МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційне і програмне забезпечення мобільної соціальної мережі»

здобувача групи ІН – 94-1 Тимчишина Максима Івановича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Максим ТИМЧИШИН
(підпис)

Керівник,
кандидат фізико-математичних наук,
старший викладач кафедри
комп'ютерних наук Дмитро ВЕЛИКОДНИЙ _____
(підпис)

Суми – 2023

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»

здобувача групи ІН – 94-1 Тимчишина Максима Івановича.

1. Тема роботи: «Інформаційне і програмне забезпечення мобільної соціальної мережі»

затверджую наказом по СумДУ від «01» червня 2023 р. №0475-VI

2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*

3. Вхідні дані до кваліфікаційної роботи_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1)Інформаційно-аналітичний огляд соціальних мереж. 2)Створення серверної частини мобільної соціальної мережі. 3)Створення графічного інтерфейсу для взаємодії з системою.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій для мобільної та серверної розробки*		
3	*Створення серверної та UI частини соціальної мережі*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: стор. 53, рис. 15, табл. 0, додаток 1, джерел 14.

Обґрунтування актуальності теми роботи – соціальні мережі на сьогодні стають одною з ключових складових нашого сучасного суспільства. З кожним роком популярність цих мереж зростає, відповідно, зростає і потреба в надійному і ефективному інформаційному та програмному забезпеченні мобільних соціальних мереж.

Об'єкт дослідження – інформаційне та програмне забезпечення мобільної соціальної мережі.

Мета роботи – розробка та вдосконалення інформаційного та програмного забезпечення мобільної соціальної мережі з метою забезпечення надійності, безпеки, ефективності та поліпшення користувацького досвіду.

Методи дослідження – моделювання логіки за допомогою технології Firebase, розробка інтерфейсу для взаємодії з системою в Android Studio.

Результати – розроблено та впроваджено удосконалення для нашої мобільної соціальної мережі. Розробленно графічний дизайн, враховуючи кращі практики щодо користувацького досвіду. А також забезпечення надійності та безпеки системи.

МОБІЛЬНА СОЦІАЛЬНА МЕРЕЖА,
ANDROID, FIREBASE, USER EXPERIENCE, ГРАФІЧНИЙ ІНТЕРФЕЙС

ЗМІСТ

ВСТУП

У нашому світі соціальні медіа стали складовою повсякденного життя, особливо в мобільному просторі, де люди проводять багато часу в таких мережах, як месенджери. Зараз існує досить багато різних типів соціальних мереж, які пропонують користувачам різноманітні можливості, від спілкування на музичних платформах до обміну фотографіями та відео.

Тому існує потреба в розробці систем соціальних мереж з можливістю створювати акаунти та публікувати пости для більш локальних цілей. Основною метою цієї роботи є розвиток власних професійних навичок у сфері комп'ютерної інженерії та прийняття самостійних рішень в реальних умовах ринку праці. Проект дасть можливість отримати досвід роботи з сучасними технологіями та набути теоретичних і практичних знань, необхідних для успішної професійної діяльності.

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1 Аналіз принципів роботи системи

Аналіз принципів роботи системи включає такі основні аспекти:

Збереження даних: для збереження даних використано базу даних, яка зберігає дані у форматі JSON на окремому сервері [1]. Ця база даних надає можливість оновлювати дані в реальному часі, що означає, що зміни, зроблені користувачами, будуть негайно відображатися для інших користувачів системи [2].

Основна функціональність: основною задачею системи є реалізація механізму створення аккаунтів для користувачів, створення та управління постами та налаштування правил доступу до записів у базі даних. Для досягнення цих цілей система повинна мати механізм асинхронної роботи на фоні, щоб забезпечити швидку та ефективну обробку завдань, які вимагають багато часу. Крім того, система повинна забезпечувати підгрузку всіх необхідних даних для користувача та відповідати на можливі виключення або помилки, що можуть виникнути під час роботи [2].

Взаємодія з базою даних: для збереження та отримання даних система використовує функціонал Firebase, який надає зручний інтерфейс для роботи з хмарною базою даних. Це дозволяє забезпечити швидке та безпечне зберігання даних, а також доступ до них з різних пристроїв та місць [2].

Фронтенд додаток: Фронтендна частина системи реалізована як додаток під операційну систему Android. Цей додаток надає користувачам зручний інтерфейс для взаємодії з системою соціальної мережі. Він дозволяє створювати аккаунти, додавати пости та налаштовувати права доступу. За допомогою фронтендного додатку користувачі можуть переглядати та редагувати дані, спілкуватися з іншими користувачами та виконувати інші функції, які передбачені соціальною мережею [3].

Загальною метою цієї системи є створення функціональної та надійної соціальної мережі для користувачів під операційною системою Android. Вона забезпечує зручний спосіб обміну даними, спілкування та співпраці між користувачами.

1.2 Інструменти для розробки системи

Інструменти для розробки системи соціальної мережі під Android включають наступні компоненти:

- **Android Jetpack Compose**: Це сучасний фреймворк, який дозволяє будувати користувацький інтерфейс за допомогою Kotlin коду. Замість традиційного XML, ви можете створювати UI компоненти з використанням декларативного підходу. Compose забезпечує швидку розробку, простоту управління станом та зручну перевикористовуваність компонентів [4].
- **Clean Architecture**: Це архітектурний підхід, який допомагає організувати додаток на незалежні шари з мінімальним зв'язком між ними. Clean Architecture включає в себе шар Domain (бізнес-логіка), шар Data (джерела даних) та шар Presentation (інтерфейс користувача). Це дозволяє вам легко змінювати або розширювати окремі частини додатку без впливу на інші компоненти [5].
- **Model-View-Intent (MVI)**: Це шаблон проектування, який допомагає керувати станом додатку. У MVI ви маєте Model, який містить бізнес-логіку та стан додатку, View, який відповідає за візуалізацію даних, та Intent, який обробляє взаємодії користувача. Цей підхід допомагає уникнути прямих залежностей між компонентами та забезпечити однозначну обробку подій [6].
- **Firebase**: Firebase - це набір хмарних сервісів від Google, які надають широкий функціонал для розробки додатків. Firebase Realtime Database забезпечує можливість синхронізувати та обмінюватись даними в реальному часі. Firebase Cloud Storage дозволяє зберігати файли, такі як зображення або

відео. Firebase Authentication надає можливість аутентифікації користувачів вашої соціальної мережі [2].

- **Dependency Injection (DI) з використанням Hilt**: DI - це підхід, що дозволяє керувати залежностями між компонентами додатку. Hilt - це бібліотека, що вбудовує DI в додаток під Android. Використовуючи Hilt, ви можете легко визначати залежності та їхній життєвий цикл, спрощуючи процес створення та внедрення об'єктів [7].
- **Kotlin Coroutines та Flow**: Kotlin Coroutines - це легкий спосіб працювати з асинхронними операціями в Kotlin [8]. Вони дозволяють зручно виконувати довготривалі операції без блокування основного потоку. Flow - це реактивний потік даних в Kotlin, який дозволяє зручно працювати з потоками даних та здійснювати обробку змін стану [9].
- **Android компоненти**: Android компоненти, такі як Activity, Fragment, ViewModel та Service, є основними будівельними блоками додатку під Android. Вони допомагають реалізувати різні функціональність соціальної мережі, включаючи навігацію між екранами, збереження та відновлення стану, кешування даних та взаємодію з Firebase сервісами [3].

Ці основні моменти разом створюють потужний набір інструментів для розробки функціональної та легко керованої системи соціальної мережі під Android.

1.3 Постановка задачі

На основі зібраних та проаналізованих літературних джерел можна сформулювати ключові етапи:

1. Створити патерни для бази даних та налаштувати аккаунт Firebase:

- Розробити схеми та патерни для бази даних, що відповідають вимогам системи.
- Створити аккаунт Firebase та налаштувати необхідні параметри, включаючи ключі API та конфігураційні файли.

2. Проробити правила доступу та підключити аутентифікацію:

- Налаштувати правила доступу до бази даних для забезпечення безпеки та конфіденційності даних.
- Підключити механізм аутентифікації Firebase для реєстрації, входу в систему та керування обліковим записом користувача.

3. Створити Firebase Storage для підгрузки файлів:

- Створити Firebase Storage для зберігання та обробки різних типів файлів, таких як зображення, відео тощо.

4. Інтегрувати залежності Firebase SDK до мобільного додатку:

- Підключити необхідні залежності Firebase SDK до мобільного додатку.
- Налаштувати додаток для використання Firebase сервісів.

5. Створити основні модулі за Clean Architecture та пакети для доставки даних:

- Розробити модулі Presentation Layer, Domain Layer та Data Layer для розділення логіки системи на компоненти та шари.
- Створити пакети, що відповідають за доставку даних між шарами.

6. Створити репозиторій та переписати колбеки під корутини:

- Реалізувати репозиторій з основним функціоналом системи.
- Переписати колбеки на використання корутинів для забезпечення асинхронного програмування та зручної обробки запитів.

7. Розділити логіку на useCases та забезпечити доступність даних:

- Розбити логіку системи на окремі useCases, відповідні кожній конкретній функціональності.
- Забезпечити простий та зручний доступ до даних для інших компонентів системи.

8. Розробити графічний інтерфейс для взаємодії з системою.

Виконання цих завдань дозволить успішно розробити та налаштувати систему соціальної мережі під операційну систему Android з використанням Compose, Clean Architecture, MVI та Firebase.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Конфігурація проекту, створення бази даних та аутентифікації

Для вирішення поставлених завдань розроблено мобільний додаток, який складається з трьох основних модулів, що мають доступ до необхідних джерел даних. Така архітектура дозволяє масштабувати контент і легко впроваджувати нові функції та оновлення [5].

На рисунку 2.1 показана архітектура, на основі якої був розроблений додаток. Використано Clean Architecture [5], що дозволяє розділити дані на різні рівні і уникнути змішування між ними. Інтерфейс додатку побудований на паттерні MVI (Model-View-Intent) [6], який реалізується за допомогою підходу станів додатку.

