КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

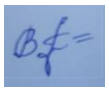
«РОЗРОБКА ANDROID-ЗАСТОСУНКУ "ЩОДЕННИК-ОРГАНАЙЗЕР ДЛЯ ЗАНЯТЬ МУЗИКОЮ"»

Виконав студент 4-го курсу
Дубінін Ілля Дмитрович

(підпис)

Науковий керінник:
доцент, кандидат фізико-математичних наук

Волохов Віктор Миколайович



(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«01» червня 2022 р.,
протокол № 10
Завідувач кафедри
Нікітченко М.С.

(підпис)

Київ - 2022

РЕФЕРАТ

Обсяг роботи: 52 сторінки, 16 ілюстрацій, 4 таблиці, 10 джерел посилань, 1 додаток.

ЗАСТОСУНОК ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID, ЗБЕРІГАННЯ НАВЧАЛЬНИХ МАТЕРІАЛІВ, МОБІЛЬНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОРГАНІЗАЦІЯ ПРОЦЕСУ НАВЧАННЯ МУЗИКАНТА, РОБОТА З ФАЙЛАМИ ВІДЕО І АУДІО, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID, ЩОДЕННИК-ОРГАНАЙЗЕР.

Об’єктом роботи є діяльність музиканта, тобто його заняття, репетиції чи виступи. Предметом роботи є організація занять або репетицій музиканта, тобто взаємодія з матеріалами, які використовуються у цьому процесі. Метою роботи є створення застосунку «Щоденник-органайзер для занять музикою» для пристроїв, які працюють на операційній системі Android.

Методи розробки: проєктування бази даних, взаємодія з файловою системою, використання власного досвіду у предметній області. Інструменти розроблення: IDE Android Studio Arctic Fox 2020.3.1, система керування базами даних SQLite, DB Browser for SQLite, мова програмування Java.

Результати розроблення: було проведено аналіз існуючих застосунків для ведення щоденників і організації інформації, розроблено застосунок для ОС Android для збереження і організації інформації, яка використовується у процесі роботи музиканта.

Створений застосунок може бути використаний музикантами різного рівня – від початківців-аматорів до студентів вищих навчальних закладів, наприклад консерваторій, і професіоналів.

Подальший розвиток продукту можливий за рахунок більш тісної співпраці з потенційними користувачами. Наприклад, за допомогою опитувань, накопичення, аналізу і впровадження досвіду використання програми, і, як результат – додавання нової функціональності у застосунок.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IDE – integrated development environment, інтегроване середовище розробки;

PDF – portable document format, формат для представлення документів у незалежному від пристрою виведення вигляді;

SDK - software development kit, засоби розробки для певної платформи;

URI – uniform resource identifier, уніфікований ідентифікатор ресурсів;

БД – база даних;

ОС – операційна система;

ПЗ – програмне забезпечення;

СКБД – система керування базами даних

ВСТУП

Оцінка сучасного стану об’єкта дослідження або розробки. Розробка Android-застосунків є дуже актуальною. З моменту виходу ОС Android на ринок у 2008 році, кількість її користувачів постійно збільшується. Вона є однією з двох найбільш популярних мобільних ОС разом з iOS від Apple [1], але при цьому має чимало переваг. Серед них можна навести набагато більшу доступність, можливість кастомізації, зручність у налаштуванні, наявність гарних засобів для розробки та можливість обрати серед них найбільш зручний. Google випускає навчальні матеріали для розробників та підтримує документацію для своєї мобільної ОС.

Актуальність роботи та підстави для її виконання. Тема роботи – «Розробка Android-застосунку "Щоденник-органайзер для занять музикою». Вона є досить актуальною, оскільки на ринку майже неможливо знайти мобільні застосунки з потрібною функціональністю. Можна знайти багато щоденників, записників та схожих на них програм, проте вони орієнтовані на повсякденне використання та рутинні справи. Ринок музичних програм наповнений трохи іншими продуктами. Здебільшого це застосунки, які надають доступ до нотного матеріалу, табулатур. Також є навчальні проекти, де можна знайти записи уроків. Недоліком цих програм є те, що там відсутня можливість зберігати свої нотатки, записи та інші матеріали і при цьому вони цілком залежать від з’єднання з Інтернет, що відбирає можливість користувача працювати автономно.

Наш застосунок має вирішити проблему зберігання матеріалів для навчання та інших записів для відслідковування прогресу, хоча б частково. Також буде можливо оцінювати власну роботу, для покращення планування занять.

Підставою для виконання даної роботи є спроба допомогти користувачам-музикантам у групуванні інформації, яка потрібна для роботи та за допомогою цього полегшенні планування своїх занять, репетицій тощо.

Мета й завдання роботи. Мета роботи є відповідною до її теми – ми маємо розробити застосунок «Щоденник-органайзер для занять музикою» для пристроїв, які працюють на ОС Android. Для того, щоб досягнути поставлену мету потрібно розв'язати кілька завдань. Наведемо їх перелік:

- Проаналізувати наявне ПЗ продуктів-конкурентів
- Визначити їх «слабкі» та «сильні» сторони
- Дослідити потреби майбутніх користувачів застосунку
- Обрати засоби розробки для виконання роботи
- Реалізувати взаємодію з БД, для локального зберігання даних
- Записати конкретні вимоги до застосунку
- Реалізувати застосунок, створити програму
- Протестувати застосунок з реальними даними та користувачами.

Об'єкт, методи й засоби дослідження або розроблення. Як було зазначено вище, об'єктом роботи є праця музиканта, його діяльність. Предметом роботи є матеріали, які використовує музикант. Метою роботи є створення застосунку для смартфонів, які працюють на ОС Android, у якому можна буде зберігати інформацію про заняття або репетиції музиканта.

Зазначимо методи дослідження обраної області. Думаю, що в нашій роботі буде доцільно використати і емпіричні, і теоретичні методи дослідження. Спочатку потрібно провести спостереження над прикладами занять і репетицій музиканта(ів), провести аналіз отриманої інформації. Після цього проведемо узагальнення і визначимо, які аспекти його діяльності варто буде зберігати у нашому застосунку. В подальшому синтезуємо отримані результати з нашими базовими уявленнями про майбутній продукт, щоб забезпечити його максимально можливу якість.

Варто приділити велику долю уваги обранню правильних засобів розробки. При розробці додатків для Android можна обирати серед таких мов програмування: Java, Kotlin, C#, JavaScript та схожі на нього. Обрана ОС використовує також інші мови, на зразок, C++, але вони майже не застосовуються при написанні програм високого рівня. За допомогою Java і Kotlin можна створювати нативні застосунки, для їх використання створено офіційну документацію від Google. За допомогою інших мов програмування можна створювати мультиплатформні додатки, наприклад за допомогою платформи Xamarin. Для розробки нашого застосунку оберемо мову програмування Java. Вона вже довгий час застосовується для створення додатків для ОС Android. Розроблено багато бібліотек, які можна використовувати для полегшення процесу розробки та написано багато статей та документацій, які допоможуть у роботі.

Також нам потрібно буде зберігати дані у застосунку, для цього зручно використати СКБД SQLite. Вона давно використовується при створенні додатків для Android та реалізується у вигляді бібліотеки, де підтримується велика частина функціональності зі стандарту SQL.

Розробка буде проводитись у IDE Android Studio. Дане середовище є рекомендованим до використання Google та має велику кількість зручних інструментів для полегшення роботи.

Можливі сфери застосування. Майбутніми клієнтами нашого застосунку є музиканти, як слідує з назви роботи. При чому додаток може бути використаний і початківцями, і користувачами близькими до професійного рівня. Оскільки на ринку бракує продуктів з даною функціональністю, тематика проекту є актуальною і може допомогти у вирішенні проблем збереження та організації роботи музиканта.

РОЗДІЛ 1. ОГЛЯД НАЯВНИХ НА РИНКУ СИСТЕМ

На ринку ПЗ для смартфонів на ОС Android бракує засобів, для організації процесу занять музиканта. Проте нам потрібно дослідити схожі застосунки, які надають можливість ведення щоденників та збереження важливої інформації.

Наведемо короткий список популярного ПЗ схожого на щоденники та зробимо його коротку характеристику.

- Daybook – зручний застосунок з гарною функціональністю. У ньому є параметри збереження паролів, диктофон, можливість створення списків справ. Є безкоштовна та платна версії застосунку.
- Daylio – популярний щоденник для Android. Наявна базова функціональність: щоденні записи, відслідковування настрою, захист застосунку за допомогою паролю, експорт записів у зовнішні файли. Повна функціональність доступна у платній версії додатку.
- Diary++ - звичайний щоденник. Має приємний зовнішній вигляд. Серед не досить звичних функцій є захист додатку за допомогою відбитку пальця, нагадування про якісь записи. Користуватись повним набором можливостей застосунку можна у платній версії.
- Diaro – досить незвичний щоденник. Є можливість збереження геотегів, простої навігації. Підтримується багато мов. Платна версія забезпечує синхронізацію з Dropbox, експорт у PDF та відключення реклами.
- Diary Book – досить простий додаток, що також популярний. Серед його переваг є можливість красивого форматування тексту, диктофон, нічний режим. Крім цього наявний красивий інтерфейс користувача.
- Five Minute Journal. Це незвичний, на перший погляд, додаток. Він орієнтований на короткі записи. Має простий інтерфейс, захист за допомогою паролю. Наявна тільки платна версія застосунку. На жаль, такої функціональності буде замало для музикантів.

