Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студентки *Бенемянович Еліни Іллівни*
(ПІБ)

академічної групи *122-19-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка персонального мобільного Android-додатку планувальник справ мовою програмування Java та СУБД SQLite*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Гуліна І.Г.*			
розділів:				
спеціальний	*доц. Гуліна І.Г.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент	*доц. Шедловський І.А.*			
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студентки *122-19-1* *Бенемянович Еліни Іллівни*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка персонального мобільного Android-додатку планувальник дій мовою програмування Java та СУБД SQLite*

затверджена наказом ректора НТУ «ДП» від «16» травня 2023 р. № 350-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проектно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***12.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***26.05.2023 р.***

Завдання видав _____ *доц. Гуліна І.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняла до виконання _____ *Бенемянович Е.І.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *10.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 62 с., 12 рис., 3 дод., 20 джерел.

Об'єкт розробки: персональний мобільний Android-додаток планувальник дій.

Мета кваліфікаційної роботи: підвищення зручності при плануванні справ, за рахунок розробки та впровадженню відповідного мобільного персонального електронного планувальника.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування застосунку, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленого програмного забезпечення, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Практичне значення полягає у створенні застосунку, що надає користувачу простий у використанні і зручний інструмент для організації розпорядку дня та для діловодства.

Актуальність програмного продукту визначається великим попитом на подібні інструменти.

Список ключових слів: МОБІЛЬНИЙ ДОДАТОК, АНДРОІД, ОРГАНАЙЗЕР, ПЛАНУВАННЯ ДІЙ, JAVA.

ABSTRACT

Explanatory note: 62 p., 12 figs., 3 app., 20 sources.

Object of development: action planner of a personal mobile Android application.

The purpose of the qualification work: to increase the convenience of planning cases, due to the development and implementation of a suitable mobile personal electronic planner.

In the introduction, the analysis and current state of the problem is considered, the purpose of the qualification work and the field of its application are specified, the justification of the relevance of the topic is given, and the statement of the task is clarified.

In the first section, an analysis of the subject area was carried out, the relevance of the task and the purpose of the development were determined, the task formulation was developed, and the requirements for software implementation, technologies and software tools were set.

In the second section, an analysis of existing solutions was performed, a platform for development was selected, the design and development of the program was performed, a description of the algorithm and structure of the application was given, input and output data were determined, the characteristics of the technical equipment parameters were given, the application was called and downloaded, the operation of the program was described.

In the economic section, the labor intensity of the developed software product is determined, the cost of work on creating the application is calculated, and the time for its creation is calculated.

The practical meaning is to create an application that provides the user with an easy-to-use and convenient tool for organizing the daily routine and for office management.

The relevance of the software product is determined by the high demand for similar tools.

Keywords: MOBILE APP, ANDROID, ORGANIZER, SCHEDULER, JAVA.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

БД - база даних;

ЕОМ - електронно-обчислювальна машина;

ІС - інформаційна система;

ПЗ - програмне забезпечення;

СУБД - система управління базами даних;

ІТ - інформаційні технології.

ВСТУП

Людина протягом усього свого життя знає та пам'ятає свій день народження, великі свята, коли день народження родичів. Але всі люди не настільки унікальні, щоб зберігати в пам'яті навіть найдрібніші деталі, наприклад, купити в магазині або рецепт пирога. Людина може приділяти всьому цьому уваги. Тому на допомогу приходять сучасні технології – мобільні пристрої. Користувачі мають можливість зберігати свої нотатки.

Програми для запису інформації є модель сховища, де дані зберігаються в базі даних. Користувач отримує зручний спосіб зберігання нотаток та може отримувати доступ до них.

Існує велика кількість різних сервісів, які надають послуги зберігання інформації. Чимось вони схожі і в чомусь є відмінності.

За допомогою таких додатків можна зручно впорядковувати та структурувати інформацію, планувати завдання, створювати нагадування та поєднувати всі записи в одному додатку.

Мета кваліфікаційної роботи: набуття практичних навичок по розробці мобільного додатоку «Планувальник справ» для операційної системи Android. Для досягнення поставленої мети вирішуються такі завдання:

1) проаналізувати предметну галузь;

2) спроектувати архітектуру програми;

3) створити базу даних для зберігання в ній нотаток;

4) зверстати інтерфейс програми;

5) провести тести програми на наявність помилок.

В якості редактору коду було обрано багатоплатформений редактор коду Android Studio. Ця програма дуже зручна у використанні, має зручний інтерфейс, швидко працює та дає змогу одночасно редагувати багато файлів. Також виявляє та виділяє синтаксичні помилки, підсвічує синтаксис та фолдинг.

Для зберігання даних було використано базу даних. Для управління базою даних була обрана СКБД SQLite.

Код програми було написано мовою програмування Java.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

У роботі розглядається розробка програми для мобільних пристроїв під управлінням ОС Андроїд «Планувальник справ».

Користувачі обговорють, який носій для запису інформації ефективніший, паперовий чи електронний. І ось чому люди віддають перевагу електронному варіанту [2].

1. Завжди під рукою.

Головний плюс електронних – вони завжди під рукою, блокнот можна десь забути чи не вмістити у сумку. А із квартири без телефону ніхто не виходить.

2. Велика кількість пам'яті.

Паперові носії, на жаль, обмежені кількістю аркушів, у цифрових із цим ситуація краща. В електронних можна друкувати текст, прати, заново записувати, перестворювати.

3. Легко змінюються налаштування дизайну (але не скрізь).

Електронний носій легко та швидко налаштувати для потреб. Змінити колір, тему чи шрифт. Така корисна річ не набридне, даючи можливість та простір для творчості.

Проте прихильники паперового варіанта також виділяють позитивні сторони.

1. Особлива енергетика.

Для когось це головний аргумент користуватись паперовими версіями. Це приблизно порівняно з паперовими книгами, які мають свою унікальну магію, блокноти так само передають неймовірні тактильні відчуття. Приємні на дотик обкладинки, папір – це все мотивує частіше відкривати щоденник.

2. Розвиток креативу.

Вже доведено, що лист, написаний від руки, дозволяє розвивати креативне мислення, приймати незвичайні рішення, шукати відповіді складні завдання. Коли пишуть на комп'ютері чи телефоні – не завжди є можливість вести лінії, креслити схеми так, як це хочеться інтуїтивно. Слід дотримуватись тих схем та напрямних, які дано в електронній версії блокнота.

Паперовий блокнот дозволяє дати повну волю фантазії. І це часто дозволяє знаходити найнестандартніші рішення головоломок.

3. Естетика канцелярії.

Паперовими блокнотами часто користуються ті, хто дуже любить гарну канцелярію. Можна наклеїти наклейку, зробити закладки чи наклейки, маркери, лінери – все це служить інструментами для планування на папері.

Короткий висновок щодо обох способів зберігання інформації такий: електронний варіант підходить для тих, хто не має вільного часу і хто дотримується раціональних поглядів на життя.

«Блокнот: Швидкі нотатки» [3]

Додаток для створення нотаток із магазину програм Play Market. Можна робити швидкі нотатки на ходу, складати список справ на день та записувати те, що потрібно запам'ятати. Тримати нотатки завжди під рукою за допомогою органайзера нотаток.

Основні можливості:

1) автозбереження;

2) видалення;

3) сортування;

4) можливість поставити кольорові нотатки (6 кольорів).

Слід також виділити і мінуси:

1) після оновлення стираються всі нотатки;

2) інтерфейс сильно завантажений зайвою інформацією.

Скріншоти наведеного прикладу зображено на рис. 1.1.

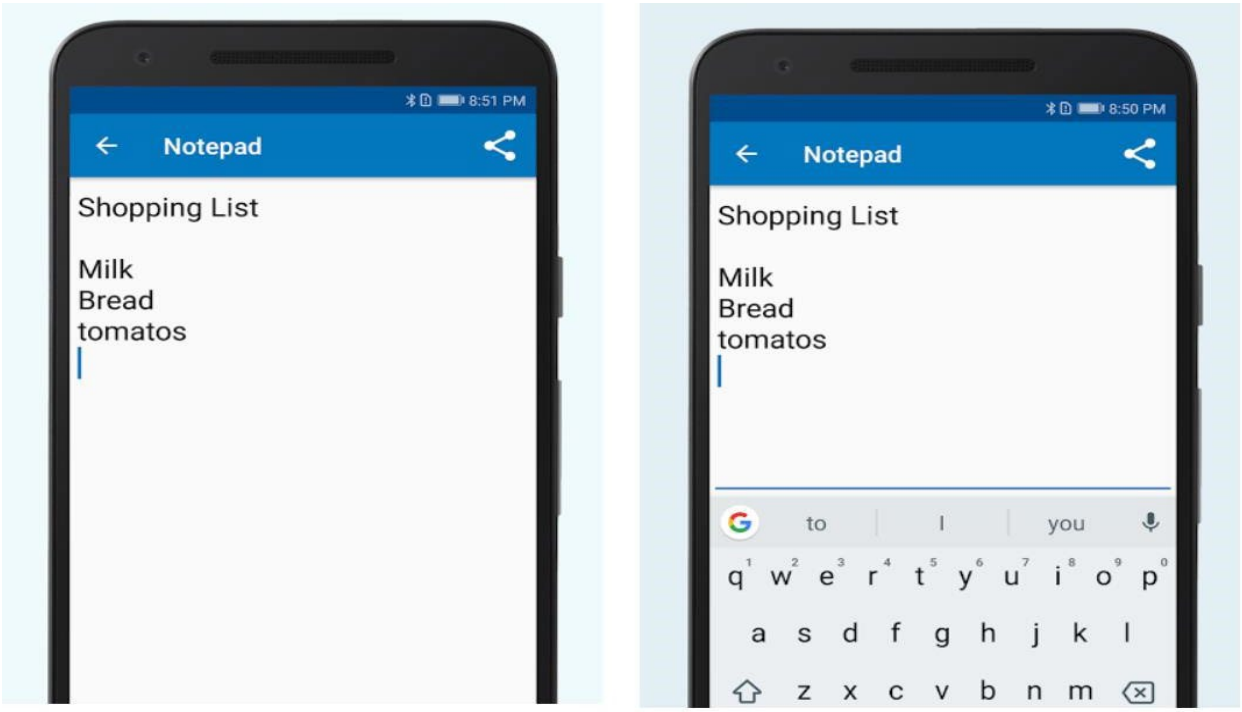

Рис. 1.1. Скріншоти програми «Блокнот: Швидкі нотатки»

"BasicNote - Нотатки, Блокнот" [4]

Другий додаток також має базові функції і не сильно відрізняється від того, яке розбиралося першим.

Можна виділити унікальність та відмінність від першого аналога:

1) можливість поставити темну тему; блокування нотаток за допомогою пароля.