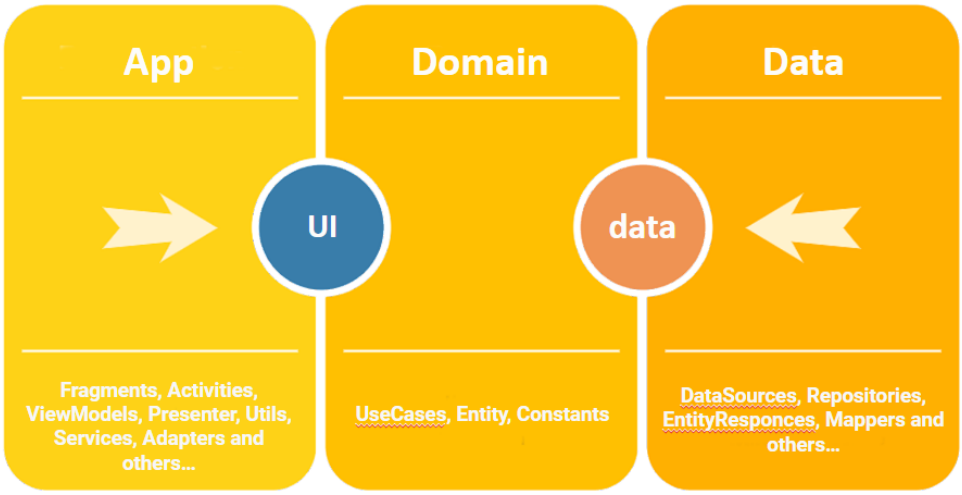


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.1 – Clean Architecture

На рисунку 2.1 показана конкретна реалізація архітектури додатку. Виділено всі операції, пов'язані з обміном даними, в окремий модуль, який незалежний від частини проекту, що відповідає за користувацький інтерфейс. Це надає можливість мобільності додатку (наприклад, підключення цього модуля до різних реалізацій проекту, таких як веб-сайт або десктопний додаток) [5].

Всі операції в нашому додатку виконуються асинхронно за допомогою корутин, які є оптимальним і швидкодіючим рішенням для мови програмування Kotlin. Використання корутин дозволяє зручно та ефективно управляти асинхронними операціями, зменшує навантаження на потоки і має певні переваги порівняно зі стандартними потоками [8].

Застосування такої архітектури та використання корутин допомагає забезпечити якісний та продуктивний розвиток нашої соціальної мережі з можливістю створення аккаунтів та лістингу постів, а також забезпечує зручний та ефективний інтерфейс для користувачів.

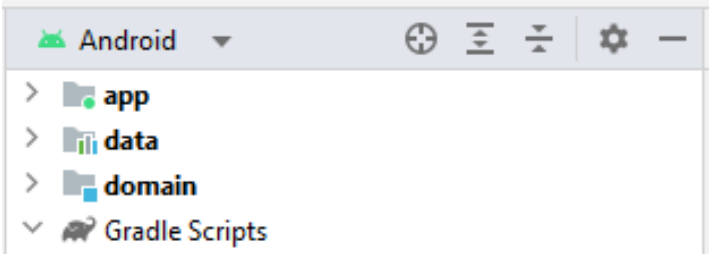


*Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.2 – Архітектура додатку (реалізація)*

Залежності DATA модулю:

```
dependencies {
    // Dagger Hilt
    implementation "com.google.dagger:hilt-android:2.44"
    implementation 'com.google.firebase:firebase-database-ktx:20.0.4'
    implementation 'com.google.firebase:firebase-firestore-ktx:24.5.0'
    kapt "com.google.dagger:hilt-compiler:2.44"

    // include modules
    implementation project(path: ':domain')

    // firebase auth
    implementation 'com.google.firebase:firebase-auth-ktx:21.3.0'

    // default
    implementation 'androidx.core:core-ktx:1.7.0'
    testImplementation 'junit:junit:4.13.2'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
}
```

Залежності DOMAIN модулю:

```
dependencies {
    // coroutine
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.1'

    // serialization
    implementation "org.jetbrains.kotlinx:kotlinx-serialization-json:1.3.0"

}
```

Залежності APP модулю:

```
dependencies {

    // include modules
    implementation project(path: ':data')
    implementation project(path: ':domain')

    // serialization
    implementation "org.jetbrains.kotlinx:kotlinx-serialization-json:1.3.0"

    implementation("io.coil-kt:coil-compose:2.1.0")

    // Jetpack Navigation
    implementation 'androidx.navigation:navigation-compose:2.5.3'

    // coroutine
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.1'

    // viewModel
    implementation 'androidx.fragment:fragment-ktx:1.5.2'
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1'
    implementation 'com.google.firebase:firebase-database-ktx:20.2.0'

    // lifecycle
    def lifecycle_version = "2.5.1"
    implementation "androidx.lifecycle:lifecycle-livedata-
ktx:$lifecycle_version"
    implementation "androidx.lifecycle:lifecycle-runtime-
ktx:$lifecycle_version"
    implementation "androidx.lifecycle:lifecycle-viewmodel-
savedstate:$lifecycle_version"
    implementation "androidx.lifecycle:lifecycle-common-
java8:$lifecycle_version"


    // Dagger Hilt
    implementation "com.google.dagger:hilt-android:2.44"
    kapt "com.google.dagger:hilt-compiler:2.44"
```

```
    implementation 'androidx.core:core-ktx:1.8.0'
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.5.1'
    implementation "androidx.compose.ui:ui:$compose_ui_version"
    implementation "androidx.compose.ui:ui-tooling-
preview:$compose_ui_version"
    implementation 'androidx.compose.material:material:1.2.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.appcompat:appcompat:1.5.1'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.4'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.0'
    androidTestImplementation "androidx.compose.ui:ui-test-
junit4:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-tooling:$compose_ui_version"
    debugImplementation "androidx.compose.ui:ui-test-
manifest:$compose_ui_version"

}
```

Завдяки ізольованості певних залежностей, модулі використовують лише ті бібліотеки що їм необхідні не підгружаючи всі підряд [12]. Також це позволяє нам у майбутньому перевикористовувати певні модулі, та міняти логіку додатку [5].

Для створення бази даних було встановлено правила доступу до контенту. Налаштовано правила таким чином, що всі користувачі можуть читати дані, але запис дозволений лише авторизованим користувачам.

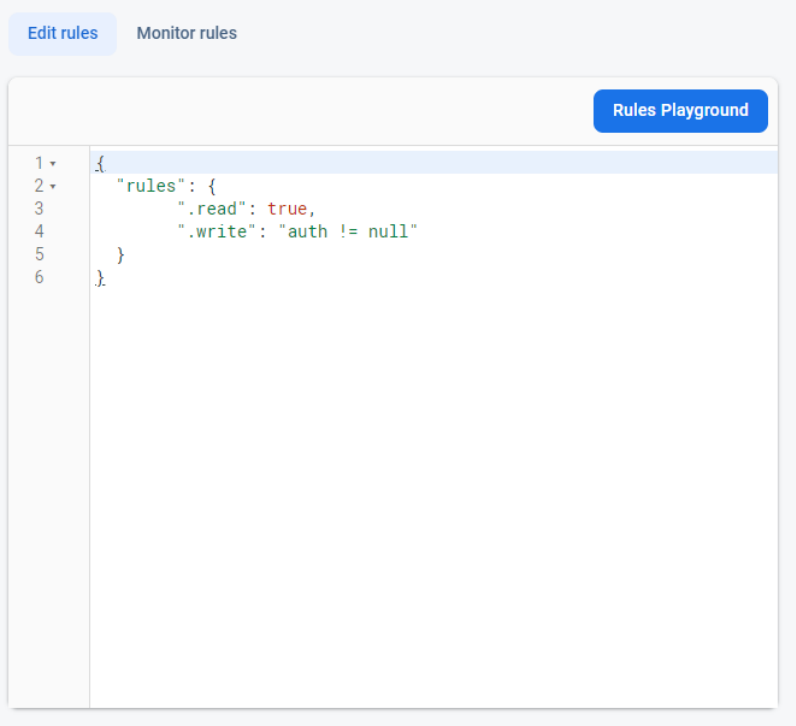


Рисунок 2.3 – Правила доступу до бази даних

Для забезпечення аутентифікації користувачів використано пошту та пароль. Користувачі реєструються в системі, вказуючи свою електронну пошту та обираючи пароль.

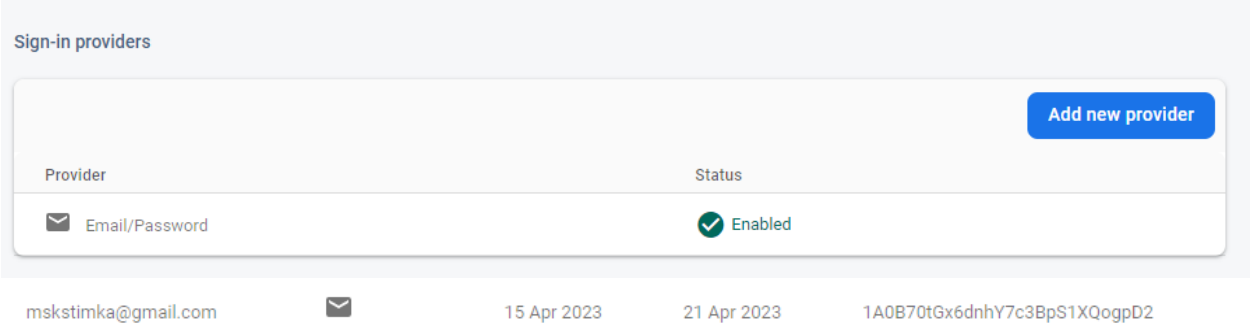


Рисунок 2.4 – Реалізація аутентифікації користувачів

Після реєстрації, користувачі можуть використовувати свою електронну пошту та пароль для входу в систему. Після успішної аутентифікації, вони отримують доступ до своїх особистих даних та можуть здійснювати записи у базу даних, якщо вони мають на це відповідні права.

Цей підхід до створення бази даних та аутентифікації забезпечує безпеку та конфіденційність даних, оскільки лише авторизовані користувачі мають можливість змінювати чи вносити записи у систему [2].

2.2 Модуль по обробці даних (DATA)

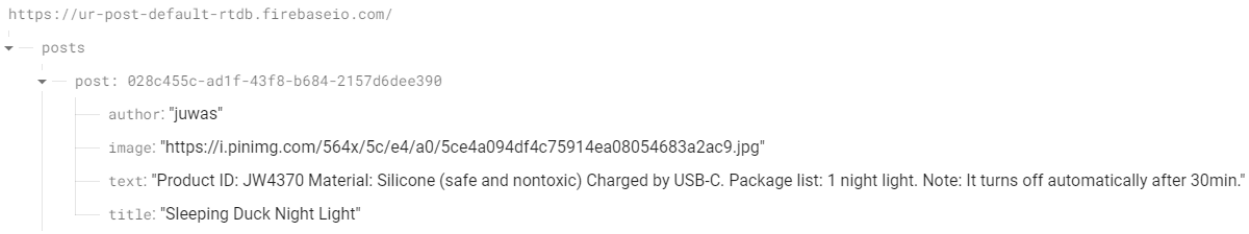
Для збереження даних користувачів використано віддалений сервер, який надає функціонал Firebase. Це дозволяє зберігати дані не локально на пам'яті пристрою, а відправляти їх на сервер для зберігання. Такий підхід забезпечує оптимізацію додатку і дозволяє ефективно використовувати обмежені ресурси смартфона [2].

