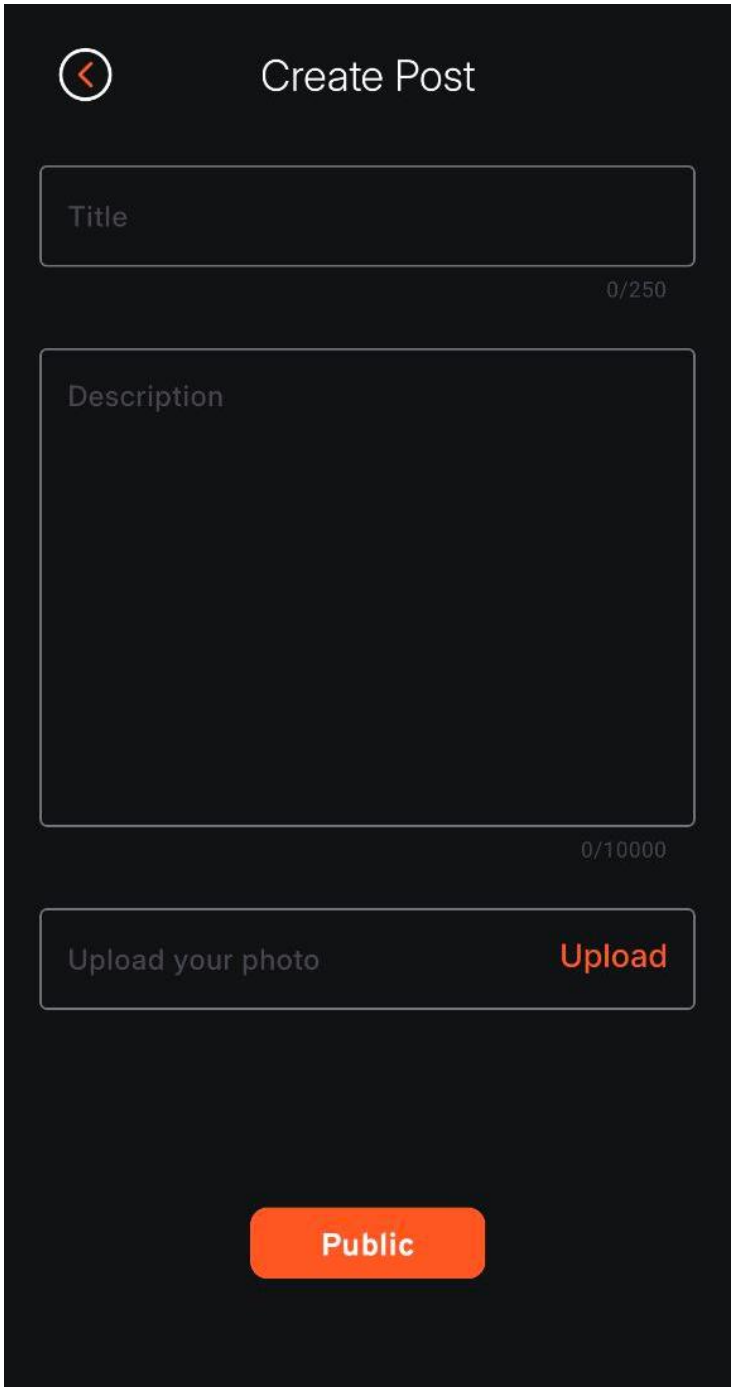


Рис 4.6 Екран форма для створення нового посту

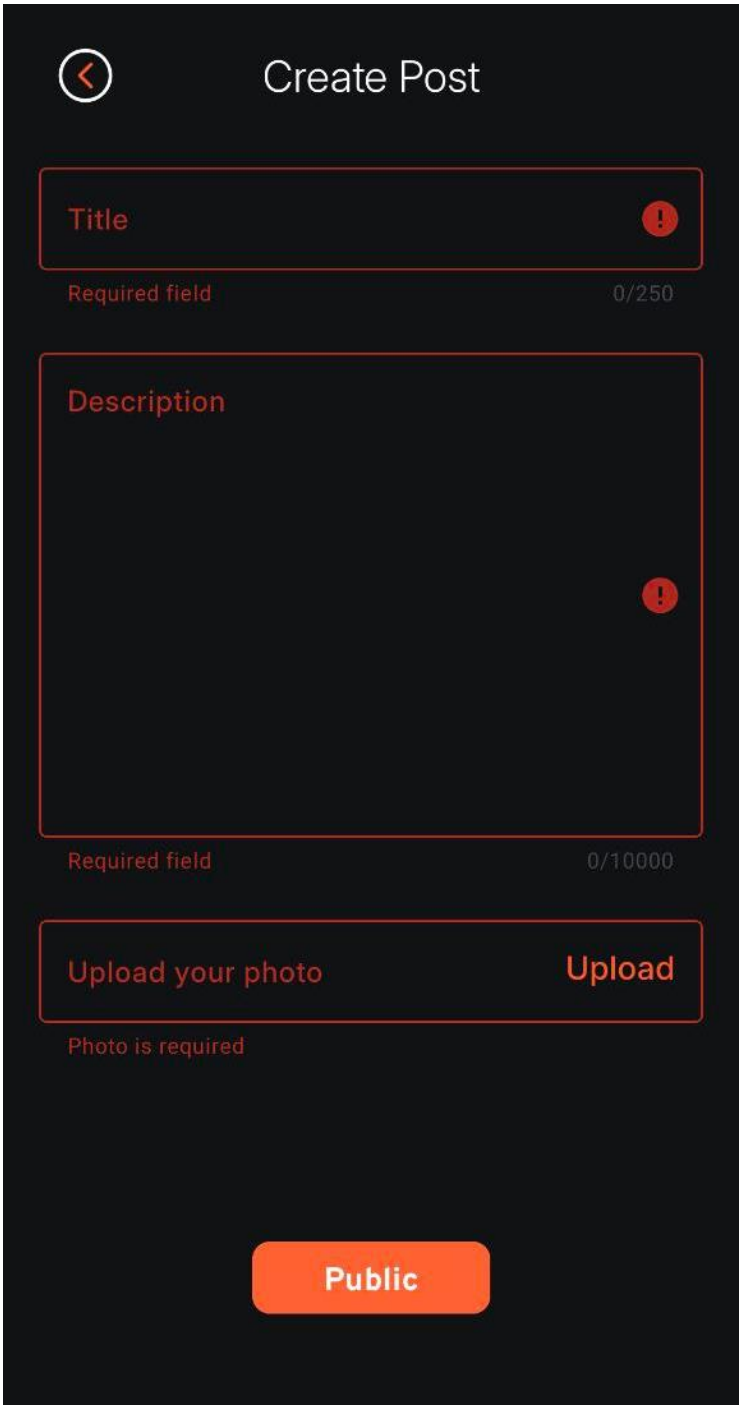


Рис 4.7 Приклад екрану створення публікації, після невдалої валідації

Екран адміністратора надає функціонал для видалення новин. У верхньому правому куті знаходиться кнопка "Delete News" (рисунок 4.8), яка дозволяє видалити обраний пост. Під основною картинкою новини наведено детальний текст із заголовком, що спрощує адміністрування контенту.


Рис 4.8 Кнопка для видалиння новини “Delete News” з панелі адміністратора

На екрані зображено інтерфейс налаштувань у режимі адміністратора (рисунок 4.9), який дає змогу повернутися до звичайного користувацького режиму. Цей екран підтримує зміну ролі адміністратора на роль звичайного користувача через підтвердження відповідної дії

