НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО

«____» _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі» спеціальності 123 «Комп'ютерна інженерія»

на тему: Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)

Виконав: студент 4 курсу, групи ІО-81

Дудка Максим Валерійович _____

Керівник: старший викладач Алещенко Олексій _____

Консультант (нормоконтроль): професор, д.т.н. _____

Сімоненко Валерій Павлович

Рецензент _____ _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ – 2022 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

«Комп'ютерні системи та мережі»

Спеціальність 123 «Комп'ютерна інженерія»

ЗАВДАННЯ

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«_____» _____ 2022 р.

на бакалаврський дипломний проєкт студента

Дудки Максима Валерійовича

1. Тема проєкту Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)
   керівник проєкту старший викладач Алещенко О.В
   Затверджені наказом по університету від «11» травня 2022 року № 1139-с
2. Термін здачі студентом закінченого проєкту (роботи)
3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*
4. Зміст розрахунково-пояснювальної записки *опис предметної області, дослідження та аналіз аналогів створеного додатку, дослідження можливих інструментів для створення проєкту, обґрунтування обраної реалізації створення даного проєкту, розробка програми, а саме реалізація обраних інструментів для її створення, створення програмного додатку, висновки.*
5. *Перелік графічного матеріалу: схема бізнес-логіки, структурна схема*.

6. Консультанти проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П.		

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2022*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2021-25.03.2022*	
4.	*Розробка структур окремих підсистем*	*25.03.2022-5.04.2022*	
5.	*Програмна реалізація системи*	*5.04.2022-15.04.2022*	
6.	*Оформлення пояснювальної записки*	*15.04.2022-20.05.2022*	
7.	*Захист програмного продукту*	*25.04.2022*	
8.	*Передзахист*	*23.05.2022*	
9.	*Захист*	*21.06.2022*	

Студентка _____ Дудка М.В.

Керівник проєкту _____ Алещенко О.В.

АНОТАЦІЯ

В цьому бакалаврському дипломному проєкті було реалізовано мобільний застосунок.

Дану програму можна використовувати для навчання написання запитів SQL та отримання практичного досвіду роботи з базами даних. Можливість зберігати інформацію, оброляти її та ділитися нею у соціальних мережах.

Застосунок написан для платформи Android на мові мові Java та Kotlin.

Ключові слова: мобільний застосунок, SQL, Java, Kotlin.

ANNOTATION

The mobile application was implemented in this bachelor's degree project.

You can use this program to learn how to write SQL queries and gain hands-on experience with databases. You have the ability to store information, process it and share it on social networks.

The application is written for the Android platform in Java and Kotlin.

Key words: mobile application, SQL, Java, Kotlin.

ОПИС АЛЬБОМУ

ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)»

Київ – 2022

справки	Форма	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
	А4		Завдання на дипломний проект	2		
	А4	***ІАЛЦ.467200.001 ОА***	Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite) Опис альбому	1		
	А4	***ІАЛЦ.467200.002 ТЗ***	Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite) Технічне завдання	3		
	А4	***ІАЛЦ.467200.003 ПЗ***	Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite) Пояснювальна записка	56		
	А4	***ІАЛЦ.467200.004 Д1***	Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite) Схема взаємодії	1		
	А4	***ІАЛЦ.467200.005 Д2***	Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite) Функціональна схема (діаграма класі)	1		
	А4	***ІАЛЦ.467200.006 Д3***	Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite) Алгоритм дій програмного забезпечення	1		
	А4	***ІАЛЦ.467200.007 Д3***	Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite) Текст програмного коду	39		

					ІАЛЦ.467200.001 ОА
		№ докум.	Підпис	Дата	
Розробив		Дудка М.В			**Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)** Опис альбому
Перевірив		Алещенко О.В.			
Н. Контр.		Сімоненко В. П.			
Затвердив					

Літ.	Аркуш	Аркушів
	1	1

НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)»

Київ – 2022

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
		№ докум.	Підпис	Дата			
Розробив		Дудка М.В.			**Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)** Технічне завдання	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Алещенко О.В.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81	
Н. Контр.		Сімоненко В. П.					
Затвердив							

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Це технічне завдання поширюється на розробку застосунку для навчання взаємодії з базою даних. Область застосування: додатковий метод отримання практичних знань з роботою з базою даних.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка застосунку для навчання взаємодії з базою даних.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література по комп’ютерних мережах і дистанційного навчання, публікації в періодичних виданнях, довідники по платформах дистанційного навчання, публікації в Інтернеті з цих питань.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту, що розробляється

- Створення бази даних за допомогою графічного інтерфейсу
- Оновлення рядку таблиці бази даних за допомогою графічного інтерфейсу
- Додавання рядку таблиці бази даних за допомогою графічного інтерфейсу
- Видалення рядку таблиці або таблицю бази даних за допомогою графічного інтерфейсу
- Використання всіх можливостей SQL, що підтримується СУБД

SQLite

- Експортування таблиці бази даних у форматі JSON та CSV
- Кастомізація зовнішнього вигляду інтерфейсу користувача

5.2 Вимоги до апаратної частини

- Мобільний пристрій на Android 6.0 та вище
- Оперативної пам’яті не менше 512 Мбайт.

6 ЕТАПИ РОЗРОБКИ

1. Вивчення літератури 21.12.2020
2. Складання та узгодження технічного завдання 04.02.2021
3. Створення мобільного застосунку 08.04.2021
4. Тестування мобільного застосунку 10.05.2021
5. Відлагодження та виправлення помилок 23.05.2021
6. Оформлення документації дипломного проєкту 25.05.2021

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)»

Київ – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

СУБД	Система управління базами даних
SQL	Structured query language, мова структурованих запитів
CSV	comma-separated value,значення, розділені комою
IDE	Integrated development environment, Інтегроване середовище розробки
JS	JavaScript
JVM	Java Virtual Machine, Віртуальна машина Java
БД	База даних
МБ	Мегабайт
API	Application Programming Interface, інтерфейс програмування застосунків
SDK	Software Development Kit
APK	Android Package
AAB	Android App Bundle
ОС	Операційна Система
ПЗ	Програмне Забезпечення

ВСТУП

В сучаному світі інформація дуже важлива валюта, її визнають у кожному куточку світу. Не просто так говорять, що хто володіє інформацією, той володіє усім світом.

Колись давно, людина відкрила полум’я, потім пригодовувала здобич, отримала необхідні поживні речовини для організму, для того що б продовжувати жити. Зробивши це, людство почало ділитися цією новиною – смажене м'ясо смачніше та поживніше за сире. Час показав, що інформація кочувала з вуст в вуста, з часом аби нічого не забути та не викривляти отримані знання – почалось занотування інформації. З розвитком писемності, інформація почала зберігатися у книгах та так і розповсюджуватися. У світі дуже багато різноманітної інформації, її потрібно зберігати, а вже тут на допомогу приходять машини – найкращі помічники людей. Ніколи не підведуть, завжди усе пам’ятають, якщо користувач правильно поставив задачу для комп’ютера. А вже тут постає питання як саме потрібно зберігати інформацію та де.

Відповідь проста – у базі даних. Що це таке? Це зібрана колекція інформації, яка зберігається дуже довго, а її керування відбувається з допомогою системи управління базами даних. СУДБ використовуються всюди від найпростіших месенджерів і до складних систем банків.

Для того аби створити і підтримувати консистентну та надійну базу даних, треба мати спеціалізовані знання з даного питання. Звичайно, що це можна зробити, прочитавши книгу і пройшовши онлайн курс. Проте, для закріплення навичок дуже важливо практикуватися – існує безліч варіантів. Ми хочемо запропонувати такий варіант для отримання практичного досвіду роботи з реляційними СУБД.

Це і є ключовим фактором для створення мобільного додатку для навчання взаємодії з СУБД.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ

1.1 Опис завдання

Завданням цієї дипломної роботи є створення мобільного застосунку для пристроїв з операційною системою Android для навчання взаємодії з базою даних, використовуючи реляційну локальну СУДБ SQLite.

Розуміння які сучасні технології можуть підійти для розробки додатку, проаналізувавши усі можливі варіанти. Обрати необхідний стек технологій для створення додатку.

Конструювання зручного та зрозумілого графічного інтерфейсу, що допоможе користувачеві отримати або покращити практичні навички з написання запитів SQL.

1.2 Опис предметного середовища

Мобільний застосунок – це програмне забезпечення, що використовується на смартфонах, планшетах і інших мобільних пристроях. Мобільні додатки можна встановити за допомогою онлайн-магазинів Google Play, App Store, App gallery. Застосунок може бути безоплатним, частково-оплатним або повністю платним [1].

Перші мобільні застосунки використовувались у якості поштових клієнтів, потім з високим попитом на цю можливість почалось створення їх для будь-якої області від додатків для замовлення їжі до ігор [21].

Розробка ПЗ для смартфонів потребує врахування їх можливостей та обмежень. Мобільні пристрої працюють використовуючи акумулятор та мають не настільки потужні процесори, аніж ПК. Треба орієнтуватися на різні розміри екану і інші технічні характеристики [22].

Для розробки додатків для мобільних пристроїв існують спеціалізовані інтегровані середовища розробки:

1.2.1 Android Studio - офіційне середовище розробки (IDE) для ОС Android від Google, побудоване на програмному забезпеченні IntelliJ IDEA від компанії JetBrains та розроблене спеціально для створення застосунків для пристроїв Android [27].

1.2.2 XCode дає розробникам уніфіковане середовище розробки для дизайну інтерфейсу користувача, програмування, тестування та відладки додатків для пристроїв Apple [23].

1.2.3 AppCode — це середовище розробки (IDE) для розробки додатків на Swift, Objective-C, C++, C та побудоване на платформі IntelliJ IDEA від компанії JetBrains [25].

1.2.4 Visual Studio Code — простий та зрозумілий редактор, підтримує JavaScript, TypeScript та має багато плагінів для інших мов програмування. Також підтримує розробку для девайсів на платформах Android і iOS [33].

Тестування мобільних додатків зазвичай відбувається на емуляторах пристроїв. Емулятори дають безкоштовний спосіб запуску та тестування мобільних застосунку у тих випадках, коли програмісти не мають доступу до певної моделі мобільного пристрою. Велике значення має графічний інтерфейс. Він має враховувати розміри екрану, інтуїтивність та інші обмеження [24].

1.3 Опис існуючих рішень

SQLite Database Editor – Android додаток для редагування SQLite баз даних в у смартфону смартфоні, зображено на рисунку 1.1.

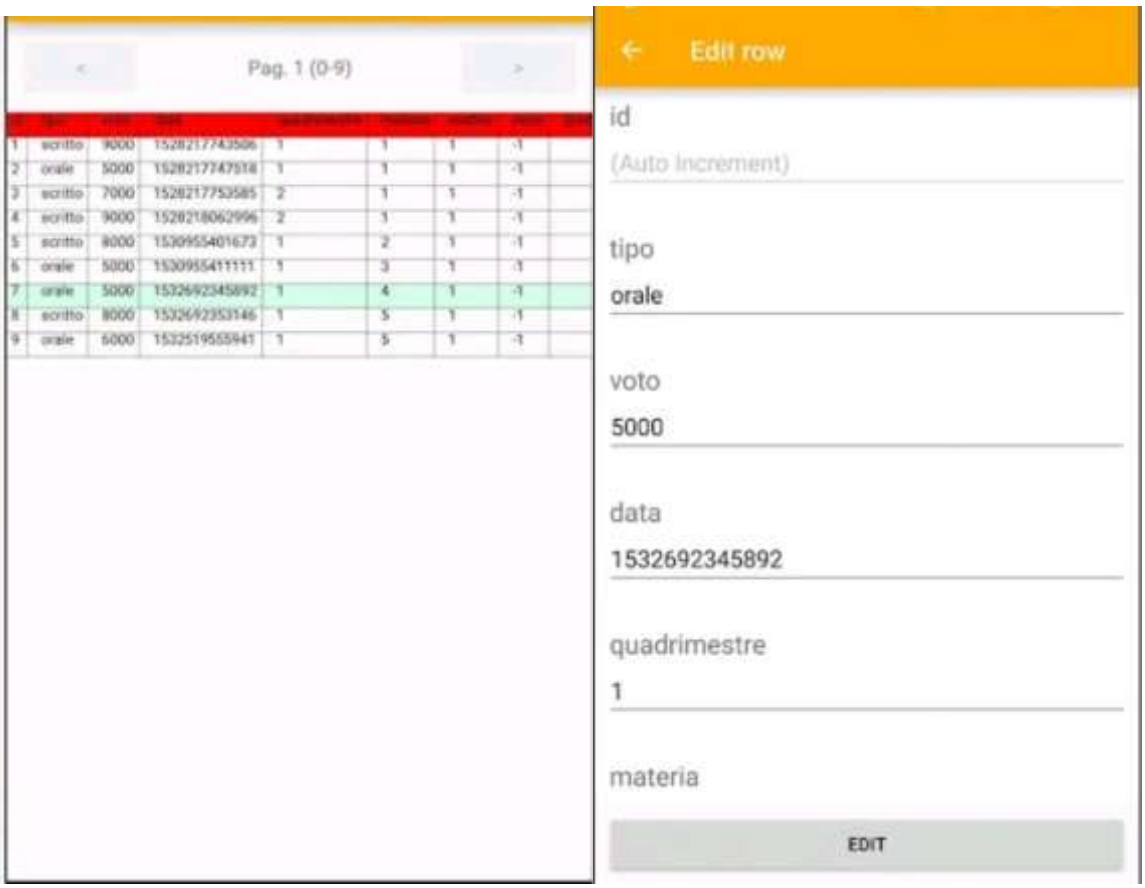


Рисунок 1.1 - SQLite Database Editor

Переваги даного веб-додатку:

- швидкість роботи
- при наявних root-правах існує можливість переглядати та редагувати бази даних інших застосунків

Серед недоліків:

- застарілий та не дуже зручний інтерфейс

PortoDB Database – Android додаток для створення баз даних, зображено на рисунку 1.2.

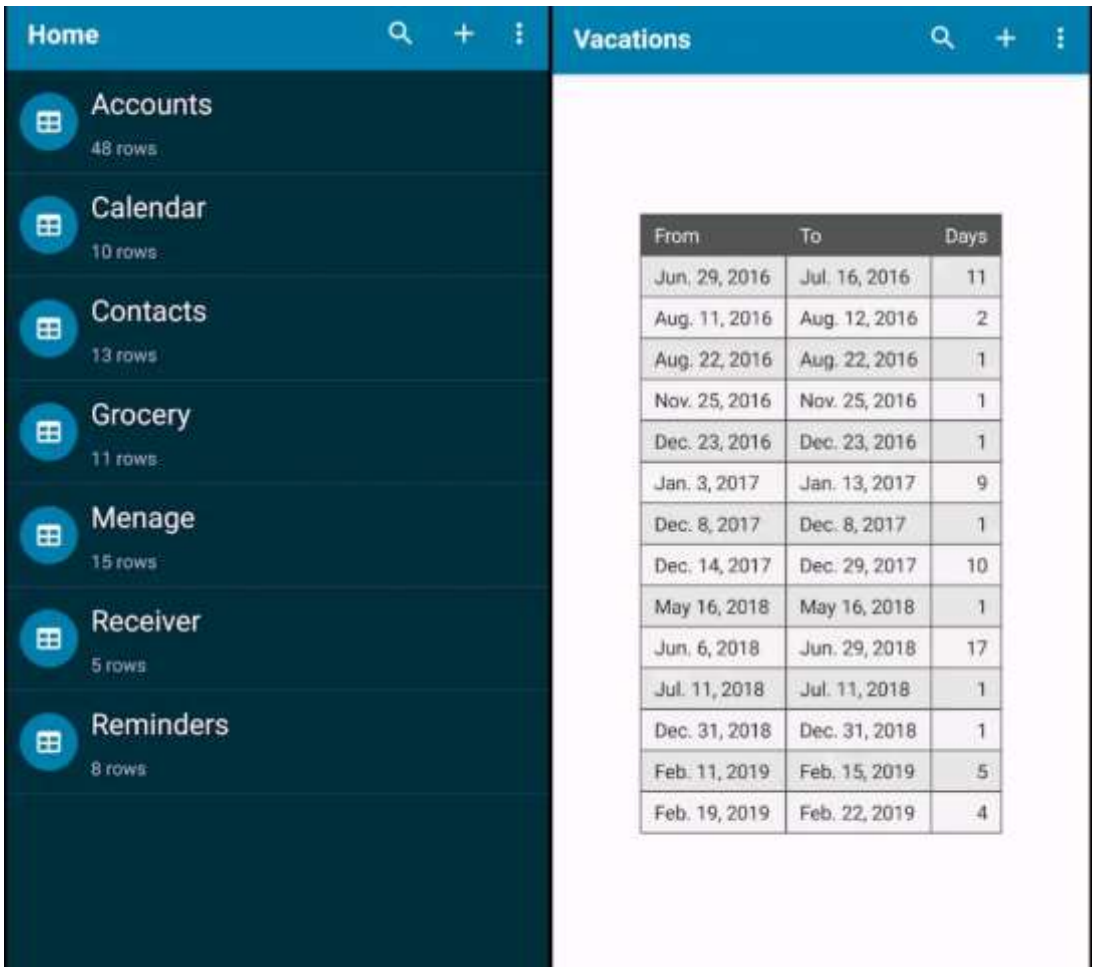


Рисунок 1.2 - PortoDB Database

Переваги даного веб-додатку:

- є можливість зберігання картинок
- є можливість зберігання в хмарі
- є можливість експорту в форматі CSV файлів

Серед недоліків:

- не дуже інтуїтивний інтерфейс
- частина функцій платна

SqlitePrime– Android застосунок для створення баз даних, зображено на рисунку 1.3.

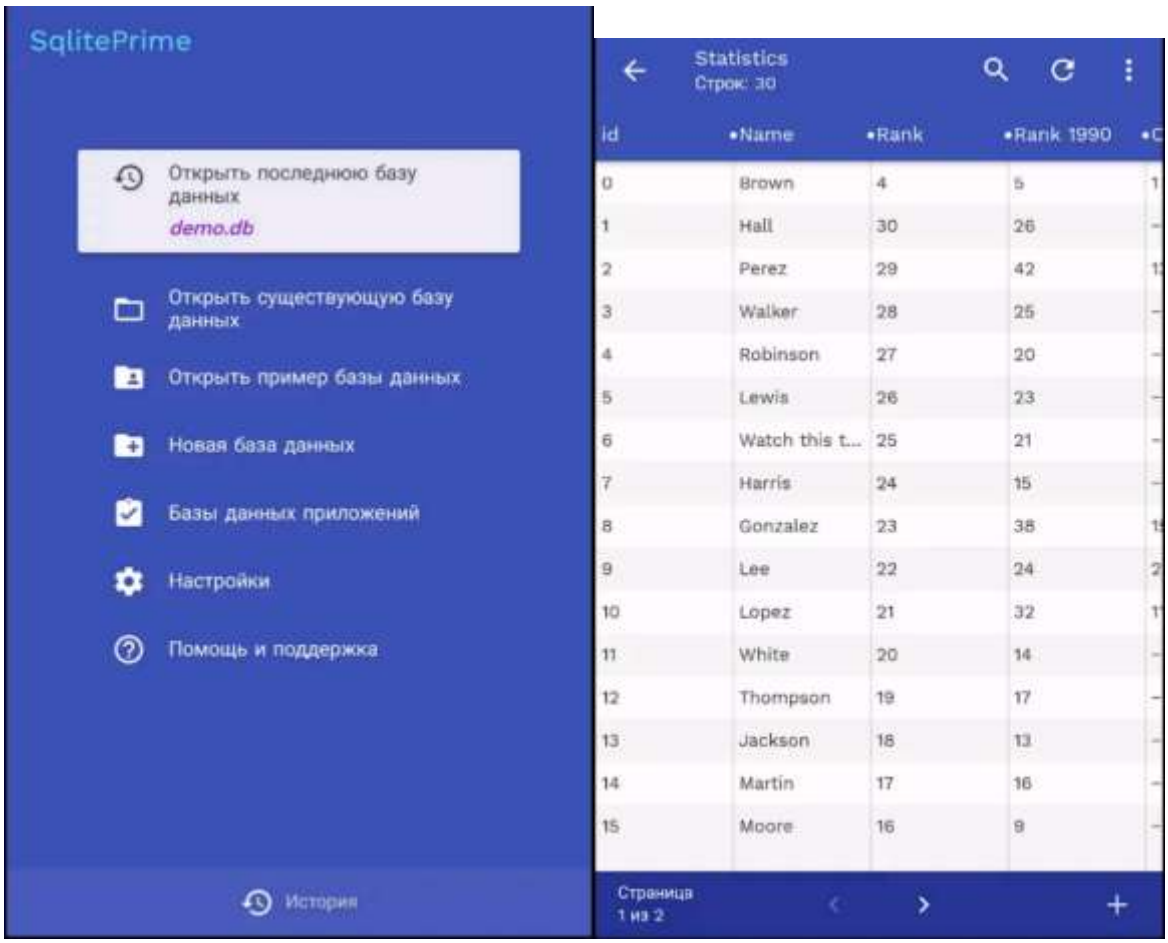


Рисунок 1.3 - SqlitePrime

Переваги даного веб-додатку:

- можливість зберігання картинок
- є можливість зберігання в хмарі
- є можливість експорту в форматі CSV файлів

Серед недоліків:

- не дуже інтуїтивний інтерфейс
- платна частина функцій

ВИСНОВОК ДО РОЗДІЛУ 1

Було проаналізовано деякі можливі варіанти інтегрованого середовища розробки та обрано Android Studio через зручні можливості для рефакторингу, створення Android проектів, збирання виконуваних файлів, можливість бачити превью інтерфейсу до запуску на емуляторі, вбудований емулятор (зручно для тестування, нема необхідності запускати на реальних девайсах з різними версіями операційної системи Android), можливість підключати плагіни на IDE компанії JetBrains.

Було проаналізовано деякі аналоги додатку для взаємодії з реляційними базами даних. Враховані деякі існуючі недоліки цих затосунків при розробці мобільного додатку.

РОЗДІЛ 2
ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Мови програмування

Треба проаналізувати можливі варіанти мов програмування для створення мобільного додатку, що забезпечують найбільш якісний перфоманс системи. Найпопулярнішими мовами вважаються Kotlin, Java, JS, Dart, Swift

2.1.1 Сучасна мова програмування, яка робить людей щасливішими, або просто Kotlin (останьою версію є 1.6.21). Це мова програмування зі статичною типізацією, яка працює поверх JVM та розробляється і підтримується компанією JetBrain [2]. Назва походить від острова, який находиться у Фінській затоці. Мова програмування була анонсована в липні 2011, а в 2012 був відкритий сирцевий код мови. На Kotlin вплинули Java, Scala, JS,Groovy, C# тому є можливість писати в функціональному стилі або в об'єктно-орієнтованому. Kotlin - мова загального призначення, вона вводить функціональні можливості сумісності Java [11]. Ідея створення Kotlin народилася з прагнення до підвищення продуктивності програмістів. Метою було зробити кращим досвід кодування практичним та ефективним способом. Перевагами використання Котлін для розробки мобільних додатків є

- Повна сумісність с Java. Тобто можна використовувати для проектів, які раніше були написані на Java не переписуючи увесь застосунок.
- Лаконічність. В даній мові програмування нема зайвих синтаксичних конструкцій або функцій, за статистикою розмір коду, який написаний на Kotlin, на 40% менше, тому допускається менша кількість помилок; Проста. За задумами автора, що перейти з будь-якої мови програмування на Kotlin можна за тиждень.
- Проста. За задумами JetBrains, перейти з будь-якої мови на Kotlin можна за тиждень. Також вважається, що цю мову програмування найлегше освоїти при першому знайомстві з програмуванням.

- Безпека. Під часу компіляції коду виконується автоматична перевірка. Null Safety (Неможливість навіть звернутися до зміної, що потенційно має значення null). Це було зроблено через те, що NullPointerException найпопулярніша помилка мови програмування Java.

Недоліками є наступні аспекти:

- Повільна швидкість компілювання. Повільніша приблизно на 30-40% ніж компіляція на мові програмування Java, це пов'язано з тим, що компіляція в віртуальній машині JVM, була все ж створена для рідної мови Java.
- Варіативність використання мови. Оскільки, розробники можуть те саме різними синтаксичними конструкціями, що приводить до неоднорідності коду.

Найвідомішими мобільними додатками, що написані на мові Kotlin є Uber (додаток для виклику таксі), Pinterest (додаток для обміну фотографіями), Evernote (додаток для ведення нотатків та поміток).

2.1.2 Написано один раз, працює всюди – чи просто мова програмування Java була розроблена Sun Microsystems, задумана Джеймсом Гослінгом і випущена в 1995 році. Останньою версією Java є Java SE 18. Ця мова програмування є високорівневою та об'єктно-орієнтованою [3]. З розвитком Java і її широкою популярністю створено безліч конфігурацій, щоб відповідати різноманітним типам платформ. Наприклад: J2EE для інтерпрайз додатків.

Перевагами є:

- Java незалежна від платформи. На відміну від багатьох мов програмування, включаючи C і C++, Java компілюється в Java байт-код. Цей байт-код розповсюджується в Інтернеті і інтерпретується віртуальною машиною (JVM) на будь-якій системі, на якій байт-код виконується.

- Проста. Java розроблена так, щоб її можна було легко вивчити. Якщо людина розуміє концепцію об’єктно-орієнтованого програмування, то Java легко освоїться [4].
- Безпечна. Java виключає можливість несанкційонованого доступу до пам’яті. Методи аутентифікації створені на шифруванні з відкритим ключем.
- Дуже надійна. Компілятор Java докладає усіх зусиль, задля того аби усунути потенціальні помилки.
- Багатопотоковість. За допомогою багатопотоковості Java можна писати складні програми, що можуть виконувати декілька завдань одночасно. Ця особливість дає можливість розробникам створювати стабільні програми, які можуть працювати безперебійно.
- Досить висока продуктивність. За допомогою Just-In-Time компіляторів Java забезпечує високий перфоманс.