Для взаємодії з базою даних Firebase використано реалізацію інтерфейсу **Repository.Firebase**, який містить основні потоки даних, необхідні для взаємодії з базою даних.

Інтерфейс Repository.Firebase:

```
interface Repository {
    // Firebase repository interface for data operations

    // Отримання поточного користувача
    suspend fun getCurrentUser(): Results<User>

    // Авторизація користувача
    suspend fun signIn(password: String, email: String): Results<User>

    // Створення нового користувача
    suspend fun createUser(password: String, email: String, displayName:
String): Results<User>

    // Вихід з облікового запису
    suspend fun signOut(): Results<User?>

    // Запис публікації
    suspend fun writePost(post: Post)

    // Читання публікацій
    suspend fun readPosts(): Results<List<Post>>

    // Підписка на отримання оновлень публікацій
    suspend fun subscribeToPosts(): Flow<List<Post>>
}
```

Отримані дані зберігаються у відповідних об'єктах, які відображають структуру бази даних. Ці дані можуть бути, наприклад, інформація про користувачів, їх профілі, пости та інші дані, пов'язані з функціоналом соціальної мережі.



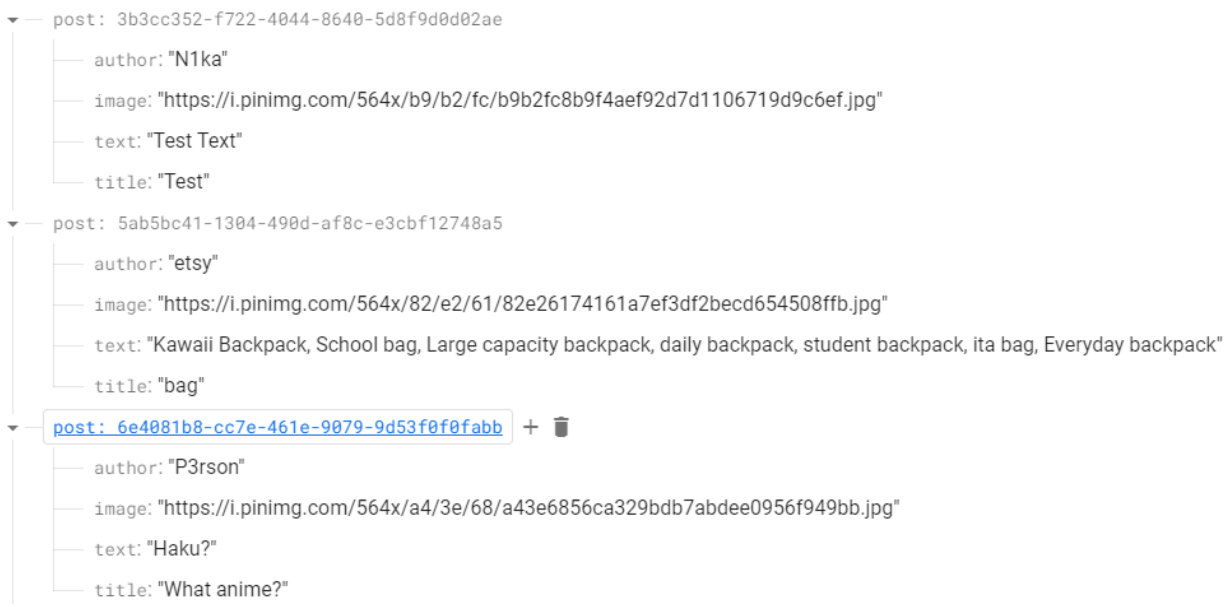


Рисунок 2.5 – Збережені дані

Для зручності організації логіки додатку використоано підхід з маркуванням кейсів дій як UseCases. Це дозволяє групувати відповідні дії та функції в окремих класах і забезпечує зрозумілу структуру та доступ до необхідних функцій [5].

Застосування Firebase для збереження даних дозволяє ефективно працювати з базою даних, забезпечуючи швидкий доступ до даних та забезпечення безпеки та надійності інформації користувачів [2].

API Firebase не підтримує асинхронний підхід з використанням корутин [2], який є популярним у мобільних додатках під операційною системою Android. Зазвичай його використовували разом з багатопотоковістю RxJava [10]. Однак, є можливість сконвертувати колбек-підхід API Firebase в асинхронний підхід, використовуючи корутини та flow [8].

Для цього створено враппер або оболочку, яка дозволяє обробляти колбеки за допомогою корутин. Основна ідея полягає в тому, що створюється корутина, яка чекає на виклик колбеку, а потім повертає результат у вигляді потоку (flow) корутин [8].

Реалізація підписки на оновлення постів:

```
override suspend fun subscribeToPosts(): Flow<List<Post>> = callbackFlow {
    // Створення callbackFlow для створення потоку

    val listener = object : ValueEventListener {
        // Створення об'єкта ValueEventListener для прослуховування змін в
базі даних

        override fun onDataChange(snapshot: DataSnapshot) {
            // Метод, який викликається при зміні даних в базі даних
            val posts = mutableListOf<Post>()
            // Створення змінної для збереження списку постів

            for (childSnapshot in snapshot.children) {
                // Ітерація через дочірні елементи бази даних

                val user = childSnapshot.getValue(Post::class.java)
                // Отримання об'єкта типу Post з дочірнього елементу

                if (user != null) {
                    posts.add(user)
                    // Додавання об'єкта Post до списку постів
                }
            }

            this@callbackFlow.trySend(posts).isSuccess
            // Відправка списку постів через потік callbackFlow
        }

        override fun onCancelled(error: DatabaseError) {
            // Метод, який викликається при скасуванні операції підписки або
помилці
            close(error.toException())
            // Закриття callbackFlow та передача винятку
        }
    }

    reference.addValueEventListener(listener)
    // Додавання ValueEventListener до посилання reference

    awaitClose { reference.removeEventListener(listener) }
    // Закриття callbackFlow та видалення слухача ValueEventListener

}.distinctUntilChanged().flowOn(Dispatchers.IO)
// Обробка результатного потоку та встановлення планувальника для введення-
виведення
```

Реалізація логіну користувача:

```
override suspend fun signIn(password: String, email: String): Results<User> {
    // Перевизначений метод для входу користувача

    return try {
        val authResult = withContext(Dispatchers.IO) {
            auth.signInWithEmailAndPassword(email, password).await()
        }.user
        // Виконання аутентифікації користувача за допомогою електронної
пошти та пароля
        // За допомогою withContext та Dispatchers.IO, операція виконується в
фоновому потоці

        when (authResult) {
            null -> Results.Error(Exception("User is null"))
```

```
            // Якщо результат аутентифікації є нульовим, повертається помилка
            else -> Results.Success(authResult.toUser())
            // В іншому випадку, результат аутентифікації конвертується в
об'єкт користувача та повертається успішний результат
        }
    } catch (e: Exception) {
        Results.Error(e)
        // У випадку виникнення винятку, повертається помилка з виключенням
    }
}
```

Реалізація створення профілю:

```
override suspend fun createUser(
    password: String,
    email: String,
    displayName: String
): Results<User> = withContext(Dispatchers.IO) {
    // Перевизначений метод для створення користувача

    return@withContext try {
        val authResult = auth.createUserWithEmailAndPassword(email,
password).await()
        // Створення нового користувача за допомогою електронної пошти та
пароля
        // За допомогою withContext та Dispatchers.IO, операція виконується в
фоновому потоці

        authResult.user?.updateProfile(

UserProfileChangeRequest.Builder().setDisplayName(displayName).build()
        )?.await()
        // Оновлення профілю користувача з вказаною назвою відображення

        val user = authResult.user
        // Отримання об'єкта User з authResult

        if (user != null) {
            Log.d("User:", user.toUser().toString())
            // Якщо об'єкт User не є нульовим, відображається
відлагоджувальне повідомлення
            Results.Success(user.toUser())
            // Повертається успішний результат з об'єктом користувача
        } else {
            Results.Error(Exception("User is Null"))
            // Якщо об'єкт User є нульовим, повертається помилка
        }
    } catch (e: Exception) {
        Results.Error(e)
        // У випадку виникнення винятку, повертається помилка з виключенням
    }
}
```

Реалізація провірки поточного користувача:

```
override suspend fun getCurrentUser(): Results<User> =
withContext(Dispatchers.IO) {
    // Перевизначений метод для отримання поточного користувача

    val currentUser = auth.currentUser
```

```
    // Отримання поточного користувача з об'єкта FirebaseAuth

    return@withContext if (currentUser != null) {
        Results.Success(currentUser.toUser())
        // Якщо поточний користувач не є нульовим, повертається успішний
результат з об'єктом користувача
    } else {
        Results.Error(Exception(DEFAULT_EXCEPTION))
        // Якщо поточний користувач є нульовим, повертається помилка зі
створеним винятком
    }
}
```

Реалізація виходу з профілю:

```
override suspend fun signOut(): Results<User?> = withContext(Dispatchers.IO)
{
    // Перевизначений метод для виходу з облікового запису користувача

    auth.signOut()
    // Виклик методу signOut() для виходу з облікового запису

    return@withContext if (auth.currentUser == null) {
        Results.Success(null)
        // Якщо поточний користувач є нульовим після виходу, повертається
успішний результат зі значенням null
    } else {
        Results.Error(Exception("Sign Out Error"))
        // Якщо поточний користувач не є нульовим після виходу, повертається
помилка зі створеним винятком
    }
}
```

Реалізація створення поста:

```
override suspend fun writePost(post: Post): Unit =
withContext(Dispatchers.IO) {
    // Перевизначений метод для створення нового посту

    try {
        val currentUser = FirebaseAuth.getInstance().currentUser
        // Отримання поточного користувача з Firebase Authentication

        if (currentUser != null) {
            reference.child(CHILD_PATH_STRING +
UUID.randomUUID()).setValue(post).await()
            // Якщо поточний користувач не є нульовим, записується новий пост
у базу даних
            // За допомогою setValue() та await(), операція виконується в
асинхронному режимі
        }

    } catch (e: Exception) {
        e.printStackTrace()
        // У випадку виникнення винятку, виводиться трасування стеку в
консоль
    }
}
```

Реалізація отримання всіх постів:

```
override suspend fun readPosts(): Results<List<Post>> =
withContext(Dispatchers.IO) {
```

```
    // Перевизначений метод для отримання списку постів

    val posts = mutableListOf<Post>()
    // Створення змінної для зберігання списку постів

    try {
        val snapshot = reference.get().await()
        // Отримання снапшоту з бази даних

        for (childSnapshot in snapshot.children) {
            // Ітерація по дочірнім елементам снапшоту

            val user = childSnapshot.getValue(Post::class.java)
            // Отримання об'єкта поста з дочірнього снапшоту

            if (user != null) {
                posts.add(user)
                // Якщо об'єкт поста не є нульовим, додається до списку
постів
            }
        }

    } catch (e: Exception) {
        return@withContext Results.Error(e)
        // У випадку виникнення винятку, повертається помилка з виключенням
    }

    return@withContext Results.Success(posts)
    // Повертається успішний результат зі списком постів
}
```

В цьому випадку використано функцію **callbackFlow**, яка дозволяє створювати потік (flow) на основі колбеків. Створюється корутина, яка очікує на виклик колбеку і відправляє його значення до потоку за допомогою функції **send**. Коли виклик колбеку завершується, закривається потік за допомогою функції **close** [8].

