МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АЕРОКОСМІЧНИЙ УНІВЕРСИТЕТ
ІМ. М. Є. ЖУКОВСЬКОГО
"ХАРКІВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ"

Кафедра комп'ютерних наук та інформаційних технологій

КУРСОВА РОБОТА

з дисципліни «Мобільні та хмарні технології»

на тему Розробка мобільного застосунку-помічника куратора академічної групи

Виконав: здобувач освіти
групи 316ст
спеціальності «122. Комп'ютерні науки»
(освітня програма – «Комп'ютеризація обробки інформації та управління»)

Бабич Ангеліна Олександрівна
(прізвище, ім'я та по батькові)

Керівник А.В. Попов
(підпис) (прізвище та ініціали)

Харків – 2023

ЗМІСТ

ВСТУП

В наш час навіть ті галузі, які традиційно були далекими від ІТ, зазнають цифровізації – впроваджуються новітні інформаційні технології, розробляється та широко використовується спеціалізоване програмне забезпечення, яке дозволяє автоматизувати бізнес-процеси і полегшити виконання рутинних завдань та щоденних операцій.

Особливо помітною цифрова трансформація є у галузі освіти з урахуванням загальної тенденції до дистанційного навчання (через пандемію COVID-19), та з огляду на військовий стан в Україні.

Дослідження свідчать, що більшість користувачів віддають перевагу мобільним застосункам над веб-застосунками [1], а аудиторія користувачів мобільних застосунків вже майже зрівнялась з кількістю користувачів їх десктопних версій [2]. Тому тема курсової роботи – розробка мобільного застосунку-помічника куратора академічної групи є *своєчасною та актуальною*.

Такий застосунок дозволить зберігати, переглядати, редагувати та видаляти інформацію про студентів академічної групи (виконувати CRUD-операції з базою даних) і тримати її завжди при собі – прямо у своєму смартфоні. Застосунок може бути також використаний викладачами, які проводять навчальні курси чи тренінги, вихователями дитсадків, і взагалі – всім, кому необхідно мати постійний доступ до інформації про певну групу осіб.

Мета роботи – створити прототип мобільного застосунку для виконання вищезазначених операцій.

Для досягнення поставленої мети слід виконати ряд *завдань*:

- спроектувати та описати структуру бази даних;
- розробити та описати алгоритми роботи застосунку;
- описати середовище розробки та використані програмні засоби;
- спроектувати та описати архітектуру застосунку та призначення основних блоків коду;

– спланувати та виконати тестування розробленого застосунку та описати результати цього тестування.

Об'єктом дослідження в цій курсовій роботі є технології розробки Android-застосунків. *Предметом* – створення мобільного застосунку для роботи з контактною інформацією з використанням бази даних SQLite.

В процесі роботи над обраною темою було використано наступні *методи дослідження*:

– *Аналіз* – для структурної декомпозиції, розчленування об'єктів дослідження на складові частини;
– *Абстрагування* – для виділення лише суттєвих характеристик об'єктів, операцій та відносин між ними;
– *Узагальнення* – для визначення спільних рис між об'єктами, виділення основних характеристик, притаманних всім об'єктам класу;
– *Формалізація* – для опису об'єктів засобами обраної мови програмування та забезпечення можливості їх дослідження формальними методами;
– *Синтез* – для поєднання окремих характеристик об'єкту в єдине ціле (а якщо точніше, то емпіричний вид синтезу).

В його поточному стані створений в ході курсової роботи застосунок вже цілком може використовуватись за призначенням. В подальшому (базуючись на відгуках користувачів застосунку) можливе додавання такої функціональності, як здійснення телефонних дзвінків чи надсилання поштових повідомлень прямо з застосунку, експорт/імпорт даних тощо.

Пояснювальна записка до курсової роботи викладена на 48 сторінках і складається з вступу, основної частини у складі п'яти розділів, висновків, списку використаних джерел з 17 найменувань, додатку з вихідними кодами застосунку, містить 12 рисунків.

1. ОПИС БАЗИ ДАНИХ

У якості бази даних для застосунку було обрано SQLite [3] – полегшену (вбудовану) реляційну систему керування базами даних, реалізовану у вигляді бібліотеки, яка надає більшість з можливостей, описаних у стандарті SQL-92. Вихідні коди SQLite поширюються як суспільне надбання (public domain), тобто можуть використовуватись будь-ким – без обмежень та безоплатно і з будь-якою метою. Фінансову підтримку розробників SQLite здійснює спеціально створений консорціум, до якого входять такі відомі компанії, як Adobe, Oracle, Mozilla, Nokia, Bentley та Bloomberg. З 2018 року SQLite, як й JSON та CSV, рекомендований Бібліотекою Конгресу США як один з форматів зберігання структурованого набору даних.

Сама бібліотека SQLite написана мовою C, проте є реалізації бібліотек і для JavaScript (sql.js, яка дозволяє обробляти файли БД безпосередньо в браузері), і для C++, Java, Python, Perl, PHP, Ruby, Haskell, Scheme, Smalltalk, Lua тощо.

Особливістю SQLite є те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite є не окремим процесом, з яким взаємодіє застосунок, а однією з бібліотек у складі програми. При використанні SQLite не потрібне додаткове ПЗ чи інфраструктура, а для роботи з даними використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі (з розширенням db) на тому пристрої, на якому виконується застосунок.

Кілька процесів або потоків можуть одночасно без жодних проблем читати дані з однієї бази. А от запис даних можна здійснити тільки в тому випадку, коли жодних інших запитів у цей час не виконується, інакше спроба запису закінчується невдачею. Іншим варіантом є автоматичне повторення спроб запису через задані проміжки часу (retry policy).

Для інших мов програмування розроблено механізм підключення й роботи з БД через цю бібліотеку: C++, Java, Python, Perl, PHP, Ruby, Haskell, Scheme, Smalltalk, Lua тощо.

Характерні риси SQlite – це самодостатність, простота, вбудовуваність, висока продуктивність. Завдяки вдалій архітектурі та надзвичайній простоті рушія SQlite, ця система керування базами даних є популярним вибором багатьох розробників, які використовують її як на мобільних пристроях чи вбудованих (embedded) системах зі скромними апаратними можливостями, так і на потужних серверах, де обробляються величезні масиви даних.

Оскільки розроблений застосунок є досить простим і, по суті, являє собою просто працюючий прототип для демонстрації роботи з базою даних SQLite, структура його бази даних є надзвичайно простою – база даних містить лише одну таблицю Student (рис. 1.1).

Рисунок 1.1 – Структура бази даних прототипу застосунку

Цілочислене поле ID є основним ключем – унікальним ідентифікатором студента, а його значення збільшується автоматично (autoincrement). Текстове поле Name містить ім’я студента, цілочислене Registration – номер його залікової книжки, а текстові поля Phone та Email призначені для збереження телефона та електронної пошти відповідно.

Для завдань такого роду немає сенсу будувати складну структуру з кількох взаємопов’язаних таблиць чи дбати про нормалізацію бази даних. В подальшому, якщо буде вирішено розширити функціональність рішення, можуть бути додані таблиці для збереження інформації про групи, викладачів

та спеціалізації. В такому випадку схема БД може бути такою, як наприклад, на рисунку 1.2.

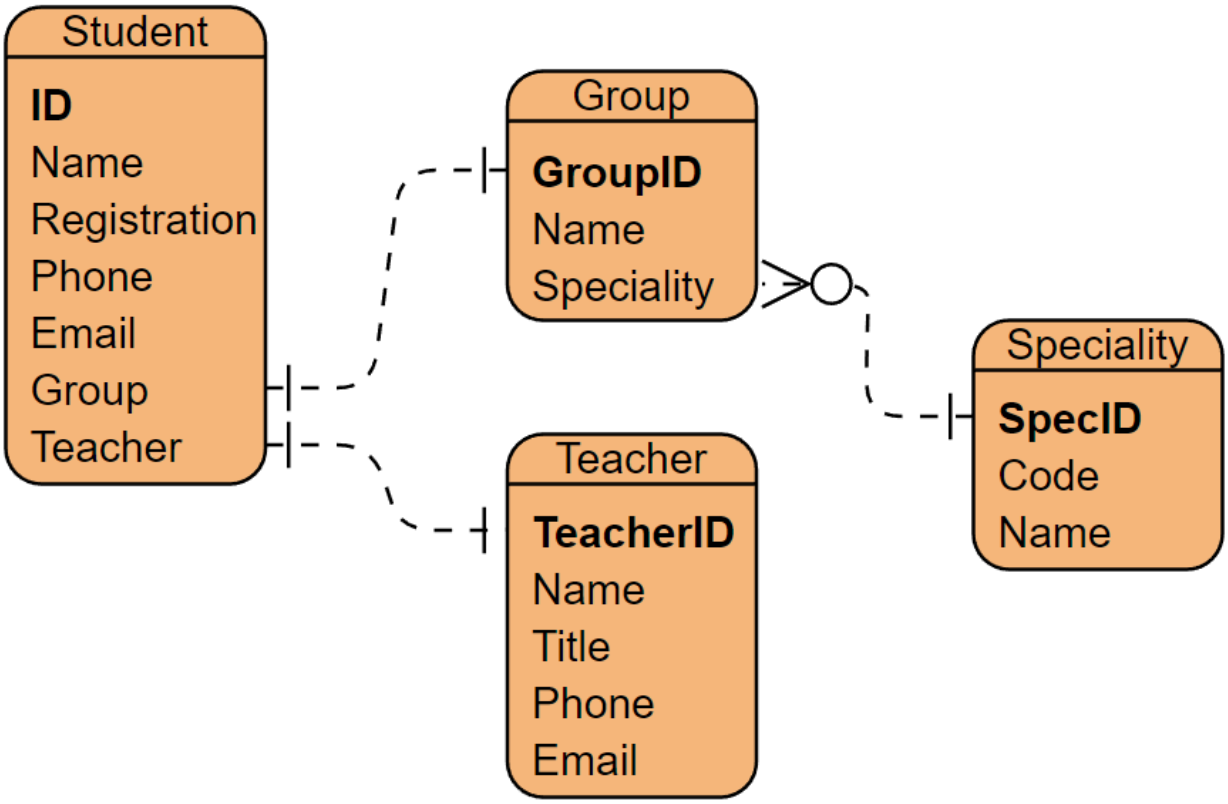


Рисунок 1.2 – Структура бази даних наступної версії застосунку

Як вже було сказано раніше, SQLite може бути використана для створення сховища даних великого обсягу – максимальна кількість таблиць може сягати 2147483646 [4], а загальний обсяг бази даних обмежений 281Тб [5]. В будь-якому випадку SQLite буде зберігати всю базу даних в одному файлі на диску, тож слід врахувати, що більшість з використовуваних на даний момент файлових систем обмежують розмір файлу меншим значенням. З цієї причини, якщо обсяг даних стає значним (наприклад, мова йде про збереження інформації про всіх студентів, співробітників, групи та спеціалізації великого університету зі значною кількістю відокремлених структурних підрозділів), кращим варіантом буде використання клієнт-серверної бази даних, оскільки в цьому випадку дані будуть розподілені між кількома файлами та (в більшості випадків) різними дисками на різних обчислювальних вузлах.

2. АЛГОРИТМИ РОБОТИ ЗАСТОСУНКУ

2.1. Алгоритми та ООП

Алгоритм – це набір інструкцій, які описують порядок дій виконавця для досягнення результату чи розв'язання завдання за скінченну кількість дій, система правил виконання дискретного процесу, яка досягає поставленої мети за скінченний час. Для комп'ютерних програм алгоритм є списком деталізованих інструкцій, що реалізують процес обчислення, який, починаючи з початкового стану, відбувається через послідовність логічних станів, яка завершується кінцевим станом [6].

Для візуалізації алгоритмів раніше зазвичай використовували блок-схеми. Тепер їм на зміну прийшли UML-діаграми діяльності (найпростішим частинним випадком яких і є блок-схеми) чи послідовностей (взаємодія об’єктів всередині системи).

Оскільки сучасні програмні системи зазвичай є об’єктно-орієнтованими, їх робота не може бути представлена у вигляді одного загального алгоритму. На відміну від традиційних підходів, коли програму розглядали як набір підпрограм, або як перелік певних інструкцій, ООП-програми можна вважати сукупністю об'єктів [7], які взаємодіють один з одним і таким чином надають користувачу програми певну функціональність. При цьому кожен об'єкт – це своєрідний незалежний скінчений автомат, який має власний стан (атрибути), демонструє певну поведінку (методи), має окреме призначенням та зону відповідальності. Для кожного з методів може бути побудована окрема діаграма діяльності, а щоб відобразити обмін повідомленнями між об’єктами створюють діаграми послідовностей.

