Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили

**М. Л. Дворецький, Ю. О. Нездолій,
С. В. Дворецька, І. О. Кандиба**

# РОЗРОБКА МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ OS Android

*Навчальний посібник*

*Для підготовки бакалаврів у галузі знань
«Інформаційні технології»*

ЧНУ
ВИДАВНИЦТВО
Миколаїв – 2021

УДК 004.451.057.5+004.58](075.8)Android
Р 62

*Рекомендовано до друку вченою радою Чорноморського національного університету імені Петра Могили (витяг з протоколу № 9 від 28.05.2020).*

***Рецензенти:***

**Субботін С. О.,** *д-р техн. наук, професор, завідувач кафедри програмних засобів Національного університету «Запорізька політехніка».*

**Ходаков В. Є.,** *д-р техн. наук, професор, професор кафедри фундаментальних дисциплін Морської академії післядипломної освіти ім. контр-адмірала Ф. Ф. Ушакова.*

Р 62 Дворецький М. Л., Нездолій Ю. О., Дворецька С. В., Кандиба І. О. Розробка мобільних застосунків для OS Android : навч. посіб. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2021. – 140 с.

**ISBN** 978-966-336-419-3

*Практичний посібник присвячений розробці мобільних застосунків для ОС Андроїд та складається з 4-х розділів. У першому розділі розглянуто загальні етапи створення простого андроїд-застосунку, взаємодію макета та активносні, передачу даних між активностями за допомогою інтентів, а також запуск активностей інших застосунків. Наведено особливості життєвого циклу активності та його вплив на роботу застосунку. Другий розділ присвячений розробці користувацького інтерфейсу та розглядає використання layout-ів різного типу. Наводяться приклади використання спискових представлень та панелей інструментів. У третьому розділі увага приділяється роботі із базами даних на прикладі БД SQLite, виконано збереження та подальша модифікація даних застосунку. У останньому, четвертому розділі розглядається клієнт-серверний режим роботи, взаємодія андроїд-застосунку із back-end та клієнт-серверною реляційною СКБД, а також розглянуто реалізацію аутентифікації та авторизації користувачів.*

УДК 004.451.057.5+004.58](075.8)Android



**ISBN 978-966-336-419-3**

# *ЗМІСТ*

ВСТУП ..... 4
1. РОЗДІЛ 1. СТВОРЕННЯ ПРОСТОГО ЗАСТОСУНКУ ..... 7
1.1. Загальні відомості та «Hello world» ..... 7
1.2. Взаємозв’язок макета та активності ..... 17
1.3. Інтенти та передача даних між активностями ..... 28
1.4. Життєвий цикл активності ..... 35
РОЗДІЛ 2. МАКЕТИ ТА РОЗРОБКА ІНТЕРФЕЙСІВ ..... 44
2.1. Макети та зовнішній вигляд ..... 44
2.2. Макети з обмеженнями ..... 58
2.3. Організація застосунків. Спискові представлення ..... 71
2.4. Кнопки та панелі інструментів. Провайдер передачі інформації ..... 83
РОЗДІЛ 3. РОБОТА ІЗ БАЗОЮ ДАНИХ SQLITE ..... 92
3.1. Створення БД та отримання даних ..... 92
3.2. Версії БД та модифікація даних ..... 101
РОЗДІЛ 4. КЛІЄНТ-СЕРВЕРНІ ЗАСТОСУНКИ ..... 108
4.1. Виконання GET http-запитів ..... 108
4.2. Робота із власним API-сервісом ..... 116
4.3. Post-запити. Організація зміни даних ..... 124
ЗАВДАННЯ ДЛЯ ІНДИВІДУАЛЬНОГО ВИКОНАННЯ ..... 134
РЕКОМЕНДОВАНА ЛІТЕРАТУРА ..... 136

# *ВСТУП*

Мобільний застосунок (Mobile app) – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Розвиток ринку мобільних застосунків починається з появою у 1998 році протоколу WAP (Wireless Application Protocol). Саме він об'єднав інтернет і мобільний зв'язок. Тепер можна було вбудувати в телефон браузер, встановити з'єднання із серверами і отримати дані на пристрій. Одним з перших телефоном з WAP-браузером був Nokia 7100, який випустили в 1999 році. Тоді ж почали з'являтися компанії, які розробляють продукти спеціально для мобільних пристроїв.

WAP надав людям не тільки ігри, але і можливість читати новини з мобільного, користуватися електронною поштою, завантажувати карти та бронювати квитки. Для цього створювалися WAP-сайти зі спеціальною розміткою для мобільних екранів. Це були прості сторінки з тексту і посилань, майже без картинок.

У 2001 Symbian стала відкритою операційною системою, і в цей же час з'явився Nokia 7650, на якому можна було встановлювати застосунки від сторонніх розробників. Це повинно було стати проривом на ринку, але не спрацювало сповна через складнощі розробки та обмежені можливості смартфонів.

У цей же час розвивався ринок Java-застосунків. Розробка програми на Java займала менше часу і підходила для Windows Mobile, Android, bada, Palm OS і BlackBerry OS. У Symbian також підтримувалося підмножина Java – J2ME, але функціональність таких програм була сильно обмежена, тому розробкою на Java під Symbian практично ніхто не займався. У Nokia не прагнули допомагати розробникам розвивати ринок. У розробників не було нормального середовища для створення проєктів. Nokia випустили інструмент для розробки, але більша частина його можливостей була платною. Ліцензія коштувала від 300 до 8000 євро, це сильно впливало на кінцеву вартість програми. У результаті ОС не змогла конкурувати з iOS і Android.

У 2007 році Стів Джобс представив світу перший iPhone. Архітектура iOS була схожа на MacOS, але система була повністю закритою. Джобс не хотів, щоб сторонні розробники могли розробляти програми для iOS, і не збирався відкривати SDK. Замість цього він хотів, щоб розробники створювали вебзастосунки, і дав можливість створювати браузерні закладки на домашньому екрані.