Недоліками є:

- Необхідно багато оперативної пам’яті. Вона потрібна для Just-in-Time компілятора, Garbage Collector, Class loader і для реалізації багатопотокових програм [12].

Netflix (додаток для перегляду кіно), Google Maps (додаток для використання навігації в телефоні) – найпопулярніші додатки, які написані на Java.

2.1.3 Swift — мова програмування, створена з використанням сучасного підходу до продуктивності та шаблонів проектування ПЗ. Створена Крисом Латтнером і компанію Apple у 2014. Назва означає швидкість [14]. Метою мови Swift є створення найкращої мови для використання, починаючи від низькорівневого системного програмування і закінчуючи мобільними застосунками та застосунками для ПК. Актуальною є версія 5.6.1. Дизайнери Swift надихались Rust, Objective-C, Haskell, Ruby, C#, Python.

Перевагами є:

- Простота коду. Swift легко прочитати, наче це підручник з англійської.
- Підтримка. Компанія Apple більше підтримує Swift, аніж Objective-C, отже усі проекти можуть переходити на цю мову програмування під час розробки.
- Швидкість. Swift приблизно на 40% швидше за Objective-C.

Недоліками є:

- Відсутня підтримка старих версій IOS;
- Майже відсутня підтримка сторонніх інтегрованих середовищ розробки.

Майже всі сучасні додатки для платформи iOS, написані на мові Swift.

2.1.4 «JS має фундаментальні проблеми, що неможливо виправити, тому був створений Dart» — розробник данної мови програмування Марк Мілллер. Ця мова належить компанії Google та з'явилась в 2011 році. Актуальна версія 2.17.0. Dart об'єктно-орієнтована мова програмування з синтаксисом у стилі C, яку можна за бажанням перекомпілювати в JavaScript або нативно під Android чи IOS. Назва походить від влучити в ціль. Dart підтримує різноманітні можлвості програмування, такі як класи, інтерфейси, дженерики, колекції. Dart широко використовується для створення single-page applications (SPA). SPA застосовуються лише до веб-програм [29]. SPA дозволяють переміщатися між різними екранами веб-програми без завантаження іншої веб-сторінки в браузер. Класичним прикладом є GMail ─ якщо ви натискаєте на повідомлення в своїй папці «Вхідні», браузер залишається на тій самій веб-сторінці, проте код JavaScript приховає «Вхідні» та виведе тіло вхідного повідомлення на екран. Перевагами є:

- Кросплатформеність. Існує фреймворк Flutter, за допомогою якого можна розробляти нативні застосунки під Android, IOS, MacOS, Windows, Linux, а такаж веб-застосунки.
- Сучасний підхід у створенні користувацьких інтерфейсів (віджети)

Необхідно значити негативні сторони:

- Динамічна типізаця. Через неї зменшується підтримуваність коду, збільшується кількість помилок в рантаймі, що не відловив компілятор..

Прикладами використання цієї мови є додатком Ebay (інтрнет-магазин).

2.1.5 Як Дуглас Крокфорд вважає: «JavaScript - найбільш неправильна мова програмування у світі». Це мова з динамічною типізацією. Вона легка у застосуванні і найчастіше використовується у якості частинм веб-сторінки, допомагає реалізувати динамічні сторінки. JS - інтерпретована мова програмування з можливістю використовувати об'єктно-орієнтоване програмування [30]. Спочатку JS був відомий як LiveScript, проте Netscape змінила назву на JavaScript, можливо, через популярність, викликане Java. Основне ядро мови вбудовано в Internet Explorer, Netscape, Chrome, Firefox, Vivaldi та інші браузери.

Перевагами є:

- Підтримка всіма сучасними браузерами.
- Angular і React Native підтримують кроссплатформені мобільні додатки..

Недоліками є:

- Жахлива динамічна типізація (є можливість підключити TypeScript де з цим ситуація краще). Навіть досвідчений розробник може не помітити помилку, наприклад, в функцію може буде переданий параметр неправильного типу, при цьому програма запуститься і відпрацює некоректно.
- Відсутня швидкість. Через віртуальну машину, що інтерпретує код – JS вважається повільної мовою програмування.
- Неточність. В js числові значення зберігаються неточно, наприклад 1.0 + 2.0 = 3.0000004, тому при розрахунку складних математичних функції математика може буде помилка розрахунків.

2.2 Бази даних

Треба проаналізувати можливі варіанти баз даних для використання у розробці мобільного додатку. Будемо розглядати такі бази: SQLite, H2, MySQl, PostgreSQL, MariaDB.

2.2.1 SQLite - написана на C бібліотека бази даних, яка реалізує механізм взаємодії з базою даних SQL з нульовою конфігурацією, тобто її не треба налаштовувати так як і інші СУБД. SQLite не є окремою програмою, як інші бази даних. SQLite має прямий доступ файлової системи, що б забезпечити зберігання інформації [10]. Уся база даних SQLite зберігається на пристрої у одному файлі, що є платформо-незалежним. SQLite важить небагато, менше аніж 400 КБ повністю налаштована база або менше 250 КБ без додаткових функцій. SQLite - автономна база даних, що означає повну відсутність зовнішніх залежностей. Архітектуру даної БД можна на рисунку 2.1.

Транзакції SQLite сумісні з ACID, та забезпечують безпечний доступ з декількох потоків або процесів [6]. SQLite підтримує більшість функцій SQL, що є в стандарті SQL92 (SQL2). SQLite написана на C і надає легкий у використанні API. SQLite доступна у UNIX (Android, Linux, Mac) і Windows (Win32, WinRT, WinCE) [17].

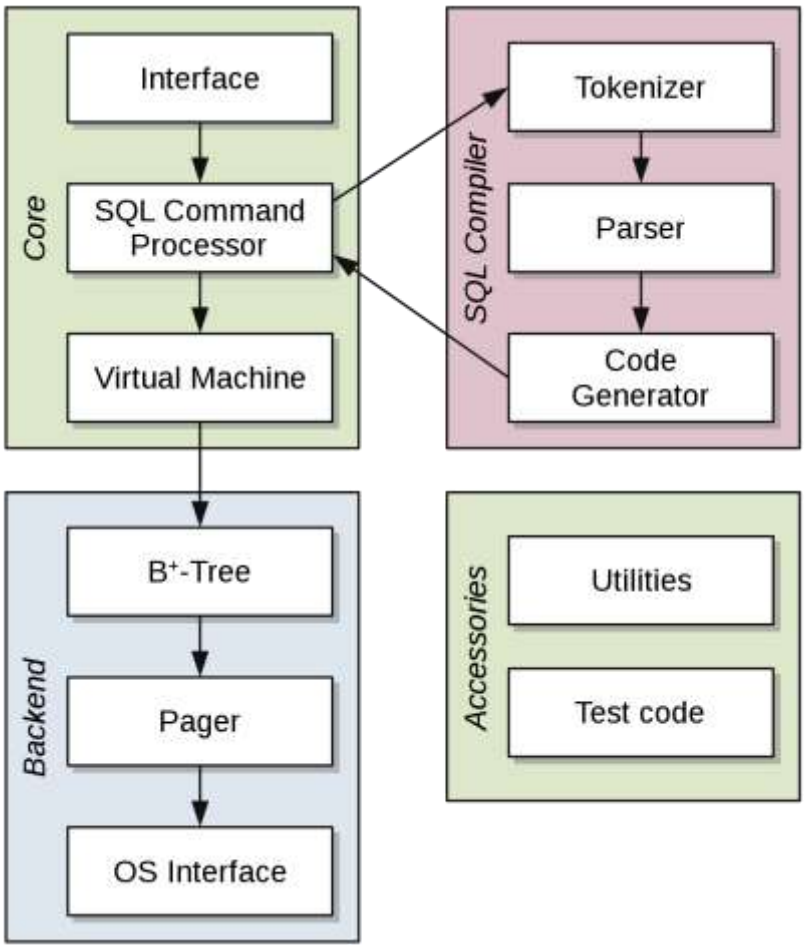


Рисунок 2.1 - Архітектура SQLite

2.2.2 H2 — база даних написана на Java з відкритим сирцевим кодом. Її можна вбудовувати в програми написані на Java або навіть запускати у клієнт-серверному режимі. В основному H2 нлаштовують на роботу як базу даних inmemory. Це означає, що дані не зберігатимуться на жорсткому диску, а будуть в оперативній пам'яті. У вбудованому режимі H2 не використовується для розробки у production середовищі, в основному використовується для тестування програм. Надзвичайно швидка, працювати з нею можна за допомогою JDBC API [7]. Невеликий розмір - приблизно 1,5 МБ розмір файлу jar з залежностями. Архітектура БД зображена на рисунку 2.2.

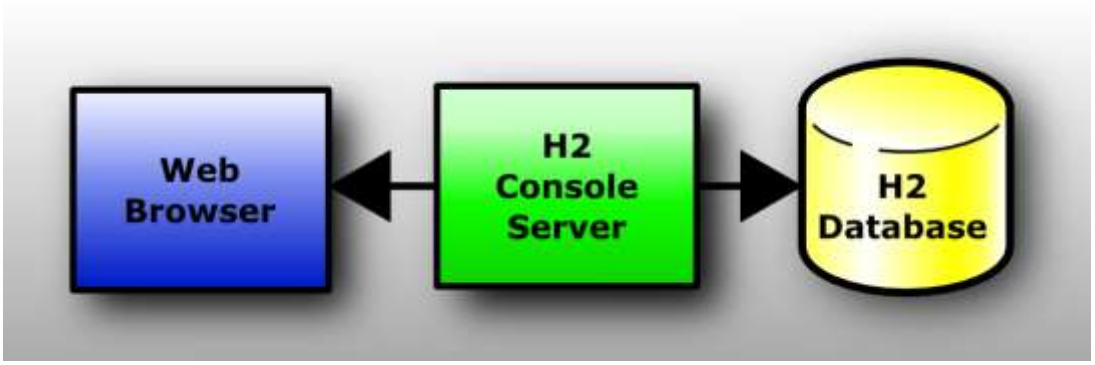


Рисунок 2.2 - Архітектура H2

2.2.3 PostgreSQL - дуже потужна об'єктно-реляційна відкрито-вихідна система баз даних. Вона активно розробляється більш ніж 15 років та має перевірену часом архітектуру, що заслужила міцну репутацію у розробників за надійність, коректність та цілісність даних [5]. PostgreSQL працює на усіх розповсюджених операційних системах, таких як Linux, UNIX-like (HP-UX, AIX, BSD, SGI IRIX, Solaris, Mac OS X, Tru64) та Windows. Вона підтримує зберігання тексту, зображенньь, звуків і відео, також включає API для C/C++, Ruby, Java, Perl, Python і Open Database Connectivity (ODBC) [20]. PostgreSQL підтримує більшу частину стандарту SQL та пропонує багато новітніх функцій і типів даних (наприклад дерева). На рисунку 2.3 показано архітектуру БД.

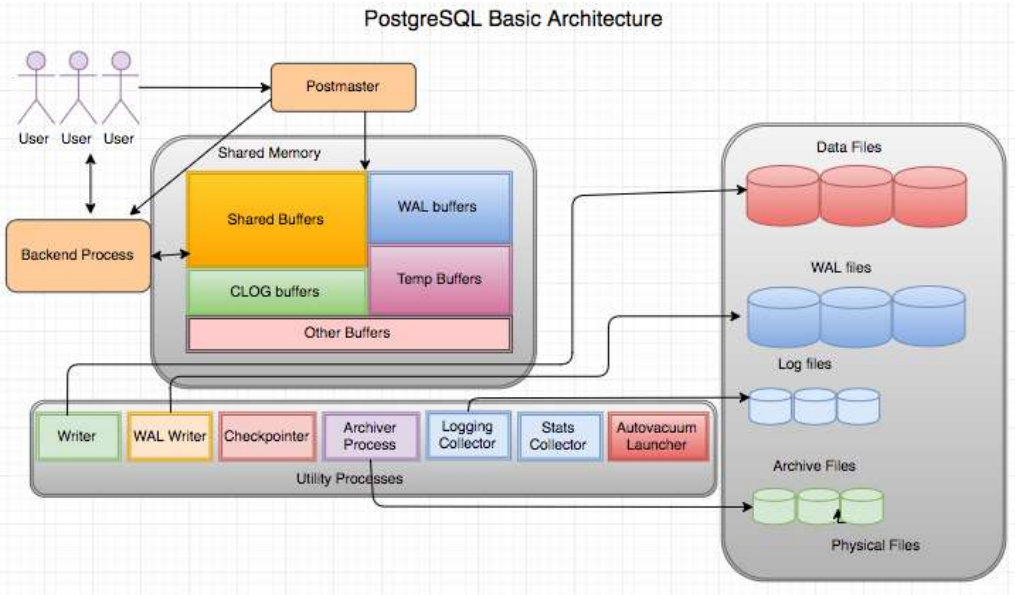


Рисунок 2.3 - Архітектура PostgreSQL

2.2.4 MySQL - швидка та проста у використанні СУБД з відкритим сирцевим кодом, що використовується для багатьох малих та великих проєктів. MySQL розробляється, підтримується та продається шведською компанією MySQL AB. MySQL являється дуже потужною програмою. Вона має велику кількість функціональних можливостей найдорожчих і найпотужніших баз даних. MySQL використовує звичайну форму мови запитів SQL [8]. MySQL працює на всіх розповсюджених операційних системах та з багатьма мовами програмування, включаючи PHP, JAVA, PERL, C, C++, тощо. MySQL працює великі набори даних. MySQL підтримує великі за розміром бази даних, більше 50 мільйонів рядків у таблиці. Існують обмеження за розмірами файлу і за замовчуванням становить 4 ГБ для таблиці, але є можливість збільшити його (якщо потужність комп'ютера та можливості операційної системи дозволяють) до гіпотетичного обмеження у 8 мільйонів терабайт. Ліцензія GPL з відкритим вихідним кодом дозволяє розробникам модифікувати програмне забезпечення бази даних відповідно до свого власного програмного продукту. На рисунку 2.4 представлена архітектура БД.

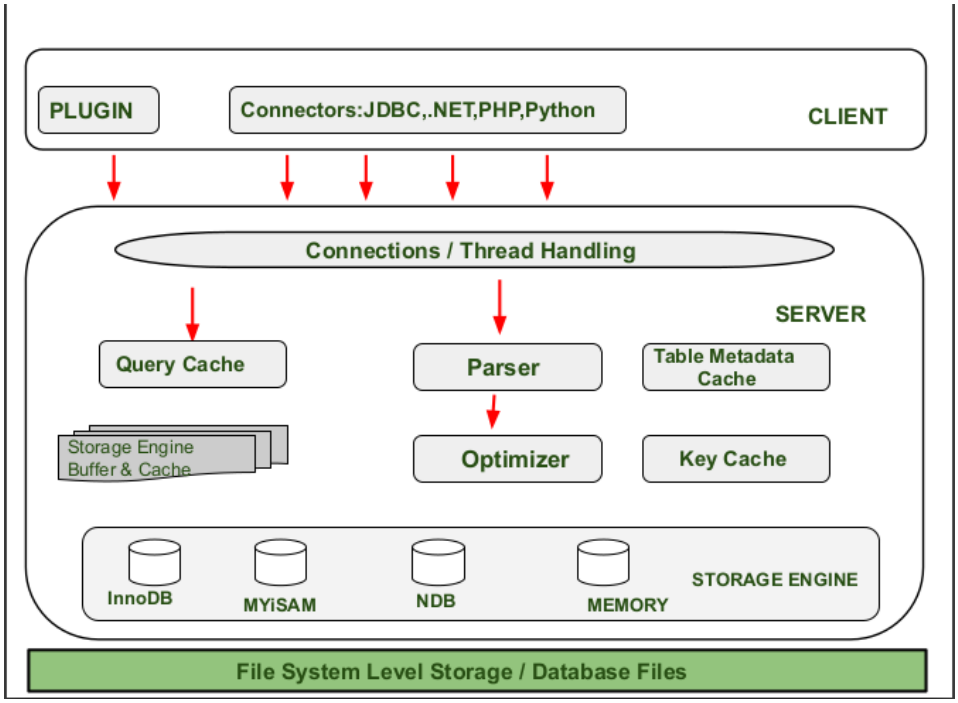


Рисунок 2.4 - Архітектура MySQL

2.2.5 MariaDB - популярне відчіплення MySQL, створене програмістами, що будували MySQL. Так сталося через занепокоєння, зв'язані з придбанням MySQL компанією Oracle. MariaDB пропонує підтримку як для невеликих обробок даних, так і величезних масивів даних для потреб підприємства. MariaDB має ціль стати повною заміною MySQL, що вимагає лише звичайного видалення MySQL з комп'ютера і встановлення MariaDB. Вся база даних MariaDB знаходиться під ліцензіями GPL, LGPL або BSD [9]. MariaDB має великий вибір механізмів для зберігання та обробки даних, включаючи високопродуктивні можливості зберігання великих масивів даних, навіть є механізми для роботи з іншими СУБД. MariaDB працює на усіх розповсюджених операційних системах та підтримує широкий набір мов програмування. Окрім того, MariaDB пропонує велику кількість операцій та команд, недоступних у MySQL, та усуває або замінює функції, що негативно можуть вплинути на продуктивність [18]. Нижче на рисунку 2.5 зображено архітектуру MariaDB.

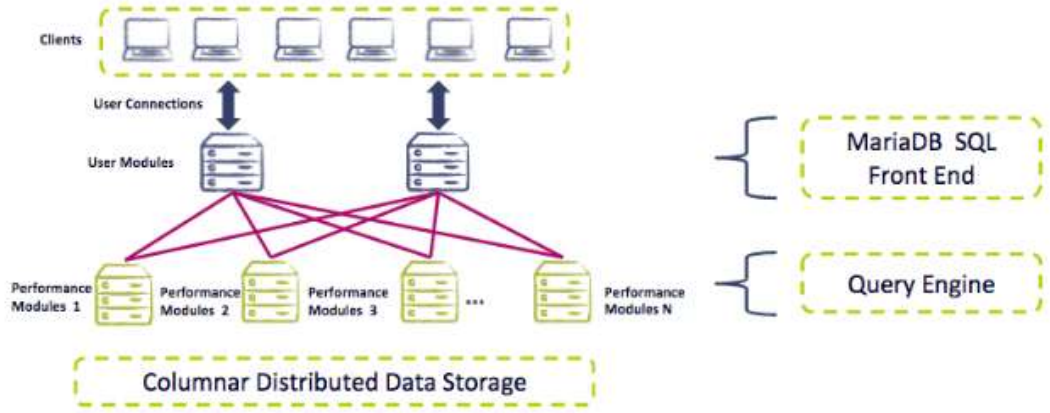


Рисунок 2.5 - Архітектура MariaDB

2.3 Зовнішній вигляд системи

Під час розробки мобільного застосунку важливим є врахування зовнішнього вигляду. Дизайн - це важливий компонент у побудові графічного інтерфейсу для взаємодії з користувачем. У світі є багато прикладів гарних ідей що не реалізувались, бо клієнту просто не було зрозуміло як управляти

програмним забезпеченням. Будуючи додаток, важливо звертати увагу на наступні пункти:

- Візуально приємний графічний інтерфейс не повинен використовувати більше аніж 3 розміри, щоб користувачі легко могли ідентифікувати елементи у графічному інтерфейсі [28].
- Макет з гарно проробленою візуальною ієрархією елементів буде легко зрозумілий користувачам. Елементи у графічному інтерфейсі мають бути вирівняні так, щоб вони були впорядковані в залежності від важливості. Такий принцип контролює якість мобільного застосунку. Якщо користувачу важко зрозуміти, де шукати певний елемент, швидше за все, макет додатку не має чіткої візуальної ієрархії елементів.
- Дизайн має бути збалансованим, щоб області графічного інтерфейсу були достатньо чіткими для спостерігача. Використання відстаней між елементами може відігравати дуже важливу роль в балансуванні ваших графічних компонентів мобільного застосунку.
- Правильне використання контрасту у дизайні може показати користувачеві помітну різницю між елементами у інтерфейсі додатку. Принцип контрасту можна використовувати, застосовуючи правильні поєднання кольорів. Це можна побачити, у найкращих графічних оболонках Android та в операційній системі iOS [31].
- Більшість користувачів мобільних застосунків сьогодні мають власну ментальну модель того, яким має бути типовий графічний інтерфейс користувача. Таким чином, юзер очікує певної узгодженості серед застосунків, які він використовує. Чим більше користувачу знайомий інтерфейс продукту, тим легше його освоїти та тим кращим буде досвід використання. Це трохи полегшує життя для дизайнерів, адже їм не треба винаходити велосипед. Будь-який дизайнер мобільних застосунків має

враховувати все це, коли мова йде про дизайн графічного інтерфейсу користувача [32].

Для того щоб, юзеру дати можливість налаштовувати застосунок під себе, можна дати можливість змінювати кольори елементів інтерфейсу або їх розміщення на екрані. Аби вирішити це питання розробники часто використовують бібліотеки.

2.3.1 Cynanea. Потужний, динамічний та веселий механізм зміни тем [15]. Названий на честь океанічного восьминога Cyanea, що вміє маскуватися та не тільки має можливість часто змінювати колір, а й також може змінювати текстуру і візерунки своєї шкіри.

2.3.2 Theme - експериментальна бібліотека тем для Android. Ця бібліотека натхненна красою Cyanea [16]. Theme використовує material-components-android, тому вона вимагає від розробників використовувати material-components-android залежність у проекті.

Недавно вийшло нове бачення створення графічного інтерфейсу від Google для Android - Jetpack Compose. Теми в Jetpack Compose компонуються з ряду низкорівневих конструкцій. Їх можна побачити у сирцевому коді MaterialTheme, а також їх можна застосувати у додатках [26]. Тема зазвичай складається з ряду компонентів, які групують загальні графічні і поведінкові концепції. Ці компоненти можна моделювати з допомогою класів.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було розглянуто основні мови програмування для розробки мобільного додатку. Аналізуючи варіанти, можна дійти до логічного висновку, що Java та Kotlin є найкращими варіантами, оскільки саме ці мови програмування є найбільш вживаними в нативній розробці під пристрої на операційній системі Android.

Було проаналізовано різноманітні варіанти реляційних СУБД. Було обрано SQLite, адже данна бібліотека є швидкою, проте найголовнішим перевагою вважається її вбудованість у еко-систему Android без необхідності у використанні додаткових зовнішніх залежностей.

Окрім того було обрано бібліотеку для кастомізації зовнішнього вигляду додатку- Cynanea. Її головна перевага у тому, що можна змінювати кольори та теми без перезавантаження застосунку. Це є дуже зручним при використанні додатку

РОЗДІЛ 3
СТВОРЕННЯ ПРОЄКТУ В ANDROID STUDIO

Для створення проєкту в Android Studio після відкриття натискаємо:

1 File

2 New

3 New project

Відкриється наступне вікно:

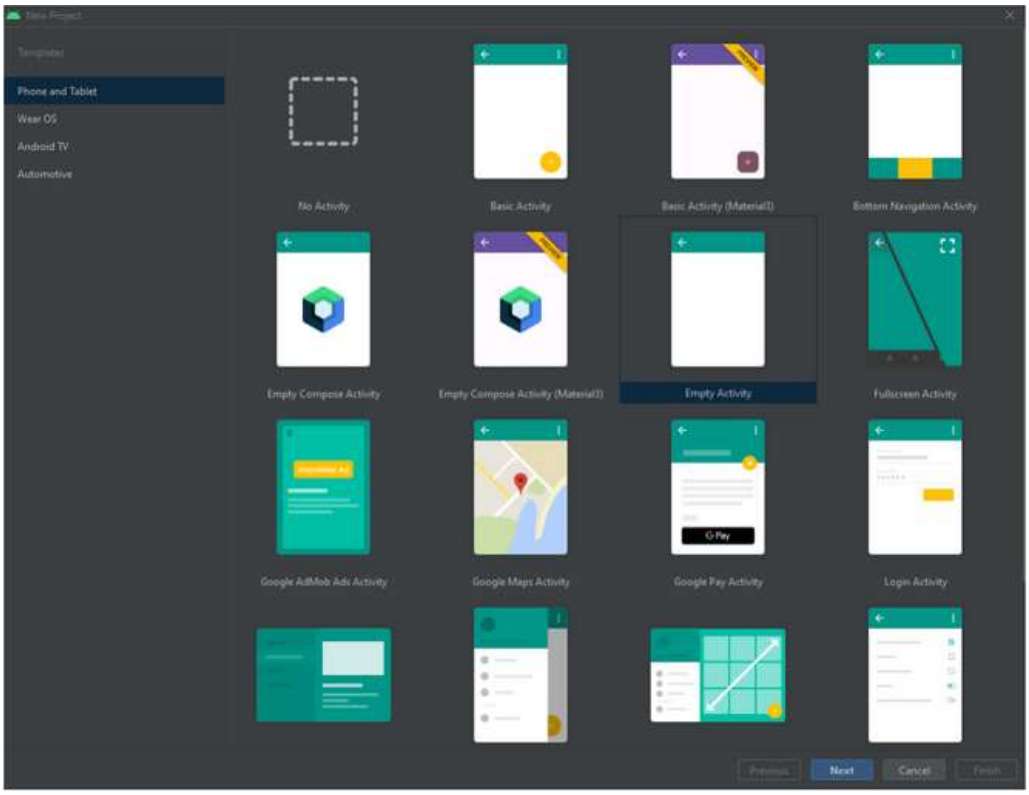


Рисунок 3.1 - Віно вибору дефолної Activity

У ньому можна вибрати під які пристрої буде створений проєкт (смартфони та планшети, годинники з Wear OS, телевізори з Android TV та інше). Також в цьому вікні треба вибрати дефолтну Activity або не створювати її.