- Journey – ще один щоденник для мобільних пристроїв. Наявні незвичні можливості, як синхронізація з Google Fit, відслідковування настрою. На жаль, використовує схему підписок, що може бути не зручним для користувача.
- Offline Diary. Як можна побачити з назви, цей застосунок може працювати без доступу до мережі, що може бути дійсно корисним. Застосунок є доволі мінімалістичним. З доступної функціональності – створення записів. Платна версія відрізняється відсутністю реклами. Можна виокремити цей застосунок через його акцент на роботу офлайн, що можна буде використати при розробці нашого продукту [2].
- Також згадаємо звичайні системні записники, що можна знайти у кожному смартфоні. Вони відрізняються у виробників телефонів, але мають здебільшого схожими функціями. Орієнтуються на прості записи, інколи можна створювати списки справ та встановлювати нагадування.

Зробимо короткий аналіз цієї інформації. Більшість застосунків-щоденників реалізують приблизно одну функціональність. Їх основна задача – створення записів, які в залежності від конкретного застосунку можуть бути просто текстовими або бути доповненими іншими матеріалами. Багато з описаних програм мають механізми захисту від доступу сторонніх осіб за допомогою паролів. Також часто зустрічаються щоденники, які орієнтовані на взаємодію з спортивними застосунками, як Google Fit, і відслідковування настрою.

Проте функціональність описаних застосунків не може повноцінно допомогти музикантам вести архів своїх занять, репетицій та групувати їх. Якщо зберігати всі записи в одному місці, як більшість щоденників, легко заплутатись і про зручність використання нашої програми можна буде забути. Серед недоліків існуючих додатків є обмежена взаємодія з файлами, які зберігаються на пристрої та реклама в безкоштовних версіях.

Отже, зробимо висновки з опрацьованої інформації та зробимо короткі нотатки, щодо того, яку функціональність можна буде розробити у нашій програмі.

Зручний щоденник для музиканта повинен групувати записи занять та репетицій по матеріалам, над якими працює музикант. Наприклад, користувачу-музиканту було б зручно створити категорію для одного твору, над яким він працює і зберігати там потрібну інформацію і записи.

Потрібно мати можливість взаємодії з файлами на пристрої. Наприклад, у користувача є ноти якоїсь композиції і він хоче їх додати до застосунку. Така функція буде корисною. Можна вести трекінг успішності занять. Так можна буде легко слідкувати за прогресом. Доречно додати функцію аналогічну камертону, для можливості налаштування інструменту.

Додаток має бути орієнтований на роботу без доступу до мережі. Зручно, коли всі дані є одразу і не потрібно чекати на те, коли завантажаться потрібні матеріали.

РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Для розробки нашого застосунку було використано такі технології:

1. Мова програмування Java
2. IDE Android Studio
3. СКБД SQLite
4. Android SDK
5. Система збирання Gradle

Мова програмування Java є найбільш часто вживаною для розробки застосунків для ОС Android. Це строго типізована, об’єктно-орієнтована мова. Вона була створена компанією Sun Microsystems, що була придбана Oracle. Останній належать і права на торгову марку. Застосунки Java транслюються в специфічний байт-код [3], таким чином вони можуть працювати на будь якій комп’ютерній архітектурі, для якої створено віртуальну Java-машину. Від часу створення цієї мови програмування було випущено багато нових версій та оновлень. Однією з найбільш популярних стала восьма версія Java. У ній було запропоновано багато нової функціональності для розробників. Наприклад, підтримка лямбда-виразів, статичні методи в інтерфейсах, посилання на методи і конструктори, функціональні інтерфейси, потоки для роботи з колекціями та ін. Найновішою версією мови є Java 18, що була випущена у березні 2022 року. Ми для розробки застосунку будемо використовувати версію Java 8, оскільки вона вже довгий час є популярною серед розробників.

При розробці додатків для Android програми компілюються в байт-код, який буде використовувати віртуальна машина Dalvik [4] (з версії Android 5.0 була замінена на ART). Така компіляція виконується додатковим інструментом Android SDK, що розроблена Google.

Розробку застосунків для Android можна вести у різних IDE. Ми зупинимось на Android Studio, оберемо версію Arctic Fox 2020.3.1 – одну з останніх. Це середовище було спеціально орієнтоване на програмування для

Android. У 2014 році Google визнала Android Studio офіційним середовищем для розробки під Android [5]. Підтримує такі мови програмування: Java, C++, Kotlin [6].

Gradle є системою автоматичного збирання. В ній розвиваються принципи, які були використані в Apache Ant і Apache Maven. У Gradle використовується DSL на основі Groovy замість звичної XML-подібної форми опису конфігурації програми [7]. На відміну від своїх попередників Gradle використовує орієнтований ациклічний граф для визначення порядку виконання завдань. Дану систему було створено для збирання великих програм, які з часом розростаються. Gradle підтримує інкрементальне збирання. Система визначає, яка частина програми була змінена і виконує тільки задачі, пов'язані з нею. Більшість плагінів Gradle призначені для роботи з Java і Groovy, але планується вихід плагінів і для інших мов програмування.

SQLite - компактна реляційна СКБД [8]. Вона не використовує патерн клієнт-сервер, натомість двигун SQLite є бібліотекою, з якою компонується програма, а не окремим процесом. Отже, обмін даними відбувається через API бібліотеки SQLite. Це зменшує час очікування та спрощує програму. Вся БД зберігається в одному файлі на комп'ютері, де виконується програма.

БД може працювати з кількома процесами одночасно. Вони без обмежень можуть читати дані. Запис до БД можна здійснити лише тоді, коли жодних інших запитів до БД не відбувається.

У SQLite можливі такі типи даних: INTEGER, REAL, TEXT та BLOB. Також підтримується особливе значення NULL.

Працювати з СКБД SQLite будемо за допомогою застосунку DB Browser for SQLite. У ньому є зручний графічний інтерфейс для створення, редагування таблиць та даних.

РОЗДІЛ 3. ПРИЗНАЧЕННЯ СИСТЕМИ

Як було згадано у попередніх розділах, на ринку майже відсутні застосунки для допомоги та організації процесу навчання або занять з музики. Ми проаналізували низку додатків, які можуть бути схожими на наш продукт, та дійшли висновку, що жоден з них не надає функціональність, яка буде корисна нашому кінцевому користувачу.

Наш щоденник має забезпечувати можливість музиканту зберігати результати своєї роботи, робити нотатки, структурувати наявну інформацію. При цьому, за можливості, слід уникати платних технологій, щоб кінцевий продукт був безкоштовний як для користувача так і для розробника. Бажано забезпечити можливість застосунку працювати офлайн та не використовувати зайву пам’ять на пристрої.

Отже, призначенням нашої системи є збереження інформації про заняття або репетиції музиканта. Вона повинна надавати доступ до таких даних, як відомості про заняття, відео- або аудіо-записи самого користувача чи інших музикантів, які можуть бути цікавими. Також застосунок має надавати можливість зберігання файлів PDF, у яких можуть бути потрібні користувачу зовнішні матеріали. Таким чином, користувачі нашого додатку зможуть зберігати всі відомості про їхні заняття у одному місці.

Крім цього в застосунку потрібно створити аналог камертону, щоб користувач міг налаштувати свої інструменти. Тобто програма повинна вміти генерувати звук заданої частоти.

У цьому розділі ми визначили призначення нашої програми. Також ми поверхнево проаналізували заплановану функціональність застосунку. У наступному розділі ми запишемо вимоги до системи більш формально.

РОЗДІЛ 4. ВИМОГИ ДО ЗАСТОСУНКУ

У попередніх розділах ми проаналізували такі аспекти нашого застосунку: мету і актуальність, інструментарій, який буде використовуватись для розробки, призначення системи, потреби користувача. Use Case-діаграму наведено у Додатку А.

Запишемо тепер вимоги до застосунку.

Мова застосунку. Основна мова застосунку – англійська.

Можливості користувача. У застосунку відсутня реєстрація та поділ користувачів на рівні доступу до можливостей застосунку. Користувач має такі можливості:

- переглядати список творів, над якими він працює;
- додавати новий твір до списку творів;
- видаляти твір зі списку;
- перейменовувати наявний у списку твір;
- переглядати список занять з певного твору;
- додавати нове заняття з певного твору;
- оцінювати стан роботи над певним твором;
- змінювати назву заняття;
- видаляти заняття;
- переглядати список зовнішніх матеріалів до певного твору;
- додавати новий матеріал;
- програвати відео- чи аудіо-матеріали;
- передивлятись матеріали у форматі PDF;
- додавати нотатки у розділ з матеріалами певного твору;
- переглядати матеріали конкретного заняття;
- додавати матеріали до заняття;
- робити нотатки до заняття;

- видаляти матеріали або нотатки з розділів з зовнішніми матеріалами чи з розділу заняття;
- змінювати матеріали або нотатки з розділів з зовнішніми матеріалами чи з розділу заняття;
- додавати відео з YouTube як матеріал до розділів з зовнішніми матеріалами чи до розділу заняття;
- програвати відео з YouTube з матеріалів, за умови доступу до Інтернет;
- переглядати інструкцію користувача до застосунку;
- генерувати звук заданої частоти (за замовчуванням 440 герц) – аналог камертону;
- переглядати інформацію про розробника і застосунок

Системні вимоги. Застосунок має коректно працювати і відображати інформацію на пристроях з ОС Android 7.0 і новіше.