Серед плюсів є також мінуси:

1) немає підтримки старих ОС, починаючи з версії 6.0) обмеження функціоналу за безкоштовного використання.

Скріншоти цього додатка представлені на рис. 1.2.

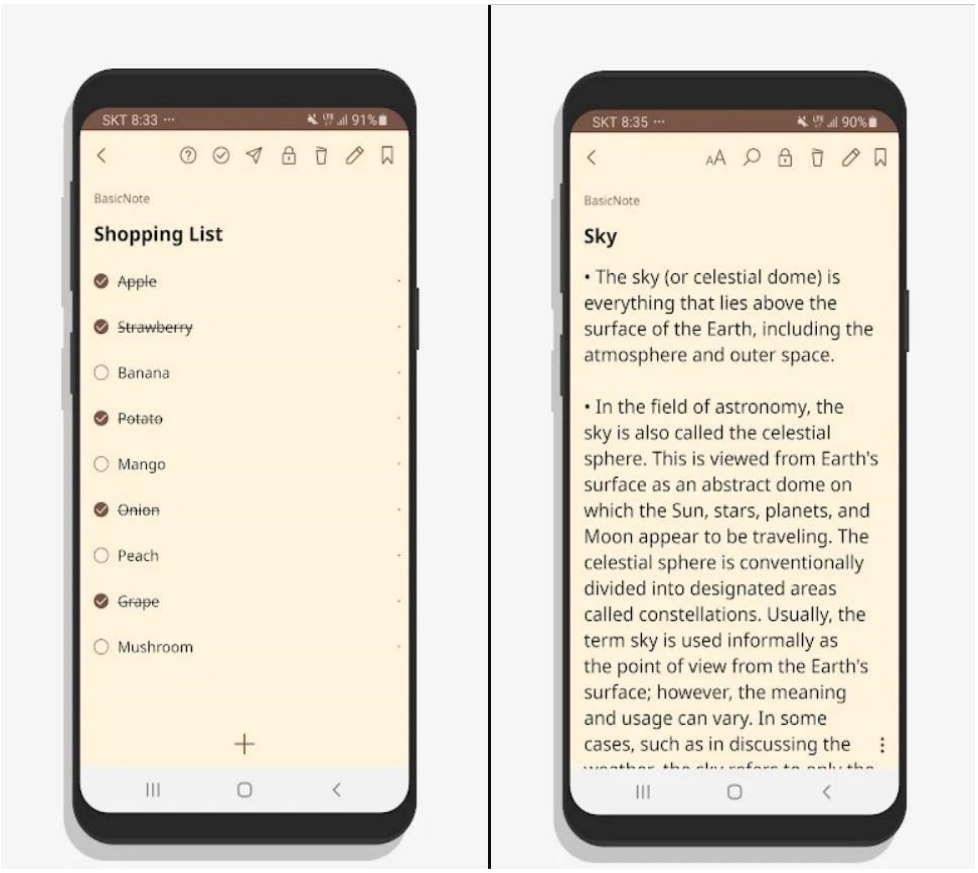

Рис. 1.2. Скріншоти "BasicNote - Нотатки, Блокнот"

Google Keep

Google Keep – додаток для ведення нотаток, розроблений Google.

Плюси:

1) приємний дизайн;

2) синхронізація нотаток.

Мінуси:

1) рідкісні вильоти із додатка;

2) оновлення, які можуть порушити стабільність програми.

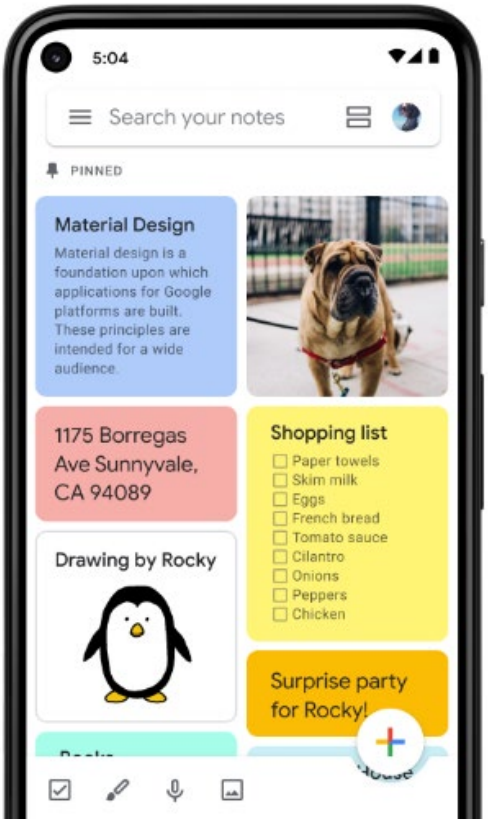

Рис. 1.3. Скріншот Google Keep

1.2. Призначення розробки та галузь застосування

Даний мобільний додаток призначений для більш якісного планування дій користувачем. Мобільний персональний електронний планувальник дій має багато властивостей для ведення якісного планування: веденя справ та подій. Програма нагадує про справи та події у постановлений користувачем час.

Дуже простий інтуітивно зрозумілий та сучасний інтерфейс допомагає легко орієнтуватися у програмі.

1.3. Підстава для розробки

Підставами для розробки та виконання кваліфікаційної роботи є:

- освітня програма спеціальності 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;

- наказ ректора Національного технічного університету «Дніпровська політехніка» № 350 -с від 16.05.2023 р;

- завдання на кваліфікаційну роботу на тему: «Розробка персонального мобільного Android-додатку планувальник дій мовою програмування Java та СУБД SQLite».

1.4. Постановка завдання

Мета кваліфікаційної роботи: набуття практичних навичок по розробці мобільного додатоку «Планувальник дій» для операційної системи Android. Для досягнення поставленої мети вирішуються такі завдання:

1) проаналізувати предметну галузь;

2) спроектувати архітектуру програми;

3) створити базу даних для зберігання в ній нотаток;

4) зверстати інтерфейс програми;

5) провести тести програми на наявність помилок.

Користувацький інтерфейс мобільного додатку має містити в собі такі особливості:

– бути зручним та інтуітивно-зрозумілим;

– мати приємну кольорову гаму;

– бути сумісним з потребами та можливостями користувача;

– забезпечувати простоту переходу від виконання однієї функції до іншої.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Функціональні вимоги, в основному, визначають яка функціональність планується для програмного забезпечення, тобто, вони описують, яка поведінка

повинна показувати системі, що розробляється. Було сформовано функціональні вимоги, які далі доведеться розібрати [9].

1. Користувач може додавати будь-яку нотатку, що містить символи різної мови, цифри і т.д.

2. Користувачу надано доступ до всіх записаних нотаток.

3. Вибір статусу «Готово», редагування та видалення нотаток користувачеві також надається.

4. Після натискання статусу «Готово», вибрана нотатка перекреслюється, показуючи, що вона вже неактивна і виконана.

5. Після виходу і закриття програми, нотатки, що збереглися, не повинні нікуди пропадати і видалятися, а з наступним входом все залишається на своїх місцях.

Нефункціональні вимоги містять у собі властивості та обмеження, що накладаються на систему. Таким чином, були сформовані дисфункції, які представлені нижче.

1. Реалізація програми має бути на ОС Android.

2. Інтегроване середовище розробки має використовуватися Android Studio.

3. Додаток має бути написаний мовою програмування Java.

4. Підтримка версії операційної системи у додатку має бути з 6.0 та новішою.

1.5.2. Вимоги до інформаційної безпеки

Для уникнення некоректної роботи програми необхідно реалізувати:

- контроль вхідних даних;
- обробку виняткових ситуацій;
- виведення повідомлень про помилки;
- можливість повторного введення даних;
- можливість безперервної роботи протягом не менше 120 годин (5 діб);

– забезпечення збереження та неушкодженого стану даних, що зберігаються в базі даних, у випадку відмови застосунку.

Особливих чи додаткових вимог до інформаційної безпеки додатку не висувається.

1.5.3. Вимоги до складу та параметрів технічних засобів

Програма не є вимогливою до складу та параметрів технічних засобів та може завантажуватись на планшетах чи смартфонах різних типів та конфігурацій під управлінням ОС Android 6.0 та пізніші версії.

Для роботи додатку необхідно не менше 100Мб вільного місця в пам’яті пристрою. Для роботи з великими об'ємами нотатків слід передбачити додатковий об'єм пам'яті для БД.

1.5.4. Вимоги до інформаційної та програмної сумісності

Програма не є вимогливою до складу та параметрів технічних та програмних засобів.

Вимоги до програмного забезпечення клієнта:

– наявність пристрою з операційною системою Android 6.0 та пізніші версії;

– ОЗП 100 Мб.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Додаток призначений для більш якісного планування дій користувачем. Мобільний персональний електронний планувальник дій має багато властивостей для ведення якісного планування: веденя справ та подій. Програма нагадує про справи та події у постановлений користувачем час.

Функціональні можливості:

– Користувач може додавати будь-яку нотатку, що містить символи різної мови, цифри і т.д.

– Користувачу надано доступ до всіх записаних нотаток.

– Вибір статусу «Готово», редагування та видалення нотаток користувачеві також надається.

– Після натискання статусу «Готово», вибрана нотатка перекреслюється, показуючи, що вона вже неактивна і виконана.

– Після виходу і закриття програми, нотатки, що збереглися, не повинні нікуди пропадати і видалятися, а з наступним входом все залишається на своїх місцях.

2.2. Опис застосованих математичних методів

Оскільки особливості предметної галузі розв'язуваної задачі не передбачають застосування додаткових та спеціальних математичних методів, при розробці мобільного додатку математичні методи не використовувалися.

2.3. Опис використаних технологій та мов програмування

При розробці додатку було вирішено використовувати в якості інструменту розробки середовище програмування Android Studio. Основні переваги цього середовища:

1) емуляція пристроїв;

2) швидкість складання програми;

3) зручний XML редактор;

4) підтримка рендеру засобами GPU;

5) безкоштовне використання;

6) підтримка всіх платформ та версій Android.

На рис. 2.1. продемонстровано скріншот середовища Android Studio.

Мова програмування Java - це мультифункціональна об'єктно-орієнтована мова з вбудованою строгою типізацією. Її використовують в: Netflix, AliExpress, Google, Intel, eBay, TripAdvisor та багатьох інших [5]. Можна виділити позитивні сторони мови:

1) простота;

2) кросплатформність;

3) безліч доступних бібліотек;

4) затребуваність.

Під «незалежністю від архітектури» мається на увазі те, що програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції.

Цього можна досягти, компілюючи початковий Java код у байт-код, який є спрощеними машинними командами. Потім програму можна виконати на будь-якій платформі, що має встановлену віртуальну машину Java, яка інтерпретує байткод у код, пристосований до специфіки конкретної операційної системи і процесора. Зараз віртуальні машини Java існують для більшості процесорів і операційних систем.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами. У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми.