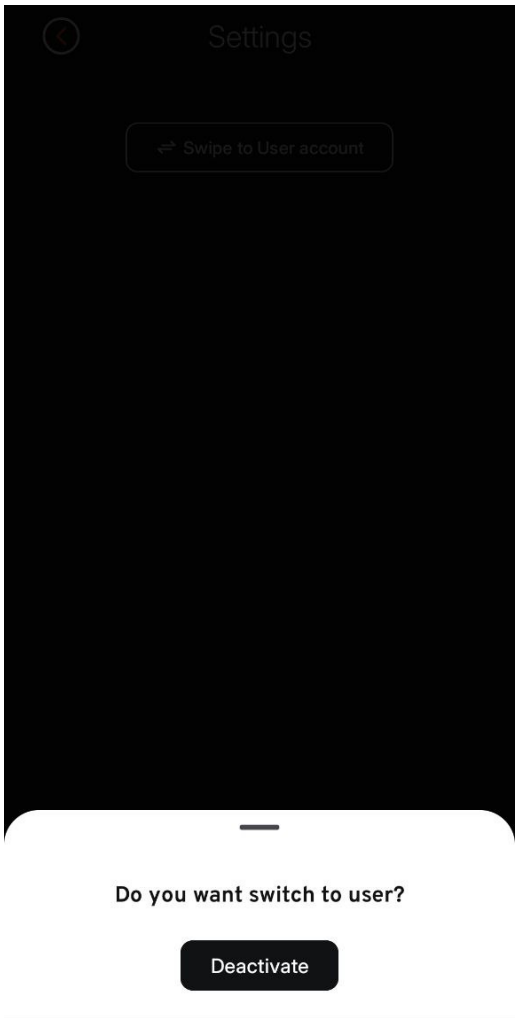


Рис. 4.9 Екран налаштувань у режимі адміністратора.

# ВИСНОВКИ

Розробка мобільного додатку та API у межах даної дипломної роботи відповідає сучасним потребам ринку інформаційних технологій, де автоматизація, зручність використання та персоналізований підхід є ключовими факторами успішних програмних продуктів. Запропонований додаток дозволяє інтегрувати сучасні підходи до управління інформацією, зокрема організацію новинного контенту, створення та редагування записів, а також підтримку адміністративного доступу. Такий функціонал сприяє зростанню ефективності взаємодії між користувачами та платформою, забезпечуючи широкий спектр можливостей для роботи з контентом.

У процесі виконання дипломної роботи було досягнуто усіх поставлених цілей і виконано низку завдань. На початкових етапах проєкту проведено аналіз ринку існуючих рішень для організації інформаційних платформ, виявлено їхні переваги та недоліки, що стало основою для визначення унікальних вимог до функціональності розроблюваного додатку. Наступним важливим етапом було обґрунтування вибору технологій, серед яких MVVM, Clean Architecture, DI (Hilt) та інші сучасні інструменти, які забезпечили модульність, масштабованість та тестованість системи. Було розроблено архітектуру додатку, створено модулі API, зокрема для авторизації, обробки даних та управління контентом. У фінальній стадії реалізації виконано програмування ключових функціональних можливостей, розроблено інтерфейс користувача та проведено тестування системи для забезпечення її стабільної роботи.

Використання сучасних технологій, таких як MVVM, дозволило чітко розділити бізнес-логіку, дані та презентаційний рівень, що значно спростило масштабування та підтримку додатку. Clean Architecture забезпечила слабку зв’язаність між компонентами, що зробило систему більш гнучкою для внесення змін або додавання нових функцій. Інтеграція DI (Hilt) сприяла автоматизації

управління залежностями, підвищуючи зручність написання та підтримки коду. Такі підходи забезпечують високу якість коду, знижують ризик появи помилок та дозволяють адаптувати систему до змін у вимогах чи умовах використання.

Розробка відповідає сучасним вимогам щодо адаптивності, оскільки система підтримує обидва режими використання: для звичайних користувачів та адміністраторів. Масштабованість архітектури дозволяє інтегрувати нові модулі або змінювати існуючі функції без необхідності суттєвих змін у базовій логіці. Зручність роботи для кінцевого користувача забезпечується продуманим інтерфейсом, оптимізованим для швидкого доступу до функцій додатку, та можливістю гнучкого управління інформацією.

Отже, результати дипломної роботи стали не лише завершеним програмним продуктом, а й основою для подальшого розвитку системи. Виявлені переваги використання сучасних технологій та архітектурних підходів дозволяють стверджувати, що розроблений додаток може бути вдосконалений у майбутньому шляхом інтеграції нових функцій, адаптації до інших ринкових потреб або розширення його можливостей для використання в суміжних галузях.