РОЗДІЛ 5. СТВОРЕННЯ БД ДЛЯ ЗАСТОСУНКУ

Для збереження інформації використаємо вбудовану БД SQLite. Потрібно буде зберігати такі сутності: твори, заняття, матеріали до творів та матеріали до занять. У цих сутностей будуть наявні відповідні атрибути. Запишемо зв’язки між сутностями:

- Між творами і заняттями виду «один-до-багатьох»
- Між творами і матеріалами до творів виду «один-до-багатьох»
- Між заняттями і матеріалами до занять виду «один-до-багатьох»

Зробимо діаграму БД для легшого розуміння предметної області (рис. 1)

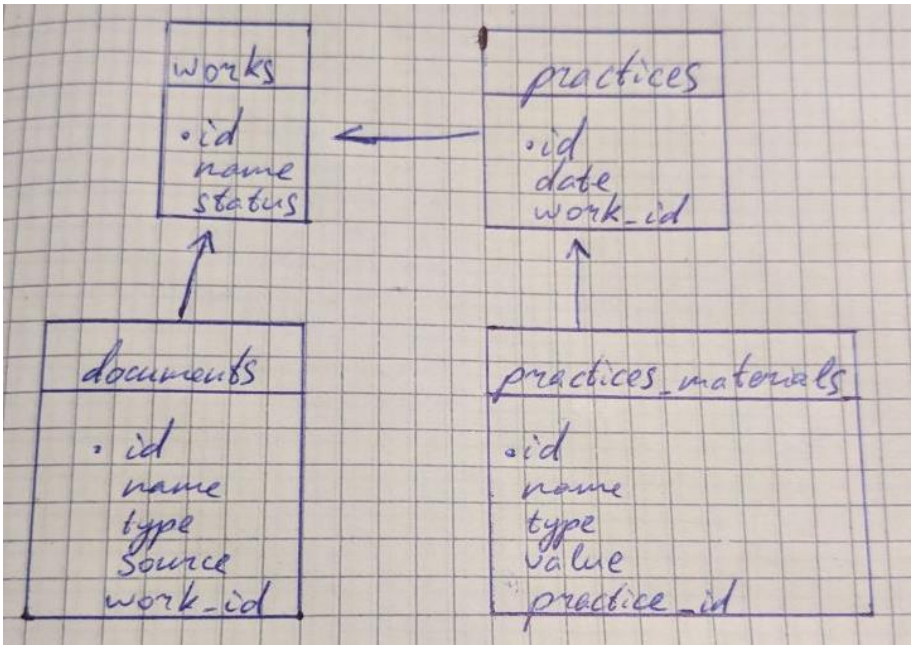


Рисунок 1 - Діаграма БД

У таблицях 1-4 опишемо дані, які будуть зберігатись у наших таблицях.

Таблиця “works” Атрибути	**Тип**	**Опис**
id	INTEGER	ключовий атрибут
name	TEXT	Назва твору, над яким працює користувач
status	INTEGER	Статус (або рейтинг) показник, через який користувач оцінює прогрес

Таблиця 1. Опис таблиці works.

Таблиця “practices” Атрибути	**Тип**	**Опис**
id	INTEGER	ключовий атрибут
date	TEXT	Дата заняття або інша зручна назва для репетиції
work_id	INTEGER	Зовнішній ключ для зв’язку з таблицею works

Таблиця 2. Опис таблиці practices.

Таблиця “documents” Атрибути	**Тип**	**Опис**
id	INTEGER	ключовий атрибут
name	TEXT	Назва файлу
type	INTEGER	Тип файлу, для зручності його представляє ціле число
source	TEXT	URI до ресурсу
work_id	INTEGER	Зовнішній ключ для зв’язку з таблицею works

Таблиця 3. Опис таблиці documents.

Таблиця “practice_materials” Атрибути	**Тип**	**Опис**
id	INTEGER	ключовий атрибут
name	TEXT	Назва файлу
type	INTEGER	Тип файлу, для зручності його представляє ціле число
value	TEXT	URI до ресурсу
practice_id	INTEGER	Зовнішній ключ для зв’язку з таблицею practices

Таблиця 4. Опис таблиці practice_materials.

РОЗДІЛ 6. РОЗРОБКА ЗАСТОСУНКУ

У попередніх розділах було визначено вимоги до застосунку та створено БД SQLite, у якій буде зберігатись інформація. Наступним етапом нашої роботи є створення Android-застосунку. Розробка буде проводитись у IDE Android Studio.

Створимо новий проєкт, вказавши в його налаштуваннях мову програмування Java і мінімальну версію SDK 24. Для використання даного застосунку на пристрої має бути ОС Android 7.0 або новіше.

6.1 Створення класів для представлення даних

Для зручної взаємодії з даними, які зберігаються в БД розробимо відповідні класи: *Work, Practice, WorkDoc, PracticeMaterial*. Усі вони реалізують інтерфейс *Serializable*, щоб можна було передавати об’єкти даних класів між активіті нашого застосунку.

Розглянемо клас *Work*. У ньому є три атрибути: *id, name, status*. Доступ до їх читання або модифікації здійснюється через публічні методи. Переглянути код цього класу можна нижче.

```
public class Work implements Serializable {
    private int id;
    private String name;
    private int status;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getStatus() {
        return status;
```

```
    }

    public void setStatus(int status) {
        this.status = status;
    }
}
```

Аналогічно реалізовано класи *Practice, WorkDoc, PracticeMaterial*. Їхні коди наведено нижче.

```
public class Practice implements Serializable {
    private int id;
    private String date;
    private int workId;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getDate() {
        return date;
    }
    public void setDate(String date) {
        this.date = date;
    }
    public int getWorkId() {
        return workId;
    }
    public void setWorkId(int workId) {
        this.workId = workId;
    }
}


public class WorkDoc implements Serializable {
    private int id;
    private MediaType type;
    private String name;
    private int workId;
    private String source;
    public int getId() {
        return id;
    }
```

```
    public void setId(int id) {
        this.id = id;
    }
    public MediaType getType() {
        return type;
    }
    public void setType(MediaType type) {
        this.type = type;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getWorkId() {
        return workId;
    }
    public void setWorkId(int workId) {
        this.workId = workId;
    }
    public String getSource() {
        return source;
    }
    public void setSource(String source) {
        this.source = source;
    }
}
public class PracticeMaterial implements Serializable {
    private int id;
    private int practiceId;
    private String name;
    private MediaType type;
    private String txt;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getPracticeId() {
        return practiceId;
```

```
    }

    public void setPracticeId(int practiceId) {
        this.practiceId = practiceId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public MediaType getType() {
        return type;
    }
    public void setType(MediaType type) {
        this.type = type;
    }
    public String getTxt() {
        return txt;
    }
    public void setTxt(String txt) {
        this.txt = txt;
    }
}
```

Також опишемо нумерований тип *MediaType*. Він використовується для зручного задання типів файлів у класах *PracticeMaterial* і *WorkDoc*. Наприклад, файл у форматі PDF буде представлятися числом 1, аудіо-файли – числом 2. Так ми не заплутаємось у збереженні та відображенні зовнішніх файлів у застосунку. Код нумерованого типу *MediaType*:

```
public enum MediaType {
    PDF(1), AUDIO(2), VIDEO(3), TXT(4), YT(5);
    private int value;
    private static Map map = new HashMap<>();
    private MediaType(int value) {
        this.value = value;
    }
    static {
        for (MediaType q : MediaType.values()) {
            map.put(q.value, q);
        }
```

```
    }

    public static MediaType valueOf(int q) {
        return (MediaType) map.get(q);
    }
    public int getValue() {
        return value;
    }
}
```

6.2 Створення класу для взаємодії з БД

При розробці застосунку нам буде потрібно взаємодіяти з БД. Тобто виконувати до неї різні запити: читання, створення, оновлення або видалення інформації. Щоб таким чином працювати з БД нам треба підвантажити її до застосунку. Для цього було створено клас *DatabaseHelper*. Цей клас розширює функціональність *SQLiteOpenHelper*. У ньому потрібно задати назву БД, шлях до неї та версію. При запуску програми вона буде перевіряти версію БД та за потреби оновлювати її. Після цього можна буде отримати об’єкт класу *SQLiteDatabase*, за допомогою якого можна здійснювати запити до БД.

Приклад використання даного класу у стартовому активіті застосунку:

```
DatabaseHelper mDBHelper;
SQLiteDatabase mDb;
…
mDBHelper = new DatabaseHelper(this);
try {
    mDBHelper.updateDataBase();
} catch (IOException mIOException) {
    throw new Error("UnableToUpdateDatabase");
}
try {
    mDb = mDBHelper.getWritableDatabase();
} catch (SQLException mSQLException) {
    throw mSQLException;
}
```

6.3 Розробка активіті застосунку

Наш застосунок буде мати кілька різних активіті [6]. Коротко опишемо, що вони будуть відображати, і які переходи між ними будуть можливі.

Стартове активіті – *MainActivity*, на ньому буде відображатися список творів, над якими працює користувач. Список відсортований від найменшого статусу до найбільшого (тобто вгорі будуть твори, над якими було б логічно працювати більше). При кліку на назву твору буде здійснюватись перехід до нового активіті *WorkActivity*, де можна буде переглянути заняття та матеріали до цього твору.

WorkActivity є Tabbed Activity. У ньому будуть дві вкладки: Practices і Media. При обранні кожної з них буде відображатись відповідний фрагмент. У Practices можна буде побачити список занять, що відсортовані в порядку додавання від найновішого до найстаршого. У Media буде наведено список зовнішніх матеріалів. До них можуть належати локальні відео або аудіо, файли PDF, відео з YouTube або просто нотатки.

