

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЛУКІЯНЧУК ВЛАДИСЛАВ ВОЛОДИМИРОВИЧ

Допускається до захисту:

завідувач кафедри

інформаційних технологій,

доктор технічних наук, доцент

_____ Т. В. Нескородева

« ____ » _____ 20__ р.

**РОЗРОБКА ПЛАНУВАЛЬНИКА ЗАВДАНЬ З ГЕОЛОКАЦІЙНИМ
ПОРАДНИКОМ ДЛЯ СИСТЕМИ ANDROID**

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (бакалаврська) робота

Керівник:

Штовба С. Д., професор кафедри

інформаційних технологій

Оцінка: ____ / ____ / ____

(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

(підпис)

Вінниця – 2022

АНОТАЦІЯ

Лукіяничук В.В. Розробка планувальника завдань з геолокаційним порадиником для системи Android. Спеціальність 122 «Комп'ютерні науки», Спеціалізація «Сучасні інформаційні технології та програмування». Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У кваліфікаційній (бакалаврській) роботі досліджено розробка геолокаційного порадиника для системи Android. Показано як проектувалась база даних, створювався дизайн та розроблялась логічна частина.

Ключові слова: android, clean architecture, шаблони проектування, розробка дизайну.

49 с., 30 рис., 11 джерел, 2 дод.

ANOTATION

Lukiianchuk V.V. Development of a task planner for android with geolocation-based adviser. Specialty 122 «Computer Science», Specialization «Modern Information Technologies and Programming». Donetsk National University named after Vasil Stus, Vinnytsia, 2022.

This qualification (bachelor's) work examines the development of a geolocational messenger for the Android system. It shows how the database was designed, the design was created and the logical part was developed.

Key words: android, clean architecture, design patterns, design development.

49 p., 30 img., 11 sources, 2 дод.

ЗМІСТ

АНОТАЦІЯ.....	2
ВСТУП	4
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	5
1.1 Аналіз об’єкту дослідження.....	5
1.2 Аналіз конкурентів	5
1.3 Завдання та вимоги до програми.....	8
Висновки до розділу 1	8
РОЗДІЛ 2 СТВОРЕННЯ ДИЗАЙНУ ПРОГРАМИ.....	9
2.1 Обґрунтування інструментарія розробника	9
2.2 Дизайн екрану із списком завдань	9
2.3 Дизайн екрану із даними про завдання	12
Висновок до розділу 2	19
РОЗДІЛ 3 ПРОЕКТУВАННЯ БАЗИ ДАНИХ.....	20
3.1 Таблиці та зв’язки між ними.....	20
3.2 Моделі та об’єкти доступу до даних	21
Висновки до розділу 3	27
РОЗДІЛ 4 РОЗРОБКА ЛОГІЧНОЇ ЧАСТИНИ	28
4.1 Шаблон проектування	28
4.2 Допоміжні класи та бібліотеки.....	28
4.3 Розробка логіки для екрану із списком завдань	33
4.4 Розробка логіки для екрану з даними про завдання.....	38
4.5 Розробка логіки для екрану з даними про завдання.....	38
Висновки до розділу 4	42
ВИСНОВКИ.....	43
СПИСОК ЛІТЕРАТУРИ.....	44
КОД ФАЙЛУ «NotesFragment.kt»	45
КОД ФАЙЛУ «NotesInfoFragment.kt»	47
ДЕКЛАРАЦІЯ	49

ВСТУП

Метою даної роботи є допомога людині детально описати та структурувати поставлені задачі, і яким би не була кількість завдань, не забути про якесь із них. Тема є актуальною тому, що кожний день перед людиною виникає маса завдань, які мають бути виконані в тому чи іншому місці, в той чи інший час. Багато з нас могли забути про щось важливе, про те, що було десь записано, через заклопотаність чи просто погану пам'ять. Найбільш загальною є ситуація, коли проходячи повз магазин людина забуває купити потрібні речі.

Зважаючи на різні життєві ситуації, завданням даної роботи є:

- Створити можливість максимально детально описати завдання
- Мінімізувати можливість забуття завдання, та, відповідно, невиконання
- Спроекувати комфортний та функціональний користувацький інтерфейс

Програма буде написана для Android системи, так як більшість смартфонів у світі використовують саме цю платформу. У ході виконання даної роботи будуть здобуті необхідні навички для правильного написання Android програм, а також будуть покращені вміння створювати привабливий інтерфейс для користувача.

Об'єкт дослідження – розробка Android програми, а саме планувальника завдань із геолокаційним порадином.

Предмет дослідження – процес розробки планувальника завдань із геолокаційним порадином

Структура даної роботи: 4 розділи, 49 сторінок, 30 рисунків, 11 джерел, 2 додатки.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У розділі описано наскільки дана робота є актуальною, та проведений аналіз конкурентів.

1.1 Аналіз об'єкту дослідження

Щоденно у людей виникають завдання, наприклад: повертаючись з роботи сходити за покупками, забрати дітей із школи та інші. В сучасному світі людина є дуже заклопотаною, через що і може не згадати виконати якість із завдань. Щоб охопити якнайбільше ситуацій, в якій дана програма може використовуватись, був впроваджений геолокаційний поради́ник. Геолокаційний поради́ник – це особливість, яка була впроваджена у програму, для можливості нагадати користувачу про місце, яке він може відвідати, відповідно до його потреб. За допомогою push-сповіщень (спливаючі вікна), користувач буде повідомлений про потрібне йому місце. Тепер незважаючи на час та місце виконання завдання, користувач завжди буде мати змогу ефективно виконати всі поставленні завдання.

1.2 Аналіз конкурентів

На ринку Android-платформи існує безліч програм, в особливості і для виконання завдань. У підрозділі дана робота буде порівняна з популярним на Play Market – Google Завдання та Список Завдань.

- **Google завдання**

А) Привабливість та зручність інтерфейсу рис 1.1.

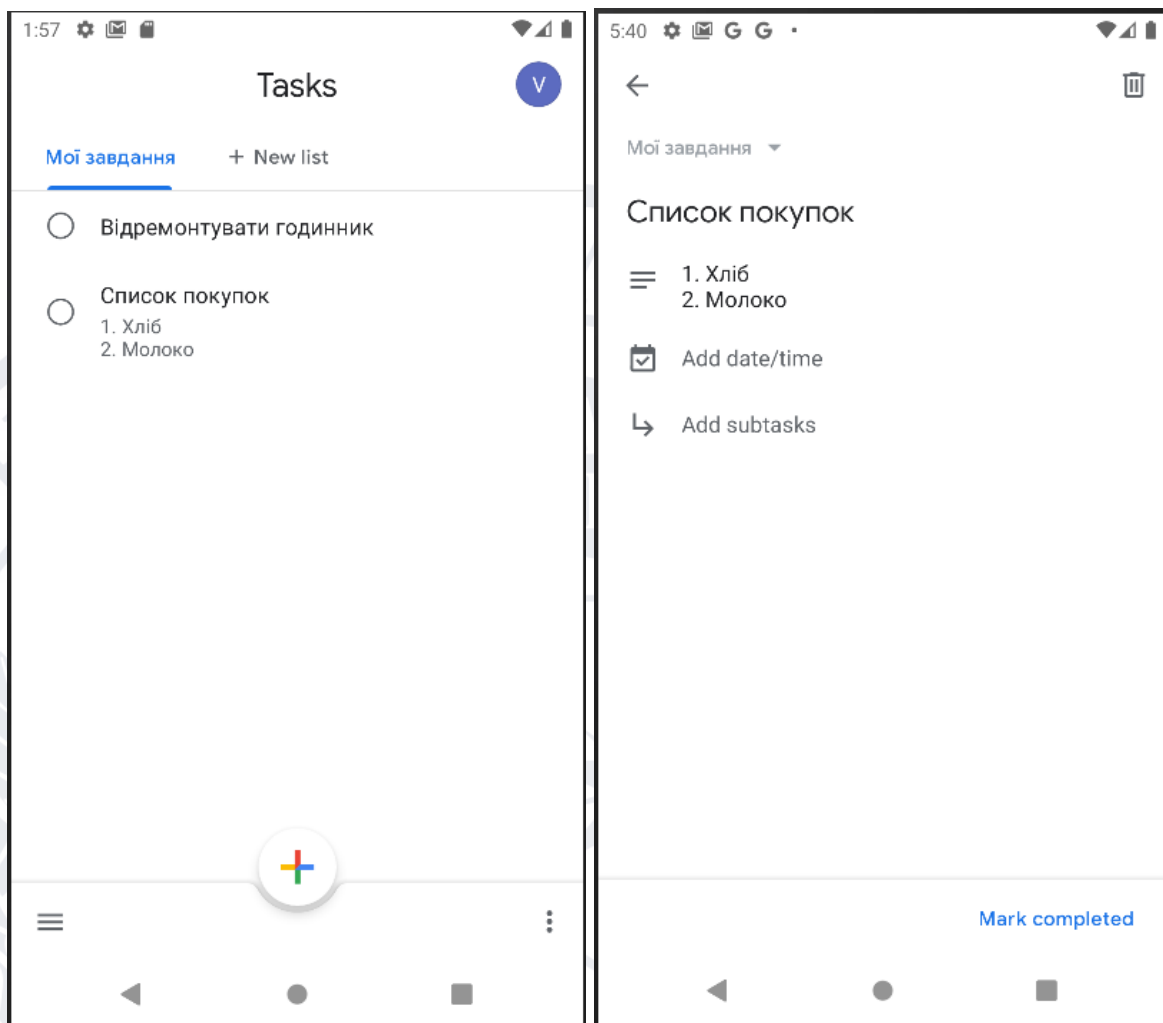


Рисунок 1.1 – Інтерфейс програми Google Завдання

Інтерфейс Google завдань має мало кольорів, що робить програму не дружньою для користувача. Список завдань зроблений дуже компактно, через що натискання на завдання – незручне.

Б) Функціонал

Функціонал програми дозволяє описати програму, та встановити час, коли воно має бути виконане. Недоліком є те, що немає можливості крім часу, встановити місце виконання завдань, що сильно обмежує користувача.

- **Список Завдань**

А) Привабливість та зручність інтерфейсу рис 1.2.

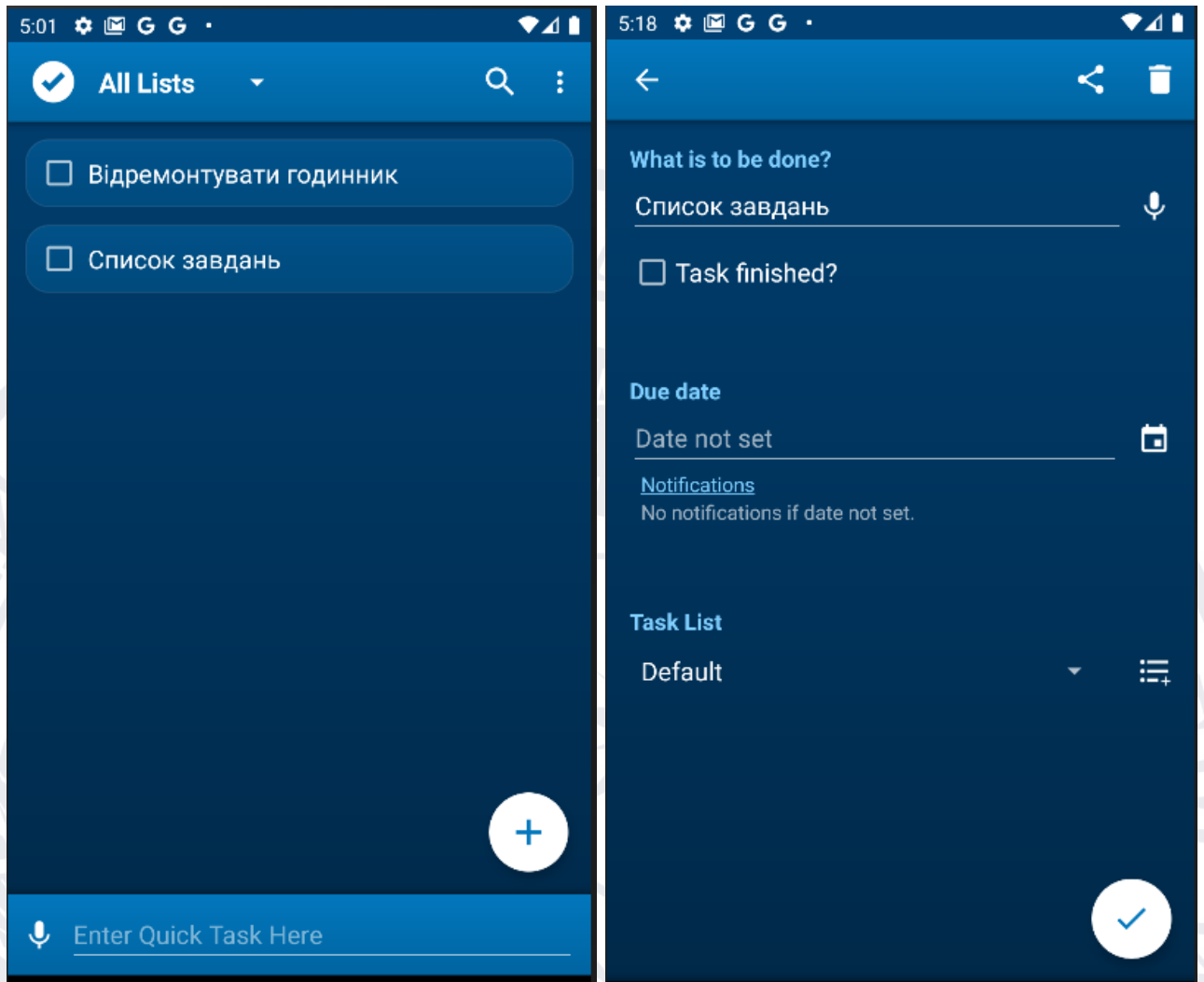


Рисунок 1.2 – Інтерфейс програми Список Завдань

Порівняно з програмою Google Завдання, даний список завдань більший, що робить натискання зручнішим. Вся програма містить синій колір та його відтінки так, як він відноситься до холодних кольорів, досвід взаємодії із програмою у користувача погіршується.