Далі натискаємо Next і потрапляємо в наступний екран:

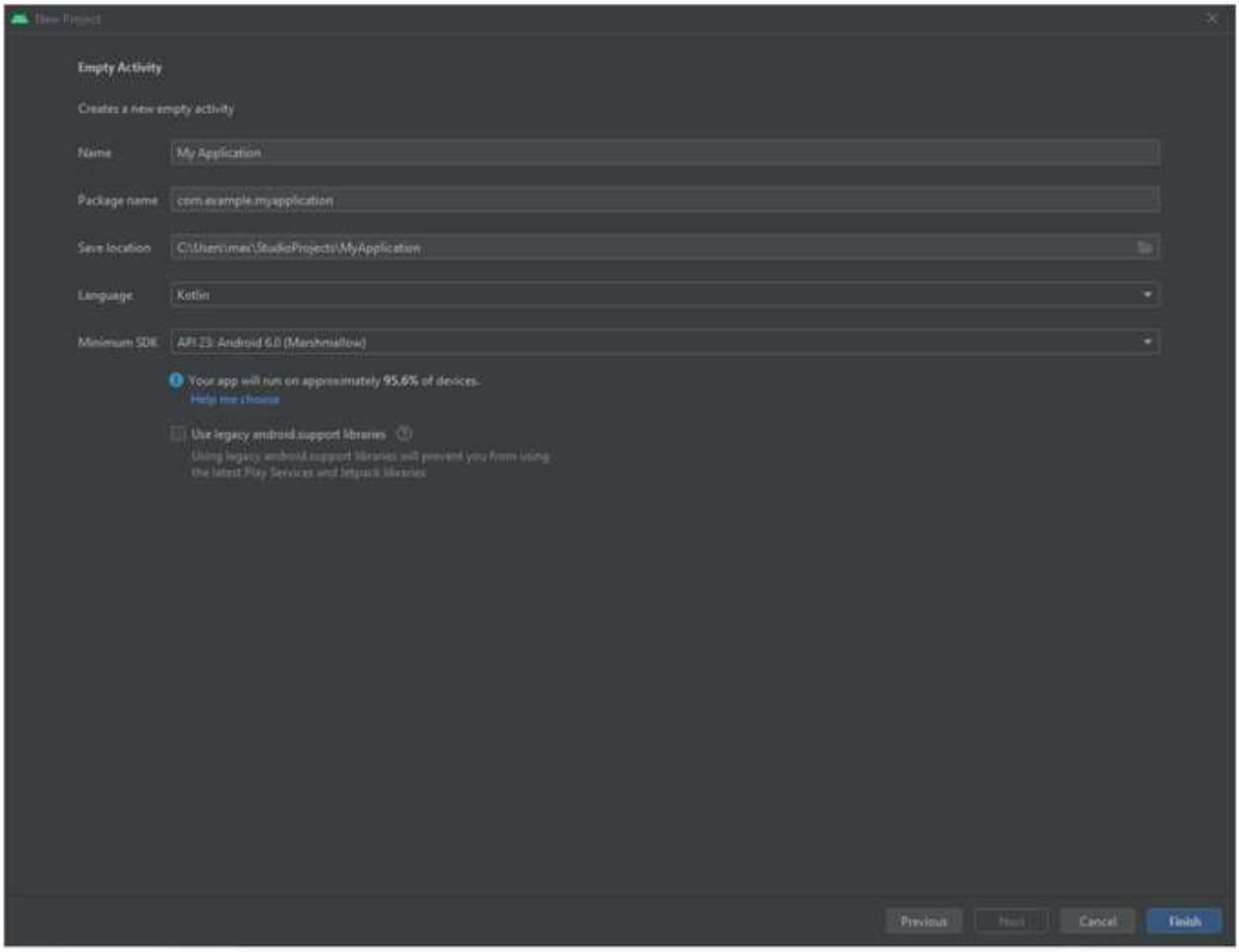


Рисунок 3.2 - Вікно створення проєкту

На цьому екрані можна вибрати назву проєкту, назву pacages, основну мову програмування, а також мінімальну версію API (тобто мінімальну версію Android). Також, можна побачити який відсоток пристроїв буде підтримувати додаток (наприклад при мінімальній версії Android 6 додаток буде підтримуватись на 95.6% смартфонів та планшетів).

Ось повна розкладка залежності мінімальної версії Android та відсотка пристроїв що зможуть підтримувати такий додаток (статистика за травень 2022 року):

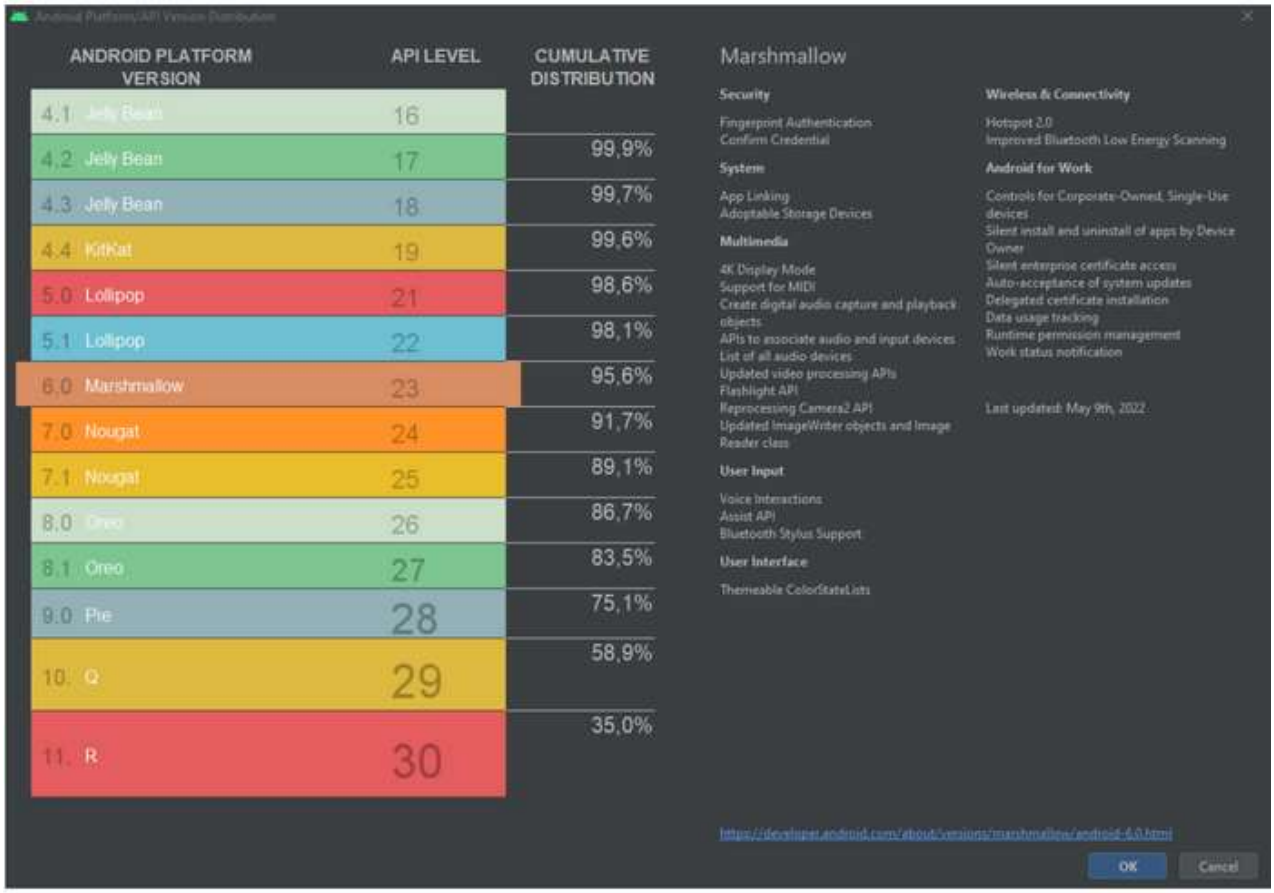


Рисунок 3.3 - Розкладка версій Android

Після цього, буде створений новий проєкт і на екрані буде щось схоже на це:

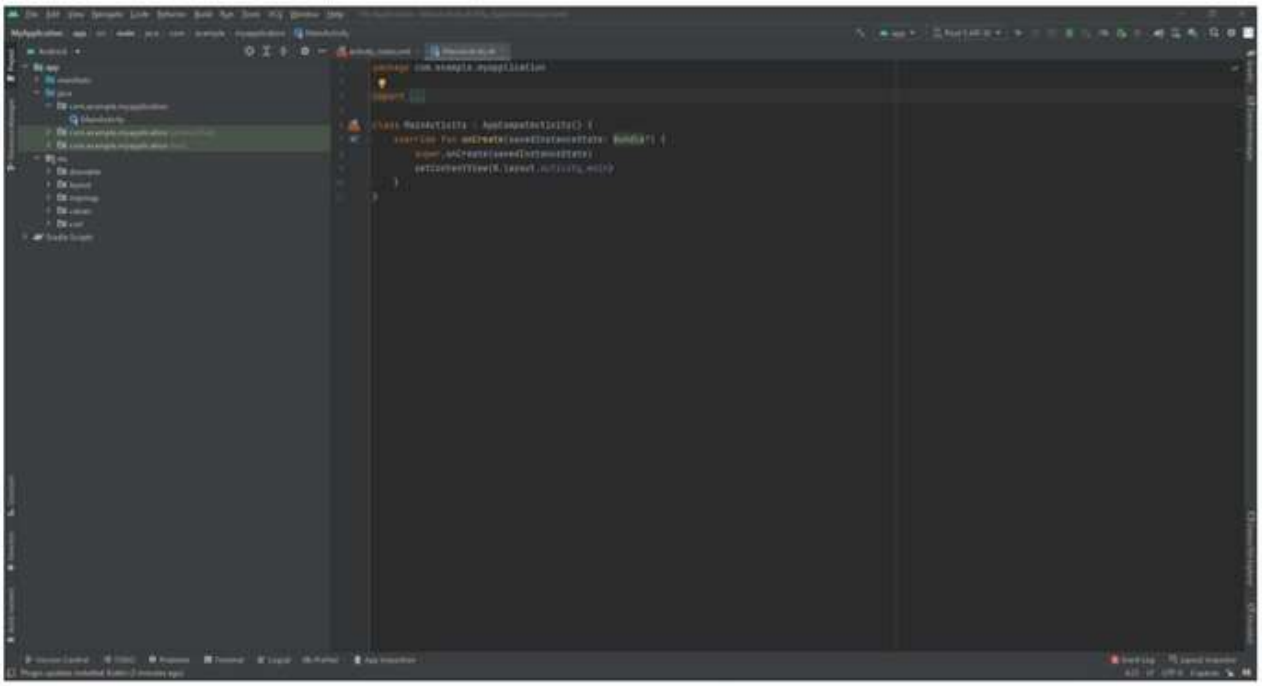


Рисунок 3.4 - Android Studio

Основні додаткові віконця в Andorid Studio:

Безпосередньо вікно вводу тексту. Саме тут можна писати, редагувати, копіювати сирцевий код програми.

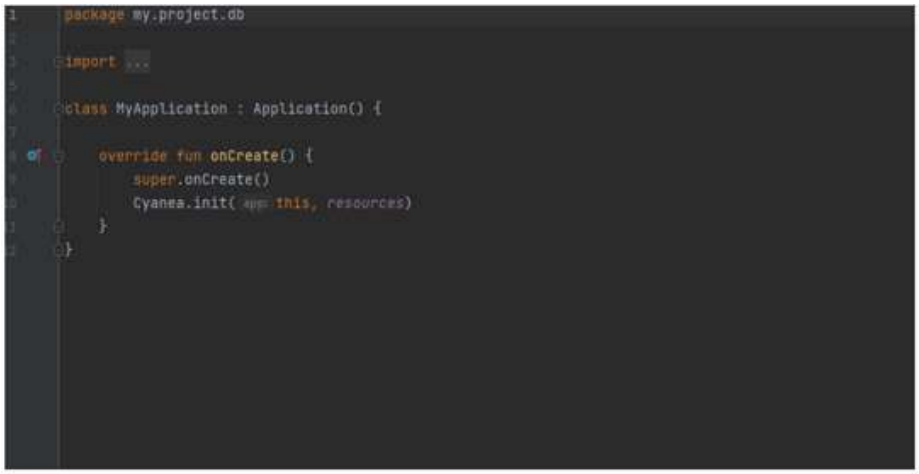


Рисунок 3.5 - Вікно вводу тексту

Файлове вікно. В ньому можна побачити всю структуру проєкту, всі файли програмного коду та розмітки, а також конфігураційні файли (маніфест тощо).

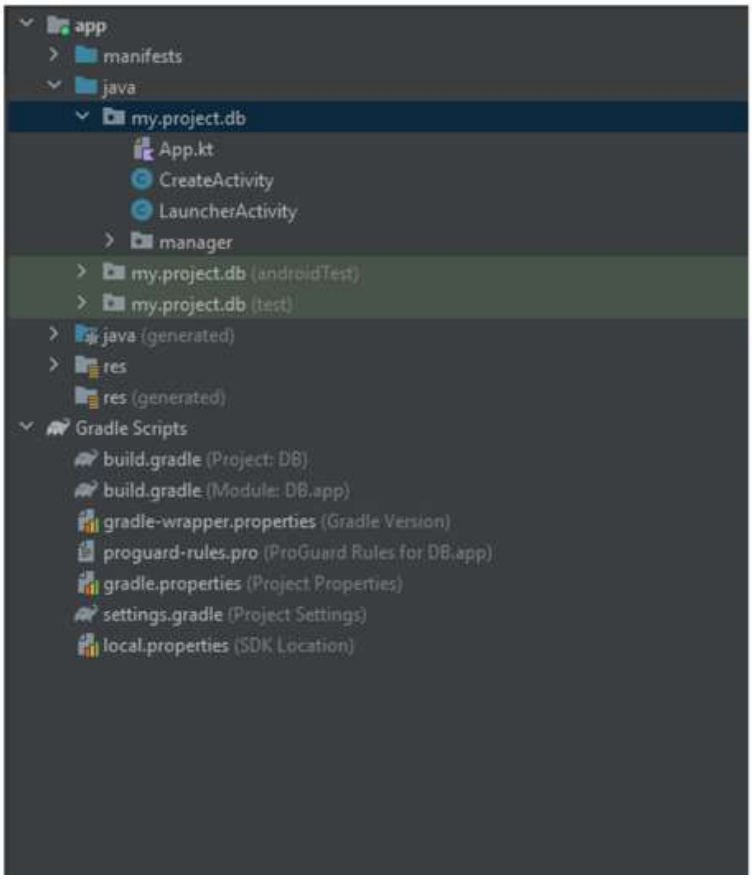


Рисунок 3.6 - Файлове вікно

Вікно для створення емуляторів пристроїв. В ньому можна створювати, змінювати, видаляти, запускати емулятори.

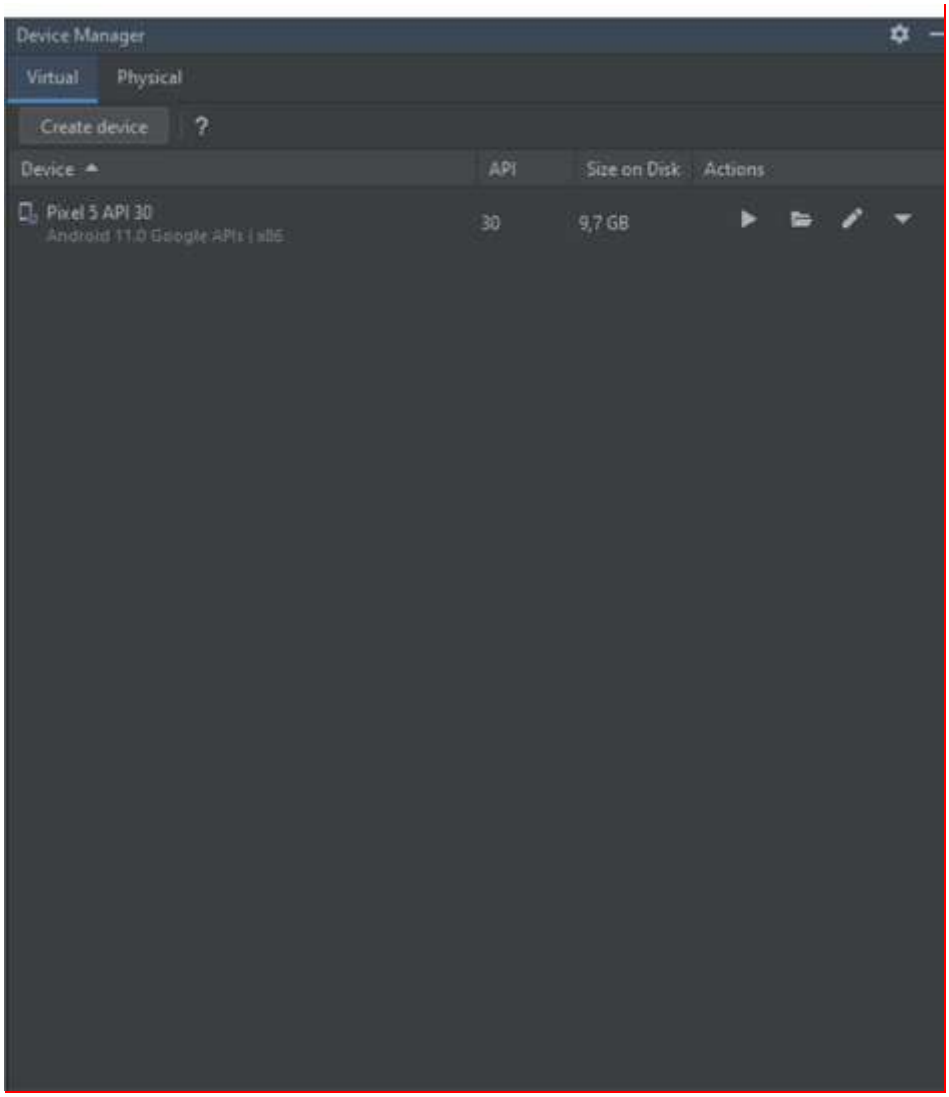


Рисунок 3.7 - Вікно для створення емуляторів

Ресурсний менеджер. Він показує усі додаткові ресурси що використовує додаток (картинки, гіфки, анімації, тощо).

Рисунок 3.8 - Ресурсний менеджер

Термінал. Для Windows це буде повноцінний PowerShell, для Unix систем це буде їх термінал.

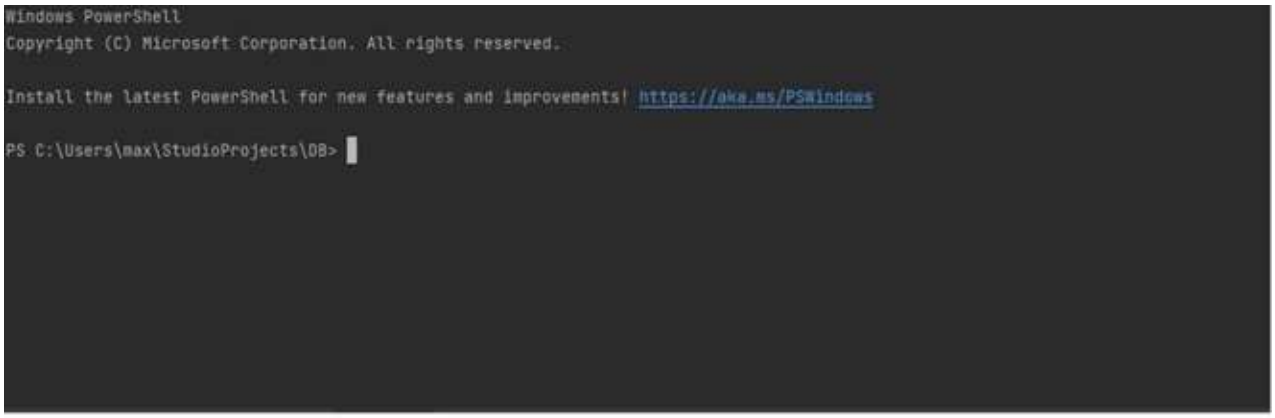


Рисунок 3.9 - Термінал

Профайлер. В ньому можна побачити в real-time режимі навантаженяя на процесор, кількість пам’яті що використовує застосунок, передачу даних та інше.

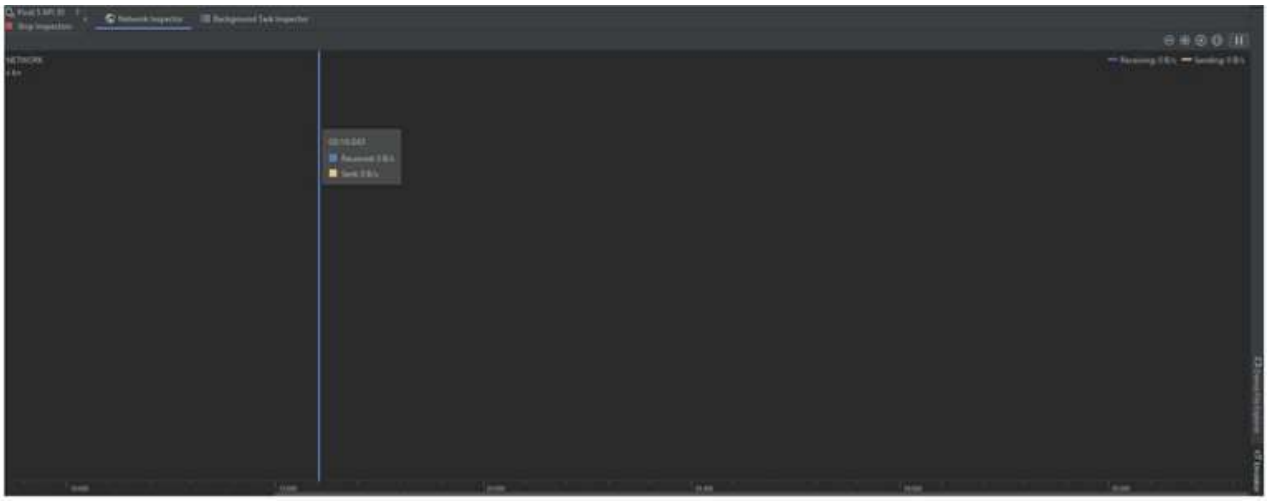


Рисунок 3.10 - Профайлер

Logcat. Сюди пишуться усі логи щодо роботи додатку. Також тут можна побачити що пішло не так у випадку крашів застосунку.

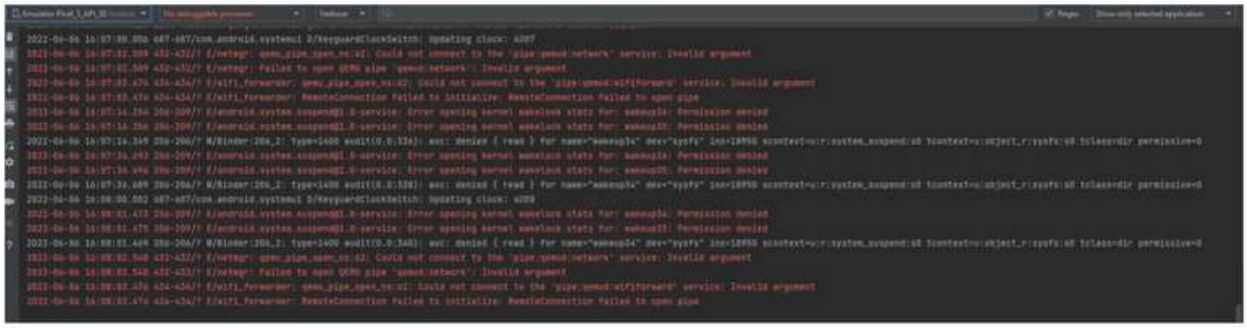


Рисунок 3.11 - Logcat

3.1 Огляд створення емуляторів в Android Studio

Інтегроване середовище розробки Android Studio має чудові можливості для створення емуляторі пристроїв. Для того що б створити новий емулятор треба перейти в Device Maneger та натиснути кнопку Create Device, після чого відкриється наступне вікно:

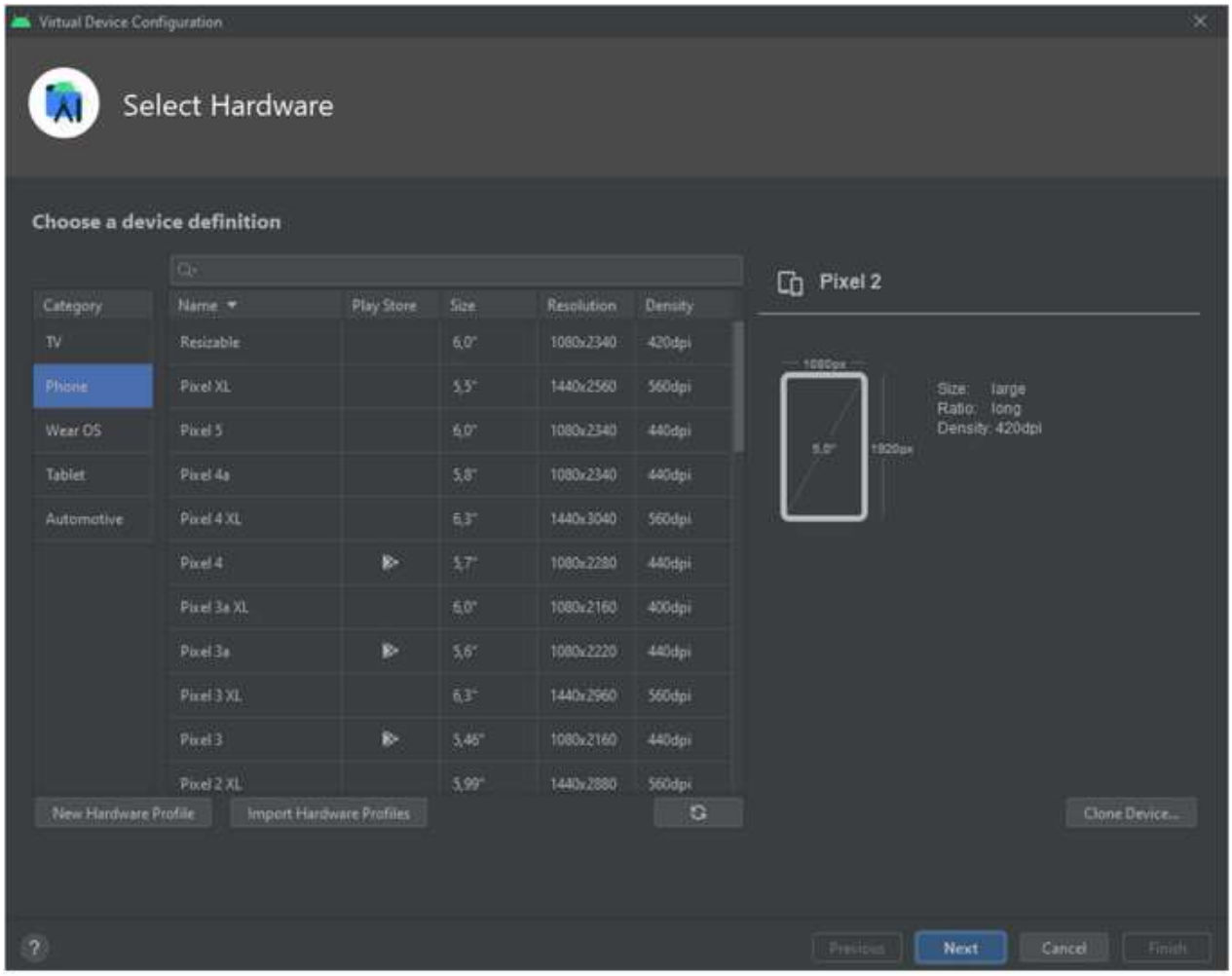


Рисунок 3.12 - Вікно вибора пристрою

У ньому можна вибрати тип девайсу, модель, його розміри, відношення сторін та навіть щільність пікселів на дюйм. Також є можливість склонувати все готовий емулятор. Якщо натиснути Next, то можна побачити наступне вікно:

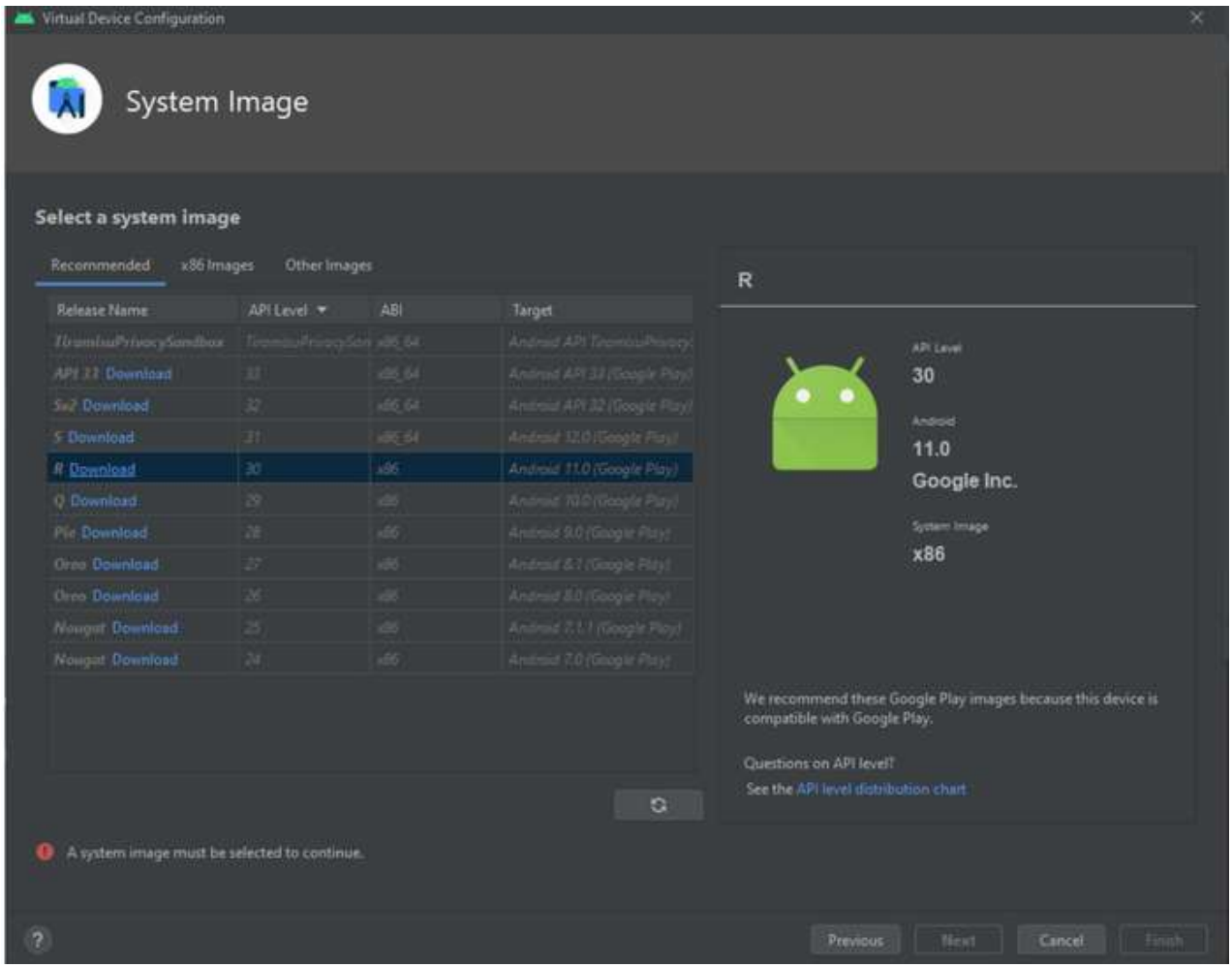


Рисунок 3.13 - Вікно виба версії ОС

В даному вікні можна вибрати версію операційної системи Android емулятора. Навіть існують образи для комп'ютерів з процесорами ARM. При першому виборі Android образ треба завантажити, але в наступні рази завантажувати нічого не треба і створення емулятора буде проходити швидше. Ось приклад вікна встановлення Android 11:

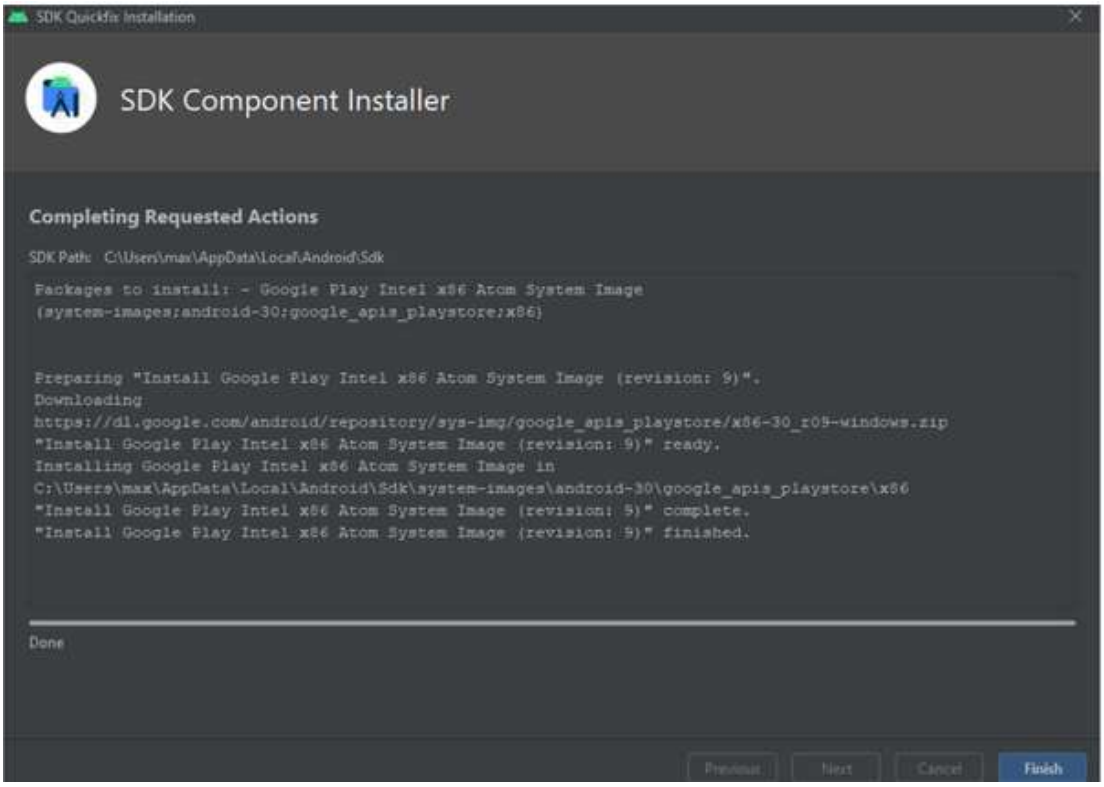


Рисунок 3.14 - Вікно встановлення Android

Після завантаження образу потрапляємо в меню додаткових налаштувань емулятора девайсу:

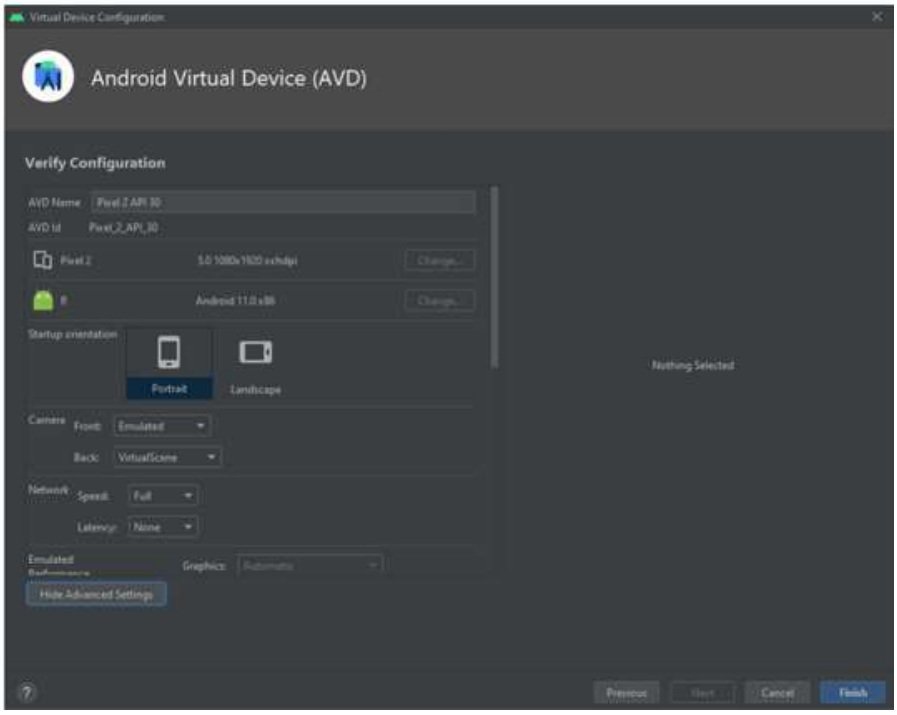


Рисунок 3.15 - Меню додаткових налаштувань

В цьому вікні можна вибрати дефолтну оріентацію пристрою, що саме буде показувати камера при включені, чи треба додати затримку в швидкості інтернету, кількість оперативної пам’яті виділеної під застосунок, а також чи буде вмикатись ситемна клавіатура. Після цих налаштувань закінчується створення емулятора.

3.2 Огляд найкорисніших shortcuts в Android Studio

Control+Alt+S – відкриває діалогове вікно за налаштуваннями.

Shift+Shift - пошук по всьому проекту.

Control+F - пошук по файлу.

Control+R - заміна символів у файлі.

Control+N - пошук по назві класу.

Control+Shift+N - пошук саме по назві файлу, а не класу.

Control+Shift+F - пошук в сирцевому коді програми.

Alt+Insert - генерацыя коду (геттери, сеттери, конструктори, методи equals(), hashCode(), toString()

Control+Shift+Enter – закінчує стейтмент.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було описано створення Android проєкту в Android Studio. Також було показано основні можливості IDE, що вбудовані в ній. Наведено приклад створення емулятора мобільного пристрою під управлінням операційної системи Android, було показано основні налаштування цього емулятора. Окрім того, було показано основні shortcuts в Android Studio, пояснено їх функціонал.

Були наведені статистичні дані з відсотком підтримуваних пристроїв в залежності від мінімальною підтримуваної версії Android, даний застосунок підтримує 95.6% пристроїв за статистикою на червень 2022 року.

РОЗДІЛ 4.
РОЗРОБКА МОДУЛЮ ВЗАЄМОДІЇ З SQLITE

В мобільній операційній системі Android для взаємодії з СУБД SQLite не потрібні додаткові залежності та бібліотеки, тобто існує низкорівневий API для взаємодії з нею. Окрім того, існує декілька реалізацій ORM – Object-Relational Mapping. Найпопулярніша - Room, це ORM для SQLite під Android від компанії Google. Проте, не дивлячись на те, що ORM допомогає зменшити кількість коду та потенційних помилок, в даному додатку їх використання неможливе, бо завчано неможливо спрогнозувати які таблиці з якими колонками будуть створені користувачем (для ORM це обов'язково треба знати усю схему таблиць, адже потрібно завчано описати класи (Entity) в які будуть мапитись рядки таблиць).

4.1 Огляд низькорівневого API (найважливіших класів)

4.1.1 SQLiteCursor - це реалізація інтерфейсу Cursor, що надає результати SQL запиту до SQLiteDatabase. SQLiteCursor не є синхронізованим, на це треба зважати при роботі багатопотокової програми (можливо доведеться робити свою синхронізацію) [1].

4.1.2 SQLiteDatabase - клас, що надає функціонал керування базою даних SQLite[1].

В SQLiteDatabase існують методи для створення, видалення, виконання команд SQL. Також існує обмеження на імена баз даних (вони повинні бути унікальними в межах одного додатку, але можуть повторюватись у інщих програмах [1].

4.1.3 SQLiteQuery - клас, що представляє запит, що зчитує отримані рядки таблиці в SQLiteQuery. Цей клас використовується класом SQLiteCursor та зазвичай не використовується розробниками. Він також не є синхронізованим[1].

4.2 Огляд найважливіших програмних компонентів для роботи з SQLite

4.2.1 SqliteActivity - клас що є нащадком Activity, тобто це екран.

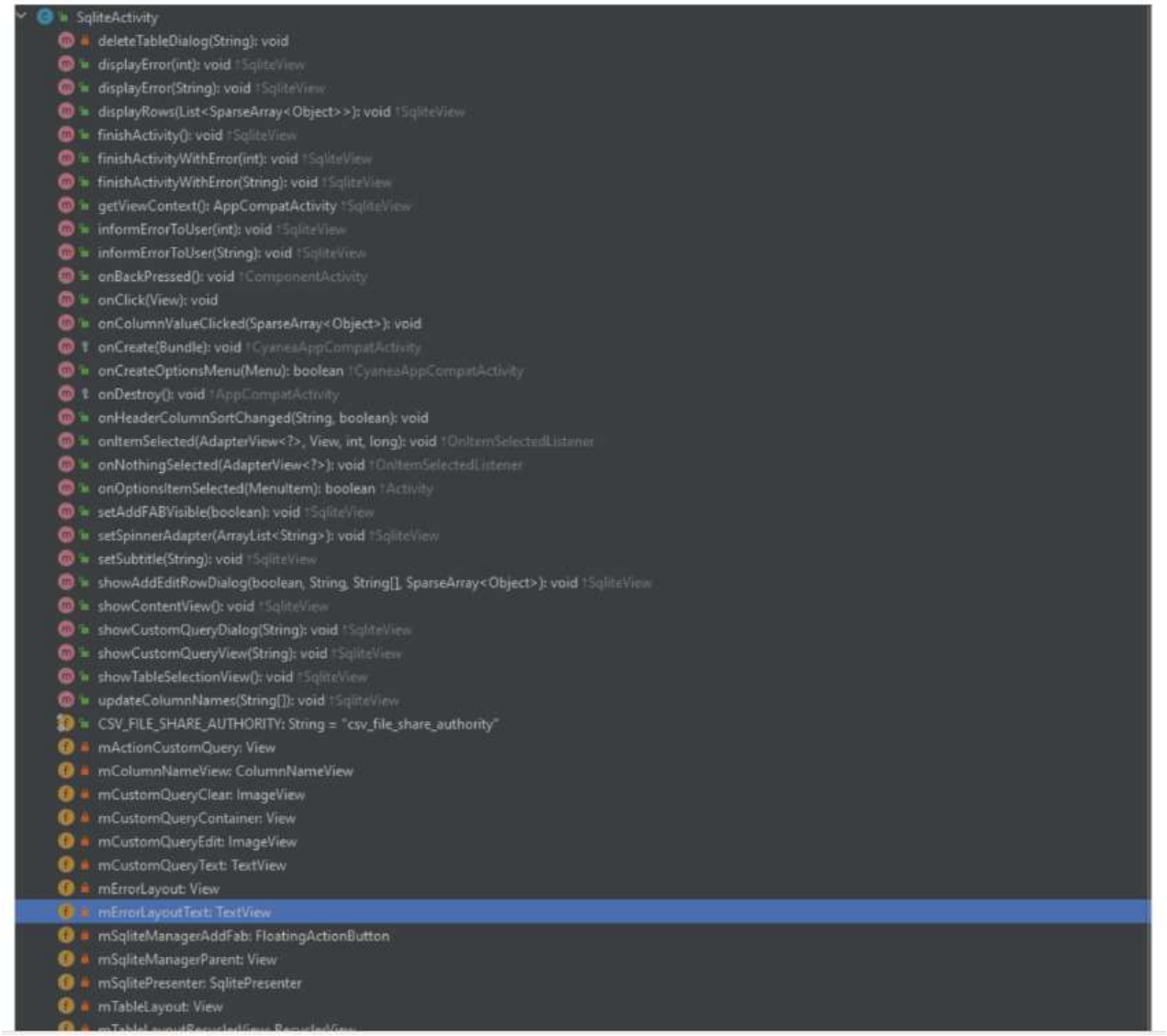


Рисунок 4.1 - SqliteActivity

Найцікавіші поля та методи:

4.2.1.1 showCustomQueryDialog(String previousCustomQuery) - викликає діалогове вікно для своїх запитів.

4.2.1.2 showAddEditRowDialog(final boolean isEdit, final String tableName, final String[] tableColumnNames, final SparseArray<Object> oldColumnValues) - викликає діалогове вікно для додавання або редагування рядку в таблиці.

4.2.1.3 onOptionsItemSelected(MenuItem item) - метод, що регулює логіку роботи тулбару.

4.2.2 SqliteResponse - клас який зберігає у собі інформацію що повертає SQLite.

Найцікавіші поля:

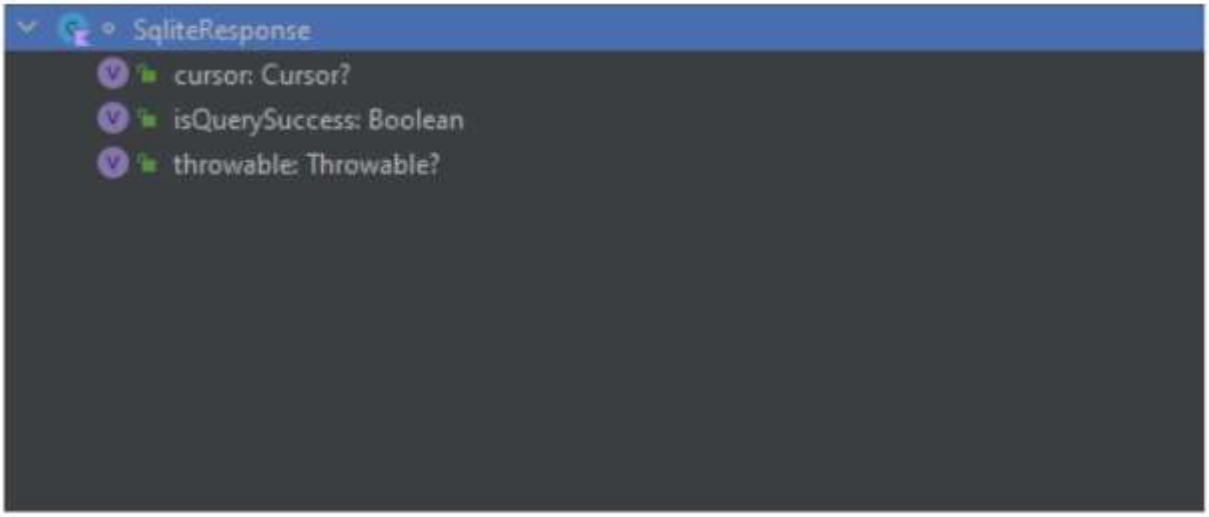


Рисунок 4.2 - SqliteResponse

4.2.2.1 Cursor - дає змогу отримати доступ до даних що повертаються з SQLite.

4.2.2.2 IsQuerySuccess - булеве значення у якому зберігається успішніть запиту в базу даних.

4.2.2.3 Throwable - помилка з якою запит був провалений.

4.2.3 SqliteResponseData – клас у якому зберігається інформація про колонки таблиці.

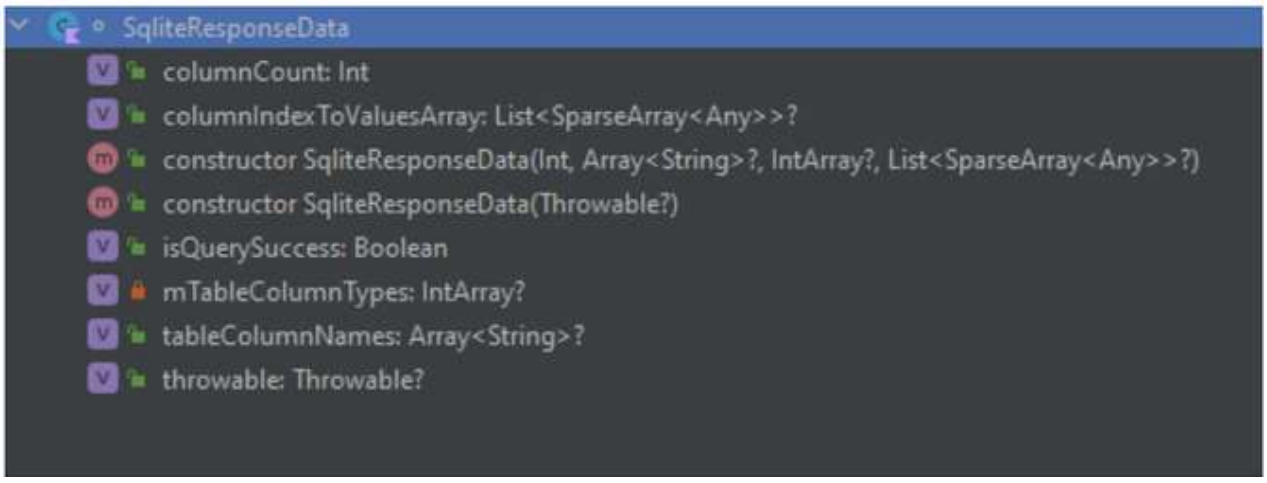


Рисунок 4.3 - SqliteResponseData

4.2.4 SqlSafeUtil - спеціальний клас для боротьби з SQL injection.

SQL injection – це техніка розміщення шкідливого коду в операторах SQL через введення веб-сторінки. Таким чином можна повністю знищити інформацію в базі даних. SQL injection є однією з найпоширеніших методик веб-злому[10].

Варіанти злому через SQL injection:

- SQL injection на основі того, що 1=1 завжди правда.

Нехай користувач вводить в поле свій id.

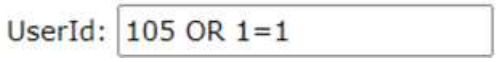


Рисунок 4.4 - Приклад SQL injection

У цьому випадку запит в базу даних буде таким:

SELECT * FROM Users WHERE UserId = 105 OR 1=1; [10]

Так як 1 = 1, то злоумисник отримає всю інформацію про усіх користувачів в системі.

- SQL injection на основі того, що ""="" завжди true

Нехай користувач вводить в поля свої username та password.

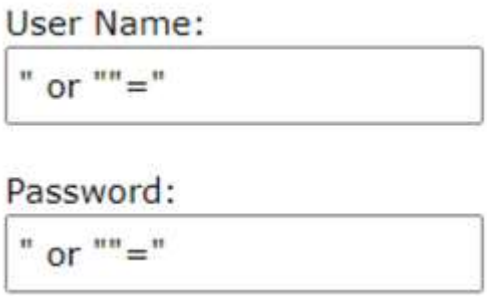


Рисунок 4.5 - Приклад полів вводу

У цьому випадку запит в базу даних буде таким:

SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""="" [10]

Так як ""="" повертає true, то злоумисник отримає всю інформацію про усіх користувачів в системі.

- SQL injection на основі того, що більшість СУБД підтримують можливість виконати декілька запитів підряд (Batched SQL Statements) [10].

Нехай користувач вводить в поле свій id.