За результатами виконання дипломної роботи було опубліковано статтю в фаховому журналі.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android Developers. Dependency Injection with Hilt. Режим доступу URL: https://developer.android.com/training/dependency-injection/hilt – Назва з екрану.

2. Clean Architecture. A detailed introduction by Robert C. Martin. Режим доступу URL: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html – Назва з екрану.

3. Мікросервісна архітектура. Автор: Орхан Гасімов, Technology Director, GlobalLogic. Режим доступу URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/ – Назва з екрану.

4. Retrofit: Type-Safe HTTP Client for Android and Java by Square, Inc. Режим доступу URL: https://square.github.io/retrofit/ – Назва з екрану.

5. Android MVVM Architecture: Understanding ViewModel and LiveData. Режим доступу URL: https://developer.android.com/topic/libraries/architecture/viewmodel – Назва з екрану.

6. Python Flask Documentation. Режим доступу URL: https://flask.palletsprojects.com/en/2.2.x/ – Назва з екрану.

7. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. – Полтава: ПУЕТ, 2022. – 67 с. Режим доступу URL: http://dspace.puet.edu.ua/handle/123456789/12851 – Назва з екрану.

8. Waterfall Model for Software Development and Testing Teams. Режим доступу URL: https://www.geeksforgeeks.org/waterfall-model/ – Назва з екрану.

9. Retrofit2 Javadoc. Режим доступу URL: https://square.github.io/retrofit/2.x/retrofit/ – Назва з екрану.

10. Material Design Guidelines. Режим доступу URL: https://material.io/design – Назва з екрану.

11. Налаштовуємо кастомну JWT-авторизацію. Режим доступу URL: https://dou.ua/forums/topic/50164/ – Назва з екрану.

12. Flask-Limiter: Rate Limiting for Flask Applications. Режим доступу URL: https://flask-limiter.readthedocs.io/en/stable/ – Назва з екрану.

13. Кращі тренди розробки мобільних додатків у 2024 році. Режим доступу URL: https://whileweb.com/uk/blog/top-trendiv-rozrobki-mobilnih-dodatkiv-u-2024-roci/ – Назва з екрану.

14. Pymysql Tutorial – Complete Guide. Режим доступу URL: https://gamedevacademy.org/pymysql-tutorial-complete-guide/ – Назва з екрану.

15. Picasso Image Loading Library for Android. Режим доступу URL: https://square.github.io/picasso/ – Назва з екрану.

16. Основи тестування REST API з Python Flask. Режим доступу URL: https://realpython.com/flask-api/ – Назва з екрану.

17. Datastore Preferences for Android. Режим доступу URL: https://developer.android.com/topic/libraries/architecture/datastore – Назва з екрану.

18. Найкращі практики проєктування REST API. Режим доступу URL: https://devzone.org.ua/post/naykrashchi-praktyky-proyektuvannia-rest-api – Назва з екрану.

19. Model-View-ViewModel (MVVM). Режим доступу URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm – Назва з екрану.

20. UseCase Red Flags and Best Practices in Clean Architecture. Режим доступу URL: https://engineering.teknasyon.com/usecase-red-flags-and-best-practices-in-clean-architecture-76e2f6d921eb – Назва з екрану.

21. Кошова О., Ольховська О., Оборожний А., Жуля А. Розробка мобільного застосунку для управління та дистрибуції новин на основі мікросервісної архітектури. Полтава: ПУЕТ, 2025. – 168. – Режим доступу URL: https://visnikkrnu.kdu.edu.ua/statti/2025_1_160.pdf . – Назва з екрану.