Але «допитливі користувачі» зламали файлову систему і почали писати інсталятори для нативних застосунків. Пізніше рада директорів Apple все ж переконала Джобса легалізувати сторонні застосунки. У підсумку в березні 2008 року iPhone SDK став доступний усім охочим, а в липні презентували App Store. App Store став поштовхом до розвитку індустрії розробки застосунків, але проблемою був Objective-C. Мало хто хотів витрачати час на вивчення нового синтаксису, адже пристрої на iOS займали ще дуже маленьку частку ринку.

Android – операційна система для мобільних пристроїв, заснована на ядрі Linux. Вона є однією з найпопулярніших ОС для смартфонів та планшетів. Також використовується в таких пристроях, як смарт-годинник, електронні книги, музичні плеєри та нетбуки. Її власником, і фактичним розробником, є компанія Google. Історія операційної системи Android починається у 2005 році. Тоді Google купила компанію Android inc, фактичного творця цієї ОС, та у 2008 році була офіційно представлена перша версія операційної системи – Android 1.0. Ця та кожна наступна версія отримувала своє кодове ім'я. Перша версія вийшла з назвою «Apple Pie» (Яблучний пиріг).

До переваг операційної системи Android відносяться:

– легка синхронізація пристроїв на Android один з одним або з іншими пристроями;

– відкритість (можливість встановлювати сторонні програми без використання магазину застосунків, що неможливо на інших платформах);

– наявність різних магазинів застосунків (крім Google Play);

– великий вибір смартфонів на Android (вона не є закритою, тому будь-який виробник може встановлювати її на свої смартфони). Тому на цій ОС можна знайти як бюджетні моделі, так і моделі преміум-класу.

До недоліків відносяться: швидка витрата заряду батареї (найпоширеніша причина критики операційної системи Android), відкритість (завдяки відкритості на Android існує порівняно велика кількість вірусів), розтрата трафіку (під час підключення до мережі система самостійно витрачає інтернет-трафік на свої потреби).

Також безумовно слід враховувати, що на сьогодні під управлінням ОС Android працюють більше 80% всіх смартфонів у світі.

Далі згадаємо популярні мови програмування для розробки мобільних застосунків для Android та iOS:

Java – строго типізована об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (в подальшому придбаної компанією Oracle). Платформа: Android, основна IDE: Android Studio.

Swift – мова, розроблена компанією Apple і призначена для розробки застосунків під iOS і OS X. Swift запозичив досить багато з C++ і Objective-C. Платформа: iOS, основна IDE: Xcode.

JavaScript – прототипно-орієнтована сценарна мова програмування. Найбільш широке застосування знайшла в браузерах як мова сценаріїв для додання інтерактивності вебсторінок, а також у кросплатформних фреймворків (React Native, Ionic, Sencha і та ін.). Платформа: iOS, Android і практично будь-яка інша.

C# – об'єктно-орієнтована мова програмування розробки застосунків для платформи Microsoft .NET Framework. В області розробки мобільних застосунків використовується у фреймворку Xamarin. Платформа: iOS, Android, Windows 10, основна IDE: Visual Studio.

Objective-C – об'єктно-орієнтована мова програмування корпорації Apple, побудована на основі мови C і Smalltalk. Зараз її замінює новий і більш простий Swift. Проте, деякий час розробники на Objective-C будуть дуже затребувані на ринку. Платформа: iOS, macOS, watchOS і tvOS.

Одне із основних питань, що виникає на початку розробки мобільного застосунку, – створювати мобільний (адаптивний) вебсайт, нативний або гібридний застосунок. Від прийнятого рішення буде залежати подальший процес розробки. Для розуміння ключових відмінностей наведемо характеристику нативного та гібридного способів розробки.

Нативні (рідні) застосунки написані мовою програмування, специфічною для платформи, для якої вони розробляються. Зазвичай це Objective-C або Swift для iOS та Java або Kotlin для Android. Нативні застосунки зазвичай мають кращу продуктивність при рендерінгу і анімації, ніж гібридні програми. Але якщо вам необхідна реалізація як для Android, так і для iOS – необхідно буде створити два застосунки. Гібридний застосунок – це мобільний застосунок, який містить вебпредставлення (по суті, ізольований екземпляр браузера) для запуску вебзастосунку із використанням вбудованої оболонки, яка може взаємодіяти із платформою пристрою та вебпредставленням. Це означає, що вебзастосунки можуть працювати на мобільному пристрої, маючи доступ до, наприклад, камери або функцій GPS. Гібридні застосунки можливі завдяки створеним інструментам, які полегшують зв'язок між вебпредставленням і платформою. Ці інструменти не є частиною офіційних платформ iOS або Android, та є сторонніми інструментами, такими як Apache Cordova. Коли гібридний застосунок буде створено, він буде скомпільований, перетворивши ваш вебзастосунок у нативний застосунок.

Цей матеріал зупиняє свою увагу на створенні саме нативних застосунків під операційну систему Android мовою Java.

# *РОЗДІЛ 1.*
# *СТВОРЕННЯ ПРОСТОГО ЗАСТОСУНКУ*

## 1.1. Загальні відомості та «Hello world»

Платформа Android складається з багатьох компонентів. У неї входять базові застосунки (наприклад, Контакти), набір програмних інтерфейсів (API) для управління зовнішнім виглядом і поведінкою застосунків, а також багатьох допоміжних файлів і бібліотек (рис. 1.1).

Під час побудови застосунків доступні ті ж API, які використовуються базовими застосунками. За допомогою цих API ви керуєте зовнішнім виглядом і поведінкою своїх застосунків. Під інфраструктурою застосунків розташовується рівень бібліотек C і C++. Для роботи з ними використовуються API. Кожен Android-застосунок виконується в окремому процесі. У самій основі системи лежить ядро Linux. В Android воно забезпечує роботу драйверів, а також таких базових сервісів, як безпека і управління пам’яттю.