2.2. Процеси, які відбуваються в Android-застосунку

Як вже було сказано раніше, процеси, які відбуваються в застосунку зручно зображати за допомогою діаграм діяльності (активності), частинним випадком яких є блок-схеми, а самі діаграми діяльності в свою чергу є різновидом графу станів скінченого автомату [8], тобто різновидом діаграми

станів. Вершинами цього графу є дії. Дія (активність) – це атомарний елемент поведінки.

В Android дія (activity) – це окремий екран у програмі (за аналогією з окремими вікнами в настільній програмі). Android-застосунок складається з одного або кількох екранів або дій.

Кожна дія проходить різні етапи в ході свого життєвого циклу і керується стеками дій – коли починається нова діяльність, попередня завжди залишається нижче неї. Існує чотири етапи життєвого циклу діяльності:

– дія знаходиться на передньому плані, тобто у верхній частині стека, вона вважається активною (виконується). Зазвичай це дія, з якою в цей момент взаємодіє користувач.
– дія втратила фокус і неповнорозмірна або прозора дія зосередилася поверх вашої діяльності. У такому випадку інша дія займає вищу позицію в багатовіконному режимі, або ця дія не може отримати фокус в поточному віконному режимі.
– дія повністю прихована іншою, вона зупиняється або приховується. Оскільки її вікно приховане, система може зачинити його, коли їй не вистачає пам’яті для роботи.
– система може видалити дію з пам’яті, попросивши її завершитися або просто примусово завершивши її процес. Коли дія знову відображається для користувача, її потрібно повністю перезапустити та відновити до попереднього стану.

Для кожного етапу Android надає нам набір з семи методів, кожен з яких має власне значення для кожного етапу життєвого циклу [9]. На рисунку 2.1 показано діаграму станів, яка ілюструє перехід від однієї дії до іншої.

Назви методів є досить красномовними й говорять самі за себе, тож, на нашу думку, не потребують додаткових пояснень. Згадані вище чотири етапи життєвого циклу діяльності на діаграмі виділені блакитним кольором, умови переходів зазначені текстом.

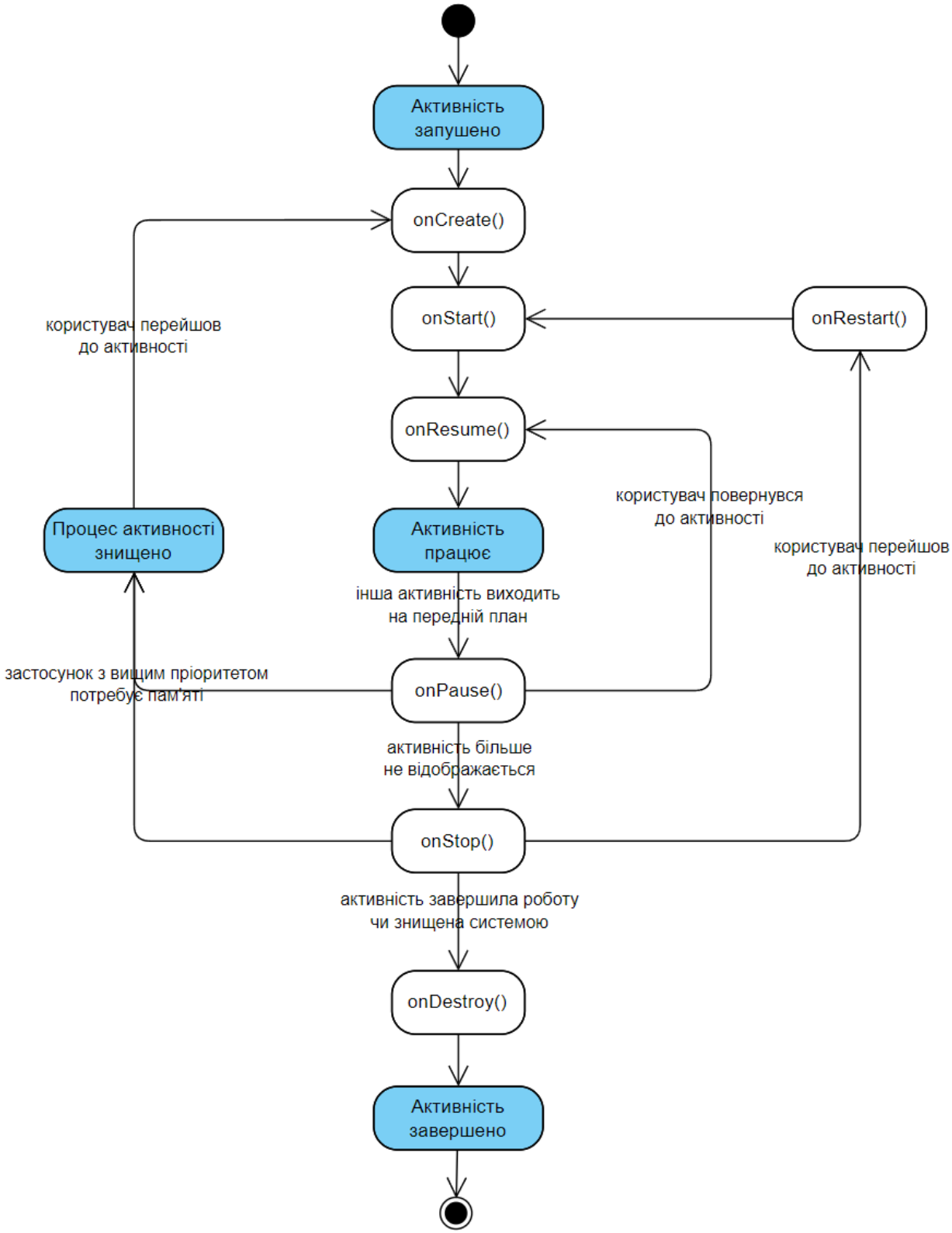


Рисунок 2.1 – Життєвий цикл дії (активності) в Android-застосунку

2.3. Алгоритм роботи зі створеним застосунком

Оскільки застосунок, створений у ході курсової роботи, є надзвичайно простим, і являє собою, по суті, лише працюючий прототип, який демонструє виконання операцій з БД SQLite, малювати діаграми активностей для кожного

методу кожного класу чи діаграми послідовностей аби продемонструвати взаємодію об’єктів, в даному випадку є недоцільним. Натомість було вирішено обмежитись діаграмою діяльності, яка демонструє дії користувача при роботі з застосунком (рисунок 2.2).

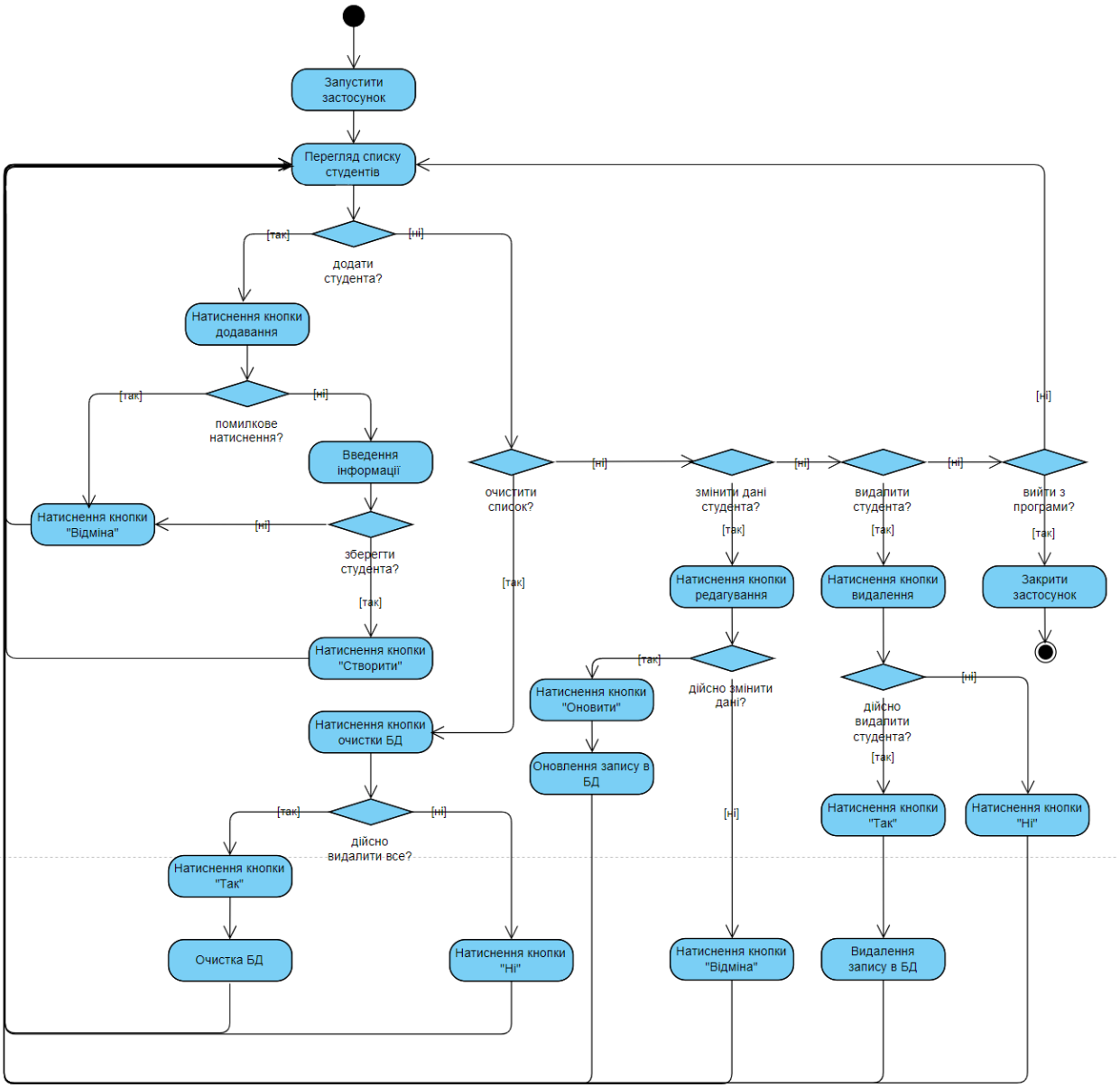


Рисунок 2.2 – Алгоритм роботи користувача з застосунком

Наведена діаграма діяльності наочно демонструє всі можливості, які надаються користувачу створеним застосунком й на нашу думку не потребує додаткових пояснень. Основною активністю в застосунку є перегляд списку студентів, тож всі дії користувача, незалежно від того чи підтвердив він їх чи відмовився від їх завершення, повертають його до списку студентів.

3. ОПИС СЕРЕДОВИЩА І ЗАСОБІВ РОЗРОБКИ

Середовище розробки – це набір апаратних і програмних засобів, процедур та інструментів для розробки, тестування та відлагодження застосунку, програми чи веб-сайту. Зазвичай таке середовище складається з трьох окремих обчислювальних вузлів, або серверних ферм, які прийнято називати DSP – машини для розробки (Development), тестування (Staging), хостингу (Production) застосунку [10].

В нашому випадку всі три ролі виконувала одна машина. Розробку застосунку було виконано на комп’ютері з наступними апаратними характеристиками:

- процесор Intel Core i9-9900, 5 Ггц
- охолодження Arctic Freezer 34 eSport DUO
- материнська плата MSI Z370-A Pro
- оперативна пам’ять HyperX 64 Гб DDR4
- відеокарта NVIDIA GeForce RTX 2060, 6 Гб GDDR6
- основний (системний диск) Western Digital 1Тб SSD Blue
- монітори 2 x MSI Optix MAG, 34” curved, ultrawide (3440x1440)
- клавіатура MSI Interceptor, миша MSI Clutch
- камера, мікрофон, звукове, ігрове обладнання, RGB-підсвітка, освітлення тощо
- широкосмугове інтернет-з’єднання 500Mbps

Слід відмітити, що конфігурація машини відповідає рекомендованим апаратним вимогам Microsoft до ПК для розробки.

Однак, навіть найпотужніший комп’ютер не має сенсу без відповідного програмного забезпечення. Наша машина працює під керуванням ОС Windows 11 Pro 64 біт (версія 22H2) з встановленими пакетами Windows Feature Experience Pack та Windows Power Toys. Основний браузер – Google Chrome (версія 110).