Основна перевага використання байт-коду — це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної» мови. Проте, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Одним із таких методів є just-in-time компіляція (JIT), що перетворює байт-код Java у машинний під час першого запуску програми, а потім кешує його. У результаті така програма запускається і виконується швидше, ніж простий інтерпретований код, але ціною додаткових витрат на компіляцію під час виконання. Складніші віртуальні машини також використовують динамічну рекомпіляцію, яка полягає в тому, що віртуальна машина аналізує поведінку запущеної програми й вибірково рекомпілює та оптимізує певні її частини. З використанням динамічної рекомпіляції можна досягти більшого рівня оптимізації, ніж за статичної компіляції, оскільки динамічний компілятор може робити оптимізації на базі знань про довкілля періоду виконання та про завантажені класи. До того ж він може виявляти так звані гарячі точки — частини програми, найчастіше внутрішні цикли, які займають найбільше часу при виконанні. JIT-компіляція та динамічна рекомпіляція збільшує швидкість Java-програм, не втрачаючи при цьому портативності.

Існує ще одна технологія оптимізації байткоду, широко відома як статична компіляція, або компіляція ahead-of-time (AOT). Цей метод передбачає, як і традиційні компілятори, безпосередню компіляцію у машинний код. Це забезпечує хороші показники в порівнянні з інтерпретацією, але за рахунок

втрати переносності: скомпільовану таким способом програму можна запустити тільки на одній, цільовій платформі.

Швидкість офіційної віртуальної машини Java значно покращилася з моменту випуску ранніх версій, до того ж, деякі випробування показали, що продуктивність JIT-компіляторів у порівнянні зі звичайними компіляторами у машинний код майже однакова. Проте ефективність компіляторів не завжди свідчить про швидкість виконання скомпільованого коду, тільки ретельне тестування може виявити справжню ефективність у даній системі.

Android сильно залежить від основних принципів Java. Android SDK складається з безлічі стандартних Java-бібліотек (бібліотеки структури даних, математичні бібліотеки, графічні бібліотеки, мережеві бібліотеки), а також спеціальні бібліотеки, які допоможуть розробити чудові програми для Android.

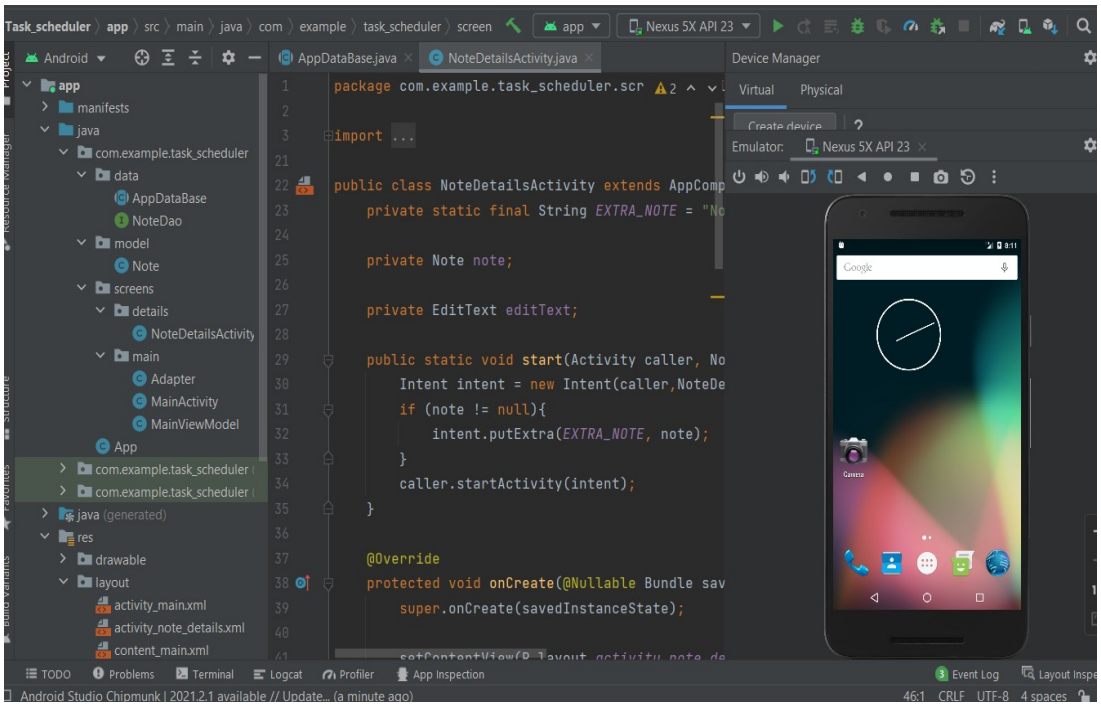

Рис. 2.1. Інтерфейс IDE Android Studio

Використані бібліотеки. Бібліотека Room надає зручну обгортку для роботи з базою даних SQLite. Room дозволяє зберігати дані в Android-додатку. Це частина нової Android Architecture, група бібліотек від Google, що підтримують доречну архітектуру додатків [14].

У Room є три основні компоненти:

1) клас database, який містить базу даних і є основною точкою доступу для базового з'єднання зі збереженими даними програми;

2) сутності даних, які представляють таблиці у базі даних докладання;

3) об'єкти доступу до даних (DAO), які надають методи, які програма може використовувати для запиту, оновлення, вставки та видалення даних у базі даних.

Архітектуру бібліотеки Room зображено на рис.2.2.

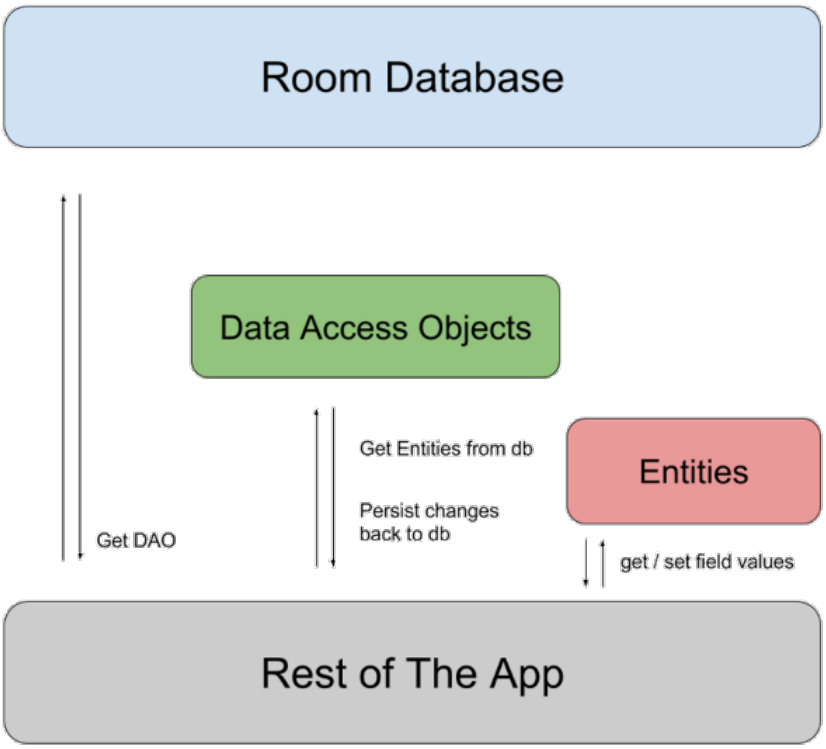

Рис.2.2. Схема архітектури Room

База даних надає додатку екземпляри DAO (Об'єкт доступу до даних), пов'язаних із цією базою даних. У свою чергу, програма може використовувати DAO для вилучення даних з бази даних як екземплярів пов'язаних об'єктів сутності даних. Додаток також може використовувати певні сутності даних для оновлення рядків із відповідних таблиць або для створення нових рядків для вставки.

Бібліотека Lifecycle дуже спрощує роботу з методами callback життєвого циклу та в цілому роботу з життєвим циклом активностей та фрагментів [15]. Основна суть цієї бібліотеки полягає в тому, що управлінням станом об'єктів повинні займатися не активності та фрагменти, що мають методи callback, а самі об'єкти, стан яких повинен змінюватися залежно від поточного статусу життєвого циклу.

SQLite – це вбудована бібліотека, яка реалізує самодостатню, безсерверну, транзакційну базу даних SQL без конфігурації. Код для SQLite знаходиться у відкритому доступі, тому його можна безкоштовно використовувати для будь-яких цілей, комерційних чи приватних. SQLite – це поширена база даних у світі з великою кількістю програм, включаючи кілька гучних проектів.

SQLite – це вбудований механізм бази даних SQL. На відміну від більшості інших баз даних SQL, SQLite не має окремого серверного процесу. SQLite читає та записує безпосередньо у звичайні дискові файли. Повна база даних SQL із кількома таблицями, індексами, тригерами та представленнями даних міститься в одному дисковому файлі. Формат файлу бази даних є крос-платформним — ви можете вільно копіювати базу даних між 32-розрядними та 64-розрядними системами або між архітектурами big-endian та little-endian. Ці функції роблять SQLite популярним вибором як формат файлу програми. Файли бази даних SQLite є рекомендованим форматом зберігання Бібліотекою Конгресу США. Думайте про SQLite не як про заміну Oracle, але як заміна fopen().

SQLite – це компактна бібліотека. Якщо всі функції ввімкнено, розмір бібліотеки може бути менше 750 КБ, залежно від цільової платформи та налаштувань оптимізації компілятора. (64-розрядний код більший. І деякі оптимізації компілятора, такі як агресивне вбудовування функцій і розгортання циклу, можуть призвести до того, що об’єктний код буде набагато більшим). Існує компроміс між використанням пам’яті та швидкістю. SQLite зазвичай працює швидше, чим більше пам’яті ви йому надаєте. Тим не менш, продуктивність зазвичай досить хороша навіть у середовищах з низьким обсягом

пам'яті. Залежно від того, як він використовується, SQLite може бути швидшим, ніж прямий ввід-вивід файлової системи .

SQLite дуже ретельно перевіряється перед кожним випуском і має репутацію дуже надійного. Більша частина вихідного коду SQLite присвячена виключно тестуванню та перевірці. Автоматизований набір тестів виконує мільйони і мільйони тестів із сотнями мільйонів окремих інструкцій SQL і досягає 100% тестового покриття гілок. SQLite витончено реагує на збої розподілу пам’яті та помилки дискового введення/виведення. Транзакції є ACID навіть якщо її переривають системні збої або збої в електроживленні. Все це перевіряється автоматизованими тестами з використанням спеціальних тестових джгутів, які імітують системні збої. Звичайно, навіть з усім цим тестуванням все ще є помилки. Але на відміну від деяких подібних проектів (особливо комерційних конкурентів), SQLite відкрито та чесно розповідає про всі помилки та надає списки помилок і щохвилинну хронологію змін коду.