Рисунок 4.6 - Приклад SQL injection

У цьому випадку запит в базу даних буде виглядати так:

SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers; [10]

Другий запит видалить таблицю в базі даних

Тому був доданий клас SqlSafeUtil

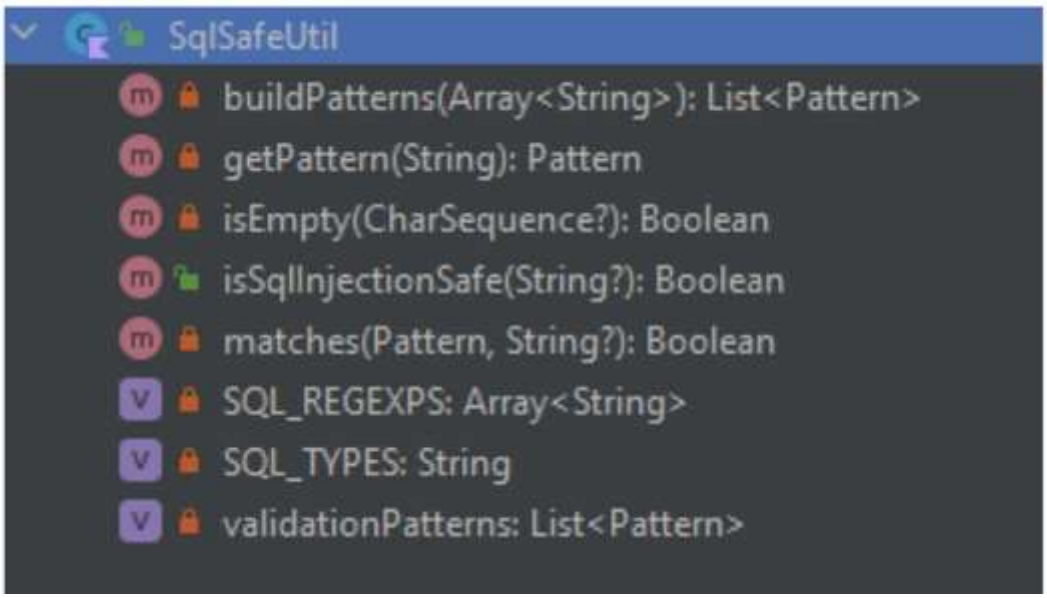


Рисунок 4.7 - SqlSafeUtil

SQL_REGEXPS - регулярний вираз для валідації запитів на мові SQL

isSqlInjectionSafe(dataString: String?) - метод що за допомогою регулярного виразу перевіряє безпечність запитів на мові SQL.

4.3 Демонстрація роботи в додатку з власними запитами на мові SQL

У цьому підрозділі буде продемонтсровано роботу з кастомними запитами користувача. Для початку створимо декілька таблиць. Аби було цікавіше зробимо таблиці з відношенням many-to-many.

4.3.1 Створення таблиць

4.3.1.1 Таблиця студентів

Для початку створимо таблицю студентів. Опис колонок:

student_id INTEGER PRIMARY KEY – колонка відповідає за унікальний айді студента.

first_name TEXT NOT NULL – колонка відповідає за ім’я студента (воно не є унікальним для таблиці).

last_name TEXT NOT NULL – колонка відповідає за прізвище студента (воно не є унікальним для таблиці).

email TEXT NOT NULL UNIQUE – колонка відповідає за електрону пошту студента (вона має бути унікальною).

phone TEXT NOT NULL UNIQUE

Сам SQL:

```
CREATE TABLE students (
                        student_id INTEGER PRIMARY KEY,
                        first_name TEXT NOT NULL,
                        last_name TEXT NOT NULL,
                        email TEXT NOT NULL UNIQUE,
                        phone TEXT NOT NULL UNIQUE
);
```

Рисунок 4.8 - SQL запит

Результат виконання SQL у додатку:

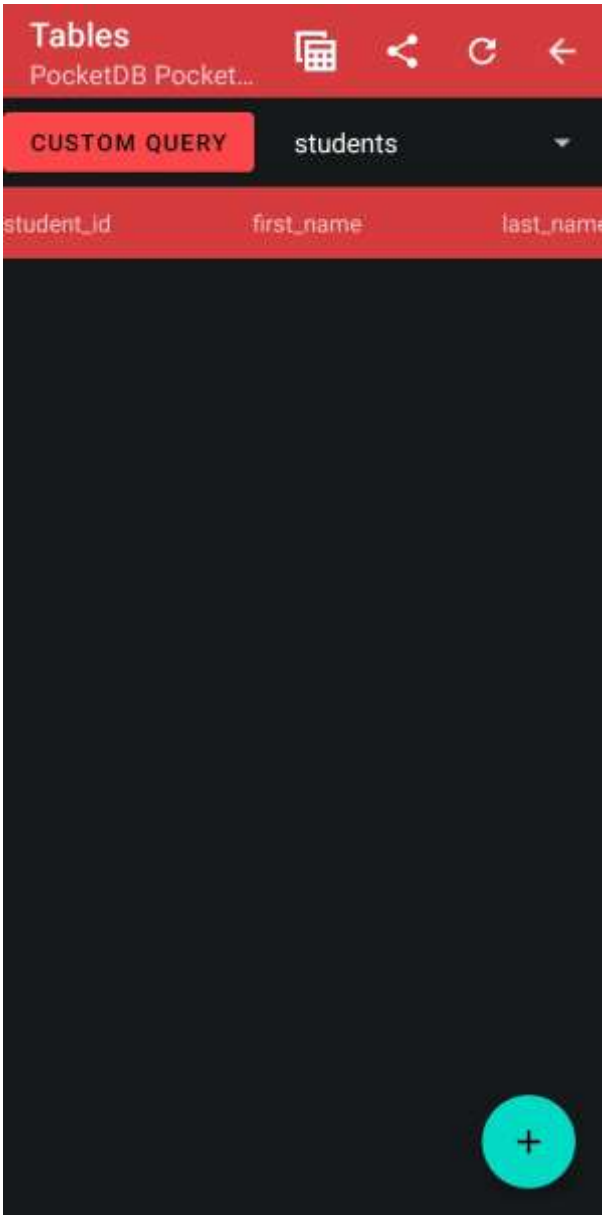


Рисунок 4.9 - Результат виконання SQL

4.3.1.2 Таблиця предметів

Далі створимо таблицю з предметами:

subject_id INTEGER PRIMARY KEY – колонка відповідає за унікальний айді предмету.

name TEXT NOT NULL UNIQUE – колонка відповідає за унікальну назву предмету.

Сам SQL:

```
CREATE TABLE subjects (
                        subject_id INTEGER PRIMARY KEY,
                        name TEXT NOT NULL UNIQUE
);
```

Рисунок 4.9 - SQL запит

Результат виконання SQL у додатку:

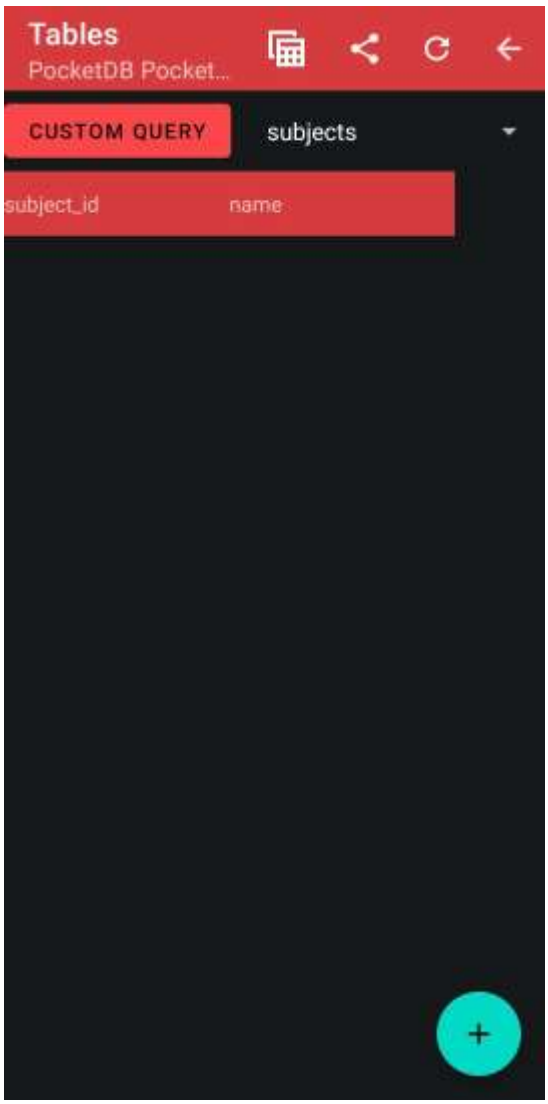


Рисунок 4.10 - Результат виконання SQL

4.3.1.3 Таблиця зв’язків студентів та предметів

Далі треба створити таблицю що буде пов’язувати різних студентів з різними предметами.

student_id INTEGER – колонка що пов’язує студента

subject_id INTEGER – колонка що пов’язує предмет

Сам SQL:

```
CREATE TABLE students_subjects(
                                student_id INTEGER,
                                subject_id INTEGER,
                                PRIMARY KEY (student_id, subject_id),
                                FOREIGN KEY (student_id)
                                    REFERENCES students (student_id)
                                    ON DELETE CASCADE
                                    ON UPDATE NO ACTION,
                                FOREIGN KEY (subject_id)
                                    REFERENCES subjects (subject_id)
                                    ON DELETE CASCADE
                                    ON UPDATE NO ACTION
);
```

Рисунок 4.11 - SQL запит

Результат виконання SQL у додатку:

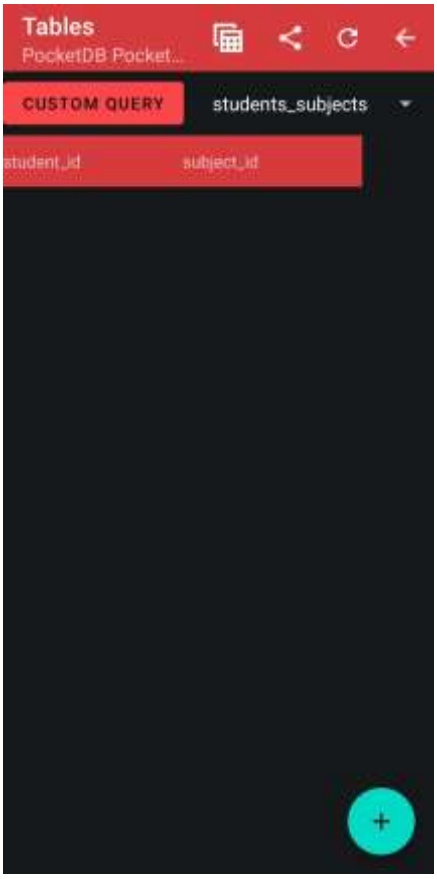


Рисунок 4.12 - Результат виконання SQL

4.3.2 Заповнення таблиць тестовими даними

4.3.2.1 Заповнення таблиці студентів тестовими даними

SQL запити:

```
INSERT INTO students  (student_id, first_name, last_name, email, phone)
VALUES(1, 'Максим', 'Дудка', 'example_mail@ukr.net', '+38011111111');

INSERT INTO students  (student_id, first_name, last_name, email, phone)
VALUES(2, 'Вероніка', 'Рахуба', 'example_mail2@ukr.net', '+38022222222');

INSERT INTO students  (student_id, first_name, last_name, email, phone)
VALUES(3, 'Олексій', 'Самутін', 'example_mail3@ukr.net', '+38033333333');
```

Рисунок 4.13 - SQL запити

Результат виконання SQL у додатку:

Рисунок 4.14 - Результат виконання SQL

4.3.2.2 Заповнення таблиці предметів тестовими даними

SQL запити:

```
INSERT INTO subjects  (subject_id, name)
VALUES(1, 'Системне програмування');

INSERT INTO subjects  (subject_id, name)
VALUES(2, 'Вища математика');
```

Рисунок 4.15 - SQL запити

Результат виконання SQL у додатку:

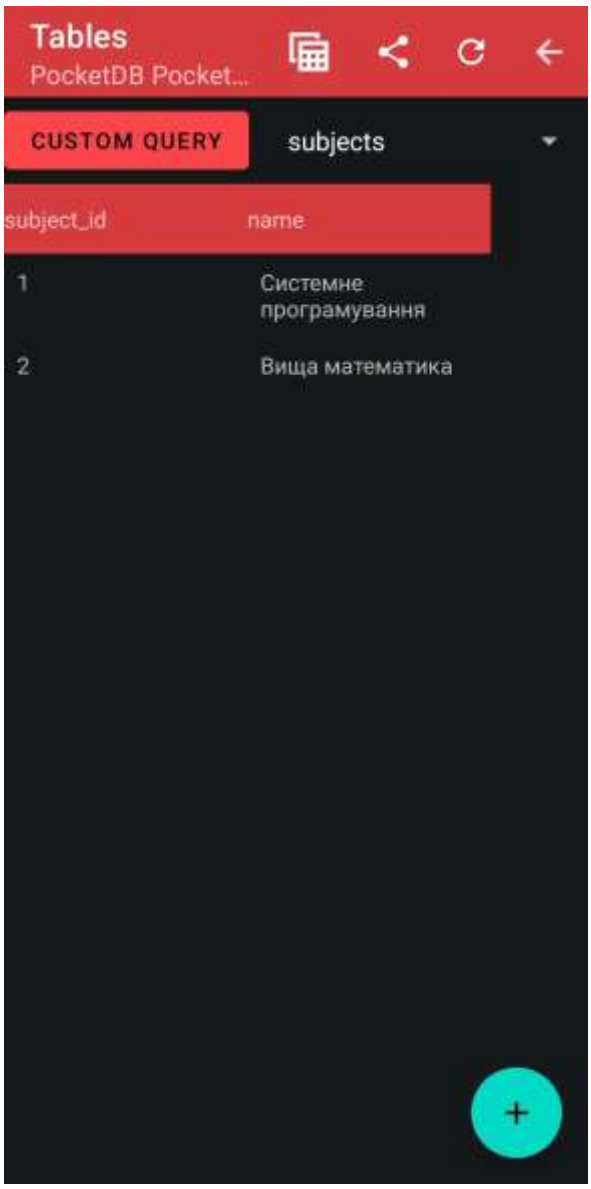


Рисунок 4.15 - Результат виконання SQL

4.3.2.3 Заповнення таблиці предметів тестовими даними

SQL запити:

```
INSERT INTO students_subjects  (student_id, subject_id)
VALUES(1, 1);

INSERT INTO students_subjects  (student_id, subject_id)
VALUES(1, 2);

INSERT INTO students_subjects  (student_id, subject_id)
VALUES(2, 1);

INSERT INTO students_subjects  (student_id, subject_id)
VALUES(2, 2);

INSERT INTO students_subjects  (student_id, subject_id)
VALUES(3, 1);

INSERT INTO students_subjects  (student_id, subject_id)
VALUES(3, 2);
```

Рисунок 4.16 - SQL запити

Після виконання цих запитів, усі студенти будуть пов’язані з усіма предметами.

Результат виконання SQL у додатку:

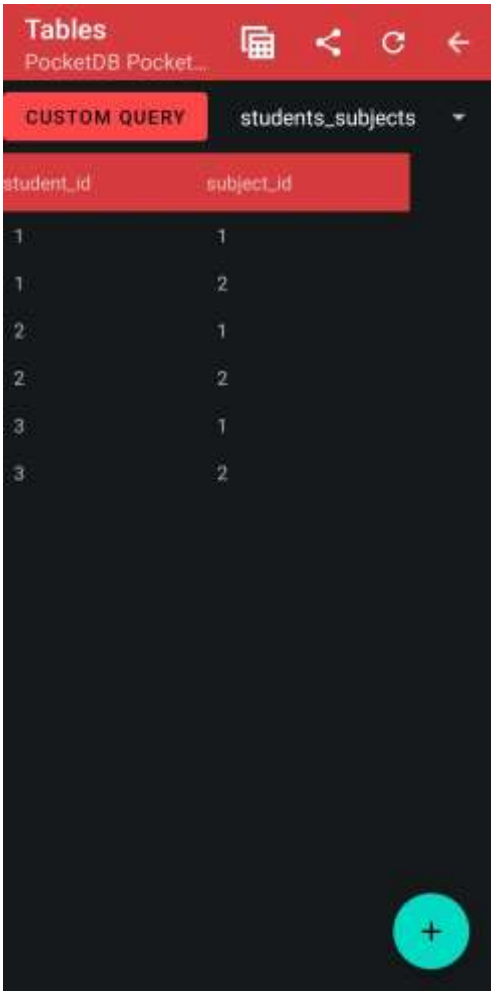


Рисунок 4.17 - Результат виконання SQL

4.3.3 Створення запитів на основі отриманої бази даних студентів та предметів

4.3.3.1 Запит на отримання email студентів у як ім’я Максим або Вероніка:

SQL:

```
SELECT email FROM students WHERE first_name = 'Максим' OR first_name = 'Вероніка'
```

Рисунок 4.18 - SQL запит

Результат виконання SQL у додатку:

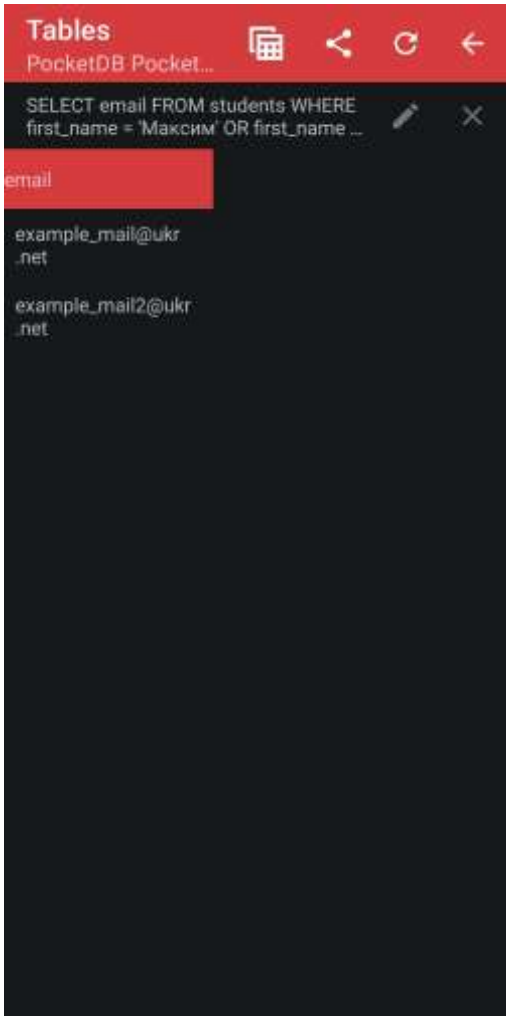


Рисунок 4.19 - Результат виконання SQL

4.3.3.2 Запит на отримання прізвищ студентів які вивчали системне програмування (запит з JOIN):

SQL:

```
SELECT last_name FROM students s
                    LEFT JOIN subjects sub ON s.student_id
WHERE sub.name = 'Системне програмування';
```

Рисунок 4.20 - SQL запит

Результат виконання SQL у додатку:

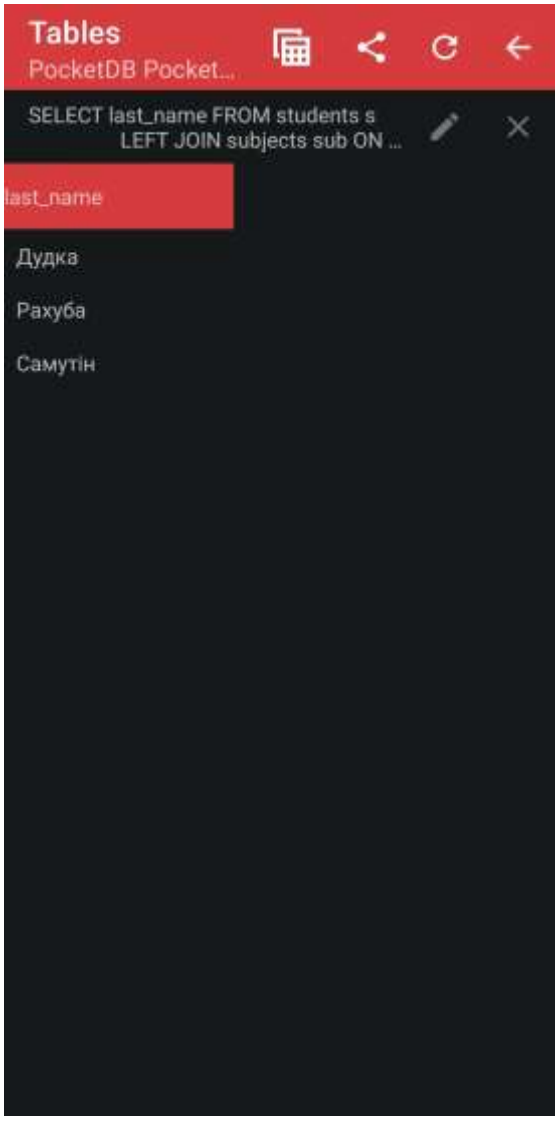


Рисунок 4.21 - Результат виконання SQL

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі було розглянуто чому вибір пав на низькорівневий API для роботи з SQLite, а не ORM (наприклад Room від компанії Google). Були наведені основні класи для роботи з базами даних в операційній системі Android. Були продемонстровані основні програмні компоненти для роботи з API СУБД SQLite.

Також було розглянуто варіанти атаки на бази даних за допомогою SQL injection та продемонстровано можливий варіант усунення такої можливості.

Окрім того було поетапно продемонстровано створення бази даних студентів та предметів, показано її заповнення, а також запити до готової бази даних повністю за допомогою користувацьких запитів.

ВИСНОВКИ

Під час виконання дипломної роботи було розроблено мобільний застосунок на операційній системі Android для покращення практичних навичок у використані мови запитів SQL. Дана програма дає можливість своїм користувачам за допомогою мобільного додатку взаємодіяти з СУБД SQLite.

В першому розділі були розглянуті інтегровані середовища розробки. Також у першому розділі були розглянуті аналоги з Google Play Market. Були встановлені їх недоліки та переваги.

В другому розділі були детально розписані мови програмування Kotlin, Java, Swift, Dart та JavaScript. Також було розглянуто реляційні СУБД їх переваги та недолікиі. Окрім того було розглянуті варіанти бібліотек для кастомізації зовнішнього вигляду застосунку.

В третьому розділі було показано процес створення Android проекту в Android Studio. Також було показано приклад створення емулятора мобільного пристрою під управлінням операційної системи Android. Також були продемонстровані найкорисніші shortcuts для роботи в Android Studio.

В четвертому розділі було розглянуто використані системні класи для в взаємодії з SQLite. також було розглянуто найцікавіші програмні компоненти що були створені для роботи с СУБД. Окрім того було показано можливості роботи з SQL запитами користувача (вкючаючи створення таблиць, їх заповнення, витягання даних в тому числи складними запитами з JOIN).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Android documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com.
2. Jetbrains Kotlin documentation [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org.
3. Java (programming language) [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Java_(programming_language).
4. Программирование под Android на Java [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/java/android/.
5. PostgreSQL - Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/postgresql/postgresql_overview.htm.
6. SQLite Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/sqlite/index.htm.
7. H2 Database - Introdaction [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/h2_database/h2_database_introduction.htm.
8. MySQL - Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/mysql/mysql-introduction.htm.
9. MariaDB - Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/mariadb/mariadb_introduction.htm.
10. SQL Injection [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/sql/sql_injection.asp.
11. Kotlin - Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/kotlin/kotlin_overview.htm.
12. Java - Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/java/java_overview.htm.

13. JavaScript - Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/javascript/javascript_overview.htm.
14. Swift - Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/swift/swift_overview.htm.
15. Cyanea [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/jaredrummler/Cyanea.
16. Theming in Compose [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/jetpack/compose/themes.
17. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL.
18. Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom//DATABASE SYSTEMS The Complete Book. 2009. P. 2—27.
19. 11 типів сучасних баз даних: короткий опис, схеми і приклади БД [Електронний ресурс] – 2021 – Режим доступу до ресурсу: https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikladi-bd.
20. Бази даних та їх види. Основні поняття. Класифікація БД. Види СУБД. MS Access [Електронний ресурс] – 2018. – Режим доступу до ресурсу: https://naurok.com.ua/bazi-danih-ta-h-vidi-osnovni-ponyattya-klasifikaciya-bd-vidi-subd-ms-access-74441.html.
21. Analyst: There’s a great future in iPhone apps [Електронний ресурс]. – 2008. – Режим доступу до ресурсу: https://web.archive.org/web/20220202154253/https://venturebeat.com/2008/06/11/analyst-theres-a-great-future-in-iphone-apps/
22. Ventola, C. Lee. Mobile Devices and Apps for Health Care Professionals: Uses and Benefits 2015.
23. Xcode 14 [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://developer.apple.com/xcode/.

24. Mobile app market to grow 270% to $189 billion by 2020, with games accounting for 55% [Електронний ресурс]. – 2016. – Режим доступу до ресурсу:https://web.archive.org/web/20180210003249/https://venturebeat.com/2016/11/02/mobile-app-market-to-grow-270-to-189-billion-by-2020-with-games-accounting-for-55/.
25. AppCode [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/objc/.
26. Custom styles in Android libraries [Електронний ресурс]. – 2011. – Режим доступу до ресурсу: https://blog.danlew.net/2011/03/03/custom_styles_in_android_libraries/.
27. Android developer portal with tools, libraries, and apps [Електронний ресурс].– Режим доступу до ресурсу: https://android-arsenal.com/.
28. A Material Design 3 Theme Engine for Android. [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://androidexample365.com/a-material-design-3-theme-engine-for-android/
29. Dart | Введение в язык. Первая программа [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://metanit.com/dart/tutorial/1.1.php/.
30. Нелогичный JavaScript потому-что такова его природа [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://drbrain.ru/articles/illogical-javascript/.
31. The Importance of User Interface Design [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework.
32. Importance of UI/UX design in mobile applications in 2022 [UPDATED] [Електронний ресурс] – 2022. – Режим доступу до ресурсу: https://bootcamp.uxdesign.cc/importance-of-ui-ux-design-in-the-development-of-mobile-apps-2c7ad63fcd3b.
33. Visual Studio Code [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://code.visualstudio.com/docs.