Важливим рішенням для розробника є вибір інтегрованого середовища розробки (IDE) – застосунку, який поєднує в собі можливості редактора

вихідних кодів, відладчика, компілятора, підтримує певну мову програмування, надаючи підказки, автодоповнення коду, статичний аналіз коду тощо, дозволяє проектувати графічний інтерфейс користувача, виконувати тестування проекту, а в деяких випадках – і проектувати майбутній застосунок з використанням UML-діаграм класів, здійснювати кодогенерацію та зворотний інжиніринг. Можливості більшості сучасних IDE можуть бути розширені за допомогою розширень та додаткових модулів.

Під час роботи над завданням курсової роботи було використано Android Studio [11] – офіційне інтегроване середовище розробки для платформи Android, яке прийшло на зміну плагіну ADT для платформи Eclipse. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, від компанії JetBrains, і розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0.

Середовище є крос-платформеним, ним можна користуватись на машинах під керуванням Linux (для тестування використовується Ubuntu), macOS, Windows, Chrome OS [12]. Android Studio надає засоби для розробки застосунків не тільки для смартфонів і планшетів, але й для носимих пристроїв на базі Wear OS, телевізорів з Android TV, окулярів віртуальної та доповненої реальності, і автомобільних інформаційно-розважальних систем з Android Auto. Для застосунків, які розроблялись з використанням Eclipse і ADT Plugin, передбачено інструмент для автоматичного імпорту існуючого проекту в Android Studio.

Android Studio адаптована для виконання типових завдань розробки застосунків для Android – до складу середовища включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, які працюють на пристроях з екранами різної роздільної здатності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Окрім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій – наприклад, нова уніфікована підсистема збірки, тестування і розгортання застосунків,

заснована на інструментарії Gradle, та з підтримкою засобів безперервної інтеграції.

Для прискорення розробки IDE надає колекцію типових елементів інтерфейсу та візуальний редактор для їх компонування (на жаль, неідеальний, як і в оригінальній IntelliJ IDEA, але його можливостей буде достатньо для більшості користувачів), який надає зручний попередній перегляд різних станів інтерфейсу (наприклад, можна подивитися як застосунок буде виглядати в різних версіях Android і на екранах різних розмірів). Для створення нестандартних інтерфейсів надається майстер створення власних інтерфейсних елементів, який підтримує використання шаблонів. Також до середовища були додані функції для завантаження типових прикладів коду з GitHub.

До складу Android Studio також включені модифіковані з урахуванням особливостей платформи Android розширені інструменти рефакторингу, перевірки сумісності з попередніми версіями, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання. До редактору додано режим швидкого внесення правок. Система підсвічування синтаксису, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегровано також підтримку оптимізатора коду ProGuard, вбудовані засоби генерації цифрових підписів та надано інтерфейс для керування локалізацією застосунків.

Разом з тим, системні вимоги середовища є досить скромними – на відміну від, наприклад, Microsoft Visual Studio, з Android Studio можна цілком комфортно працювати навіть на пересічних офісних ПК. На даний момент системні вимоги виглядають наступним чином:

- операційна система Windows (7/8/10/11, 32 та 64 біт)/Linux (з середовищем GNOME/KDE)/OSX/macOS(10.10-13.2.1)/ChromeOS
- оперативна пам’ять 4 Гб мінімум, рекомендовано 8 Гб або більше
- обсяг на диску 500 Мб для самої IDE, 4 Гб або більше для SDK
- пакет Java Development Kit (JDK) версії 8 або новішої

– роздільна здатність екрану 1280x800 мінімум

Для роботи над застосунком нами було використано портативну Android Studio версії 4.2.1 від PortApps [13] з вбудованою OpenJDK 64 біт, що дозволило виконувати повний спектр завдань з розробки мобільних застосунків без необхідності повноцінного встановлення IDE. Загальний вигляд Android Studio, запущеної на нашій машині, показано на рисунку 3.1.

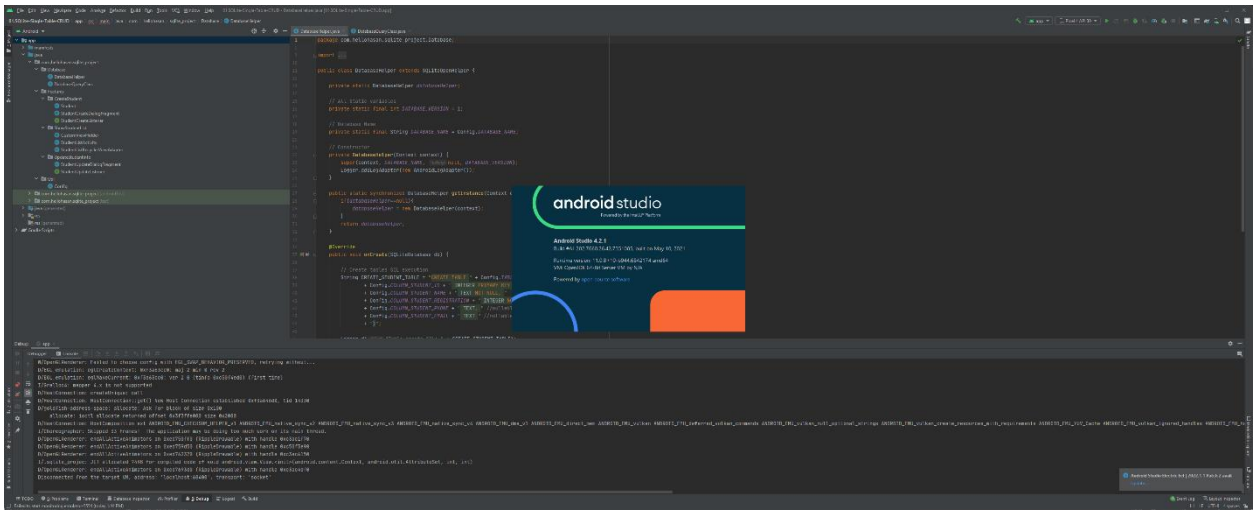


Рисунок 3.1 – Android Studio в режимі відлагодження застосунку

Для відлагодження застосунку використовувались Android 11 (API Level 30) та Android 13 (API Level 33). Було створено віртуальний пристрій Pixel 4 з роздільною здатністю екрану 1080x2280 (440 dpi) та 10 Гб дискового простору (рисунок 3.2).

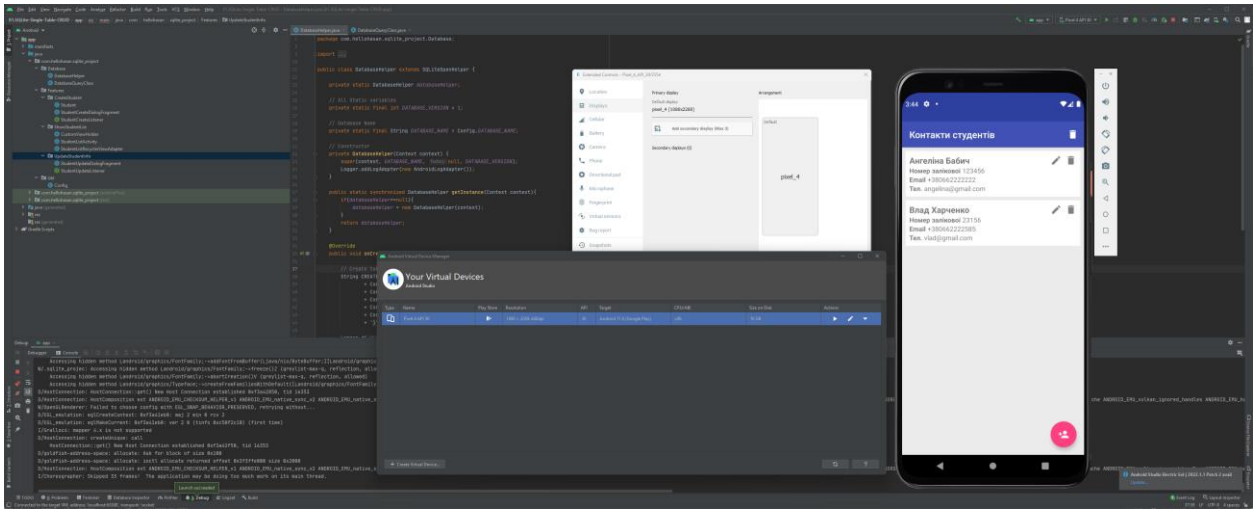


Рисунок 3.2 – Віртуальний пристрій, використаний для відлагодження застосунку

Також для тестування застосунку було використано фізичний пристрій OnePlus 7 (Qualcomm SM8150 Snapdragon 855, 12 Гб оперативної пам’яті /

256 Гб вбудованої, екран 6.41 дюйма з роздільною здатністю 1080x2340), працюючий під керуванням Android 12.

Результати тестування застосунку (як з використанням віртуального пристрою, так і за допомогою віртуального пристрою в емуляторі) наведені у п’ятому розділі пояснювальної записки «Тестування програми».

4. АРХІТЕКТУРА ЗАСТОСУНКУ

Оскільки створений в ході курсової роботи застосунок є надзвичайно простим – по суті, працюючим прототипом для демонстрації виконання операцій з БД SQLite, його архітектура є досить простою. Взагалі, архітектура програмного продукту – це абстракція програмної системи, опис того, з яких основних складових частин (підсистем, компонентів, класів) складається система. При цьому ця абстракція може бути різного рівня – на найвищому рівні можна говорити про підсистеми, на рівні нижче ми обговорюємо пакети, ще нижче – класи.

Якщо зобразити структуру пакетів нашого застосунку за допомогою відповідної UML-діаграми (отриманої за допомогою зворотного інжинірингу в StarUML), це буде щось подібне до картини, показаної на рисунку 4.1.

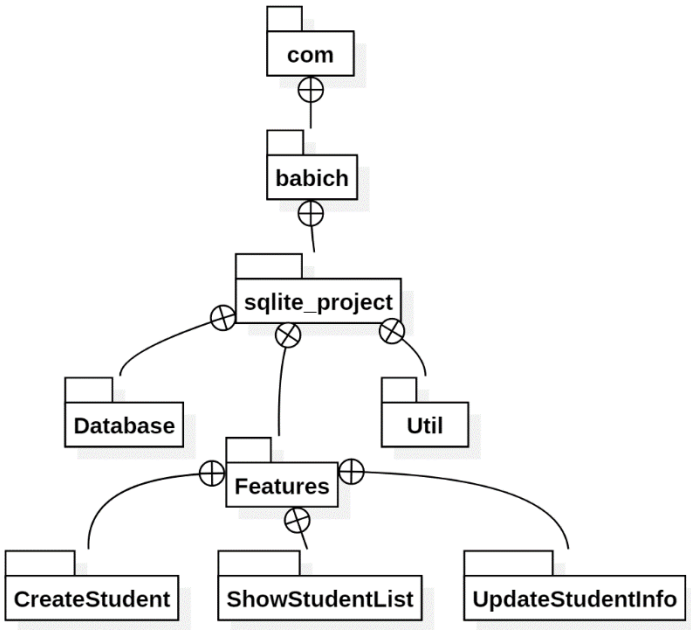


Рисунок 4.1 – Діаграма пакетів створеного застосунку

Назви пакетів є досить красномовними й однозначно описують призначення кожного з них. Так, наприклад, пакет Database містить класи для роботи з БД – DatabaseHelper та DatabaseQueryClass (назви теж говорять самі за себе), пакет Util – клас Config, який містить необхідні для роботи константи, пакет Features в свою чергу містить пакети, які надають класи для роботи зі студентами, відображення списку студентів та оновлення інформації про студента.

Повну і детальну діаграму класів (отриману за допомогою зворотного інжинірингу в Android Studio за допомогою плагіна UML Generator), показано на рисунку 4.2.