Цей підхід дозволяє працювати з API Firebase за допомогою корутин, що полегшує асинхронну обробку даних та зменшує ризик виникнення проблем, пов'язаних з багатопотоковістю [2]. Крім того, використання потоку (flow) дозволяє зручно працювати з послідовними асинхронними подіями, такими як завантаження файлів або отримання стрічки постів з бази даних [8].

Інтерфейс DataSource:

```
interface DataSource {

    interface Auth {
        suspend fun getCurrentUser(): Results<User>
        // Функція для отримання поточного користувача

        suspend fun signIn(password: String, email: String): Results<User>
```

```
        // Функція для входу користувача

        suspend fun createUser(password: String, email: String, displayName:
String): Results<User>
        // Функція для створення нового користувача

        suspend fun signOut(): Results<User?>
        // Функція для виходу з облікового запису користувача
    }

    interface DataBase {
        suspend fun subscribeToPosts(): Flow<List<Post>>
        // Функція для підписки на оновлення списку постів

        suspend fun writePost(post: Post)
        // Функція для створення нового поста

        suspend fun readPosts(): Results<List<Post>>
        // Функція для отримання списку постів
    }
}
```

DataSource використано як інтерфейс який спілкується однієї сторони з репозиторіями (що знаходяться по ієрархії вище), а з іншої з Firebase API [5].

2.3 Бізнес логіка, та незалежний модуль Domain

У проекті використовується незалежний модуль, який називається "Domain". В цьому модулі зосереджена бізнес-логіка додатку, включаючи основні сутності та використовувані кейси (useCases). Особливістю цього модуля є його незалежність від модуля "app" і модуля "data". Це означає, що модуль "Domain" не має прямого знання про деталі реалізації інтерфейсу Repository.Firebase (зображений на рисунку 2.6) або інших конкретних технічних реалізацій [5].

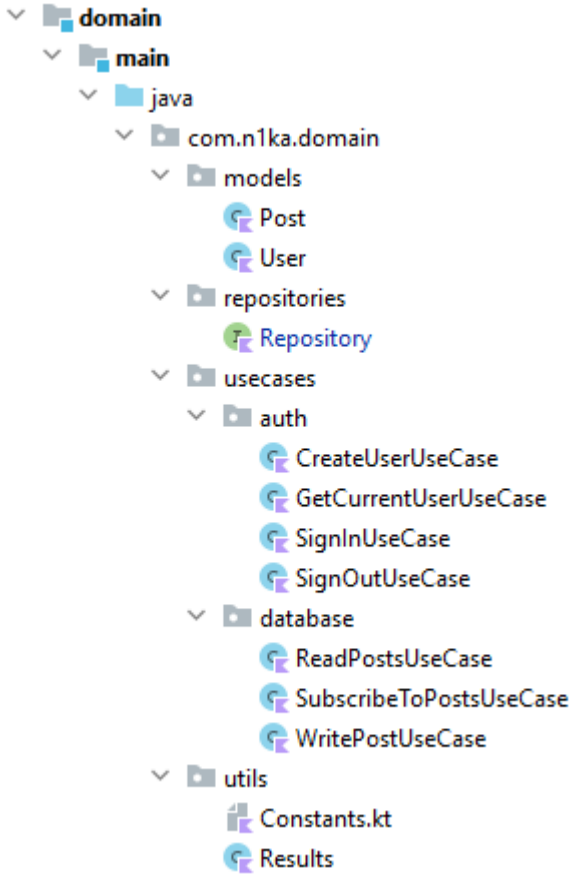


Рисунок 2.6 – Domain модуль

Модуль "Domain" виконує роль посередника між модулем "app" та модулем "data". Він містить основну логіку додатку, яка не залежить від певних платформенних деталей або джерел даних. Це забезпечує високу переносимість та можливість змінювати або розширювати реалізацію модуля "data" без необхідності змінювати код у модулі "Domain" [5].

У модулі "Domain" визначено основні сутності, такі як користувачі, пости, коментарі тощо. Також тут містяться важливі кейси (useCases), які відображають основні дії та операції, які можна виконати в додатку [11]. Наприклад, створення нового посту, отримання списку користувачів, відправлення повідомлення тощо. Кейси взаємодіють з реалізацією інтерфейсу Repository.Firebase, що дозволяє взаємодіяти з базою даних Firebase [5].

Цей незалежний модуль "Domain" дозволяє нам розділити логіку додатку на чіткі компоненти та забезпечити чистоту бізнес-логіки, що

полегшує тестування, розширення та підтримку коду. Він створює абстракцію між різними модулями додатку, забезпечуючи їх взаємодію через визначені контракти та інтерфейси, а не через конкретні реалізації [5]. Також такий підхід позволяє у подальшому проводити тестування за допомогою підходу Mockk [13].

Сутність Post:

```
@Serializable
class Post {

    var author: String? = null
    // Властивість, що зберігає ім'я автора поста

    var title: String? = null
    // Властивість, що зберігає заголовок поста

    var text: String? = null
    // Властивість, що зберігає текст поста

    var image: String? = null
    // Властивість, що зберігає посилання на зображення поста

    constructor() {}
    // Порожній конструктор для серіалізації/десеріалізації

    constructor(author: String?, text: String?, title: String?, image:
String?) {
        // Конструктор, що ініціалізує всі властивості класу
        this.author = author
        this.text = text
        this.title = title
        this.image = image
    }
}
```

Сутність User:

```
data class User(
    val uid: String,
    val displayName: String?,
    val email: String?,
    val photoUrl: String?
)
```

Вище вказані сутності використовуються як контейнер для того щоб отримати дані за покласти в них, для подальшого використання та виводу користувачеві за допомогою інтерфейсу.

3. РОЗРОБКА ГРАФІЧНОГО ІНТЕРФЕЙСУ

3.1 Розробка домашнього екрану

Основним стеком нашої графічної частини додатку буде написання нативного UI за допомогою jetpack compose. Данний спосіб позволить використовувати одну мову програмування (Kotlin) для логічної частини дизайну та графічних елементів [4]. В перспективі це дасть можливість використовувати додаток на різні платформи [14].

При відкритті додатку, користувача зустрічає домашня сторінка, з постами різних користувачів.

Весь екран розподілений на 6 composable функцій:

- MainScreen – основний контент головної сторінки.
- CustomCurcularProgressBar – прогресс бар для відображення загрузки контенту (поки контент не загрузився)
- DrawerContentAuthorized – відвигающе вікно що показує авторизацію, якщо користувач залогінений
- DrawerContentNoAuthorized – відвигающе вікно що показує статус не авторизованого користувача, та направляє його до login/registration screen.
- PostList – контейнер для списку постів
- PostItem – айтем з дизайну поста

Програмний код домашньої сторінки (MainScreen):

```
@OptIn(ExperimentalMaterialApi::class)
@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun MainScreen(viewModel: HostViewModel, navigateTo: (route: Screen) -> Unit)
{
    // Зберігання стану бокової панелі
    val drawerState = rememberDrawerState(initialValue = DrawerValue.Closed)

    // Зберігання ширини бокової панелі
    var drawerWidth by remember {
        mutableStateOf(drawerState.offset.value)
    }

    // Зберігання зсуву контенту
    val contentOffset = remember {
        derivedStateOf {
            drawerState.offset.value
```

```
        }
    }

    // Створення області корутин
    val coroutineScope = rememberCoroutineScope()

    // Зберігання стану Scaffold
    val scaffoldState = rememberScaffoldState(drawerState = drawerState)

    // Підписка на подію в ViewModel при запуску компонента
    LaunchedEffect(Unit) {
        viewModel.sendEvent(HostEvent.SubscribeToPosts)
    }

    // Обробник натискання кнопки "назад"
    BackHandler(onBack = {
        if (scaffoldState.drawerState.isOpen) {
            coroutineScope.launch { scaffoldState.drawerState.close() }
        } else {
            scaffoldState.drawerState.offset
            // Збереження стану бокової панелі
        }
    })

    // Застосування теми додатку
    UrPostTheme(false) {

        // Побудова Scaffold
        Scaffold(
            scaffoldState = scaffoldState,
            drawerElevation = 0.dp,
            drawerScrimColor = Color.Transparent,

            // Плаваюча кнопка "створити пост"
            floatingActionButton = {
                val xPos = (abs(drawerWidth) - abs(contentOffset.value))
                FloatingActionButton(
                    modifier = Modifier
                        .offset(
                            x = with(LocalDensity.current) {
                                max(0.dp, xPos.toDp() - 56.dp)
                            }
                        ),
                    onClick = {
                        navigateTo(Screen.WritePost)
                    },
                    backgroundColor = colorResource(id =
R.color.post_background)
                ) {
                    Icon(
                        tint = Color.White,
                        imageVector = Icons.Default.Add,
                        contentDescription = "Создать пост"
                    )
                }
            },

            // Верхній бар
            topBar = {
                TopAppBar(
                    elevation = 0.dp,
                    title = {
                        Text(
```

```
                                text = stringResource(id =
R.string.top_app_bar_name),
                                fontFamily = FONT_FAMILY,
                                color = Color.Black
                            )
                        },
                        backgroundColor = colorResource(id =
R.color.background_color),
                        navigationIcon = {
                            IconButton(onClick = {
                                coroutineScope.launch {
                                    drawerState.open()
                                }
                            }) {
                                Icon(
                                    imageVector = Icons.Filled.Menu,
                                    contentDescription =
stringResource(R.string.icon_menu_description),
                                    tint = Color.Black
                                )
                            }
                        }
                    )
                },

                // Вміст бокової панелі
                drawerContent = {
                    if (state.isAuthorized) {
                        viewModel.sendEvent(HostEvent.GetCurrentUser)
                        val user = state.user

                        if (user != null) {
                            DrawerContentAuthorized(
                                user = user,
                                sendEvent = { event -> viewModel.sendEvent(event)
}
                            )
                        }
                    } else {
                        DrawerContentNoAuthorized(
                            navigateTo = navigateTo,
                            closeDrawer = {
                                coroutineScope.launch { drawerState.close() }
                            }
                        )
                    }
                },

                // Основний вміст
                content = {
                    if (state.isShowProgress) {
                        CustomCircularProgressBar()
                    } else {
                        val xPos = (abs(drawerWidth) - abs(contentOffset.value))
                        Box(
                            Modifier
                                .fillMaxSize()
                                .offset(
                                    x = with(LocalDensity.current) {
                                        max(0.dp, xPos.toDp() - 56.dp)
                                    }
                                )
                        ) {
```

```
                        PostList(
                            list = state.list,
                            navigateTo = navigateTo,
                            selectPost = { post ->
viewModel.sendEvent(HostEvent.SelectPost(post)) }
                        )
                    }
                }
            },
        )
    }
}
```

