Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмного інструменту управління базами даних в Android-застосунках

Виконав: студент IV курсу, групи СПс-42

спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Шеховцов В.І. (прізвище та ініціали)

Керівник (підпис) Цуприк Г.Б. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль - 2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
«__» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Студенту Шеховцову Вадиму Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного інструменту управління базами даних в Android-застосунках

Керівник роботи Цуприк Галина Богданівна, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «15» 04 2024 року № 4/7-344

2. Термін подання студентом завершеної роботи 26.06.2024р.

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1. Огляд предметної області.
2. Теоретична частина
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титулка. 2. Мета, задачі дослідження. 3. Процес розробки Room БД в Android застосунках 4. Результати порівняння альтернативних рішень. 5. Діаграма послідовності пропонованого програмного рішення. 6. ПЗ, яке використано в роботі. 7. Взаємодія компонентів програмного рішення. 8. Алгоритм аналізу компонентів Room БД 9. Діаграма компонентів генератора. 10. Діаграма взаємодії компонентів генератора. 11. Скріншоти роботи створеного ПЗ. 12. Схема взаємодії користувача з плагіном для Android Studio 13. Екрани створення відношень та зовнішнього ключа між таблицями 14. Висновки. Основні результати проведеного дослідження

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці			

7. Дата видачі завдання 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	16.04 – 18.04.24	*Виконано*
2.	Підбір джерел про управління базами даних		
	в Android-застосунках	19.04 – 22.04.24	*Виконано*
3.	Опрацювання джерел про управління		
	базами даних в Android- застосунках	23.04 – 29.04.24	*Виконано*
4.	Проведення дослідження щодо управління		
	базами даних в Android- застосунках	30.04 – 04.05.24	*Виконано*
5	Розроблення програмного коду	05.05 – 12.05.24	*Виконано*
6.	Оформлення розділу «Огляд предметної області»	13.05 – 18.05.24	*Виконано*
7.	Оформлення розділу «Теоретична частина»		*Виконано*
8.	Оформлення розділу «Практична частина»	19.05 – 24.05.24	*Виконано*
9.	Виконання завдання до підрозділу «Безпека		
	життєдіяльності, основи хорони праці»	25.05 – 31.05.24	*Виконано*
10.	Оформлення кваліфікаційної роботи		*Виконано*
11.	Нормоконтроль	01.06 – 05.06.24	*Виконано*
12.	Перевірка на плагіат	06.06 – 09.06.24	*Виконано*
13.	Попередній захист кваліфікаційної роботи	10.06 – 14.06.24	*Виконано*
14.	Захист кваліфікаційної роботи	27.06.24	

Студент _____ Шеховцов М.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота бакалавра містить: 52 сторінки, 21 рисунок, 36 джерел та 2 додатки.

БАЗА ДАНИХ, ІНТЕРФЕЙС КОРИСТУВАЧА, ПЛАГІН, СУТНІСТЬ, ТАБЛИЦЯ, ANDROID, DAGGER 2, ROOM

У кваліфікаційній роботі бакалавра проведено огляд предметної області, описано компоненти бази даних Room та процес її розробки , проаналізовано, альтернативні програмні рішення .

Описано принцип роботи програмного рішення, наведено архітектуру розробки, виконано аналіз структури бази даних. Забезпечено генерацію компонентів бази даних.

Розроблено користувальницький інтерфейс, наведено функціональні можливості. Докладно описано програмне рішення. Наведено приклад використання розробки для реальних даних.

DATABASE, USER INTERFACE, PLUG-IN, ENTITY, TABLE, ANDROID, DAGGER 2, ROOM

In the bachelor's thesis, an overview of the subject area was carried out, the components of the Room database and the process of its development were described, alternative software solutions were analyzed.

The principle of operation of the software solution is described, the development architecture is given, and the analysis of the database structure is performed. Generation of database components is ensured.

The user interface has been developed, and the functional possibilities have been given. The software solution is described in detail. An example of using development for real data is given.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

Room — бібліотека, яка спрощує роботу з базою даних Android SQLite. Це інтерфейс більш високого рівня абстракції, використання якого дає змогу зменшити кількість коду, зробити його чистішим і спростити роботу.

БД – база даних.

ОС – операційна система.

ПЗ – програмне забезпечення.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Можливість локального зберігання даних дає змогу мобільним програмам зберігати дані в БД пристрою, що надає підтримку функціональності програмного рішення в умовах автономності [1].

Система Android надає програмам кілька способів збереження даних на пристрої [2]:

– Preferences – для збереження простих типів даних у вигляді пари ключ-значення;

– Internal/External storage - використовуються, коли розробнику недостатньо Preferences для зберігання даних. Наприклад, коли потрібно зберегти різні файли мультимедійних файлів, документи та інші файли великих розмірів$

– Databases - використовується зберігання структурованих типів даних у пристрої [3].

Android - застосунки, що обробляють структуровані типи даних, мають значні переваги над локальним зберіганням цих даних:

– доступ до даних програми без Інтернету. Кешування структурованих типів даних дозволяє відображати дані для користувача і в автономному режимі, оскільки програма може отримати дані без Інтернету з локального сховища [4];

– швидкість відображення даних. Наявність збережених на пристрої даних дозволяє відображати інформацію, доки дані оновлюються з мережі. З локального сховища дані завантажуються швидко [5], через що користувач не повинен спостерігати довгі екрани завантаження в застосунках;

– зниження енергоспоживання мобільного застосунку [6].

Для обробки локальних структурованих даних в Android творці середовища Google розробили бібліотеку Room [7], котра є однією із частин Android - застосунків [8] і за рахунок підтримки розробників інтегрується з існуючими та майбутніми компонентами формування застосунків від Google. Через це бібліотека має велику популярність серед Android -розробників [9].

Для створення та підтримки Room бази даних в Android - застосунку необхідно писати шаблонний код для розробки компонентів та інтеграції, що збільшує тимчасові витрати на розробку проекту, а кількість програмного коду, котрий потрібно буде аналізувати в майбутньому для розвитку проекту, збільшить кількість тимчасових витрат на його підтримку.

Автоматизація генерації коду для компонентів БД Room та її інтеграції в проект дозволить скоротити трудовитрати на розробку та підтримку БД в Android - застосунках.

Метою роботи є розробка програмного інструментарію для автоматизації управління локальними БД в Android - застосаунках.

Для досягнення мети потрібно вирішити наступні завдання:

– створити компонент для створення таблиць БД;

– реалізувати рішення щодо генерації TypeConverters [10], які дозволять зберігати непримітивні типи даних;

– додати можливість генерації асинхронних (Coroutines, RxJava) базових запитів до таблиць;

– розробити компонент створення зв'язків між таблицями;

– створити компонент для аналізу існуючої БД програми, щоб дозволити користувачеві інтегрувати до неї нові сутності;

– написати компонент для інтеграції нових сутностей у код проекту за допомогою Dagger 2 [12];

– розробити плагін в Android Studio і об'єднати всі вищезгадані пункти в інтерфейсі для можливої роботи з ними користувачів.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Компоненти бази даних Room

БД Room в Android - застосунках складається з 3 основних компонентів [13]:

– Entity. Є таблицею у БД. Room створює таблицю для кожного класу, що має @Entity інструкцію, поля у класі відповідають стовпцям у таблиці [14];

– DAO. Інтерфейси, які відповідають визначення методів для доступу до Entity. Щоб створити метод, необхідно скористатися @Query анотацією нам сигнатурою методу, в якій користувач за допомогою SQL -операторів вказує, яку інформацію він хоче отримати з Entity [15];

– AppDatabase. Клас визначає конфігурацію БД і є основною точкою доступу застосунку до збережених даних. Клас містить у собі налаштування БД, номер її версії, список entities, в якому перераховані всі сутності, пов'язані з БД, абстрактні методи без аргументів, які повертають екземпляри DAO -класу.

Компоненти, необхідні визначення відносин між сутностями таблиці:

– проміжний клас даних. Цей клас даних моделює відносини "один до одного" та "один до багатьох" між Entity. Він містить у собі екземпляри Entity - класів, між якими створюються відносини, і @Relation [16] анотацію, в якій містяться назви стовпців батьківської та дочірньої таблиці за якими вони зв'язуються;

– асоціативна Entity. Ця сутність потрібна для опису ставлення "багато до багатьох". Зберігає у собі перехресні посилання стовпці сутностей, якими відбувається зв'язування [17].

Компоненти, необхідні для підтримки БД та її інтеграції до проекту:

– міграції. Після редагування структури вже існуючої БД в Android - застосунках необхідно змінити її версію, що призведе до видалення збережених даних. Щоб запобігти їх втрату, розробникам необхідно писати SQL - запити, які виконують оновлення даних [18];

– асинхронні запити DAO -класів. Щоб запити не блокували інтерфейс

користувача, доступ до БД не може здійснюватися в основному потоці. Це обмеження означає, що запити до БД мають бути асинхронними.

Для здійснення асинхронних запитів Room підтримує Kotlin Coroutines та RxJava, які дозволяють виконувати наступні 3 категорії запитів:

- One-shot запити на запис, які вставляють, оновлюють чи видаляють дані БД;
- One-shot запити на читання, які зчитують дані з вашої БД лише один раз і повертають результат з моментальним знімком БД у цей час;
- Observable запити на читання, які зчитують дані з вашої БД щоразу, коли базові таблиці змінюються, і видають нові значення, що відображають ці зміни [19].

– Dagger 2. Використовується для надання доступу до компонентів Room БД за допомогою патерна Dependency Injection в Android - застосунку.

Компоненти, що описують тип даних стовпців сутностей:

– TypeConverters. БД дозволяють зберігати лише примітивні типи даних, але Room надає можливість зберігання специфічних даних за допомогою TypeConverters. Для цього вони при виконанні операції вставки переводять тип даних користувача в String і зберігають в БД саме String. Під час операції читання, конвертери переводять об'єкт String в тип даних користувача. Варто зазначити, що для кожного специфічного типу даних доводиться писати вручну новий TypeConverter;

– анотації. Опис атрибутів Entity здійснюється за допомогою анотацій. Основні інструкції Room:

- @PrimaryKey - вказує первинний ключ Entity [20];
- @ColumnInfo - вказує ім'я атрибута Entity;
- @Embedded - для вкладення класів з полями в Entity [21];
- @ForeignKey - дозволяє створювати зовнішні ключі [22].

1.2 Опис процесу розробки бази даних Room

Процес розробки БД в Android - застосунку та її інтеграції в код проекту представлені на рис. 1.1.

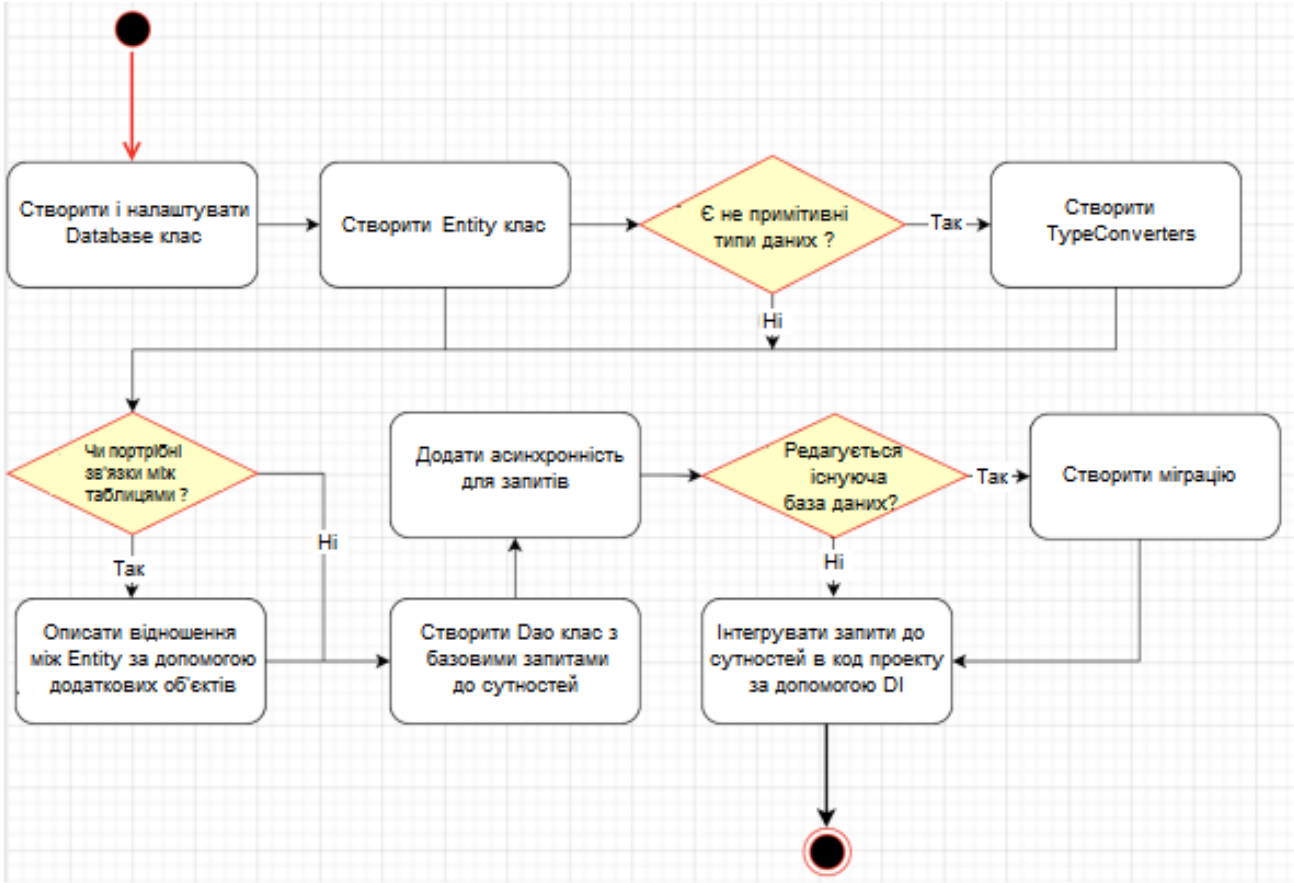

Рисунок 1.1 – Процес розробки Room БД в Android- застосунках

Перш за все розробнику необхідно створити Database (клас БД), в якому буде зберігатися інформація про версію БД, міграції, Dao і Entity -класи.

Далі необхідно створити Entity -клас, в якому будуть описані всі атрибути сутності.

Якщо були додані атрибути не примітивного типу даних, потрібно буде написати TypeConverter, який призначений для переведення формату атрибута в string при вставці об'єкта в таблицю, і назад із string у певний формат при вилученні об'єкта з таблиці.

Якщо потрібно описати відносини між сутностями БД, потрібно створити додаткові об'єкти. Що стосується відносин один до одного і один до багатьох, потрібно створити звичайний клас, а для відносин багато до багатьох доведеться створювати асоціативну таблицю.

Потім йде створення Dao -класу, який визначає запити до БД.