Б) Функціонал

Дана програма містить недоліки як і Google Завдання – неможливість встановити місце виконання завдання. Ще один недолік – це відсутність поля для детального описання завдання, що також накладає додаткові обмеження.

1.3 Завдання та вимоги до програми

Завданням є створити програму максимально комфортною, з точки зору функціоналу, привабливості та зручності, в якій будуть задовільнені всі потреби користувача. На підставі недоліків, які були сформовані у минулих підрозділах потрібно сформулювати наступні вимоги, яких програма має притримуватись:

- Інтерфейс програми має бути виконаний в теплих кольорах
- Можливість встановлення необхідних місць на карті
- Віджети мають бути зручними для натискання
- Можливість детально описати завдання

Висновки до розділу 1

Розглянуті програми для виконання завдань мають недоліки. Врахувавши вищенаведені правильна, дана програма буде більш необхідною для користувача.

РОЗДІЛ 2

СТВОРЕННЯ ДИЗАЙНУ ПРОГРАМ

В даному розділі описано як створюється інтерфейс для кожного екрану програми, а також, які інструменти для створення використовувались.

2.1 Обґрунтування інструментарію розробника

На підставі вимог до розробки, дизайн як і вся розробка програми створювалася в середовищі для розробки Android програм – Android Studio. Android studio – інтегроване середовище розробки для платформи Android. Дана програма адаптована для типових завдань, що вирішуються в процесі розробки застосунків. Для прискорення розробки програм представлена колекцій типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу. Для створення нестандартний інтерфейсі присутній майстер створення власних оформлень, що підтримує використання шаблонів.

2.2 Дизайн екрану із списком завдань

Дизайн був розробленим максимально зручним і зрозумілим для користувача. Завдання розміщені в дві колонки з оптимальною відстанню між ними. Нові завдання будуть додаватись на початку списку (у верхньому лівому куті). Завдання мають заголовок і опис. На задньому фоні розміщена картинка для покращення дизайну. Кнопка «додати завдання» розміщена у лівому нижньому куті. На тулбарі, у правому верхньому куті є кнопка для сортування завдань.

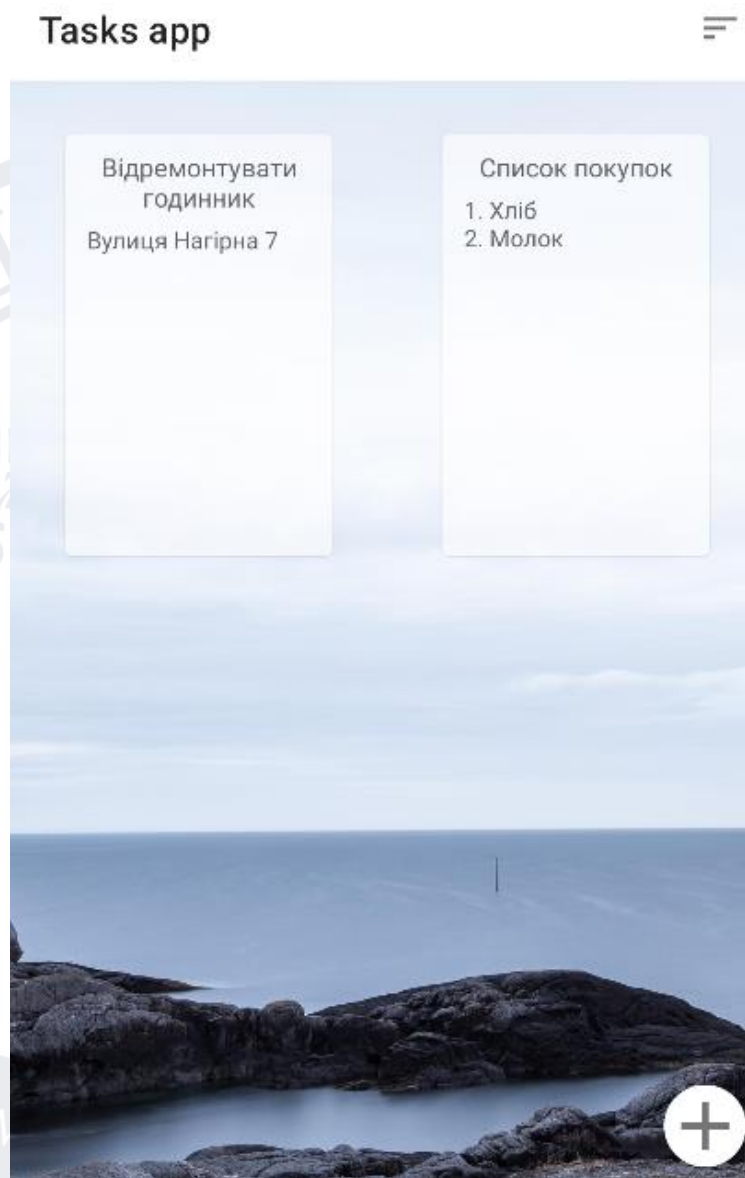


Рисунок 2.1 – Екран із списком завдань

Створений інтерфейс програми не містить кнопки «Видалити», тому ця можливість буде досягатися за допомогою більш довгого натискання на завдання, після чого з'явиться інший Toolbar, який буде мати необхідну іконку.

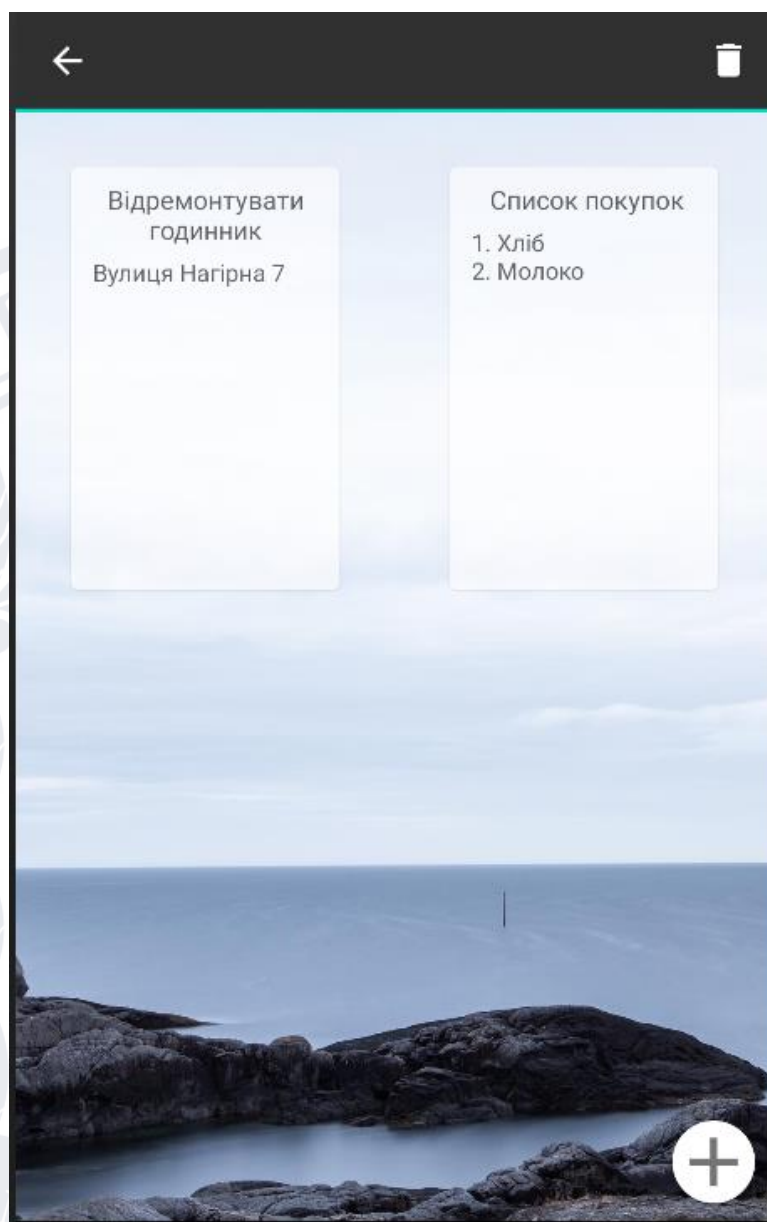


Рисунок 2.2 – Toolbar для видалення завдань

Дизайн який зображений на рис. 2.3. необхідно створити у layout файлах за допомогою XML розмітки. Можна виділити такі складові:

- Toolbar для списку задач - місце де розміщені заголовки і меню, робить навігацію простішою і дає швидкий доступ до частих операцій). Для цього був створений layout файл, який має назву `toolbar_list_note`, який містить один віджет – Toolbar. Також на панелі задача має бути іконка сортування, для зручності знаходження задач. Дану іконку необхідно створити в папці `drawable`, файл має назву `ic_sort`. Було створене

спеціальне меню - `menu_toolbar_list_note`, в яке передається дана іконка і присвоюється в тулбарі.

- Кнопка «додати завдання» - для можливості створювати нові завдання. Android SDK немає віджета із схожим інтерфейсом, тому дизайн був створений у папці `drawable` і має назву `add_note.xml`.
- Layout завдання списку – щоб створити елемент (задачі) в списку використовується `MaterialCardView` і в ньому описані додаткові дані про компоновку елементів інтерфейсу. Файл має назву `recycle_view_item_note`

Всі вище вказані елементи компоненти інтерфейсу мають бути поєднані в одному layout – `fragment_list_note`, після чого він буде встановлений у фрагмент. Для того, щоб можна було розміщувати компоненти, потрібно використовувати контейнер. Саме в нього будуть поміщатись компоненти. З самого початку ієрархії контейнер `FrameLayout`, так як елементи мають бути розміщені один на одному. Наступним вкладеним контейнером є `ConstraintLayout` [8] він дозволяє створювати гнучкі та масштабовані візуальні інтерфейси. В цьому контейнері вже розміщені компоненти інтерфейсу. Перший необхідно підключити `Toolbar`, далі створюється `RecyclerView` – призначений для роботи із списками, потім прив'язуються кнопка «додати» та фонові картинка.

2.3 Дизайн екрану із даними про завдання

Файл в якому створений дизайн має назву `fragment_note_info`. І має наступний дизайн який наводиться на рис. 1.4.