Після кліку на якесь заняття буде виконано перехід на *PracticeActivity*. В ньому будуть матеріали до заняття. Це теж може бути відео, аудіо, файли PDF, відео з YouTube чи просто текстові нотатки.

На кожному активіті буде плаваюча кнопка «Додати». За допомогою неї можна буде створити новий запис у відповідному активіті. Також на ActionBar буде трьохточкове меню, через яке можна перейти до керівництва користувача або інформації про застосунок.

Редагування та видалення записів зі списків відбувається так. Потрібно натиснути і потримати об’єкт, який хочете змінити. Тоді з’явиться діалогове вікно з двома кнопками: редагувати і видалити.

Зазначимо, що файли PDF відкриваються у окремому активіті *PDFViewActivity*.

Також буде реалізовано можливість передавання даних з застосунку YouTube до нашої програми. Відео з цього сервісу можна буде додавати або до матеріалів до твору, або до матеріалів до самих занять. Для цього створено кілька нових активіті: *GetMediaActivity* і *GetPracticeMaterialWorkActivity*. Перше відповідає за додавання матеріалу до твору, в ньому наводиться перелік робіт, клікнувши на потрібну можна додати до неї нове відео. *GetPracticeMaterialWorkActivity* працює аналогічно, тільки після твору потрібно обрати і заняття, до якого додається відео.

6.4 Робота з БД

Дані у нашому застосунку зберігаються в БД, тому для їх отримання і оновлення потрібно здійснювати запити. В попередніх розділах ми вже розповіли про специфіку SQLite і описали допоміжний клас для оновлення і завантаження БД.

Запити до БД відбуваються за допомогою об’єкту класу *SQLiteDatabase*.

Опишемо метод *setData* у класі *MainActivity*, в якому з БД отримується список творів.

```
void setData(){
    Cursor cursor = mDb.rawQuery("SELECT * FROM works ORDER
BY status ASC", null);
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
        Work work = new Work();
        work.setId(cursor.getInt(0));
        work.setName(cursor.getString(1));
        work.setStatus(cursor.getInt(2));
        items.add(work);
        cursor.moveToNext();
    }
}
```

У об’єкті *mDb* (який відповідає класу *SQLiteDatabase*) викликається метод *rawQuery*, у якому задається SQL запит. Результатом запиту повинні бути повністю всі записи з таблиці works, відсортовані в порядку зростання

поля status. Результат цієї операції передається в об’єкт класу *Cursor*. В цьому об’єкті відбувається взаємодія з отриманими даними.

Далі ми переходимо до першого запису в результаті (*cursor.moveToFirst()*). Після цього в циклі проходимо всі отримані записи. Умовою припинення циклу є досягнення кінця даних.

Через кожний запис ми створюємо новий об’єкт класу *Work*. У нього задаються відповідні поля *id*, *name*, *status*. Після цього до списку *items* (він є глобальним та заданим для всього *MainActivity*) додаємо цей новий твір.

Далі переходимо до нового запису (*cursor.moveToNext()*).

Опишемо створення нових записів у БД. Прикладом залишимо *MainActivity*, у методі *addWork* виводиться діалогове вікно з полем для введення назви нового твору. Далі відбувається перевірка, чи не створено такого твору раніше, якщо новий запис не є дублікатом, то виконується запит до БД. Це реалізовано в таких рядках коду.

```
ContentValues cv = new ContentValues();
cv.put("name", input);
cv.put("status", 1);
long insCount = mDb.insert("works", null, cv);
```

У об’єкті класу *ContentValues* зберігаються дані, які ми хочемо передати в запит у форматі «ключ – значення». Значенням *name* буде введена назва твору, значенням *status* – 1, оскільки користувач тільки почав працювати над твором.

Після цього передаємо даний об’єкт у метод *insert* об’єкту *mDb*, також у цьому методі задаємо назву таблиці, до якої ми хочемо дописати дані.

Даний метод після виконання повертає кількість доданих рядків до таблиці.

Опишемо редагування даних. Також використаємо для прикладу таблицю *works*. Отже, користувач захотів оновити назву твору, тоді після задання нової

назви теж здійснюється перевірка, чи не є вона дублікатом. Якщо вона не повторюється, то виконуємо такий блок коду.

```
ContentValues cv = new ContentValues();
cv.put("name", input);
mDb.update("works", cv, "id=?", new
String[]{String.valueOf(work.getId())});
```

Як і в попередньому випадку дані зберігаються у об'єкті класу *ContentValues*. Оновлення даних відбувається в методі *update* об'єкту класу *SQLiteDatabase*, в ньому потрібно задати назву таблиці, в якій дані будуть оновлюватись, наші дані та умову для визначення потрібного рядку. Як можна побачити, в цьому випадку ми визначаємо рядок по полю *id*, а значення цього поля ми зберегли в об'єкті *work*, який відповідав нашій старій назві твору.

Наведемо приклад видалення даних з БД. Нехай, ми хочемо видалити запис про заняття з таблиці *practices*. Тоді виконується такий код.

```
practices.remove(practice);
mDb.delete("practices", "id=?", new
String[]{String.valueOf(practice.getId())});
```

Це трохи схоже на оновлення даних, але є відмінності у використанні методу *delete* об'єкту *mDb*. Нові дані до запиту не передаються. Тільки відбувається вибір рядку за значенням поля *id*.

Зазначимо, що принцип взаємодії з БД аналогічний у всіх активіті застосунку. Тому було наведено тільки по одному прикладу кожного використаного типу запитів.

6.5 Відображення даних на екрані

Дані, які отримуємо з БД відображаємо у вигляді списку. Це можна реалізувати використавши клас *RecyclerView*. Наведемо приклад використання *RecyclerView* у стартовому активіті нашого застосунку.

Для початку потрібно у файл розмітки XML додати відповідний тег.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/listViewWork"
    android:layout_width="0dp"
    android:layout_height="0dp"

app:layoutManager="androidx.recyclerview.widget.LinearLayoutMana
ger"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    />
```

Створимо відповідний об'єкт у методі *onCreate* нашого активіті.

```
RecyclerView recyclerView =
(RecyclerView)findViewById(R.id.listViewWork);
WorkAdapter.OnWorkClickListener workClickListener = new
WorkAdapter.OnWorkClickListener(){
    @Override
    public  void onWorkClick(Work work){
        Intent intent = new Intent(getApplicationContext(),
WorkActivity.class);
        intent.putExtra(Work.class.getSimpleName(), work);
        startActivity(intent);
    }
};
WorkAdapter adapter = new WorkAdapter(this, items,
workClickListener, this);
recyclerView.setAdapter(adapter);
```

Крім ініціалізації об'єкту *RecyclerView* тут описано використання класу *WorkAdapter*. У ньому задається взаємодія з елементами списку. Використовується новий графічний елемент, який вже відповідає кожному елементу списку. В нашому випадку це файл work_item.xml, в ньому описано три елементи: картинка, що позначає пункт списку, текстові поля для назви твору і оцінки його прогресу.

WorkAdapter наслідується від класу *RecyclerView.Adapter*, у ньому перевизначаються методи *onCreateViewHolder*, *onBindViewHolder* і

getItemCount. В них описується взаємодія з класом *ViewHolder*, який представляє власне сам графічний елемент зі списку.

У *onCreateViewHolder* визначається який файл розмітки буде використовуватись.

```
public ViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
    View view = inflater.inflate(R.layout.work_item, parent, false);
    return new ViewHolder(view, recyclerViewInterface);
}
```

У *onBindViewHolder* встановлюються значення для полів у елементі, як назва твору. Також визначається лістенер на клік по елементу.

```
public void onBindViewHolder(ViewHolder holder, int position) {
    Work work = works.get(position);
    holder.nameView.setText(work.getName());
    holder.statusView.setText(Integer.toString(work.getStatus()));
    holder.itemView.setOnClickListener(new View.OnClickListener(){
        @Override
        public void onClick(View v)
        {
            onClickListener.onWorkClick(work);
        }
    });
}
```

getItemCount повертає кількість елементів у списку.

```
public int getItemCount() {return works.size();}
```

Тепер після опису основної частини *WorkAdapter* зрозумілий блок коду з *MainActivity*.

Логіку обробки коротких кліків по елементам списку було базово описано. Тепер пояснимо обробку довгих натискань (нагадаємо, що вони використовуються для редагування та видалення записів).

Для цього створимо інтерфейс *RecyclerViewInterface*.

```
interface RecyclerViewInterface {
    void onItemLongClick(int position);
}
```

Як можна побачити, у ньому є метод *onItemLongClick*, який потрібно буде описати у активіті, в якому ми хочемо обробляти довгі натискання. Таким є *MainActivity*. Опишемо цей метод в ньому.