Далі йде створення асинхронних запитів до БД за допомогою Coroutines чи RxJava.

Наступним пунктом є створення міграції версії БД. Міграція потрібна для опису, як вже існуюча БД на пристрої користувача буде оновлюватися, при цьому не стерши повністю наявні дані.

Останнім етапом є інтеграція Dao -класу до компонентів програми за принципом Dependency Injection за допомогою Dagger 2.

Таким чином, з метою зменшити кількість однотипного коду, а також скоротити час на інтеграцію БД у застосунок, має сенс цей процес максимально автоматизувати та спростити шляхом автоматичної генерації коду.

1.3 Альтернативні програмні рішення

Візуальні інструменти для створення, редагування та розробки БД представлені в суміжних областях програмування. Наприклад, DB Browser [23], який дозволяє розробникам працювати з БД, сумісними з SQLite. Плагін використовує інтерфейс у вигляді електронних таблиць, за допомогою яких він може створювати та редагувати БД.

MySQL Workbench [24] - універсальний інструмент для розробників та адміністраторів БД. Є візуальним редактором, за допомогою якого розробники та адміністратори можуть генерувати, проектувати та керувати БД.

DbSchema [25] - інструмент для проектування та управління будь-якою БД SQL, NoSQL або Cloud. Надає користувачам достатньо простий графічний

інтерфейс, при допомозі якого можна отримати доступ до структури БД та керувати нею.

У сфері мобільних розробок на платформі iOS представлений фреймворк Core Data [26], який керує та зберігає дані у застосунку. За допомогою візуального редактора Core Data розробники можуть створювати таблиці, визначати типи даних та відносини між таблицями без написання ручного коду.

Перед порівнянням існуючих рішень в Android -розробці був визначений список функціональних можливостей, якими повинен мати інструмент для генерації та інтеграції БД:

– генерація класу налаштування БД (Database клас);
– генерація Entity сутностей БД;
– можливість створення зв'язків між сутностями БД;
– генерація Dao класів у базовими запитами до сутностей;
– створення міграцій за зміни версії БД;
– генерація Type Converters у разі використання непримітивних типів даних;
– інтеграція у код проекту за допомогою DI.

Серед аналогів є наступні програмні рішення.

SQLScout [27] - даний плагін генерує схеми для БД, або таблиці по sql - запиту. Таким чином, саме редагування і створення сутностей підтримується тільки через вбудований sql -редактор, що може призвести до помилок [28].

Room model builder [29] - плагін для генерації Entity -класів на основі Json - файлу, в якому користувач описує всі атрибути сутності.

Roomigrant [30] – допоміжна бібліотека для автоматичного створення міграції бібліотеки Android Room за допомогою генерації коду під час компіляції.

Представлені інструменти було розглянуто у межах критеріїв, які наведені вище, результати порівняння показані у табл. 1.1.

Таблиця 1.1 – Результати порівняння альтернативних рішень

	SQLScout	Room model builder	Roomigrant
Генерація та налаштування Database класу	-	-	+
Генерація Entity	+	+	-
Створення відносин між сутностями	-	-	-
Генерація Dao класів із запитами	-	-	-
Створення міграцій	+	-	+
Генерація TypeConverters	+	-	-
Інтеграція за допомогою DI	-	-	-

Отже, з проведеного аналізу видно, що жоден із наведених вище інструментів не задовольняє повністю описаним критеріям у контексті цієї проблеми.

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Принцип роботи програмного рішення

На основі аналізу існуючих рішень було запропоновано концепцію програмного засобу, що розробляється в рамках даної випускної кваліфікаційної роботи (рис. 2.1).

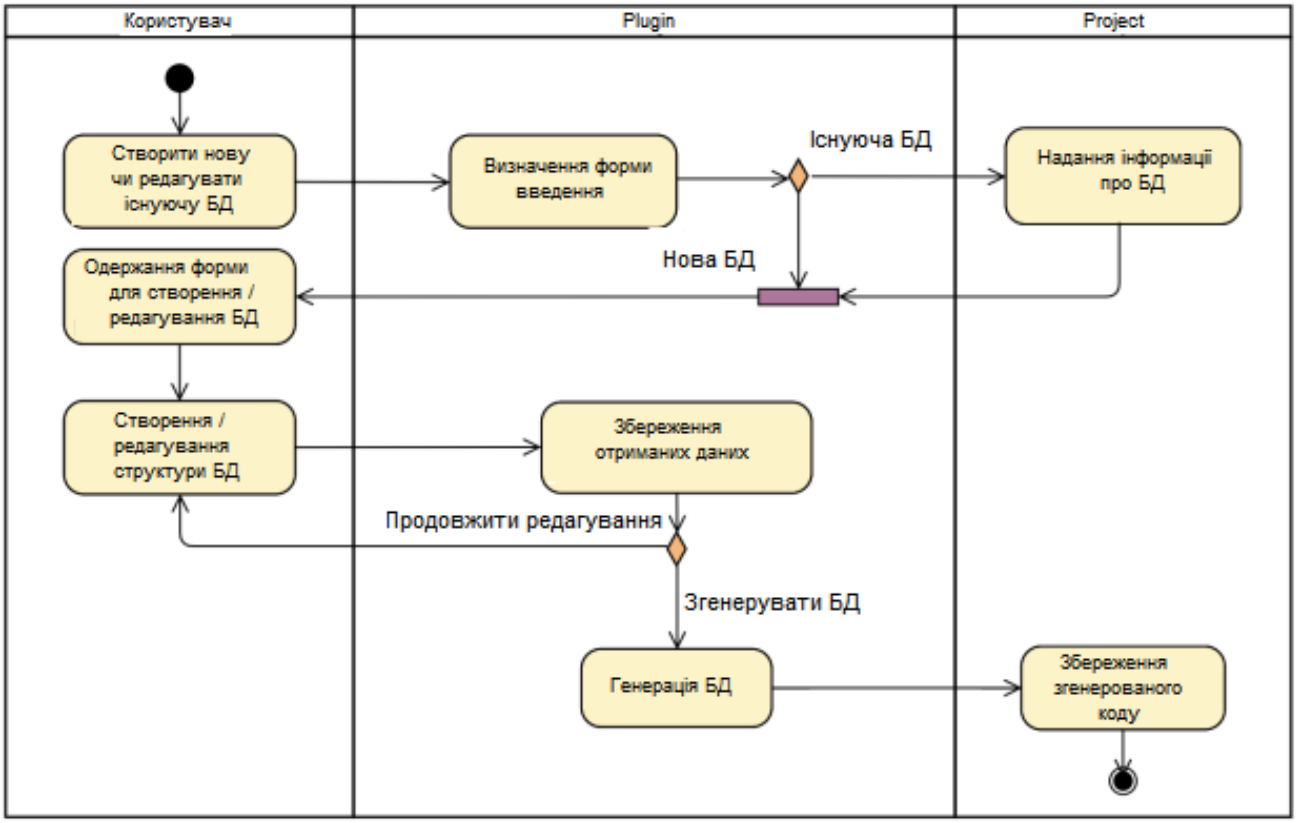

Рисунок 2.1 – Діаграма послідовності програмного рішення

Для початку роботи з програмним рішенням, користувачеві необхідно вибрати між редагуванням існуючої БД та створення нової. Якщо вибрано варіант з редагуванням існуючих компонентів, плагін надасть користувачеві інформацію про них, інакше БД Room буде створюватися з нуля. Далі розробник вносить зміни до структури БД, а введена ним інформація зберігається у плагіні, на основі якої він генерує код компонентів Room БД.

2.2 Архітектура програмного рішення

Для розробки програмного засобу було обрано мову програмування Kotlin. Компоненти Room БД, що генеруються, також реалізуються цією мовою. Загальну структуру програмного рішення можна розбити на 5 компонентів (рис. 2.2).

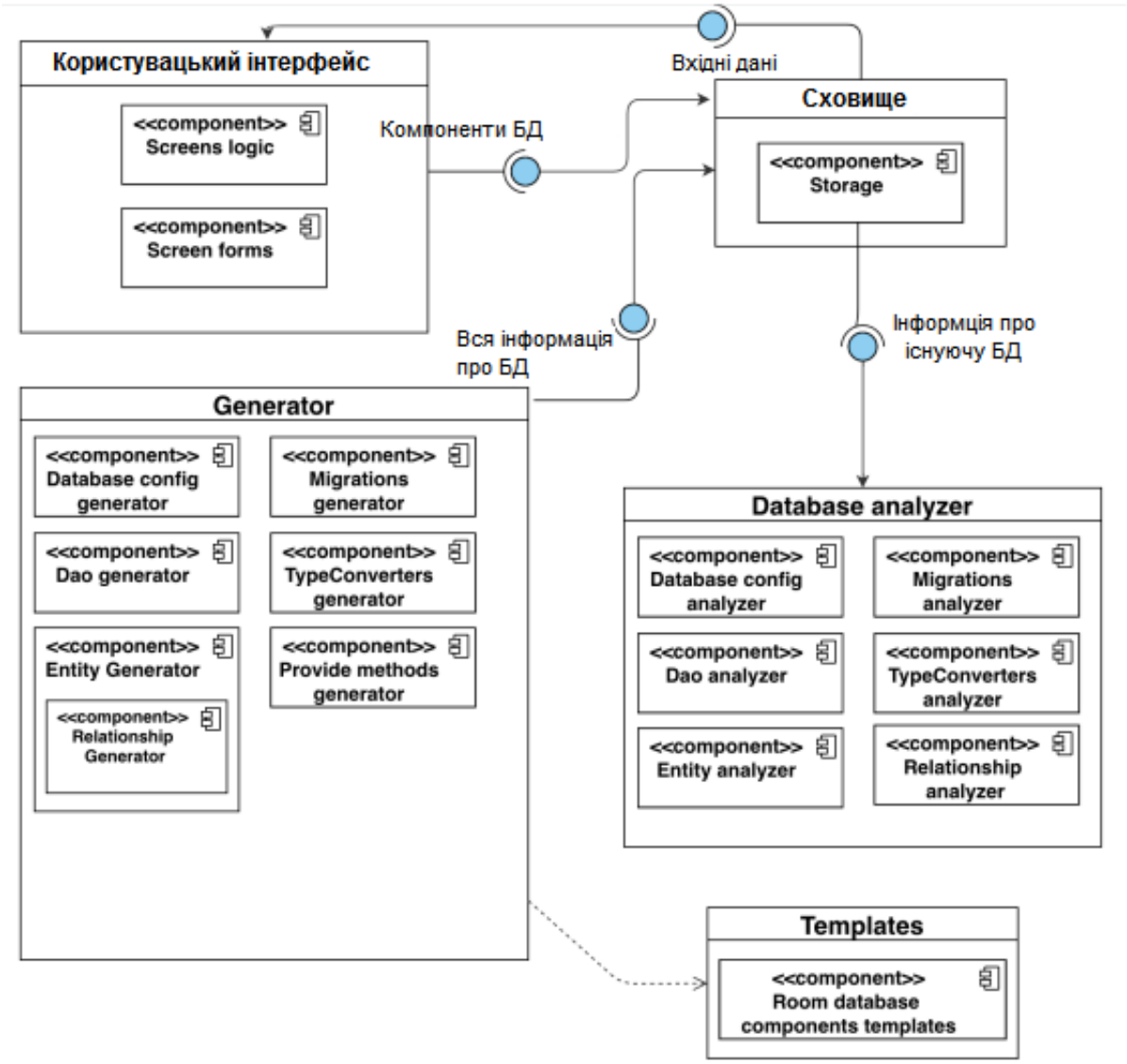

Рисунок 2.2 – Взаємодія компонентів програмного рішення

1. Інтерфейс користувача (UI) є набором екранів для взаємодії з користувачем, за допомогою яких відбувається опис і редагування генерованої БД Room для Android -проектів.

2. Storage зберігає в собі інформацію про існуючі сутності БД у проекті та створені в процесі взаємодії з програмним рішенням користувачем даних. Надає цю інформацію інтерфейсу користувача для відображення структури БД і генераторам для її генерації.

3. Database analyzer - компонент, який містить у собі логіку для зчитування існуючої БД Room Android -проекту і збереження її в Storage, щоб користувач міг взаємодіяти з нею за допомогою інтерфейсу користувача і редагувати її структуру.

4. Templates містить у собі файли-шаблони з розширенням ftl, що у директорії templates. Дані шаблони описують компоненти бази даних для генерації та подаються на вхід генераторам.

5. Generators разом з Database analyzer є одним із основних компонентів у програмному рішенні. Кожен генератор є незалежним від інших та відповідає за генерацію конкретного компонента структури БД Room. Генератори одержують на вхід ftl шаблони та дані про компоненти із Storage і видають згенеровані класи.

2.3 Аналіз структури бази даних

Аналіз структури існуючої Room БД необхідний для надання користувачеві інформації про неї та можливості її редагування.

На рис. 2.3 зображено алгоритм аналізу компонентів Room БД. Для аналізу структури, в першу чергу відбувається перетворення списку файлів "files" у введеній користувачем директорії, у список рядків для кожного файлу "filesLines".

Після цього плагін передає "filesLines" на вхід реалізаціям інтерфейсу Analyzer, які за допомогою методу findComponent виробляють фільтрацію "filesLines" на файли, за аналіз яких відповідає конкретна реалізація Analyzer "componentFiles". Після фільтрації вхідних даних викликається метод analyze для парсингу ”componentFiles”. Основними елементами для парсингу є інструкції

Room, завдяки яким плагін розуміє який компонент описує даний клас і які властивості він має.

Далі всі отримані дані упаковуються в класи моделі компонентів "componentModels", які надаються на вхід інтерфейсу користувача і редагуються.