База коду SQLite підтримується міжнародною командою розробників, які працюють над SQLite повний робочий день. Розробники продовжують розширювати можливості SQLite і підвищувати його надійність і продуктивність, зберігаючи при цьому зворотну сумісність з опублікованими специфікаціями інтерфейсу, синтаксисом SQL і форматом файлу бази даних. Вихідний код є абсолютно безкоштовним для будь-кого, але також доступна професійна підтримка.

2.4. Опис структури системи та алгоритмів її функціонування

Діаграма варіантів використання мобільного додатка представлена на рис. 2.3.

У ході проектування було виділено єдиний актор – користувач, який і взаємодіятиме з додатком.

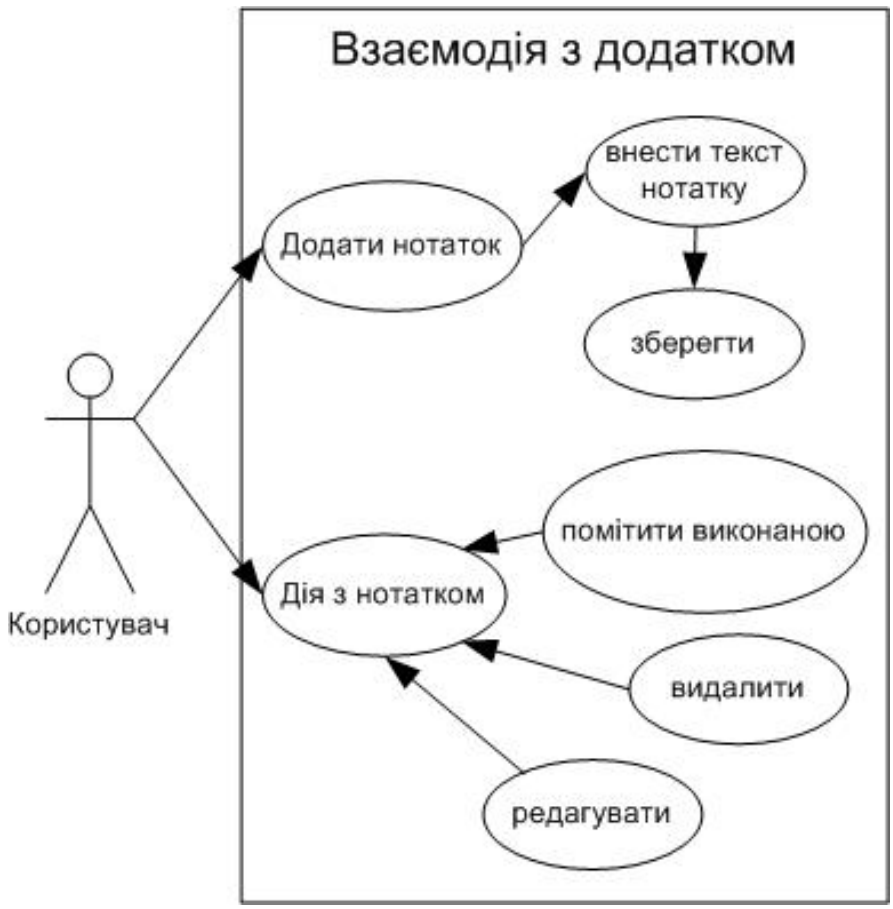

Рис.2.3. Діаграма варіантів використання

Користувач може додавати нотатки, натискаючи на певну кнопку програми.

Користувач потрапляє на другий екран та вводить певний текст. Потім необхідно натиснути кнопку для збереження введеної раніше нотатки.

Після введення нотатки, користувач повертається на початковий екран, і там на нього чекає вписана нотатка.

Для будь-якої нотатки є статус «Готово», його можна натиснути, нотатка закреслиться. Функція редагування дозволяє виправити записану нотатку раніше. Функція видалення видаляє нотатку з головного екрана і з програми.

Діаграма діяльності представлена у формі графа діяльності, у якого вершини – це стан дії, а дуги – переходи від одного стану до наступного.

Перша діаграма діяльності під час створення нотатки представлена на рис. 2.4.

Іноді при моделюванні течії бізнес-процесів буде корисніше розбити стан діяльності на діаграмах діяльності на деякі групи, кожна з яких представляє відділ компанії, який відповідає за ту чи іншу роботу. Групи називаються доріжками, оскільки візуально кожна група відокремлюється від сусідньої вертикальної межею, як доріжки в басейні. Доріжки – це різновид пакетів, що описують пов'язану сукупність робіт.

Кожній доріжці на діаграмі надається унікальне ім'я. В даному випадку їх 2: користувач і система додатка, що містить базу даних.

Початковий стан починається на доріжці користувача, оскільки саме після його дій процес запускається.

Насамперед, користувачеві потрібно створити замітку. Він натискає кнопку додавання, а система формує запит SQL, потім база даних виконує його.

Користувач друкує певний текст і зберігає його. Система обробляє запит та база даних виконує.

Після створення нотатки система перекидає користувача на головний екран нотаток, де вже буде створена нотатка. Потім кінцевий стан.

Тепер необхідно спроектувати діаграму діяльності, але вже для створеної замітки, тобто які можна виконати дії з нею і як вони будуть відображені в самій системі.

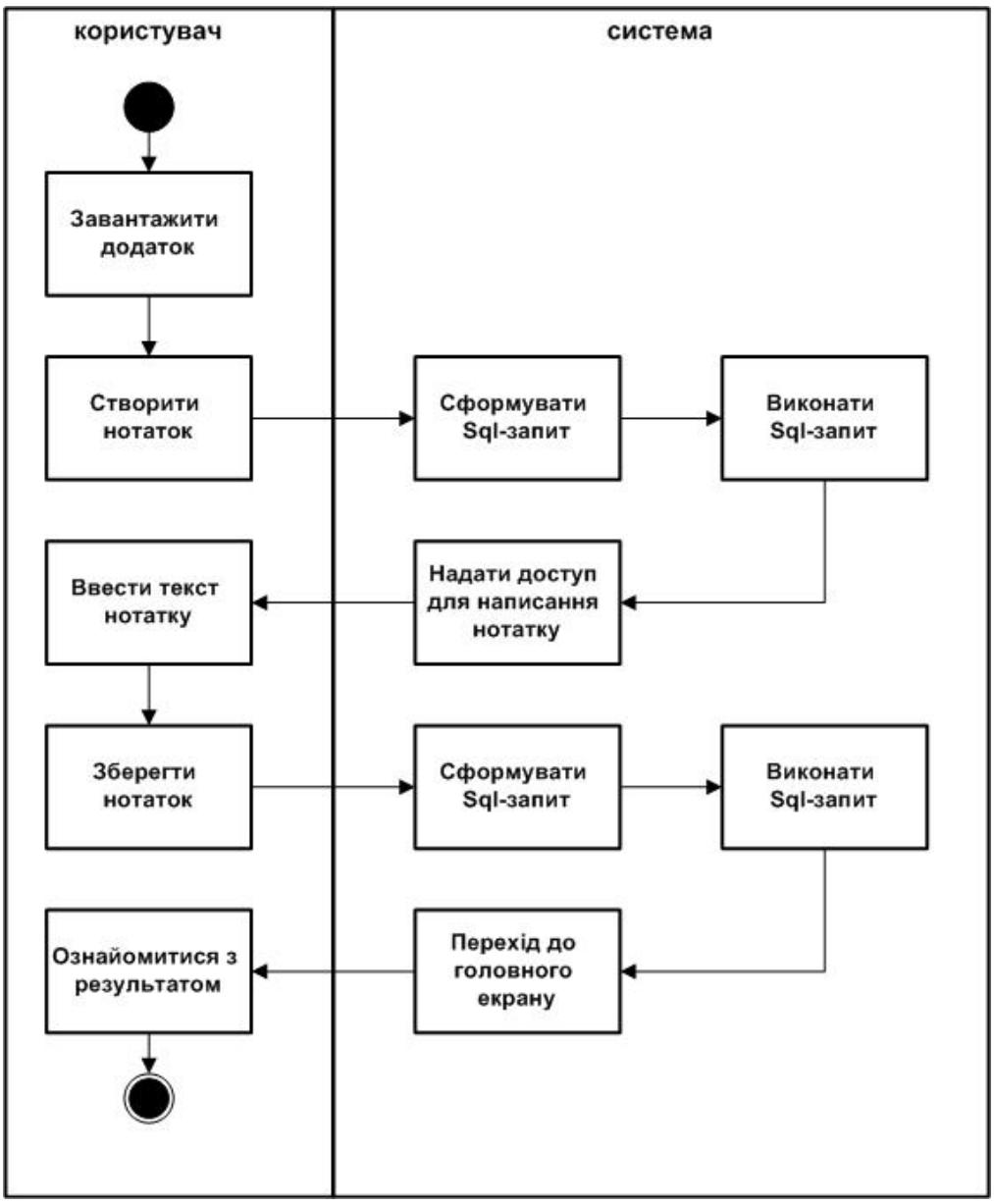

Рис. 2.4. Діаграма діяльності під час створення нотатки

На рис 2.5. продемонстровано діаграму діяльності при редагуванні нотатки.

Однією з дій є вибір статусу готовності для справи. Користувач натискає Готово, далі система формує запит, БД його виконує. Після всіх дій, користувачеві відображається перекреслене (неактивне) завдання на головній сторінці програми.

Друга дія, яку можна виконати – це видалити нотатку. Користувач натискає певну кнопку, щоб видалити нотатку. Система надсилає запит базі даних на видалення, і нотатка успішно видаляється.