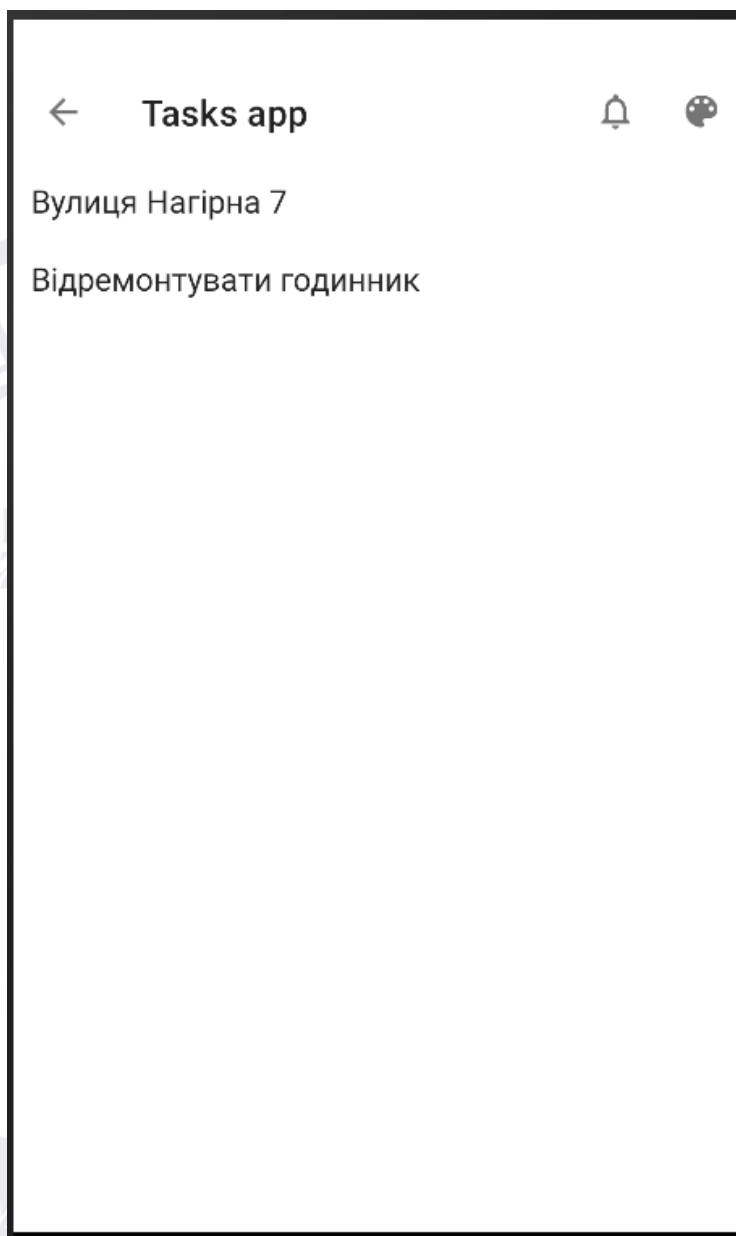


Рисунок 2.3 – Екран із деталями про завдання

Можна виділити такі складові:

- Заголовок та опис задачі. Знаючи, що екрани «додати задачу» та «змінити задачу» виглядають дуже схоже, в особливості ці два поля ідентичні на цих екранах. Є резон створити окремий layout файл – `note_info`, і в ньому створити ці поля. Після цих дій є можливість підключити файл і не повторювати код двічі.
- Toolbar має назву `toolbar_note_info`. В папці `drawable` створюються іконки дзвіночка – `ic_notification` та палітри – `ic_color`. В папці `menu`

потрібно створити меню, яке буде мати два пункти. Для створених пунктів в поле icon встановлюються файли іконок.

Натиснувши на палітру з'явиться діалогове вікно, в якому можна буде обрати колір завдання серед п'ятнадцяти можливих. Після чого колір Toolbar зміниться на обраний. Ця можливість була впроваджена, щоб покращити інтерфейс програми, а також користувач буде мати змогу встановлювати різний пріоритети для завдань. Натиснувши на Cancel діалогове вікно має закриватись. При натисканні на "Save" налаштування зберігаються.

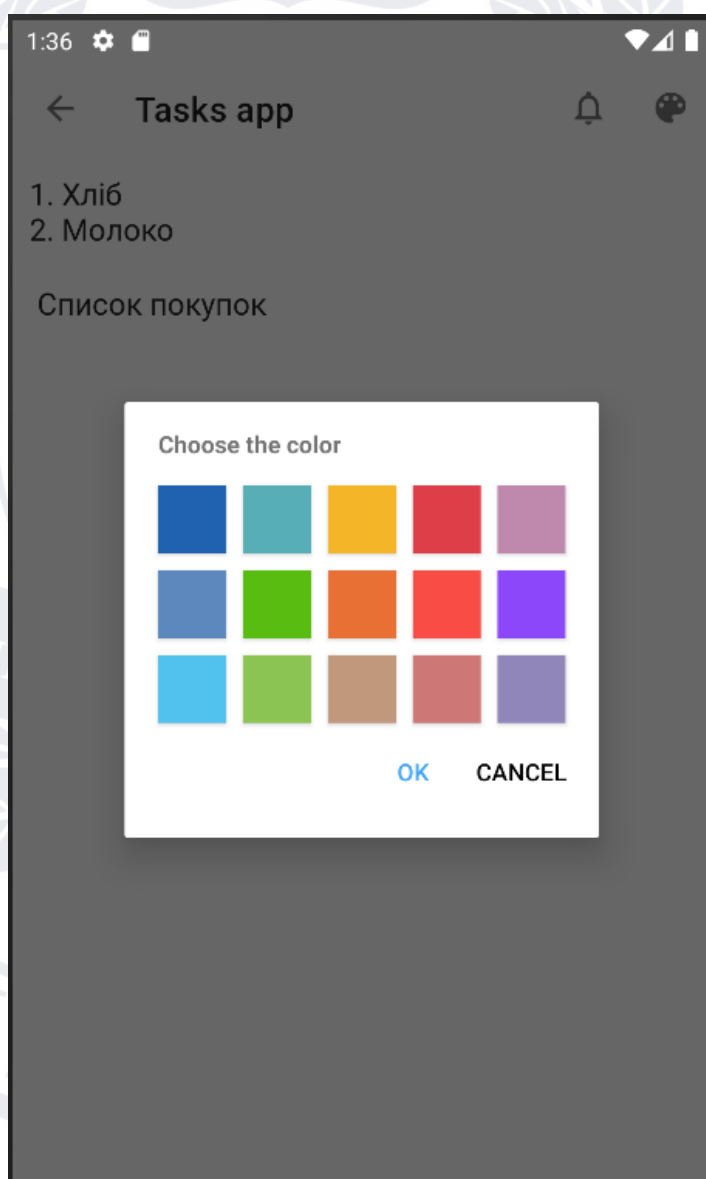


Рисунок 2.4 – Вікно вибору кольору

Натиснувши на іконку дзвіночка - відкриється діалогове вікно для налаштування додаткової інформації про місце та час виконання завдання. Цей дизайн створюється у файлі – `dialog_fragment_note_notification`.

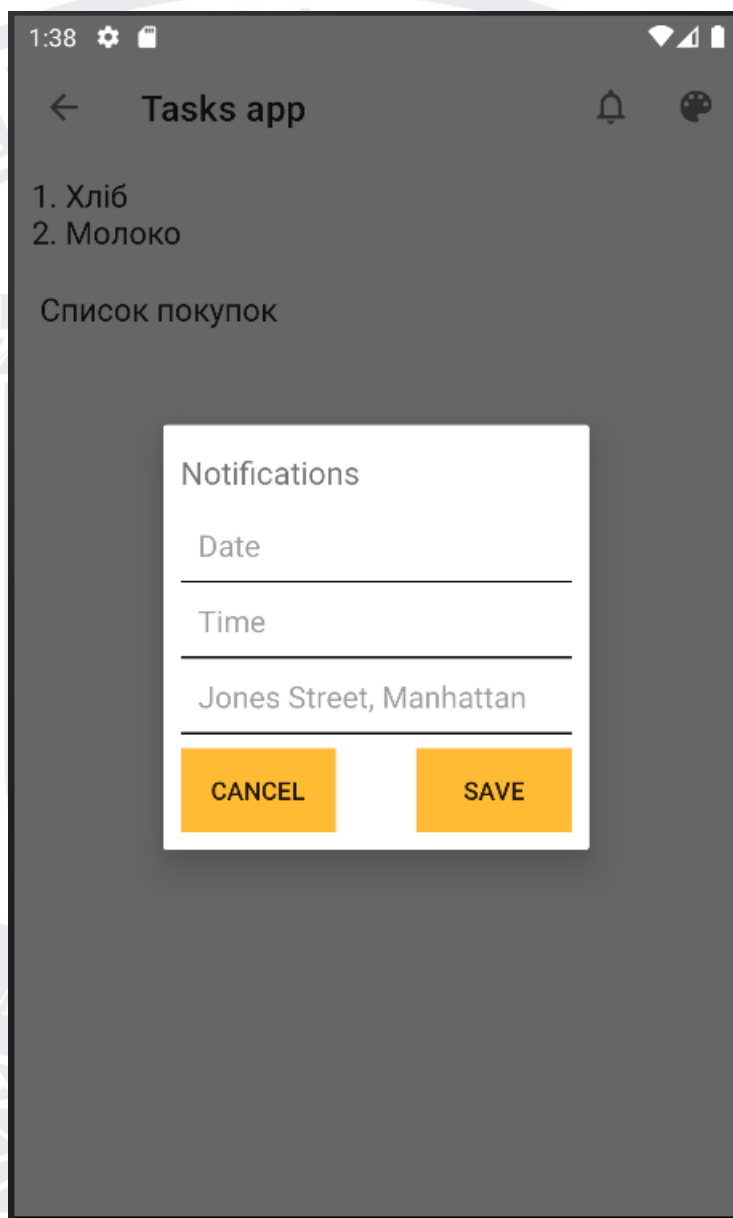


Рисунок 2.5 – Вікно обрання кольору

На кожне поле є можливість натиснути, після чого відобразиться відповідне діалогове вікно.

1. Після натискання на поле date з'явиться діалогове вікно для вказання дати виконання завдання. Файл має назву – `fragment_date_picker`. Є

можливість обрати день, рік та місяць. Після натискання «Ok» чи «Cancel» діалогове вікно буде закрите.

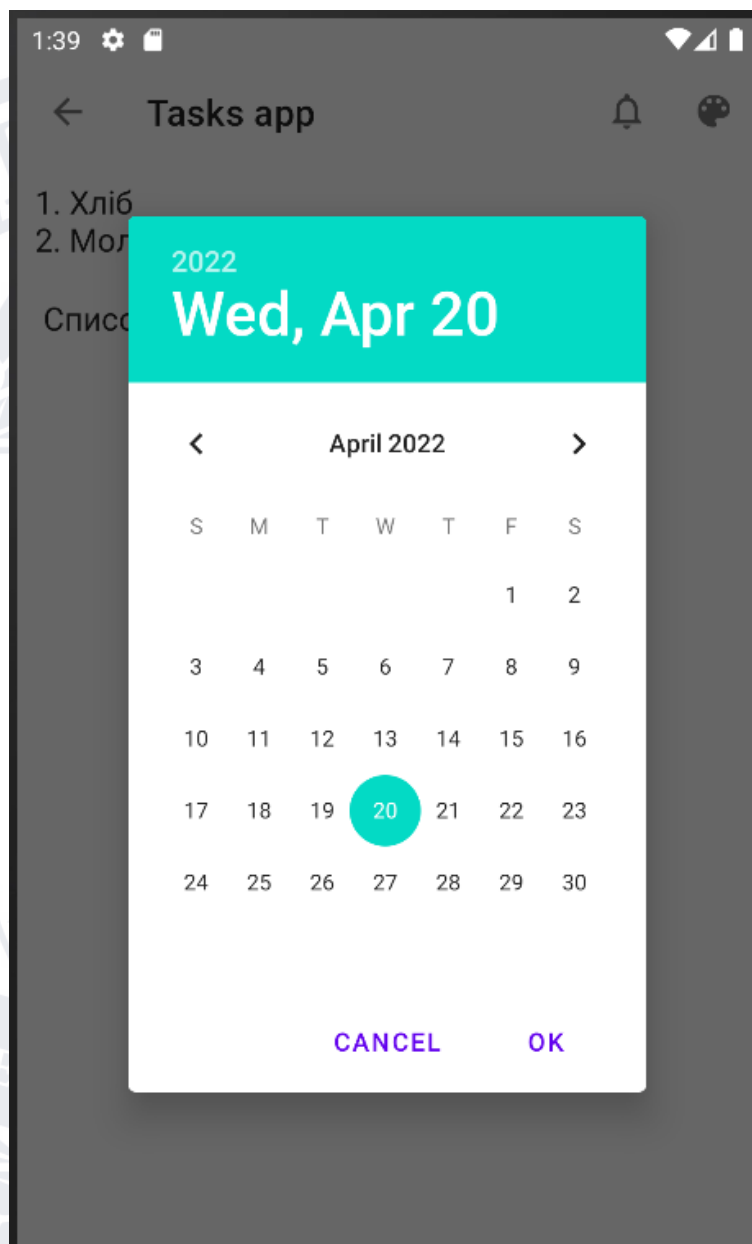


Рисунок 2.6 – Вікно обрання дати

- Після натискання на поле time з'явиться діалогове вікно для вказання часу, коли має бути виконане завдання. Є можливість обрати години та хвилини. Після натискання «OK» чи «Cancel» діалогове вікно буде закрите.