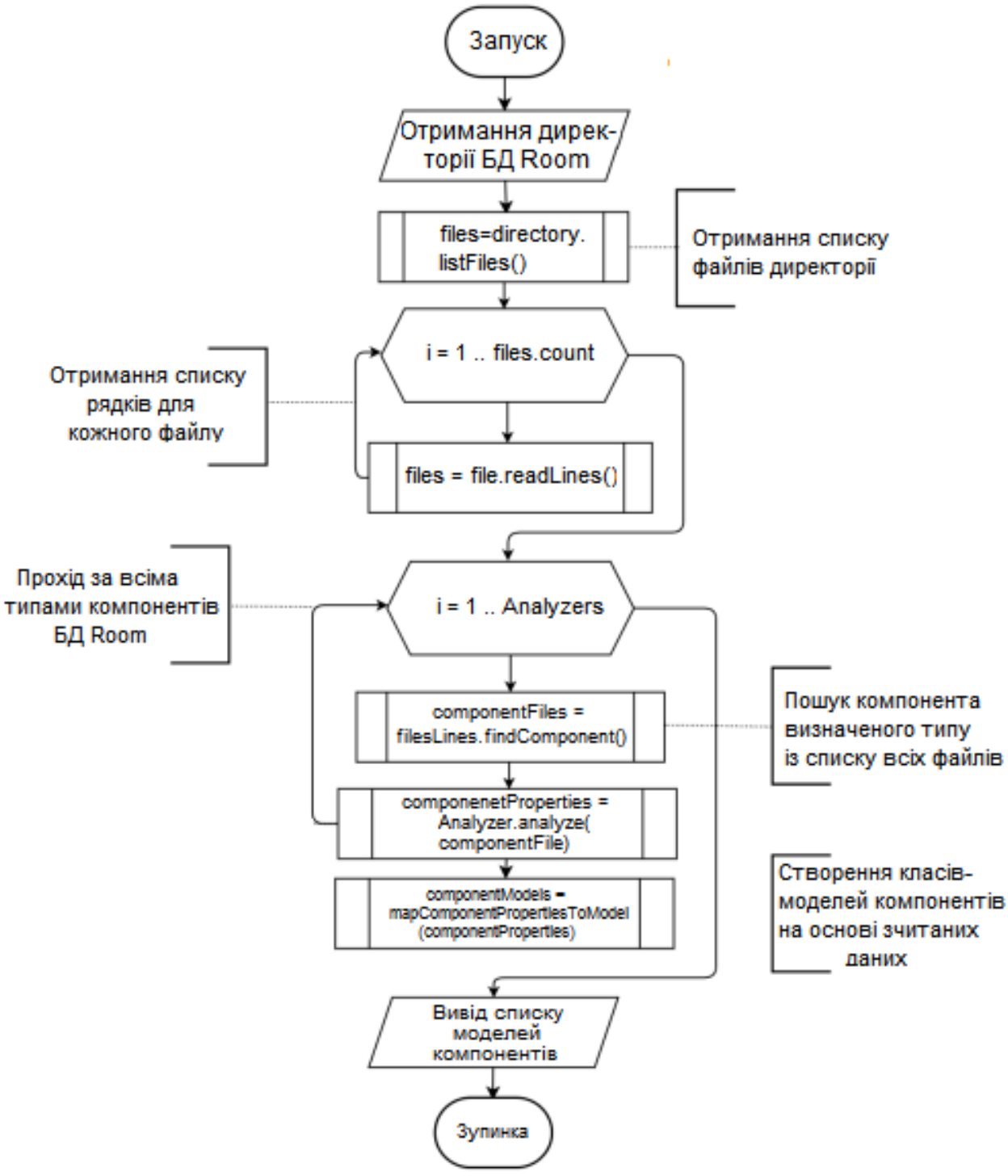

Рисунок 2.3 – Алгоритм аналізу компонентів Room БД

Як видно з діаграми на рис 2.4, основним компонентом є Analyzer, який містить у собі залежність від інших класів аналізаторів системи. Його основна функція полягає у передачі даних для аналізу всім класам та збереження отриманих даних у Storage. У момент своєї ініціалізації проходить по всіх залежних генераторах і викликає у них метод analyze.

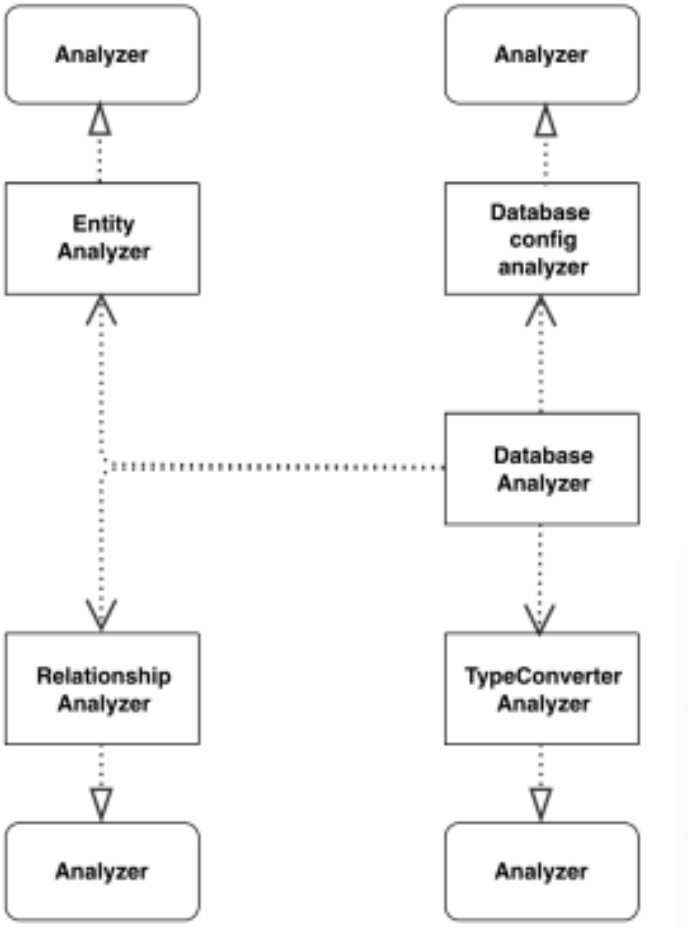

Рисунок 2.4 – Діаграма компонентів аналізатора

Кожен аналізатор реалізує метод analyze, який описаний в інтерфейсі DatabaseAnalyzer та відповідає за реалізацію парсингу окремого компонента БД Room.

DatabaseConfigAnalyzer призначений для парсингу класу конфігурації БД, з якого плагін отримує список імен всіх Entity таблиць за рахунок списку entities, вказаного в класі, та Dao -інтерфейсів із запитами до них, список яких міститься у

вигляді абстрактних методів у класі конфігурації. Після цього аналізатори даних компонентів розглядатимуть лише файли з такими іменами. Це зроблено для того, щоб не включати до структури зайві сутності. Клас конфігурації БД виявляється за допомогою інструкції @AppDatabase. Крім інформації про її компоненти, з неї ми отримуємо її версію, яка вказана в анотації @AppDatabase та ім'я.

EntityAnalyzer використовується для парсингу існуючих Entity таблиць БД Room. Аналізатор визначає, у якому файлі лежить таблиця з допомогою інструкції @Entity. Далі йде пошук стовпців таблиці, які є властивостями kotlin data class і вказуються анотацією @ColumnInfo, де константним значенням позначається ім'я стовпця. Наступним етапом є пошук первинних ключів, вони налаштовуються за допомогою анотації @PrimaryKey та властивості autoIncrement.

Relationship Analyzer після парсингу таблиць БД визначає відносини між ними і створює зовнішні ключі. Зовнішні ключі будуються на основі їх опису в Entity інструкцією @ForeignKey. Таблиця, з якою зв'язується поточна, вказується в параметрі entity, з нього плагін бере ім'я та шукає його вже готових моделях таблиці, які є результатом роботи EntityAnalyzer. Інформація про стовпці батьківської та дочірньої таблиці міститься в параметрах parentColumns і childColumns відповідно. Параметри onUpdate та onCascade вказують на стратегію обмеження CASCADE, NO_ACTION, RESTRICT, SET_DEFAULT або SET_NULL.

Для визначення існуючих відносин один до одного, один до багатьох і багато до багатьох, визначаються класи з їх описом завдяки анотації @Relation, в якій вказуються parentColumn та entityColumn параметри, що містять інформацію про колонки таблиць, за якими будуються відносини. Самі таблиці описуються з допомогою анотації @Embedded і властивостями data класу kotlin. AssociatedBy параметр анотації @Relation дозволяє визначити назву асоціативної таблиці, яка є основним компонентом зв'язку багато до багатьох.

TypeConvertersAnalyzer використовується для визначення існуючих допоміжних класів для зберігання не примітивних типів даних, щоб на етапі генерації відредагованої структури БД можна було використовувати готові TypeConverters, а не створювати нові. Парсинг відбувається за тими класами, які

містять анотацію @TypeConverter, а тип даних визначається типами, що повертаються в його методах.

2.4 Генерація компонентів бази даних

У розробленому програмному рішенні генератори є класами, що реалізують інтерфейс Generator. Всі вони призначені для створення конкретного компонента БД Room в Android -проектах, на основі даних, введених користувачем через інтерфейс (UI).

Діаграма класів генераторів системи зображена на рис. 2.5.

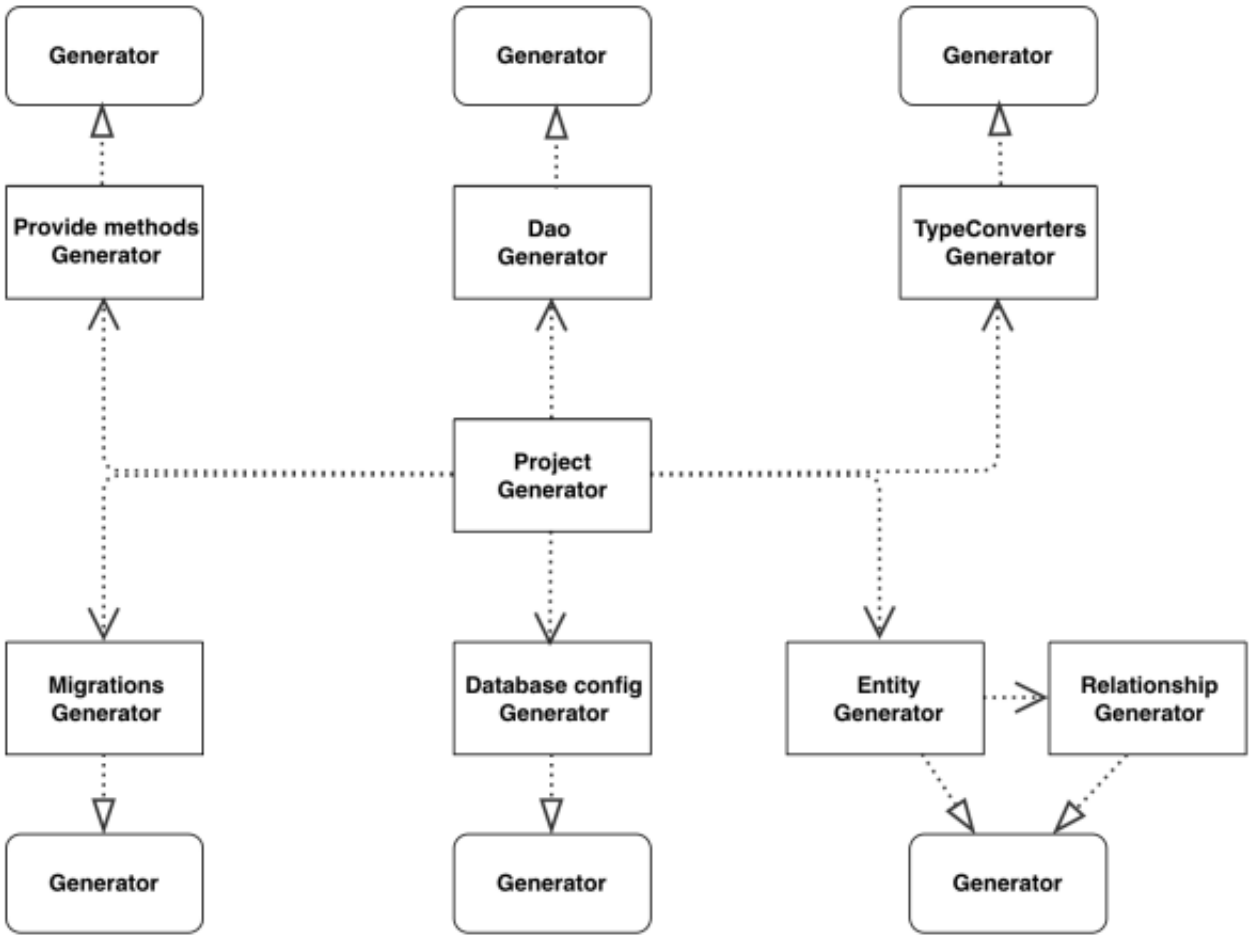

Рисунок 2.5 – Діаграма компонентів генератора

Для створення компонентів всі генератори використовують Freemarker [31] - механізм шаблонів на основі Java, який використовується для генерації текстового виведення. Для цього на вхід Freemarker подається текстовий файл (шаблон), що містить бажаний висновок. Також шаблони містять наповнювачі, такі як ${name}, і навіть деяку логіку, таку як умовні вирази, цикли тощо. Крім шаблону на вхід потрібно передати підготовлені дані, що відповідають заповнювачам. Freemarker динамічно вставляє ці дані та виходить динамічно заповнений документ.

На рис. 2.6 зображено алгоритм генерації компонентів Room БД. На самому початку генератор отримує на вхід моделі компонентів, на основі яких генеруватимуться дані, створює об'єкт конфігурації і викликає метод generate у кожної реалізації інтерфейсу Generator.

Насамперед цей метод створює об'єкт шаблону Freemarker за допомогою виклику методу getTemplate у об'єкта конфігурації. Метод getTemplate приймає на вхід параметр “templatePath”, який містить інформацію про шлях шаблону, на основі якого буде генеруватися файл.

Після налаштування шаблону, генератор створює об'єкт “templateDataMap” - структуру даних виду HashMap<String, Any>, де String - назва ключа, яким ми можемо звернутися до неї з шаблону для підстановки даних у заповнювачі ${key}, Any - тип значення , яка буде підставлена на місце відповідного дзвінка всередині шаблону. Значення змінних можуть бути простими рядками, числами та примітивними значеннями, а також списками, картами або довільними об'єктами Kotlin/Java.

Далі метод створює екземпляр класу FileWriter , який використовується для потокового запису символів у файл, шлях до якого ми вказуємо за допомогою параметра “targetFilePath”, і викликає метод process у об'єкта шаблону для генерації файлу.