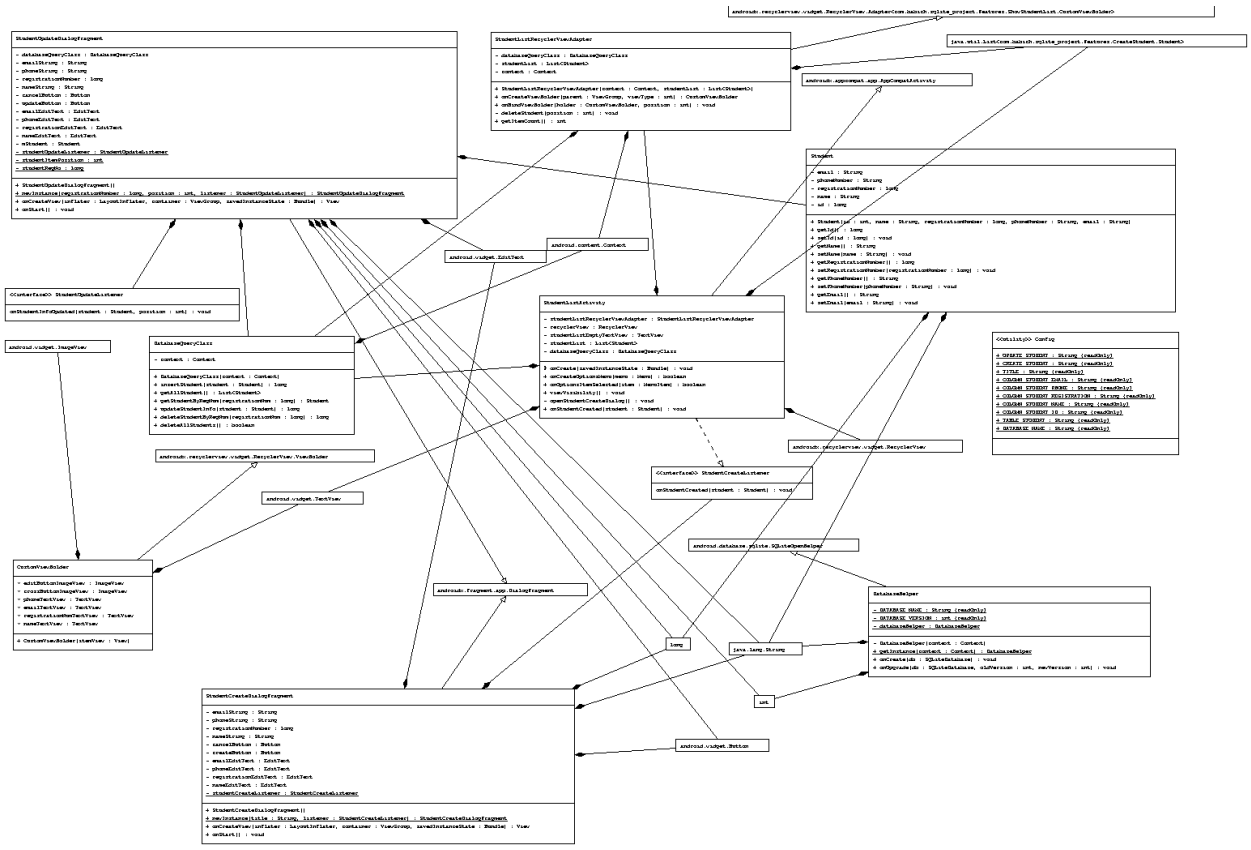


Рисунок 4.2 – Детальна діаграма класів створеного застосунку

Оскільки площа аркуша А4 досить невелика, а інформації, показаної на діаграмі, досить багато, в такому розмірі діаграма є практично нечитабельною і може використовуватись лише для отримання загального уявлення про структуру застосунку.

Натомість ми наведемо тут частину коду деяких важливих для розуміння роботи нашого застосунку класів. А почнемо ми з класу DatabaseHelper, який є нащадком класу SQLiteOpenHelper – стандартного класу з пакету android.database.sqlite.SQLiteDatabase. Це допоміжний клас, який відповідає за створення баз даних SQLite та за керування їх версіями. Для того, щоб скористатися його можливостями, ми створили його підклас (DatabaseHelper), в якому (окрім приватного конструктора та методу для отримання екземпляру класу – це реалізація паттерну Singleton (одинак), який обмежує кількість екземплярів класу до одного) перевизначили два методи – onCreate() та

onUpgrade(). Клас дозволяє відкривати існуючі бази даних, створювати нові бази даних та оновлювати версії баз даних. Фрагмент коду нашого класу DatabaseHelper виглядає наступним чином:

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private static DatabaseHelper databaseHelper;

    // All Static variables
    private static final int DATABASE_VERSION = 1;

    // Database Name
    private static final String DATABASE_NAME = Config.DATABASE_NAME;

    // Constructor
    private DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
        Logger.addLogAdapter(new AndroidLogAdapter());
    }

    public static synchronized DatabaseHelper getInstance(Context context){
        if(databaseHelper==null){
            databaseHelper = new DatabaseHelper(context);
        }
        return databaseHelper;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        // Create tables SQL execution
        String CREATE_STUDENT_TABLE = "CREATE TABLE " + Config.TABLE_STUDENT
           + "(" + Config.COLUMN_STUDENT_ID
           + " INTEGER PRIMARY KEY AUTOINCREMENT, "
           + Config.COLUMN_STUDENT_NAME + " TEXT NOT NULL, "
           + Config.COLUMN_STUDENT_REGISTRATION
           + " INTEGER NOT NULL UNIQUE, "
           + Config.COLUMN_STUDENT_PHONE + " TEXT, " //nullable
           + Config.COLUMN_STUDENT_EMAIL + " TEXT " //nullable
           + ")";

        Logger.d("Table create SQL: " + CREATE_STUDENT_TABLE);

        db.execSQL(CREATE_STUDENT_TABLE);

        Logger.d("DB created!");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
        // Drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + Config.TABLE_STUDENT);

        // Create tables again
        onCreate(db);
    }

}
```

Наступний важливий клас – це клас DatabaseQueryClass, назва якого теж є досить красномовною, тож однозначно свідчить про його призначення – виконання операцій з записами у таблиці БД. Клас містить методи для додавання студента (insertStudent), отримання всіх студентів з БД (getAllStudent), пошуку студента за номером залікової (getStudentByRegNum), оновлення інформації про студента (updateStudentInfo), видалення студента (відібраного за номером залікової – deleteStudentByRegNum), та видалення всіх записів з таблиці (deleteAllStudents). Фрагмент коду цього класу виглядає наступним чином:

```
public class DatabaseQueryClass {

    private Context context;

    public DatabaseQueryClass(Context context){
        this.context = context;
        Logger.addLogAdapter(new AndroidLogAdapter());
    }

    public long insertStudent(Student student){

        long id = -1;
        DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
        SQLiteDatabase sqLiteDatabase = databaseHelper.getWritableDatabase();

        ContentValues contentValues = new ContentValues();
        contentValues.put(Config.COLUMN_STUDENT_NAME, student.getName());
        contentValues.put(Config.COLUMN_STUDENT_REGISTRATION,
        student.getRegistrationNumber());
        contentValues.put(Config.COLUMN_STUDENT_PHONE,
        student.getPhoneNumber());
        contentValues.put(Config.COLUMN_STUDENT_EMAIL, student.getEmail());

        try {
            id = sqLiteDatabase.insertOrThrow(Config.TABLE_STUDENT, null,
                 contentValues);
        } catch (SQLiteException e){
            Logger.d("Exception: " + e.getMessage());
            Toast.makeText(context, "Operation failed: " + e.getMessage(),
                           Toast.LENGTH_LONG).show();
        } finally {
            sqLiteDatabase.close();
        }

        return id;
    }

    public List<Student> getAllStudent(){

        DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
        SQLiteDatabase sqLiteDatabase = databaseHelper.getReadableDatabase();

        Cursor cursor = null;
        try {
```

```
            cursor = sqLiteDatabase.query(Config.TABLE_STUDENT, null, null,
                    null, null, null, null, null);

            /**
                аналог
                String SELECT_QUERY = String.format("SELECT %s, %s, %s, %s,
                                        %s FROM %s", Config.COLUMN_STUDENT_ID,
                                        Config.COLUMN_STUDENT_NAME,
                                        Config.COLUMN_STUDENT_REGISTRATION,
                                        Config.COLUMN_STUDENT_EMAIL,
                                        Config.COLUMN_STUDENT_PHONE,
                                        Config.TABLE_STUDENT);
                 cursor = sqLiteDatabase.rawQuery(SELECT_QUERY, null);
             */

            if(cursor!=null)
                if(cursor.moveToFirst()){
                    List<Student> studentList = new ArrayList<>();
                    do {
                        int id = cursor.getInt(cursor.getColumnIndex(
                                Config.COLUMN_STUDENT_ID));
                        String name = cursor.getString(cursor.getColumnIndex(
                                      Config.COLUMN_STUDENT_NAME));
                        long registrationNumber = cursor.getLong(
                                                  cursor.getColumnIndex(
                                    Config.COLUMN_STUDENT_REGISTRATION));
                        String email = cursor.getString(
                                       cursor.getColumnIndex(
                                       Config.COLUMN_STUDENT_EMAIL));
                        String phone = cursor.getString(
                                       cursor.getColumnIndex(
                                       Config.COLUMN_STUDENT_PHONE));

                        studentList.add(new Student(id, name,
                                        registrationNumber, email, phone));
                    } while (cursor.moveToNext());

                    return studentList;
                }
        } catch (Exception e){
            Logger.d("Exception: " + e.getMessage());
            Toast.makeText(context, "Operation failed",
            Toast.LENGTH_SHORT).show();
        } finally {
            if(cursor!=null)
                cursor.close();
            sqLiteDatabase.close();
        }

        return Collections.emptyList();
    }

    public Student getStudentByRegNum(long registrationNum){

        DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
        SQLiteDatabase sqLiteDatabase = databaseHelper.getReadableDatabase();

        Cursor cursor = null;
        Student student = null;
        try {

            cursor = sqLiteDatabase.query(Config.TABLE_STUDENT, null,
                    Config.COLUMN_STUDENT_REGISTRATION + " = ? ", new
                    String[]{String.valueOf(registrationNum)},
```

```
                null, null, null);

        /**
             аналог
             String SELECT_QUERY = String.format("SELECT * FROM %s WHERE
                                   %s = %s", Config.TABLE_STUDENT,
                                   Config.COLUMN_STUDENT_REGISTRATION,
                                   String.valueOf(registrationNum));
             cursor = sqLiteDatabase.rawQuery(SELECT_QUERY, null);
         */

        if(cursor.moveToFirst()){
            int id = cursor.getInt(cursor.getColumnIndex(
                                   Config.COLUMN_STUDENT_ID));
            String name = cursor.getString(cursor.getColumnIndex(
                                           Config.COLUMN_STUDENT_NAME));
            long registrationNumber = cursor.getLong(
                                    cursor.getColumnIndex(
                                    Config.COLUMN_STUDENT_REGISTRATION));
            String phone = cursor.getString(cursor.getColumnIndex(
                                           Config.COLUMN_STUDENT_PHONE));
            String email = cursor.getString(cursor.getColumnIndex(
                                           Config.COLUMN_STUDENT_EMAIL));

            student = new Student(id, name, registrationNumber, phone,
                                  email);
        }
    } catch (Exception e){
        Logger.d("Exception: " + e.getMessage());
        Toast.makeText(context, "Operation failed",
        Toast.LENGTH_SHORT).show();
    } finally {
        if(cursor!=null)
            cursor.close();
        sqLiteDatabase.close();
    }

    return student;
}

public long updateStudentInfo(Student student){

    long rowCount = 0;
    DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
    SQLiteDatabase sqLiteDatabase = databaseHelper.getWritableDatabase();

    ContentValues contentValues = new ContentValues();
    contentValues.put(Config.COLUMN_STUDENT_NAME, student.getName());
    contentValues.put(Config.COLUMN_STUDENT_REGISTRATION,
                      student.getRegistrationNumber());
    contentValues.put(Config.COLUMN_STUDENT_PHONE,
                      student.getPhoneNumber());
    contentValues.put(Config.COLUMN_STUDENT_EMAIL, student.getEmail());

    try {
        rowCount = sqLiteDatabase.update(Config.TABLE_STUDENT,
                                         contentValues,
                Config.COLUMN_STUDENT_ID + " = ? ",
                new String[] {String.valueOf(student.getId())});
    } catch (SQLiteException e){
        Logger.d("Exception: " + e.getMessage());
        Toast.makeText(context, e.getMessage(),
        Toast.LENGTH_LONG).show();
    } finally {
```

```
            sqLiteDatabase.close();
        }

        return rowCount;
    }

    public long deleteStudentByRegNum(long registrationNum) {
        long deletedRowCount = -1;
        DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
        SQLiteDatabase sqLiteDatabase = databaseHelper.getWritableDatabase();

        try {
            deletedRowCount = sqLiteDatabase.delete(Config.TABLE_STUDENT,
                                    Config.COLUMN_STUDENT_REGISTRATION
                                    + " = ? ",
                                    new String[]{
                                           String.valueOf(registrationNum)});
        } catch (SQLiteException e){
            Logger.d("Exception: " + e.getMessage());
            Toast.makeText(context, e.getMessage(),
            Toast.LENGTH_LONG).show();
        } finally {
            sqLiteDatabase.close();
        }

        return deletedRowCount;
    }

    public boolean deleteAllStudents(){
        boolean deleteStatus = false;
        DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
        SQLiteDatabase sqLiteDatabase = databaseHelper.getWritableDatabase();

        try {
            sqLiteDatabase.delete(Config.TABLE_STUDENT, null, null);

            long count = DatabaseUtils.queryNumEntries(sqLiteDatabase,
                                                    Config.TABLE_STUDENT);

            if(count==0)
                deleteStatus = true;