# **ДОДАТОК** А
## Код програми API

main.py

```
from flask import Flask, request, jsonify
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address

from AuthService import AuthService
from data.DefaultTransaction import DefaultTransaction

app = Flask(__name__)

repository = DefaultTransaction()

limiter = Limiter(
    get_remote_address,  # Визначає клієнта за IP
    app=app,
)


@app.errorhandler(429)
def ratelimit_error(e):
    return "Too many requests for 1 minute. Please try again later", 429


@app.before_request
def auth():
    if request.endpoint in ['news', 'activate']:
        return

    auth_header = request.headers.get("Authorization")
    if not auth_header:
        return "Authorization header is missing", 401

    if not AuthService.verify_jwt(auth_header):
        return "Authorization Token is not correct", 401


@app.route("/api/news", methods=["GET"])
@limiter.limit("60 per minute")
def news():
    page = int(request.args.get('page', 1))
    offset = (page - 1) * 5
    return repository.news(offset)


@app.route("/api/post", methods=["POST"])
@limiter.limit("1 per minute")
def post():
    data = request.get_json()
```

```
    if not data.get("title", None) or not data.get("desc", None) or not data.get("img_url", None):
        return "Lose some params of (`title`, `desc` or `img_url`)", 400

    return repository.post(data['title'], data['desc'], data['img_url'])


@app.route("/api/delete", methods=["DELETE"])
@limiter.limit("1 per minute")
def delete():
    identify = request.args.get('identify', None)

    if not identify:
        return "Lose param (`identify`)", 400

    return repository.remove(identify)


@app.route("/api/activate", methods=["POST"])
@limiter.limit("1 per minute")
def activate():
    data = request.get_json()

    if not data.get('uuid', None) or not data.get('android_id', None):
        return "Lose some params of (`uuid` or `android_id`)", 400

    return repository.register_auth(data['uuid'], data['android_id'])
```

AuthService.py

```
import datetime
import hashlib

import jwt

from config import SECRET_KEY


class AuthService:

    @staticmethod
    def create_jwt(data, expires_in):
        payload = data.copy()
        payload['exp'] = datetime.datetime.utcnow() + datetime.timedelta(seconds=expires_in)
        return jwt.encode(payload, SECRET_KEY, algorithm='HS256')

    @staticmethod
    def verify_jwt(token):
        try:
            return jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
        except jwt.ExpiredSignatureError:
```

```
            return None
        except jwt.InvalidTokenError:
            return None
```

DefaultDataBase.py

```
from asyncio.log import logger

import pymysql
import pymysql.cursors

from config import DATABASE_PASSWORD, DATABASE_NAME


class DefaultDataBase:

    def __init__(self):
        self._connect()

    def _connect(self):
        """Створює нове з'єднання."""
        self._connection = pymysql.connect(
            host="localhost",
            user="root",
            password=DATABASE_PASSWORD,
            db=DATABASE_NAME,
            charset="utf8mb4",
            cursorclass=pymysql.cursors.DictCursor,
            autocommit=False
        )

    def _insert(self, query, args=None):
        try:
            with self._connection.cursor() as cursor:
                result = cursor.execute(query, args)
                return result
        except Exception as e:
            logger.error(f"_insert: {e}")
            self._connection.rollback()

    def _update(self, query, args=None):
        try:
            with self._connection.cursor() as cursor:
                result = cursor.execute(query, args)
                return result
        except Exception as e:
            logger.error(f"_update: {e}")
            self._connection.rollback()

    def _delete(self, query, args=None):
        try:
            with self._connection.cursor() as cursor:
```

```
            result = cursor.execute(query, args)
            return result
        except Exception as e:
            logger.error(f"_delete: {e}")
            self._connection.rollback()

    def _select_one(self, query, args=None):
        try:
            with self._connection.cursor() as cursor:
                cursor.execute(query, args)
                return cursor.fetchone()
        except Exception as e:
            logger.error(f"_select_one: {e}")

    def _select(self, query, args=None):
        try:
            with self._connection.cursor() as cursor:
                cursor.execute(query, args)
                return cursor.fetchall()
        except Exception as e:
            logger.error(f"_select_all: {e}")

    def _commit(self):
        """Коміт транзакції."""
        try:
            self._connection.commit()
        except Exception as e:
            logger.error(f"commit: {e}")
            self._connection.rollback()

    def _rollback(self):
        """Відкат транзакції."""
        try:
            self._connection.rollback()
        except Exception as e:
            logger.error(f"rollback: {e}")

    def _ensure_connection(self):
        """Перевіряє, чи з'єднання активне, і створює нове, якщо потрібно."""
        if not self._connection or not self._connection.open:
            self._connect()

    def _close(self):
        """Закриває з'єднання з базою даних."""
        try:
            if self._connection and self._connection.open:
                self._connection.close()
        except Exception as e:
            logger.error(f"close: {e}")
```