```
public void onItemLongClick(int position) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    View layout = getLayoutInflater().inflate(R.layout.custom_alert_edit,
null);
    ImageButton editb = layout.findViewById(R.id.edit_btn);
    builder.setView(layout);
    AlertDialog dialog = builder.create();
    editb.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            editWork(position);
            dialog.dismiss();
        }
    });
    dialog.show();
}
```

Можна побачити, що в ньому після довгого натиску описано відображення діалогу з можливістю почати редагування запису.

Зв’язок даного методу з класом *WorkAdapter* реалізується через передачу класу, що реалізує інтерфейс. Здійснюється під час створення об’єкту цього класу. Тоді викликається його конструктор, де можна побачити як передається інтерфейс:

```
WorkAdapter(Context context, List<Work> works,
OnWorkClickListener onClickListener, RecyclerViewInterface
recyclerViewInterface) {
    this.onClickListener = onClickListener;
    this.works = works;
    this.inflater = LayoutInflater.from(context);
    this.recyclerViewInterface = recyclerViewInterface;
}
```

Таким чином відбувається відображення даних у вигляді списків і у інших розділах нашого застосунку.

6.6 Відображення відео- та аудіо-файлів у застосунку

Перегляд відео- та аудіо-файлів є нетривіальною задачею. Для цього було використано бібліотеку ExoPlayer [9]. Це медіапрогравач, що може бути використаний у застосунках, які створюються для ОС Android. Він надає альтернативний до стандартного MediaPlayer API для програвання відео і аудіо, що можуть знаходитись локально чи в мережі. У даного плеєру є широкі можливості для кастомізації і налаштування.

Для початку додамо залежність у файл build.gradle на рівні модулю. Це буде мати такий вигляд:

```
implementation 'com.google.android.exoplayer:exoplayer:2.17.1'
```

Далі створимо файл розмітки XML, де опишемо елемент, який буде містити в собі програвач відео (та аудіо). Кореневим тегом буде *ConstraintLayout*. У ньому буде два елементи: текстове поле для назви файлу та тег плеєру. Наведемо останній:

```
<com.google.android.exoplayer2.ui.StyledPlayerView
    android:id="@+id/playerv"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:use_artwork="false"
    app:auto_show="true"
    app:hide_on_touch="false"
    app:shutter_background_color="#0000AA"
    app:layout_constraintTop_toBottomOf="@id/materialvideoname"/>
```

Опишемо взаємодію з програвачем у коді. Одним з класів, які використовують плеєр є *MyPracticeMaterialAdapter*. У ньому описуються елементи, які є у списку матеріалів до заняття. Зрозуміло, що ці елементи можуть виглядати по різному. Одним з можливих випадків є, коли вони є

відеоматеріалом. Це якраз випадок використання функціональності бібліотеки ExoPlayer.

Створимо клас *ExoPlayerViewHolder*, що наслідується від *RecyclerView.ViewHolder*. У ньому буде описуватись відео- або аудіо-матеріал для відтворення. Наведемо код цього класу. Він досить короткий.

```
public class ExoPlayerViewHolder extends RecyclerView.ViewHolder{
    TextView titleTextView;
    StyledPlayerView playerv;
    public ExoPlayerViewHolder(@NonNull View itemView){
        super(itemView);
        titleTextView = itemView.findViewById(R.id.materialvideoname);
        playerv = itemView.findViewById(R.id.playerv);
        itemView.setOnLongClickListener(new
View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View v) {
                if(recyclerViewInterface != null){
                    int pos = getAdapterPosition();
                    if(pos != RecyclerView.NO_POSITION){
                        recyclerViewInterface.onItemLongClick(pos);
                    }
                }
                return true;
            }
        });
    }
}
```

StyledPlayerView – це один з класів бібліотеки ExoPlayer, який зручно використовувати для візуального зображення програвача. Повернемось до класу *MyPracticeMaterialAdapter*.

Створимо об’єкт класу *ExoPlayer*, який буде глобальним у класі *MyPracticeMaterialAdapter*.

```
public ExoPlayer player;
```

У методі *onCreateViewHolder* вкажемо, що потрібно використовувати клас *ExoPlayerViewHolder*, коли маємо відповідний тип матеріалу:

```
modelView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.materialvideo,
parent, false);
viewHolder = new ExoPlayerViewHolder(modelView);
```

У методі *onBindViewHolder* відбувається задання даних для відео- або аудіо-матеріалу. Ми отримуємо потрібний об'єкт класу *PracticeMaterial*. Далі встановлюємо потрібні значення в текстове поле елементу, визначаємо параметри програвача, передаємо його в об'єкт класу *ExoPlayerViewHolder*. Це можна побачити в блоці коду, який наведено нижче.

```
PracticeMaterial practiceMaterial = listItems.get(position);
ExoPlayerViewHolder exoPlayerViewHolder =
(ExoPlayerViewHolder)holder;
exoPlayerViewHolder.titleTextView.setText(practiceMaterial.getName())
;
player = new ExoPlayer.Builder(context).build();
exoPlayerViewHolder.playerv.setPlayer(player);
exoPlayerViewHolder.playerv.setControllerHideOnTouch(false);
exoPlayerViewHolder.playerv.setResizeMode(AspectRatioFrameLayout.
RESIZE_MODE_FIXED_WIDTH);
exoPlayerViewHolder.playerv.setMinimumHeight(24);
MediaItem mediaItem = MediaItem.fromUri(practiceMaterial.getTxt());
player.setMediaItem(mediaItem);
player.prepare();
```

Щоб програвач зупинявся, коли ми здійснюємо перехід з його активіті, перевизначимо методи *onPause* та *onDestroy* у класі *PracticeActivity*. У них ми будемо зупиняти об'єкт player класу *ExoPlayer* (цей об'єкт було створено в класі *MyPracticeMaterialAdapter*). Наведемо дані методи.

```
protected void onPause() {
    super.onPause();
    try {
        myPracticeMaterialAdapter.player.stop();
    } catch (Exception e){}
}
protected void onDestroy() {
    super.onDestroy();
    try {
```

```
            myPracticeMaterialAdapter.player.stop();
        } catch (Exception e){}
        mDBHelper.close();
}
```

Окремим випадком є, коли ми маємо відеоматеріал з YouTube. Опишемо відтворення такого типу даних в наступному підрозділі.

6.7 Відтворення відео з сервісу YouTube

Для відтворення відео з сервісу YouTube у застосунках на ОС Android було розроблено офіційний API від Google – YouTube Android Player API. Проте є певні мінуси у його використанні. Одним з найбільших недоліків є те, що його використання у застосунку може бути платним. Хоч для підключення цього API додаткових коштів не потрібно, при великій кількості запитів даний сервіс може почати відраховувати гроші з карти. Ще одним мінусом є нерегулярність оновлень даного API. Одним з виходів для відтворення відео з YouTube без використання вищеописаного API є вбудований клас *WebView*, що є аналогом браузеру. В такому випадку в нашому застосунку буде відкриватися вікно зі сторінкою з потрібним відео на YouTube, але разом з ним буде багато додаткової непотрібної інформації, на зразок рекомендованих відео, коментарів тощо. Виходом з даної ситуації буде використання бібліотеки android-youtube-player [10], яка є API для програвання відео з YouTube всередині Android WebView. При використанні цієї бібліотеки відео буде відображатися без зайвої інформації, майже як у стандартному застосунку YouTube.

Для початку роботи з даною бібліотекою потрібно додати залежність у файл build.gradle.

```
implementation
'com.pierfrancescosoffritti.androidyoutubeplayer:core:11.0.1'
```

Як і звичайні аудіо- чи відео-матеріали відео з YouTube будуть відображатися у списку матеріалів до занять, чи твору взагалі. Наведемо приклад використання даних відео як і в попередньому підрозділі.

Оскільки відео з YouTube зберігаються не локально, для взаємодії з ними потрібен доступ до Інтернет. Для цього потрібно задати відповідний дозвіл у файлі AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```

У файлі materialyt.xml додамо один тег *YouTubePlayerView*, в якому опишемо правила відображення та розміщення відео на екрані.

```
<com.pierfrancescosoffritti.androidyoutubeplayer.core.player.views.YouTubePlayerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/webv"
    app:autoPlay="false"
    android:layout_margin="7sp"
    android:paddingTop="10sp"
    android:paddingBottom="10sp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    android:longClickable="true"
    android:background="@drawable/list_back"
    />
```

Далі перейдемо до класу *MyPracticeMaterialAdapter*, всередині якого додамо клас *YTViewHolder*. Він буде мати такий вигляд.

```
public class YTViewHolder extends RecyclerView.ViewHolder{
    YouTubePlayerView webView;
    YouTubePlayer youTubePlayer;
    String currentVideoId;
    public YTViewHolder(@NonNull YouTubePlayerView itemView){
        super(itemView);
        webView = itemView;
        webView.addYouTubePlayerListener(new
AbstractYouTubePlayerListener() {
            @Override
            public void onReady(@NonNull YouTubePlayer
initializedYouTubePlayer) {
                youTubePlayer = initializedYouTubePlayer;
                youTubePlayer.cueVideo(currentVideoId, 0);
```

```
            }
        });
        itemView.setOnLongClickListener(new
View.OnLongClickListener() {
            @Override
            public boolean onLongClick(View v) {
                if(recyclerViewInterface != null){
                    int pos = getAdapterPosition();
                    if(pos != RecyclerView.NO_POSITION){
                        recyclerViewInterface.onItemLongClick(pos);
                    }
                }
                return true;
            }
        });
    }
    void cueVideo(String videoId) {
        currentVideoId = videoId;
        if(youTubePlayer == null)
            return;
        youTubePlayer.cueVideo(videoId, 0);
    }
}
```

Як можна побачити, у ньому серед атрибутів є об'єкт класу *YouTubePlayerView*, який відповідає елементу з файлу materialyt.xml, інтерфейс *YouTubePlayer*, через який задаємо налаштування відтворення відео. Також є рядок *currentVideoId*, що має зберігати id відео на сервісі YouTube. Метод *cueVideo* завантажує відео, але не починає його автоматичне відтворення.

У клас *MyPracticeMaterialAdapter* потрібно додати атрибут класу *Lifecycle*, для легкого забезпечення правильної поведінки нашого плеєра в процесі використання застосунку.

Наведемо фрагмент з методу *onCreateViewHolder*, який виконується при обробці матеріалу-відео з YouTube.