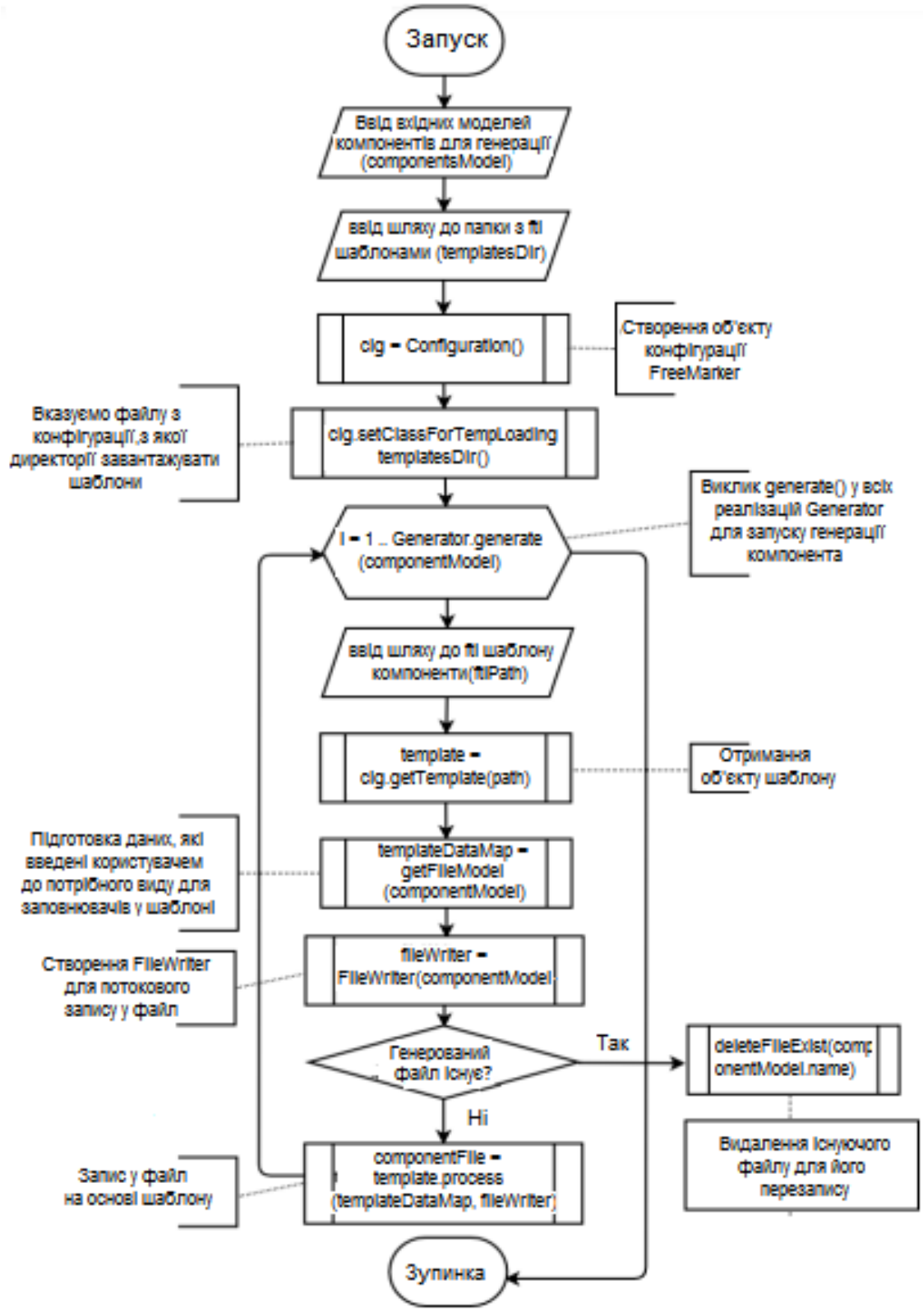

Рисунок 2.6 – Алгоритм генерації компонентів Room БД

Як видно з діаграми, зображеної на рис. 2.7, основним компонентом є Database Generator, який містить у собі залежність від інших генераторів системи.

Його основна функція полягає у передачі даних про компоненти, які потрібно створити всім генератором. У момент своєї ініціалізації проходить по всіх залежних генераторах і викликає у них метод generate.

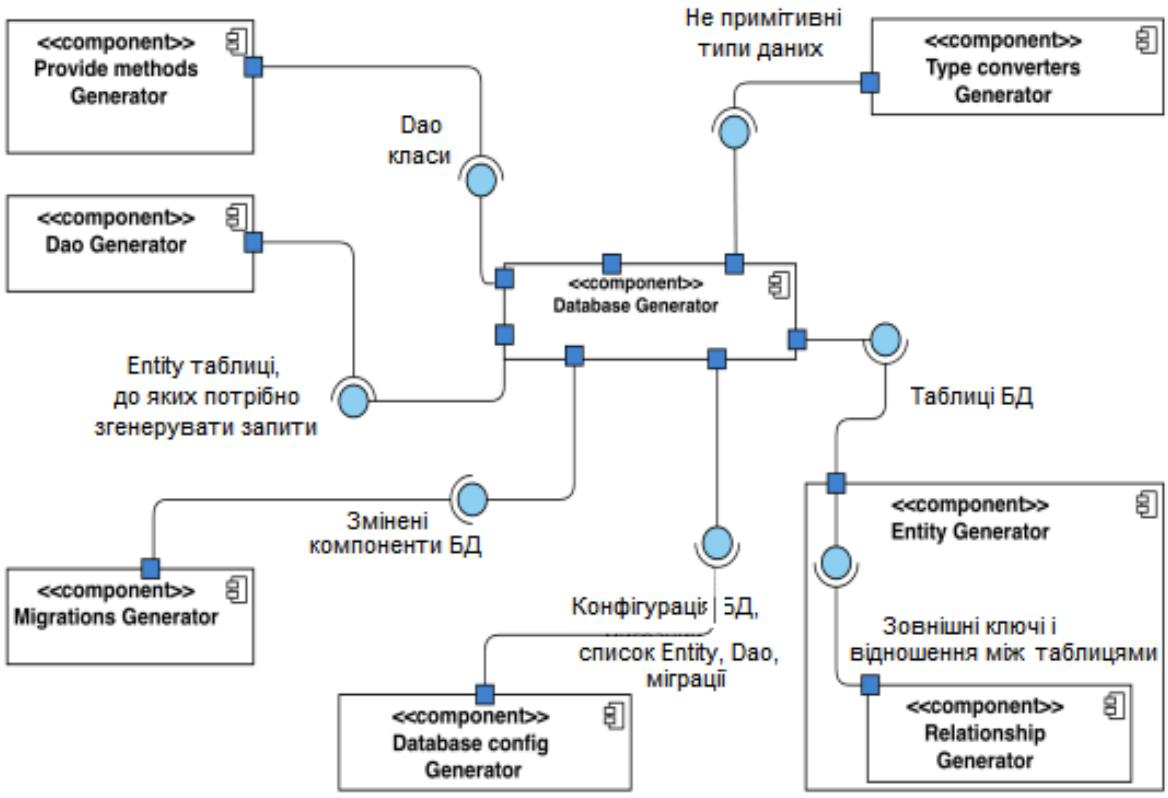

Рисунок 2.7 – Діаграма взаємодії компонентів генератора

Database Config Generator відповідає за генерацію AppDatabase класу конфігурації БД Room.

Entity Generator використовується для створення таблиць БД. Після проходження процесу генерації таблиць він звертається до Relationship Generator, якому передає інформацію про таблиці, які мають бути пов'язані різними відносинами.

Dao Generator призначений для створення Dao -інтерфейсів із запитами до Entity -таблиць БД. На вхід отримує інформацію по таблицях та їх відносинам, для яких потрібно створити запити, та тип асинхронної обгортки для них.

TypeConverters Generator використовується для генерації допоміжних класів

для зберігання непримітивних типів даних у таблиці. На вхід отримує інформацію про не примітивні типи даних і до яких таблиць та запитів повинні додаватися допоміжні класи.

Migration Generator використовується для генерації міграцій, які зберігають цілісність існуючої БД на Android -пристроях після оновлення структури існуючої БД Room. Для цього отримує інформацію про версію БД та всі зміни її структури, які зробив користувач. Компонент сортує змінені дані Added, Edited, Deleted. Далі виходячи з групи будується SQL запит із додаванням, редагуванням або видаленням сутності з структури БД Room. Після створення SQL запиту, він подається на вхід шаблонизатору, який створює об'єкт міграції міграцію.

Provide methods Generator створює у зазначеній користувачем директорії або у певному класі provide методи для доступу до запитів та таблиць БД Room.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Користувальницький інтерфейс

Для створення інтерфейсу був обраний Swing Framework [32], який володіє потужною множиною компонентів, що забезпечують високий рівень налаштувань, широкі функціональні можливості, і використовується для створення віконних застосунків.

За допомогою Swing були розроблені такі екрани для плагіна:

1. Ініціалізаційний екран.

На цьому екрані користувач бачить кнопки "Open exist" і "Create new", створені за допомогою компонента JButton, що представляє кнопки Swing. Кнопка Open exist переводить користувача на екран з вибором існуючої БД, а кнопка Create new дозволяє йому створити нову БД.

2. Вибір наявної БД (рис. 3.1).

Після натискання на кнопку Open exist, користувачу стає доступним діалогове вікно з вибором директорії, де зберігається існуюча БД Room. У Swing діалогове вікно представляє клас JFileChooser, який дозволяє отримати шлях до вибраного файлу або директорії. JFileChooser має кілька фільтрів для вибору (fileSelectionMode): FILES_ONLY, DIRECTORIES_ONLY та FILES_AND_DIRECTORIES.

У нашому випадку був застосований фільтр DIRECTORIES_ONLY, оскільки пошук компонентів існуючої БД здійснюється за каталогом, а не в конкретному файлі.

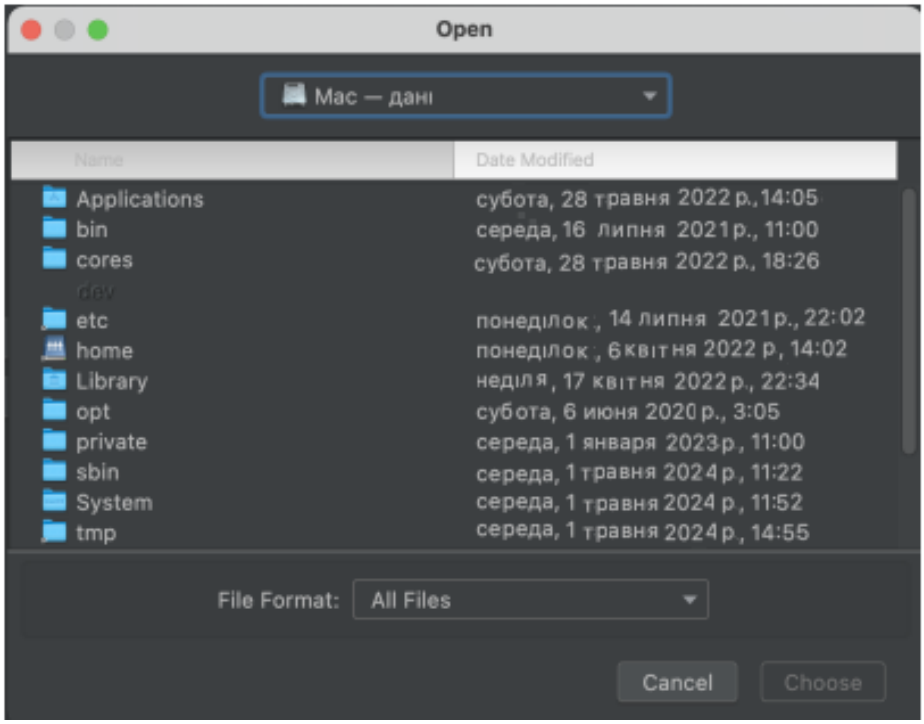

Рисунок 3.1 – Вибір існуючої БД

3. Екран створення імені БД.

Якщо користувач натиснув “Create new” на екрані ініціалізації, він потрапляє на екран введення імені БД. Ім'я вводиться в текстове поле, створене за допомогою Swing JTextField, який необхідний для введення тексту. Кнопка “Enter” служить для підтвердження вибору імені та переходу до наступного екрану.

4. Екран вибору директорії для створення компонентів БД Room.

На цьому екрані користувач отримує доступ до вибору директорії, в якій буде здійснено генерацію БД. Для цього потрібно вибрати елемент Select directory, компонента JComboBox, який являє собою список, що випадає. JComboBox надає можливість відслідковувати вибір елементів зі списку за допомогою методу addItemListener, за допомогою якого після вибору елемента Select directory відкривається діалогове вікно JFileChooser, де потрібно вибрати директорію для генерації.

Вибір директорії підтверджується через клікання на кнопці Enter, після чого зберігається в Storage.

5. Список таблиць БД (рис. 3.2).

Основними компонентами цього екрану EntityListForm є JScrollPane та JList. JList у Swing використовується для відображення даних у вигляді списку . Дані в компонент поставляються за допомогою DefaultListModel, який зберігає в собі список елементів і, у разі зміни даних даних, повідомить про це JList для його оновлення. Також за допомогою методу setSelectionMode встановлюється режим вибору елементів. Оскільки нам потрібно вибирати для відкриття таблиці зі списку лише одну з них, було встановлено режим SINGLE_SELECTION. Для можливості скролла списку компонент JList встановлюється в JScrollPane.

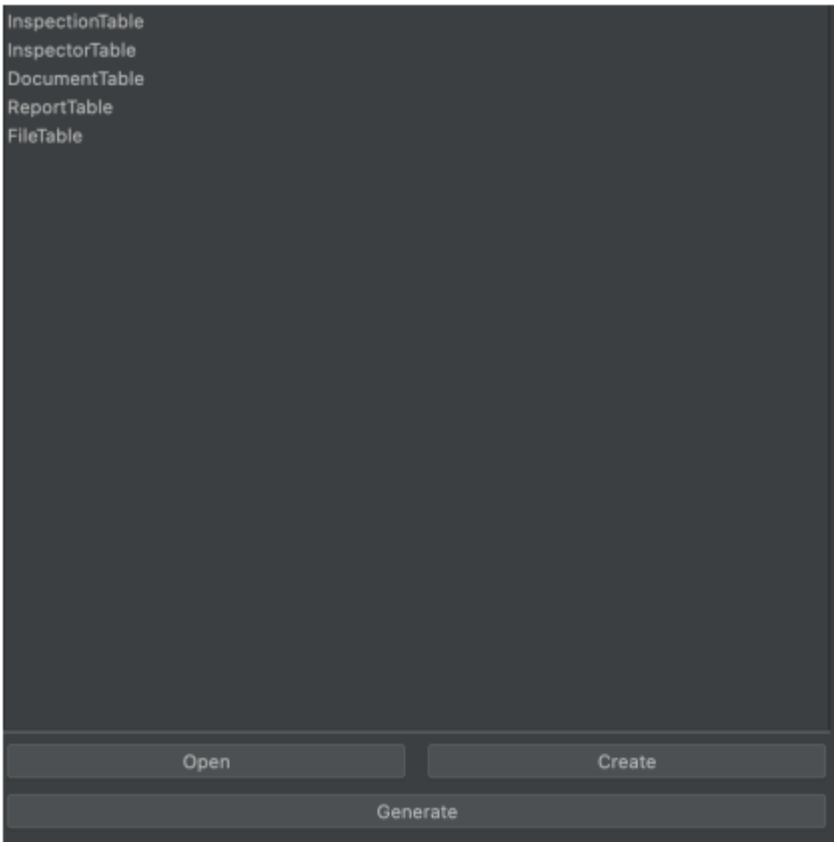

Рисунок 3.2 – Список таблиць БД

За допомогою кнопки Open користувач може відкривати вибрану ним таблицю, а за допомогою Create відкрити вікно для створення нової. Кнопка Generate запускає процес генерації описаних користувачем компонент БД Room.

6. Екран таблиці БД (рис. 3.3).