DefaultRepository.py

```
from data.DefaultDataBase import DefaultDataBase


class DefaultRepository(DefaultDataBase):

    def _news(self, offset):
        query = "SELECT * FROM `news` ORDER BY `created` DESC LIMIT 5 OFFSET %s;"
        return self._select(query, (offset, ))

    def _post(self, title, desc, img_data):
        query = "INSERT INTO `news` (`title`, `desc`, `img_url`) VALUES (%s, %s, %s);"
        return self._insert(query, (title, desc, img_data))

    def _remove(self, identify):
        query = "DELETE FROM `news` WHERE `id` = %s;"
        return self._delete(query, (identify,))

    def _set_admin(self, uuid, android_id):
        query = "UPDATE `keys` SET `android_id` = %s, `activated` = NOW() WHERE `uuid` = %s;"
        return self._update(query, (android_id, uuid))

    def _is_admin(self, uuid, android_id):
        query = "SELECT * FROM `keys` WHERE `uuid` = %s AND `android_id` = %s;"
        return self._select_one(query, (uuid, android_id))
```

DefaultTransaction.py

```
from asyncio.log import logger
from datetime import datetime

from flask import jsonify

from AuthService import AuthService
from data.DefaultRepository import DefaultRepository


class DefaultTransaction(DefaultRepository):

    def news(self, offset):
        """
        Робить запит з метою витягування всіх постів
        """
        try:
            # Починаємо транзакцію
            self._ensure_connection()
            self._connection.begin()

            news_response = self._news(offset)

            if not news_response:
                raise Exception("Error: request database is null")
```

```
            # Конвертуємо нестандартні типи (наприклад, bytes)
            for item in news_response:
                if isinstance(item.get('img_data'), bytes):
                    item['img_data'] = item['img_data'].decode('utf-8')  # Декодуємо у текст
                if isinstance(item.get('created'), datetime):
                    item['created'] = item['created'].isoformat()  # Перетворюємо дату у ISO-формат

            # Повертаємо результат
            return jsonify({"result": True, "news": news_response}), 200

        except Exception as e:
            logger.error(f"Transaction failed: {e}")
            return str(e), 400
        finally:
            self._close()

    def register_auth(self, uuid, android_id):
        """
        Реєструє або перевіряє адміна на валідність
        """
        try:
            # Починаємо транзакцію
            self._ensure_connection()
            self._connection.begin()

            # need authorization weekly
            new_token = AuthService.create_jwt(data={"uuid": uuid, "android_id": android_id},
expires_in=3600 * 24 * 7)

            if not self._is_admin(uuid, android_id) and not self._set_admin(uuid, android_id):
                raise Exception("Error: registration fail, auth key is not correct")

            self._commit()

            return jsonify({"result": True, "Token": new_token}), 200

        except Exception as e:
            logger.error(f"Transaction failed: {e}")
            return str(e), 400
        finally:
            self._close()

    def post(self, title, desc, img_url):
        """
        Додає новину до бази
        """
        try:
            # Починаємо транзакцію
            self._ensure_connection()
            self._connection.begin()
```

```
            if not self._post(title, desc, img_url):
                raise Exception("Error: database can`t add this")

            self._commit()

            return jsonify({"result": True, "message": "post success"}), 200

        except Exception as e:
            logger.error(f"Transaction failed: {e}")
            return str(e), 400
        finally:
            self._close()

    def remove(self, identify):
        """
        Видаляє новину з бази по id
        """
        try:
            # Починаємо транзакцію
            self._ensure_connection()
            self._connection.begin()

            if not self._remove(identify):
                raise Exception("Error: database can`t delete this")

            self._commit()

            return jsonify({"result": True, "message": "delete success"}), 200

        except Exception as e:
            logger.error(f"Transaction failed: {e}")
            return str(e), 400
        finally:
            self._close()
```