ДОДАТОК 1

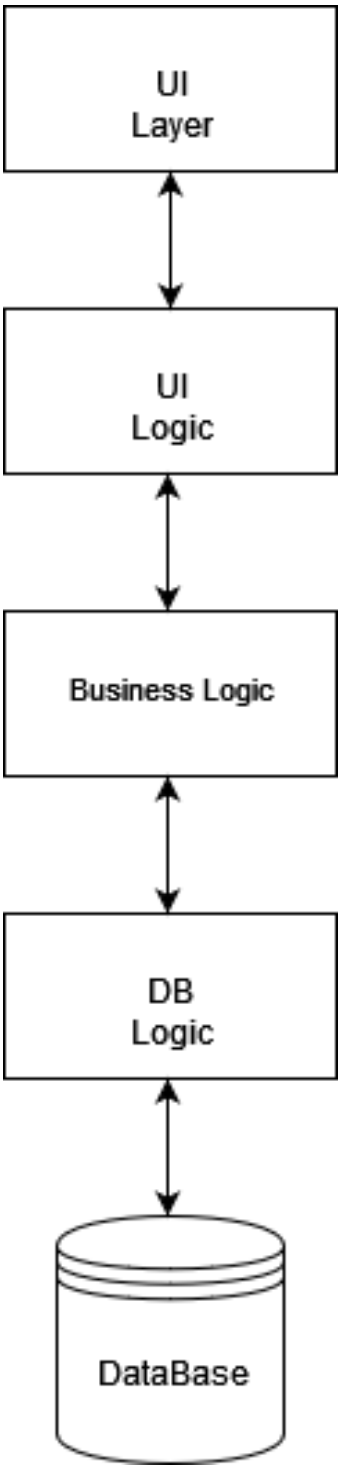
Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)

Схема взаємодії

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		Дудка М.В.			Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Алещенко О.В.			Схема взаємодії		НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81
Н. Контр.		Сімоненко В. П.					
Затвердив							

ДОДАТОК 2

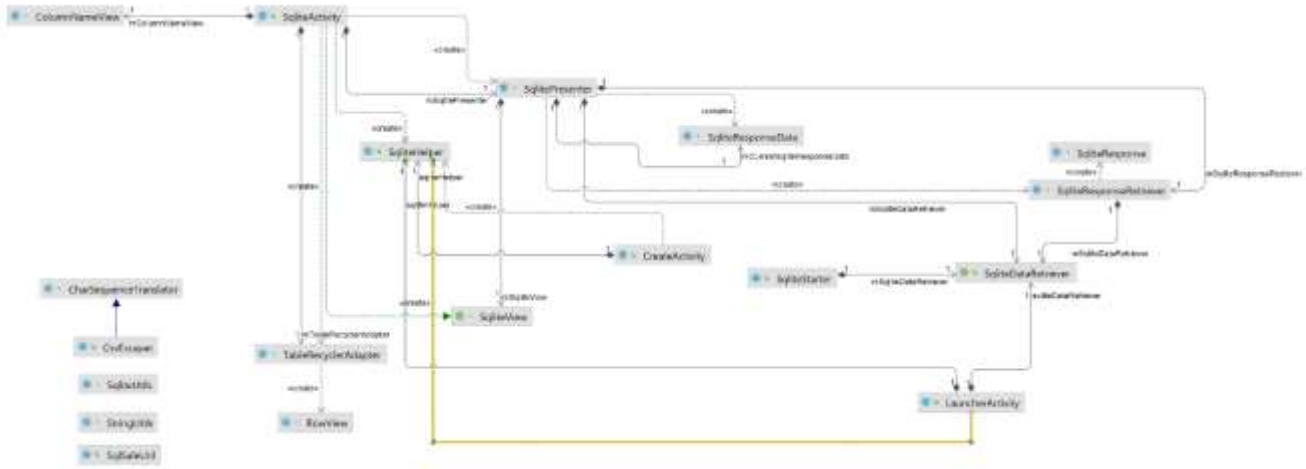
Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)

Функціональна схема (діаграма класів)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробив		Дудка М.В.			Застосунок на платформі Android для навчання взаємодії з базою даних(модуль роботи з SQLite)Функціональна схема (діаграма класів)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Алещенко О.В.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81	
Н. Контр.		Сімоненко В. П.					
Затвердив							

ДОДАТОК 3

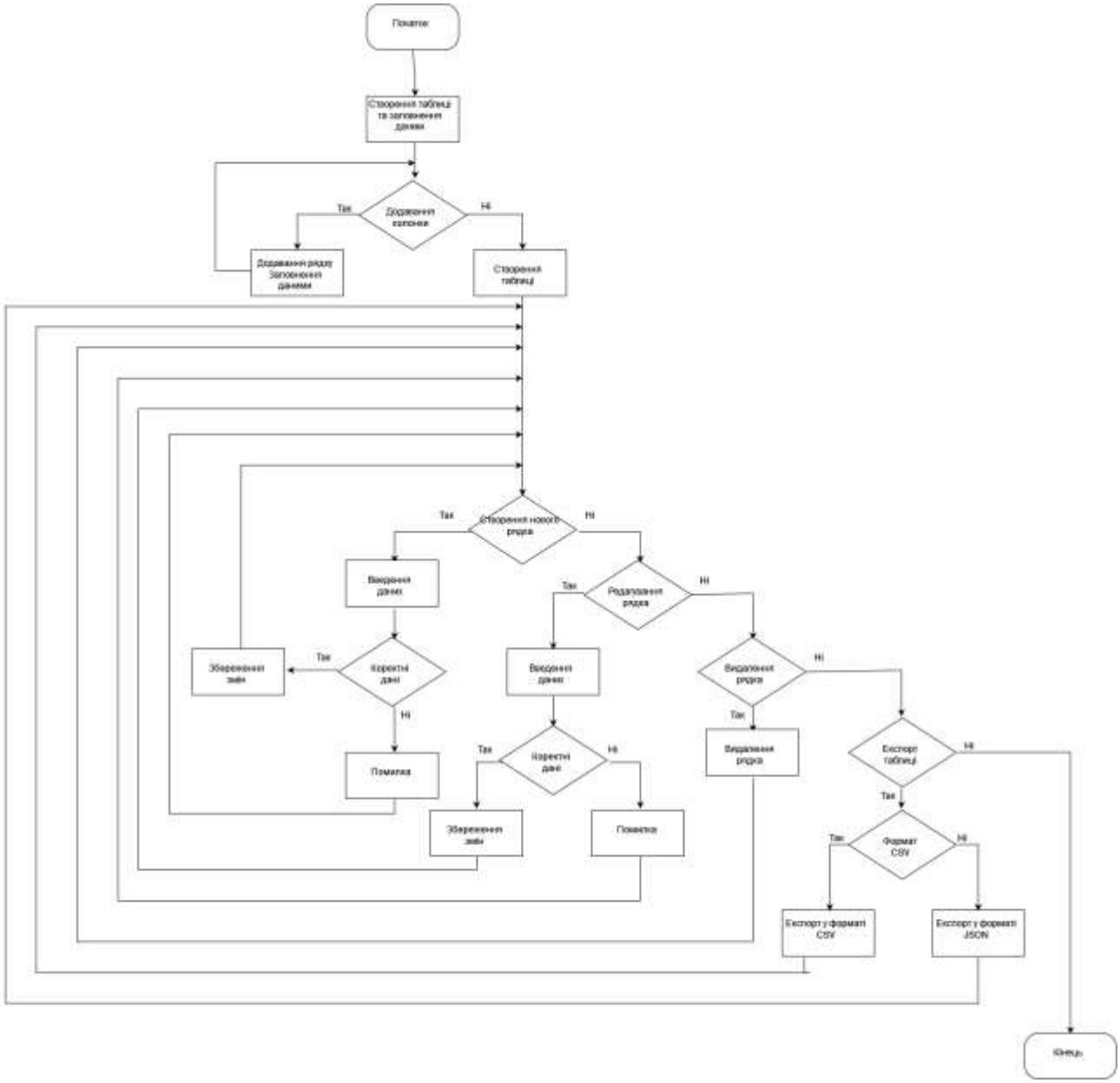
Застосунок на платформі Android для навчання взаємодії з базою даних
(модуль роботи з SQLite)

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		Дудка М.В.			Застосунок на платформі Android для навчання взаємодії з базою даних ((модуль роботи з SQLite)	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Алещенко О.В,			Алгоритм дій програмного забезпечення		НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-81
Н. Контр.		Сімоненко В. П.					
Затвердив							

ДОДАТОК 4

Застосунок на платформі Android для навчання взаємодії з базою даних (модуль роботи з SQLite)

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 39

Київ 2022 р

App.kt

```
package my.project.db

import android.app.Application
import com.jaredrummler.cyanea.Cyanea

class MyApplication : Application() {

    override fun onCreate() {
        super.onCreate()
        Cyanea.init(this, resources)
    }
}
```

CreateActivity.java

```
package my.project.db;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.widget.Toolbar;

import android.content.DialogInterface;
import android.content.Intent;
import android.database.sqlite.SQLiteException;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import com.jaredrummler.cyanea.app.CyaneaAppCompatActivity;


import java.util.ArrayList;


import my.project.db.manager.SqliteDataRetriever;
import my.project.db.manager.SqliteHelper;
import my.project.db.manager.SqliteStarter;



public class CreateActivity extends CyaneaAppCompatActivity {


    private LinearLayout mainLinearLayout;
    private final ArrayList<String> names = new ArrayList<>();
```

```
    private final ArrayList<String> types = new ArrayList<>();
    private SqliteHelper sqliteHelper;

@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_create);
        sqliteHelper = new SqliteHelper(this);
        mainLinearLayout = findViewById(R.id.main);
        mainLinearLayout.addView(addLine(new LinearLayout(this)));
        setTitle(R.string.add_table);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
    }
public void onClick(View view) {
        LinearLayout lin = new LinearLayout(this);
        addLine(lin);
        mainLinearLayout.addView(lin);
    }

    private LinearLayout addLine(LinearLayout linearLayout) {
        EditText ed1 = new EditText(this);
        ed1.setHint(getString(R.string.name) + (mainLinearLayout.getChildCount()
- 1));
        Spinner ed2 = new Spinner(this);
        ArrayList<String> spinnerArray = new ArrayList<>();
        spinnerArray.add(getString(R.string.text));
        spinnerArray.add(getString(R.string.integer));
        spinnerArray.add(getString(R.string.real));
        ArrayAdapter<String> spinnerArrayAdapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, spinnerArray);
        ed2.setAdapter(spinnerArrayAdapter);
        LinearLayout.LayoutParams param = new LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.MATCH_PARENT,
                LinearLayout.LayoutParams.MATCH_PARENT,
                1.0f
        );
        ed1.setLayoutParams(param);
        ed2.setLayoutParams(param);
        linearLayout.addView(ed1);
        linearLayout.addView(ed2);
        return linearLayout;
    }
```

```
public void onCreateClick(View view) {
        String tableName = validateTableName();
        if (tableName == null) {
            Toast.makeText(this, getString(R.string.wrong_table_name),
Toast.LENGTH_LONG).show();
            return;
        }
        createTableDialog(tableName);
    }

    private void createTableDialog(String name) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder
                .setTitle(getString(R.string.create_table_sure) + name + "?")
                .setCancelable(false)
                .setPositiveButton(getString(R.string.yes), new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        try {
                            getNamesAndTypes();
                            sqliteHelper.createTable(name, names, types);
                            names.clear();
                            types.clear();
                            dialog.cancel();
                            SqliteDataRetriever sqliteDataRetriever = new
LauncherActivity.HelperSqliteDataRetriever(sqliteHelper);

SqliteStarter.launchSqliteManager(CreateActivity.this, sqliteDataRetriever,
BuildConfig.APPLICATION_ID);
                            finish();
                        } catch (SQLiteException e) {
                            Toast.makeText(CreateActivity.this,
getString(R.string.exist), Toast.LENGTH_LONG).show();
                        }
                    }
                })
                .setNegativeButton(getString(R.string.no), new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        dialog.cancel();
                    }
                }).show();
    }
```

```
    private void getNamesAndTypes() {
        for (int i = 2; i < mainLinearLayout.getChildCount(); i++) {
            LinearLayout supp = (LinearLayout) mainLinearLayout.getChildAt(i);
            TextView suppText = (TextView) supp.getChildAt(0);
            Spinner suppSpinner = (Spinner) supp.getChildAt(1);
            names.add(validateName(suppText));
            types.add(suppSpinner.getSelectedItem().toString());
        }
    }
    private String validateName(TextView name) {
        String supp = name.getText().toString();
        if (supp.equals("")) {
            return name.getHint().toString();
        }
        if (supp.contains(" ")) {
            return supp.replace(" ", "_");
        }
        return supp;
    }

    private String validateTableName() {
        EditText editText = findViewById(R.id.tableName);
        String supp = editText.getText().toString();
        if (supp.equals("")) {
            return "name";
        }
        if (Character.isDigit(supp.charAt(0)) || supp.contains(".") ||
supp.contains("-")) {
            return null;
        }
        return supp;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.create_menu, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (item.getItemId() == R.id.back) {
```

```
            onBackPressed();
            return true;
        }
        return super.onOptionsItemSelected(item);
    }



Override
    public void onBackPressed() {
        startActivity(new Intent(CreateActivity.this, LauncherActivity.class));
        finish();
    }
}
```

LauncherActivity.java

```
package my.project.db;

import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import androidx.annotation.NonNull;
import androidx.core.content.ContextCompat;

import com.jaredrummler.cyanea.app.CyaneaAppCompatActivity;
import com.jaredrummler.cyanea.prefs.CyaneaSettingsActivity;

import my.project.db.manager.SqliteDataRetriever;
import my.project.db.manager.SqliteHelper;
import my.project.db.manager.SqliteStarter;

public class LauncherActivity extends CyaneaAppCompatActivity implements
View.OnClickListener {

    SqliteHelper sqliteHelper;
    SqliteDataRetriever sqliteDataRetriever;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_launcher);
        sqliteHelper = new SqliteHelper(LauncherActivity.this);
        sqliteDataRetriever = new HelperSqliteDataRetriever(sqliteHelper);

        if (sqliteHelper.getTableNames().size() < 2) {
            Button button = findViewById(R.id.show_all_tables);
            markButtonDisable(button);
        }

        findViewById(R.id.create_new_table).setOnClickListener(this);
        findViewById(R.id.show_all_tables).setOnClickListener(this);
        findViewById(R.id.customisation_button).setOnClickListener(this);
    }

    private void markButtonDisable(Button button) {
        button.setEnabled(false);
        button.setTextColor(ContextCompat.getColor(button.getContext(),
```

```
R.color.white));
        button.setBackgroundColor(
                ContextCompat.getColor(
                        button.getContext(),
                        android.R.color.darker_gray
                )
        );
    }


    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.show_all_tables) {
            SqliteStarter.launchSqliteManager(LauncherActivity.this,
sqliteDataRetriever, BuildConfig.APPLICATION_ID);
            finish();
        } else if (v.getId() == R.id.customisation_button) {
            startActivity(new Intent(this, CyaneaSettingsActivity.class));
        } else if (v.getId() == R.id.create_new_table) {
            startActivity(new Intent(LauncherActivity.this,
CreateActivity.class));
            finish();
        }
    }

    public static class HelperSqliteDataRetriever implements SqliteDataRetriever
{
        SqliteHelper mSqliteHelper;
        SQLiteDatabase mSQLiteDatabase;

        HelperSqliteDataRetriever(SqliteHelper sqliteHelper) {
            mSqliteHelper = sqliteHelper;
            mSQLiteDatabase = mSqliteHelper.getWritableDatabase();
        }

        @Override
        public Cursor rawQuery(@NonNull String query, String[] selectionArgs) {
            if (mSQLiteDatabase == null || !mSQLiteDatabase.isOpen()) {
                mSQLiteDatabase = mSqliteHelper.getWritableDatabase();
            }
            return mSQLiteDatabase.rawQuery(query, selectionArgs);
        }

        @Override
        public String getDatabaseName() {
            return mSqliteHelper.getDatabaseName();
        }

        @Override
        public void freeResources() {

        }
    }
}
```

ColumnNameView.java

```
package my.project.db.manager;

import android.annotation.SuppressLint;
import android.annotation.TargetApi;
import android.content.Context;
import android.graphics.drawable.Drawable;
import android.os.Build;
import android.text.Spannable;
```

```
import android.text.SpannableString;
import android.text.style.ImageSpan;
import android.util.AttributeSet;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;

import androidx.annotation.Nullable;

import my.project.db.R;

@SuppressLint("Instantiatable")
class ColumnNameView extends LinearLayout implements View.OnClickListener {

    private String[] mTableColumnNames;

    public static final byte NO_SORT = 0;
    public static final byte ASCENDING_SORT = 1;
    public static final byte DESCENDING_SORT = 2;

    private ColumnHeaderSortChangeListener mColumnHeaderSortChangeListener;

    public ColumnNameView(Context context) {
        this(context, null);
    }

    public ColumnNameView(Context context, @Nullable AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public ColumnNameView(Context context, @Nullable AttributeSet attrs, int
defStyleAttr) {
        super(context, attrs, defStyleAttr);
        init();
    }

    @TargetApi(Build.VERSION_CODES.LOLLIPOP)
    public ColumnNameView(Context context, AttributeSet attrs, int defStyleAttr,
int defStyleRes) {
        super(context, attrs, defStyleAttr, defStyleRes);
        init();
    }

    private void init() {
        setOrientation(HORIZONTAL);
    }

    private TextView getTextView() {
        return (TextView)
LayoutInflater.from(getContext()).inflate(R.layout.column_text_header_item,
this, false);
    }

    ColumnNameView setData(String[] tableColumnNames) {
        removeAllViews();

        mTableColumnNames = tableColumnNames;

        int index = 0;
        for (String currentColumnName : mTableColumnNames) {
            TextView textView = getTextView();
            TableNameHeaderViewTag tableNameHeaderViewTag = new
TableNameHeaderViewTag();
```

```
            tableNameHeaderViewTag.columnIndex = index;
            tableNameHeaderViewTag.sortOrder = NO_SORT;
            textView.setTag(tableNameHeaderViewTag);
            textView.setText(currentColumnName);
            textView.setOnClickListener(this);
            addView(textView);
            index++;
        }
        return this;
    }

    private class TableNameHeaderViewTag {
        private byte sortOrder = 0;
        private int columnIndex = 0;
    }

    @Override
    public void onClick(View v) {
        TableNameHeaderViewTag tableNameHeaderViewTag = (TableNameHeaderViewTag)
v.getTag();
        byte nextSortOrder = getNextSortOrder(tableNameHeaderViewTag.sortOrder);

        for (int i = 0; i < getChildCount(); i++) {
            View currentChild = getChildAt(i);
            if (currentChild instanceof TextView) {
                ((TableNameHeaderViewTag) currentChild.getTag()).sortOrder =
NO_SORT;
                int columnIndex = ((TableNameHeaderViewTag)
currentChild.getTag()).columnIndex;
                ((TextView)
currentChild).setText(mTableColumnNames[columnIndex]);
            }
        }

        boolean isAscendingOrder = nextSortOrder == ASCENDING_SORT;
        if (mColumnHeaderSortChangeListener != null) {

mColumnHeaderSortChangeListener.onHeaderColumnSortChanged(mTableColumnNames[tabl
eNameHeaderViewTag.columnIndex], isAscendingOrder);
        }

        tableNameHeaderViewTag.sortOrder = nextSortOrder;
        Drawable ascendingDescendingIcon;
        if (isAscendingOrder) {
            ascendingDescendingIcon =
getContext().getResources().getDrawable(R.drawable.ic_sort_ascending_white_24dp)
;
        } else {
            ascendingDescendingIcon =
getContext().getResources().getDrawable(R.drawable.ic_sort_descending_white_24dp
);
        }

        String tableName =
mTableColumnNames[tableNameHeaderViewTag.columnIndex];
        Spannable spannableString = new SpannableString(tableName + "    ");
        ascendingDescendingIcon.setBounds(0, 0, 56, 56);
        ImageSpan image = new ImageSpan(ascendingDescendingIcon,
ImageSpan.ALIGN_BASELINE);
        spannableString.setSpan(image, tableName.length() + 2,
spannableString.length(), Spannable.SPAN_INCLUSIVE_EXCLUSIVE);
        ((TextView) v).setText(spannableString);
    }
```

```
    public void setColumnHeaderSortChangeListener(ColumnHeaderSortChangeListener
columnHeaderSortChangeListener) {
        this.mColumnHeaderSortChangeListener = columnHeaderSortChangeListener;
    }

    public String[] getTableColumnNames() {
        return mTableColumnNames;
    }

    private byte getNextSortOrder(int currentSortOrder) {
        if (currentSortOrder == NO_SORT) {
            return ASCENDING_SORT;
        } else if (currentSortOrder == ASCENDING_SORT) {
            return DESCENDING_SORT;
        } else if (currentSortOrder == DESCENDING_SORT) {
            return ASCENDING_SORT;
        }
        return ASCENDING_SORT;
    }

    interface ColumnHeaderSortChangeListener {
        void onHeaderColumnSortChanged(String columnName, boolean
isAscendingOrder);
    }
}
```

CharSequanceTranslator.kt

```
package my.project.db.manager.csv

import java.io.IOException
import java.io.StringWriter
import java.io.Writer

abstract class CharSequenceTranslator {
    @Throws(IOException::class)
    abstract fun translate(input: CharSequence?, index: Int, out: Writer?): Int
    fun translate(input: CharSequence?): String? {
        return if (input == null) {
            null
        } else try {
            val writer = StringWriter(input.length * 2)
            translate(input, writer)
            writer.toString()
        } catch (ioe: IOException) {
            throw RuntimeException(ioe)
        }
    }

    @Throws(IOException::class)
    private fun translate(input: CharSequence?, out: Writer?) {
        requireNotNull(out) { "The Writer must not be null" }
        if (input == null) {
            return
        }
        var pos = 0
        val len = input.length
        while (pos < len) {
            val consumed = translate(input, pos, out)
            if (consumed == 0) {
                val c1 = input[pos]
                out.write(c1.toInt())
                pos++
                if (Character.isHighSurrogate(c1) && pos < len) {
                    val c2 = input[pos]
```

```
                    if (Character.isLowSurrogate(c2)) {
                        out.write(c2.toInt())
                        pos++
                    }
                }
                continue
            }

            for (pt in 0 until consumed) {
                pos += Character.charCount(Character.codePointAt(input, pos))
            }
        }
    }
}
```

CsvEscaper.kt

```
package my.project.db.manager.csv

import java.io.IOException
import java.io.Writer

class CsvEscaper() : CharSequenceTranslator() {
    @Throws(IOException::class)
    override fun translate(input: CharSequence?, index: Int, out: Writer?): Int
{
        check(index == 0) { "Should never reach the [1] index" }
        if (StringUtils.containsNone(input.toString(), *CSV_SEARCH_CHARS)) {
            out!!.write(input.toString())
        } else {
            out!!.write(CSV_QUOTE.toInt())
            out.write(
                StringUtils.replace(
                    input.toString(),
                    CSV_QUOTE_STR,
                    CSV_QUOTE_STR + CSV_QUOTE_STR
                )
            )
            out.write(CSV_QUOTE.toInt())
        }
        return Character.codePointCount(input!!, 0, input.length)
    }

    companion object {
        var instance: CsvEscaper? = null
            get() {
                if (field == null) {
                    field = CsvEscaper()
                }
                return field
            }
            private set
        private const val CSV_DELIMITER = ','
        private const val CSV_QUOTE = '"'
        private const val LF = '\n'
        private const val CR = '\r'
        private const val CSV_QUOTE_STR = CSV_QUOTE.toString()
        private val CSV_SEARCH_CHARS = charArrayOf(CSV_DELIMITER, CSV_QUOTE, CR,
LF)
    }
}
```

StringUtils.kt

```
package my.project.db.manager.csv
```

```
internal object StringUtils {
    private const val INDEX_NOT_FOUND = -1
    fun containsNone(cs: CharSequence?, vararg searchChars: Char): Boolean {
        if (cs == null || searchChars == null) {
            return true
        }
        val csLen = cs.length
        val csLast = csLen - 1
        val searchLen = searchChars.size
        val searchLast = searchLen - 1
        for (i in 0 until csLen) {
            val ch = cs[i]
            for (j in 0 until searchLen) {
                if (searchChars[j] == ch) {
                    if (Character.isHighSurrogate(ch)) {
                        if (j == searchLast) {
                            return false
                        }
                        if (i < csLast && searchChars[j + 1] == cs[i + 1]) {
                            return false
                        }
                    } else {
                        return false
                    }
                }
            }
        }
        return true
    }

    fun replace(text: String, searchString: String, replacement: String?):
String {
        return replace(text, searchString, replacement, -1)
    }

    private fun replace(
        text: String,
        searchString: String,
        replacement: String?,
        max: Int,
        ignoreCase: Boolean = false
    ): String {
        var searchString = searchString
        var max = max
        if (isEmpty(text) || isEmpty(searchString) || replacement == null || max
== 0) {
            return text
        }
        var searchText = text
        if (ignoreCase) {
            searchText = text.toLowerCase()
            searchString = searchString.toLowerCase()
        }
        var start = 0
        var end = searchText.indexOf(searchString, start)
        if (end == INDEX_NOT_FOUND) {
            return text
        }
        val replLength = searchString.length
        var increase = replacement.length - replLength
        increase = if (increase < 0) 0 else increase
        increase *= if (max < 0) 16 else if (max > 64) 64 else max
        val buf = StringBuilder(text.length + increase)
        while (end != INDEX_NOT_FOUND) {
```

```
            buf.append(text, start, end).append(replacement)
            start = end + replLength
            if (--max == 0) {
                break
            }
            end = searchText.indexOf(searchString, start)
        }
        buf.append(text, start, text.length)
        return buf.toString()
    }

    private fun isEmpty(cs: CharSequence?): Boolean {
        return cs == null || cs.isEmpty()
    }
}
```