Рисунок 1.1 – Компоненти платформи Andoid

Пристрої на базі Android не запускають файли .class і .jar. Замість цього для підвищення швидкості та ефективності використання акумуляторів Android-пристрої використовують власні оптимізовані формати компільованого коду. Це означає, що ви не зможете скористатися звичайним середовищем розробки мовою Java – вам також знадобляться спеціальні інструменти для перетворення відкомпільованого коду в формат android, установки його на Android-пристроях і налагодження програми, коли вона запрацює.

Усі ці інструменти входять до складу Android SDK. Пакет Android Software Development Kit (SDK) містить бібліотеки та інструменти, необхідні для розробки Android-застосунків.

Зокрема, до складу SDK входять:

– SDK Platform – окрема платформа для кожної версії Android;

– SDK Tools – інструменти відлагодження та тестування, а також інші корисні службові програми. Включає набір платформно-незалежних інструментів.

IntelliJ IDEA – одна з найпопулярніших інтегрованих середовищ розробки (IDE) для програмування на Java. Android Studio – версія IDEA, яка включає версію Android SDK і додаткові інструменти графічних інтерфейсів, що спрощують розробку застосунків. Крім редактора і доступу до інструментів та бібліотек Android SDK, Android Studio надає шаблони, що спрощують створення нових класів і застосунків, а також засоби для виконання таких операцій, як упаковка застосунків та їх запуск.

Що стосується запуску застосунку, є два варіанти. Варіант перший – запустити застосунок на фізичному пристрої. Варіант другий – скористатися емулятором Android, вбудованим в Android SDK. Емулятор дозволяє створити один або кілька віртуальних пристроїв Android (Android Virtual Device, AVD) і запустити застосунок в емуляторі так, мов він виконується на фізичному пристрої. За останні роки Google сильно попрацював над своїм емулятором і перетворив його в один із кращих інструментів для розробки: швидкий, гнучкий і корисний під час тестування й налагодженні програм. Емулятор Android може імітувати роботу смартфона, планшета, годинника Wear OS та пристроїв Android TV.

Для того, щоб створити новий AVD, потрібно запустити AVD Manager, вибравши в Android Studio в меню Tools – AVD Manager. Відкриється вікно менеджера, в якому буде відображатися список створених емуляторів (рис. 1.2).

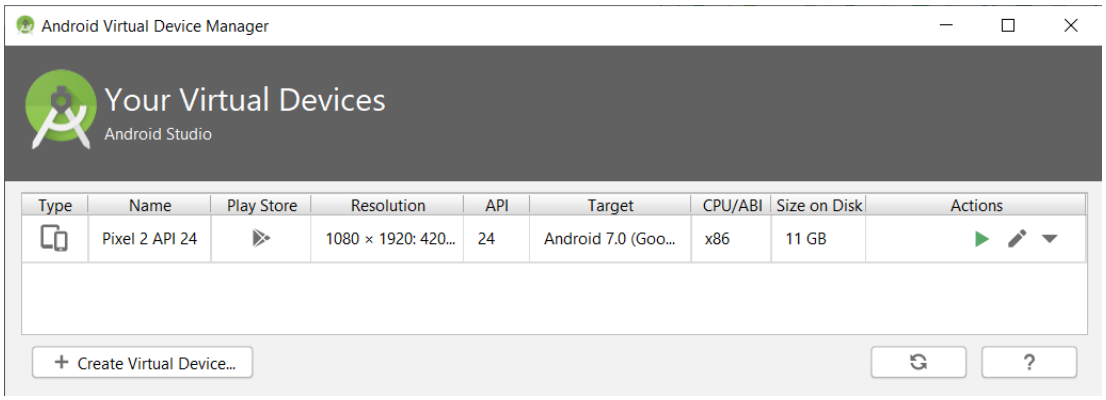



Рисунок 1.2 – Список створених емуляторів

Щоб створити новий емулятор, потрібно натиснути на Create Virtual Device в Менеджері AVD. Відкриється вікно, в якому буде запропоновано вибрати тип пристрою і профіль (рис. 1.3).

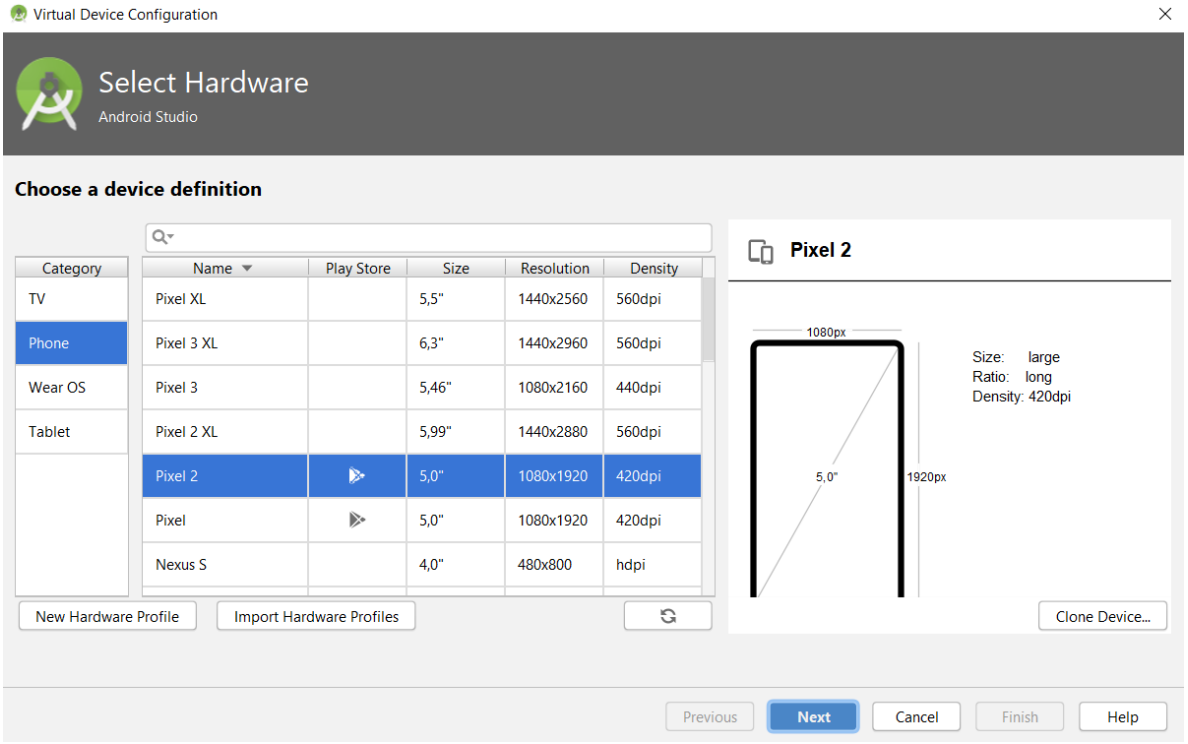



Рисунок 1.3 – Вибір типу пристрою

Після того, як буде обраний профіль, потрібно натиснути на Next для переходу далі. Тут потрібно вибрати, який образ системи використовувати (рис. 1.4).