Рисунок 3.1 – Графічний вигляд MainScreen

Для того щоб користувач не відчував дискомфорту під час того коли контент загружається, потрібно показувати загружочні елементи.

Програмний код елементу загрузки (CustomCircularProgressBar):

```
@Composable
private fun CustomCircularProgressBar() {
    // Розміщення прогрес-бару в центрі екрану
    Row(
        modifier = Modifier
            .fillMaxSize()
            .background(colorResource(id = R.color.background_color)),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.Center
    ) {
        // Відображення прогрес-бару
        CircularProgressIndicator(
            modifier = Modifier.size(size = 100.dp),
            color = colorResource(id = R.color.post_background),
            strokeWidth = 10.dp
        )
    }
}
```

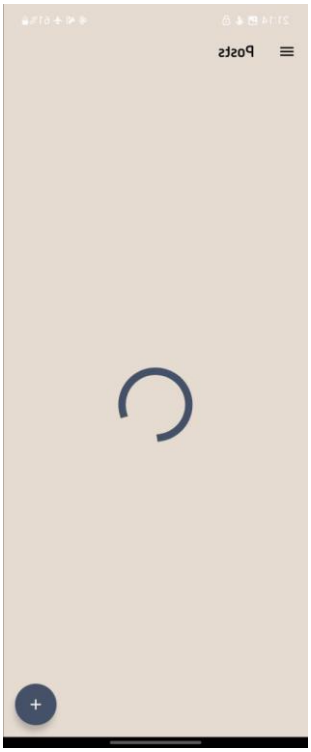


Рисунок 3.2 – Графічний вигляд елементу загрузки

Для зручної взаємодії з аккаунтом, було розробленно висуваюче вікно drawer, який при свайпі вправо відкривається змістивши основний контент

вправо. У данного елементу дизайна є два стани: авторизований та неавторизований.

Програмний код авторизованого висувного меню (DrawerContentAuthorized):

```
@Composable
fun DrawerContentAuthorized(user: User, sendEvent: (event: HostEvent) ->
Unit) {
    // Вертикальний контейнер для вмісту панелі бічного меню користувача
    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(
                colorResource(id = R.color.background_color)
            ),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        // Іконка облікового запису користувача
        Icon(
            imageVector = Icons.Filled.AccountCircle,
            modifier = Modifier.size(150.dp),
            tint = colorResource(id = R.color.post_background),
            contentDescription = "Account",
        )

        // Ім'я користувача
        Text(
            text = user.displayName.toString(),
            fontFamily = FONT_FAMILY,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(top = 12.dp),
            color = Color.Black
        )

        // Email користувача
        Text(
            text = user.email.toString(),
            fontFamily = FONT_FAMILY,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(top = 12.dp),
            color = Color.Black
        )

        // Кнопка вийти з облікового запису
        Button(
            colors = ButtonDefaults.buttonColors(
                backgroundColor = colorResource(id =
R.color.error_text_color)
            ),
            onClick = {
                sendEvent(HostEvent.SignOut)
            },
            shape = RoundedCornerShape(20.dp)
        ) {
            Text(
                text = "Sign Out",
                fontFamily = FONT_FAMILY,
                fontWeight = FontWeight.Bold,
                color = Color.White
```

```
            )
        }
    }
}
```

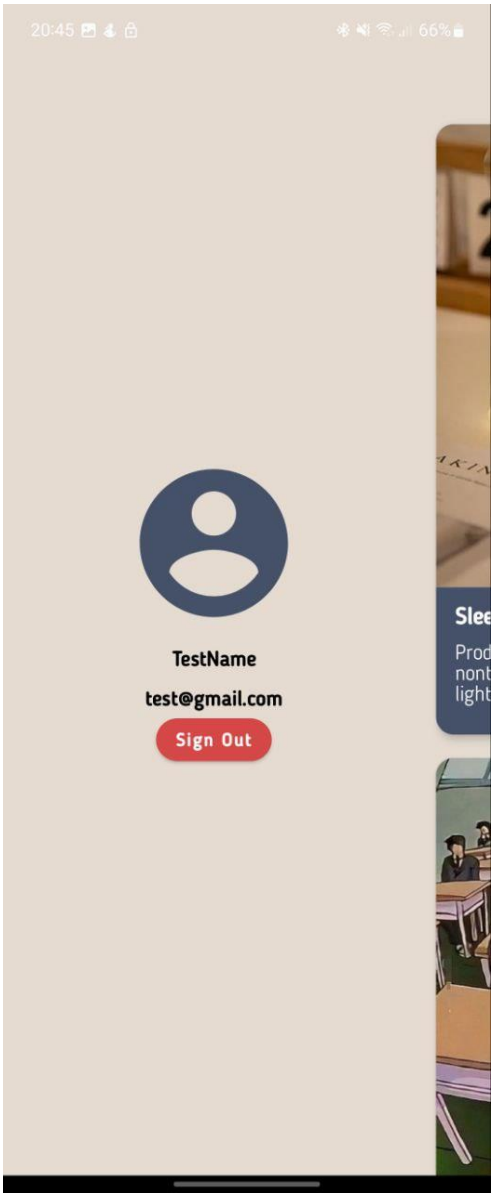


Рисунок 3.3 – Графічний вигляд авторизованого висувного меню

Програмний код неавторизованого висувного меню (DrawerContentNoAuthorized):

```
@Composable
fun DrawerContentNoAuthorized(navigateTo: (route: Screen) -> Unit,
closeDrawer: () -> Unit) {
    // Вертикальний контейнер для вмісту панелі бічного меню невидимого
користувача
    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(
                colorResource(id = R.color.background_color)
            ),
```

```
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            // Іконка облікового запису користувача
            Icon(
                imageVector = Icons.Filled.AccountCircle,
                modifier = Modifier.size(150.dp),
                tint = colorResource(id = R.color.post_background),
                contentDescription = "Account",
            )

            // Кнопка для переходу до екрану авторизації
            Button(
                colors = ButtonDefaults.buttonColors(
                    backgroundColor = colorResource(id = R.color.post_background)
                ),
                onClick = {
                    navigateTo(Screen.Authorization)
                    closeDrawer()
                },
                shape = RoundedCornerShape(20.dp)
            ) {
                Text(
                    text = stringResource(R.string.sign_in_button_text),
                    fontFamily = FONT_FAMILY,
                    fontWeight = FontWeight.Bold,
                    color = Color.White
                )
            }

            // Текстовий елемент, що вказує на відсутність авторизації
            Text(
                text = stringResource(R.string.not_authorized_text),
                fontFamily = FONT_FAMILY,
                fontWeight = FontWeight.Bold,
                modifier = Modifier.padding(top = 12.dp),
                color = colorResource(id = R.color.error_text_color)
            )
        }
    }
```

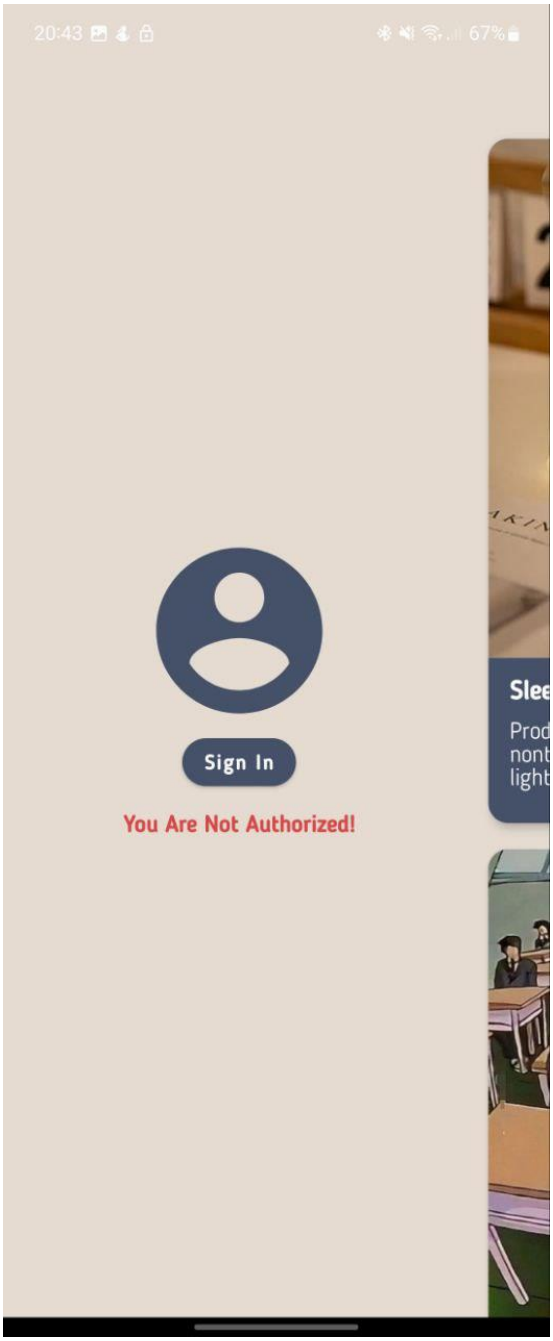


Рисунок 3.4 – Графічний вигляд неавторизованого висувного меню

Для відображення списку постів було створенно контейнер для елементів списку.