```
YouTubePlayerView youTubePlayerView = (YouTubePlayerView)
LayoutInflater.from(parent.getContext()).inflate(R.layout.materialyt,
parent, false);
lifecycle.addObserver(youTubePlayerView);
```

```
viewHolder = new YTViewHolder(youTubePlayerView);
return viewHolder;
```

Також опишемо роботу з відео у методі *onBindViewHolder*.

```
PracticeMaterial practiceMaterial = listItems.get(position);
YTViewHolder ytViewHolder = (YTViewHolder)holder;
String source = practiceMaterial.getTxt();
String id = source.substring(source.indexOf("be/") + 3);
ytViewHolder.cueVideo(id);
```

Можна побачити, що в даному фрагменті ми фіксуємо наш матеріал, дізнаємось посилання на відео з YouTube та потім отримуємо з нього id. За знайденим id починається підвантаження відео.

6.8 Відображення PDF файлів у застосунку

У нашому застосунку потрібно реалізувати підтримку відображення PDF файлів. Це буде використано в таких розділах, як матеріали до твору і матеріали до заняття. Також керівництво користувача буде зберігатись в окремому файлі, який можна буде переглядати через застосунок.

Для перегляду PDF файлів було використано бібліотеку android-pdf-viewer. За її допомогою можна зручно відкривати файли у нашому застосунку. Реалізовано підтримку жестів: перегортання сторінок, збільшення і зменшення масштабу. Розглянемо її використання.

Спочатку додамо залежність в файл build.gradle.

```
implementation 'com.github.barteksc:android-pdf-viewer:3.2.0-beta.1'
```

На прикладі PDFViewActivity опишемо роботу з файлами. У відповідний файл розмітки XML вставимо тег *PDFView*.

```
<com.github.barteksc.pdfviewer.PDFView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/pdfView"
    />
```

Перейдемо до класу *PDFViewActivity*, нам потрібно створити об’єкт класу *PDFView*, за допомогою якого будемо взаємодіяти з файлом PDF.

```
PDFView pdfView;
```

У методі *onCreate* визначимо відповідний графічний елемент та вкажемо налаштування відображення PDF файлу.

```
pdfView = findViewById(R.id.pdfView);
pdfView.fromUri(Uri.parse(practiceMaterial.getTxt()))
        .enableSwipe(true)
        .enableDoubletap(true)
        .swipeHorizontal(true)
        .password(null)
        .pageSnap(true)
        .autoSpacing(true)
        .pageFling(true)
        .load();
```

Як можна побачити, ми задаємо розташування файлу, вказуємо що можливе використання жесту “свайп”, подвійного кліку та те що перегортання сторінок відбувається по горизонталі. Також визначено, що у файлі відсутній пароль та сторінки займають всю можливу площу елемента. В кінці методом *load* вказуємо, що потрібно завантажити файл.

6.9 Імпортування відео з YouTube до застосунку

Передавання даних з одного застосунку до іншого є досить нетривіальною задачею для розробників програм для ОС Android. Додавати відео з сервісу YouTube до нашого застосунку можна через звичайну функцію «Поширити» з програми YouTube. Користувач при перегляді потрібного відео може натиснути цю кнопку та обрати серед можливих варіантів нашу програму. Як було описано раніше, можна додавати відео або в матеріали до всього твору, або до певного заняття. За це будуть відповідати різні активіті. Наведемо приклад взаємодії з YouTube активіті, яке додає відео-матеріал до твору загалом.

По перше, нам потрібно створити нове активіті, назвемо його *GetMediaActivity*. Зовнішнім виглядом воно повинне бути схожим на стартовий

екран нашого застосунку зі списком творів. Обравши певний твір, користувач додасть до нього поширене відео.

Для можливого отримання даних з інших застосунків потрібно змінити файл AndroidManifest.xml. У ньому є список активіті застосунку. Серед них є і *GetMediaActivity*. Змінимо його тег на наступний:

```
<activity
    android:name=".GetMediaActivity"
    android:exported="true"
    android:label="Dyplom App: Media">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data
            android:host="www.youtube.com"
            android:mimeType="text/*" />
    </intent-filter>
</activity>
```

Атрибут *exported* із значенням *true* вказує, що дане активіті може бути запущене з іншого застосунку. У тезі *intent-filter* описано які дані буде отримувати програма, в даному випадку це буде текст (ми будемо отримувати посилання на відео з YouTube).

У методі *onCreate* класу *GetMediaActivity* додамо такі рядки коду:

```
if (Intent.ACTION_SEND.equals(action) && type != null) {
    if ("text/plain".equals(type)) {
        handleSendText(intent); // Handle text being sent
    }}
```

Якщо у активіті передається текст, буде виконано метод handleSendText. Наведемо його код.

```
void handleSendText(Intent intent){
    Context context = this;
    String sharedText = intent.getStringExtra(Intent.EXTRA_TEXT);
    setData();
    WorkAdapter.OnWorkClickListener workClickListener = new
WorkAdapter.OnWorkClickListener(){
```

```
        @Override
        public  void onWorkClick(Work work){
            ContentValues cv = new ContentValues();
            cv.put("name", "YouTube video");
            cv.put("type", 5);
            cv.put("source", sharedText);
            cv.put("work_id", work.getId());
            long insCount = mDb.insert("documents", null, cv);
            Toast.makeText(context, "Media added",
Toast.LENGTH_SHORT).show();
            Intent intentn = new Intent(context, MainActivity.class);
            intentn.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK
|Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(intentn);
            finish();
        }
    };
    WorkAdapter adapter = new WorkAdapter(this, items,
workClickListener, this);
    recyclerView.setAdapter(adapter);
}
```

У цьому методі ми отримуємо текст, який передано до застосунку. Після цього отримується список творів, над якими працює користувач. Також визначається обробка кліку на певний твір – до нього буде додано новий матеріал. Після додавання матеріалу буде запущене стартове активіті, так само як при простому запуску застосунку.

6.9 Розробка функціональності камертону

Створимо нове активіті, яке можна буде використовувати за потреби налаштувати інструмент. У ньому буде кнопка – відтворити звук та поле для задання потрібної частоти (у герцах).

Опишемо як генерується звук у коді застосунку. В методі *onCreate* додамо прослуховувач кліків по кнопці.

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        generateTone(Double.valueOf(freqtext.getText().toString()));
```

```
        playSound();
    }
});
```

У класі активіті створимо такі глобальні атрибути:

```
AudioTrack audioTrack;
int duration = 2;
int sampleRate = 44100;
int numberOfSamples = duration * sampleRate;
double[] sample = new double[numberOfSamples];
byte[] generatedSound = new byte[2 * numberOfSamples];
```

У методі *generateTone* ми за вказаною частотою генеруємо звук. Використовується клас для обчислення математичних операцій *Math*.

```
void generateTone(double frequency){
    for (int i = 0; i < numberOfSamples; ++i) {
        sample[i] = Math.sin(2 * Math.PI * i / (sampleRate / frequency));
    }
    int index = 0;
    for (double dVal : sample){
        short val = (short)(dVal * 32767);
        generatedSound[index++] = (byte)(val & 0x00ff);
        generatedSound[index++] = (byte)((val & 0xff00) >>> 8);
    }
}
```

У методі *playSound* за допомогою об'єкту *audioTrack* відтворюється згенерований звук.

```
void playSound(){
    audioTrack = new AudioTrack(AudioManager.STREAM_MUSIC,
            sampleRate, AudioFormat.CHANNEL_OUT_MONO,
            AudioFormat.ENCODING_PCM_16BIT, generatedSound.length,
            AudioTrack.MODE_STATIC);
    audioTrack.write(generatedSound, 0, generatedSound.length);
    audioTrack.play();
}
```

У цьому розділі ми описали деталі розробки нашого застосунку. Зрозуміло, що деякі базові аспекти було розглянуто досить поверхнево, проте звернуто увагу на технологічні процеси додавання нетривіальної функціональності до програми.

Ми навели деякі принципові фрагменти коду, переглянути код нашого застосунку повністю можна на GitHub за посиланням: https://github.com/illia20/DiaryOrganizerOfMusiciansPractice.

У наступному розділі опишемо процес використання нашого застосунку.

РОЗДІЛ 7. ОПИС РОБОТИ ЗАСТОСУНКУ

Опишемо роботу нашої програми у вигляді короткого керівництва користувача.

При запуску застосунку користувач може бачити список творів, над якими працює. Поряд з назвою твору відображається його статус, іншими словами оцінка прогресу у роботі. Твори відсортовані в порядку від найменшого статусу до найбільшого (тобто твір, над яким логічно було б працювати більше буде знаходитись вгорі). Стартовий екран застосунку можна побачити на рис. 2.

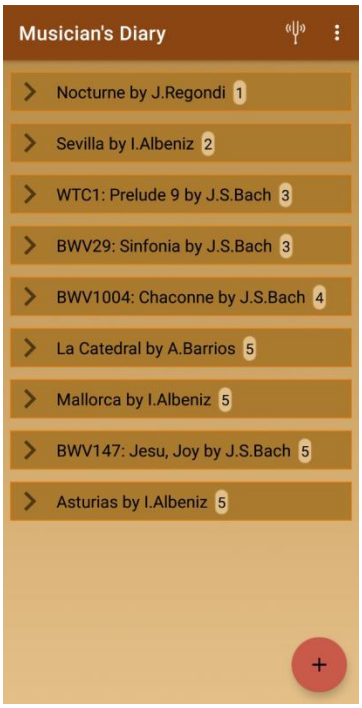


Рисунок 2 – стартовий екран застосунку зі списком творів

Щоб додати новий твір до списку потрібно натиснути на кнопку справа внизу екрану, тоді буде виведене діалогове вікно (можна побачити на рис. 3), у яке потрібно додати назву твору. За замовчуванням твору буде встановлено статус 1, який є найнижчим (найвищим є 5). Якщо користувач вкаже назву твору, який вже є у списку, нового елементу до списку додано не буде.