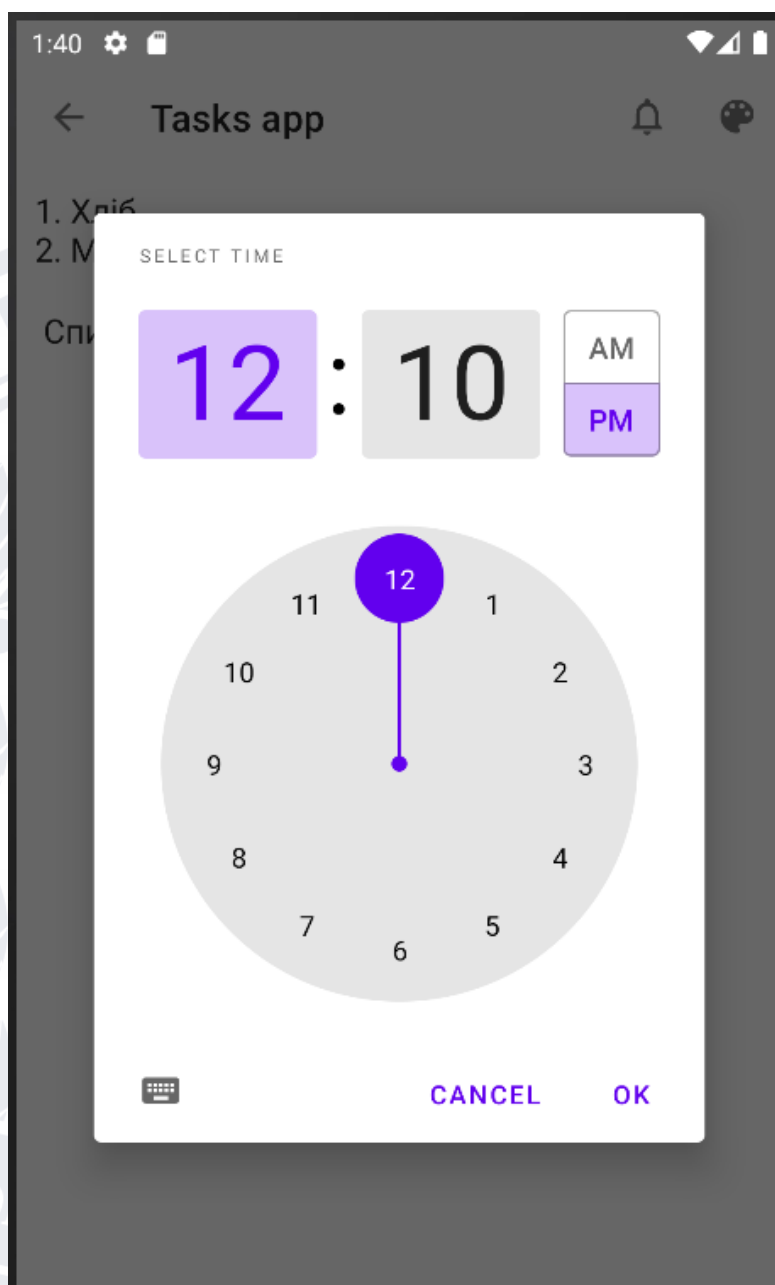


Рисунок 2.7 – Вікно обрання часу

- Після натискання на поле для встановлення місця відкриється екран картою і буде запрошений дозвіл на отримання теперішнього розміщення користувача. Якщо користувач не дасть дозвіл на його місце знаходження – карта закриється і юзер буде повернутий на екран з діалоговим вікном встановлення нагадування. На карті, у верхньому правому куті розміщена кнопка, при натисканні якої карта приблизиться до локації користувача. Ще одна можливість – це ставити

маркери на карті, коли користувач буде знаходитись поблизу одного із них йому будуть надходити повідомлення.

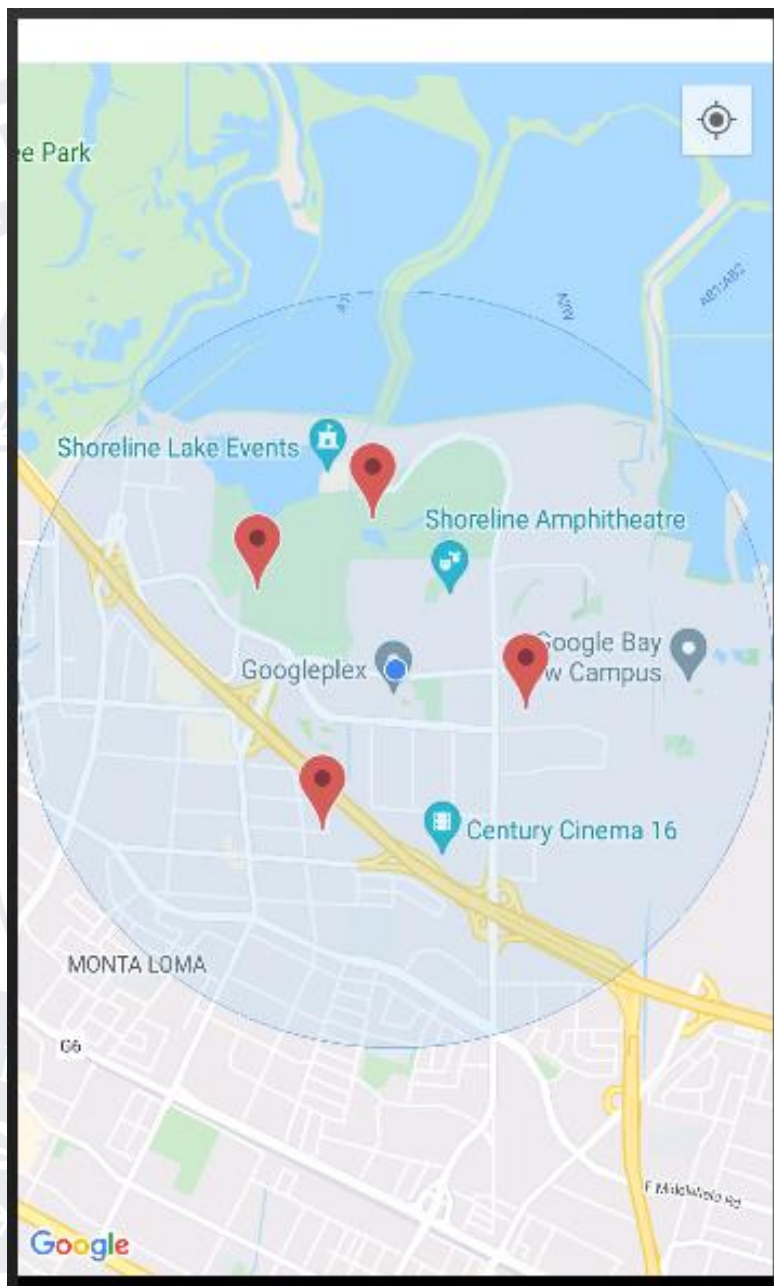
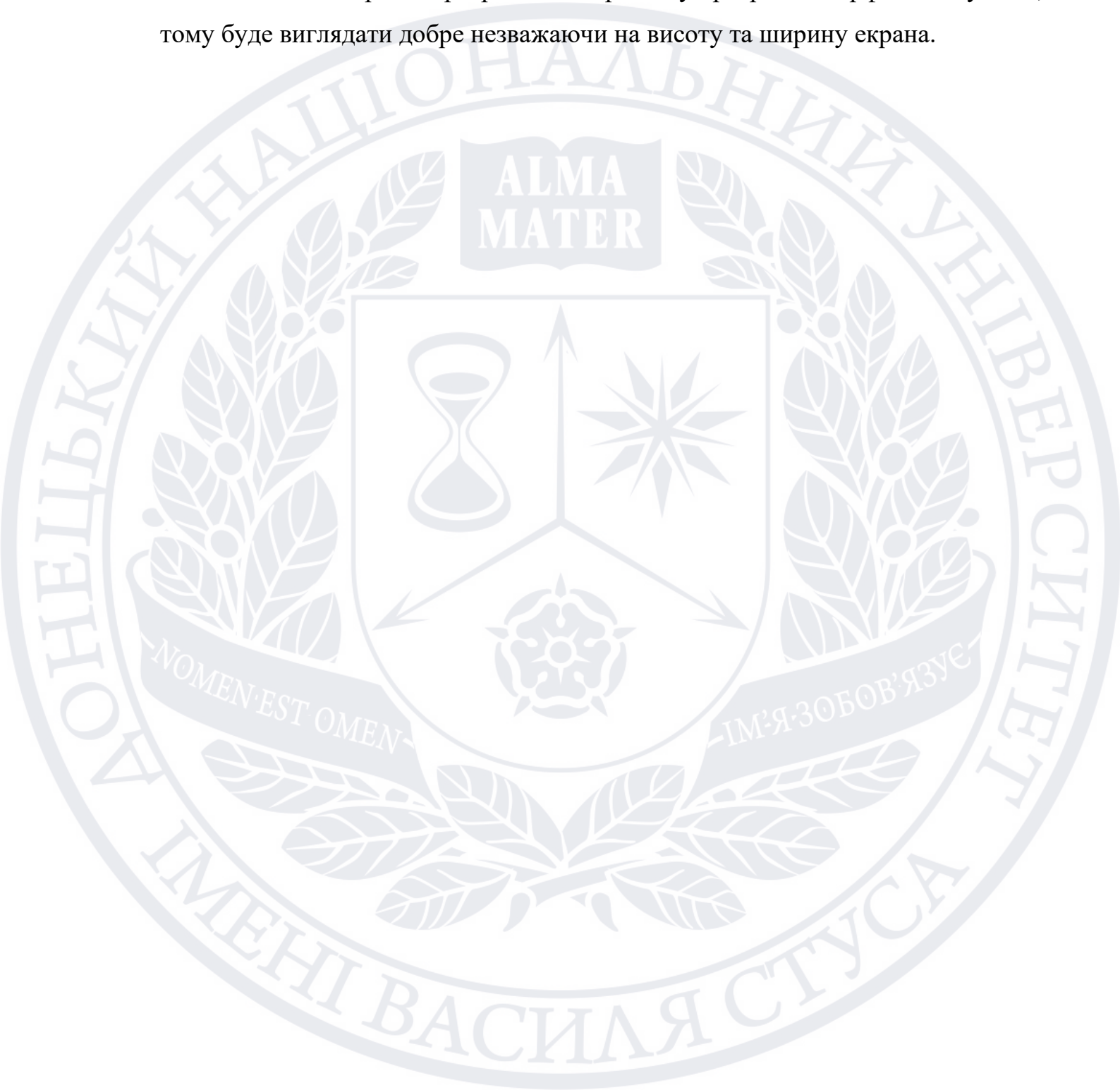


Рисунок 2.8 – Активність з картою

Якщо встановлені і дата і місце то нотифікації будуть з'являтись незалежно чи користувач знаходиться біля встановленого ним місця або часу.

Висновок до розділу 2

За допомогою необхідного інструментарію розробника був розроблений дизайн для всіх екранів програми. Створений у програмі інтерфейс є гнучким, тому буде виглядати добре незважаючи на висоту та ширину екрана.



РОЗДІЛ 3

ПРОЕКТУВАННЯ БАЗИ ДАНИХ

Android підтримує одну із найпоширеніших система управління базами даних – SQLite. SQLite – це база даних SQL з відкритим кодом, яка зберігає дані в текстовому файлі на пристрої [7]. SQLite підтримує всі функції реляційної бази даних. Щоб отримати доступ до цієї бази не потрібно встановлювати ніякі з'єднання, наприклад JDBC. Натомість, щоб спростити роботу з SQLite в даній програмі використовується Room. Room – це спосіб зберігання даних програми в Android-програмі. Не менш важливим є те, що Room є частиною нової Android Architecture, група бібліотек від Google.

3.1 Таблиці та зв'язки між ними

Спочатку необхідно виділити дані, які мають зберігатися в базі даних. Кожна задача буде мати опис, назву, колір, який буде вказувати на важливість задачі чи просто для покращення інтерфейсу користувача. Тому ці дані потрібно зібрати в одній таблиці.

Note	
title	String
descripton	String
color	Int

Рисунок 3.1 – таблиця Note

Також має бути можливість надходжень повідомлень користувачу про місце, яке він обрав на карті, дату та час. Для цього потрібно створити спеціальний клас Notification, він буде вкладений в таблицю Note. Notification буде мати наступні поля: minute, hour, day, month, year. Також буде потрібно створити клас Market, щоб зберігати точки на карті, які встановить користувач.