Програмний код контейнера для списку постів (PostList):

```
@Composable
fun PostList(
    list: List<Post>,
    navigateTo: (route: Screen) -> Unit,
    selectPost: (post: Post) -> Unit
) {
    // State для збереження стану прокрутки списку
```

```
    val scrollState = rememberLazyListState()

    // Контейнер, що заповнює весь доступний простір
    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(colorResource(id = R.color.background_color))
    ) {
        // Список, що підтримує лініву відображення даних
        LazyColumn(state = scrollState) {
            items(list.size) { post ->
                // Відображення окремого елемента списку (пости)
                PostItem(post = list[post], navigateTo = navigateTo,
selectPost = selectPost)
            }
        }
    }
}
```

А також для заповнення цього списку розроблено розмітку поста, якому передається певна сутність з данними, та ініціалізації для відображення.

Програмний код розмітки поста (PostItem):

```
@Composable
fun PostItem(
    post: Post,
    navigateTo: (route: Screen) -> Unit,
    selectPost: (post: Post) -> Unit
) {
    // Створення моделі для отримання зображення
    val model = ImageRequest.Builder(LocalContext.current)
        .data(post.image)
        .size(Size.ORIGINAL)
        .crossfade(true)
        .build()

    // Завантаження зображення засобами Compose
    val image = rememberAsyncImagePainter(model)

    // Відображення карточки для поста
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(10.dp)
            .clickable {
                selectPost(post)
                navigateTo(Screen.Details)
            },
        shape = RoundedCornerShape(15.dp),
        elevation = 5.dp,
        backgroundColor = colorResource(id = R.color.post_background)
    ) {
        SelectionContainer {
            Column(
                modifier = Modifier
                    .fillMaxWidth()
            ) {
                // Відображення зображення поста
                Image(
```

```
                painter = image,
                contentDescription = null,
                modifier = Modifier
                    .fillMaxWidth()
                    .wrapContentHeight(),
                contentScale = ContentScale.Crop
            )

            // Відображення заголовку поста
            Text(
                text = post.title.toString(),
                fontWeight = FontWeight.Bold,
                modifier = Modifier.padding(bottom = 8.dp, start = 16.dp,
top = 12.dp),
                color = Color.White,
                fontFamily = FONT_FAMILY,
                fontSize = 18.sp
            )

            // Відображення тексту поста
            Text(
                text = post.text.toString(),
                style = MaterialTheme.typography.body1,
                modifier = Modifier
                    .padding(horizontal = 16.dp)
                    .fillMaxWidth(),
                color = Color.White,
                fontFamily = FONT_FAMILY,
                fontWeight = FontWeight.Light
            )

            // Відображення автора поста
            Text(
                text = post.author.toString(),
                color = Color.White,
                modifier = Modifier
                    .padding(end = 16.dp, bottom = 8.dp)
                    .fillMaxWidth(),
                fontFamily = FONT_FAMILY,
                fontWeight = FontWeight.Light,
                fontSize = 12.sp,
                textAlign = TextAlign.End
            )
        }
    }
}
}
```

Рисунок 3.5 – Графічний вигляд розмітки поста

3.2 Розробка екрану авторизації

Для авторизації та створення профілю соціальної мережі, було розробленно окремий екран який запрошує для вводу необхідні кредетали, та авторизує користувача в системі [2].

На екрані є дві основні кнопки Log In та Registration. Якщо у вас є аккаунт то при вводі коректного логіну та пароля та нажання на відповідну кнопку вас авторизує в систему.

Якщо ви новий користувач то при вводі логіну та пароля у вас запросить ввод вашого імені у вспливаючому вікні.

Програмний код екрану авторизації (AuthorizationScreen):

```
@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun AuthorizationScreen(
    navigateTo: (route: Screen) -> Unit,
    viewModel: HostViewModel
) {
    // Оголошення змінних для електронної пошти та пароля
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }

    // Оголошення змінної для повідомлення
    val messageState = remember { mutableStateOf<String?>(null) }
```

```
    val state by viewModel.state.collectAsState()

    // Встановлення теми за замовчуванням
    UrPostTheme(false) {
        val coroutineScope = rememberCoroutineScope()
        val snackbarHostState = remember { SnackbarHostState() }
        val scaffoldState = rememberScaffoldState(snackbarHostState =
snackbarHostState)
        val showDialog = remember { mutableStateOf(false) }

        // Відображення Snackbar з повідомленням
        LaunchedEffect(messageState.value != null) {
            val message = messageState.value

            if (message != null) {
                scaffoldState.snackbarHostState.showSnackbar(message)
                messageState.value = null
                viewModel.sendEvent(HostEvent.ClearMessages)
            }
        }

        // Оновлення повідомлення в залежності від стану
        LaunchedEffect(state) {
            messageState.value = state.message
        }

        Scaffold(scaffoldState = scaffoldState) {
            Column(
                modifier = Modifier
                    .fillMaxSize()
                    .background(colorResource(id =
R.color.background_color)),
                verticalArrangement = Arrangement.Center,
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                // Заголовок екрану авторизації
                Text(
                    text = "Authorization",
                    fontFamily = FONT_FAMILY,
                    style = MaterialTheme.typography.h5
                )

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

                // Введення електронної пошти
                OutlinedTextField(
                    value = email,
                    onValueChange = { email = it },
                    maxLines = 1,
                    label = { Text(fontFamily = FONT_FAMILY, text = "Email")
},
                    textStyle = TextStyle(
                        fontSize = 16.sp,
                        fontWeight = FontWeight.Light,
                        letterSpacing = 0.5.sp,
                        fontFamily = FONT_FAMILY
                    )
                )

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

                // Введення пароля
                OutlinedTextField(
```

```
                    value = password,
                    onValueChange = { password = it },
                    label = { Text(fontFamily = FONT_FAMILY, text =
"Password") },
                    visualTransformation = PasswordVisualTransformation()
                )

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

                // Кнопка для входу
                Button(
                    colors = ButtonDefaults.buttonColors(
                        backgroundColor = colorResource(id =
R.color.post_background)
                    ),
                    onClick = {
                        viewModel.sendEvent(HostEvent.SignIn(email = email,
password = password))
                        navigateTo(Screen.Main)
                    },
                    shape = RoundedCornerShape(20.dp)
                ) {
                    Text(fontFamily = FONT_FAMILY, color = Color.White, text
= "Log In")
                }

                // Кнопка для реєстрації
                Button(
                    colors = ButtonDefaults.buttonColors(
                        backgroundColor = colorResource(id =
R.color.post_background)
                    ),
                    onClick = {
                        showDialog.value = true
                    },
                    shape = RoundedCornerShape(20.dp)
                ) {
                    Text(fontFamily = FONT_FAMILY, color = Color.White, text
= "Registration")
                }
            }

            // Діалогове вікно для створення профілю користувача
            if (showDialog.value) {
                ProfileNameAlertDialog(
                    onPositiveButtonClick = { displayName ->
                        viewModel.sendEvent(
                            HostEvent.CreateUser(
                                email = email,
                                password = password,
                                displayName = displayName
                            )
                        )
                    },
                    navigateTo = navigateTo
                )
            }
        }
    }
}
```

Програмний код вспливаючого вікна при регістрації:

```
@Composable
fun ProfileNameAlertDialog(
    onPositiveButtonClick: (String) -> Unit,
    navigateTo: (route: Screen) -> Unit
) {
    // Оголошення змінної для імені профілю
    var profileName by remember { mutableStateOf("") }

    AlertDialog(
        backgroundColor = colorResource(id = R.color.background_color),
        onDismissRequest = { },
        title = {
            Text(
                textAlign = TextAlign.Center,
                fontFamily = FONT_FAMILY,
                text = "Enter your profile name"
            )
        },
        text = {
            OutlinedTextField(
                value = profileName,
                onValueChange = { profileName = it },
                label = { Text(fontFamily = FONT_FAMILY, text = "Profile
name") },
                modifier = Modifier.fillMaxWidth()
            )
        },
        confirmButton = {
            // Кнопка підтвердження
            Button(
                colors = ButtonDefaults.buttonColors(
                    backgroundColor = colorResource(id =
R.color.post_background)
                ),
                onClick = {
                    onPositiveButtonClick(profileName)
                    navigateTo(Screen.Main)
                },
                shape = RoundedCornerShape(20.dp)
            ) {
                Text(
                    text = "Ok",
                    fontFamily = FONT_FAMILY,
                    fontWeight = FontWeight.Bold,
                    color = Color.White
                )
            }
        },
        modifier = Modifier.padding(16.dp)
    )
}
```