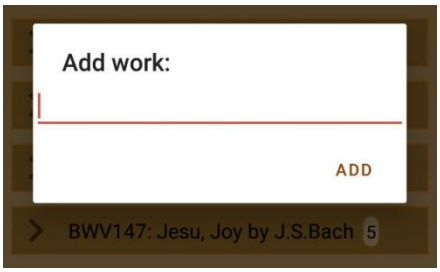


Рисунок 3 – діалогове вікно для додавання нового твору

Щоб змінити назву твору потрібно натиснути на елемент списку і потримати. Буде виведено діалогове вікно з кнопкою «редагувати» (приклад можна побачити на рис. 4), якщо на неї натиснути можна буде ввести нову назву твору.

Рисунок 4 – діалогове вікно з кнопкою «редагувати»

Перейдемо до екрану роботи над твором (для цього потрібно натиснути на один з творів у списку на стартовому екрані). На екрані буде зображено дві вкладки: «Practices» і «Media». На першій можна буде побачити список занять та оцінку статусу твору (рис. 5). Оцінку можна змінити, увівши у текстове поле нове значення статусу (воно може бути цілим числом від 1 до 5 включно). Взаємодія зі списком занять відбувається схожим чином з роботою з творами на попередньому екрані. Якщо потрібно відредагувати заняття чи видалити, потрібно натиснути і потримати на елемент списку, після чого буде виведено діалогове вікно з відповідними двома кнопками (можна побачити на рис. 6).

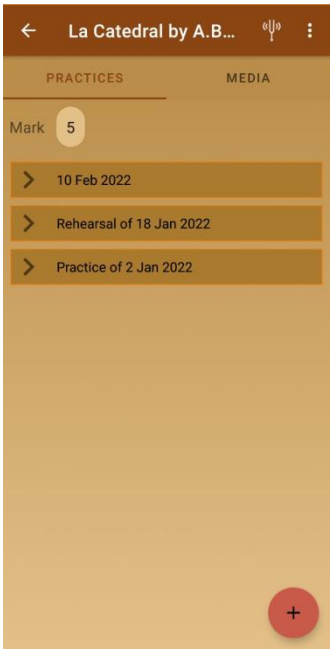


Рисунок 5 – сторінка зі списком занять над твором

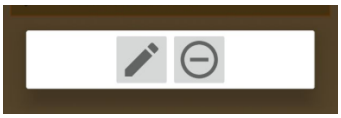


Рисунок 6 – діалогове вікно для редагування чи видалення заняття

У вкладці «Media» (приклад наведено на рис. 7) можна додавати різні матеріали до твору. Ними можуть бути локальні відео- чи аудіо-файли, файли PDF або просто текстові нотатки. Щоб додати новий матеріал потрібно натиснути на кнопку справа внизу екрану, далі з’явиться діалогове вікно (приклад такого вікна можна побачити на рис. 8), де потрібно обрати тип нашого файлу (матеріалу). В залежності від обраного пункту буде запущено застосунок – файловий менеджер, чи виведено поле для введення тексту (якщо ми хочемо додати нотатку). У файловому менеджері (приклад зображено на рис. 9) можна буде зручно обрати потрібний файл. Наприклад, якщо ми обрали тип файлу відео – відповідні файли будуть підсвічуватись у стандартному файловому менеджері ОС Android.

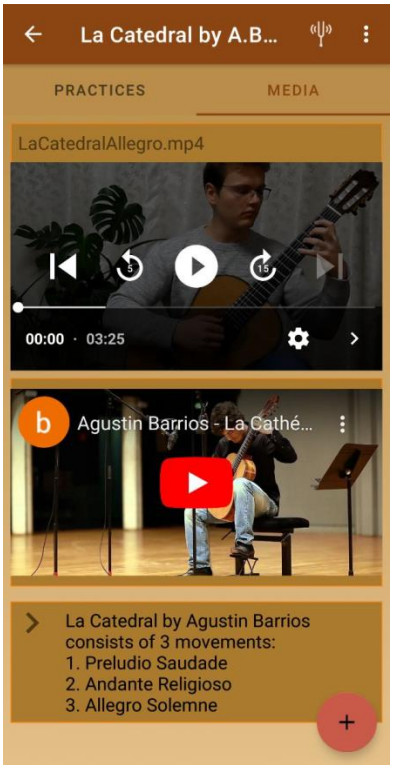


Рисунок 7 – вкладка Media

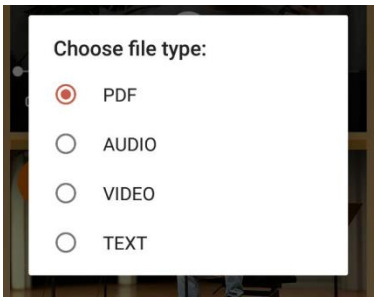


Рисунок 8 – діалогове вікно для обрання типу нового матеріалу

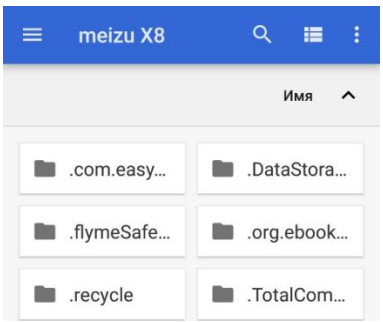


Рисунок 9 – фрагмент файлового менеджера

Змінити чи видалити матеріал можна аналогічно відповідній процедурі зі списком занять.

Також зазначимо, що матеріалом можуть бути відео з сервісу YouTube, але вони додаються до застосунку через функцію «Поширити» програми, через яку ми знайшли це відео.

Також на цьому екрані відбувається видалення творів, якщо нам це потрібно – натискаємо на кнопку справа вгорі екрану (значок «Три точки») і обираємо пункт «Remove Work» (рис. 10). Зазначимо, що тоді буде видалено і всю інформацію про заняття і матеріали цього твору із застосунку.

Рисунок 10 – меню з можливістю видалення твору

При натисканні на заняття здійснюється перехід на екран з інформацією про нього (рис. 11). Тут теж можна додавати відео- і аудіо-матеріали, файли PDF чи нотатки. Також можуть бути імпортовані відео з YouTube.

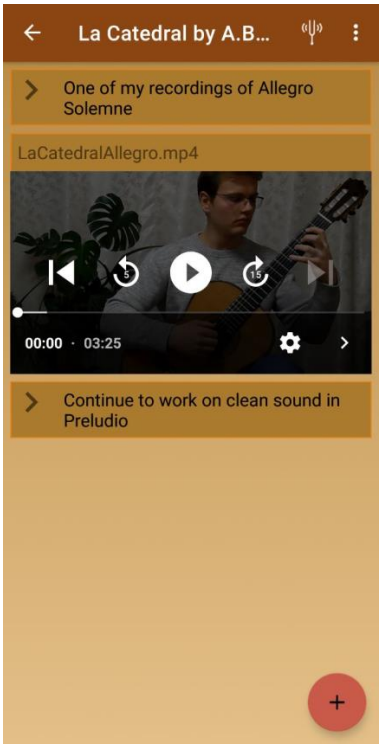


Рисунок 11 – приклад екрану з інформацією про окреме заняття

Перегляд відео- і аудіо-матеріалів, відео з YouTube і нотаток відбувається на цьому ж екрані. Елементи керування стандартні для ОС Android. Щоб переглянути файл PDF, потрібно натиснути на кнопку, яка розташована справа від його назви (рис. 12). Тоді у новому екрані буде відкрито даний файл, який можна буде зручно переглядати (рис. 13).

Рисунок 12 – представлення матеріалу типу PDF та кнопка для перегляду файлу

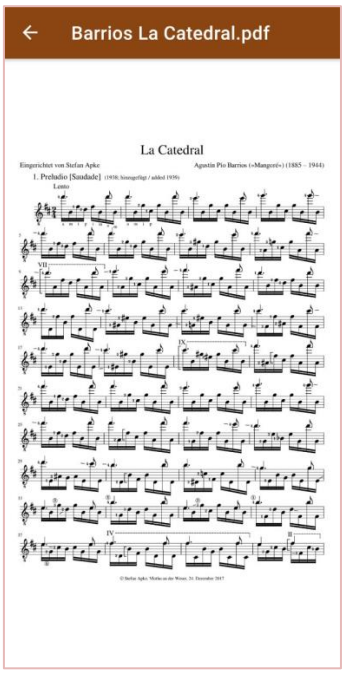


Рисунок 13 – перегляд PDF файлу у застосунку

На всіх екранах застосунку можна побачити кнопку для переходу до активіті камертону та трьох-точкове меню (рис. 14), в якому можна перейти до посібника користувача («Manual») або екрану з короткою інформацією про застосунок («About»), останню можна побачити на рис. 15.