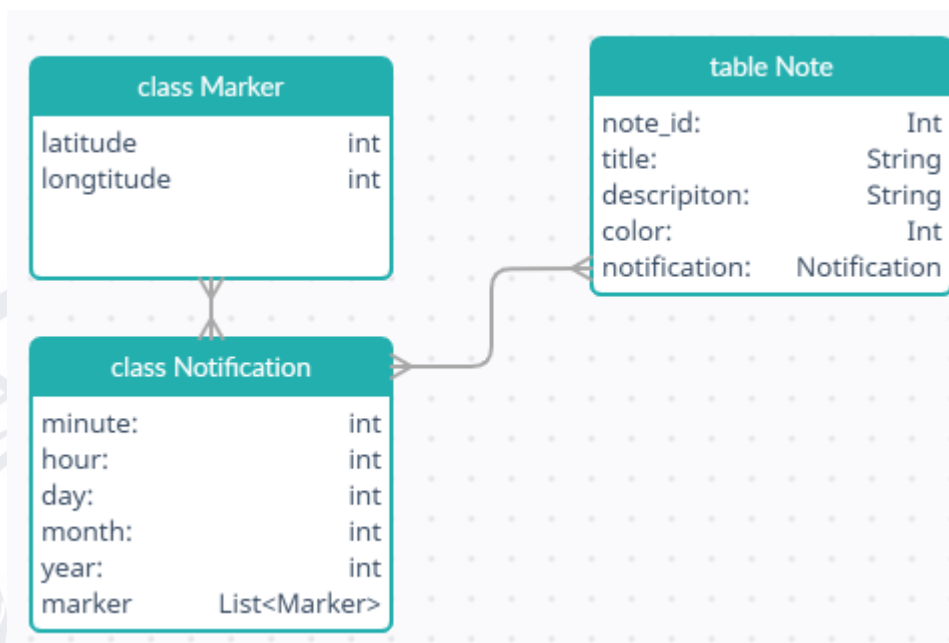


Рисунок 3.2 – Зв'язки між таблицями

3.2 Моделі та об'єкти доступу до даних

в Room Існує три основних компонента:

1. Клас бази даних, який має базу і є основною точкою для доступу для базового з'єднання із збереженими даними програми.
2. Об'єкти даних, які представляють таблиці в базі даних.
3. Об'єкти доступу до даних (DAO), які надають методи, які програми буде використовувати для запиту, оновлення, вставки та видалення в базі даних.

Клас бази даних дає програмі екземпляри DAO, зв'язаних з цією базою даних. В свою чергу, програма використовує базу DAO для отримання даних з бази даних у вигляді об'єктів даних [11].

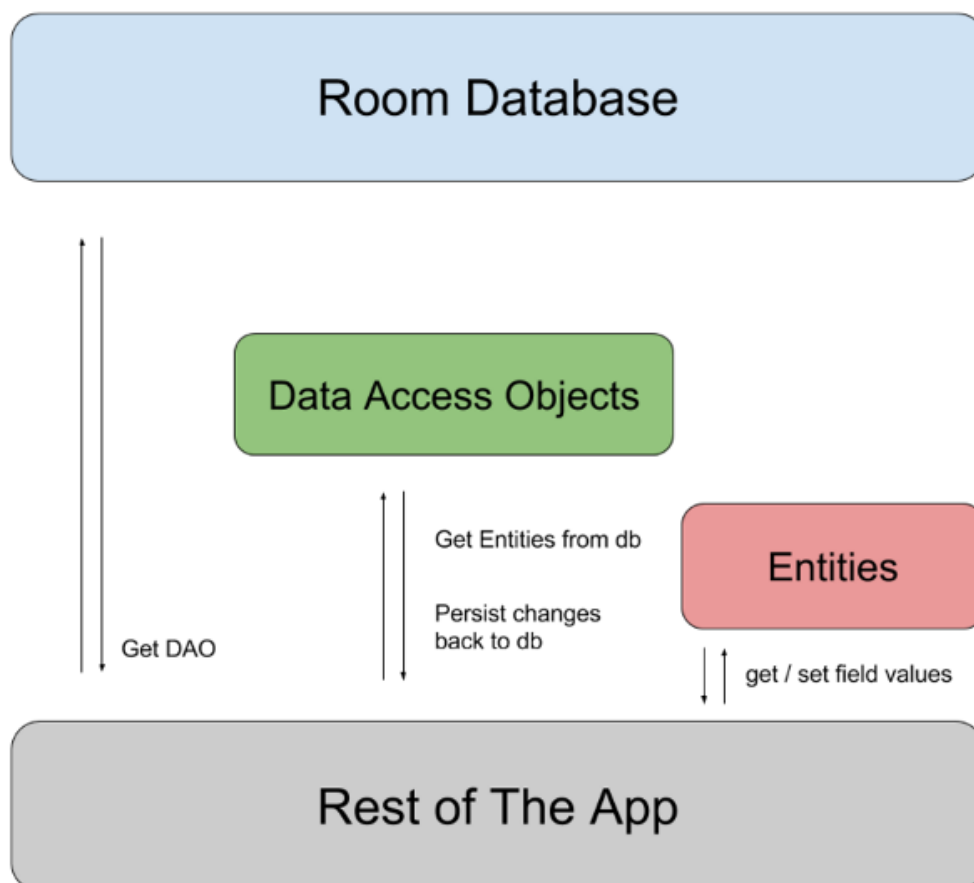


Рисунок 3.3 – компоненти бази даних

Об'єкти даних також називають моделями, їх є декілька в даній програмі. Першим кроком створюється модель Note, вона має такі ж поля які представлені на рис. 3.4.

```

@Entity(tableName = "note_table")
data class Note (
    @PrimaryKey(autoGenerate = true) var noteId: Int = 0,
    @ColumnInfo(name = "title") val title: String = "",
    @ColumnInfo(name = "description") val description: String = "",
    @ColumnInfo(name = "color") val color: Int? = 0,
    @Embedded var notification: Notification? = null
)
  
```

Рисунок 3.4 – class Note

Клас Notification з двома додатковими методами для форматування дати та часу, клас анотований Serializable для того, щоб його можна було правильно зберегти, він зберігається як послідовність байтів.

```
@Serializable
class Notification(var hour: Int = 20, var minute: Int = 10) {
    var year: Int? = 2020
    var month: Int? = 15
    var day: Int? = 25

    fun dateString(): String {
        return if (month != null) {
            String.format(
                "${year.toString()}. " +
                "$month.toString()}. " +
                day.toString()
            )
        } else {
            "2020.10.10"
        }
    }

    fun timeString(): String {
        return String.format(
            "$hour" +
            minute.toString()
        )
    }
}
```

Рисунок 3.5 – classNotification

Після створення необхідних моделей (таблиць) наступним кроком необхідно створити методи для доступу до них. Потрібно створити інтерфейс який буде підписаний анотацією @Dao, а його методи анотацією @Query. В дужках створюється стрічка із запитом [10].

Потрібно створити наступні методи:

1. getNotes – Створює запит в таблицю note_table, з якої буде повернутий список завдань.

```
@Query( value: "SELECT * FROM note_table")
suspend fun getNotes(): List<Note>
```

Рисунок 3.6 – метод getNotes

2. `getNoteById` – в аргументах передається ідентифікатор завдання, використовуючи його, здійснюється пошук по всім запитам з таблиці `note_table` рис.

```
@Query( value: "SELECT * FROM note_table WHERE noteId = :id")
suspend fun getNoteById(id: Int): Note
```

Рисунок 3.7 – метод getNoteById

3. `deleteNote` – в параметрах функції передається завдання, йде пошук по всій таблиці, якщо воно існує то буде видалене. Використовується призначена для видалення анотація `@Delete`.

```
@Delete
fun deleteNote(note: Note)
```

Рисунок 3.8 – метод deleteNote

4. `insertNote` – в аргументах передається масив значень, він буде доданий в таблицю `note_table`. Використовується спеціальна анотація `@Insert` для вставлення в таблицю нових значень.

```
@Insert
suspend fun insertNote(vararg note: Note)
```

Рисунок 3.9 – метод insertNote

5. `delete` – видалляє всі записи з таблиці.

```
@Query( value: "DELETE FROM note_table")
fun delete()
```

Рисунок 3.10 – метод delete

6. `getColor` – в аргументах приймає ідентифікатор завдання, після знаходження відповідного повертається `Int` значення, яке відповідає кольору рис.

```
@Query( value: "SELECT color FROM note_table")
suspend fun getColor(): Int
```

Рисунок 3.11 – метод `getColor`

7. `updateDescription` – в аргументах приймає строку – опис завдання та ідентифікатор завдання. Якщо минулий опис існував – він буде переписаний на новий, інакше буде вставлене нове значення.

```
@Query( value: "UPDATE note_table SET description = :description WHERE noteId = :id")
suspend fun updateDescription(description: String, id: Int)
```

Рисунок 3.12 – метод `updateDescription`

8. `updateColor` – в аргументах приймає строку – колір та ідентифікатор завдання. Якщо раніше був встановлений колір – він буде переписаний на новий, інакше буде вставлене нове значення.

```
@Query( value: "UPDATE note_table SET color = :color WHERE noteId = :id")
suspend fun updateColor(color: Int, id: Int)
```

Рисунок 3.13 – метод `updateColor`

9. `updateTitle` – в аргументах приймає строку – заголовок завдання та ідентифікатор завдання.

```
@Query( value: "UPDATE note_table SET title = :title WHERE noteId = :id")
suspend fun updateTitle(title: String, id: Int)
```

Рисунок 3.14 – метод `updateTitle`

10. `updateDate` – в аргументах приймає рік, місяць, день та ідентифікатор завдання. використовуючи ідентифікатор здійснюється пошук по всім

запитам з таблиці `note_table` та у відповідні стовбці таблиці встановлюються нові значення.

```
@Query( value: "UPDATE note_table SET description = :description WHERE noteId = :id")
suspend fun updateDescription(description: String, id: Int)
```

Рисунок 3.15 – метод `updateDate`

11. `updateTime` – в аргументах приймає години, хвилини та ідентифікатор завдання. Використовуючи ідентифікатор здійснюється пошук по всіх запитам з таблиці `note_table` та у відповідні стовбці таблиці встановлюються нові значення.

```
@Query( value: "UPDATE note_table SET hour = :hour, minute = :minute WHERE noteId = :id")
suspend fun updateTime(hour: Int, minute: Int, id: Int)
```

Рисунок 3.16 – метод `updateTime`

Останнім кроком створюється база даних. Наступний код визначає клас `NoteDatabase` для збереження бази даних. `NoteDatabase` визначає конфігурацію бази даних і служить основною точкою доступу програми до збереження даних. Цей клас відповідає наступним необхідним умовам:

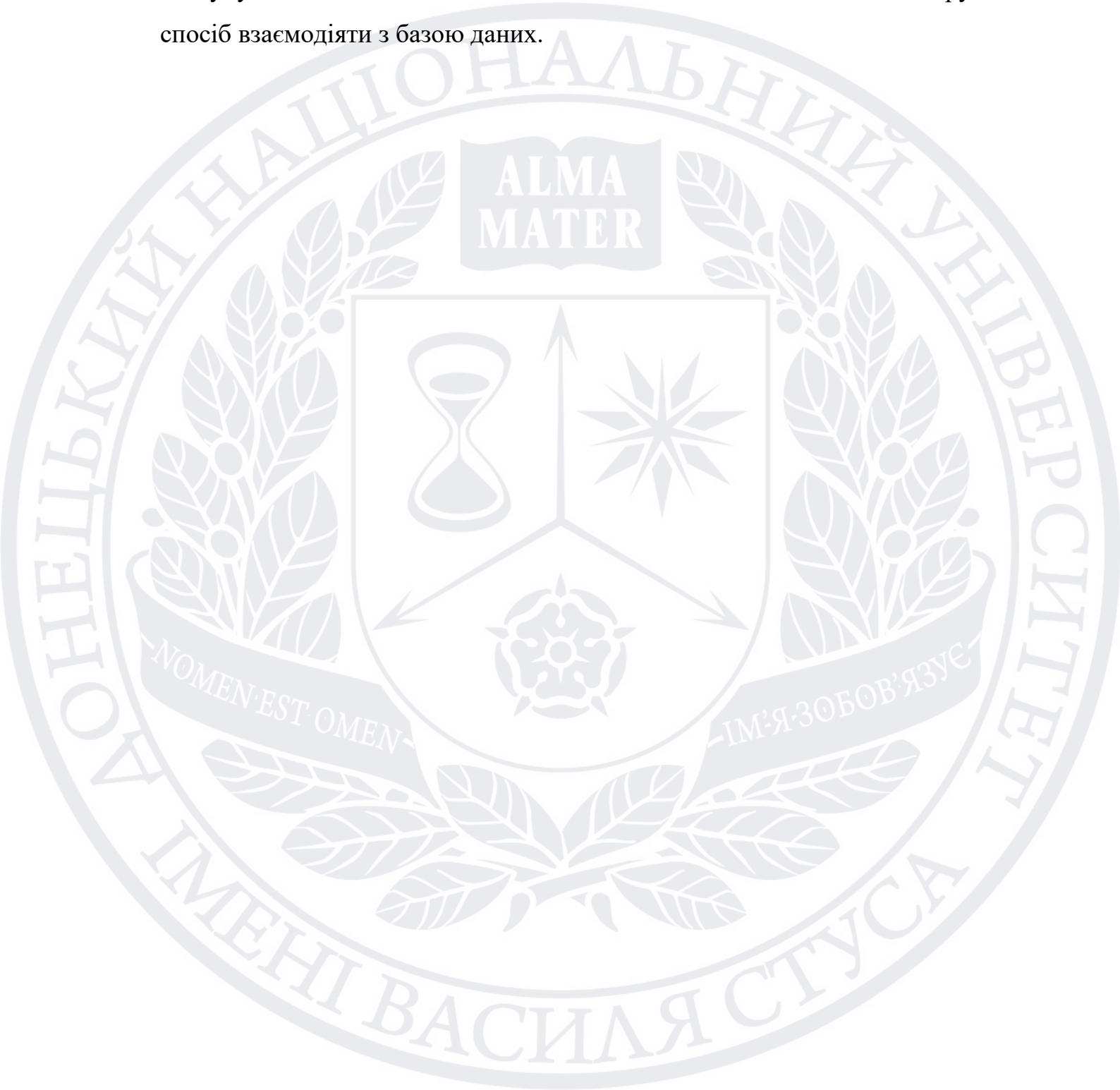
1. Клас має бути анотований анотацією `@Database` який включає масив сутностей, в якому перерахуванні всі об'єкти даних
2. Клас має бути абстрактним та розширювати `RoomDatabase`. Для кожного класу DAO, зв'язаного з базою даних клас бази даних має
3. встановлювати абстрактний метод, який має нульові аргументи і повертає екземпляр класу DAO.

```
@androidx.room.Database(entities = [Note::class], version = 8)
abstract class NotesDatabase : RoomDatabase() {
    abstract fun noteDao(): NoteDao
}
```