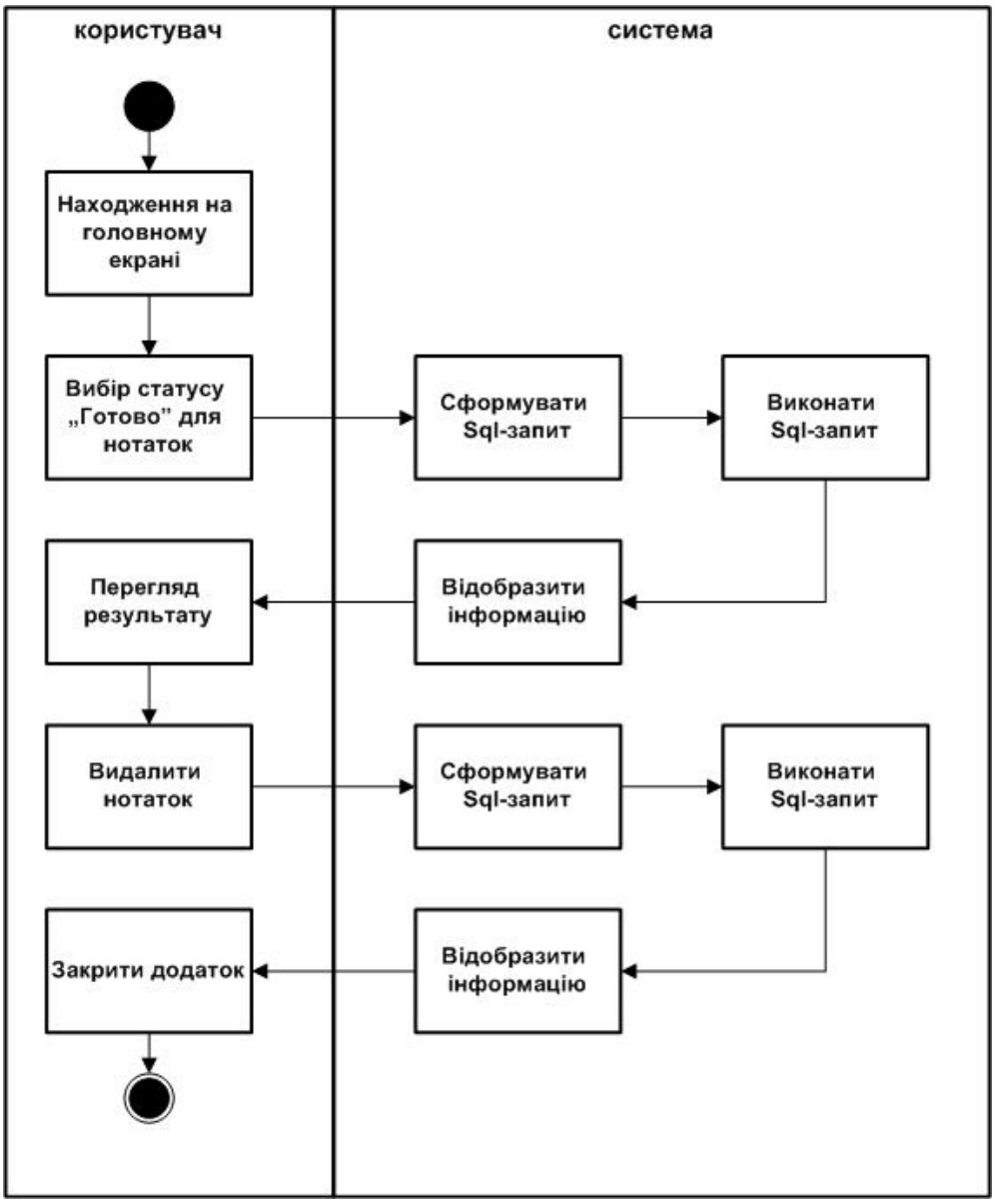

Рис. 2.5. Діаграма при редагувані замітки

Архітектура програми. Програма складається з 4 шарів. Зазвичай, варто починати з внутрішнього - шар сутностей, його називають шаром моделі, в ньому міститься вся логіка.

Допоміжний шар попереднього – шар варіантів використання, він включає бізнес-логіку.

Третій шар – шар презентації, який містить у собі патерни та адаптери інтерфейсу.

Останній шар, з яким взаємодіє користувач, – шар інтерфейсу.

Архітектуру програми зображено на рис. 2.6.

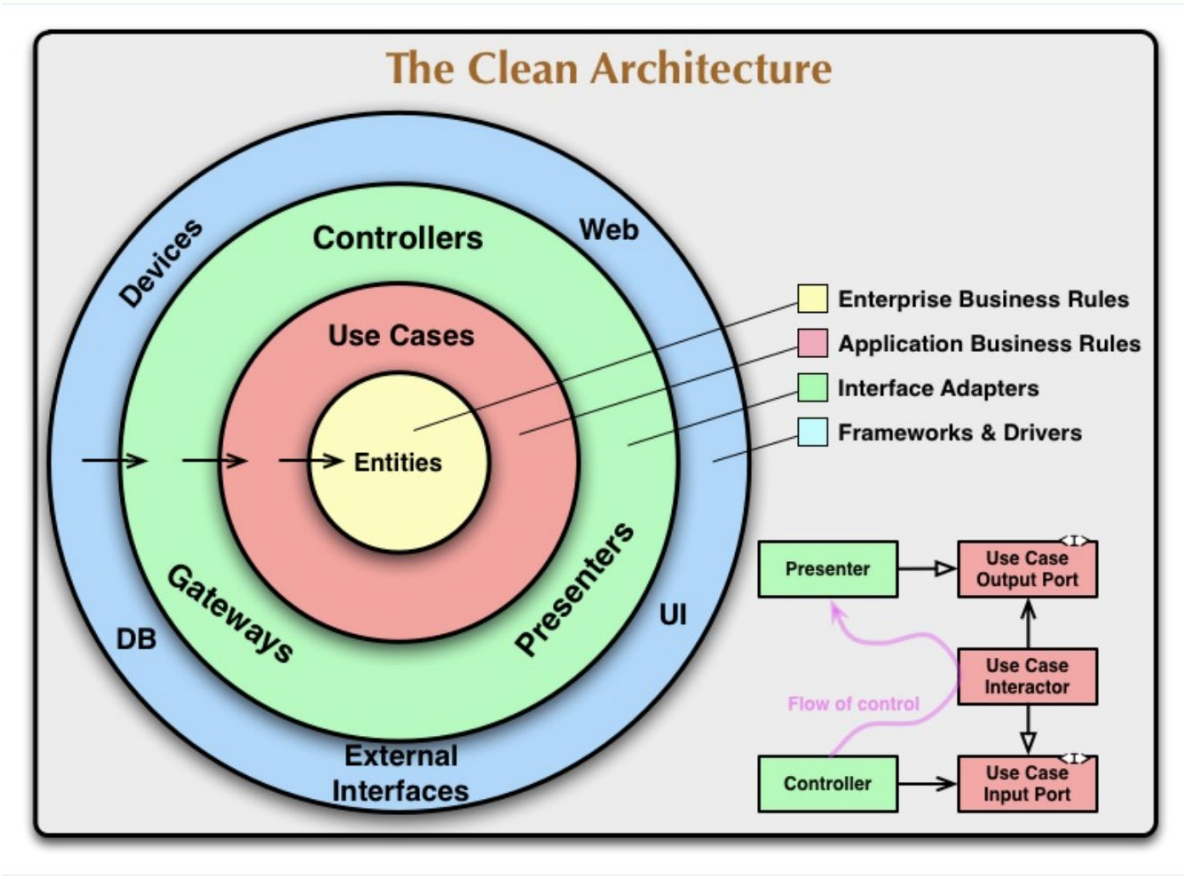

Рис. 2.6. Схематичне зображення чистої архітектури

Реалізація бази даних. Шар бази даних.

Було створено об'єкт NoteDao, через який здійснюватиметься доступ до таблиці БД – запис, читання та ін. операції.

Створення шару БД наведено в лістінгу 2.1.

OnConflictStrategy.REPALCE - означає, що якщо користувач захоче вставити замітку в БД з ID, який вже існує, то замість вставки буде зроблена заміна старої сутності на нову [1].

@Delete – видаляє існуючу нотатку.

@Update – функція оновлення.

@Query getAll - вибір всіх (нотаток).

@Query getAllLiveData – повертає не просто список, а й

LiveData.

LiveData хороший тим, що це спеціальний об'єкт, на який може підписатися інтерфейс користувача і щоразу, коли в таблиці з нотатками буде щось змінюватися, LiveData стане оновлюватися і повідомляти всім своїм передплатникам, що дані оновилися. Автоматично це не доведеться писати, це дуже зручно, щоб оновлювати список з даними.

@Query loadAllByIds – завантажити всі нотатки, які мають Id зі списку. Двокрапка перед параметром означає, що це місце підставляється параметр з відповідним ім'ям, тобто. автоматична підстановка [1].

findById – вибірка з Id.

```
        @Query("SELECT * FROM Notes WHERE uid = :uid LIMIT
1")     Note findById(int uid);

        @Query("SELECT * FROM Notes WHERE uid IN (:userIds)")
List<Notes> loadAllByIds(int[] userIds);

        @Query("SELECT * FROM Notes")
        List<Notes> getAll();

        @Query("SELECT * FROM Notes")
        LiveData<List<Notes>> getAllLiveData();

        @Insert(onConflict                          =
OnConflictStrategy.REPLACE)
        @Delete
        void delete(Notes notes);

        @Update
             void update(Notes notes);
```

Лістінг 2.1. Реалізація шару БД (NoteDao)

Основна частина бази даних. Тепер потрібно створити клас AppDataBase. Цей клас є всі.ю БД загалом, тобто. якщо клас Note визначає конкретну сутність, NoteDao визначає інтерфейс і виконання запитів у БД. AppDataBase вже відноситься до всіх сутностей одразу, повністю вся БД. Необхідно зазначити, які сутності у ній будуть.

У разі вона одна, Note [7].

Вибір версії – важливий параметр, при розробці програми не особливо потрібен, але коли програма вже перебуває у працездатному стані, то при оновленні програми якщо знадобиться додати нові таблиці та нові поля до вже існуючих таблиць, навряд чи хтось захоче щоразу видаляти БД та створювати її наново. Тому використовується механізм версій, знаючи яка поточна версія і яка в новій БД можна виконати скрипт міграції, дії над базою, перш за все робиться щоб оновити з однієї версії на іншу, за умови, що минулі дані залишаться на місці.

База даних повинна містити абстрактний метод, який дозволяє отримати доступ до NoteDataAccessObject поточної моделі.

Коли клас створюється, автоматично генерується (за допомогою бібліотеки Room) реалізація з класу NoteDao. Потрібно просто надати опис для БД, все інше Room бере на себе.

БД одна на всю програму і логічно буде створити її при старті програми, і щоб вона існувала протягом усього часу, поки працює програма. Щоб реалізувати цю логіку, необхідний об'єкт, який живе під час роботи програми.

Реалізація бази даних, абстрактного класу, представлено в лістінгу 2.2.

```
@Database(entities = {Notes.class}, version = 1, exportSchema = false) public abstract class AppDataBase extends RoomDatabase { public abstract NoteDao as noteDao();}
```

Лістінг 2.2. Реалізація класу AppDataBase

Реалізація класу App. Реалізація класу наведено в лістінгу 2.3.

У Application є метод onCreate, цей метод гарантовано буде викликаний до того, як почнеться повноцінна робота програми та користувач зможе взаємодіяти зі стартовим вікном і т.д., дана ініціалізація виконається за будь-якого старту програми, за будь-якого старту процесу.