## Код програми Мобільний додаток

### DefaultRepository

```
package com.artemqq5.flutist.domain

import com.artemqq5.flutist.data.api.model.CreatePostRequest
import com.artemqq5.flutist.domain.model.defaultAPI.NewsResponseModel
import okhttp3.ResponseBody
import retrofit2.Response

interface DefaultRepository {

    suspend fun loadNews(page: Int): Response<NewsResponseModel>
    suspend fun postNews(createPostRequest: CreatePostRequest, token: String): Response<ResponseBody>
    suspend fun deleteNews(identify: Int, token: String): Response<ResponseBody>
```

```
}
```

DefaultRepositoryImpl

```
package com.artemqq5.flutist.data

import com.artemqq5.flutist.data.api.DefaultAPI
import com.artemqq5.flutist.data.api.model.CreatePostRequest
import com.artemqq5.flutist.domain.DefaultRepository
import com.artemqq5.flutist.domain.model.defaultAPI.NewsResponseModel
import jakarta.inject.Inject
import okhttp3.ResponseBody
import retrofit2.Response

class DefaultRepositoryImpl @Inject constructor(private val defaultAPI: DefaultAPI): DefaultRepository {
    override suspend fun loadNews(page: Int): Response<NewsResponseModel> {
        return defaultAPI.news(page)
    }

    override suspend fun postNews(
        createPostRequest: CreatePostRequest,
        token: String
    ): Response<ResponseBody> {
        return defaultAPI.createPost(token, createPostRequest)
    }

    override suspend fun deleteNews(
        identify: Int,
        token: String
    ): Response<ResponseBody> {
        return defaultAPI.deletePost(token, identify)
    }

}
```

LoadNewsUseCase

```
package com.artemqq5.flutist.domain.usecase

import com.artemqq5.flutist.domain.DefaultRepository
import com.artemqq5.flutist.domain.model.defaultAPI.NewsResponseModel
import jakarta.inject.Inject
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow
import retrofit2.Response

class LoadNewsUseCase @Inject constructor(private val defaultRepository: DefaultRepository) {

    suspend fun execute(page: Int): Flow<Result<Response<NewsResponseModel>>> = flow {
        try {
            val news = defaultRepository.loadNews(page)
```

```
            emit(Result.success(news))
        } catch (e: Exception) {
            emit(Result.failure(e))
        }
    }
}
```

LoadViewModel

```
package com.artemqq5.flutist.presentation.viewmodel

import android.util.Log
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.artemqq5.flutist.App
import com.artemqq5.flutist.App.Companion.MAIN_TAG
import com.artemqq5.flutist.domain.model.defaultAPI.NewsModel
import com.artemqq5.flutist.domain.model.defaultAPI.NewsResponseModel
import com.artemqq5.flutist.domain.usecase.LoadNewsUseCase
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch
import retrofit2.Response
import javax.inject.Inject

@HiltViewModel
class LoadViewModel @Inject constructor(private val loadNewsUseCase: LoadNewsUseCase) :
ViewModel() {

    private val _news = MutableStateFlow<List<NewsModel>>(emptyList())
    val news: StateFlow<List<NewsModel>> = _news

    private val _isLoading = MutableStateFlow(false)
    val isLoading: StateFlow<Boolean> = _isLoading

    private val _isRefreshing = MutableStateFlow(false)
    val isRefreshing: StateFlow<Boolean> = _isRefreshing

    private var currentPage = 1

    init {
        if (currentPage == 1) {
            fetchNews(true)
        }
    }

    fun fetchNews(isRefresh: Boolean = false) {
        if (_isLoading.value) return

        if (isRefresh) {
```

```
            currentPage = 1
            _news.value = emptyList()
            _isRefreshing.value = true
        }

        viewModelScope.launch(Dispatchers.IO) {
            _isLoading.value = true
            loadNewsUseCase.execute(currentPage).collect { result ->
                result.onSuccess {
                    if (!it.isSuccessful) {
                        Log.i(MAIN_TAG, "Load news not success: ${it.errorBody()?.string()}")
                        return@onSuccess
                    }

                    it.body()?.let { response ->
                        if (response.news.isNotEmpty()) {
                            val updatedNews = if (isRefresh) {
                                response.news
                            } else {
                                _news.value + response.news
                            }

                            _news.value = updatedNews.distinctBy { news -> news.id }.sortedByDescending {
news -> news.id }
                            currentPage++
                        }
                    }
                }.onFailure {
                    Log.i(MAIN_TAG, "Load news failure: $it")
                }

                _isLoading.value = false
                _isRefreshing.value = false
            }
        }
    }

}
```