Рисунок 3.17 – class `NoteDatabase`

Висновки до розділу 3

Були створені моделі, а також зв'язки між ними. Були створені методи доступу до цих таблиць та база даних. За допомогою Room з'явився зручний спосіб взаємодіяти з базою даних.



РОЗДІЛ 4

РОЗРОБКА ЛОГІЧНОЇ ЧАСТИНИ

В цьому розділі буде описано як розроблялася логічна частина програми, Буде встановлена взаємодія з дизайном та базою даних. Кожний підрозділ вирішує свою задачу. Тут буде описана заключна робота після виконання якої програма буде готова до використання.

4.1 Шаблон проектування

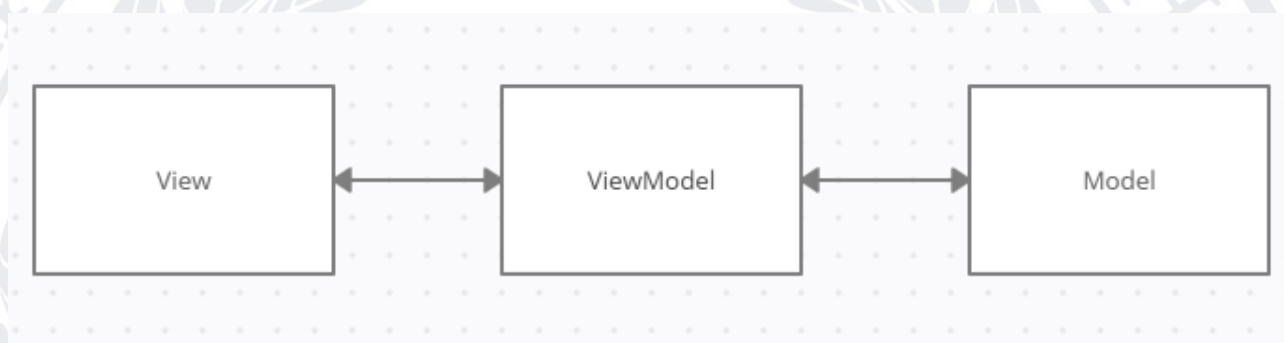


Рисунок 4.1 – Шаблон проектування MVVM

Представлення – містить структуроване визначення того, що користувачі отримають на екранах. Тут не міститься ніякої логіки.

Модель представлення – це компонент який зв’язує модель та представлення.

Відповідає за передачу даних. Модель – це рівень бізнес-даних і він не зв’язаний ні з яким графічним представленням.

Відповідно цьому шаблону проектування буде так сконструйований кожний екран програми.

4.2 Допоміжні класи та бібліотеки

В даному підрозділі буде описано, які бібліотеки Android Jetpack використовуються, а також створені базові класи, та як їх правильно використовувати. Android Jetpack – це набір бібліотек, направлених на то, щоб допомогти розробнику легко писати високоякісні програми, підтримуючи більш старі версії Android.

Fragment Navigation

Архітектура компонентів Navigation дозволяє спростити реалізацію навігації між призначення (destinations) у програмі [2]. За замовчуванням, Navigation підтримує фрагменти (Fragments) та активності (Activities) в якості екранів призначення. Набір екранів призначення має назву граф навігації (navigation graph) програми. Крім екранів призначення на графі навігації є з'єднання між ними, які називаються діями (actions). папка navigation, яка розміщена в папці з ресурсами містить графи навігації.

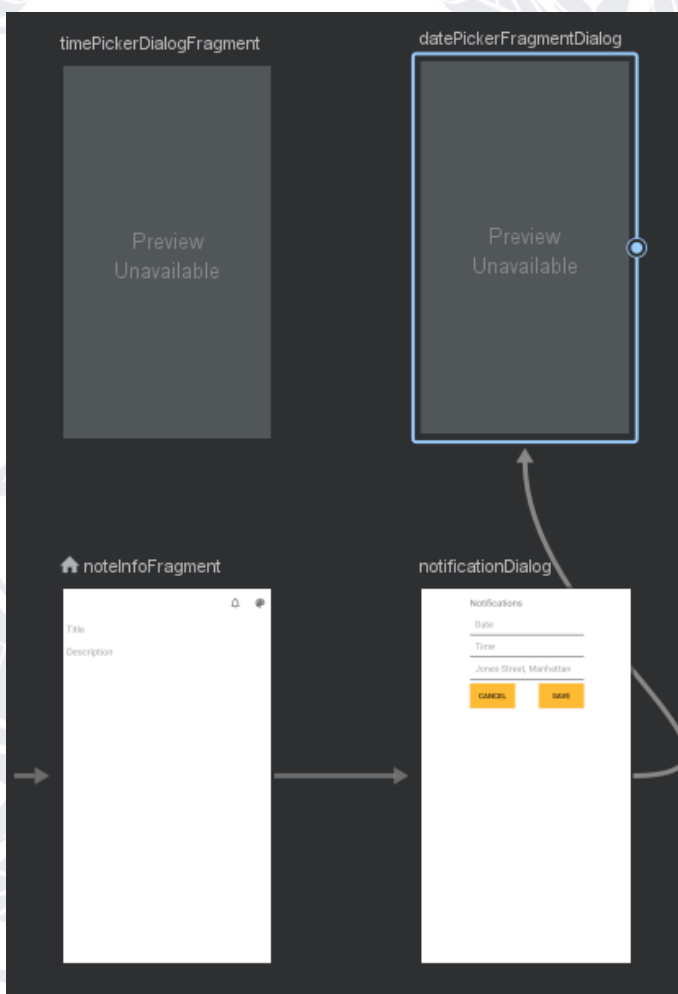


Рисунок 4.2 – Перший граф навігації

Дана програма використовує шаблон в якому існує одна активність (окремий екран в Android, дозволяє розміщувати всі компоненти користувацького інтерфейсу або віджити на екрані) та з безлічі фрагментів.

Фрагмент представляє собою багаторазове використання користувацького інтерфейсу. Одним з ключовим переваг даного підходу є те, що операції у фрагментах виконуються швидше чим створення нових видів активності. Перший граф навігації був створений для екрану з інформацією про задачі. Першим екраном з якого починається навігація є `noteInfoFragment`, він зв'язаний з `notificationDialog` (діалог для встановлення повідомлень) і може відкрити два інших діалоги, це `datePickerFragmentManager` (діалог обрання дати) та `timePickerDialogFragment` (діалог для обрання часу). Також є ще один граф навігації, який може привести до всіх інших графів.

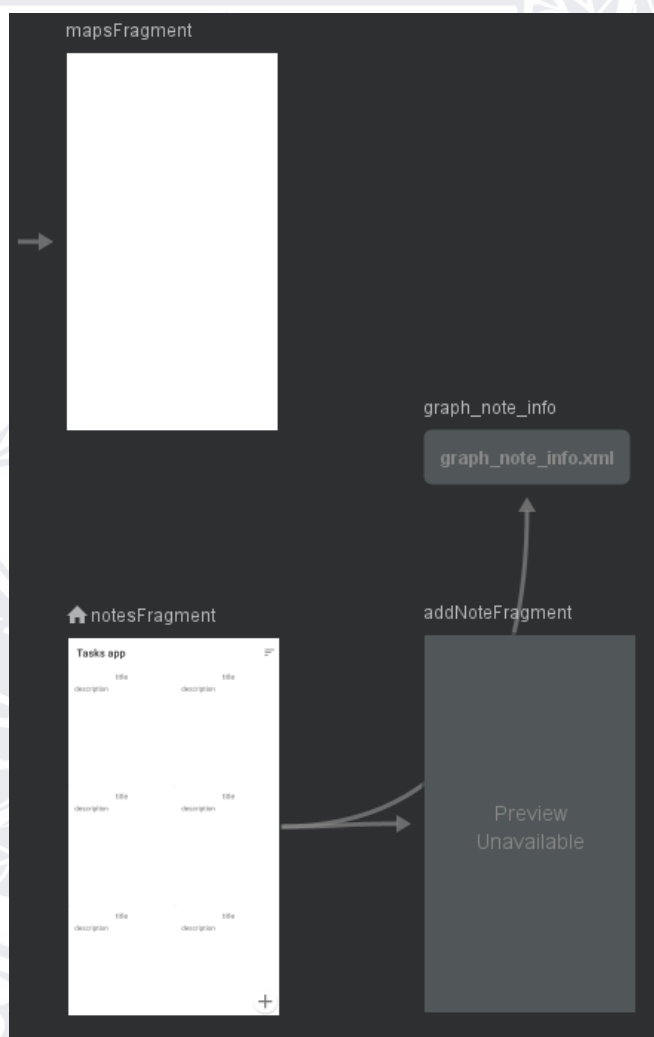


Рисунок 4.3 – Другий граф навігації

Після запуску програми створюється цей граф і відкривається перший фрагмент – `notesFragment` (фрагмент із списком всіх задач). Далі натиснувши на

кнопку “Add” запуститься `addNoteFragment`, а натиснувши на якусь із задач відкриється внутрішній граф, який зображений на рисунку та має назву `graph_note_info`. `Maps_Fragment` не зв’язаний ніякою дією з іншими фрагментами, це зроблено, щоб до нього була можливість дістатись з будь-якого місця програми, для цього `maps_fragment` був об’явлений як `global action`, це показує стрілочка зліва від нього.

Dependency injection

Класам часто потрібно силки на інші класи [3]. Ці необхідні класи називаються залежностями. Клас створює необхідну йому залежність. Програма може дати ці залежності при конструюванні класу, або передавати їх функціями, яким потрібна ця залежність. В даній програмі використовується `Hilt`.

Data Binding

Бібліотека `Data Binding Library`, яка являється частиною `Android Jetpack`, дозволяє прив’язувати компоненти інтерфейсу в макетах до джерел даних в програмі, використовуючи декларативний формат, а не програмно [4]. Іншими словами, `Data Binding` допомагає організувати роботу з `View` так, щоб не приходилось писати багато інший методів.

Coroutines

Корутини – це патерн проектування паралелізму, який можна використовувати в `Android` для спрощення кода, який виконується асинхронно [1]. В `Android` корутини допомагають керувати довготривалими задачами, які в іншому випадку можуть заблокувати головний потік (потік, на якому відображається інтерфейс програми) і призвести до несприйнятливості програми.

Використовуючи корутини можна замітити збільшення продуктивності, а також писати більш чистий і лаконічний код

LiveData

LiveData – це клас спостерігач даних. На відміну від простих спостерігачів, LiveData враховує життєвий цикл інших компонентів програми, таких як фрагменти чи активності [5]. Завдяки цьому LiveData обновлює тільки ті спостерігачі, які знаходяться в активному стані життєвого циклу.

ColorPicker

Бібліотека для зручного обирання кольору.

Glide

Бібліотека призначена для асинхронного завантаження зображень, ресурсів чи файлової системи з мережі, їх хешування та відображення [6].