RowView.kt

```
package my.project.db.manager

import android.content.Context
import android.text.TextUtils
import android.util.SparseArray
import android.view.LayoutInflater
import android.view.View
import android.widget.FrameLayout
import android.widget.LinearLayout
import android.widget.TextView
import my.project.db.R
import java.util.*

internal class RowView(context: Context?, columnCount: Int) :
    FrameLayout(context!!) {
    private val mColumnCount: Int
    private val mTextViewList: ArrayList<TextView>
    private val mRowViewContainer: LinearLayout
    private var mColumnIndexToValues: SparseArray<Any?>? = null
    private val textView: TextView
        private get() = LayoutInflater.from(context)
            .inflate(R.layout.column_text_item, this, false) as TextView

    fun setData(columnIndexToValues: SparseArray<Any?>) {
        require(columnIndexToValues.size() == mColumnCount) { "columnIndexValues
count doesn't match columnCount" }
        mColumnIndexToValues = columnIndexToValues
        for (i in 0 until mColumnIndexToValues!!.size()) {
            val columnEntry = mColumnIndexToValues!![i]
            if (columnEntry == null) {
                mTextViewList[i].text = "(NULL)"
            } else {
                if (columnEntry is String &&
TextUtils.isEmpty(columnEntry.toString())) {
                    mTextViewList[i].text = "(EMPTY)"
                } else if (columnEntry is ByteArray) {
                    mTextViewList[i].text = Arrays.toString(columnEntry as
ByteArray?)
                } else {
                    mTextViewList[i].text = columnEntry.toString()
                }
            }
        }
    }

    init {
        val inflater =
```

```
            getContext().getSystemService(Context.LAYOUT_INFLATER_SERVICE) as
LayoutInflater
        inflater.inflate(R.layout.selectable_horizontal_linear_layout, this,
true)
        mRowViewContainer =
            findViewById<View>(R.id.sqlite_manager_row_view_container) as
LinearLayout
        mColumnCount = columnCount
        mTextViewList = ArrayList(columnCount)
        for (i in 0 until columnCount) {
            val textView = textView
            mTextViewList.add(textView)
            mRowViewContainer.addView(textView)
        }
    }
}
```

SqliteActivity.java

```
package my.project.db.manager;

import android.content.DialogInterface;
import android.content.Intent;
import android.database.sqlite.SQLiteException;
import android.os.Bundle;
import android.util.SparseArray;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.StringRes;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.AppCompatSpinner;
import androidx.appcompat.widget.Toolbar;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.google.android.material.snackbar.Snackbar;
import com.google.android.material.textfield.TextInputEditText;
import com.google.android.material.textfield.TextInputLayout;
import com.jaredrummler.cyanea.app.CyaneaAppCompatActivity;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import my.project.db.CreateActivity;
import my.project.db.LauncherActivity;
import my.project.db.R;

public class SqliteActivity extends CyaneaAppCompatActivity implements
SqliteView, AdapterView.OnItemSelectedListener,
```

```
ColumnNameView.ColumnHeaderSortChangeListener, View.OnClickListener,
TableRecyclerAdapter.Listener {

    private SqlitePresenter mSqlitePresenter;
    private TableRecyclerAdapter mTableRecyclerAdapter;

    private View mSqliteManagerParent;
    private Toolbar mToolbar;

    private AppCompatSpinner mTableSelectionSpinner;

    private View mTableSelectionContainer;
    private View mCustomQueryContainer;
    private TextView mCustomQueryText;
    private ImageView mCustomQueryEdit;
    private ImageView mCustomQueryClear;

    private View mErrorLayout;
    private TextView mErrorLayoutText;

    private View mTableLayout;
    private ColumnNameView mColumnNameView;
    private RecyclerView mTableLayoutRecyclerView;

    private FloatingActionButton mSqliteAddFab;

    private View mActionCustomQuery;

    public static final String CSV_FILE_SHARE_AUTHORITY =
"csv_file_share_authority";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sqlite_manager);

        mSqlitePresenter = new
SqlitePresenter(SqliteStarter.mSqliteDataRetriever,
getIntent().getStringExtra(CSV_FILE_SHARE_AUTHORITY));

        mSqliteManagerParent = findViewById(R.id.sqlite_manager_parent);
        mTableSelectionSpinner =
findViewById(R.id.sqlite_manager_table_selection_spinner);
        mErrorLayout = findViewById(R.id.sqlite_manager_error_layout);
        mTableLayout = findViewById(R.id.sqlite_manager_table_layout);
        mColumnNameView = findViewById(R.id.sqlite_manager_table_layout_header);
        mErrorLayoutText = findViewById(R.id.sqlite_manager_error_layout_text);
        mTableLayoutRecyclerView =
findViewById(R.id.sqlite_manager_table_layout_recycler_view);
        mActionCustomQuery =
findViewById(R.id.sqlite_manager_action_custom_query);
        mSqliteAddFab = findViewById(R.id.sqlite_manager_add_fab);

        mTableSelectionContainer =
findViewById(R.id.sqlite_manager_table_selection_container);
        mCustomQueryContainer =
findViewById(R.id.sqlite_manager_custom_query_container);
        mCustomQueryText = findViewById(R.id.sqlite_manager_custom_query_text);
        mCustomQueryEdit = findViewById(R.id.sqlite_manager_custom_query_edit);
        mCustomQueryClear =
findViewById(R.id.sqlite_manager_custom_query_clear);
        mToolbar = findViewById(R.id.toolbar);
        setSupportActionBar(mToolbar);
        setTitle(getString(R.string.tables));
```

```
        mTableRecyclerAdapter = new TableRecyclerAdapter(null);
        mTableRecyclerAdapter.setListener(this);
        mTableLayoutRecyclerView.setLayoutManager(new
LinearLayoutManager(this));
        mTableLayoutRecyclerView.setAdapter(mTableRecyclerAdapter);
        mTableSelectionSpinner.setOnItemSelectedListener(this);
        mColumnNameView.setColumnHeaderSortChangeListener(this);
        mActionCustomQuery.setOnClickListener(this);
        mSqliteAddFab.setOnClickListener(this);
        mCustomQueryEdit.setOnClickListener(this);
        mCustomQueryClear.setOnClickListener(this);

        mSqlitePresenter.bindView(this);
    }

    @Override
    protected void onDestroy() {
        mSqlitePresenter.unBindView(this);
        super.onDestroy();
    }

    @Override
    public void setSubtitle(String subtitle) {
        mToolbar.setSubtitle(subtitle);
    }

    @NonNull
    @Override
    public AppCompatActivity getViewContext() {
        return this;
    }

    @Override
    public void finishActivityWithError(@StringRes int errorMessageId) {
        finishActivityWithError(getString(errorMessageId));
    }

    @Override
    public void finishActivityWithError(String error) {
        Toast.makeText(this, error, Toast.LENGTH_LONG).show();
        finishActivity();
    }

    @Override
    public void finishActivity() {
        SqliteStarter.clearReferences();
        finish();
    }

    @Override
    public void displayError(@StringRes int errorMessageId) {
        displayError(getString(errorMessageId));
    }

    @Override
    public void displayError(String error) {
        mErrorLayout.setVisibility(View.VISIBLE);
        mTableLayout.setVisibility(View.GONE);
        mErrorLayoutText.setText(error);
    }

    @Override
    public void informErrorToUser(@StringRes int errorMessageId) {
```

```
        informErrorToUser(getString(errorMessageId));
    }

    @Override
    public void informErrorToUser(String error) {
        Snackbar.make(mSqliteManagerParent, error, Snackbar.LENGTH_LONG).show();
    }

    @Override
    public void setAddFABVisible(boolean visible) {
        if (visible) {
            mSqliteAddFab.show();
        } else {
            mSqliteAddFab.hide();
        }
    }

    @Override
    public void showCustomQueryView(String customQuery) {
        mTableSelectionContainer.setVisibility(View.GONE);
        mCustomQueryContainer.setVisibility(View.VISIBLE);

        mCustomQueryText.setText(customQuery);
    }

    @Override
    public void showTableSelectionView() {
        mTableSelectionContainer.setVisibility(View.VISIBLE);
        mCustomQueryContainer.setVisibility(View.GONE);
    }

    @Override
    public void showCustomQueryDialog(String previousCustomQuery) {
        LayoutInflater inflater = this.getLayoutInflater();
        final View dialogView = inflater.inflate(R.layout.custom_query_dialog,
null);

        final TextInputEditText customQueryEditText =
dialogView.findViewById(R.id.sqlite_manager_custom_query_edit_text);
        customQueryEditText.setText(previousCustomQuery);

        Button cancelButton = dialogView.findViewById(R.id.cancel);
        Button queryButton = dialogView.findViewById(R.id.query);

        final AlertDialog alertDialog
                = new AlertDialog.Builder(this)
                .setView(dialogView)
                .setTitle(R.string.custom_query)
                .setMessage(R.string.custom_query_hint)
                .create();

        cancelButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                alertDialog.dismiss();
            }
        });

        queryButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

mSqlitePresenter.onCustomQuerySubmitted(customQueryEditText.getText().toString()
);
```

```
                alertDialog.dismiss();
            }
        });

        alertDialog.show();
    }

    @Override
    public void showAddEditRowDialog(final boolean isEdit, final String
tableName, final String[] tableColumnNames, final SparseArray<Object>
oldColumnValues) {
        LayoutInflater inflater = this.getLayoutInflater();
        SqliteHelper sqliteHelper = new SqliteHelper(this);
        final ArrayList<TextInputEditText> editTextViews = new
ArrayList<>(tableColumnNames.length);

        final View dialogView =
inflater.inflate(R.layout.add_edit_dialog_container, null);

        LinearLayout linearLayout =
dialogView.findViewById(R.id.sqlite_manager_add_edit_dialog_container);
        Button deleteButton =
dialogView.findViewById(R.id.sqlite_manager_add_edit_dialog_delete);
        Button cancelButton =
dialogView.findViewById(R.id.sqlite_manager_add_edit_dialog_cancel);
        Button updateButton =
dialogView.findViewById(R.id.sqlite_manager_add_edit_dialog_update);

        for (int i = 0; i < tableColumnNames.length; i++) {
            View columnView = inflater.inflate(R.layout.add_edit_dialog_item,
null);
            ((TextInputLayout)
columnView.findViewById(R.id.sqlite_manager_add_edit_dialog_text_input_layout)).
setHint(tableColumnNames[i]);
            TextInputEditText currentInputEditText =
columnView.findViewById(R.id.sqlite_manager_add_edit_dialog_edit_text);
            editTextViews.add(currentInputEditText);
            linearLayout.addView(columnView);
            if (oldColumnValues != null && oldColumnValues.get(i) != null) {
                if (oldColumnValues.get(i) instanceof byte[]) {
                    currentInputEditText.setText(Arrays.toString((byte[])
(oldColumnValues.get(i))));
                } else {

currentInputEditText.setText(oldColumnValues.get(i).toString());
                }
            }
        }

        if (!isEdit) {
            deleteButton.setVisibility(View.GONE);
            updateButton.setText(R.string.add);
        }

        final AlertDialog alertDialog
                = new AlertDialog.Builder(this)
                .setView(dialogView)
                .setTitle(isEdit ? R.string.update_row_dialog_title :
R.string.add_row_dialog_title)
                .setMessage(getString(isEdit ?
R.string.update_row_dialog_message : R.string.add_row_dialog_message,
tableName))
                .create();
```

```
        updateButton.setOnClickListener(v -> {
            final ArrayList<String> columnValues = new
ArrayList<>(tableColumnNames.length);
            for (TextInputEditText currentEditText : editTextViews) {
                columnValues.add(currentEditText.getText().toString());
            }
            for (String x : columnValues) {
                if (!new SqlSafeUtil().isSqlInjectionSafe(x)) {
                    Toast.makeText(SqliteActivity.this,
getString(R.string.safe), Toast.LENGTH_LONG).show();
                    return;
                }
            }
            if (!sqliteHelper.isTypesCorrect(tableName, tableColumnNames,
columnValues)) {
                Toast.makeText(SqliteActivity.this, getString(R.string.safe),
Toast.LENGTH_LONG).show();
                return;
            }

            if (isEdit) {
                mSqlitePresenter.updateRow(tableName, tableColumnNames,
oldColumnValues, columnValues);
            } else {
                mSqlitePresenter.addRow(tableName, tableColumnNames,
columnValues);
            }

            alertDialog.dismiss();
        });

        cancelButton.setOnClickListener(v -> alertDialog.dismiss());

        deleteButton.setOnClickListener(v -> {
            mSqlitePresenter.deleteRow(tableName, tableColumnNames,
oldColumnValues);
            alertDialog.dismiss();
        });

        alertDialog.show();
    }

    @Override
    public void setSpinnerAdapter(ArrayList<String> tableNames) {
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, tableNames);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        mTableSelectionSpinner.setAdapter(adapter);
    }

    @Override
    public void showContentView() {
        mErrorLayout.setVisibility(View.GONE);
        mTableLayout.setVisibility(View.VISIBLE);
    }

    @Override
    public void updateColumnNames(String[] columnNames) {
        mColumnNameView.setData(columnNames);
    }

    @Override
    public void displayRows(List<SparseArray<Object>> columnIndexToValuesArray)
```

```
{
        mTableRecyclerAdapter.setData(columnIndexToValuesArray);
    }

    @Override
    public void onHeaderColumnSortChanged(String columnName, boolean
isAscendingOrder) {
        String selectedTableName =
mTableSelectionSpinner.getSelectedItem().toString();
        mSqlitePresenter.onSortChangedOrderChanged(selectedTableName,
columnName, isAscendingOrder);
    }

    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position,
long id) {
        String selectedTableName =
mTableSelectionSpinner.getSelectedItem().toString();
        mSqlitePresenter.fetchTableData(selectedTableName);
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {

    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.sqlite_manager_action_custom_query) {
            mSqlitePresenter.onCustomQueryClicked();
        } else if (v.getId() == R.id.sqlite_manager_add_fab) {
            try {
                String selectedTableName =
mTableSelectionSpinner.getSelectedItem().toString();
                mSqlitePresenter.onAddFabClicked(selectedTableName,
mColumnNameView.getTableColumnNames());
            } catch (NullPointerException e) {

            }
        } else if (v.getId() == R.id.sqlite_manager_custom_query_edit) {
            mSqlitePresenter.onCustomQueryClicked();
        } else if (v.getId() == R.id.sqlite_manager_custom_query_clear) {
            String selectedTableName =
mTableSelectionSpinner.getSelectedItem().toString();
            mSqlitePresenter.fetchTableData(selectedTableName);
            recreate();
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.table_menu, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (item.getItemId() == R.id.action_query_refresh) {
            Object selectedTableName = mTableSelectionSpinner.getSelectedItem();
            if (selectedTableName == null) {
                return super.onOptionsItemSelected(item);
            }
            mSqlitePresenter.onRefreshClicked(selectedTableName.toString());
```

```
            return true;
        }
        if (item.getItemId() == R.id.action_add_table) {
            startActivity(new Intent(SqliteActivity.this,
CreateActivity.class));
            finish();
            return true;
        }
        if (item.getItemId() == R.id.action_export_result_as_csv) {
            Object selectedTableName = mTableSelectionSpinner.getSelectedItem();
            if (selectedTableName == null) {
                return super.onOptionsItemSelected(item);
            }

mSqlitePresenter.onExportResultAsCSVClicked(selectedTableName.toString());
            return true;
        }
        if (item.getItemId() == R.id.action_export_result_as_json) {
            Object selectedTableName = mTableSelectionSpinner.getSelectedItem();
            if (selectedTableName == null) {
                return super.onOptionsItemSelected(item);
            }

mSqlitePresenter.onExportResultAsJsonClicked(selectedTableName.toString());
            return true;
        }
        if (item.getItemId() == R.id.action_drop_table) {
            Object selectedTableName = mTableSelectionSpinner.getSelectedItem();
            if (selectedTableName == null) {
                return super.onOptionsItemSelected(item);
            }
            deleteTableDialog(selectedTableName.toString());
            return true;
        }
        if (item.getItemId() == R.id.back) {
            onBackPressed();
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

    private void deleteTableDialog(String name) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder
                .setTitle(getString(R.string.delete_table_sure))
                .setCancelable(false)
                .setPositiveButton(getString(R.string.yes), new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        try {
                            dialog.cancel();
                            mSqlitePresenter.deleteTable(name);
                            recreate();
                        } catch (SQLiteException e) {
                            Toast.makeText(SqliteActivity.this,
getString(R.string.table_not_exist), Toast.LENGTH_LONG).show();
                        }
                    }
                })
                .setNegativeButton(getString(R.string.no), new
DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        dialog.cancel();
                    }
```

```
                }).show();
    }

    @Override
    public void onColumnValueClicked(SparseArray<Object> columnValues) {
        String selectedTableName =
mTableSelectionSpinner.getSelectedItem().toString();
        mSqlitePresenter.onColumnValueClicked(selectedTableName,
mColumnNameView.getTableColumnNames(), columnValues);
    }

    @Override
    public void onBackPressed() {
        startActivity(new Intent(SqliteActivity.this, LauncherActivity.class));
        finish();
    }
}
```

SqliteDataRetriever.java

```
package my.project.db.manager;

import android.database.Cursor;

import androidx.annotation.NonNull;

public interface SqliteDataRetriever {

    Cursor rawQuery(@NonNull String query, String[] selectionArgs);

    String getDatabaseName();

    @Deprecated
    void freeResources();
}
```

SqliteHelper.kt

```
package my.project.db.manager

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import android.os.AsyncTask

class SqliteHelper(context: Context?) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {
    override fun onCreate(db: SQLiteDatabase) {
        createTables(db)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int)
{}
    fun clearTables() {
        val db = this.writableDatabase
        dropTables(db)
    }

    fun dropTables(db: SQLiteDatabase?) {}
    fun isTypesCorrect(
        tableName: String,
        columnNames: Array<String?>,
        values: ArrayList<String?>
```

```
    ): Boolean {
        val db = this.readableDatabase
        val query = "PRAGMA table_info($tableName)"
        loop@ for (i in columnNames.indices) {
            val cursor = db.rawQuery(query, null)
            cursor.moveToPosition(i)
            println(cursor.getString(cursor.getColumnIndex("type")))
            when (cursor.getString(cursor.getColumnIndex("type"))) {
                "TEXT" -> {}
                "INTEGER" -> {
                    return if (isInteger(values[i])) {
                        break@loop
                    } else {
                        false
                    }
                }
                "REAL" -> {
                    return if (isFloat(values[i])) {
                        break@loop
                    } else {
                        false
                    }
                }
            }
            cursor.close()
        }
        return true
    }

    private fun isFloat(s: String?): Boolean {
        if (s == null) return true
        try {
            s.toFloat()
        } catch (e: NumberFormatException) {
            return false
        }
        return true
    }

    private fun isInteger(s: String?): Boolean {
        if (s == null) return true
        for (i in 0 until s.length) {
            if (i == 0 && s[i] == '-') {
                return if (s.length == 1) false else continue
            }
            if (!Character.isDigit(s[i])) return false
        }
        return true
    }

    val tableNames: ArrayList<String>
        get() {
            val sqLiteDatabase = this.readableDatabase
            val arrTblNames = ArrayList<String>()
            val cursor =
                sqLiteDatabase.rawQuery("SELECT name FROM sqlite_master WHERE
type='table'", null)
            if (cursor.moveToFirst()) {
                while (!cursor.isAfterLast) {

arrTblNames.add(cursor.getString(cursor.getColumnIndex("name")))
                    cursor.moveToNext()
                }
            }
```

```
            cursor.close()
            sqLiteDatabase.close()
            return arrTblNames
        }

    fun createTables(db: SQLiteDatabase?) {}
    fun createTableDataAsync(
        name: String?,
        columnNames: ArrayList<String?>,
        types: ArrayList<String?>
    ) {
        val task = CreateDataAsync(this, name, columnNames, types)
        task.execute()
    }

    fun createTable(name: String?, columnNames: ArrayList<String?>, types:
ArrayList<String?>) {
        val sqLiteDatabase = this.writableDatabase
        val creation = StringBuilder("")
        creation.append("CREATE TABLE ").append(name).append(" (unique_id
INTEGER PRIMARY KEY, ")
        for (i in columnNames.indices) {
            creation.append(columnNames[i])
            creation.append(addType(types[i]))
        }
        creation.setLength(creation.length - 2)
        creation.append(")")
        sqLiteDatabase.execSQL(creation.toString())
        sqLiteDatabase.close()
    }

    private fun addType(type: String?): String {
        return when (type) {
            "text" -> " TEXT, "
            "integer" -> " INTEGER, "
            "real" -> " REAL, "
            else -> ""
        }
    }

    private class CreateDataAsync internal constructor(
        private val mDb: SqliteHelper,
        private val name: String?,
        private val columnNames: ArrayList<String?>,
        private val types: ArrayList<String?>
    ) :
        AsyncTask<Void?, Void?, Void?>() {

        override fun doInBackground(vararg params: Void?): Void? {
            mDb.createTable(name, columnNames, types)
            return null
        }
    }

    companion object {
        private const val DATABASE_VERSION = 2

        private const val DATABASE_NAME = "PocketDB"
    }
}
```