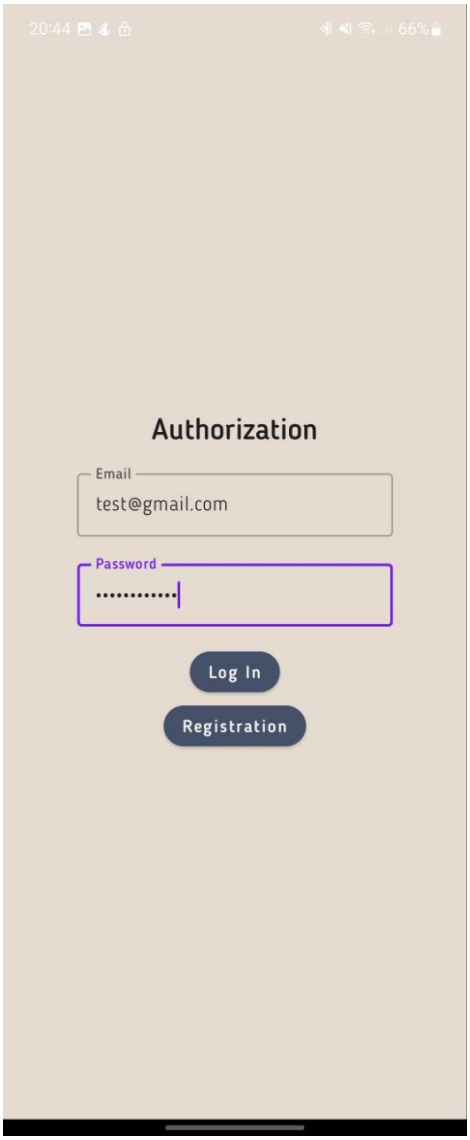


Рисунок 3.6 – Графічний вигляд екрану авторизації

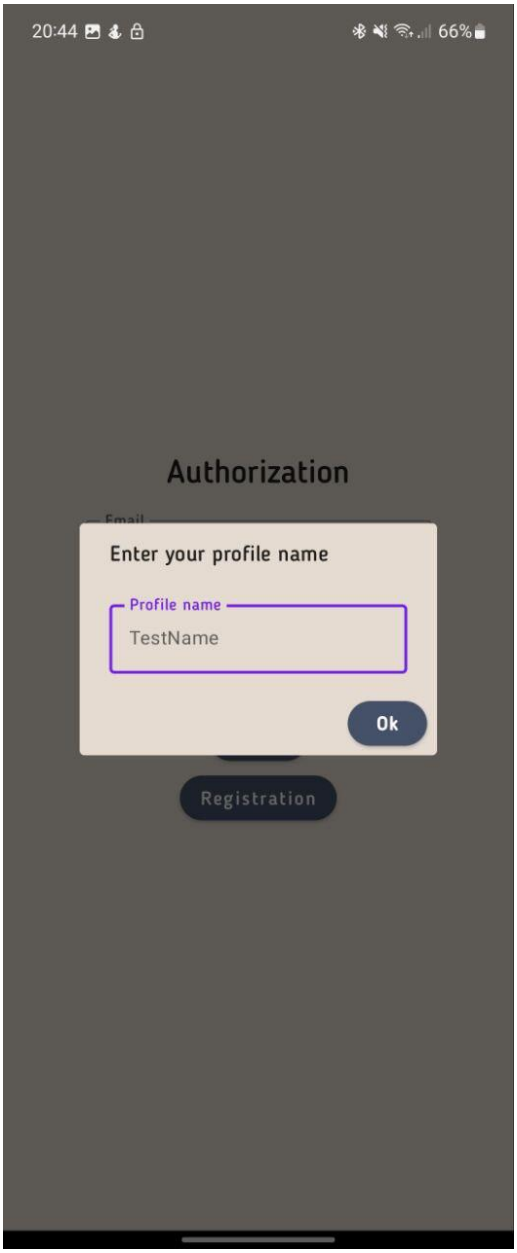


Рисунок 3.7 – Графічний вигляд вспливаючого меню при регістрації

При правильно вводі кредеталів, вас успішно авторизує в систему, що дасть доступ до написання постів особисто.

3.3 Розробка екрану детального перегляду поста

При кліку на пост на домашній сторінці, для детального перегляду, розроблено окремий екран, який показує пост у весь розмір.

Програмний код екрану для перегляду деталей поста (DetailsPostScreen):

```
@Composable
fun DetailsPostScreen(viewModel: HostViewModel) {

    val state by viewModel.state.collectAsState()

    val post = state.selectedPost

    // Завантаження зображення для посту
    val model = ImageRequest.Builder(LocalContext.current)
        .data(post?.image)
        .size(Size.ORIGINAL)
        .crossfade(true)
        .build()

    // Створення painter для асинхронного завантаження зображення
    val image = rememberAsyncImagePainter(model)

    SelectionContainer(Modifier.background(colorResource(id =
R.color.background_color))) {

        Card(
            modifier = Modifier
                .fillMaxSize()
                .padding(10.dp),
            shape = RoundedCornerShape(15.dp),
            elevation = 5.dp,
            backgroundColor = colorResource(id = R.color.post_background)
        )
        {
            Column(
                modifier = Modifier
                    .fillMaxSize()
                    .background(colorResource(id = R.color.post_background))
            ) {

                Image(
                    painter = image,
                    contentDescription = null,
                    modifier = Modifier
                        .fillMaxWidth()
                        .wrapContentHeight(),
                    contentScale = ContentScale.Crop
                )

                Text(
                    text = post?.title.toString(),
                    fontWeight = FontWeight.Bold,
                    modifier = Modifier.padding(bottom = 8.dp, start = 16.dp,
top = 12.dp),
                    color = Color.White,
                    fontFamily = FONT_FAMILY,
                    fontSize = 18.sp
                )
```

```
                    Text(
                        text = post?.text.toString(),
                        style = MaterialTheme.typography.body1,
                        modifier = Modifier
                            .padding(horizontal = 16.dp)
                            .fillMaxWidth(),
                        color = Color.White,
                        fontFamily = FONT_FAMILY,
                        fontWeight = FontWeight.Light
                    )

                    Text(
                        text = post?.author.toString(),
                        color = Color.White,
                        modifier = Modifier
                            .padding(end = 16.dp, bottom = 8.dp)
                            .fillMaxWidth(),
                        fontFamily = FONT_FAMILY,
                        fontWeight = FontWeight.Light,
                        fontSize = 12.sp,
                        textAlign = TextAlign.End
                    )
                }
            }
        }
    }
```

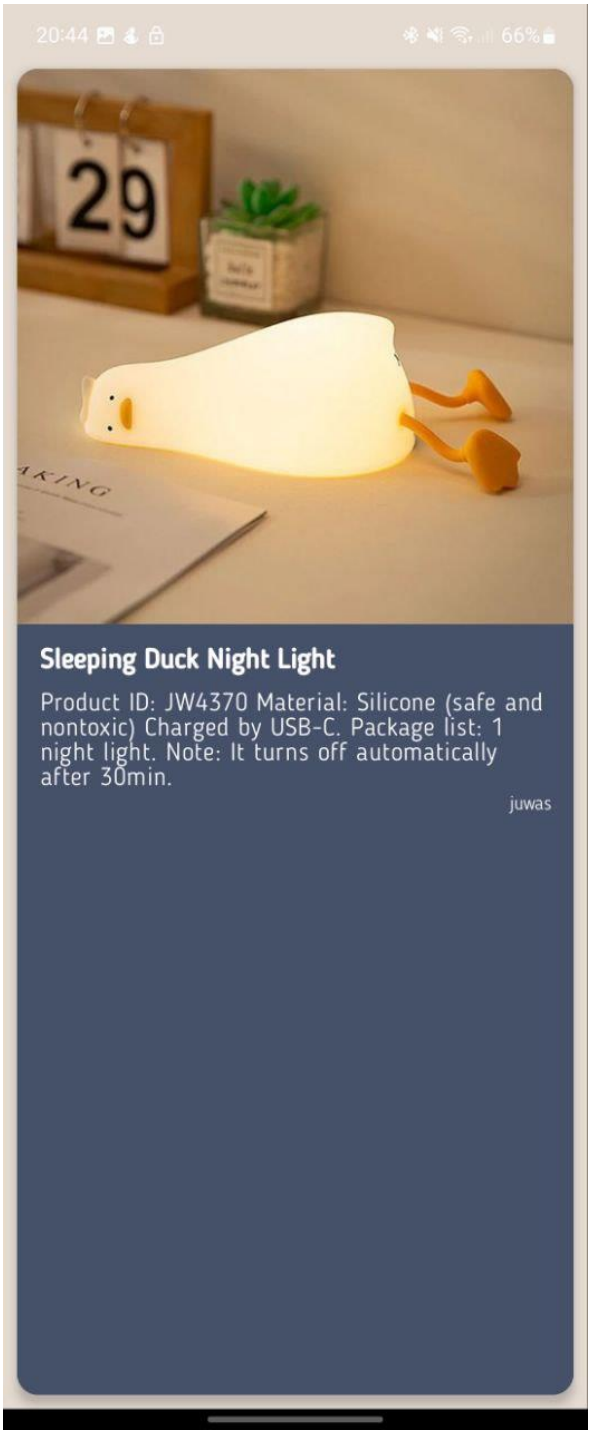


Рисунок 3.7 – Графічний вигляд екрану для перегляду деталей поста

При кліку на пост із списку, користувача перекидує на більш детальний перегляд у весь екран (показано на рисунку 3.7). В подальшому цей екран міститиме ще більше функціоналу.

3.4 Розробка екрану створення постів

Коли користувач авторизований у нього є змога створити власний пост у якому може виразити свою певну думку. Перейти до данного екрану можна при кліку на круглу кнопку з надписом «+».