Базові класи

1. BaseFragment. В даній програмі існує багато фрагментів, і у всіх них є схожий функціонал, щоб не повторювати однаковий код декілька раз, його було винесено в клас BaseFragment. Всі інші фрагменти будуть його наслідувати. Щоб при створенні фрагмента не викликати onCreateView (метод життєвого циклу фрагмента) і не приєднувати до нього layout, за допомогою view binding, layout буде передаватись в конструкторі. метод onBackPressed() викликається, коли натиснута кнопка назад, він буде повертати користувача на минулий відкритий екран, якщо це перший екран програми то програма закриється.
1. BaseFragmentManager. Має такий самий функціонал як і BaseFragment, але унаслідується не від класу Fragment, а від DialogFragment. Даний фрагмент може бути розширеним лише фрагментами, які мають відображати діалогові вікна.
2. MainActivity. Ця активність є єдиною у всій програмі. Саме з неї розпочинається запуск програми. Її layout файл містить NavigaitonHostFragment – фрагмент бібліотеки Fragment Navigation, він

створює граф навігації. В активності ініціалізуються клас NavController за допомогою якого буде можливість фрагментам відкривати інші фрагменти.

4.3 Розробка логіки для екрану із списком завдань

Спочатку необхідно створити екран із задачами, а потім поєднати дані бази даних із списком задач. NotesFragment – фрагмент із задачами. Перший клас, який необхідно створити це NotesRepository. Репозиторій буде відповідати за визначення джерела даних. На даному екрані потрібне одне джерело даних - NoteDao, репозиторій буде отримувати дані з бази даних, обробляти і передавати їх у view model, а також отримавши дані, буде їх записувати або поновлювати в базі даних. Створивши клас, йому необхідно передавати NoteDao в конструкторі.

За допомогою dependency injection, в класі AppModule, описується як отримати це джерело даних. Анотація Provide зберігає в графі залежностей, відповідний клас.

Після об'явлення класу, необхідно створити методи доступу до бази даних. Виклик бази даних неможливий на головному потоці тому, що інтерфейс може промальовуватись із затримками. Щоб вирішити цю проблему використовуються корутини. Ключове слово suspend необхідно написати перед об'явленням функції, а функції очікування будуть виконуватись в спеціальному скоупі.

Метод insert() отримує в параметрах створену задачу, а noteDao записує її в базу даних. Використовується ключове слово – suspend, щоб функція виконувалась асинхронно. Метод приймає в аргументах клас Note, який містить дані задачі. Так, як suspend функції мають виконуватись у scope, потрібно використати CoroutineScope і скориставшись методом launch, даного scope, запустити нову корутину, та викликати метод insertNote класу NoteDao, щоб записати передану задачу в базу даних.

Після того, як було описано модель і всі її частини, необхідно її реалізувати. Для цього потрібно створимо клас `NotesViewModel`, який буде унаслідуватися від класу `ViewModel` `Android Jetpack`. Клас `ViewModel` створений, щоб зберігати і керувати даними, зв'язаними з UI відносно життєвого циклу. Він дозволяє пережити змінення конфігурації, наприклад, повернення екрана. Якщо б не використовувався клас `ViewModel`, то після повернення екрана, дані не були б збережені.

Метод `setNotes()` призначений для збереження нових даних в полі `listOfNotes`, які будуть отримані з бази. Використовується ключове слово `suspend`, щоб всі дії не виконувались в одному потоці, це прискорить роботу програми. Будь-яка корутина виконується у `scope`, дана функція має виконуватись на головному потоці програми, саме тому передається у `CoroutinesScope` потрібний диспатчер - `Dispatchers.Main`. Дані отримуються за допомогою репозиторія – `NotesRepository`, викликавши метод – `listOfNotes()`, з допомогою функції `reverse()`, список завдань буде поставлений у зворотному напрямку.

Змінна `listOfNotes` є спостерігачем, який прив'язаний до життєвого циклу `view model`, коли знищиться `view model` тоді і `listofNotes`.

Коли `view model` створена, вже є можливість обмінюватись даними з базою даних.

Далі є можливість створити фрагмент. Він буде унаслідуваний від `BaseFragment` і помічений анотацією `@AndroidEntryPoint`, яка належить фреймворку впровадження залежностей – `Hilt`, і означає, що в цей фрагмент можуть бути впроваджені залежності. Необхідно перевизначити змінну – `bindingInflater`, щоб бібліотека - `view binding` розуміла, як правильно створити змінні, які будуть прив'язані до фрагмента – `NoteInfoFragment`.

Щоб об'явити поле `viewModel` в `NoteInfoFragment`, був використаний делегат `activityViewModels`, який знаходить всі залежності з графа залежностей, він є

дженериком і тому потрібно передати `NoteInfoViewModel`. Змінна приватна тому, що вона має використовуватись тільки в даному фрагменті.

Наступним кроком необхідно створити список (`RecyclerView`), який буде утримувати в собі задачі. Для зв'язування даних з компонентом необхідно створити `adapter`. Клас адаптера має назву `NewAdapter`, в конструкторі приймає дві функції, перша – що має виконуватись при довгому натисканні на задачу і друга при короткому та унаслідкується від `ListAdapter`. `ListAdapter` є дженерик типов, він приймає два аргументи, першим з яких є клас даних, у програмі це клас `Note`, та `ViewHolder` клас, який надає дані для layout-представлення завдання, це клас `NotesViewHolder`.

В адаптері створюється `view holder` (описує представлення елемента і метадані про його місце в `RecyclerView`), в конструкторі, за допомогою `view binding`, передається конкретна `view` і прикріплюється до `view holder`.

У методі `onBindViewHolder` встановлюються дані, які будуть відображатись, і додаються слухачі натискання на задачу. Метод `getItem` класу `ViewHolder` приймає позицію завдання в списку, та повертає саме завдання, воно записується у змінну `currentUser`. З допомогою `ViewBinding`, у поле `note`, зберігається `currentUser`. Метод `executePendingBindings()` викликається, щоб прив'язування не відкладався, а виконувався як можна швидше.

В списках може міститися великий об'єм інформації, тому важливо забезпечити зручність прокрутки при оновленні його контенту. Для цих цілей використовується службовий клас `DiffUtil`, який надає цю можливість. Також `DiffUtil` використовується для порівняння задач і якщо при створенні нової, вона є ідентична тій, яка вже існує в списку, то нова не створиться. Метод `areItemTheSame` приймає старе та нове завдання, порівнює їх та повертає `Boolean` значення чи завдання однакові. Метод `areContentsTheSame()` приймає в аргументах стару та нову завдання та порівнює змінні класу `Note`, якщо вони

однакові повертає Boolean значення true, відповідно false, якщо різні. Ці всі методи складають логіку адаптера.

Щоб зв'язати RecyclerView і Adapter використовується ViewBinding.

Користуючись перевагами view binding, методи присвоєних класів викликаються не в коді, а в layout файлі, що робить фрагмент більш чистим. Для цього потрібно створити об'єкт NoteRecyclerViewBinding і в ньому створити методи bindRecyclerViewAdapter для layout файла. Метод приймає в аргументах адаптер та присвоює за допомогою ViewBinding. Враховуючи, що даний код може використовувати Java, метод був позначений анотацією JvmStatic, яка дає постанову компілятору, створити додатковий статистичний метод bindRecyclerViewAdapter, щоб не викликати клас Companion, коли буде потрібний даний метод.

Перший метод, щоб приєднати адаптер має назву – setCustomAdapter, за допомогою анотації @BindAdapter він створиться у layout файлі. Та другий метод – setItemDecoration, створить метод – setSpaceItemDecoration. Він встановлює декоратора для списку. Метод також позначений анотацією JvmStatic.

у layout потрібно створити змінні присвоєних класів, щоб викликати їх методи.

В файлі fragment_list_note, За допомогою змінних, у списку (RecyclerView) викликаємо необхідні методи.

В NoteInfoFraagment необхідно створити слухача поля listOfNotes класа NotesViewModel, щоб після будь-яких оновлень в базі даних отримувались нові дані і встановлювались в адаптер. Метод submitList() класу NewAdapter, приймає новий список завдань та встановлює його в адаптер.

Далі у методі onCreateView(), він відноситься до життєвого циклу фрагмента, потрібно почати слухати ці дані викликавши метод setNotices() класа NotesViewModel.

Останнім кроком фрагмент має перевизначити метод довгого натискання на задачу. При довгому натисканні має відкриватись інший toolbar, щоб можна було видалити завдання. Для цього потрібно створити колбек `ActionMode.Callback`, він має методи, які потрібно перевизначити для правильного відображення toolbar. Метод `onCreateActionMode()` прикріплює необхідні іконки, які будуть відображатись на toolbar та повертає `true`, якщо все виконалось правильно. Метод `onActionItemClicked()` обробляє натискання на іконки. Даний toolbar містить всього одну іконку для видалення задачі. Після її натискання викликається метод класу `NotesViewModel`, який має назву `deleteNote()`, він буде видаляти обрану задачу. Далі викликається метод `setNotices()` класу `NotesViewModel`, щоб вставити новий список, без видаленої задачі. Останнім кроком, викликається метод `finish()`, щоб закрити даний toolbar() та повернутись до минулого.

Наразі елементи списку знаходяться дуже близько один до одного, що не є зручним для натискання на завдання, а також візуально не виглядає привабливо. Для виправлення цих недоліків був створений спеціальний декоратор `RecyclerViewItemDecoration`, який був присвоєний у `layout` файлі, він приймає `Int` значення – відступ між задачами у списку. Декоратор збільшує відступи між елементами в методі `getItemOffsets()`.

Останнім кроком необхідно додати можливість додавати нові завдання. Для цього після натискання кнопки «дати» буде відкриватися новий фрагмент, який має назву `AddNoteFragment`. В методі `init` створюється колбек натискання кнопки назад `requireActivity().onBackPressedDispatcher.addCallback`. В аргументах передається колбек `onBackPressed`, в ньому перевіряється якщо поля `title` і `description` пусті, то нове завдання не зберігається, в іншому випадку з допомогою метода `insert()` класу `AddNoteViewModel` передається нове завдання з даними, які введені у полях. Останнім кроком, щоб кнопка назад працювала, викликається метод `popBackStack()`, який належить бібліотеці `FragmentNavigation`, він повертає користувача на минулий екран.

4.4 Розробка логіки для екрану з даними про завдання

У цьому підрозділі необхідно розробити логіку для наступних пунктів:

- Зміни в полях `description` і `title` зберігалися в базі даних
- Можливість взаємодії з діалоговими вікнами встановлення нотифікацій та кольору, збереження налаштувань в базі даних
- Взаємодія з картою, встановлення маркерів на ній

Як і у минулому розділі розробка розпочинається з репозиторія. Буде доцільно використовувати вже існуючий репозиторій – `NotesRepository` тому, що деякі його методи також будуть необхідні на цьому екрані. Потрібно створити додаткові методи для взаємодії з базою даних:

1. Для отримання кольору потрібно створити метод `getColor()`, він повертає цілочисельне значення кольору. Колір буде отримуватись з `NoteDao`, з допомогою метода `getColor()`.
2. Для оновлення кольору в базі даних створюється метод `updateColor()`. В аргументах першим значенням передається цілочисельне значення кольору, а другим ідентифікаційний номер завдання. З допомогою метода `updateColor()` класу `NoteDao` завдання зберігаються в базі даних.
3. Для оновлення опису завдання створюється метод `updateDescription()`. В аргументах передається стрічка з описом та ідентифікатор завдання. Використовуючи метод `updateDescription()` класу `NoteDao` новий опис буде записуватись в базу даних.
4. Метод `updateTitle()` оновлює заголовок завдання. Він приймає в аргументах стрічку з назвою заголовка та ідентифікатор завдання. За допомогою метода `updateTitle()` класу `NoteDao` в базу даних буде записано нове значення.
5. Для оновлення дати був створений метод `updateDate`. В аргументах передається рік, місяць, день та ідентифікатор завдання. Метод `updateDate()` класу `NoteDao` буде оновлювати значення дати в базі даних.

Наступним кроком створюється `NotesViewModel` та передається репозиторій в конструкторі. Після створення `view model` вона передається у `NoteInfoFragment`.

Щоб не використовувати важкі операції отримання даних з бази даних, строки `description` і `title` будуть передаватися з фрагмента `NotesFragment`, за допомогою `safe argument`, бібліотеки `Fragment Navigation`. Це є рекомендованим способом передавання аргументів між фрагментами тому, що аргументи є безпечними (перевіряється тип аргументів). В методі `init()` отриманні аргументи будуть присвоюватись полям за допомогою `view binding`.

На даний момент, після відкриття завдання значення полів будуть відповідати тим, які були в обраного завдання із списку завдань. Щоб після змінення полів нові значення записувалися в базу даних потрібно викликати `view model` в методі `onResume()` і передавати нові значення в неї. Відповідний метод `view model` буде викликатися після змінення будь-якого з полів.