SqlitePresenter.kt

```
package my.project.db.manager

import android.database.Cursor
import android.text.TextUtils
import android.util.SparseArray
import androidx.appcompat.app.AppCompatActivity
import my.project.db.R
import java.util.*

internal class SqlitePresenter(
    private val mSqliteDataRetriever: SqliteDataRetriever,
    private val mFileShareAuthority: String
) {
    private val mSqliteResponseRetriever: SqliteResponseRetriever
    var view: SqliteView? = null
        private set
    private val mTableNames = ArrayList<String>()
    private var mIsCustomQuery
            = false
    private var mCurrentsqliteResponseData
            : SqliteResponseData? = null
    private var mPreviousCustomQuery: String? = null
    fun bindView(sqliteView: SqliteView?) {
        view = sqliteView
        changeSubTitle()
        loadTableNames()
    }

    fun unBindView(sqliteView: SqliteView?) {
        view = null
    }

    private fun changeSubTitle() {
        val databaseName = mSqliteDataRetriever.databaseName
        val appName = getApplicationName(view!!.viewContext)
        view!!.setSubtitle((if (TextUtils.isEmpty(databaseName)) "" else
"$databaseName ") + appName)
    }

    private fun loadTableNames() {
        mTableNames.clear()
        val sqliteResponse = mSqliteResponseRetriever.getData(TABLE_FETCH_QUERY,
null)
        if (sqliteResponse.isQuerySuccess) {
            val cursor = sqliteResponse.cursor
            cursor!!.moveToFirst()
            try {
                do {
                    mTableNames.add(cursor.getString(0))
                } while (cursor.moveToNext())
            } catch (e: Exception) {
                view!!.displayError(R.string.database_has_no_tables)
                return
            }
            cursor.close()
        } else {
            if (sqliteResponse.throwable != null) {

view!!.finishActivityWithError(sqliteResponse.throwable!!.message)
            } else {

view!!.finishActivityWithError(R.string.error_while_fetching_table_names)
            }
        }
```

```
            mSqliteDataRetriever.freeResources()
            if (mTableNames.isEmpty()) {
                view!!.displayError(R.string.database_has_no_tables)
            } else {
                mTableNames.remove("android_metadata")
                view!!.setSpinnerAdapter(mTableNames)
            }
        }

        fun fetchTableData(selectedTableName: String?) {
            if (selectedTableName == null || TextUtils.isEmpty(selectedTableName)) {
                view!!.displayError(R.string.please_select_a_table)
                return
            }
            val sqliteResponseData =
                getSqliteResponseDataForSelectedTable(selectedTableName, null, true)
            displayData(sqliteResponseData, false, true)
        }

        fun onRefreshClicked(selectedTableName: String?) {
            if (mIsCustomQuery) {
                val sqliteResponseData =
                    getSqliteResponseDataForCustomQuery(mPreviousCustomQuery!!,
null, true)
                displayData(sqliteResponseData, true, true)
            } else {
                if (selectedTableName == null ||
TextUtils.isEmpty(selectedTableName)) {
                    view!!.informErrorToUser(R.string.please_select_a_table)
                    return
                }
                val sqliteResponseData =
                    getSqliteResponseDataForSelectedTable(selectedTableName, null,
true)
                displayData(sqliteResponseData, false, true)
            }
        }

        fun onExportResultAsJsonClicked(selectedTableName: String?) {
            SqliteUtils.shareSqliteResponseDataAsJsonFile(
                view!!.viewContext,
                mCurrentsqliteResponseData!!,
                selectedTableName,
                mFileShareAuthority
            )
        }

        fun onExportResultAsCSVClicked(selectedTableName: String?) {
            SqliteUtils.shareSqliteResponseDataAsCsvFile(
                view!!.viewContext,
                mCurrentsqliteResponseData!!,
                selectedTableName,
                mFileShareAuthority
            )
        }

        fun onColumnValueClicked(
            tableName: String?,
            tableColumnNames: Array<String?>?,
            columnValues: SparseArray<Any?>?
        ) {
            if (tableName == null || TextUtils.isEmpty(tableName)) {
                view!!.informErrorToUser(R.string.please_select_a_table)
                return
```

```
        }
        view!!.showAddEditRowDialog(true, tableName, tableColumnNames,
columnValues)
    }

    fun onAddFabClicked(tableName: String?, tableColumnNames: Array<String?>?) {
        if (tableName == null || TextUtils.isEmpty(tableName)) {
            view!!.informErrorToUser(R.string.please_select_a_table)
            return
        }
        view!!.showAddEditRowDialog(false, tableName, tableColumnNames, null)
    }

    fun onSortChangedOrderChanged(
        selectedTableName: String?,
        orderBy: String?,
        isAscendingOrder: Boolean
    ) {
        if (mIsCustomQuery) {
            if (mPreviousCustomQuery!!.toUpperCase().contains("ORDER BY")) {

view!!.informErrorToUser(R.string.custom_query_contacins_order_by)
                onRefreshClicked(selectedTableName)
                return
            } else {
                val sqliteResponseData = getSqliteResponseDataForCustomQuery(
                    mPreviousCustomQuery!!,
                    orderBy,
                    isAscendingOrder
                )
                displayData(sqliteResponseData, true, false)
            }
        } else {
            if (selectedTableName == null ||
TextUtils.isEmpty(selectedTableName)) {
                view!!.informErrorToUser(R.string.please_select_a_table)
                return
            }
            val sqliteResponseData =
                getSqliteResponseDataForSelectedTable(selectedTableName,
orderBy, isAscendingOrder)
            displayData(sqliteResponseData, false, false)
        }
    }

    fun onCustomQueryClicked() {
        view!!.showCustomQueryDialog(mPreviousCustomQuery)
    }

    fun onCustomQuerySubmitted(customQuery: String) {
        if (TextUtils.isEmpty(customQuery)) {
            view!!.informErrorToUser(R.string.empty_custom_query)
            return
        }
        val sqliteResponseData =
getSqliteResponseDataForCustomQuery(customQuery, null, true)
        mPreviousCustomQuery = customQuery
        displayData(sqliteResponseData, true, true)
    }

    fun addRow(
        tableName: String,
        tableColumnNames: Array<String>,
        columnValues: ArrayList<String>
```

```
    ) {
        if (tableColumnNames.size != columnValues.size) {
            view!!.informErrorToUser(R.string.table_column_length_error)
            return
        }
        var columnNameQuery = "("
        var columnValueQuery = "("
        var columnIndex = 0
        for (currentColumnValue in columnValues) {
            val columnName = tableColumnNames[columnIndex]
            columnIndex++
            if (!TextUtils.isEmpty(currentColumnValue)) {
                columnNameQuery = "$columnNameQuery$columnName,"
                columnValueQuery = "$columnValueQuery\'$currentColumnValue\',"
            }
        }
        if (columnNameQuery.endsWith(",")) {
            columnNameQuery = columnNameQuery.substring(0,
columnNameQuery.length - 1) + ")"
            columnValueQuery = columnValueQuery.substring(0,
columnValueQuery.length - 1) + ")"
            val query =
                "INSERT INTO $tableName $columnNameQuery VALUES
$columnValueQuery"
            val sqliteResponseData = getSqliteResponseDataFromQuery(query, null)
            if (sqliteResponseData.isQuerySuccess) {
                view!!.informErrorToUser(R.string.add_row_success)
                onRefreshClicked(tableName)
            } else {
                if (sqliteResponseData.throwable != null) {

view!!.informErrorToUser(sqliteResponseData.throwable.message)
                } else {
                    view!!.informErrorToUser(R.string.add_row_error)
                }
            }
        } else {
            view!!.informErrorToUser(R.string.all_columns_cant_be_empty)
            return
        }
    }

    fun deleteTable(name: String) {
        val query = "DROP TABLE $name"
        val sqliteResponseData = getSqliteResponseDataFromQuery(query, null)
    }

    fun deleteRow(
        tableName: String,
        tableColumnNames: Array<String>,
        oldColumnValues: SparseArray<Any?>
    ) {
        if (tableColumnNames.size != oldColumnValues.size()) {
            view!!.informErrorToUser(R.string.table_column_length_error)
            return
        }
        val whereCondition = getWhereCondition(tableColumnNames,
oldColumnValues)
        if (whereCondition != null) {
            val query = "DELETE FROM $tableName WHERE $whereCondition"
            val sqliteResponseData = getSqliteResponseDataFromQuery(query, null)
            if (sqliteResponseData.isQuerySuccess) {
                view!!.informErrorToUser(R.string.delete_row_success)
                onRefreshClicked(tableName)
```

```
            } else {
                if (sqliteResponseData.throwable != null) {

view!!.informErrorToUser(sqliteResponseData.throwable.message)
                } else {
                    view!!.informErrorToUser(R.string.delete_row_error)
                }
            }
        } else {
            view!!.informErrorToUser(R.string.all_columns_cant_be_empty)
            return
        }
    }

    fun updateRow(
        tableName: String,
        tableColumnNames: Array<String>,
        oldColumnValues: SparseArray<Any?>,
        columnValues: ArrayList<String>
    ) {
        if (tableColumnNames.size != oldColumnValues.size() ||
tableColumnNames.size != columnValues.size) {
            view!!.informErrorToUser(R.string.table_column_length_error)
            return
        }
        val whereCondition = getWhereCondition(tableColumnNames,
oldColumnValues)
        var updateQuery = ""
        var columnIndex = 0
        for (currentColumnValue in columnValues) {
            val columnName = tableColumnNames[columnIndex]
            columnIndex++
            if (!TextUtils.isEmpty(currentColumnValue)) {
                val oldColumnValue = oldColumnValues[columnIndex]
                if (oldColumnValue is ByteArray) {
                    if (!Arrays.toString(oldColumnValue as ByteArray?)
                            .equals(currentColumnValue, ignoreCase = true)
                    ) {
                        view!!.informErrorToUser(R.string.blob_not_supported)
                        return
                    }
                } else {
                    updateQuery = "$updateQuery$columnName =
\'$currentColumnValue\', "
                }
            }
        }
        if (!TextUtils.isEmpty(updateQuery) && whereCondition != null) {
            updateQuery = updateQuery.substring(0, updateQuery.length - 2) + " "
            val query = "UPDATE $tableName SET $updateQuery WHERE
$whereCondition"
            val sqliteResponseData = getSqliteResponseDataFromQuery(query, null)
            if (sqliteResponseData.isQuerySuccess) {
                view!!.informErrorToUser(R.string.update_row_success)
                onRefreshClicked(tableName)
            } else {
                if (sqliteResponseData.throwable != null) {

view!!.informErrorToUser(sqliteResponseData.throwable.message)
                } else {
                    view!!.informErrorToUser(R.string.update_row_error)
                }
            }
        } else {
```

```
            view!!.informErrorToUser(R.string.all_columns_cant_be_empty)
            return
        }
    }

    private fun getWhereCondition(
        tableColumnNames: Array<String>,
        columnValues: SparseArray<Any?>
    ): String? {
        var whereCondition = ""
        var columnIndex = 0
        for (currentColumnName in tableColumnNames) {
            val columnValue = columnValues[columnIndex]
            columnIndex++
            if (columnValue != null && columnValue !is ByteArray) {
                whereCondition =
"$whereCondition$currentColumnName=\'$columnValue\' AND "
            }
        }
        return if (whereCondition.endsWith(" AND ")) {
            whereCondition.substring(0, whereCondition.length - 5)
        } else {
            null
        }
    }

    private fun displayData(
        sqliteResponseData: SqliteResponseData,
        isCustomQuery: Boolean,
        updateColumnNames: Boolean
    ) {
        mCurrentsqliteResponseData = sqliteResponseData
        if (sqliteResponseData.isQuerySuccess) {
            view!!.showContentView()
            mIsCustomQuery = isCustomQuery
            if (updateColumnNames) {
                view!!.updateColumnNames(sqliteResponseData.tableColumnNames)
            }
            view!!.displayRows(sqliteResponseData.columnIndexToValuesArray)
        } else {
            if (sqliteResponseData.throwable != null) {
                if (isCustomQuery) {

view!!.informErrorToUser(sqliteResponseData.throwable.message)
                } else {
                    view!!.displayError(sqliteResponseData.throwable.message)
                }
            } else {
                if (isCustomQuery) {
                    view!!.informErrorToUser(R.string.error_while_fetching_data)
                } else {
                    view!!.displayError(R.string.error_while_fetching_data)
                }
            }
        }
        if (mIsCustomQuery) {
            view!!.showCustomQueryView(mPreviousCustomQuery)
            view!!.setAddFABVisible(false)
        } else {
            view!!.showTableSelectionView()
            view!!.setAddFABVisible(true)
        }
    }
```

```
    private fun getSqliteResponseDataForCustomQuery(
        customQuery: String,
        orderBy: String?,
        isAscendingOrder: Boolean
    ): SqliteResponseData {
        val orderByWithIncrement =
            if (orderBy == null) null else orderBy + if (isAscendingOrder) ""
else " DESC"
        val customQueryWithIncrement =
            customQuery + if (orderByWithIncrement == null) "" else " ORDER BY
$orderByWithIncrement"
        return getSqliteResponseDataFromQuery(customQueryWithIncrement, null)
    }

    private fun getSqliteResponseDataForSelectedTable(
        selectedTableName: String,
        orderBy: String?,
        isAscendingOrder: Boolean
    ): SqliteResponseData {
        val orderByWithIncrement =
            if (orderBy == null) null else orderBy + if (isAscendingOrder) ""
else " DESC"
        val query =
            "SELECT * FROM " + selectedTableName + if (orderByWithIncrement ==
null) "" else " ORDER BY $orderByWithIncrement"
        return getSqliteResponseDataFromQuery(query, null)
    }

    private fun getSqliteResponseDataFromQuery(
        query: String,
        selectionArgs: Array<String>?
    ): SqliteResponseData {
        val sqliteResponse = mSqliteResponseRetriever.getData(query,
selectionArgs)
        return if (sqliteResponse.isQuerySuccess) {
            try {
                val cursor = sqliteResponse.cursor
                val selectedTableColumnNames = cursor!!.columnNames
                val selectedTableColumnTypes =
IntArray(selectedTableColumnNames.size)
                val columnCount = cursor.columnCount
                val valuesArray: MutableList<SparseArray<Any>> =
ArrayList(cursor.count)
                if (cursor.moveToFirst()) {
                    do {
                        val columnValuePair = SparseArray<Any>(columnCount)
                        for (columnIndex in 0 until columnCount) {
                            val fieldType = cursor.getType(columnIndex)
                            selectedTableColumnTypes[columnIndex] = fieldType
                            if (fieldType == Cursor.FIELD_TYPE_NULL) {
                                columnValuePair.put(columnIndex,
cursor.getString(columnIndex))
                            } else if (fieldType == Cursor.FIELD_TYPE_INTEGER) {
                                columnValuePair.put(columnIndex,
cursor.getInt(columnIndex))
                            } else if (fieldType == Cursor.FIELD_TYPE_FLOAT) {
                                columnValuePair.put(columnIndex,
cursor.getFloat(columnIndex))
                            } else if (fieldType == Cursor.FIELD_TYPE_STRING) {
                                columnValuePair.put(columnIndex,
cursor.getString(columnIndex))
                            } else if (fieldType == Cursor.FIELD_TYPE_BLOB) {
                                columnValuePair.put(columnIndex,
cursor.getBlob(columnIndex))
```

```
                            } else {
                                columnValuePair.put(columnIndex,
cursor.getString(columnIndex))
                            }
                        }
                        valuesArray.add(columnValuePair)
                    } while (cursor.moveToNext())
                }
                SqliteResponseData(
                    columnCount,
                    selectedTableColumnNames,
                    selectedTableColumnTypes,
                    valuesArray
                )
            } catch (exception: Exception) {
                SqliteResponseData(exception)
            } finally {
                mSqliteDataRetriever.freeResources()
            }
        } else {
            mSqliteDataRetriever.freeResources()
            SqliteResponseData(sqliteResponse.throwable)
        }
    }

    private fun getApplicationName(appCompatActivity: AppCompatActivity): String
{
        val applicationInfo = appCompatActivity.applicationInfo
        val stringId = applicationInfo.labelRes
        return if (stringId == 0) applicationInfo.nonLocalizedLabel.toString()
else appCompatActivity.getString(
            stringId
        )
    }

    companion object {
        private const val TABLE_FETCH_QUERY =
            "SELECT name _id FROM sqlite_master WHERE type ='table'"
    }

    init {
        mSqliteResponseRetriever = SqliteResponseRetriever(mSqliteDataRetriever)
    }
}
```

SqliteResponse.kt

```
package my.project.db.manager

import android.database.Cursor

internal class SqliteResponse {
    var cursor: Cursor? = null
    var throwable: Throwable? = null
    var isQuerySuccess = false
}
```

SqliteResponseData.kt

```
package my.project.db.manager
```

```
import android.util.SparseArray

internal class SqliteResponseData {
    val columnCount: Int
    val tableColumnNames: Array<String>?
    private val mTableColumnTypes: IntArray?
    val columnIndexToValuesArray: List<SparseArray<Any>>?
    val throwable: Throwable?
    val isQuerySuccess: Boolean

    constructor(
        columnCount: Int,
        tableColumnNames: Array<String>?,
        tableColumnTypes: IntArray?,
        columnIndexToValuesArray: List<SparseArray<Any>>?
    ) {
        this.tableColumnNames = tableColumnNames
        mTableColumnTypes = tableColumnTypes
        this.columnCount = columnCount
        this.columnIndexToValuesArray = columnIndexToValuesArray
        isQuerySuccess = true
        throwable = null
    }

    constructor(throwable: Throwable?) {
        columnCount = -1
        tableColumnNames = null
        mTableColumnTypes = null
        columnIndexToValuesArray = null
        this.throwable = throwable
        isQuerySuccess = false
    }
}
```

SqliteResponseRetriever.kt

```
package my.project.db.manager

internal class SqliteResponseRetriever(private val mSqliteDataRetriever:
SqliteDataRetriever) {
    fun getData(query: String, selectionArgs: Array<String>?): SqliteResponse {
        val sqliteResponse = SqliteResponse()
        try {
            val cursor = mSqliteDataRetriever.rawQuery(query, selectionArgs)
            sqliteResponse.cursor = cursor
            sqliteResponse.isQuerySuccess = true
        } catch (throwable: Throwable) {
            sqliteResponse.throwable = throwable
            sqliteResponse.isQuerySuccess = false
        }
        return sqliteResponse
    }
}
```

SqliteStarter.java

```
package my.project.db.manager;

import android.content.Context;
import android.content.Intent;

public class SqliteStarter {
```

```
    static SqliteDataRetriever mSqliteDataRetriever;

    public static void launchSqliteManager(Context context, SqliteDataRetriever
sqliteDataRetriever, String fileShareAuthority) {
        mSqliteDataRetriever = sqliteDataRetriever;
        Intent sqliteManagerIntent = new Intent(context, SqliteActivity.class);
        if (fileShareAuthority != null && fileShareAuthority.trim().length() >
0) {

sqliteManagerIntent.putExtra(SqliteActivity.CSV_FILE_SHARE_AUTHORITY,
fileShareAuthority);
        }

context.startActivity(sqliteManagerIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
));
    }

    static void clearReferences() {
        mSqliteDataRetriever = null;
    }
}
```

SqliteUtils.kt

```
package my.project.db.manager

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ShareCompat
import androidx.core.content.FileProvider
import my.project.db.manager.csv.CsvEscaper
import org.json.JSONArray
import org.json.JSONException
import org.json.JSONObject
import java.io.File
import java.io.FileOutputStream
import java.util.*

internal object SqliteUtils {
    fun shareSqliteResponseDataAsJsonFile(
        context: AppCompatActivity,
        sqliteResponseData: SqliteResponseData,
        selectedTableName: String?,
        fileShareAuthority: String?
    ) {
        if (fileShareAuthority == null || fileShareAuthority.trim { it <= ' '
}.isEmpty()) {
            return
        }
        val toBeWrittenString =
getSqliteResponseDataJsonString(sqliteResponseData)
        val fileName =
            "json_export" + (if (selectedTableName == null) "" else
"_$selectedTableName") + ".txt"
        createTempFileAndShare(context, fileShareAuthority, toBeWrittenString,
fileName, "text/*")
    }

    private fun getSqliteResponseDataJsonString(sqliteResponseData:
SqliteResponseData): String {
        return try {
            getSqliteResponseDataJson(sqliteResponseData).toString(4)
        } catch (e: JSONException) {
```

```
                ""
            }
        }

        private fun getSqliteResponseDataJson(sqliteResponseData:
SqliteResponseData): JSONArray {
            val resultSet = JSONArray()
            if (!sqliteResponseData.isQuerySuccess) {
                return resultSet
            }
            for (columnIndexToValues in
sqliteResponseData.columnIndexToValuesArray!!) {
                val rowObject = JSONObject()
                for (i in 0 until columnIndexToValues.size()) {
                    val columnEntry = columnIndexToValues[i]
                    try {
                        if (columnEntry == null) {
                            rowObject.put(sqliteResponseData.tableColumnNames!![i],
"")
                        } else if (columnEntry is ByteArray) {
                            rowObject.put(
                                sqliteResponseData.tableColumnNames!![i],
Arrays.toString(
                                    columnEntry
                                )
                            )
                        } else {
                            rowObject.put(sqliteResponseData.tableColumnNames!![i],
columnEntry)
                        }
                    } catch (e: JSONException) {
//                        e.printStackTrace();
                    }
                }
                resultSet.put(rowObject)
            }
            return resultSet
        }

        fun shareSqliteResponseDataAsCsvFile(
            context: AppCompatActivity,
            sqliteResponseData: SqliteResponseData,
            selectedTableName: String?,
            FileShareAuthority: String?
        ) {
            if (!sqliteResponseData.isQuerySuccess || FileShareAuthority == null ||
FileShareAuthority.trim { it <= ' ' }
                    .isEmpty()) {
                return
            }
            val csvString = StringBuilder()
            var firstElement = true
            for (columnName in sqliteResponseData.tableColumnNames!!) {
                if (!firstElement) {
                    csvString.append(",")
                }
                firstElement = false
                csvString.append("\"").append(columnName).append("\"")
            }
            csvString.append("\n")
            for (columnIndexToValues in
sqliteResponseData.columnIndexToValuesArray!!) {
                firstElement = true
                for (i in 0 until columnIndexToValues.size()) {
```

```
                if (!firstElement) {
                    csvString.append(",")
                }
                firstElement = false
                val columnEntry = columnIndexToValues[i]
                if (columnEntry != null) {
                    if (columnEntry is String) {
                        csvString.append(escapeCsv(columnEntry))
                    } else if (columnEntry is ByteArray) {
                        csvString.append(
                            escapeCsv(
                                Arrays.toString(
                                    columnEntry
                                )
                            )
                        )
                    } else {
                        csvString.append(columnEntry)
                    }
                }
            }
            csvString.append("\n")
        }
        val fileName =
            "sqlite_export" + (if (selectedTableName == null) "" else
"_$selectedTableName") + ".csv"
        createTempFileAndShare(
            context,
            FileShareAuthority,
            csvString.toString(),
            fileName,
            "text/csv"
        )
    }

    private fun createTempFileAndShare(
        context: AppCompatActivity,
        fileShareAuthority: String,
        csvString: String,
        csvFileName: String,
        type: String
    ) {
        try {
            val fileDir = File(context.filesDir, "sqlite")
            fileDir.mkdir()
            val file = File(fileDir, csvFileName)
            file.createNewFile()
            val fileOutputStream = FileOutputStream(file)
            fileOutputStream.write(csvString.toByteArray())
            fileOutputStream.close()
            val uriToCSVFIle = FileProvider.getUriForFile(context,
fileShareAuthority, file)
            val shareIntent = ShareCompat.IntentBuilder
                .from(context)
                .setStream(uriToCSVFIle)
                .setType(type)
                .intent
            shareIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
            val pm = context.packageManager
            if (shareIntent.resolveActivity(pm) != null) {
                context.startActivity(shareIntent)
            }
        } catch (e: Exception) {
//                    e.printStackTrace();
```

```
        }
    }

    private fun escapeCsv(input: String): String {
        return CsvEscaper().translate(input)!!
    }
}
```

SqliteView.java

```
package my.project.db.manager;

import android.util.SparseArray;

import androidx.annotation.NonNull;
import androidx.annotation.StringRes;
import androidx.appcompat.app.AppCompatActivity;

import java.util.ArrayList;
import java.util.List;

interface SqliteView {
    void setSubtitle(String subtitle);

    void finishActivityWithError(@StringRes int errorMessageId);

    void finishActivityWithError(String error);

    void finishActivity();

    @NonNull
    AppCompatActivity getViewContext();

    void displayError(@StringRes int errorMessageId);

    void displayError(String error);

    void informErrorToUser(@StringRes int errorMessageId);

    void informErrorToUser(String error);

    void setSpinnerAdapter(ArrayList<String> tableNames);

    void showContentView();

    void updateColumnNames(String[] columnNames);

    void displayRows(List<SparseArray<Object>> columnIndexToValuesArray);

    void showCustomQueryDialog(String previousCustomQuery);

    void setAddFABVisible(boolean visible);

    void showAddEditRowDialog(boolean isEdit, String tableName, String[]
tableColumnNames, SparseArray<Object> oldColumnValues);

    void showTableSelectionView();

    void showCustomQueryView(String customQuery);
}
```

SqlSafeUtils.kt

```
package my.project.db.manager

import java.util.regex.Pattern

public class SqlSafeUtil {
    private val SQL_TYPES =
        "TABLE, TABLESPACE, PROCEDURE, FUNCTION, TRIGGER, KEY, VIEW,
MATERIALIZED VIEW, LIBRARY" +
                "DATABASE LINK, DBLINK, INDEX, CONSTRAINT, TRIGGER, USER,
SCHEMA, DATABASE, PLUGGABLE DATABASE, BUCKET, " +
                "CLUSTER, COMMENT, SYNONYM, TYPE, JAVA, SESSION, ROLE, PACKAGE,
PACKAGE BODY, OPERATOR" +
                "SEQUENCE, RESTORE POINT, PFILE, CLASS, CURSOR, OBJECT, RULE,
USER, DATASET, DATASTORE, " +
                "COLUMN, FIELD, OPERATOR"
    private val SQL_REGEXPS = arrayOf(
        "(?i)(.*)(\\b)+(OR|AND)(\\s)+(true|false)(\\s)*(.*)",
        "(?i)(.*)(\\b)+(OR|AND)(\\s)+(\\w)(\\s)*(\\=)(\\s)*(\\w)(\\s)*(.*)",
        "(?i)(.*)(\\b)+(OR|AND)(\\s)+(equals|not
equals)(\\s)+(true|false)(\\s)*(.*)",
        "(?i)(.*)(\\b)+(OR|AND)(\\s)+([0-9A-Za-z_'][0-9A-Za-
z\\d_']*)(\\s)*(\\=)(\\s)*([0-9A-Za-z_'][0-9A-Za-z\\d_']*)(\\s)*(.*)",
        "(?i)(.*)(\\b)+(OR|AND)(\\s)+([0-9A-Za-z_'][0-9A-Za-
z\\d_']*)(\\s)*(\\!\\=)(\\s)*([0-9A-Za-z_'][0-9A-Za-z\\d_']*)(\\s)*(.*)",
        "(?i)(.*)(\\b)+(OR|AND)(\\s)+([0-9A-Za-z_'][0-9A-Za-
z\\d_']*)(\\s)*(\\<\\>)(\\s)*([0-9A-Za-z_'][0-9A-Za-z\\d_']*)(\\s)*(.*)",
        "(?i)(.*)(\\b)+SELECT(\\b)+\\s.*(\\b)(.*)",
        "(?i)(.*)(\\b)+INSERT(\\b)+\\s.*(\\b)+INTO(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+UPDATE(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+DELETE(\\b)+\\s.*(\\b)+FROM(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+UPSERT(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+SAVEPOINT(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+CALL(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+ROLLBACK(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+KILL(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+DROP(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+CREATE(\\b)+(\\s)*(" + SQL_TYPES.replace(
            ",".toRegex(),
            "|"
        ) + ")(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+ALTER(\\b)+(\\s)*(" + SQL_TYPES.replace(
            ",".toRegex(),
            "|"
        ) + ")(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+TRUNCATE(\\b)+(\\s)*(" + SQL_TYPES.replace(
            ",".toRegex(),
            "|"
        ) + ")(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+LOCK(\\b)+(\\s)*(" + SQL_TYPES.replace(
            ",".toRegex(),
            "|"
        ) + ")(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+UNLOCK(\\b)+(\\s)*(" + SQL_TYPES.replace(
            ",".toRegex(),
            "|"
        ) + ")(\\b)+\\s.*(.*)",
        "(?i)(.*)(\\b)+RELEASE(\\b)+(\\s)*(" + SQL_TYPES.replace(
            ",".toRegex(),
            "|"
        ) + ")(\\b)+\\s.*(.*)",
```

```
            "(?i)(.*)(\\b)+DESC(\\b)+(\\w)*\\s.*(.*)",
            "(?i)(.*)(\\b)+DESCRIBE(\\b)+(\\w)*\\s.*(.*)",
            "(.*)(/\\*|\\*/|;){1,}(.*)",
            "(.*)(-){2,}(.*)"
    )
    private val validationPatterns = buildPatterns(SQL_REGEXPS)
    fun isSqlInjectionSafe(dataString: String?): Boolean {
        if (isEmpty(dataString)) {
            return true
        }
        for (pattern in validationPatterns) {
            if (matches(pattern, dataString)) {
                return false
            }
        }
        return true
    }

    private fun matches(pattern: Pattern, dataString: String?): Boolean {
        val matcher = pattern.matcher(dataString)
        return matcher.matches()
    }

    private fun buildPatterns(expressionStrings: Array<String>): List<Pattern> {
        val patterns: MutableList<Pattern> = ArrayList()
        for (expression in expressionStrings) {
            patterns.add(getPattern(expression))
        }
        return patterns
    }

    private fun getPattern(regEx: String): Pattern {
        return Pattern.compile(regEx, Pattern.CASE_INSENSITIVE or
Pattern.UNICODE_CASE)
    }

    private fun isEmpty(cs: CharSequence?): Boolean {
        return cs == null || cs.length == 0
    }
}
```

TableRecyclerAdapter.java

```
package my.project.db.manager;

import android.util.SparseArray;
import android.view.View;
import android.view.ViewGroup;

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

import java.util.List;

class TableRecyclerAdapter extends
RecyclerView.Adapter<TableRecyclerAdapter.ViewHolder> {

    private List<SparseArray<Object>> mColumnIndexToValuesArray;
    private Listener mListener;

    TableRecyclerAdapter(@Nullable List<SparseArray<Object>>
columnIndexToValuesArray) {
```

```
        mColumnIndexToValuesArray = columnIndexToValuesArray;
    }

    @Override
    public int getItemViewType(int position) {
        return mColumnIndexToValuesArray.get(position).size();
    }

    void setData(List<SparseArray<Object>> mColumnIndexToValuesArray) {
        this.mColumnIndexToValuesArray = mColumnIndexToValuesArray;
        notifyDataSetChanged();
    }

    void setListener(Listener listener) {
        this.mListener = listener;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
        return new ViewHolder(new RowView(parent.getContext(), viewType));
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        holder.mRowView.setData(mColumnIndexToValuesArray.get(position));
    }

    @Override
    public int getItemCount() {
        if (mColumnIndexToValuesArray == null) {
            return 0;
        }

        return mColumnIndexToValuesArray.size();
    }

    class ViewHolder extends RecyclerView.ViewHolder {

        final RowView mRowView;

        ViewHolder(RowView itemView) {
            super(itemView);
            mRowView = itemView;

            mRowView.setOnClickListener(v -> {
                int position = getAdapterPosition();
                int layoutPosition = getLayoutPosition();
                if (position != RecyclerView.NO_POSITION && mListener != null) {

mListener.onColumnValueClicked(mColumnIndexToValuesArray.get(layoutPosition));
                }
            });
        }
    }

    interface Listener {
        void onColumnValueClicked(SparseArray<Object> columnValues);
    }
}
```