NewsFragment

```
package com.artemqq5.flutist.presentation.fragments

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.core.os.bundleOf
import androidx.fragment.app.Fragment
import androidx.fragment.app.activityViewModels
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
```

```
import androidx.navigation.fragment.findNavController
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.artemqq5.flutist.R
import com.artemqq5.flutist.databinding.FragmentNewsBinding
import com.artemqq5.flutist.domain.model.defaultAPI.NewsModel
import com.artemqq5.flutist.domain.usecase.token.VerificationTokenUseCase
import com.artemqq5.flutist.presentation.adapters.CallbackClickItem
import com.artemqq5.flutist.presentation.adapters.NewsAdapter
import com.artemqq5.flutist.presentation.viewmodel.LoadViewModel
import com.artemqq5.flutist.presentation.viewmodel.TokenViewModel
import com.google.android.material.snackbar.Snackbar
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch
import javax.inject.Inject

@AndroidEntryPoint
class NewsFragment : Fragment(), CallbackClickItem {

    private lateinit var binding: FragmentNewsBinding

    private val loadViewModel: LoadViewModel by activityViewModels()
    private val tokenViewModel: TokenViewModel by viewModels()

    @Inject
    lateinit var verificationTokenUseCase: VerificationTokenUseCase

    // Lazy initialization of the adapter to ensure it's created only when needed
    private val newsAdapter by lazy {
        NewsAdapter(this)
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentNewsBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.recyclerNews.adapter = newsAdapter

        tokenViewModel.getTokenValidate { validate, _ ->
            validate?.let {
                if (validate) {
                    binding.addNews.visibility = View.VISIBLE
                } else {
```

```
                Snackbar.make(
                    binding.root,
                    getString(R.string.token_had_been_overdue),
                    Snackbar.LENGTH_LONG
                )
                    .show()
            }
        }
    }

    // Adds a scroll listener to trigger pagination when the user scrolls to the end
    binding.recyclerNews.addOnScrollListener(object : RecyclerView.OnScrollListener() {
        override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
            super.onScrolled(recyclerView, dx, dy)
            val layoutManager = recyclerView.layoutManager as LinearLayoutManager
            if (layoutManager.findLastCompletelyVisibleItemPosition() == newsAdapter.itemCount -
1) {
                loadViewModel.fetchNews()
            }
        }
    })

    binding.swipeRefresh.setOnRefreshListener {
        loadViewModel.fetchNews(true)
    }

    viewLifecycleOwner.lifecycleScope.launch {
        loadViewModel.news.collect { newsList ->
            newsAdapter.submitList(newsList)
        }
    }

    viewLifecycleOwner.lifecycleScope.launch {
        loadViewModel.isRefreshing.collect { isRefreshing ->
            binding.swipeRefresh.isRefreshing = isRefreshing
        }
    }

    viewLifecycleOwner.lifecycleScope.launch {
        loadViewModel.isLoading.collect { isLoading ->
            binding.progressBar.visibility =
                if (isLoading && !binding.swipeRefresh.isRefreshing) View.VISIBLE else
View.GONE
        }
    }

    binding.settings.setOnClickListener {
        findNavController().navigate(R.id.action_newsFragment_to_settingsFragment)
    }

    binding.addNews.setOnClickListener {
```

```
            findNavController().navigate(R.id.action_newsFragment_to_createFragment)
        }

    }

    override fun clickItem(newsModel: NewsModel) {
        if (findNavController().currentDestination?.id == R.id.newsFragment) {
            findNavController().navigate(
                R.id.action_newsFragment_to_readFragment,
                bundleOf("news" to newsModel)
            )
        }
    }

}
```