Метод onCreate() буде викликаний під час створення або перезапуску активності. Сама система може запускати та зупиняти поточні вікна залежно від

того, які події відбуваються. Усередині цього методу налаштовують статичний інтерфейс активності. Також відбувається ініціалізація статичних даних активності, що пов'язують дані зі списками тощо. Метод пов'язує з необхідними даними та ресурсами.

Створюється БД і туди передається контекст програми, потім клас, в якому опис БД та її ім'я. Запит до бази робиться з основного потоку (.allowMainthreadQueries). Зазвичай у додатку запити до БД займають дуже багато часу і в основному не слід, варто виносити всі такі дії та операції в потоках background.

Після створення бази, необхідно отримати NoteDao із запитами.

.allowMainThreadQueries() відключає перевірку запиту основного потоку на наявність місця.

Метод build() у класі використовується для побудови потоку.

Повертає збудований потік.

super.onCreate є callback (зворотний виклик) між станами. ОС викликає ці callback, коли діяльність переходить з одного стану в інший.

```
public class App extends Application {
     private AppDataBase database;
     private static App instance;
     public static App getInstance()
{return instance;}
    @Override
    public void onCreate(){
        super.onCreate(savedInstanceState); instance = this;
        database                                                    =
Room.databaseBuilder(getApplicationContext(),
                AppDataBase.class, "app-db-ts")
                .allowMainThreadQueries()
                .build();
        noteDao = database.noteDao();
     public AppDataBase getDatabase()
{ return database; }
    public    void    setDatabase(AppDataBase    database)    {
this.database = database;
    }
    public NoteDao getNoteDao()
```

```
{ return noteDao; }
    public void setNoteDao(NoteDao noteDao)
{ this.noteDao = noteDao; }
}
```

Лістінг 2.3. Реалізація класу App

Реалізація шару презентації.

ViewModel – клас, що дозволяє Activity та фрагментам зберігати необхідні їм об'єкти живими при повороті екрана.

Після створення шару БД варто перейти до наступного – шар презентації.

Потрібно надати доступ до даних про список нотаток, щоб відобразити цей список на головному екрані.

LiveData – список нотаток.

Реалізація представлено в лістінгу 2.4.

```
public class MainViewModel extends ViewModel
{     private     LiveData<List<Note>>     noteLiveData     =
App.getInstance().getNoteDao().getAllLiveData(); }
public LiveData<List<Note>> getNoteLiveData()
{ return noteLiveData; }
```

Лістінг 2.4. Реализация класса ViewModel

Реалізація інтерфейсу користувача. Останній шар – верстка графічного інтерфейсу.

Графічний інтерфейс для Android програми будується на основі ієрархії View та ViewGroup об'єкти. View об'єкти – це віджети інтерфейсу користувача, такі як кнопки або текстові поля і ViewGroup – це невидимий вид контейнерів, які визначають розташування дочірніх уявлень, як наприклад, у сітці або вертикальному списку [6].

Більшою мірою верстка реалізується у файлах проекту з розширенням xml [8].

XML розшифровується як Extensible Markup Language.

Мова розмітки трохи відрізняється від мови програмування. У той час як мова програмування (C#, C++, Java, Kotlin, Python, BASIC) дозволяє визначати поведінку, взаємодію та умови; мова розмітки використовується більше для опису даних та, в даному випадку, макетів. Мови програмування створюють динамічну взаємодію, тоді як мови розмітки зазвичай обробляють такі речі, як статичні інтерфейси користувача.

Верстка першого (головного) вікна представлено в лістінгу 2.5.

```
     <androidx.coordinatorlayout.widget.CoordinatorLayout
xmlns:android="schemas.android.com/apk/res/android"
xmlns:app="schemas.android.com/apk/res-auto"
xmlns:tools="schemas.android.com/tools"
android:layout_height="match_parent"
android:layout_width="match_parent"
tools:context=".screens.main.Main">

        <com.google.android.material.appbar.AppBarLayout
android:layout_height="wrapout_content"
android:layout_width="match_parent"

android:theme="@style/Theme.Task_scheduler.AppBarOverlay">
            <androidx.widget.Toolbar
android:id="@+id/toolbar"
                android:layout_height="?attr/actionBarSize"
android:layout_width="match_parent"
android:background="?attr/colorPrimary"

app:popupTheme="@style/Theme.Task_scheduler.PopupOverlay" />

</com.google.android.material.appbar.AppBarLayout>
        <include layout="@layout/content_main" />
    <com.google.android.material.floatingactionbutton.FloatingAct
ionButton         android:id="@+id/fab"
            android:layout_height="wrapout_content"
android:layout_width="wrap_content"
android:layout_margin="@dimen/fab_margin"
android:layout_gravity="bottom|end"

app:srcCompat="@android:drawable/ic_input_adding"/>
    </androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Лістінг 2.5. Верстка головного вікна

android:layout_height є висотою. android:layout_width є шириною.

Для встановлення зовнішніх відступів використовується атрибут layout_margin.

match_parent дозволяє розтягнути елемент по всій ширині чи висоті контейнера. Варто зазначити, що це значення застосовується до всіх контейнерів, крім ConstraintLayout.

Також у додатку є друге вікно, для введення нотатки та подальшим збереженням. В лістінгу 2.6. реалізація другого вікна.

```
    <LinearLayout
xmlns:android="schemas.android.com/apk/res/android"
xmlns:app="schemas.android.com/apk/res-auto"
android:orientation="vertical"
android:layout_height="match_parent"
android:layout_width="match_parent">

        <androidx.appcompat.widget.Toolbar
android:id="@+id/toolbar"
            android:layout_height="?attr/actionBarSize"
android:layout_width="match_parent"
android:background="?attr/colorPrimary" />

        <EditText
            android:id="@+id/text"
            android:layout_height="wrap_content"
android:layout_width="match_parent"
android:hint="@string/note_text"
android:minHeight="210dp"
android:gravity="top"        android:padding="20dp"/>