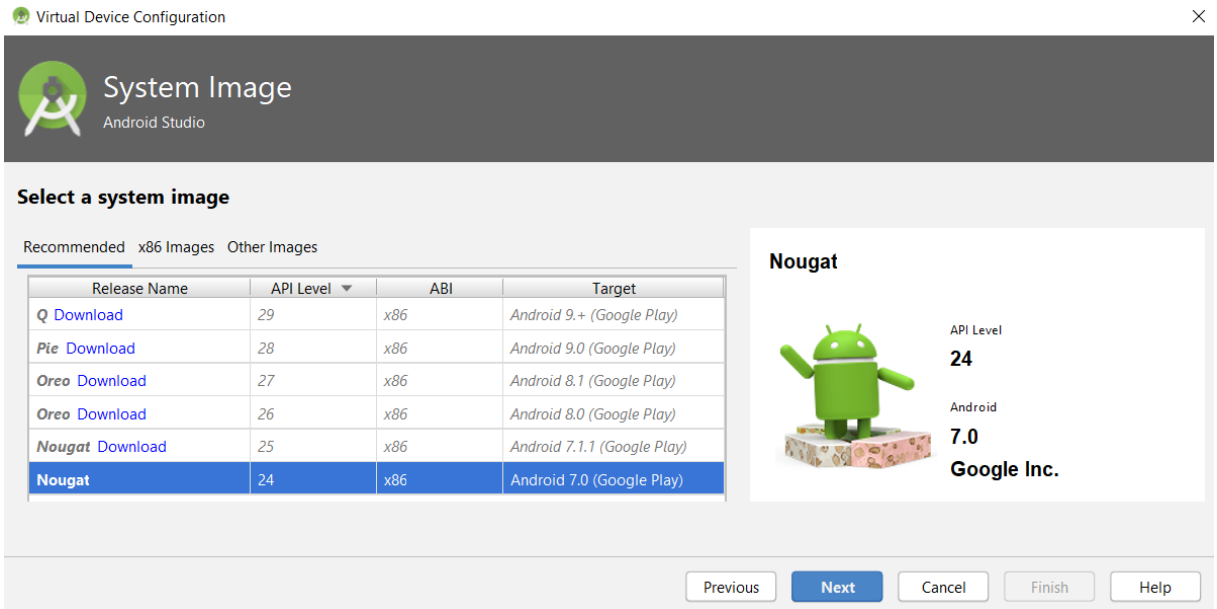



Рисунок 1.4 – Вибір образу системи

На вкладці Recommended перераховані рекомендовані образи системи. Інші вкладки містять більш повний список доступних образів. Справа наводиться інформація про обраний образ (рівень API, версія). Зауважимо, що образи x86 працюють на емуляторі найшвидше. Рівень API важливий, оскільки якщо він буде менший, ніж той, що зазначений у маніфесті застосунку, застосунок не зможе бути встановлено на цей емулятор.

Якщо образ раніше не був завантажений, поруч з назвою з'явиться кнопка Download, натискання на яку почне процес завантаження. Для завантаження образу потрібно доступ до Інтернету.

Щоб перейти на наступний етап, потрібно натиснути Next. У новому вікні буде запропоновано змінити додаткові властивості AVD (рис. 1.5).

Після того, як AVD буде налаштований, залишиться тільки натиснути Finish. Створений AVD можна буде побачити у вікні Менеджера AVD. Протестувати додаток на емуляторі можна, натиснувши на кнопку Run в Android Studio (рис. 1.6).

Відкривається вікно Select Deployment Target, в якому буде запропоновано обрати, на якому пристрої потрібно запускати застосунок (рис. 1.7).