        } catch (SQLiteException e){
            Logger.d("Exception: " + e.getMessage());
            Toast.makeText(context, e.getMessage(),
            Toast.LENGTH_LONG).show();
        } finally {
            sqLiteDatabase.close();
        }

        return deleteStatus;
    }

}
```

Останній фрагмент коду вже неоднозначно вказує на важливість ще одного класу, а саме класу Config, який містить в собі всі потрібні для роботи нашого застосунку константи (назва БД, таблиці для зберігання записів про студентів, назви полів таблиці тощо) . Код цього класу виглядає наступним чином:

```
public class Config {

    public static final String DATABASE_NAME = "student-db";

    //column names of student table
    public static final String TABLE_STUDENT = "student";
    public static final String COLUMN_STUDENT_ID = "_id";
    public static final String COLUMN_STUDENT_NAME = "name";
    public static final String COLUMN_STUDENT_REGISTRATION =
                                            "registration_no";
    public static final String COLUMN_STUDENT_PHONE = "phone";
    public static final String COLUMN_STUDENT_EMAIL = "email";

    //others for general purpose key-value pair data
    public static final String TITLE = "title";
    public static final String CREATE_STUDENT = "create_student";
    public static final String UPDATE_STUDENT = "update_student";
}
```

Повні вихідні коди застосунку містяться у додатку А.

5. ТЕСТУВАННЯ ПРОГРАМИ

Якщо мова йде про тестування створеного застосунку, то почати слід з юніт-тестів [14] (модульне тестування) – це єдиний вид тестів, за які відповідає розробник. Дуже часто компанії налаштовують свої системи контролю версій таким чином, що розробник не має змоги завантажити результати своєї роботи до репозиторію доки всі юніт-тести не будуть успішними. Юніт-тести побудовані на основі надзвичайно простої ідеї – для кожного методу кожного класу надати еталонні значення аргументів і порівняти отримані результати з тими, які мали б бути отримані для цих значень аргументів. Якщо очікуване та фактичне значення співпадають – це означає, що тест пройдено.

Для написання таких тестів слід скористатись відповідною бібліотекою. В нашому випадку це бібліотека jUnit. Для того щоб JUnit знав, що метод вашого тестового класу є тестом, потрібно додати до нього анотацію @Test. JUnit містить у собі клас Assert, який саме й дозволяє порівнювати фактичні значення з очікуваними та виводить помилку, якщо значення не збігаються.

Ось таким, наприклад, міг би бути юніт-тест нашого застосунку, за допомогою якого ми перевіряємо коректність задання номера залікової книжки при створенні студента:

```
public class MyUnitTests {
    @Test
    public void testCreateStudent() throws Exception{
        Student student = new Student(1,
                "Jphn",
                11111,
                "+380665555555",
                "john@mail.com");
        assertEquals(11111, student.getRegistrationNumber());
    }
}
```

Android Studio містить вбудовані засоби для юніт-тестування. Результат виконання нашого тесту ми могли б побачити наступним чином (рисунок 5.1).

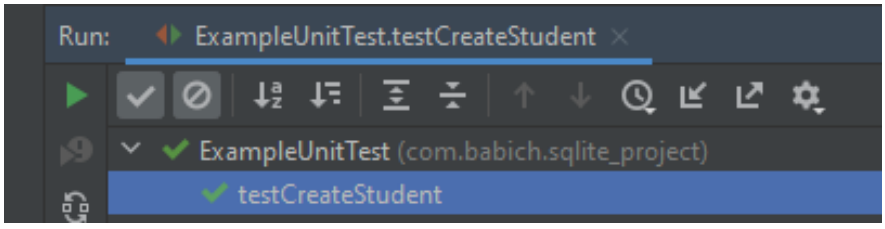


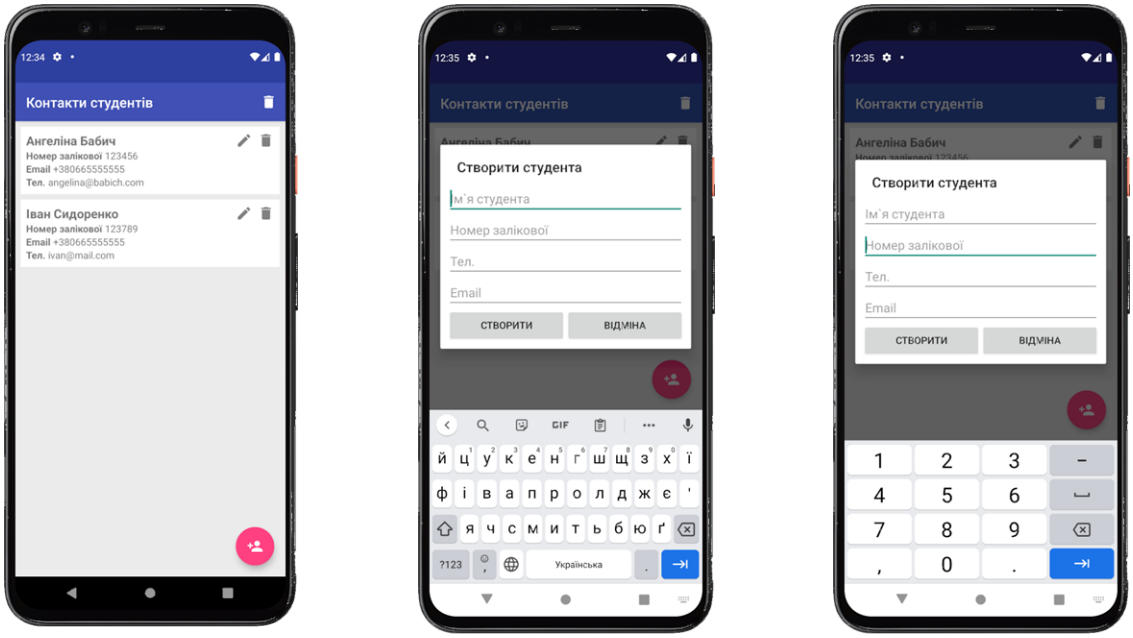
Рисунок 5.1 – Результат виконання юніт-тесту

Оскільки створений застосунок, як вже неодноразово було сказано, є надзвичайно простим, було прийнято рішення не писати юніт-тести, а зосередитись на ручному функціональному тестуванні та тестуванні сумісності. Основний вид тестів в нашому випадку – це функціональне тестування. Мета цього виду тестів полягає у виявленні невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог [15]. Часто такі тести автоматизують за допомогою різноманітного програмного забезпечення, але тести на функціональність можуть бути проведені й в ручному режимі. Найпростіший варіант – просто перевірити коректність виконання основних операцій в застосунку (його функціональність) в Android емуляторі.

В емуляторі нами були виконані наступні тести:

– тест на створення студента (з перевіркою можливості вводу даних неправильних типів);
– тест на редагування інформації (з перевіркою можливості вводу даних неправильних типів);
– тест на видалення студента (з перевіркою можливості відміни операції);
– тест на очищення БД (з перевіркою можливості відміни операції);

Всі тести було пройдено успішно, дефектів не виявлено (що й не дивно, враховуючи простоту застосунку). Результати тестів показані на рисунку 5.2.



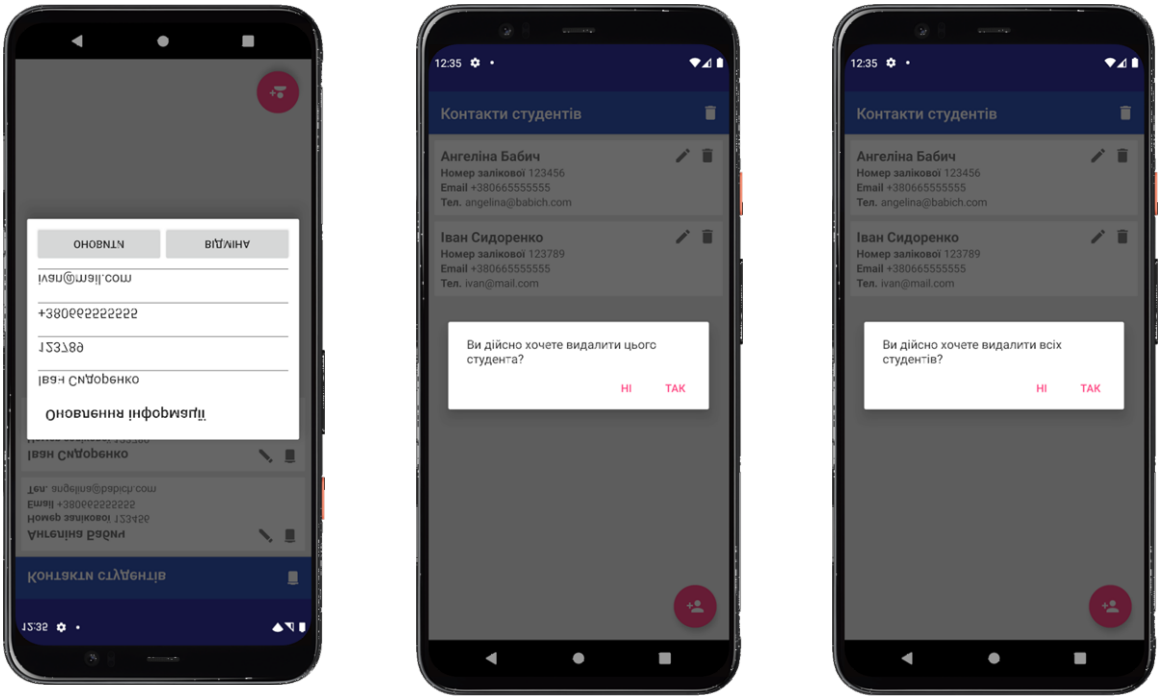


Рисунок 5.2 – Тестування застосунку в емуляторі

Запис всіх виконаних в ході тестування дій (анімоване GIF-зображення) доступне за посиланням https://gifyu.com/image/S7odd.

Ще один вид тестів, які є сенс виконувати у нашому випадку – це тести сумісності [16]. В нашому випадку – це перевірка коректної роботи продукту (функціональності нашого застосунку) на різних мобільних пристроїв. Найпростішим варіантом виконання такого виду тестів буде скористатись сервісами на кшталт BrowserStack App Live [17]. В безкоштовному варіанті він надає можливість тестування на реальних пристроях різних моделей та різних виробників – треба просто завантажити свій APK-файл і пам’ятати, що тривалість сесії обмежена двома хвилинами (рисунок 5.3). Є можливість налаштування пристрою у відповідності до ваших вимог та запису сесії.

Всі тести, проведені в емуляторі, були повторені також за допомогою BrowserStack App Live на таких реальних пристроях (розташованих у німецькому датацентрі компанії, у Франкфурті на Майні):

– Samsung Galaxy S22+
– Huawei P30
– Oppo Reno 3 Pro
– Moto G9 Play

– Google Pixel 5
– Vivo Y50

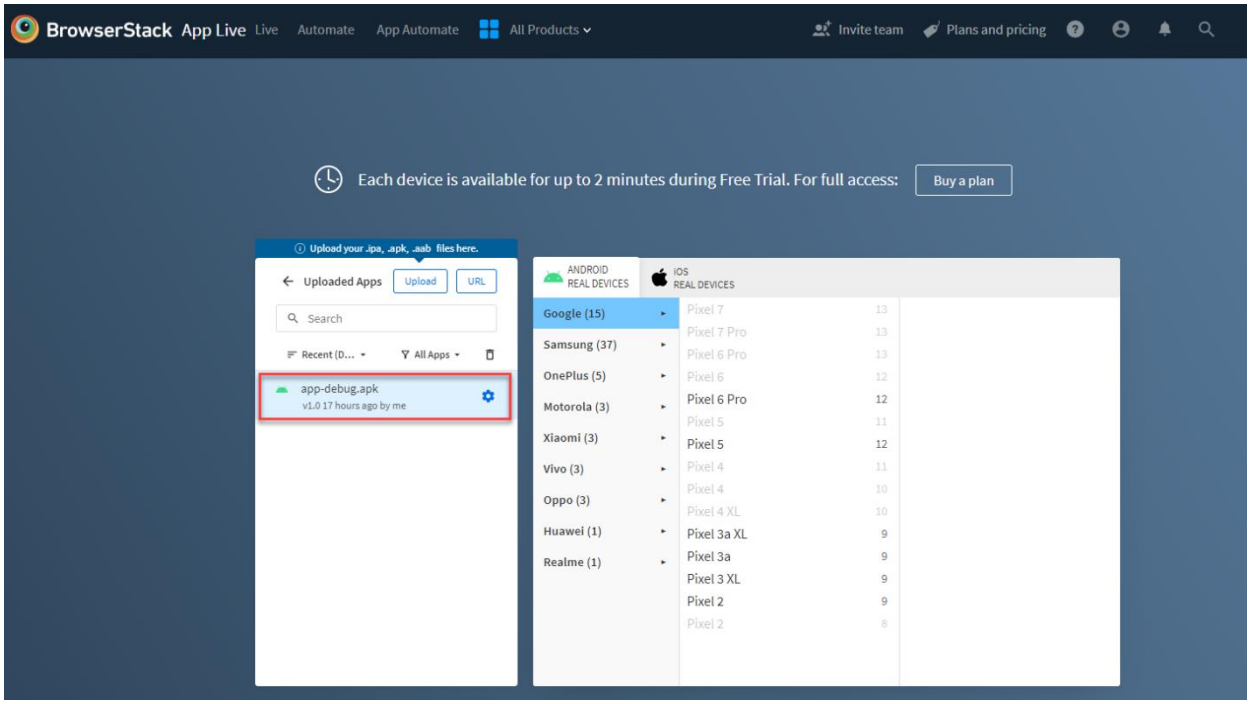


Рисунок 5.3 – Підготовка до тестування на реальних пристроях

Ми спеціально підібрали такий собі мікс з актуальних та дещо застарілих моделей телефонів аби переконатися, що наш застосунок буде працювати коректно, незалежно від версії ОС (Android 11/12/13), розміру та роздільної здатності екрану. Слід відмітити, що не всі моделі, і навіть пристрої не всіх виробників є доступними у безкоштовному плані користування сервісом. Так, наприклад, апарати популярного бренда OnePlus доступні лише за умови придбання преміум-передплати (однак, в нашому розпорядженні був фізичний пристрій OnePlus 7 12/256Гб, тож ми мали змогу «прогнати» на ньому всі тести). Те ж саме стосується й пристроїв від Xiaomi та Realme.

І знову – всі тести було пройдено успішно, дефектів не виявлено. Єдина проблема була пов’язана з помилкою в реалізації самого BrowserStack – при активації запису сесії неможливо скористатись екранною клавіатурою пристрою. Саме з цієї причини ми не надаємо посилання на запис дій, виконуваних нами в ході тестування застосунку на реальних пристроях. Звісно ж, ми повідомили про цю помилку розробників сервісу.

Результати тестів показані на рисунку 5.4.

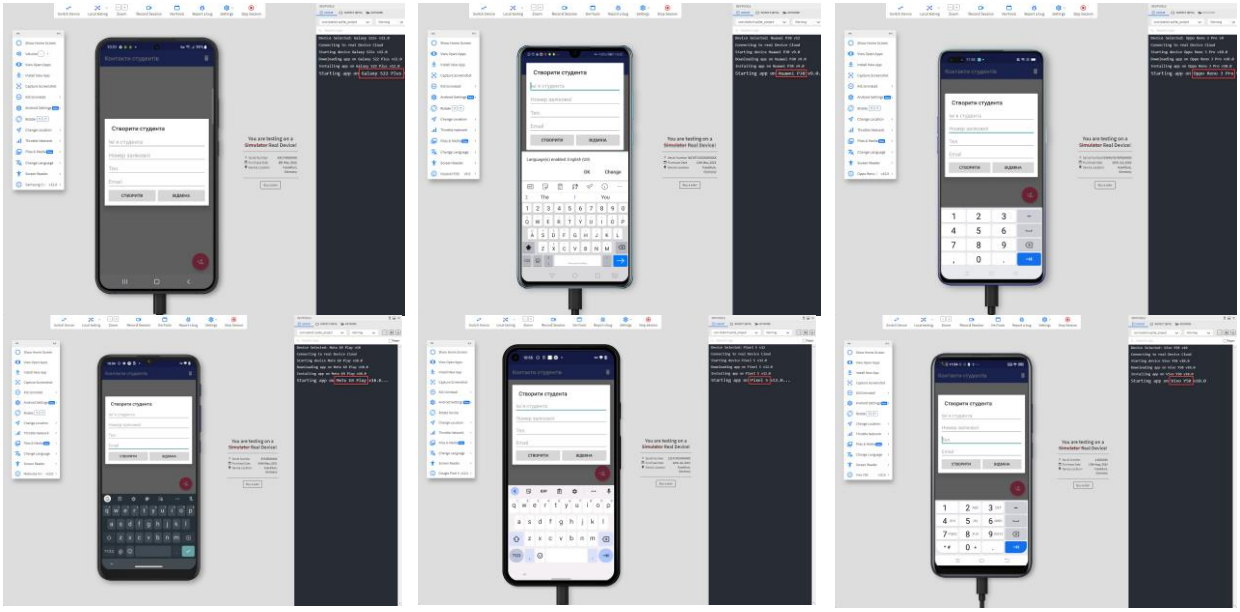


Рисунок 5.4 – Тестування застосунку на реальних пристроях

За результатами тестування можна зробити висновок, що створений програмний продукт є достатньо якісним аби перейти до етапу пілотного впровадження (що можливо й буде зроблено у ВСП «ППФК НТУ «ХПІ», де працює автор курсової роботи). APK-файл доступний для завантаження та встановлення зі сторінки релізів репозиторію – https://github.com/angelina-babych/Kursova-kHAI/releases/tag/v0.0.1

ВИСНОВКИ

Під час написання курсової роботи я працювала над завданням з розробки мобільного застосунку-помічника куратора академічної групи, яке є, безперечно, своєчасним та актуальним.

В результаті було створено мобільний застосунок для платформи Google Android, який дозволяє зберігати, переглядати, редагувати та видаляти інформацію про студентів академічної групи (виконувати CRUD-операції з базою даних) і тримати її завжди при собі – прямо у своєму смартфоні. Застосунок стане в нагоді керівникам груп, викладачам, які проводять навчальні курси чи тренінги, вихователям дитсадків, і взагалі – всім, кому необхідно мати постійний доступ до інформації про певну групу осіб.

Застосунок було створено за допомогою офіційного інтегрованого середовища розробки для Android, а саме Android Studio, яке прийшло на зміну плагіну ADT для платформи Eclipse. Вихідні коди застосунку та файл для встановлення доступні в публічному репозиторії https://github.com/angelina-babych/Kursova-kHAI. Тестування застосунку було виконане як в емуляторі, так і на реальних пристроях (за допомогою сервісу BrowserStack App Live), працюючих під керуванням Android 11/12/13 з різними розмірами екрану та різною роздільною здатністю, й не виявило дефектів, які перешкоджали б його використанню.

Вважаю, завдання курсової роботи було виконано у повному обсязі. Застосунок працює й готовий до пілотного впровадження, за результатами якого до нього, можливо, будуть додані нові функціональні можливості (здійснення телефонних дзвінків чи надсилання поштових повідомлень прямо з застосунку, експорт/імпорт даних тощо). Пояснювальна записка до курсової роботи містить вичерпний опис структури бази даних, алгоритмів роботи, середовища та засобів розробки, архітектури застосунку та підходів до його тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 85% of consumers favour apps over mobile websites // Econsultancy: The Digital Marketing and Ecommerce Experts: [Веб-сайт]. URL: https://econsultancy.com/85-of-consumers-favour-apps-over-mobile-websites/ (дата звернення: 09.03.2023).
2. Mobile vs Desktop Usage Statistics for 2023 // Research.com: [Веб-сайт]. URL: https://research.com/software/mobile-vs-desktop-usage (дата звернення: 09.03.2023).
3. SQLite // Вікіпедія: Вільна енциклопедія: [Веб-сайт]. URL: https://uk.wikipedia.org/wiki/SQLite (дата звернення: 09.03.2023).
4. What Are The Limitations Of SQLite // DBTalks: [Веб-сайт]. URL: https://www.dbtalks.com/tutorials/learn-sqlite/what-are-the-limitations-of-sqlite (дата звернення: 09.03.2023).
5. Appropriate Uses For SQLite // SQLite: [Веб-сайт]. URL: https://www.sqlite.org/whentouse.html (дата звернення: 09.03.2023).
6. Алгоритм // Вікіпедія: Вільна енциклопедія: [Веб-сайт]. URL: https://uk.wikipedia.org/wiki/Алгоритм (дата звернення: 11.03.2023).
7. Об'єктно-орієнтоване програмування // Вікіпедія: Вільна енциклопедія: [Веб-сайт]. URL: https://uk.wikipedia.org/wiki/Об%27єктно-орієнтоване_програмування (дата звернення: 11.03.2023).
8. Діаграма діяльності // Вікіпедія: Вільна енциклопедія: [Веб-сайт]. URL: https://uk.wikipedia.org/wiki/Діаграма діяльності (дата звернення: 11.03.2023).
9. Activity Lifecycle in Android with Demo App // Geeks for Geeks: [Веб-сайт]. URL: https://www.geeksforgeeks.org/activity-lifecycle-in-android-with-demo-app/ (дата звернення: 11.03.2023).
10. What Is a Development Environment? (Definition and Types) // Indeed: [Веб-сайт]. URL: https://www.indeed.com/career-advice/career-development/development-environment (дата звернення: 11.03.2023).
11. Android Studio // Вікіпедія: Вільна енциклопедія: [Веб-сайт]. URL: https://uk.wikipedia.org/wiki/Android_Studio (дата звернення: 11.03.2023).
12. Android development // ChromeOS: [Веб-сайт]. URL: https://chromeos.dev/en/android-environment#install-android-studio-on-chrome-os (дата звернення: 11.03.2023).
13. Android Studio™ portable // Portapps: [Веб-сайт]. URL: https://portapps.io/app/android-studio-portable/ (дата звернення: 11.03.2023).
14. Модульне тестування // Вікіпедія: Вільна енциклопедія: [Веб-сайт]. URL: https://uk.wikipedia.org/wiki/Модульне_тестування (дата звернення: 11.03.2023).
15. Функціональне тестування // Вікіпедія: Вільна енциклопедія: [Веб-сайт]. URL: https://uk.wikipedia.org/wiki/Функціональне_тестування (дата звернення: 11.03.2023).

16. Тестування сумісності ПЗ // Вікіпедія: Вільна енциклопедія: [Веб-сайт]. URL: https://uk.wikipedia.org/wiki/Тестування_сумісності_ПЗ (дата звернення: 12.03.2023).
17. App Live // BrowserStack: [Веб-сайт]. URL: https://app-live.browserstack.com/ (дата звернення: 12.03.2023).

ДОДАТОК А. ВИХІДНІ КОДИ ЗАСТОСУНКУ

Код класу DatabaseHelper

```
package com.babich.sqlite_project.Database;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import com.babich.sqlite_project.Util.Config;
import com.orhanobut.logger.AndroidLogAdapter;
import com.orhanobut.logger.Logger;

public class DatabaseHelper extends SQLiteOpenHelper {

    private static DatabaseHelper databaseHelper;

    // All Static variables
    private static final int DATABASE_VERSION = 1;

    // Database Name
    private static final String DATABASE_NAME = Config.DATABASE_NAME;

    // Constructor
    private DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
        Logger.addLogAdapter(new AndroidLogAdapter());
    }

    public static synchronized DatabaseHelper getInstance(Context context){
        if(databaseHelper==null){
            databaseHelper = new DatabaseHelper(context);
        }
        return databaseHelper;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        // Create tables SQL execution
        String CREATE_STUDENT_TABLE = "CREATE TABLE "
                                    + Config.TABLE_STUDENT + "("
                                    + Config.COLUMN_STUDENT_ID
                                    + " INTEGER PRIMARY KEY AUTOINCREMENT, "
                                    + Config.COLUMN_STUDENT_NAME
                                    + " TEXT NOT NULL, "
                                    + Config.COLUMN_STUDENT_REGISTRATION
                                    + " INTEGER NOT NULL UNIQUE, "
                                    + Config.COLUMN_STUDENT_PHONE
                                    + " TEXT, " //nullable
                                    + Config.COLUMN_STUDENT_EMAIL
                                    + " TEXT " //nullable
                                    + ")";

        Logger.d("Table create SQL: " + CREATE_STUDENT_TABLE);

        db.execSQL(CREATE_STUDENT_TABLE);

        Logger.d("DB created!");
    }
```

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
        // Drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + Config.TABLE_STUDENT);

        // Create tables again
        onCreate(db);
    }

}
```