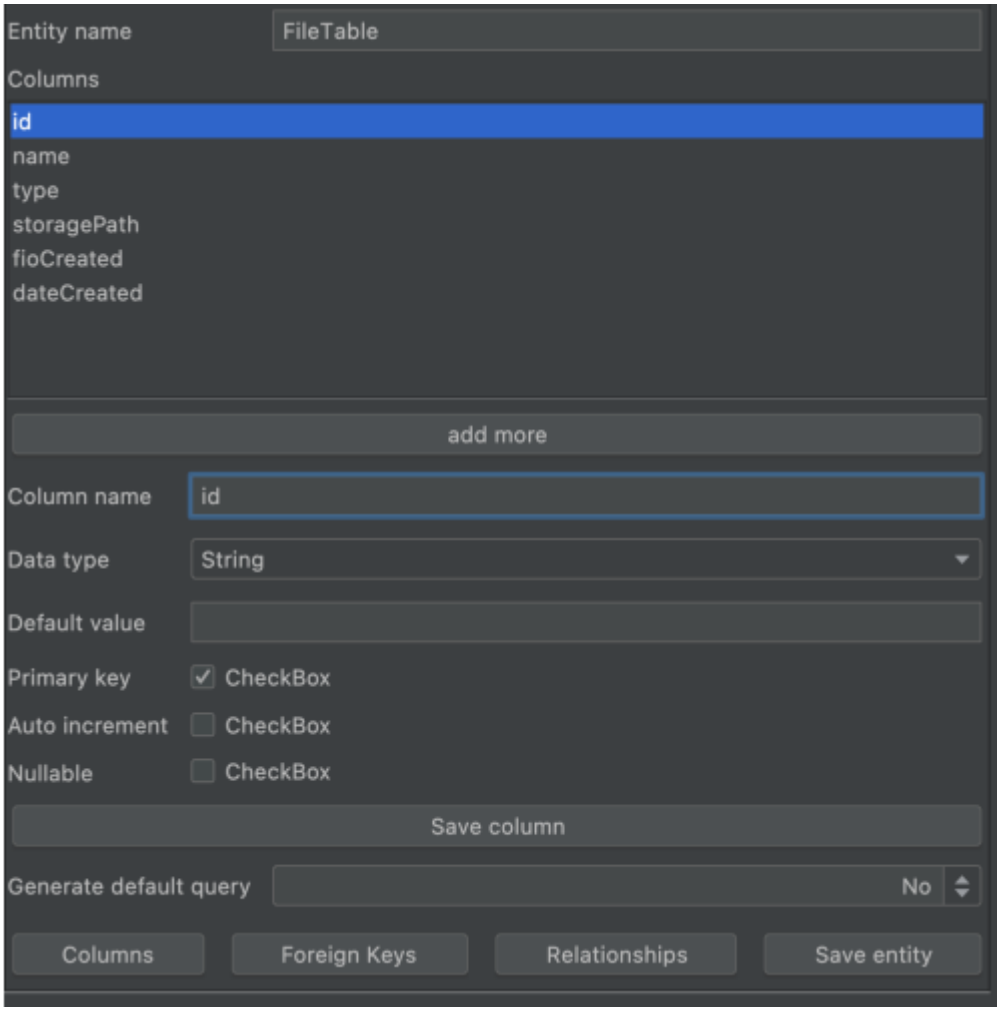

Рисунок 3.3 – Екран таблиці БД

На цьому екрані користувач може редагувати існуючу або сформувати нову таблицю і складається з наступних компонентів:

– поле “Entity name” містить інформацію про ім'я таблиці, яке використовується для генерації;

– у списку columnsList JList представлені колонки таблиці БД. Режим у списку встановлений SINGLE_SELECTION, а за допомогою ListSelectionListener, який дозволяє відстежувати елементи, що вибираються, після вибору певної колонки плагін відображає інформацію про неї в елементах нижче;

– кнопка “Add more” додає нову колонку до списку;

– Column name JTextField містить інформацію про ім'я колонки;

– Default value JTextField потрібний для введення значення за промовчанням колонки. Наприклад, для типу Int може бути 0;

– Data type JComboBox зберігає інформацію про вибраний тип даних для колонки. Після натискання на JComboBox користувач побачить список, що випадає, в якому для вибору йому будуть доступні різні примітивні типи даних (Int, String ітд) і їх масивні варіації (List<Int>, List<String> ітд). Також користувачеві доступний вибір не примітивних типів даних, що описуються в Data class Kotlin файлах. Для цього при натисканні в списку на Custom відобразиться вікно JFileChooser з встановленим фільтром FILES_ONLY, який дозволяє вибирати тільки файли;

– Primary key – чи є колонка первинним ключем. Auto_increment - чи потрібно генерувати значення за промовчанням для primary key;

– Nullable - чи тип даних колонки є спеціальним типом Kotlin [33]. Так як перелічені властивості колонки можуть бути або true, або false, було прийнято рішення використовувати для заповнення інформації про них JCheckBox, який має лише 2 стани (активний або неактивний).

Кнопка Save column дозволяє зберігати введену користувачем інформацію про колонку.

7. Екран створення зовнішнього ключа (рис. 3.4).

Після натискання на кнопку “Foreign Key”s користувач перейде на екран створення/редагування зовнішніх ключів. Для ідентифікації зовнішніх ключів користувачеві необхідно заповнити його ім'я у полі “Foreign key name”. Далі у списках, що випадають JComboBox відбувається налаштування зовнішнього ключа. У списку “Parent table” вказується таблиця (головна), з якою вихідна таблиця (залежна) буде зв'язуватися. Після вибору головної таблиці, список із її стовпцями встановлюється у “Parent column” JComboBox.

Далі заповнюються колонки таблиць у списках column і Parent column, за якими відбуватиметься зв'язування і накладатиметься обмеження, які користувач вказує в onUpdate і onDelete списках, які застосовуються щодо знищення та редагування зв'язаного рядка відповідно із головної таблиці.

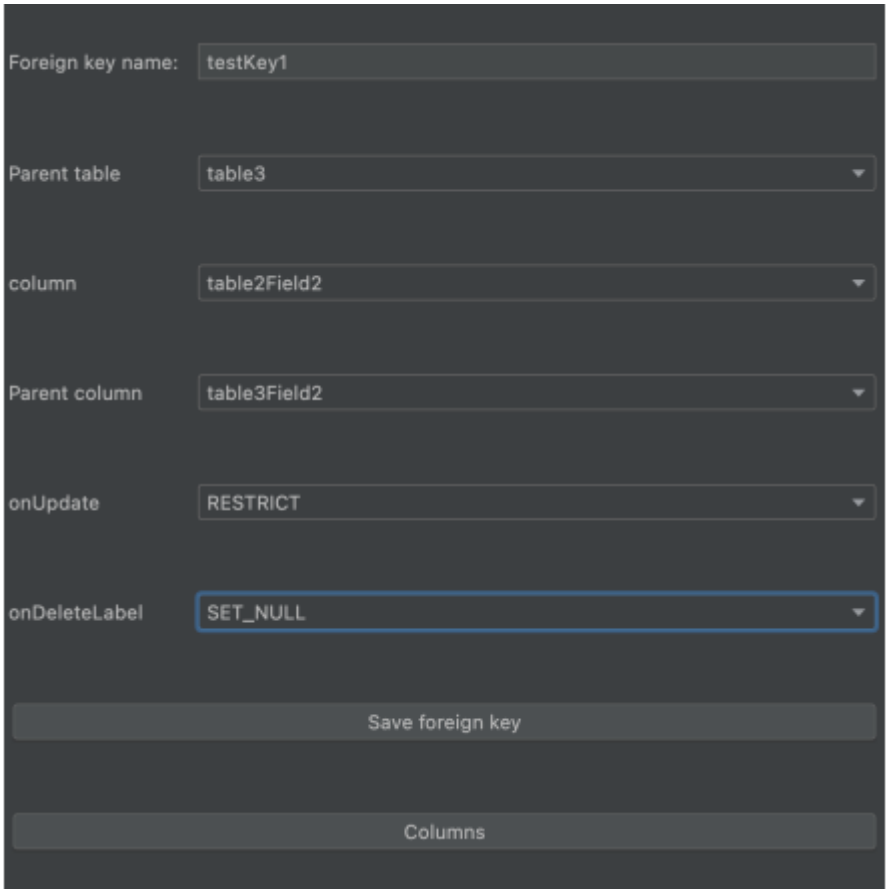

Рисунок 3.4 – Екран створення зовнішнього ключа

Діями можуть бути такі опції, котрі будуть застосовані в дочірній таблиці при зміні головної таблиці:

– CASCADE – автоматично видаляє рядки із залежної таблиці;
– SET NULL – замінює рядки на null у залежній таблиці;
– RESTRICT - відхиляє видалення чи зміну у головній таблиці;
– SET DEFAULT – встановлює значення за замовчуванням.

Далі користувач може зберегти створений або відредагований зовнішній ключ при натисканні на кнопку “Save foreign key”

8. Екран редагування відношень між таблицями (рис. 3.5).

Після натискання на кнопку "Relationship" (рис. 3.3) користувач переходить на екран створення/редагування відношень таблиці.

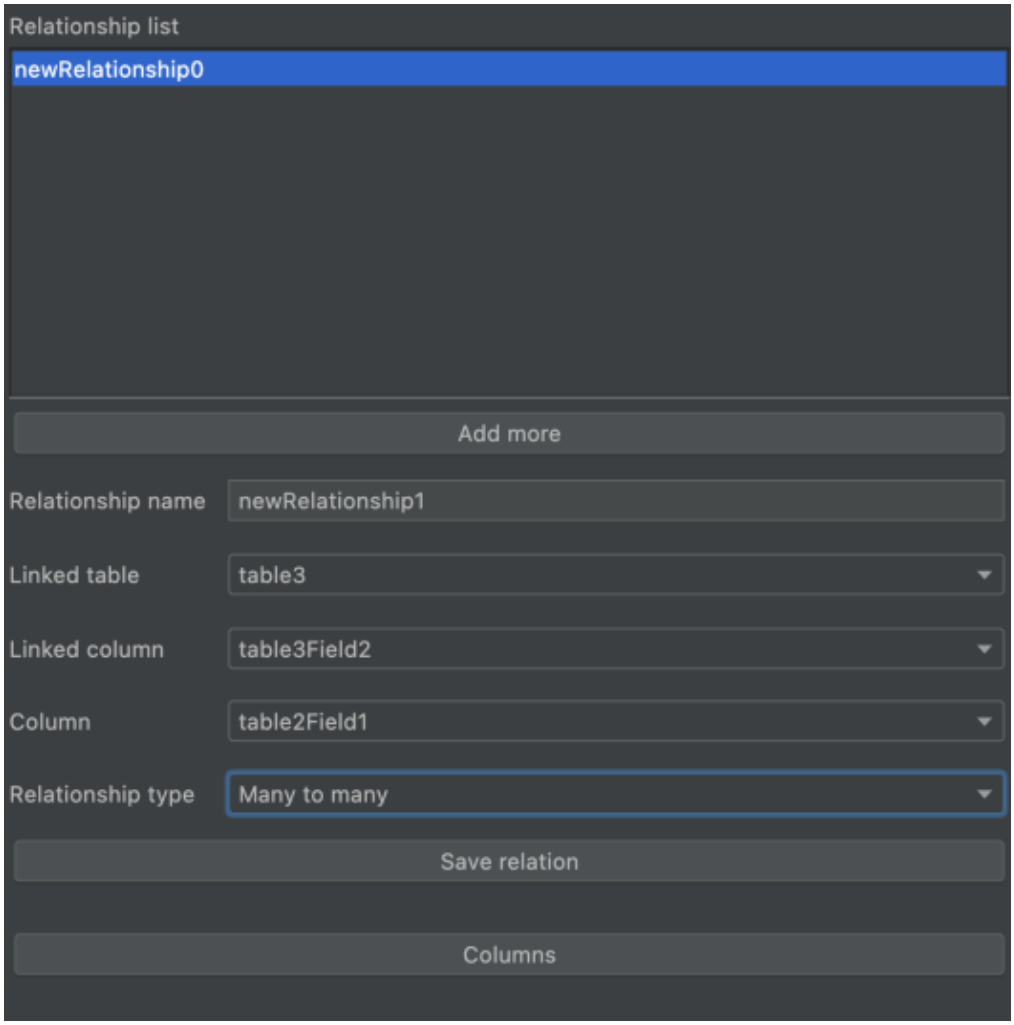

Рисунок 3.5 – Екран редагування відношень між таблицями

Для ідентифікації відношень потрібно задати ім'я, після чого за допомогою списків, що випадають, заповнюється інформація про зовнішній ключ. Поле "Linked table" містить інформацію про те, з якою таблицею ми хочемо пов'язати поточну відношеннями. Після вибору пов'язаної таблиці, список із її стовпцями встановлюється у “Linked column” JComboBox. Інформація про стовпці, що представляють зв'язок між двома таблицями міститься в списках “Linked column” і “column”, де перша вказує на стовпець зв'язаної таблиці, а друга на стовпець поточної таблиці. У списку “Relationship type” користувач вибирає один із типів відносин між зазначеними таблицями:

– One to one - тут елементу однієї сутності можна поставити у відповідність лише один елемент іншої;

– One to many/Many to one - у цьому типі зв'язку кілька рядків із дочірньої

таблиці залежать від одного рядка в батьківській таблиці;

– Many to many - при цьому типі зв'язку один рядок з таблиці А може бути пов'язаний з множиною рядків з таблиці В. У свою чергу один рядок з таблиці може бути пов'язаний з множиною рядків з таблиці А.

3.2 Функціональні можливості

3.2.1 Опис програмного рішення

Розроблене програмне рішення є плагіном для середовища розробки Android Studio, який є графічним інтерфейсом з набором екранів для редагування і створення структури БД Room в Android - застосунках.

Взаємодія користувача з плагіном відповідає схемі, яка представлена у вигляді UML- діаграми послідовності на рис. 3.6.

Розглянемо докладніше цю діаграму. Першою дією є "Запуск IDE Android Studio", на даному етапі користувач повинен відкрити існуючий або створити новий проект в Android Studio.

Далі йде крок "Встановлення плагіна", на якому користувачеві необхідно завантажити та встановити плагін. Для досягнення цього результату є кілька варіантів:

– завантажити плагін у середовищі розробки Android Studio у розділі Plugins, після чого він буде автоматично встановлений;

– завантажити плагін зі стороннього джерела в форматі .jar . Далі відкрити Диспетчер плагінів в Android Studio і завантажити в нього завантажений .jar файл.

Рисунок 3.6 – Схема взаємодії користувача з плагіном для Android Studio

Після відкриття плагіна користувач переходить до кроку "Редагування/створення структури бази даних", на якому йому необхідно взаємодіяти з інтерфейсом користувача для роботи з БД.

Наступним кроком є “Генерація компонентів бази даних”, до якого користувач переходить після натискання кнопки “Generate”. Після цього буде відбуватися генерація компонентів БД Room і перехід до останнього кроку "Отримання згенерованих компонентів бази даних", на якому користувачі стають доступними всі компоненти БД Room, що згенерували.