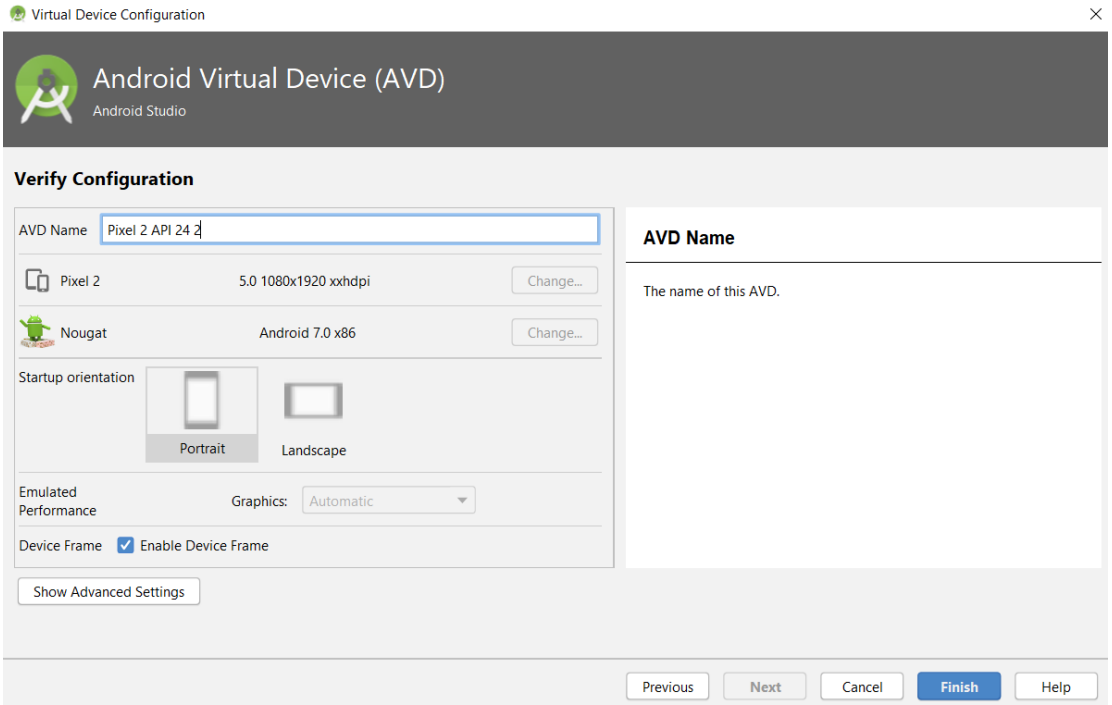



Рисунок 1.5 – Додаткові властивості AVD

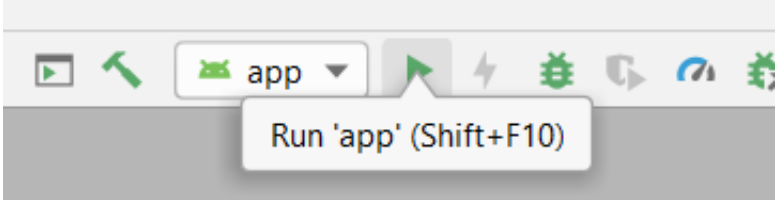



Рисунок 1.6 – Запуск емулятора

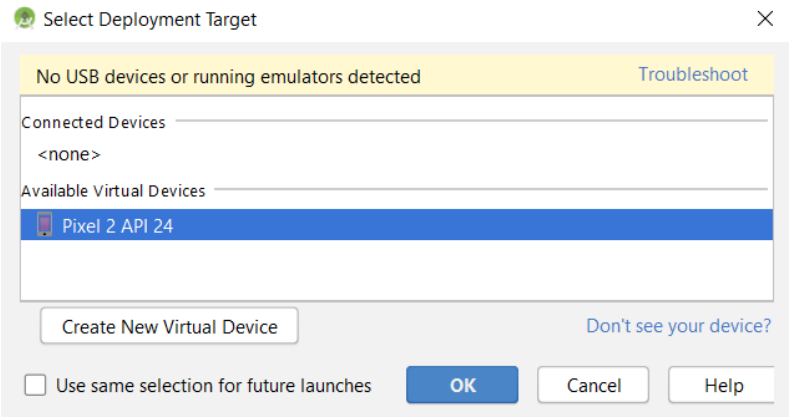



Рисунок 1.7 – Вибір емулятора для запуску застосунку

Після натискання OK почнеться запуск обраного емулятора, якщо він не запущений, або установка APK на емулятор.

Далі розглянемо створення тестового проєкту «Привіт, світ». Для цього запускаємо Android Studio та створюємо новий проєкт File –> New –> New Project (рис. 1.8).

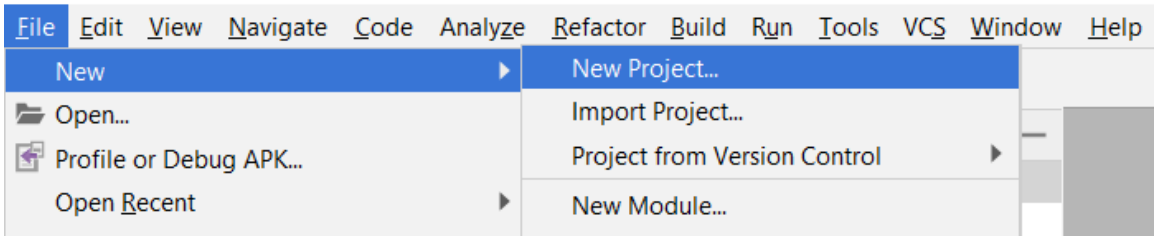



Рисунок 1.8 – Створення нового проєкту в Android Studio

Обираємо вкладку «Phone and tablet», тип активності «Empty activity» та натискаємо «next» (рис. 1.9).

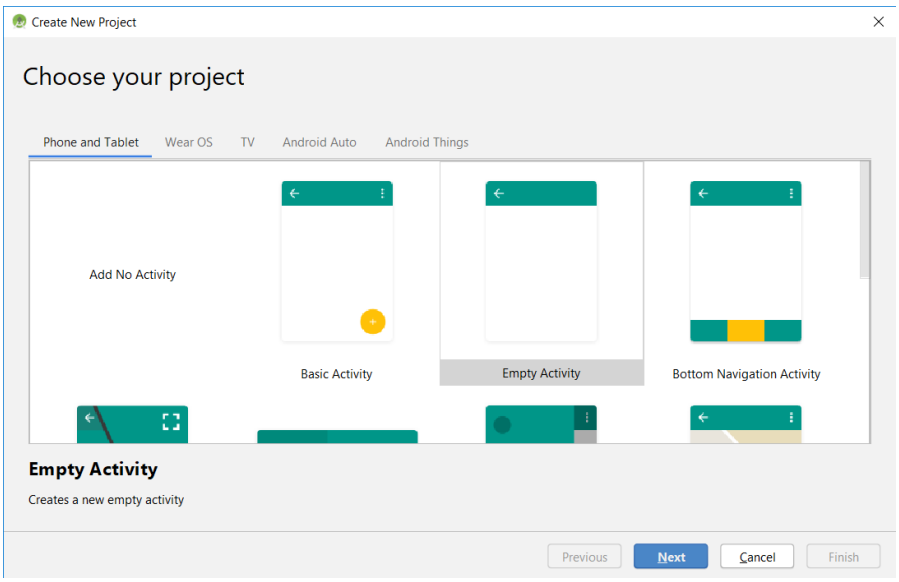



Рисунок 1.9 – Обрання типу активності

На наступному кроці необхідно задати ім'я проєкту, пакета, розташування файлів, мову розробки та мінімальну версію ОС, що буде підтримувати майбутній застосунок (рис. 1.10). Задамо ім'я «MyHello», пакет «com.chnulabs.myhello», мову розробки «Java» та minimum API level 19:Android 4.4. Ім'я пакета грає дуже важливу роль в Android, тому що воно використовується Android-пристроями для однозначної ідентифікації застосунку. Рівні API збільшуються з виходом кожної чергової версії Android. Якщо тільки ви не хочете, щоб застосунок працював лише на найновіших пристроях, варто вибрати один із нижчих рівнів API.