Код класу DatabaseQueryClass

```
package com.babich.sqlite_project.Database;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.DatabaseUtils;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.widget.Toast;

import com.babich.sqlite_project.Features.CreateStudent.Student;
import com.babich.sqlite_project.Util.Config;
import com.orhanobut.logger.AndroidLogAdapter;
import com.orhanobut.logger.Logger;

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

public class DatabaseQueryClass {

    private Context context;

    public DatabaseQueryClass(Context context){
        this.context = context;
        Logger.addLogAdapter(new AndroidLogAdapter());
    }

    public long insertStudent(Student student){

        long id = -1;
        DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
        SQLiteDatabase sqLiteDatabase = databaseHelper.getWritableDatabase();

        ContentValues contentValues = new ContentValues();
        contentValues.put(Config.COLUMN_STUDENT_NAME, student.getName());
        contentValues.put(Config.COLUMN_STUDENT_REGISTRATION,
        student.getRegistrationNumber());
        contentValues.put(Config.COLUMN_STUDENT_PHONE,
        student.getPhoneNumber());
        contentValues.put(Config.COLUMN_STUDENT_EMAIL, student.getEmail());

        try {
            id = sqLiteDatabase.insertOrThrow(Config.TABLE_STUDENT, null,
                                              contentValues);
        } catch (SQLiteException e){
            Logger.d("Exception: " + e.getMessage());
            Toast.makeText(context, "Operation failed: " + e.getMessage(),
        Toast.LENGTH_LONG).show();
```

```
        } finally {
            sqLiteDatabase.close();
        }

        return id;
    }

    public List<Student> getAllStudent(){

        DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
        SQLiteDatabase sqLiteDatabase = databaseHelper.getReadableDatabase();

        Cursor cursor = null;
        try {

            cursor = sqLiteDatabase.query(Config.TABLE_STUDENT, null, null,
                                          null, null, null, null, null);

            /**
                 аналог

                 String SELECT_QUERY = String.format("SELECT %s, %s, %s, %s,
                                       %s FROM %s", Config.COLUMN_STUDENT_ID,
                                       Config.COLUMN_STUDENT_NAME,
                                       Config.COLUMN_STUDENT_REGISTRATION,
                                       Config.COLUMN_STUDENT_EMAIL,
                                       Config.COLUMN_STUDENT_PHONE,
                                       Config.TABLE_STUDENT);
                 cursor = sqLiteDatabase.rawQuery(SELECT_QUERY, null);
             */

            if(cursor!=null)
                if(cursor.moveToFirst()){
                    List<Student> studentList = new ArrayList<>();
                    do {
                        int id = cursor.getInt(cursor.getColumnIndex(
                                               Config.COLUMN_STUDENT_ID));
                        String name = cursor.getString(cursor.getColumnIndex(
                                      Config.COLUMN_STUDENT_NAME));
                        long registrationNumber = cursor.getLong(
                                                  cursor.getColumnIndex(
                                 Config.COLUMN_STUDENT_REGISTRATION));
                        String email = cursor.getString(
                         cursor.getColumnIndex(Config.COLUMN_STUDENT_EMAIL));
                        String phone = cursor.getString(
                         cursor.getColumnIndex(Config.COLUMN_STUDENT_PHONE));

                        studentList.add(new Student(id, name,
                                        registrationNumber, email, phone));
                    } while (cursor.moveToNext());

                    return studentList;
                }
        } catch (Exception e){
            Logger.d("Exception: " + e.getMessage());
            Toast.makeText(context, "Operation failed",
            Toast.LENGTH_SHORT).show();
        } finally {
            if(cursor!=null)
                cursor.close();
            sqLiteDatabase.close();
        }

        return Collections.emptyList();
```

```
}

public Student getStudentByRegNum(long registrationNum){

    DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
    SQLiteDatabase sqLiteDatabase = databaseHelper.getReadableDatabase();

    Cursor cursor = null;
    Student student = null;
    try {

        cursor = sqLiteDatabase.query(Config.TABLE_STUDENT, null,
                Config.COLUMN_STUDENT_REGISTRATION + " = ? ", new
                                String[]{String.valueOf(registrationNum)},
                null, null, null);

        /**
             аналог

             String SELECT_QUERY = String.format("SELECT * FROM %s WHERE
                                   %s = %s", Config.TABLE_STUDENT,
                                   Config.COLUMN_STUDENT_REGISTRATION,
                                   String.valueOf(registrationNum));
             cursor = sqLiteDatabase.rawQuery(SELECT_QUERY, null);
         */

        if(cursor.moveToFirst()){
            int id = cursor.getInt(cursor.getColumnIndex(
                                                Config.COLUMN_STUDENT_ID));
            String name = cursor.getString(cursor.getColumnIndex(
                                                 Config.COLUMN_STUDENT_NAME));
            long registrationNumber = cursor.getLong(
                                               cursor.getColumnIndex(
                                             Config.COLUMN_STUDENT_REGISTRATION));
            String phone = cursor.getString(cursor.getColumnIndex(
                                               Config.COLUMN_STUDENT_PHONE));
            String email = cursor.getString(cursor.getColumnIndex(
                                               Config.COLUMN_STUDENT_EMAIL));

            student = new Student(id, name, registrationNumber, phone,
                                                                    email);
        }
    } catch (Exception e){
        Logger.d("Exception: " + e.getMessage());
        Toast.makeText(context, "Operation failed",
        Toast.LENGTH_SHORT).show();
    } finally {
        if(cursor!=null)
            cursor.close();
        sqLiteDatabase.close();
    }

    return student;
}

public long updateStudentInfo(Student student){

    long rowCount = 0;
    DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
    SQLiteDatabase sqLiteDatabase = databaseHelper.getWritableDatabase();

    ContentValues contentValues = new ContentValues();
    contentValues.put(Config.COLUMN_STUDENT_NAME, student.getName());
    contentValues.put(Config.COLUMN_STUDENT_REGISTRATION,
```

```
                                        student.getRegistrationNumber());
        contentValues.put(Config.COLUMN_STUDENT_PHONE,
                                                student.getPhoneNumber());
        contentValues.put(Config.COLUMN_STUDENT_EMAIL, student.getEmail());

        try {
            rowCount = sqLiteDatabase.update(Config.TABLE_STUDENT,
                    contentValues,
                    Config.COLUMN_STUDENT_ID + " = ? ",
                    new String[] {String.valueOf(student.getId())});
        } catch (SQLiteException e){
            Logger.d("Exception: " + e.getMessage());
            Toast.makeText(context, e.getMessage(),
            Toast.LENGTH_LONG).show();
        } finally {
            sqLiteDatabase.close();
        }

        return rowCount;
    }

    public long deleteStudentByRegNum(long registrationNum) {
        long deletedRowCount = -1;
        DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
        SQLiteDatabase sqLiteDatabase = databaseHelper.getWritableDatabase();

        try {
            deletedRowCount = sqLiteDatabase.delete(Config.TABLE_STUDENT,
                                    Config.COLUMN_STUDENT_REGISTRATION
                                    + " = ? ",
                                    new String[]{
                                            String.valueOf(registrationNum)});
        } catch (SQLiteException e){
            Logger.d("Exception: " + e.getMessage());
            Toast.makeText(context, e.getMessage(),
            Toast.LENGTH_LONG).show();
        } finally {
            sqLiteDatabase.close();
        }

        return deletedRowCount;
    }

    public boolean deleteAllStudents(){
        boolean deleteStatus = false;
        DatabaseHelper databaseHelper = DatabaseHelper.getInstance(context);
        SQLiteDatabase sqLiteDatabase = databaseHelper.getWritableDatabase();

        try {
            sqLiteDatabase.delete(Config.TABLE_STUDENT, null, null);

            long count = DatabaseUtils.queryNumEntries(sqLiteDatabase,
                                                    Config.TABLE_STUDENT);

            if(count==0)
                deleteStatus = true;

        } catch (SQLiteException e){
            Logger.d("Exception: " + e.getMessage());
            Toast.makeText(context, e.getMessage(),
            Toast.LENGTH_LONG).show();
        } finally {
            sqLiteDatabase.close();
        }
```

```
        return deleteStatus;
    }

}
```

Код класу Student

```
package com.babich.sqlite_project.Features.CreateStudent;

public class Student {
    private long id;
    private String name;
    private long registrationNumber;
    private String phoneNumber;
    private String email;

    public Student(int id, String name, long registrationNumber, String
                   phoneNumber, String email) {
        this.id = id;
        this.name = name;
        this.registrationNumber = registrationNumber;
        this.phoneNumber = phoneNumber;
        this.email = email;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public long getRegistrationNumber() {
        return registrationNumber;
    }

    public void setRegistrationNumber(long registrationNumber) {
        this.registrationNumber = registrationNumber;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
```

```
        this.email = email;
    }
}
```

Код класу StudentCreateDialogFragment

```
package com.babich.sqlite_project.Features.CreateStudent;

import android.app.Dialog;
import android.os.Bundle;
import androidx.fragment.app.DialogFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;

import com.babich.sqlite_project.Util.Config;
import com.babich.sqlite_project.Database.DatabaseQueryClass;
import com.babich.sqlite_project.R;


public class StudentCreateDialogFragment extends DialogFragment {

    private static StudentCreateListener studentCreateListener;

    private EditText nameEditText;
    private EditText registrationEditText;
    private EditText phoneEditText;
    private EditText emailEditText;
    private Button createButton;
    private Button cancelButton;

    private String nameString = "";
    private long registrationNumber = -1;
    private String phoneString = "";
    private String emailString = "";

    public StudentCreateDialogFragment() {
        // Required empty public constructor
    }

    public static StudentCreateDialogFragment newInstance(String title,
        StudentCreateListener listener){
        studentCreateListener = listener;
        StudentCreateDialogFragment studentCreateDialogFragment = new
        StudentCreateDialogFragment();
        Bundle args = new Bundle();
        args.putString("title", title);
        studentCreateDialogFragment.setArguments(args);

        studentCreateDialogFragment.setStyle(DialogFragment.STYLE_NORMAL,
        R.style.CustomDialog);

        return studentCreateDialogFragment;
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        View view = inflater.inflate(R.layout.fragment_student_create_dialog,
```

```
            container, false);

            nameEditText = view.findViewById(R.id.studentNameEditText);
            registrationEditText = view.findViewById(R.id.registrationEditText);
            phoneEditText = view.findViewById(R.id.phoneEditText);
            emailEditText = view.findViewById(R.id.emailEditText);
            createButton = view.findViewById(R.id.createButton);
            cancelButton = view.findViewById(R.id.cancelButton);

            String title = getArguments().getString(Config.TITLE);
            getDialog().setTitle(title);

            createButton.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    nameString = nameEditText.getText().toString();
                    registrationNumber =
                    Integer.parseInt(registrationEditText.getText().toString());
                    phoneString = phoneEditText.getText().toString();
                    emailString = emailEditText.getText().toString();

                    Student student = new Student(-1, nameString,
                                registrationNumber, phoneString, emailString);

                    DatabaseQueryClass databaseQueryClass = new
                                                DatabaseQueryClass(getContext());

                    long id = databaseQueryClass.insertStudent(student);

                    if(id>0){
                        student.setId(id);
                        studentCreateListener.onStudentCreated(student);
                        getDialog().dismiss();
                    }
                }
            });

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


            return view;
        }

        @Override
        public void onStart() {
            super.onStart();
            Dialog dialog = getDialog();
            if (dialog != null) {
                int width = ViewGroup.LayoutParams.MATCH_PARENT;
                int height = ViewGroup.LayoutParams.WRAP_CONTENT;
                //noinspection ConstantConditions
                dialog.getWindow().setLayout(width, height);
            }
        }

    }
```

Код інтерфейсу StudentCreateListener

```
package com.babich.sqlite_project.Features.CreateStudent;

public interface StudentCreateListener {
    void onStudentCreated(Student student);
}
```