    </LinearLayout>
```

Лістінг 2.6. Верстка другого вікна

За допомогою android:orientation задається орієнтація розмітки.

android:minHeight обмежує мінімальні значення.

Атрибут android:background визначає фоновий колір елемента.

Для встановлення внутрішніх відступів використовується атрибут android:padding.

Атрибут android:gravity відповідає за вирівнювання вмісту всередині елемента.

2.5. Обгрунтування та організація вхідних та вихідних даних програми

Вхідними даними є введені користувачем дані або обрані управляючі команди.

Вихідними даними є дані, що отримує користувач від системи.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Для написання кваліфікаційної роботи та прогорамного забезпечення використовувався ноутбук Ноутбук Dell Vostro 15 3500 знаступними технічними характеристиками: екран 15.6" WVA (1920x1080) Full HD, / Intel Core i3-1115G4 (3.9 — 4.1 ГГц) / RAM 8 ГБ / SSD 256 ГБ / Intel UHD / без ОД / LAN / Wi-Fi / Bluetooth / вебкамера.

2.6.2. Використані програмні засоби

Для написання програмного продукту було обрано мову програмування Java та середовище AndroidStudio.

Система керування базами даних SQLite

2.6.3. Виклик та завантаження програми

Спосіб виклику програми з відповідного носія даних та умови його завантаження є стандартними для запуску мобільних додатків та може завантажуватись на смартфонах різних типів та конфігурацій під управлінням різних ОС Андроїд.

Для того щоб встановити додаток на мобільний пристрій потрібно обрати файл інсталятор в файловій системі мобільного, файл має розширення .apk

2.6.4. Опис інтерфейсу користувача

Користувацький інтерфейс мобільного додатку є зручним та інтуітивно-зрозумілим, також має приємну кольорову гаму. В свою чергу єсумісним з потребами та можливостями користувача і забезпечувати простоту переходу від виконання однієї функції до іншої.

Скриншоти разробленого додатку приведено на рис. 2.7.–2.9.

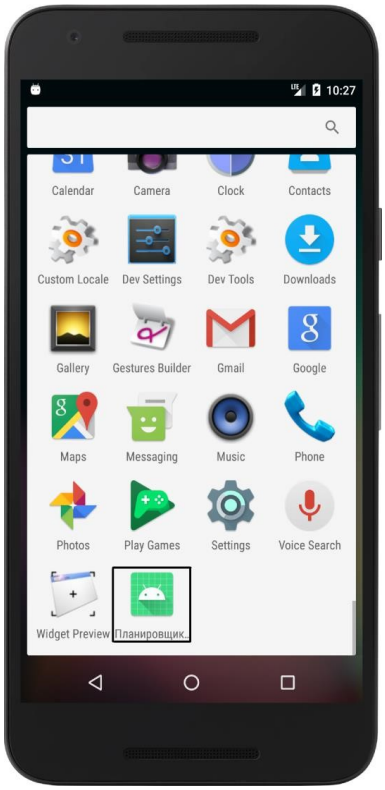

Рис.2.7. Скріншот іконки додатку

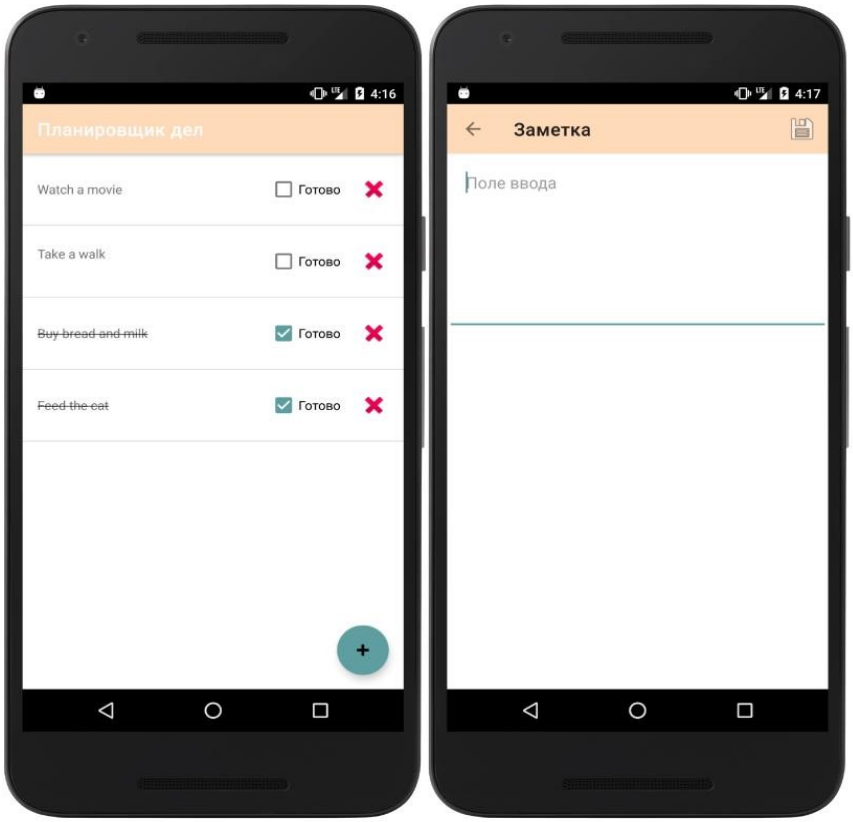

Рис. 2.8. Скріншоти додатку з зображенням нотатків

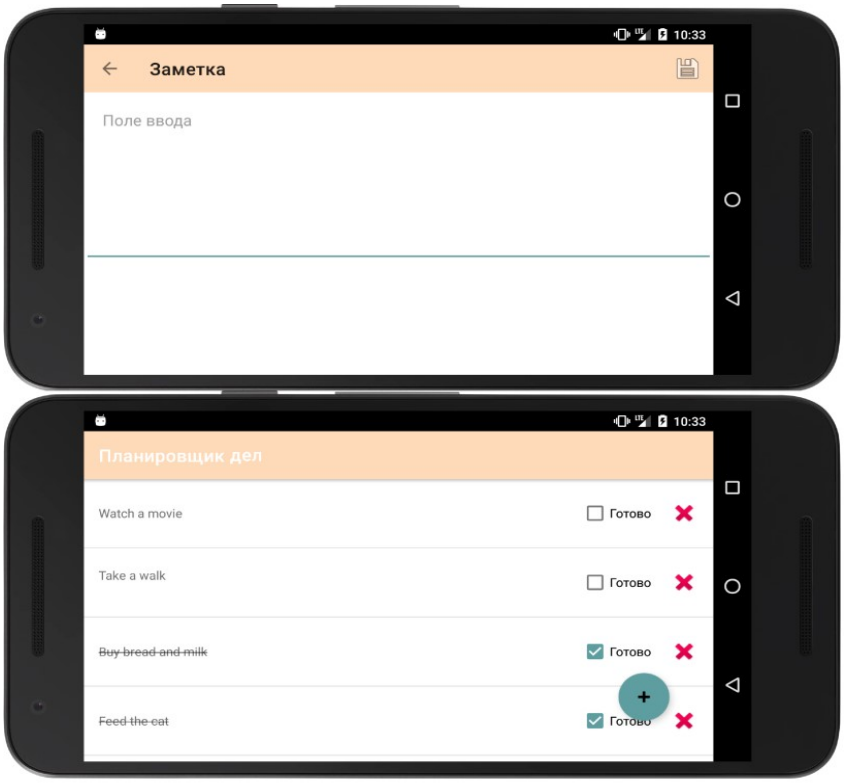

Рис. 2.9. Скріншоти горизонтального розміщення єкрану

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки інформаційної системи

Початкові дані:

1. передбачуване число операторів програми – 1200;
2. коефіцієнт складності програми – 1,6;
3. коефіцієнт корекції програми в ході її розробки – 0,05;
4. годинна заробітна плата програміста – 120 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;
7. вартість машино-години ЕОМ – 13 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \quad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмногму забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q - передбачуване число операторів (1200);

C - коефіцієнт складності програми (1,6);

p - коефіцієнт корекції програми в ході її розробки (0,05).

Звідси умовне число операторів в програмі:

$$Q = 1{,}6 \cdot 1200 \cdot (1 + 0{,}05) = 2016$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1.2). З урахуванням коефіцієнта кваліфікації k = 1,2, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (2016 \cdot 1{,}2) / (75 \cdot 1{,}2) = 26{,}88 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин,}\quad (3.2)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 2016 / (20 \cdot 1{,}2) = 84 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин.}$$

$$t_n = 2016 / (25 \cdot 1{,}2) = 67{,}2 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\ \text{, людино-годин.}$$

$$t_{отл} = 2016 / (5 \cdot 1{,}2) = 336 \text{ людино-годин.}$$

- за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 \cdot t_{отл}\ \text{, людино-годин.}$$

$$t^{k}_{отл} = 1,5 \cdot 336 = 504 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{д} = t_{др} + t_{до} \text{ , людино-годин,}$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ , людино-годин,}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0,75 \cdot t_{др} \text{ , людино-годин.}$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = 2016 / (18 \cdot 1,2) = 93,33 \text{ людино-годин.}$$

$$t_{до} = 0,75 \cdot 93,33 = 70 \text{ людино-годин.}$$

$$t_{д} = 93,33 + 70 = 163,33 \text{ людино-годин.}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 26,88 + 84 + 67,2 + 336 + 163,33 = 727,41 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,}$$

де: t - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 120 грн / год, отримуємо:

$$З_{ЗП} = 727{,}41 \cdot 120 = 87\,289{,}2 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot С_{мч}, \text{ грн,} \quad (3.3)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ - вартість машино-години ЕОМ, грн/год (13 грн/год).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 336 \cdot 13 = 4368 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 87\ 289{,}2 + 4368 = 91\ 657{,}2 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.}$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Звідси витрати на створення програмного продукту:

$$T = 727{,}41 / 1 \cdot 176 \approx 4 \text{ міс.}$$

Висновок. Вартість розробленого програмного забезпечення становить 91 657,2. грн. і не вимагає додаткових витрат при розробці програми. Очікуваний час розробки становить 4 місяці. Цей термін пов'язаний зі значним числом операторів, і включає час на дослідження і розробку алгоритму вирішення поставленого завдання, програмування по готовому алгоритму, налагодження програми і підготовку документації.

ВИСНОВКИ

В рамках даної кваліфікаційної роботи було розроблено програму для мобільних пристроїв «Планувальник справ» для операційних систем Android, що дозволяє створювати нотатки для виконання певних побутових та інших справ. Було вирішено такі завдання.

1. Зроблено аналіз предметної галузі та аналогів.

2. Спроектовано архітектуру програми.

3. Реалізовано додаток.

4. Виконано тести на працездатність.

Обсяг підсумкового додатка становив близько 1200 операторів.

Для написання програмного продукту було обрано мову програмування Java та середовище AndroidStudio. Для роботи з даними було обрано та використано систему керування базами даних SQLite.

Надалі планується розширити функціонал програми.

Додадуться такі функції.

1. Версія для електронного розумного годинника (Samsung, Huawei, Xiaomi та ін.).

2. Функція таймер-нагадування.

3. Версія для iOS.

4. Функція «Архів» (кошик), де зберігатимуться віддалені нотатки обмежений час.

5. Віджет, щоб виставити нотатки на головному екрані мобільного пристрою.

Також у кваліфікаціфйній роботі було визначено трудомісткість розробленої системи 727 люд-год., проведений підрахунок вартості роботи по створенню програми 91 657,2 грн. та розраховано час на її створення майже 4 місяца.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android Studio Documentation [Електроний ресурс] URL:https://developer.android.com/docs.
2. Блокнот: Швидкі нотатки [Електроний ресурс] URL:https://play.google.com/store/apps/details?id=com.komorebi.memo
3. BasicNote - Нотатки, Блокнот [Електроний ресурс] URL: https://play.google.com/store/apps/details?id=notizen.basic.notes.notas.note.note pad
4. Побудова простого інтерфейсу користувача [Електроний ресурс] URL: https://www.fandroid.info/postroenie-prostogopolzovatelskogo-interfejsa
5. Робота з базами даних SQLite в Android [Електроний ресурс] URL: https://www.fandroid.info/urok-34-rabota-s-bazami-dannyh-sqlite-vandroid
6. Визначення інтерфейсу у файлі XML [Електроний ресурс] URL: https://metanit.com/java/android/2.8.php
7. Паттерни проектування [Електроний ресурс] URL: https://habr.com/ua/company/ruvds/blog/427293/?reply_to=19279145.
8. Архітектурні шаблони [Електроний ресурс] URL: https://studfile.net/tpreview/9993795
9. Гріффітс Д., Гріффітс Д. Head First. Програмування для Android. - 2016. - 704 с.: Іл.
10. Дейтел П., Дейтел Х., Уолд А. Android для розробників. - 3-тє вид. -, 2016. - 512 с.: Іл.
11. Wiley J., Sons J. Початок Java Programming: The Object-Oriented Approach. - Indianapolis: Crosspoint Boulevard, 2015. - 672 p.
12. Сьєрра К., Бейтс Б. Вивчаємо Java. - 2015. - 720 с.
13. Савітч Уолтер. Мова Java. Курс програмування / Волтер Савітч. – К.: 2015. – 928 с.
14. Хабібуллін Ільдар. Самовчитель Java / Ільдар Хабібуллін. – К.:, 2014. – 768 с.

15. Шілдт Герберт. Java 8. Посібник для початківців / Герберт Шілдт. – К.: Вільямс, 2015. – 720 с.

16. Еккель Брюс. Філософія Java / Брюс Еккель. – К.: 2016. – 809 с.

17. Barry Burd. Android Application Development All-in-One For Dummies® / Barry Burd. – 2011. – 816 с.

18. Гарнаєв Андрій. WEB-програмування на Java та JavaScript / Андрій Гарнаєв, Сергій Гарнаєв. - 2012. - 179 с.

19. Голощапов Олексій. Google Android. Програмування для мобільних пристроїв (CD-ROM) / Олексій Голощапов. - 2011. - 438 с.

20. Методичні рекомендації до виконання кваліфікаційних робіт здобувачів першого рівня вищої освіти спеціальності 122 Комп'ютерні науки/, О.С. Шевцова; Д : НТУ «Дніпровська політехніка», 2021. – 65 с.

ДОДАТОК А

КОД ПРОГРАМИ

```
package com.example.task_scheduler;

import android.app.Application;

import androidx.room.Room;

import com.example.task_scheduler.data.AppDataBase;
import com.example.task_scheduler.data.NoteDao;

public class App extends Application {

    private AppDataBase database;
    private NoteDao noteDao;

    private static App instance;

    public static App getInstance(){
        return instance;
    }

    @Override
    public void onCreate(){
        super.onCreate();

        instance = this;

        database = Room.databaseBuilder(getApplicationContext(),
                AppDataBase.class, "app-db-ts")
                .allowMainThreadQueries()
                .build();

        noteDao = database.noteDao();
    }

    public AppDataBase getDatabase() {
        return database;
    }

    public void setDatabase(AppDataBase database) {
        this.database = database;
    }
```

```
    public NoteDao getNoteDao() {
        return noteDao;
    }

    public void setNoteDao(NoteDao noteDao) {
        this.noteDao = noteDao;
    }
}
```

```
package com.example.task_scheduler.model;

import android.os.Parcel;
import android.os.Parcelable;

import androidx.room.ColumnInfo;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

// создание полей
@Entity
public class Note implements Parcelable {

    @PrimaryKey(autoGenerate = true) // автоматическое создание
ключей
    public int uid; // уникальный идентификатор для каждой сущности