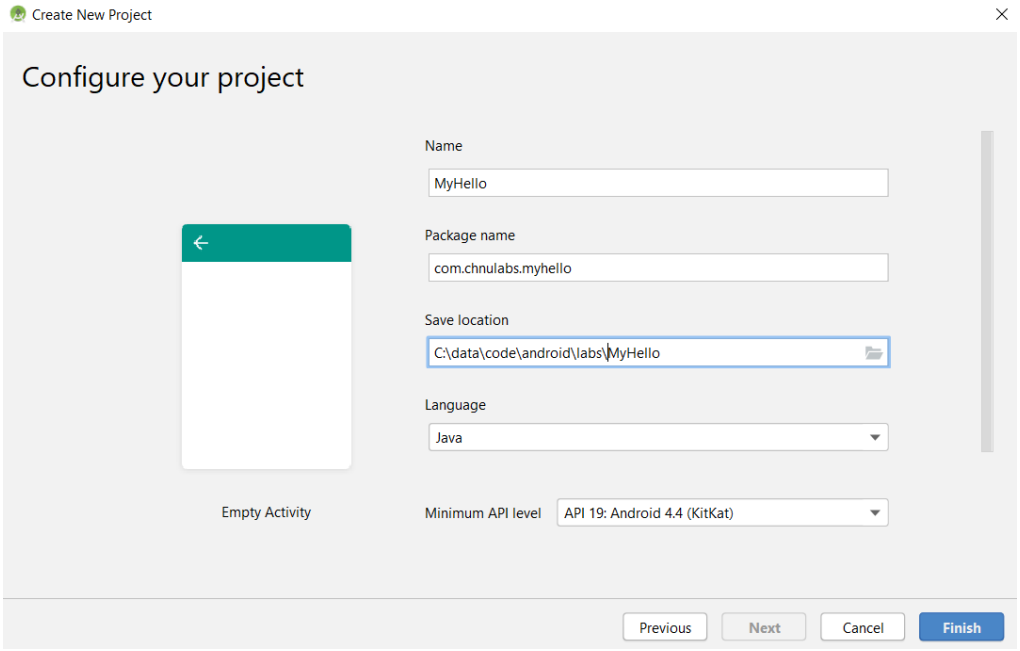



Рисунок 1.10 – Конфігурування проєкту

Кожен Android-застосунок складається з екранів, а кожен екран складається з активності і макета. Активність – одна чітко визначена операція, яку може виконати користувач. Наприклад, у застосунку можуть бути присутніми активності для складання повідомлення електронної пошти, знаходження контакту або створення знімка. Активності зазвичай асоціюються з одним екраном і програмуються на Java (або Kotlin). Макет описує зовнішній вигляд екрана. Макети створюються у вигляді файлів в розмітці XML і повідомляють Android, де розташовуються ті чи інші елементи екрана.

Розглянемо послідовність дій взаємодії пристрою Android, активності та макета:

– пристрій запускає застосунок і створює об'єкт активності;
– об'єкт активності визначає макет;

– активність дає команду на вивід макета на екран;

– користувач взаємодіє із макетом, що відображається на екрані пристрою;

– активність реагує на події макета та виконує відповідний код застосунку;

– у разі необхідності активність у відповідь на дії користувача оновлює дані макета;

– користувач бачить зміни на екрані пристрою.

Після створення нового проєкту (рис. 1.8–1.10) у ньому вже є одна активність із макетом. На рис. 1.11 можна побачити файли java\com.chnulabs.myhello\MainActivity (активність) та res\layout\activity_main.xml (макет).

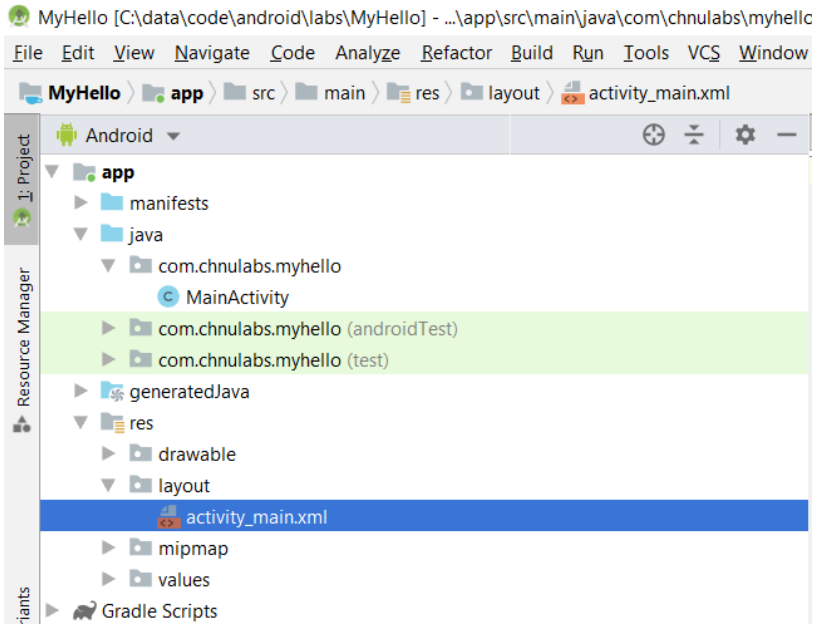



Рисунок 1.11 – Файли активності та макету

Для перегляду і зміни файлів використовуються різні редактори Android Studio. Зробіть подвійний клік на файлі, з яким ви хочете працювати; його вміст з'являється в середині вікна Android Studio (рис. 1.12).