3.2.2 Приклад використання програмного рішення

Для демонстрації використання програмного рішення відкриємо за допомогою плагіна директорію з існуючою БД Room (Додаток А). Далі вибираємо робочу директорію, в яку хочемо зберегти файли, що згенеровані плагіном.

На екрані списку таблиць БД відображається таблиця InspectionInfoLocal, що вже є в структурі проекту, яка зберігає в собі інформацію про проведення інспекції деякого заводу (рис. 3.7).

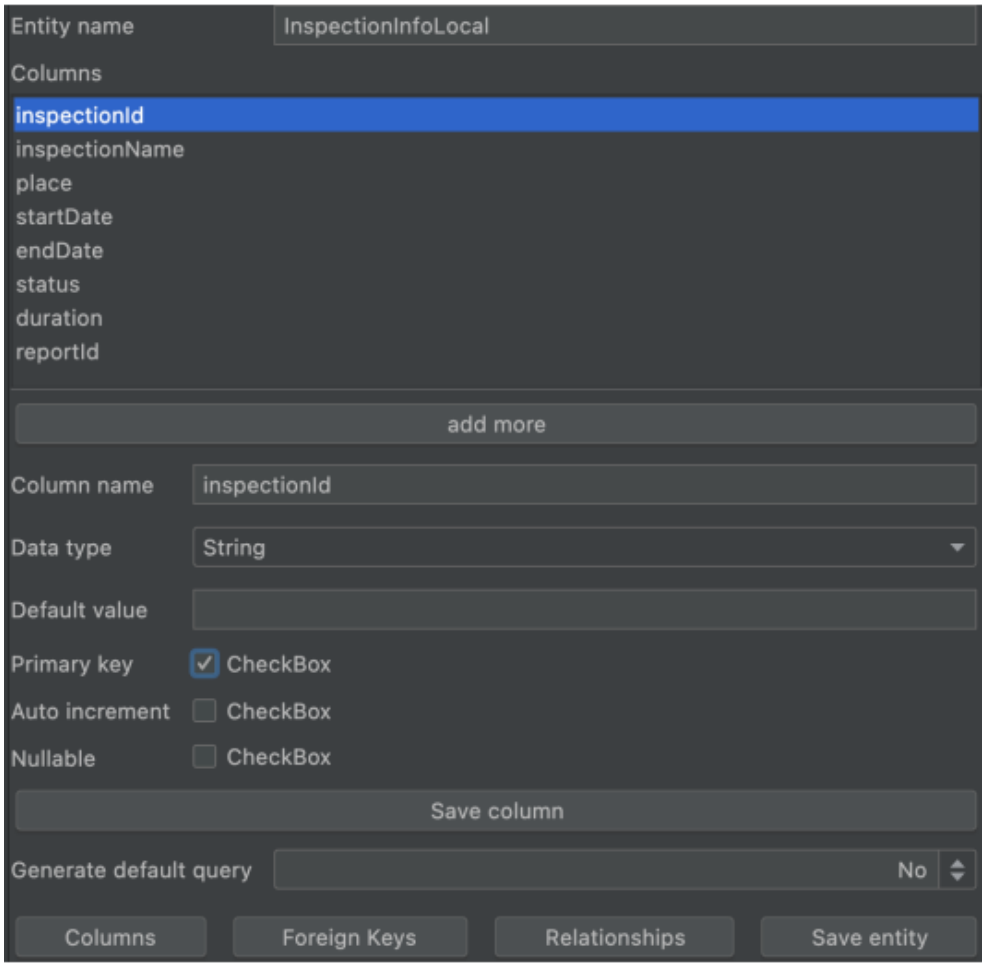

Рисунок 3.7 – Таблиця InspectionInfoLocal

Кожна інспекція може мати свого інспектора, тому необхідно створити таблицю з інформацією про інспектора InspectorInfoLocal. Для цього потрібно перейти на екран створення нової таблиці БД за допомогою кнопки Create. Далі

додаємо стовпці з інформацією про інспектора (рис. 3.8): ідентифікатор інспектора (inspectorId), ПІП (fio), організацію (organization), вік (age). Первинним ключем таблиці буде стовпець inspectorId.

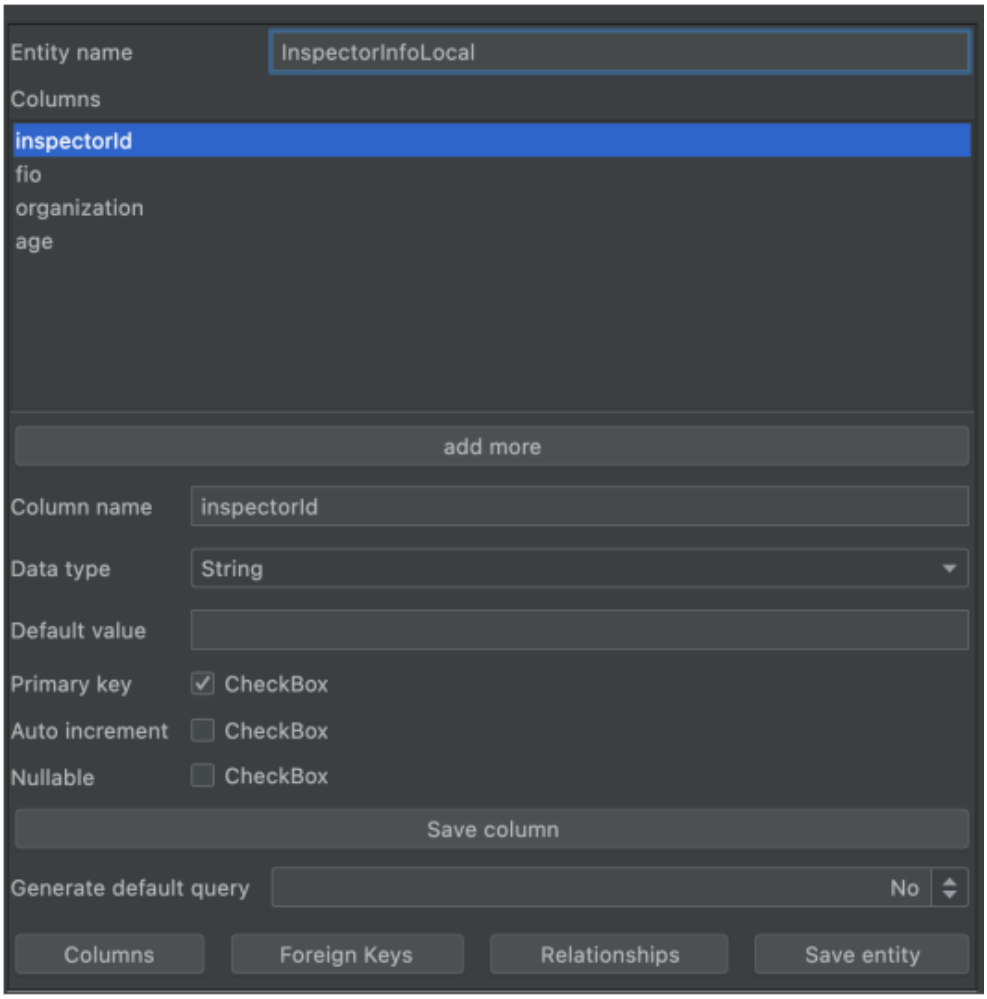

Рисунок 3.8 – Таблиця InspectorInfoLocal

Інспектор та інспекція повинні бути пов'язані відношенням багато до багатьох, оскільки один інспектор може виконувати кілька інспекцій, а одна інспекція може виконуватися кількома інспекторами. Тому перейдемо на екран створення відношеньн за допомогою кнопки Relationships і додамо зв'язок багато до багатьох. Зв'язковими колонками будуть первинні ключі inspectorId і inpectionId (рис. 3.9).

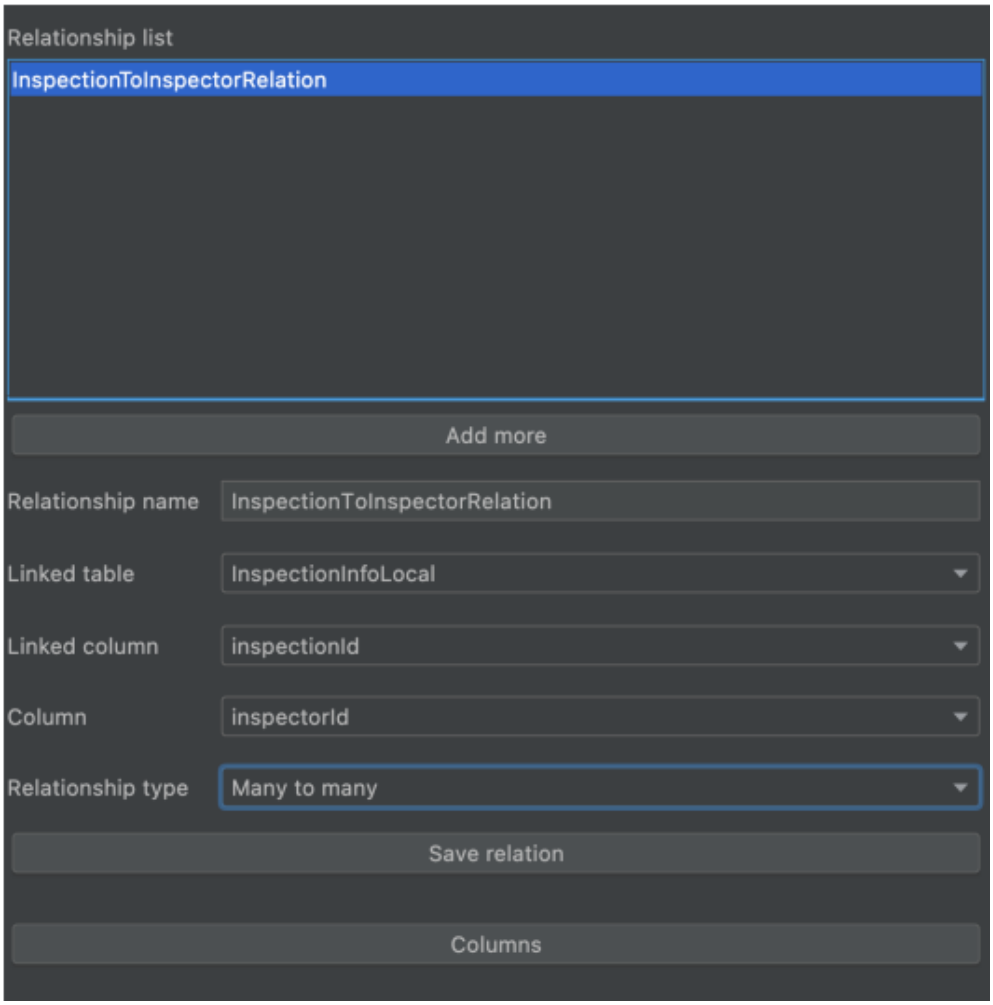

Рисунок 3.9 – Екран створення відношень між таблицями InspectionInfoLocal та InspectorInfoLocal

У таблиці InspectionInfo є стовпець з ідентифікатором звіту інспекції. Створимо таблицю з детальною інформацією про звіти інспекції ReportInfoLocal, яка містить ідентифікатор звіту (reportId), ім'я звіту (reportName), тип звіту (reportType), статус звіту (reportStatus), версію звіту (versionNumber) та список їхньою короткою інформацією (documentsInfo) (рис. 3.10).

Статус у звіті може бути відсутнім, тому надамо йому значення “0” за замовчуванням у полі “Default value”.

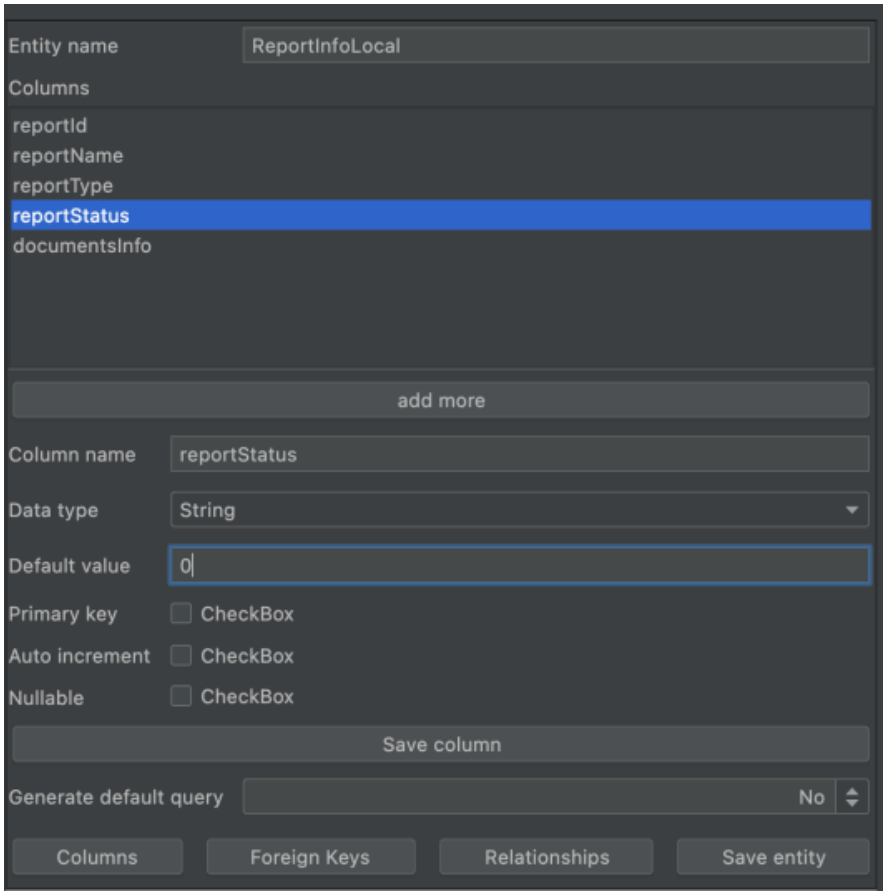

Рисунок 3.10 – Таблиця ReportInfoLocal

Тип даних documentsInfo є списком з елементів DocumentInfo data -класу, в якому міститься коротка інформація про документи, шлях до якого вказується після натискання на елемент Custom із списку Data type , що випадає (рис. 3.11). Щоб зберігати не примітивний тип даних у БД, для нього буде згенеровано TypeConverter.