Програмний код екрану для створення постів (CreatePostScreen):

```
@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun CreatePostScreen(
    viewModel: HostViewModel,
    navigateTo: (route: Screen) -> Unit,
) {
    var text by remember { mutableStateOf("") }
    var imageUri by remember { mutableStateOf<Uri?>(null) }
    var title by remember { mutableStateOf("") }

    Scaffold(
        // Конфігурація Scaffold для екрану створення посту
        modifier =
Modifier.background(colorResource(R.color.background_color)),
        topBar = {
            // Верхня панель додатка
            TopAppBar(
                backgroundColor = colorResource(id =
R.color.background_color),
                title = { Text(fontFamily = FONT_FAMILY, text = "Create
Post") },
                navigationIcon = {
                    IconButton(onClick = { /* Handle navigation back */ }) {
                        Icon(Icons.Default.ArrowBack, contentDescription =
null)
                    }
                }
            )
        },
        content = {
            Column(
                // Вертикальний контейнер для розміщення елементів на екрані
                modifier = Modifier
                    .fillMaxSize()
                    .background(colorResource(id = R.color.background_color))
                    .padding(16.dp),
                horizontalAlignment = Alignment.CenterHorizontally,
            ) {
                // Розділ для введення заголовку поста
                OutlinedTextField(
                    value = title,
                    onValueChange = { title = it },
                    label = { Text(fontFamily = FONT_FAMILY, text = "Title")
},
                    modifier = Modifier.fillMaxWidth(),
                    textStyle = TextStyle(
                        fontSize = 16.sp, // Розмір шрифта
                        fontWeight = FontWeight.Light, // Насиченість шрифта
                        letterSpacing = 0.5.sp, // Міжсимвольне відстань
                        fontFamily = FONT_FAMILY
```

```
                    )
                )

                // Розділ для введення тексту поста
                OutlinedTextField(
                    value = text,
                    onValueChange = { text = it },
                    label = { Text(fontFamily = FONT_FAMILY, text = "Enter
post text") },
                    modifier = Modifier.fillMaxWidth(),
                    textStyle = TextStyle(
                        fontSize = 16.sp, // Розмір шрифта
                        fontWeight = FontWeight.Light, // Насиченість шрифта
                        letterSpacing = 0.5.sp, // Міжсимвольне відстань
                        fontFamily = FONT_FAMILY
                    )
                )

                // Розділ для вибору зображення зі списку або додавання за
посиланням
                Text(
                    fontFamily = FONT_FAMILY,
                    text = "Choose image from list or add by link"
                )

                // Список зображень за замовчуванням
                val defaultImages = listOf(

"https://i.pinimg.com/564x/5c/e4/a0/5ce4a094df4c75914ea08054683a2ac9.jpg",

"https://i.pinimg.com/564x/37/e8/12/37e81220e2d7bc96e2327ca85e49dfc6.jpg",

"https://i.pinimg.com/564x/b9/b2/fc/b9b2fc8b9f4aef92d7d1106719d9c6ef.jpg",
                )

                LazyRow {
                    items(defaultImages.size) { index ->
                        Image(
                            painter =
rememberImagePainter(defaultImages[index]),
                            contentDescription = null,
                            modifier = Modifier
                                .padding(4.dp)
                                .size(72.dp)
                                .clip(MaterialTheme.shapes.small)
                                .clickable {
                                    // Встановлення URI зображення зі списку
за замовчуванням
                                    imageUri =
Uri.parse(defaultImages[index])
                                }
                        )
                    }
                }

                // Розділ для введення URL зображення
                OutlinedTextField(
                    value = imageUri?.toString() ?: "",
                    onValueChange = {
                        // Встановлення URI зображення з введеного посилання
                        imageUri = if (it.isNotEmpty()) Uri.parse(it) else
null
                    },
```

```
                    label = { Text(fontFamily = FONT_FAMILY, text = "Enter
image URL") },
                    modifier = Modifier.fillMaxWidth(),
                    textStyle = TextStyle(
                        fontSize = 16.sp, // Розмір шрифта
                        fontWeight = FontWeight.Light, // Насиченість шрифта
                        letterSpacing = 0.5.sp, // Міжсимвольне відстань
                        fontFamily = FONT_FAMILY // Ім'я шрифта
                    )
                )

                // Кнопка для створення поста
                Button(
                    colors = ButtonDefaults.buttonColors(
                        backgroundColor = colorResource(id =
R.color.post_background)
                    ),
                    onClick = {
                        // Створення об'єкту поста і відправлення події для
створення поста
                        val post = Post().apply {
                            this.title = title
                            this.image = imageUri.toString()
                            this.text = text
                        }
                        viewModel.sendEvent(HostEvent.WritePost(post))
                        navigateTo(Screen.Main)
                    },
                    shape = RoundedCornerShape(20.dp),
                    modifier = Modifier.padding(top = 10.dp)
                ) {
                    Text(
                        text = "Create Post",
                        fontFamily = FONT_FAMILY,
                        fontWeight = FontWeight.Bold,
                        color = Color.White
                    )
                }
            }
        }
    )
}
```

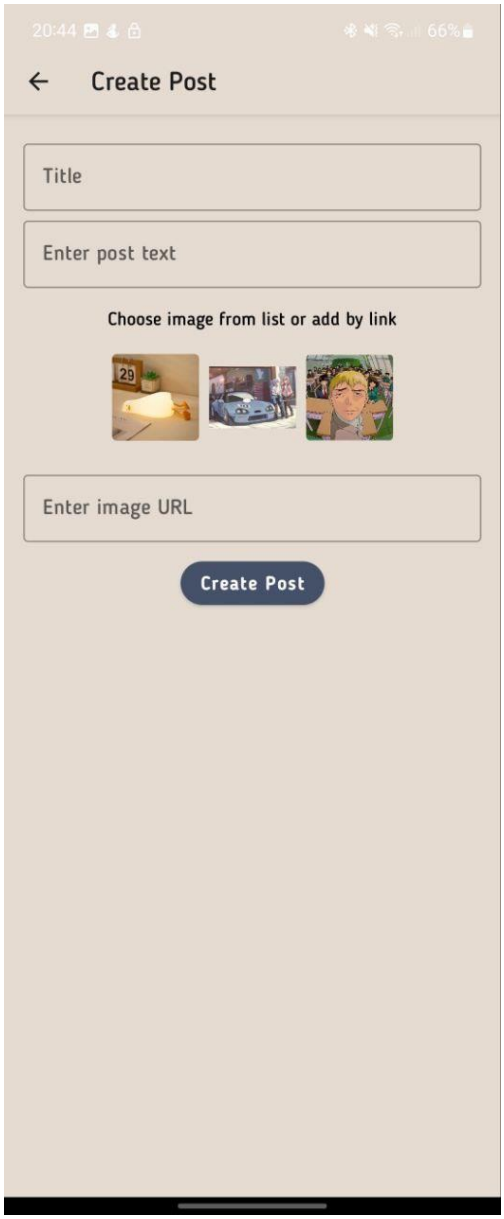


Рисунок 3.8 – Графічний вигляд екрану для створення постів

Якщо користувач ввів правильно всі необхідні поля, то натиснувши на кнопку «Create Post» його перекине на домашню сторінку зі всіма постами, де також буде його новостворенний.

3.5 Навігація між екранами

Для навігації між екранами створенно NavGraph у якому вписано всі можливі навігаційні моменти. Передаючи інтент данного классу виконується навігація на певних зворотніх викликах [4].

Основні направлення для екранів обернуті в запечатаний класс, що дозволяє уникати повторюючого коду.

Програмний код сутності для навігації екранів:

```
sealed class Screen(val route: String) {
    object Main : Screen(route = MAIN_ROUTE)
    object Authorization : Screen(route = AUTHORIZATION_ROUTE)
    object Details : Screen(route = DETAILS_ROUTE)
    object WritePost : Screen(route = WRITE_POST_ROUTE)
}
```

Програмний код винесення в контансти ідентифікаторів екрану:

```
const val MAIN_ROUTE = "main_screen"
const val AUTHORIZATION_ROUTE = "authorization_screen"
const val DETAILS_ROUTE = "details_screen"
const val WRITE_POST_ROUTE = "write_post_screen"
```

Програмний код composable функції для ініціалізації навігацій:

```
@Composable
fun NavGraph(hViewModel: HostViewModel) {
    val navController = rememberNavController()

    // Визначення навігаційного графа
    NavHost(navController = navController, startDestination =
Screen.Main.route) {
        // Екран Main
        composable(Screen.Main.route) {
            MainScreen(hViewModel) { screen ->
                navController.navigate(
                    route = screen.route
                )
            }
        }

        // Екран Authorization
        composable(Screen.Authorization.route) {
            AuthorizationScreen(
                navigateTo = { screen ->
                    navController.navigate(screen.route)
                },
                viewModel = hViewModel
            )
        }

        // Екран Details
        composable(Screen.Details.route) {
```

```
            DetailsPostScreen(viewModel = hViewModel)
        }

        // Екран WritePost
        composable(Screen.WritePost.route) {
            CreatePostScreen(viewModel = hViewModel) { screen ->
                navController.navigate(screen.route)
            }
        }
    }
}
```

Завдяки навігаційному графу, compose при виклику функції navigateTo перенаправлятиме нас у необхідний екран. Перемальовуючи всі елементи з нуля [4].

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було знайдено та проаналізовано ряд джерел, пов'язаних з розробкою мобільних соціальних мереж для Android. В ході дослідження були використані сучасні технології, такі як Firebase для зберігання та пошуку даних, Compose для реалізації графічного інтерфейсу та Coroutines для асинхронного програмування.

Основною метою розробки було створення функціональної та надійної соціальної мережі, де користувачі могли б спілкуватися, обмінюватися даними та встановлювати права доступу. Для досягнення цих цілей були використані потужні інструменти Firebase, Compose та Coroutines.

Firebase використовувався для зберігання та пошуку даних. Цей сервіс дозволяє швидко та безпечно працювати з хмарними базами даних та надає доступ до даних з різних пристроїв та місць.

Для реалізації графічного інтерфейсу мобільної соціальної мережі було використано Compose, який дозволяє створювати ефективні та гнучкі інтерфейси за допомогою декларативного підходу до програмування.

Для реалізації асинхронного програмування було використано Coroutines. Такий підхід дозволяє ефективно обробляти трудомісткі завдання та забезпечує безперебійну і швидку роботу додатку.

Завдяки використанню цих технологій була досягнута мета розробки соціальної мережі для мобільних пристроїв.

Підтримка Android. Користувачі можуть зручно спілкуватися, обмінюватися даними та встановлювати права доступу за допомогою зручного графічного інтерфейсу, а дані зберігаються та передаються надійно та безпечно завдяки Firebase.

Таким чином, використовуючи Firebase, Compose та Coroutines для розробки мобільних соціальних мереж для Android, ви можете створювати функціональні та надійні додатки, які задовольняють потреби користувачів у зручному спілкуванні та обміні інформацією.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jackson, W. (2019). JSON Quick Syntax Reference. Apress.
2. Офіційна документація Firebase [Електронний ресурс] // Firebase. (n.d.). – 2023. – Режим доступу до ресурсу: https://firebase.google.com/docs.
3. Android Programming: The Big Nerd Ranch Guide (4th Edition), 2020. – (Phillips B., Stewart C., Marsicano K.).
4. Jetpack Compose by Tutorials: Learning the New Android UI Toolkit, 2021 – Raywenderlich Tutorial Team.
5. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2019 – Martin, R. C.
6. Mastering Android Development with Kotlin: Deep dive into the world of Android to create robust applications with Kotlin. Packt Publishing, 2018 – Vasic, M.
7. Dependency Injection: Principles, Practices, and Patterns. Manning Publications, 2019 – van Deursen, S., Seemann, M.
8. Kotlin Coroutines: Asynchronous Programming Techniques. Addison-Wesley Professional, 2021 – Vivo M.
9. Pro Android with Kotlin: Developing Modern Mobile Apps. Apress, 2019 - Komatineni, S., Sarcar, V.
10. Reactive Programming with Kotlin: Learn how to build Android apps with RxJava 2, RxAndroid and Kotlin. Packt Publishing, 2018 - Hell, R.
11. Android Studio 4.0 Development Essentials - Kotlin Edition: Developing Android Apps Using Android Studio 4.0, Kotlin and Jetpack. Payload Media, 2020 - Smyth, N.
12. Mastering Kotlin: Learn Advanced Kotlin Programming Techniques to Build Apps for Android, iOS, and the Web. Packt Publishing, 2020 - Borysov, A.
13. Android Test-Driven Development by Tutorials: Learn Android TDD by Building Real-World Apps. Razeware LLC, 2021 - Raywenderlich Tutorial Team.

14. Kotlin Multiplatform Mobile: Going Beyond Android and iOS. Apress, 2021 - Kanchinadam, K.