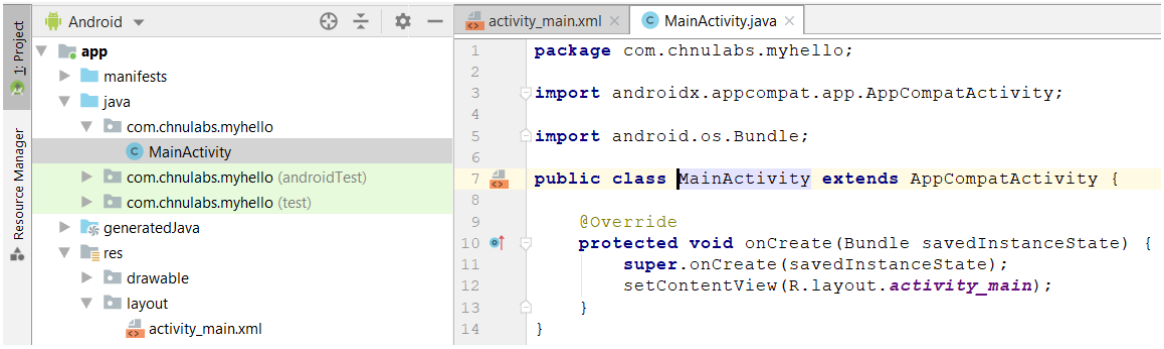



Рисунок 1.12 – Файл MainActivity

Більшість файлів відображається в редакторі коду. По суті це звичайний текстовий редактор, але з підтримкою таких додаткових можливостей, як кольорове виділення синтаксису і перевірка коду.

Під час редагування макета з'являється додаткова можливість: замість редагування розмітки XML можна використовувати візуальний редактор (рис. 1.13). Візуальний редактор дозволяє перетягнути компоненти графічного інтерфейсу на макет і розмістити їх так, як ви вважаєте за потрібне. Редактор коду і візуальний редактор забезпечують різні представлення одного файлу, і ви можете перемикатися між ними (рис. 1.14).