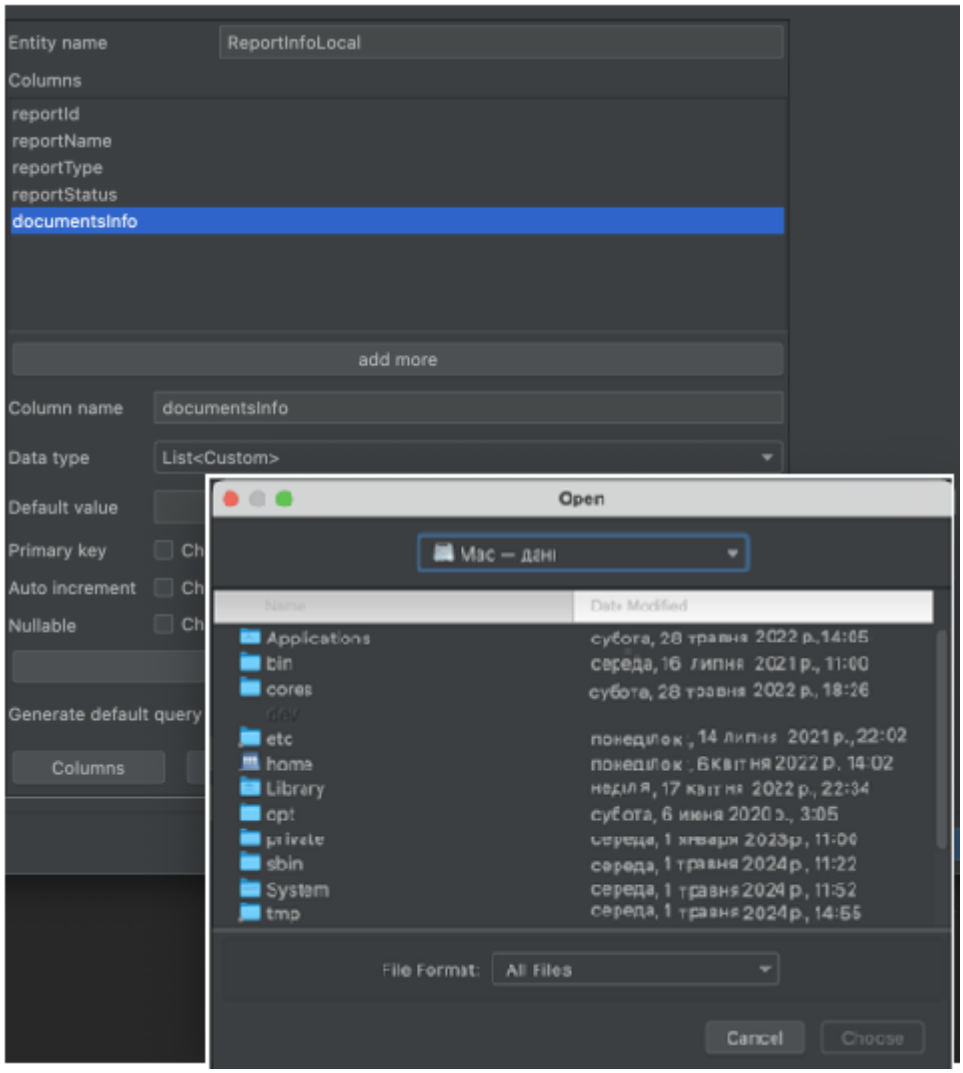

Рисунок 3.11 – Вибір не примітивного типу даних

Також зв'яжемо таблицю звіту з таблицею інспекції за допомогою зовнішнього ключа, який вказуватиме на reportIdз батьківської таблиці та reportIdз дочірньої таблиці (рис. 3.12).

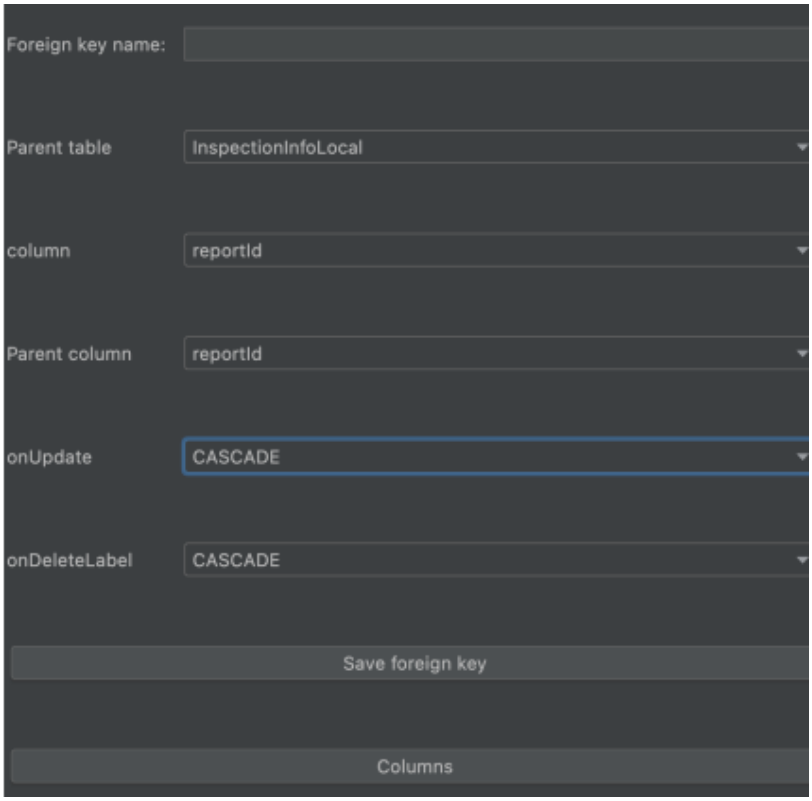

Рисунок 3.12 – Екран створення зовнішнього ключа між таблицями InspectionInfoLocal та ReportInfoLocal

Далі зберігаємо зміни зовнішнього ключа та повертаємося до таблиці, де вказуємо в JSpinner Generate default query варіант з Coroutines (рис. 3.13), щоб згенерувати клас з асинхронними coroutines запитами до таблиці зі звітами.

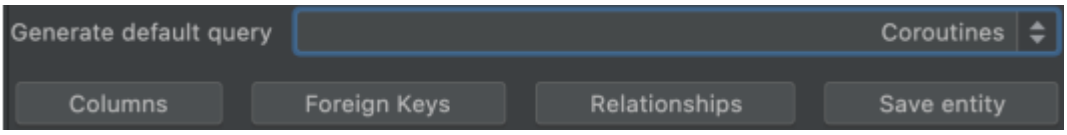

Рисунок 3.13 – Вибір типу асинхронних запитів

Зберігаємо таблицю, повертаємося до списку таблиць і натискаємо кнопку “Generate”. В результаті одержуємо набір згенерованих компонентів Room БД (Додаток Б).

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Класифікація шкідливих та небезпечних виробничих факторів

Шкідливий виробничий фактор – небажане явище, яке супроводжує виробничий процес і вплив якого на працюючого може призвести до погіршення самопочуття, зниження працездатності, захворювання, виробничо зумовленого чи професійного, і навіть смерті, як результату захворювання. Небезпечний виробничий фактор – небажане явище, яке супроводжує виробничий процес і дія якого за певних умов може призвести до травми або іншого раптового погіршення здоров'я працівника (гострого отруєння, гострого захворювання) і навіть до раптової смерті [34].

Поділ несприятливих чинників виробничого середовища на шкідливі та небезпечні зумовлене різним характером їх дії на людський організм, тим, що вони потребують різних заходів та засобів для боротьби з ними та профілактики викликаних ними ушкоджень, а також рядом причин організаційного характеру. В той же час між шкідливими та небезпечними виробничими факторами інколи важко провести чітку межу. Один і той же чинник може викликати травму і профзахворювання (наприклад, високий рівень іонізуючого або теплового випромінювання може викликати опік або навіть призвести до миттєвої смерті, а довготривала дія порівняно невисокого рівня цих же факторів – до хвороби; пилинка, що потрапила в око, спричиняє травму, а пил, що осідає в легенях, – захворювання, що зветься пневмоконіоз). Через це всі несприятливі виробничі чинники часто розглядаються як єдине поняття – небезпечний та шкідливий виробничий фактор (НШВФ) [35]. За своїм походженням та природою дії всі НШВФ можна поділити на 5 груп: фізичні, хімічні, біологічні, психофізіологічні та соціальні До фізичних НШВФ відносяться машини та механізми або їх елементи, а також вироби, матеріали, заготовки тощо, які рухаються або обертаються; конструкції, які руйнуються; системи, устаткування або елементи обладнання, які знаходяться під підвищеним тиском; підвищена запиленість та загазованість

повітря; підвищена або понижена температура повітря, поверхонь приміщення, обладнання, матеріалів; підвищені рівні шуму, вібрації, ультразвуку, інфразвуку; підвищений або понижений барометричний тиск та його різкі коливання; підвищена та понижена вологість; підвищена швидкість руху та підвищена іонізація повітря; підвищений рівень іонізуючих випромінювань; підвищене значення напруги в електричній мережі; підвищені рівні статичної електрики, електромагнітних випромінювань; підвищена напруженість електричного, магнітного полів; відсутність або нестача світла; недостатня освітленість робочої зони; підвищена яскравість світла; понижена контрастність; прямий та віддзеркалений блиск; підвищена пульсація світлового потоку; підвищені рівні ультрафіолетової та інфрачервоної радіації; гострі крайки, зачипки, шершавість на поверхні заготовок, інструментів та обладнання; розташування робочого місця на значній висоті відносно землі (підлоги); слизька підлога; невагомість.

Хімічні НШВФ:

- за характером дії на організм людини поділяються на токсичні, задушливі, наркотичні, подразнюючі, сенсибілізуючі, канцерогенні, мутагенні та такі, що впливають на репродуктивну функцію;

- за шляхами проникнення в організм людини поділяються на такі, що потрапляють через: 1) органи дихання; 2) шлунково-кишковий тракт; 3) шкіряні покриви та слизова оболонка;

- які перебувають у різному агрегатному стані: 1)твердому 2)газоподібному 3)рідкому.

Біологічні НШВФ – це: - патогенні мікроорганізми (бактерії, віруси, рикетсії, спірохети, грибки, найпростіші) та продукти їхньої життєдіяльності; - макроорганізми (тварини та рослини) та продукти їхньої життєдіяльності. До психофізіологічних НШВФ відносяться фізичні (статичні та динамічні) перевантаження і нервово-психічні перевантаження (розумове перенапруження, перенапруження аналізаторів, монотонність праці, емоційні перевантаження). 6 Соціальні НШВФ – це неякісна організація роботи, понаднормова робота, змушеність праці в колективі з поганими відносинами між його членами, соціальна

ізольованість з відривом від сім_ї, зміна біоритмів, незадоволеність роботою, фізична та/або словесна образа та її ризик, насильство та його ризик. Один і той же НШВФ за природою своєї дії може належати водночас до різних груп.

4.2 Вплив вібрації на людину

Вібрація - це механічні коливання пружних тіл або коливальні рухи механічних систем. Для людини вібрація є видом механічного впливу, який має негативні наслідки для організму [34].

Причиною появи вібрації є неврівноважені сили та ударні процеси в діючих механізмах. Створення високопродуктивних потужних машин і швидкісних транспортних засобів при одночасному зниженні їх матеріалоємності неминуче призводить до збільшення інтенсивності і розширення спектру вібраційних та віброакустичних полів. Цьому сприяє також широке використання в промисловості і будівництві високоефективних механізмів вібраційної та віброударної дії.

Дія вібрації може приводити до трансформування внутрішньої структури і поверхневих шарів матеріалів, зміни умов тертя і зносу на контактних поверхнях деталей машин, нагрівання конструкцій. Через вібрацію збільшуються динамічні навантаження в елементах конструкцій, стиках і сполученнях, знижується несуча здатність деталей, ініціюються тріщини, виникає руйнування обладнання. Усе це приводить до зниження строку служби устаткування, зростання імовірності аварійних ситуацій і зростання економічних витрат. Вважають, що 80% аварій в машинах і механізмах здійснюється внаслідок вібрації. Крім того, коливання конструкцій часто є джерелом небажаного шуму. Захист від вібрації є складною і багатоплановою в науково-технічному та важливою у соціальноекономічному відношеннях проблемою нашого суспільства [35].

Вплив вібрації на людину залежить від її спектрального складу, напрямку дії, прикладення, тривалості впливу, а також від індивідуальних особливостей людини.

При оцінці вібраційного впливу потрібно враховувати, що коливальні процеси притаманні живому організму. В основі серцевої діяльності і кровообігу та біострумів мозку лежать ритмічні коливання. Внутрішні органи людини можна розглядати як коливальні системи з пружними зв'язками. Частоти їх власних коливань лежать у діапазоні 3..6 Гц. Частоти власних коливань плечового пояса, стегон і голови щодо опорної поверхні (положення стоячи) складають 4...6 Гц, голови щодо плечей (положення сидячи) 25...30 Гц.

При впливі на людину зовнішніх коливань (хитавиці, струсів, вібрації) відбувається їхня взаємодія з внутрішніми хвильовими процесами, виникнення резонансних явищ. Так, зовнішні коливання частотою менш 0,7 Гц утворюють хитавицю і порушують у людини нормальну діяльність вестибулярного апарата. Інфразвукові коливання (менш 16 Гц), впливаючи на людину, пригнічують центральну нервову систему, викликаючи почуття тривоги, страху. При певній інтенсивності на частоті 6..7 Гц інфразвукові коливання, втягуючи у резонанс внутрішні органи і систему кровообігу, здатні викликати травми, розриви артерій, тощо [36].

Вібрація, що діє на людину, має широкий діапазон – від десятих часток одного до декількох тисяч Гц. Характерними ознаками шкідливого впливу вібрації на людину є можливі зміни у функціональному стані: підвищена втома, збільшення часу моторної реакції, порушення вестибулярної реакції. Медичними дослідженнями встановлено, що вібрація є подразником периферичних нервових закінчень, розташованих на ділянках тіла людини, що сприймають зовнішні коливання. Адекватним фізичним критерієм оцінки її впливу на організм людини є коливальна енергія, що виникає на поверхні контакту, а також енергія, поглинена тканинами і передана опорно-руховому апарату та іншим органам. У результаті впливу вібрації виникають нервовосудинні розлади, ураження кістково-суглобної та інших систем організму. Відзначаються, наприклад, зміни функції щитовидної залози, сечостатевої системи, шлунково-кишкового тракту. Так, медичні дослідження показали , що у працюючих в умовах вібрації відбуваються значні зміни кістковосуглобної системи, які виражаються у функціональній перебудові

кісткової тканини, регіональному остеопорозі, кістовидних утвореннях у кістках, асептичному некрозі кісток, хронічних переломах. Відзначається, що терміни виникнення змін у кістках у працівників вібраційних професій коливається в межах від 6-8 місяців до 2-5 років.

Шкідливість вібрації збільшується при одночасному впливі на людину таких факторів, як знижена температура, підвищений шум, запиленість повітря, тривала статична напруга тощо. Сучасна медицина розглядає виробничу вібрацію як могутній стрес-фактор, що має негативний вплив на психомоторну працездатність, емоційну сферу і розумову діяльність, підвищує ймовірність виникнення різних захворювань і нещасних випадків. Особливо небезпечний тривалий вплив вібрації для жіночого організму. Широкий комплекс патологічних відхилень, викликаний впливом вібрації на організм людини, кваліфікується як віброзахворювання [35].