Далі необхідно додати можливість повертатись на минулий екран (екран із списком завдань). Це досягається за допомогою додавання відповідного колбека в активність.

Останнім кроком потрібно налаштувати всі діалогові вікна, які відповідають за встановлення нотифікацій. Першим необхідно налаштувати діалогове вікно, яке відкривається після натискання на іконку дзвіночка. Для цього потрібно створити фрагмент `NotificationDialog`, який унаслідуеться від `BaseFragmentDialog`.

Це діалогове вікно має відкриватися у `NoteInfoFragment`. Так як воно вже створене, потрібно у фрагменті `NoteInfoFragment` налаштувати логіку відкриття цього діалогового вікна. Це досягається перевизначивши метод базового фрагмента (від якого унаслідуються всі інші фрагменти) `OnOptionsItemSelected()`. І в ньому створюється навігація до `NotificationDialogFragment`. Викликавши метод `navigate()` бібліотеки

FragmentManager передається дія

action_noteInfoFragment_to_notificationDialog та другим аргументом дані.

Далі створюються функції, які будуть містити логіку відображення діалогових фрагментів вибору часу та місця. Функція відображення дати –

OnDatePickListener(). Після обрання дати функція повертає такі дані: рік, місяць та день. Повернуті дані будуть зберігатись у базі даних, за допомогою функції updateDate().

Наступна функція створена для задання часу та має назву – SetTimePicker(). В ній створюється діалогове вікно, а в конструкторі задаються початкові значення, які має відобразити діалогове вікно. Далі встановлюються слухачі натискання на кнопки. Натиснувши на кнопку «ОК» - час буде зберігатись в базі даних. Натиснувши на кнопку «Cancel» - діалогове вікно закриється.

В перевизначеному методі фрагмента – onCreateView() за допомогою view binding викликаються ці функції.

Останнім кроком потрібно встановити фрагмент для взаємодії з картою.

MapsFragment перевизнача багато слухачів та колбеків, за допомогою яких буде взаємодія з картою.

Відображення карти встановлюється в методі onCreateView. Карта відображається асинхронно, щоб не блокувати екран.

При відкритті фрагмента користувач спочатку має побачити запит на отримання його локації. Для цього викликається метод onRequestPermissionsResult.

Якщо користувач дав дозвіл на отримання локації буде викликана функція enableMyLocation, як збереже в базі даних значення true, інакше false.

Метод onMapReady запуститься коли карта буде готова для відображення. В цьому методі буде встановлені слухачі натискання на кнопку приблизити та взаємодію з картою, запуститься запит на отримання локації користувача та встановляться маркери.

Щоб видалити маркер на нього потрібно натиснути ще раз, ця логіка встановлена в методі OnMarkerClick.

Останнім кроком потрібно перевизначити метод натискання на кнопку приблизити onMyLocationButtonClick().

4.5 Приклади використання програми

Після відкриття програми користувач побачить пустий список із завданнями. Для додавання нових необхідно натиснути на кнопку додати, яка знаходиться у правому нижньому куті. Після натискання користувач буде бачити два поля, вони призначені для введення заголовку та опису завдання відповідно. Щоб встановити інші нотифікації потрібно натиснути на кнопку дзвіночка, яка розміщена у правому верхньому куті. Натиснувши, користувач буде бачити три поля, після натискання на них будуть відкриватися відповідні діалогові вікна для встановлення дати, часу. Після натискання на поле з картою, користувачу буде відображене діалогове вікно, яке запросить дозвіл на отримання теперішнього місця розташування користувача, якщо користувач відхилить дозвіл, то екран з картою закриється тому, що для відправлення нотифікацій потрібно знати де знаходиться користувач. Якщо дозвіл буде наданий відкриється карта, на якій юзер буде мати змогу встановити маркери відповідно до тих місць, де йому потрібно виконати завдання. Натиснувши на кнопку «Приблизити» карта встановиться на те місце, де знаходиться користувач. Нотифікації будуть відображатись користувачу при досягненні часу, який був встановлений чи місця на карті. Якщо користувачу потрібно ділити завдання на групі, то на екрані з додавання завдання існує можливість встановити колір, який буде помічати відповідну групу, також ця можливість буде і на екрані з даними про завдання. Групи – це лише один із можливих способів використання кольору, кожний користувач вирішує сам для чого йому потрібна ця можливість. Додавши завдання воно буде відображатись у списку на головному екрані, нові завдання будуть додавати з початку (лівий верхній кут). Завдання, які були виконані необхідно видаляти, для цього потрібно

затиснути одне із них на екрані із списком завдань, після чого відкриється додаткове меню у верхній частині екрану, натиснувши на іконку з кошиком, завдання буде видалено.

Висновки до розділу 4

Була написана вся логіка програми. Використовувалось багато популярних бібліотек, які спрощують розробку. Використання шаблону проектування – One Activity дійсно пришвидшує програму та робить навігацію простішою. Шаблон проектування MVVM зробив компоненти програми менш зв'язними, а саме написання коду більш швидким.

ВИСНОВКИ

У бакалаврській роботі був досліджений процес розробки програми для планування завдань на системі Android. Код був написаний зважаючи на правильні стилі його написання та правила. Бібліотеки, які були використані дійсно спрощують поєднання різних шарів програми. Був створений зручний інтерфейс з яким користувач може легко взаємодіяти. Проблема, що людина може забути виконати завдання, була вирішена за допомогою додання нотифікацій, які будуть приходити користувачу, коли він досягне вказаного місця на карті або прийде заданий в програмі час. Можна зробити висновок, що дана програма покриває більшість можливих випадків, при яких користувач може забути виконати поставлене завдання.

СПИСОК ЛІТЕРАТУРИ

1. Kotlin корутини. Документація.
URL: <https://developer.android.com/kotlin/coroutines>
2. Навігація фрагментів. Автор: Дмитро Виноградов. Стаття. URL:
<https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/557-urok-24-android-navigation-component-vvedenie.html>
3. Інверсія залежностей. Документація.
URL: <https://developer.android.com/training/dependency-injection/hilt-android>
4. Data binding. Стаття. Автор: Дмитро Виноградов.
URL: <https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/551-urok-18-data-binding-osnovy.html>
5. LiveData. Стаття.
URL: <https://www.geeksforgeeks.org/livedata-in-android-architecture-components/>
6. Glide. Стаття. Автор: Олександр Клімов.
URL: <http://developer.alexanderklimov.ru/android/library/glide.php>
7. Room. Стаття. Автор: Олександр Клімов.
URL: <http://developer.alexanderklimov.ru/android/room.php>
8. Constraint layout. Стаття. Автор: Олександр Клімов.
URL: <http://developer.alexanderklimov.ru/android/layout/constraintlayout.php>
9. Стилізація android-програми. Пост.
URL: <https://habr.com/ru/company/citymobil/blog/507896/>
10. Зберігання даних android. Пост.
URL: <https://habr.com/ru/post/336196/>
11. Структура Room. Документація.
URL: <https://developer.android.com/training/data-storage/room>

Код файлу «NotesFragment.kt»

```

1. @AndroidEntryPoint
2. class NotesFragment(override val bindingInflater: (LayoutInflater,
   ViewGroup?, Boolean) -> FragmentListNoteBinding =
   FragmentListNoteBinding::inflate) :
3.     BaseFragment<FragmentListNoteBinding>(), OnLongClickListNotesItem {
4.
5.     val noticeViewModel by
6.         hiltNavGraphViewModels<NotesViewModel>(R.id.nav_graph)
7.
8.     @Inject
9.     lateinit var myHandler: MyHandler
10.
11.     override fun onCreateView(view: View, savedInstanceState: Bundle?) {
12.         super.onCreateView(view, savedInstanceState)
13.         noticeViewModel.setNotices()
14.     }
15.
16.     override fun init() {
17.         val notesAdapter = NewAdapter(this) {
18.             val bundle: Bundle = Bundle().apply {
19.                 putString("description", it.description)
20.                 putString("title", it.title)
21.                 putInt("id", it.noteId)
22.             }
23.             //findNavController().setGraph(R.navigation.graph_note_info,
24.             bundle)
25.             findNavController().navigate(R.id.graph_note_info, bundle)
26.         }
27.
28.         binding.apply {
29.             viewModel = noticeViewModel
30.             adapter = notesAdapter
31.             handler = myHandler
32.         }
33.
34.         noticeViewModel.listOfNotices.observe(viewLifecycleOwner) {
35.             it.let(notesAdapter::submitList)
36.         }
37.
38.         val toolbar = binding.toolbarListNoteId as Toolbar
39.         toolbar.inflateMenu(R.menu.menu_toolbar_note_info)
40.         (activity as AppCompatActivity).setSupportActionBar(toolbar)
41.         setHasOptionsMenu(true)
42.     }
43.
44.     override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
45.         inflater.inflate(R.menu.menu_toolbar_list_note, menu)
46.         super.onCreateOptionsMenu(menu, inflater)
47.     }
48.
49.     private fun setBackgroundImage() {

```

```

49.         val imageContainer: ImageView? =
binding.root.findViewById(R.id.pictureContainer)
50.         val url =
    "https://picsum.photos/${imageContainer?.width}/${imageContainer?.height}/?ra
ndom"
51.         Glide.with(this)
52.             .load(url)
53.             .into(imageContainer!!)
54.     }
55.
56.
57.     override fun onLongClick(note: Note) {
58.         val callbackList = object : ActionMode.Callback {
59.             override fun onCreateActionMode(mode: ActionMode?, menu: Menu?):
Boolean {
60.                 mode?.menuInflater?.inflate(R.menu.menu_contextual_toolbar_list_note, menu)
61.                 return true
62.             }
63.
64.             override fun onPrepareActionMode(mode: ActionMode?, menu: Menu?):
Boolean {
65.                 return true
66.             }
67.
68.             override fun onActionItemClicked(mode: ActionMode?, item:
MenuItem?): Boolean {
69.                 return when (item?.itemId) {
70.                     R.id.list_contextual_delete -> {
71.                         CoroutineScope(Dispatchers.Default).launch {
72.                             noticeViewModel.deleteNote(note)
73.                         }
74.                         noticeViewModel.setNotices()
75.                         mode?.finish()
76.                         true
77.                     }
78.                     else -> false
79.                 }
80.             }
81.
82.             override fun onDestroyActionMode(mode: ActionMode?) {
83.
84.             }
85.         }
86.         (activity as AppCompatActivity).startSupportActionMode(callbackList)
87.     }
88. }

```



```

50.             override fun onCancel() {
51.                 TODO("Not yet implemented")
52.             }
53.         })
54.         true
55.     }
56.     else -> super.onOptionsItemSelected(item)
57. }
58. }
59.
60. override fun init() {
61.     noteId = arguments?.getInt("id")!!
62.     binding.noteInfo.title.text =
63.     SpannableStringBuilder(arguments?.getString("description"))
64.     binding.noteInfo.description.text =
65.     SpannableStringBuilder(arguments?.getString("title"))
66.
67.     val configuration = AppBarConfiguration(findNavController().graph)
68.
69.     val window = requireActivity().window
70.
71.     viewModel.color.observe(this) { color ->
72.         window.statusBarColor = color
73.     }
74.
75.     requireActivity().onBackPressedDispatcher.addCallback(onBackPressed {
76.         view?.post { findNavController().popBackStack() }
77.     })
78.
79.     (activity as
80.     AppCompatActivity).setSupportActionBar(binding.toolbarNoteInfo as Toolbar)
81.     (activity as
82.     AppCompatActivity).setHasOptionsMenu(true)
83.     (activity as
84.     AppCompatActivity).supportActionBar?.setDisplayHomeAsUpEnabled(true);
85.     (activity as
86.     AppCompatActivity).supportActionBar?.setDisplayShowHomeEnabled(true);
87.
88.     requireActivity().onBackPressedDispatcher.addCallback(object :
89.     OnBackPressedCallback(true) {
90.         override fun handleOnBackPressed() {
91.             view?.post { findNavController().popBackStack() }
92.         }
93.     })
94. }
95. }

```

Декларація щодо унікальності текстів роботи
та невикористання матеріалів інших авторів без посилань

Лукіяничук Владислав Володимирович

Прізвище, ім'я, по батькові

Факультет інформаційних і прикладних технологій

Факультет

122 Комп'ютерні науки

Шифр і назва спеціальності

Сучасні інформаційні технології та програмування

Освітня програма

ДЕКЛАРАЦІЯ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (бакалаврська) робота на тему:

«Розробка планувальника завдань з геолокаційним порадиником для системи android» є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувались іншими особами, а також дані та інформація не отримувались у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна (бакалаврська) робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

16.05.2022

дата



підпис