Код класу StudentListActivity

```
package com.babich.sqlite_project.Features.ShowStudentList;

import android.content.DialogInterface;
import android.os.Bundle;
import com.google.android.material.floatingactionbutton.FloatingActionButton;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import androidx.appcompat.widget.Toolbar;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;

import com.babich.sqlite_project.Database.DatabaseQueryClass;
import com.babich.sqlite_project.Features.CreateStudent.Student;
import
com.babich.sqlite_project.Features.CreateStudent.StudentCreateDialogFragment;
import
com.babich.sqlite_project.Features.CreateStudent.StudentCreateListener;
import com.babich.sqlite_project.R;
import com.babich.sqlite_project.Util.Config;
import com.orhanobut.logger.AndroidLogAdapter;
import com.orhanobut.logger.Logger;

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

public class StudentListActivity extends AppCompatActivity implements
StudentCreateListener{

    private DatabaseQueryClass databaseQueryClass = new
                                                    DatabaseQueryClass(this);

    private List<Student> studentList = new ArrayList<>();

    private TextView studentListEmptyTextView;
    private RecyclerView recyclerView;
    private StudentListRecyclerViewAdapter studentListRecyclerViewAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_student_list);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        Logger.addLogAdapter(new AndroidLogAdapter());

        recyclerView = (RecyclerView) findViewById(R.id.studentRecyclerView);
        studentListEmptyTextView = (TextView)
        findViewById(R.id.emptyStudentListTextView);
```

```
        studentList.addAll(databaseQueryClass.getAllStudent());

        studentListRecyclerViewAdapter = new
                            StudentListRecyclerViewAdapter(this, studentList);
        recyclerView.setLayoutManager(new LinearLayoutManager(this,
                            LinearLayoutManager.VERTICAL, false));
        recyclerView.setAdapter(studentListRecyclerViewAdapter);

        viewVisibility();

        FloatingActionButton fab = (FloatingActionButton)
                                                    findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                openStudentCreateDialog();
            }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_main, menu);

        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        if(item.getItemId()==R.id.action_delete){

            AlertDialog.Builder alertDialogBuilder = new
                                                AlertDialog.Builder(this);
            alertDialogBuilder.setMessage(
                                "Ви дійсно хочете видалити всіх студентів?");
            alertDialogBuilder.setPositiveButton("Так",
                    new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface arg0, int arg1) {
                            boolean isAllDeleted =
                                      databaseQueryClass.deleteAllStudents();
                            if(isAllDeleted){
                                studentList.clear();

                       studentListRecyclerViewAdapter.notifyDataSetChanged();
                                viewVisibility();
                            }
                        }
                    });

            alertDialogBuilder.setNegativeButton("Hi",new
                                        DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    dialog.dismiss();
                }
            });

            AlertDialog alertDialog = alertDialogBuilder.create();
            alertDialog.show();
        }
```

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

    public void viewVisibility() {
        if(studentList.isEmpty())
            studentListEmptyTextView.setVisibility(View.VISIBLE);
        else
            studentListEmptyTextView.setVisibility(View.GONE);
    }

    private void openStudentCreateDialog() {
        StudentCreateDialogFragment studentCreateDialogFragment =
        StudentCreateDialogFragment.newInstance("Створити студента", this);
        studentCreateDialogFragment.show(getSupportFragmentManager(),
                                                    Config.CREATE_STUDENT);
    }

    @Override
    public void onStudentCreated(Student student) {
        studentList.add(student);
        studentListRecyclerViewAdapter.notifyDataSetChanged();
        viewVisibility();
        Logger.d(student.getName());
    }

}
```

Код класу StudentListRecyclerViewAdapter

```
package com.babich.sqlite_project.Features.ShowStudentList;

import android.content.Context;
import android.content.DialogInterface;
import androidx.appcompat.app.AlertDialog;
import androidx.recyclerview.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Toast;

import com.babich.sqlite_project.Database.DatabaseQueryClass;
import com.babich.sqlite_project.Features.CreateStudent.Student;
import
com.babich.sqlite_project.Features.UpdateStudentInfo.StudentUpdateDialogFragm
ent;
import
com.babich.sqlite_project.Features.UpdateStudentInfo.StudentUpdateListener;
import com.babich.sqlite_project.R;
import com.babich.sqlite_project.Util.Config;
import com.orhanobut.logger.AndroidLogAdapter;
import com.orhanobut.logger.Logger;

import java.util.List;

public class StudentListRecyclerViewAdapter extends
                                    RecyclerView.Adapter<CustomViewHolder> {

    private Context context;
    private List<Student> studentList;
    private DatabaseQueryClass databaseQueryClass;

    public StudentListRecyclerViewAdapter(Context context, List<Student>
```

```
        studentList) {
        this.context = context;
        this.studentList = studentList;
        databaseQueryClass = new DatabaseQueryClass(context);
        Logger.addLogAdapter(new AndroidLogAdapter());
    }

    @Override
    public CustomViewHolder onCreateViewHolder(ViewGroup parent,
            int viewType) {
        View view = LayoutInflater.from(context).inflate(
                                        R.layout.student_item, parent, false);
        return new CustomViewHolder(view);
    }

    @Override
    public void onBindViewHolder(CustomViewHolder holder, int position) {
        final int itemPosition = position;
        final Student student = studentList.get(position);

        holder.nameTextView.setText(student.getName());
        holder.registrationNumTextView.setText(String.valueOf(
                                            student.getRegistrationNumber()));
        holder.emailTextView.setText(student.getEmail());
        holder.phoneTextView.setText(student.getPhoneNumber());

        holder.crossButtonImageView.setOnClickListener(new
            View.OnClickListener() {
            @Override
            public void onClick(View view) {
                AlertDialog.Builder alertDialogBuilder = new
                                                AlertDialog.Builder(context);
                alertDialogBuilder.setMessage(
                                "Ви дійсно хочете видалити цього студента?");
                        alertDialogBuilder.setPositiveButton("Так",
                                new DialogInterface.OnClickListener() {
                                    @Override
                                    public void onClick(DialogInterface arg0,
                                    int arg1) {
                                        deleteStudent(itemPosition);
                                    }
                                });

                alertDialogBuilder.setNegativeButton("Hi",new
                                        DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                    }
                });

                AlertDialog alertDialog = alertDialogBuilder.create();
                alertDialog.show();
            }
        });

        holder.editButtonImageView.setOnClickListener(new
            View.OnClickListener() {
            @Override
            public void onClick(View view) {
                StudentUpdateDialogFragment studentUpdateDialogFragment =
                                    StudentUpdateDialogFragment.newInstance(
                                            student.getRegistrationNumber(),
                                    itemPosition, new StudentUpdateListener() {
```

```
                    @Override
                    public void onStudentInfoUpdated(Student student,
                        int position) {
                        studentList.set(position, student);
                        notifyDataSetChanged();
                    }
                });
                studentUpdateDialogFragment.show(((StudentListActivity)
                                    context).getSupportFragmentManager(),
                                                Config.UPDATE_STUDENT);
            }
        });
    }

    private void deleteStudent(int position) {
        Student student = studentList.get(position);
        long count = databaseQueryClass.deleteStudentByRegNum(
                                    student.getRegistrationNumber());

        if(count>0){
            studentList.remove(position);
            notifyDataSetChanged();
            ((StudentListActivity) context).viewVisibility();
            Toast.makeText(context, "Студента успішно видалено",
            Toast.LENGTH_LONG).show();
        } else
            Toast.makeText(context,
            "Студента не видалено. Щось пішло не так!",
                                        Toast.LENGTH_LONG).show();

    }

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

Код класу CustomViewHolder

```
package com.babich.sqlite_project.Features.ShowStudentList;

import androidx.recyclerview.widget.RecyclerView;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;

import com.babich.sqlite_project.R;

public class CustomViewHolder extends RecyclerView.ViewHolder {

    TextView nameTextView;
    TextView registrationNumTextView;
    TextView emailTextView;
    TextView phoneTextView;
    ImageView crossButtonImageView;
    ImageView editButtonImageView;

    public CustomViewHolder(View itemView) {
        super(itemView);

        nameTextView = itemView.findViewById(R.id.nameTextView);
```

```
            registrationNumTextView =
                        itemView.findViewById(R.id.registrationNumTextView);
            emailTextView = itemView.findViewById(R.id.emailTextView);
            phoneTextView = itemView.findViewById(R.id.phoneTextView);
            crossButtonImageView = itemView.findViewById(R.id.crossImageView);
            editButtonImageView = itemView.findViewById(R.id.editImageView);
        }
    }
```

Код класу StudentUpdateDialogFragment

```
package com.babich.sqlite_project.Features.UpdateStudentInfo;

import android.app.Dialog;
import android.os.Bundle;
import androidx.fragment.app.DialogFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;

import com.babich.sqlite_project.Database.DatabaseQueryClass;
import com.babich.sqlite_project.Features.CreateStudent.Student;
import com.babich.sqlite_project.R;
import com.babich.sqlite_project.Util.Config;


public class StudentUpdateDialogFragment extends DialogFragment {

    private static long studentRegNo;
    private static int studentItemPosition;
    private static StudentUpdateListener studentUpdateListener;

    private Student mStudent;

    private EditText nameEditText;
    private EditText registrationEditText;
    private EditText phoneEditText;
    private EditText emailEditText;
    private Button updateButton;
    private Button cancelButton;

    private String nameString = "";
    private long registrationNumber = -1;
    private String phoneString = "";
    private String emailString = "";

    private DatabaseQueryClass databaseQueryClass;

    public StudentUpdateDialogFragment() {
        // Required empty public constructor
    }

    public static StudentUpdateDialogFragment newInstance(long
        registrationNumber, int position, StudentUpdateListener listener){
        studentRegNo = registrationNumber;
        studentItemPosition = position;
        studentUpdateListener = listener;
        StudentUpdateDialogFragment studentUpdateDialogFragment = new
                                            StudentUpdateDialogFragment();
        Bundle args = new Bundle();
        args.putString("title", "Оновлення інформації");
```

```
        studentUpdateDialogFragment.setArguments(args);

        studentUpdateDialogFragment.setStyle(DialogFragment.STYLE_NORMAL,
                                                    R.style.CustomDialog);

        return studentUpdateDialogFragment;
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        View view = inflater.inflate(R.layout.fragment_student_update_dialog,
                                                        container, false);

        databaseQueryClass = new DatabaseQueryClass(getContext());

        nameEditText = view.findViewById(R.id.studentNameEditText);
        registrationEditText = view.findViewById(R.id.registrationEditText);
        phoneEditText = view.findViewById(R.id.phoneEditText);
        emailEditText = view.findViewById(R.id.emailEditText);
        updateButton = view.findViewById(R.id.updateStudentInfoButton);
        cancelButton = view.findViewById(R.id.cancelButton);

        String title = getArguments().getString(Config.TITLE);
        getDialog().setTitle(title);

        mStudent = databaseQueryClass.getStudentByRegNum(studentRegNo);

        if(mStudent!=null){
            nameEditText.setText(mStudent.getName());
            registrationEditText.setText(String.valueOf(
                                        mStudent.getRegistrationNumber()));
            phoneEditText.setText(mStudent.getPhoneNumber());
            emailEditText.setText(mStudent.getEmail());

            updateButton.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    nameString = nameEditText.getText().toString();
                    registrationNumber = Integer.parseInt(
                              registrationEditText.getText().toString());
                    phoneString = phoneEditText.getText().toString();
                    emailString = emailEditText.getText().toString();

                    mStudent.setName(nameString);
                    mStudent.setRegistrationNumber(registrationNumber);
                    mStudent.setPhoneNumber(phoneString);
                    mStudent.setEmail(emailString);

                    long id = databaseQueryClass.updateStudentInfo(mStudent);

                    if(id>0){
                        studentUpdateListener.onStudentInfoUpdated(mStudent,
                                                        studentItemPosition);
                        getDialog().dismiss();
                    }
                }
            });

            cancelButton.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
```

```
                    getDialog().dismiss();
                }
            });

        }

        return view;
    }

    @Override
    public void onStart() {
        super.onStart();
        Dialog dialog = getDialog();
        if (dialog != null) {
            int width = ViewGroup.LayoutParams.MATCH_PARENT;
            int height = ViewGroup.LayoutParams.WRAP_CONTENT;
            //noinspection ConstantConditions
            dialog.getWindow().setLayout(width, height);
        }
    }

}
```

Код інтерфейсу StudentUpdateListener

```
package com.babich.sqlite_project.Features.UpdateStudentInfo;

import com.babich.sqlite_project.Features.CreateStudent.Student;

public interface StudentUpdateListener {
    void onStudentInfoUpdated(Student student, int position);
}
```

Код класу Config

```
package com.babich.sqlite_project.Util;

public class Config {

    public static final String DATABASE_NAME = "student-db";

    //column names of student table
    public static final String TABLE_STUDENT = "student";
    public static final String COLUMN_STUDENT_ID = "_id";
    public static final String COLUMN_STUDENT_NAME = "name";
    public static final String COLUMN_STUDENT_REGISTRATION =
                                            "registration_no";
    public static final String COLUMN_STUDENT_PHONE = "phone";
    public static final String COLUMN_STUDENT_EMAIL = "email";

    //others for general purpose key-value pair data
    public static final String TITLE = "title";
    public static final String CREATE_STUDENT = "create_student";
    public static final String UPDATE_STUDENT = "update_student";
}
```

Повні вихідні коди проекту (разом з усіма допоміжними файлами та APK-файлом) доступні в публічному репозиторію: https://github.com/angelina-babych/Kursova-kHAI