Вібрація як фізичний чинник виробничого середовища спостерігається в металообробній, гірничодобувній, металобудівній, машинобудівній, авіаційній та інших галузях народного господарства. Джерелом вібрації можуть бути різні механізми, вібраційне устаткування, віброінструменти, акустичні системи, транспортні та сільськогосподарські машини.

Загальна вібрація поділяється на транспортну вібрацію, яка діє на людину на робочих місцях в транспортних засобах (трактори сільськогосподарські та промислові, самохідні сільськогосподарські машини (комбайни), тягачі, грейдери ті інші); транспортно-технологічну вібрацію, яка діє на людину на робочих місцях машин з обмеженою рухливістю (екскаватори, крани промислові та будівельні, гірничі комбайни, транспорт виробничих приміщень та інші) та технологічну вібрацію, яка діє на людину на робочих місцях стаціонарних машин чи передається на робочі, де немає джерел вібрації (верстати та метало-деревообробне, пресувально-ковальське обладнання, ливарні машини, електричні машини, насосні агрегати та вентилятори, обладнання для буріння свердловин, бурові верстати, машини для тваринництва, очищення та сортування зерна (у тому числі сушарні), обладнання промисловості будматеріалів (крім бетоноукладачів), установки хімічної та нафтохімічної промисловості та інші.

Оператори машин, які зазнають у процесі трудової діяльності впливу вібрації, підлягають попереднім та періодичним медичним оглядам відповідно до Порядку проведення медичних оглядів працівників певних категорій, затвердженого Наказом МОЗ України від 21.05.2007 р. №246. Обов'язкові попередні (під час прийняття на роботу) та періодичні (протягом трудової діяльності) медичні огляди дозволять визначити стан здоров'я працівника та можливість виконання без погіршення стану здоров'я професійних обов'язків, своєчасно виявити ранні ознаки хронічного професійного захворювання, забезпечує динамічне спостереження за станом здоров`я в умовах дії шкідливих та небезпечних факторів і трудового процесу, вирішує питання щодо можливості продовжувати роботу в умовах дії шкідливих та небезпечних факторів і трудового процесу [35].

За результатами періодичних медичних оглядів роботодавець забезпечує проведення відповідних оздоровчих заходів Заключного акта у повному обсязі та усуває причини, що призводять до професійних захворювань. Організовує проведення лабораторних досліджень умов праці на робочих місцях та вживає заходів до усунення небезпечних і шкідливих для здоров'я виробничих факторів.

До роботи операторами машин допускаються особи не молодші 18 років, які пройшли попередній медичний огляд, мають відповідну кваліфікацію та ознайомлені з характером впливу вібрації на організм.

ВИСНОВОК

В результаті виконання даної роботи було розроблено програмний інструмент, який дозволяє автоматизувати процес управління локальними БД в Android - застосунках.

Для досягнення поставленої мети було виконано такі завдання:

– створено компонент для генерації Entity таблиць бази даних на основі введених користувачем даних;

– реалізовано рішення щодо генерації TypeConverters, які дозволяють зберігати в Entity таблицях непримітивні типи даних, за рахунок їх конвертації в String при вставці та назад при отриманні;

– додана можливість створення базових запитів до таблиць з асинхронними викликами (Coroutines, RxJava);

– реалізовано компонент для створення зовнішніх ключів та зв'язків один до одного, багато до багатьох, багато до одного між таблицями;

– реалізовано компонент для аналізу існуючої БД програми, що здійснюється за рахунок парсингу її компонентів, що дає можливість користувачеві інтегрувати до неї нові сутності;

– розроблено компонент для інтеграції сутностей у код проекту за допомогою Dagger 2, який генерує provide методи доступу до нових компонентів;

– створено плагін у середовищі Android Studio, що поєднує всі перелічені вище пункти у своєму інтерфейсі, через який користувач може створювати та редагувати структуру БД Room.

Розроблене в рамках даної роботи рішення дозволить скоротити часові витрати на розробку та підтримку БД Room в Android проектах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nickerson, R.C. Selecting a Stored Data Approach for Mobile Apps / Robert C. Nickerson, Francois B. Mourato-Dussault // SciELO - Scientific Electronic Library Online. URL: https://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-1876201600 0300004 (дата звернення: 29.02.2024)

2. Kormas, D. Analysis of data storage methods available in the Android SDK // Journal of Computer Sciences Institute. URL: https://ph.pollub.pl/index.php/jcsi/article/view/2759 (дата звернення: 29.02.2024)

3. Understand how structured data works // Google Search Central. URL:https://developers.google.com/search/docs/advanced/structured-data/in tro-structured-data (дата звернення: 29.02.2024)

4. Мережеве кешування в iOS // Habr - Site for publishing articles. URL: https://habr.com/company/e-legion/ blog/192308/ (дата звернення: 13.03.2024)

5. Kozłowski, M. Android Databases Performance Tests — CRUD // Proandroid Dev - Site for publishing articles. URL: https://proandroiddev.com/android-databases-performance-crud-a963dd7bb0 eb (дата звернення: 13.03.2024)

6. Kaushik, D. Caching to Reduce Mobile App Energy Consumption / Kaushik D., Debra Vander Meer // Article. URL: https://www.researchgate.net/publication/ 319639602_Caching_to_Reduce_ Mobile_App_Energy_Consumption (дата звернення: 13.03.2024)

7. Room Database. Android Developers // Android Developers. URL: https://developer.android.com/training/data-storage/room (дата звернення: 15.03.2024)

8. Android Architecture Components // Raywenderlich. The best investment for your mobile development career. URL: https://www.raywenderlich.com/books/ advanced-android-app-architecture/v1.0/chapters/4-android-architecture-components (дата звернення: 15.03.2024)

9. Кількість репозиторіїв з Room database на Github. URL: https://github.com/search?q=Room+database&type=Repositories (дата звернення: 16.03.2024)

10. Jalava, E. Converter- eli muuntajaluokka 3.2.2 // Android-sovelluksen koodin ja arkkitehtuurin uudistaminen. URL: https://www.theseus.fi/bitstream/handle/10024/185308/jalava_elisa.pdf?sequ ence=2 — P. 20-22 (дата звернення: 16.03.2024)

11. Concurrency in Android development – Kotlin Coroutines and RxJava // IC-Online. URL: https://iconline.ipleiria.pt/handle/10400.8/5840 (дата звернення: 17.03.2024)

12. Опис бібліотеки dagger 2. URL: https://habr.com/articles/320676/ (дата звернення: 19.03.2024)

13. Огляд Room Persistence Library. URL: ttps://uandcode.com/android-tutorial/room-part-2/ (дата звернення: 12.03.2024)

14. Defining data using Room entities // Android Developers. URL: https://developer.android.com/training/data-storage/room/defining-data (дата звернення: 26.03.2024)

15. Accessing data using Room DAOs // Android Developers. URL: https://developer.android.com/training/data-storage/room/accessing-data (дата звернення: 29.04.2024)

16. Relation // Android Developers. URL: https://developer.android.com/reference/kotlin/androidx/room/Relation (дата звернення: 29.03.2024)

17. Tupper, C.D. Associative Entity // ScienceDirect Website. URL:https://www.sciencedirect.com/topics/computer-science/associative-ent ity (дата звернення: 29.03.2024)

18. Muntenescu, F. Understanding migrations with Room // Medium - Site for publishing articles. URL: https://medium.com/android developers/understanding-migrations-with-room-f01e04b07929 (дата звернення: 14.04.2024)

19. Розробка мобільного застосунку, що взаємодіє з сервером, із використанням архітектурних компонентів ОС Andorid і Custom View. URL: https://research-journal.org/en/engineering/razrabotka-mobilnogo-prilozheni ya-vzaimodejstvuyushhego-s-serverom-s-ispolzovaniem-arxitekturnyx-kom ponentov-operacionnoj-sistemy-andorid-i-custom-view/ (дата звернення: 14.04.2024)

20. SQL: Primary Keys // Tech on the net. URL:

https://www.techonthenet.com/sql/primary_keys.php#:~:text=In%20SQL%2C%20a%20primary%20key,a%20primary%20key%20in%20SQL (дата звернення: 18.04.2024)

21. Embedded // Android Developers. URL: https://developer.android.com/reference/ androidx/room/Embedded (дата звернення: 19.04.2024)

22. SQL FOREIGN KEY Constraint // W3Schools. URL: https://www.w3schools.com /sql/sql_foreignkey.asp (дата звернення: 22.04.2024)

23. DB Browser for SQLite // The Official home of the DB Browser for SQLite. URL: https://sqlitebrowser.org/ (дата звернення: 22.04.2024)

24. MySQL Workbench // The world's most popular open source database. URL: https://www.mysql.com/products/workbench (дата звернення: 22.04.2024)

25. DbSchema: Design, Document & Deploy Databases // DbSchema Documentation. URL: https://dbschema.com/ (дата звернення: 23.04.2024)

26. Core Data // Apple developers Website. URL: https://developer.apple.com/documentation/coredata#overview (дата звернення: 23.04.2024)

27. Manage databases in Android devices, in real time, without leaving Android Studio and IntelliJ IDEA // SQLScout Website. URL: https://www.idescout.com/features/ (дата звернення: 24.04.2024)

28. Barakrishnan, G. Room Model Builder // Github - web service for hosting IT projects and their joint development. URL: https://github.com/gokul42252/Room-Model-Builder (дата звернення: 25.04.2024)

29. Rawat, A. Using Room Database | Android Jetpack // Medium - Site for publishing articles. URL: https://medium.com/mindorks/using-room-database-android-jetpack-675a89 a0e942 (дата звернення: 28.04.2024)

30. Lesovyi, R. Roomigrant // Github - web service for hosting IT projects and their joint development. URL: https://github.com/MatrixDev/Roomigrant (дата звернення: 10.05.2024)

31. What is Apache FreeMarker? // FreeMarker Website. URL: https://freemarker.apache.org/ (дата звернення: 10.05.2024)

32. Using the Swing Application Framework // Oracle Cloud Applications and Cloud Platform. URL: https://www.oracle.com/technical-resources/articles/javase/swingappfr.html (дата звернення: 11.05.2024)

33. Null Safety // Kotlin Programming Language. URL: https://kotlinlang.org/docs/null-safety.html (дата звернення: 13.05.2024)

34. Зеркалов Д.В. Безпека життєдіяльності та основи охорони праці. Навчальний посібник. К.: «Основа». 2016. – 267 с.

35. Яремко З. М. Безпека життєдіяльності: Навч. посіб. — Львів., 2005. – 301 с.

36. Желібо Є. П. Заверуха Н.М., Зацарний В.В. Безпека життєдіяльності. Навчальний посібник. – К.: Каравела, 2004. -328 с.

ДОДАТОК А

Існуючий проект Room бази даних

Клас конфігурації бази даних

```
import android.content.Context
import androidx.room.Database import
androidx.room.Room import
androidx.room.RoomDatabase
import uk.nipigas.smartinspector.core db.AppDatabase
@Database(
   version = 1, entities = [
      InspectionInfoLocal::class,
   ] ,
   exportSchema = false
)
abstract class AppDatabase : RoomDatabase() {
   companion object {
      private var instance: AppDatabase? =
      null @Synchronized
      fun get(context: Context):
         AppDatabase { if (instance ==
         null) {
            instance =
               Room.databaseBuilder(
               context.applicationContext
               ,
               AppDatabase::class.j ava, "app.db"
            )
               .fallbackToDestructiveMigration()
               .build()
         }
         return instance!!
      }
   }
   abstract fun inspectionInfoLocal(): InspectionInfoLocal
}
```

Таблиця InspectionInfoLocal

```
import androidx.room.ColumnInfo
import androidx.room.Entity import
ru.nipigas.smartinspector.core
db.test.InspectionInfoLocal.Companion.INSPECTION
_ID
import
ru.nipigas.smartinspector.core
db.test.InspectionInfoLocal.Companion.INSPECTION INFO TABLE@Entity(
   tableName =
```

```
   INSPECTION_INFO_TABLE,
   primaryKeys = [INSPECTION_ID]
) data class InspectionInfoLocal(
   @ColumnInfo(name =
   INSPECTION_ID) val inspectionId:
   String,
   @ColumnInfo(name =
   INSPECTION_NAME) val
   inspectionName: String,
   @ColumnInfo(name = PLACE) val
   place: String,
   @ColumnInfo(name = START_DATE)
   val startDate: Long,
   @ColumnInfo(name = END DATE) val
   endDate: Long,
   @ColumnInfo(name = STATUS) val
   status: String,
   @ColumnInfo(name = DURATION) val
   duration: String,
   @ColumnInfo(name =
   PARENT_REPORT_ID) val reportId:
   String,
) {
 companiobject {
   constval INSPECTI INFO TABLE = "inspection table
   constval INSPECTI NAME = "inspection name" "
   constval INSPECTI ID = "inspection id"
   constval PLACE = "place"
   constval START    = "start date"
   constval END DATE "end date"
   constval STATUS = status"
   constval DURATION "duration"
   constval PARENT REPORT ID = "parent

   }
}
```

ДОДАТОК Б

Приклад згенерованих компонентів Room бази даних

Згенерований файл таблиці ReportInfoLocal.kt

```
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.ForeignKey
import androidx.room.TypeConverters
import generated.ReportInfoLocal.Companion.REPORT INFO LOCAL
import generated.ReportInfoLocal.Companion.REPORT ID
import
ru.nipigas.smartinspector.core
db.converters.ListOfIrReportLocalConverter
import uk.nipigas.smartinspector.core db.test.DocumentInfo
import uk.nipigas.smartinspector.core
db.test.ListOfDocumentInfoTypeConverter @Entity(
   tableName = REPORT_INFO_LOCAL,
   primaryKeys = [REPORT_ID],
   foreignKeys = [
      ForeignKey(
         entity = InspectionInfoLocal::class,
         parentColumns = arrayOf(InspectionInfoLocal.REPORT
         ID), childColumns = arrayOf(REPORT ID), onDelete =
         ForeignKey.CASCADE, onUpdate = ForeignKey.CASCADE,
      ),
   ]
)
@TypeConverters(ListOfDocumentInfoTypeConverter::class) data class
ReportInfoLocal(
   @ColumnInfo(name = ID REPORT) var
   reportId: String,
   @ColumnInfo(name = REPORT_NAME)
   var reportName: String,
   @ColumnInfo(name = REPORT_TYPE)
   var reportType: String,
   @ColumnInfo(name = REPORT_STATUS)
   var reportStatus: String,
   @ColumnInfo(name =
   DOCUMENTS_INFO) var
   documentsInfo: DocumentInfo
) {
companion object {
   const val REPORT_INFO_LOCAL = "reportInfoLocal"
   const val REPORT_ID = "reportId"
   const val REPORT_NAME = Const "reportName"
   const val REPORT_TYPE = "reportType"
   const val REPORT_STATUS = "reportStatus"
   const val DOCUMENTS INFO = "documentsInfo"
```