    @ColumnInfo(name = "text")
    public String text; // текст заметки

    @ColumnInfo(name = "timestamp")
    public long timestamp; // когда заметка была создана

    @ColumnInfo(name = "done")
    public boolean done; // поле, чтобы узнать, сделано или не
сделано дело

    public Note() {

    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
```

```
        Note note = (Note) o;

        if (uid != note.uid) return false;
        if (timestamp != note.timestamp) return false;
        if (done != note.done) return false;
        return text != null ? text.equals(note.text) : note.text ==
null;
    }

    @Override
    public int hashCode() {
        int result = uid;
      result = 31 * result + (text != null ? text.hashCode() : 0);
    result = 31 * result + (int) (timestamp ^ (timestamp >>> 32));
        result = 31 * result + (done ? 1 : 0);
        return result;
    }
    protected Note(Parcel in) {
        uid = in.readInt();
        text = in.readString();
        timestamp = in.readLong();
        done = in.readByte() != 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeInt(uid);
        dest.writeString(text);
        dest.writeLong(timestamp);
        dest.writeByte((byte) (done ? 1 : 0));
    }

    @Override
    public int describeContents() {
        return 0;
    }

    public static final Creator<Note> CREATOR = new Creator<Note>()
{
        @Override
        public Note createFromParcel(Parcel in) {
            return new Note(in);
        }

        @Override
```

```
        public Note[] newArray(int size) {
            return new Note[size];
        }
    };
}
```

```
package com.example.task_scheduler.data;

import androidx.lifecycle.LiveData;
import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.OnConflictStrategy;
import androidx.room.Query;
import androidx.room.Update;

import com.example.task_scheduler.model.Note;

import java.util.List;

// объект, через который будет осуществляться доступ к таблице БД
(запись, чтение и тд)
@Dao // запросы
public interface NoteDao {
    @Query("SELECT * FROM Note")
    List<Note> getAll();

    @Query("SELECT * FROM Note")
    LiveData<List<Note>> getAllLiveData();

    @Query("SELECT * FROM Note WHERE uid IN (:userIds)")
    List<Note> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM Note WHERE uid = :uid LIMIT 1")
    Note findById(int uid);

    @Insert(onConflict = OnConflictStrategy.REPLACE) // замена
старой заметки на новую
    void insert(Note note);

    @Update
    void update(Note note);

    @Delete
    void delete(Note note);
```

```
}

package com.example.task_scheduler.data;

import androidx.room.Database;
import androidx.room.RoomDatabase;

import com.example.task_scheduler.model.Note;

@Database(entities = {Note.class}, version = 1, exportSchema =
false)
public abstract class AppDataBase extends RoomDatabase {
    public abstract NoteDao noteDao();

}



package com.example.task_scheduler.screens.main;

import android.app.Activity;
import android.graphics.Paint;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;

import com.example.task_scheduler.App;
import com.example.task_scheduler.R;
import com.example.task_scheduler.model.Note;
import
com.example.task_scheduler.screens.details.NoteDetailsActivity;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import androidx.recyclerview.widget.SortedList;

import java.util.List;

public              class              Adapter              extends
RecyclerView.Adapter<Adapter.NoteViewHolder> {

    private SortedList<Note> sortedList;
```

```
    public Adapter(){
        sortedList = new SortedList<>(Note.class, new
SortedList.Callback<Note>() {
            @Override
            public int compare(Note o1, Note o2) {
                if (!o2.done && o1.done){
                    return 1;
                }
                if (o2.done &&!o1.done){
                    return -1;
                }
                return (int) (o2.timestamp - o1.timestamp);
            }

            @Override
            public void onChanged(int position, int count) {
                notifyItemRangeChanged(position, count);
            }

            @Override
            public boolean areContentsTheSame(Note oldItem, Note
newItem) {
                return oldItem.equals(newItem);
            }

            @Override
            public boolean areItemsTheSame(Note item1, Note item2)
{
                return item1.uid == item2.uid;
            }

            @Override
            public void onInserted(int position, int count) {
                notifyItemRangeInserted(position, count);
            }

            @Override
            public void onRemoved(int position, int count) {
                notifyItemRangeRemoved(position, count);
            }

            @Override
            public void onMoved(int fromPosition, int toPosition) {
                notifyItemMoved(fromPosition, toPosition);
            }
        });
```

```
    }

    @NonNull
    @Override
    public NoteViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
        return new NoteViewHolder(

LayoutInflater.from(parent.getContext()).inflate(R.layout.item_not
e_list, parent, false));
    }

    @Override
    public void onBindViewHolder(@NonNull NoteViewHolder holder, int
position) {
        holder.bind(sortedList.get(position));
    }

    @Override
    public int getItemCount() {
        return sortedList.size();
    }

    public void setItems(List<Note> notes){
        sortedList.replaceAll(notes);
    }

    static class NoteViewHolder extends RecyclerView.ViewHolder{

        TextView noteText;
        CheckBox completed;
        View delete;

        Note note;

        boolean silentUpdate;

        public NoteViewHolder(@NonNull View itemView){
            super(itemView);

            noteText = itemView.findViewById(R.id.note_text);
            completed = itemView.findViewById(R.id.completed);
            delete = itemView.findViewById(R.id.delete);

            itemView.setOnClickListener(new View.OnClickListener()
{
```

```
                @Override
                public void onClick(View view) {

NoteDetailsActivity.start((Activity)itemView.getContext(), note);
                }
            });

            delete.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    App.getInstance().getNoteDao().delete(note);
                }
            });

            completed.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
                @Override
                public     void     onCheckedChanged(CompoundButton
buttonView, boolean isChecked) {
                    if (!silentUpdate){
                        note.done = isChecked;

App.getInstance().getNoteDao().update(note);
                    }
                    updateStrokeOut();
                }
            });
        }

        public void bind(Note note){
            this.note = note;

            noteText.setText(note.text);
            updateStrokeOut();

            silentUpdate = true;
            completed.setChecked(note.done);
            silentUpdate = false;
        }

        private void updateStrokeOut(){
            if (note.done){
                noteText.setPaintFlags(noteText.getPaintFlags()   |
Paint.STRIKE_THRU_TEXT_FLAG);
            } else{
```

```
                noteText.setPaintFlags(noteText.getPaintFlags()  &~
Paint.STRIKE_THRU_TEXT_FLAG);
            }
        }
    }
}
```

```
package com.example.task_scheduler.screens.main;

import android.os.Bundle;

import com.example.task_scheduler.R;

import com.example.task_scheduler.model.Note;
import
com.example.task_scheduler.screens.details.NoteDetailsActivity;
import
com.google.android.material.floatingactionbutton.FloatingActionBut
ton;

import androidx.appcompat.app.AppCompatActivity;


import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProviders;
import androidx.recyclerview.widget.DividerItemDecoration;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import androidx.appcompat.widget.Toolbar;

import android.view.View;

import java.util.List;


public class MainActivity extends AppCompatActivity {

    private RecyclerView recyclerView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
```

```
        setSupportActionBar(toolbar);

        recyclerView = findViewById(R.id.list);
        LinearLayoutManager     linearLayoutManager     =     new
LinearLayoutManager(this, RecyclerView.VERTICAL,false);
        recyclerView.setLayoutManager(linearLayoutManager);
        recyclerView.addItemDecoration(new
DividerItemDecoration(this, DividerItemDecoration.VERTICAL));

        Adapter adapter = new Adapter();
        recyclerView.setAdapter(adapter);

        FloatingActionButton fab = findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
            NoteDetailsActivity.start(MainActivity.this, null);
            }
        });

        MainViewModel                mainViewModel                =
ViewModelProviders.of(this).get(MainViewModel.class);
        mainViewModel.getNoteLiveData().observe(this,           new
Observer<List<Note>>() {
            @Override
            public void onChanged(List<Note> notes) {
                adapter.setItems(notes);
            }
        });
    }
}
```

```
package com.example.task_scheduler.screens.main;

import androidx.lifecycle.LiveData;
import androidx.lifecycle.ViewModel;

import com.example.task_scheduler.App;
import com.example.task_scheduler.model.Note;

import java.util.List;

public class MainViewModel extends ViewModel {
```

```
    private        LiveData<List<Note>>        noteLiveData        =
App.getInstance().getNoteDao().getAllLiveData();

    public LiveData<List<Note>> getNoteLiveData() {
        return noteLiveData;
    }
}
```

```
package com.example.task_scheduler.screens.details;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Parcelable;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;


import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import com.example.task_scheduler.App;
import com.example.task_scheduler.R;
import com.example.task_scheduler.model.Note;

public class NoteDetailsActivity extends AppCompatActivity {
    private     static     final     String     EXTRA_NOTE     =
"NoteDetailsActivity.EXTRA_NOTE";

    private Note note;

    private EditText editText;

    public static void start(Activity caller, Note note){
        Intent              intent              =              new
Intent(caller,NoteDetailsActivity.class);
        if (note != null){
            intent.putExtra(EXTRA_NOTE, note);
        }
        caller.startActivity(intent);
```

```
    }

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState){
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_note_details);

        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setHomeButtonEnabled(true);

        setTitle(getString(R.string.note_details_title));
        editText = findViewById(R.id.text);

        if (getIntent().hasExtra(EXTRA_NOTE)){
            note = getIntent().getParcelableExtra(EXTRA_NOTE);
            editText.setText(note.text);
        } else {
            note = new  Note();
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu){
        getMenuInflater().inflate(R.menu.menu_details,menu);
        return super.onCreateOptionsMenu(menu);
    }

    @SuppressLint("NonConstantResourceId")
    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item){
        switch (item.getItemId()){
            case android.R.id.home:
                finish();
                break;
            case R.id.action_save:
                if (editText.getText().length() > 0){
                    note.text = editText.getText().toString();
                    note.done = false;
                    note.timestamp = System.currentTimeMillis();
                    if (getIntent().hasExtra(EXTRA_NOTE)){

App.getInstance().getNoteDao().update(note);
                    } else{
```

```
App.getInstance().getNoteDao().insert(note);
                    }
                    finish();
                }
                break;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Розробка персонального мобільного Android-додатку планувальник дій мовою програмування Java та СУБД SQLite»
студентки групи 122-19-1 Бенемянович Еліни Іллівни

Керівник економічного розділу
Зав.каф. ПЕП та ПУ, д.е.н. **О.Г. Вагонова**
професор каф. ПЕП та ПУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файла	Опис
Пояснювальні документи	
Кваліфікаційна робота.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота.pdf	Пояснювальна записка до кваліфікаційної роботи. в форматі PDF
Програма	
Кваліфікаційна робота.zip	Архів. Містить коди програми і откомпільовану програму
Презентація	
Кваліфікаційна робота.ppt	Презентація кваліфікаційної роботи.