Рисунок 14 – трьохточкове меню на стартовому екрані

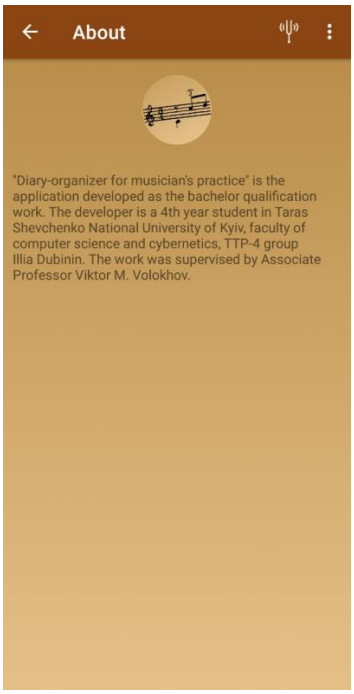


Рисунок 15 – екран з інформацією про застосунок

Екран для використання камертону виглядає так: велика кругла кнопка з рисунком камертону для початку відтворення звуку заданої частоти та поле введення для задання потрібної частоти в герцах. Вигляд цього екрану можна побачити на рис. 16.

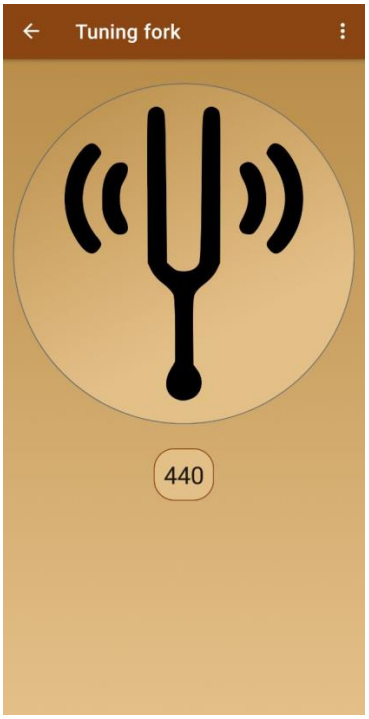


Рисунок 16 – екран для роботи з камертоном

ВИСНОВКИ

У результаті виконання даної роботи ми розробили застосунок для ОС Android – «Щоденник-органайзер для занять музикою». Він може використовуватись музикантами для збереження та структуризації матеріалів до свої занять або репетицій. Також зазначимо, що програма можлива для використаннями як музикантами-початківцями, так і для виконавців рівня вищих навчальних закладів і професіоналів.

Оцінимо отримані результати та їх відповідність сучасному рівню технічних і наукових знань та технологій. Розроблений застосунок може забезпечувати виконання свого основного завдання – збереження інформації про заняття користувачів-музикантів. Можлива взаємодія з файлами відео- і аудіо-матеріалів, файлів типу PDF та відео з сервісу YouTube, як описано у вимогах. Основна частина додатку розроблена за допомогою мови програмування Java, яку можна цілком вважати актуальною, тому що в наш час вона є одним з найпопулярніших засобів створення застосунків для ОС Android. Для збереження даних у програмі використано СКБД SQLite. Цей продукт є одним з основних способів роботи з інформацією в мобільних застосунках. Розробка здійснювалась у IDE Android Studio, яка є офіційним засобом написання програм для ОС Android. Таким чином, можна зробити висновок, що результати роботи цілком відповідають сучасному рівню технічних і наукових знань.

Розглянемо можливі сфери використання нашого застосунку. Як ми зазначили в призначенні розроблюваної системи, майбутніми користувачами програми можуть бути музиканти не залежно від їхнього рівня та інструменту. Застосунок може допомогти у збереженні результатів роботи та її процесу. Програма забезпечує структуризацію роботи по різним творам. До кожного твору можна додавати заняття в яких над ним працювали та відповідні матеріали. Детально процес взаємодії з застосунком було описано в Розділі 7.

Майбутня підтримка застосунку буде визначена вимогами користувачів, які можуть бути уточнені в процесі використання програми.

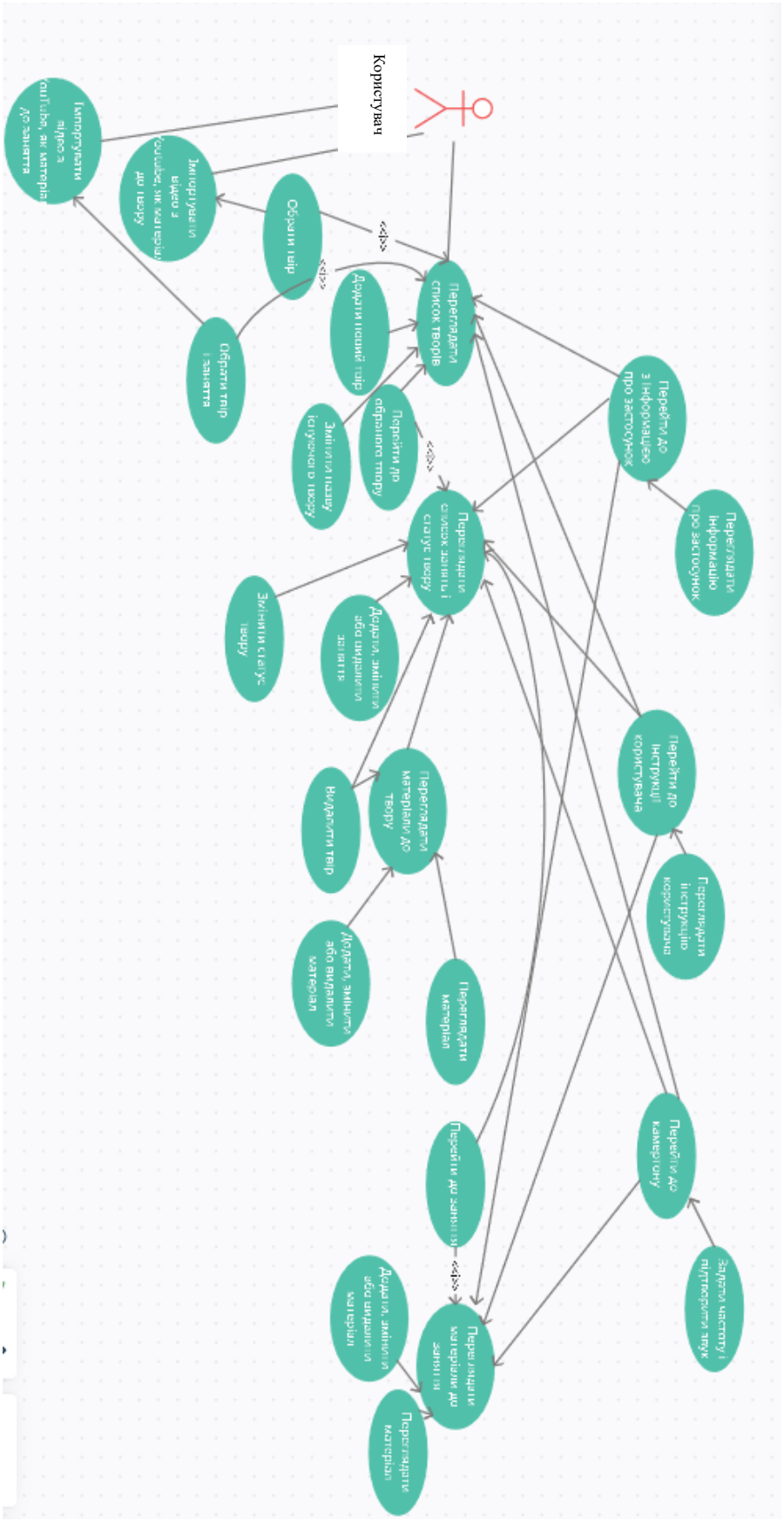
Визначимо доцільність досліджень чи розробок за нашою тематикою. Тема застосунку є важливою, тому що на ринку бракує продуктів з аналогічною функціональністю. Також розробка додатків для ОС Android є актуальним напрямом для досліджень, оскільки кількість пристроїв з даною ОС постійно збільшується. Мова розробки для Android, наприклад Java, теж розвивається, з’являються нові бібліотеки для розширення можливостей програмістів та полегшення процесу розробки.

Отже, підводячи підсумки роботи вкажемо, що було досягнуто завдань, встановлених на початку процесу розробки. Ми використали актуальні технології для розробки застосунків для ОС Android та створили працюючу програму з потрібною функціональністю.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. History of Mobile Applications by prof. John F. Clark – University of Kentucky, 2012
2. 10 best diary apps and journal apps for Android – Android Authority : веб-сайт. URL: https://www.androidauthority.com/best-diary-apps-journal-apps-android-892375/
3. Introduction to Programming in Java: An Interdisciplinary Approach / Robert Sedgewick, Kevin Wayne – Princeton University, 2007 – 736 с.
4. andbook! : Android Programming / Nicolas Gramlich, 62 с.
5. Android Developer Fundamentals Course: Practical Workbook / Google Developer Training Team, - Google, 2016 – 566 с.
6. Android Studio 3.0 Development Essentials Android 8 Edition / Neil Smyth – Payload Media, Inc., 2017 – 864 с.
7. Gradle Essentials / Kunal Dabir Abhinandan – Packt Publishing Ltd., 2015 – 274 с.
8. Using SQLite / Jay A. Kreibich, - O'Reilly, 2010 – 528 с.
9. ExoPlayer : веб-сайт. URL: https://exoplayer.dev/
10. android -youtube-player: YouTube Player library for Android and Chromecast, stable and customizable : веб-сайт. URL: https://github.com/PierfrancescoSoffritti/Android-YouTube-Player

ДОДАТОК А Use-Case діаграма застосунку



Примітка. Зв’язки між прецедентами типу «include» позначено «і», решта зв’язків має тип «extend».