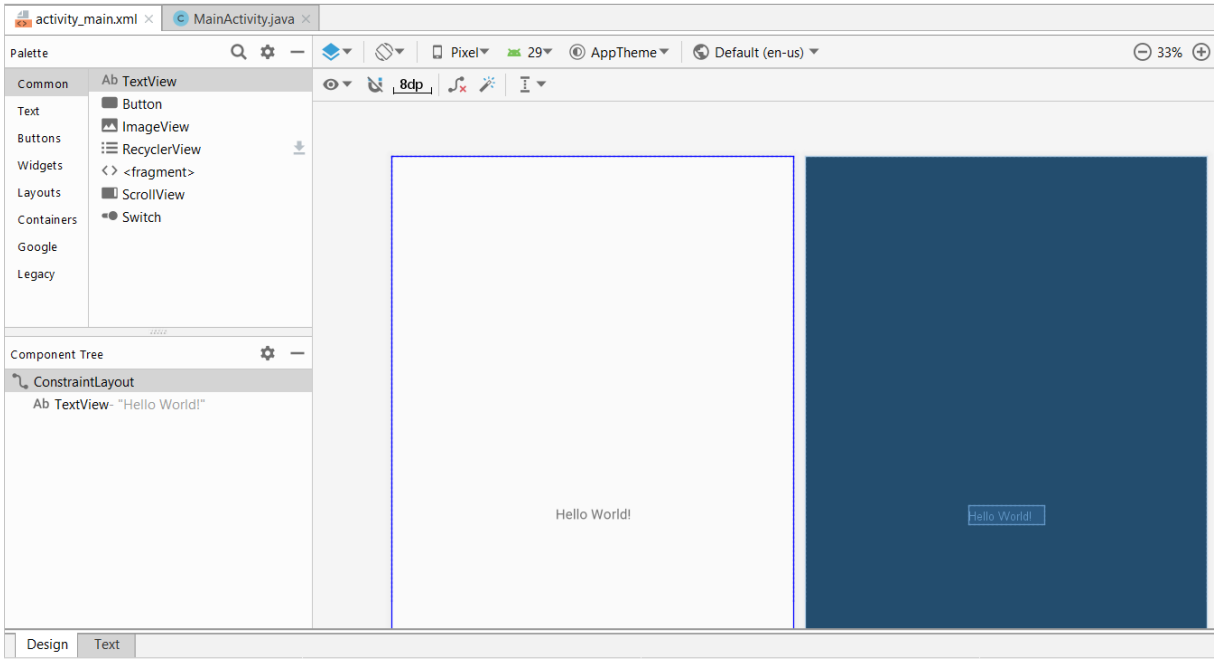



Рисунок 1.13 – Візуальний редактор xml-файлу

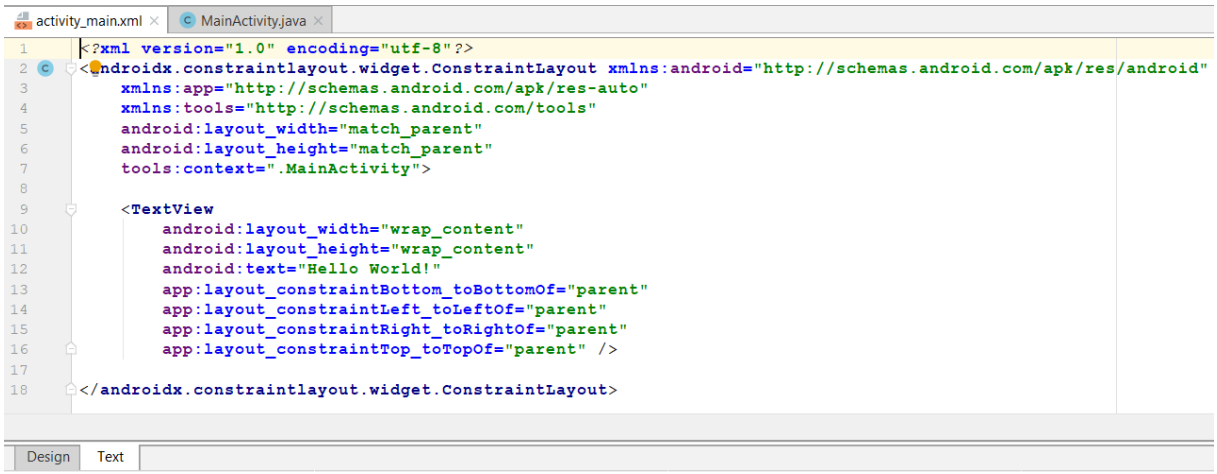



Рисунок 1.14 – Редактор коду xml-файлу

Змінимо розмір шрифту для елемента TextView нашого макета. Для цього на вкладці design виділимо його, у вікні властивостей знайдемо textSize та встановимо 36sp (рис. 1.15).

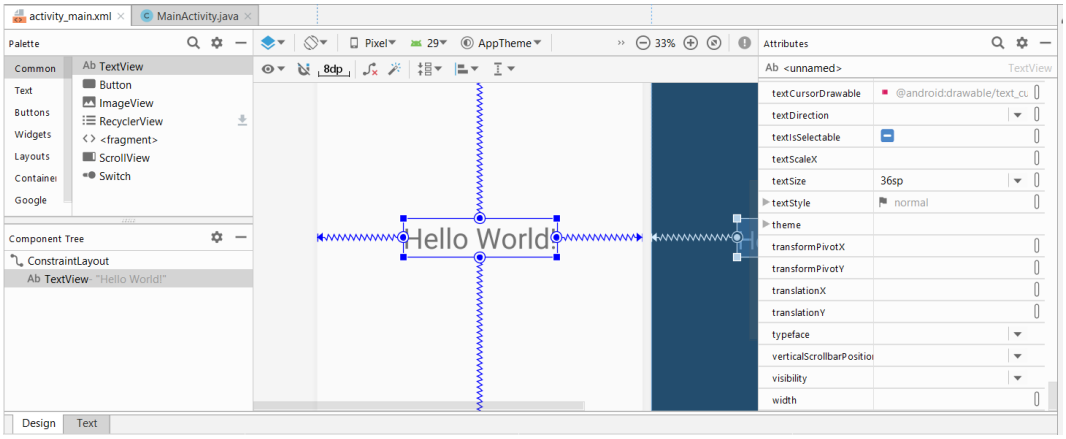



Рисунок 1.15 – Зміна розміру шрифту для елементу TextView

Після цього перейдемо у вкладку text та звернімо увагу на рядок «android:textSize="36sp"», що додався до описання елемента TextView (рис. 1.16)

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="36sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

Рисунок 1.16 – Зміни у коді activity_main.xml

Виконаємо наш застосунок на віртуальному пристрої. Для цього виконуємо команду run app із меню run (рис. 1.17) або натиснувши run app на панелі інструментів.

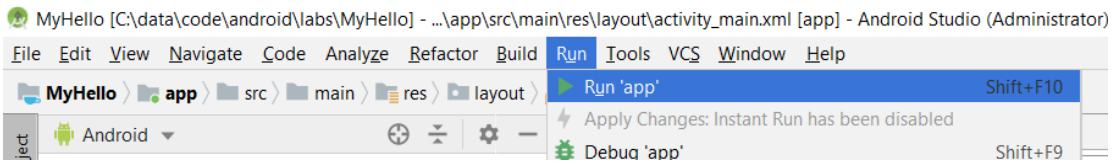



Рисунок 1.17 – Запуск застосунку на виконання

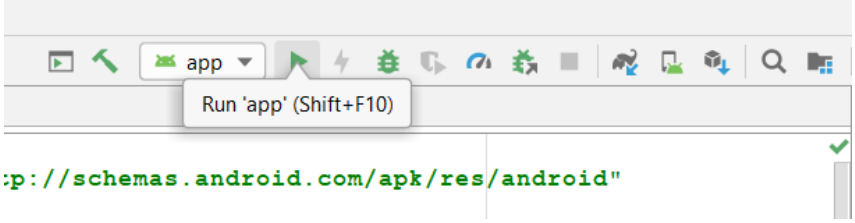



Рисунок 1.18 – Запуск застосунку на виконання через панель інструментів

Обираємо віртуальний пристрій та натискаємо ok (рис. 1.19).

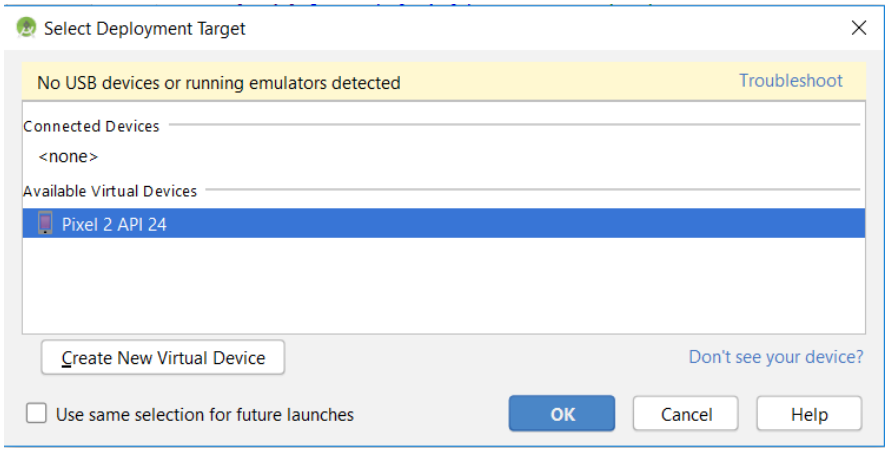



Рисунок 1.19 – Вибір віртуального пристрою

Після запуску емулятора та встановлення на нього нашого застосунку MyHello бачимо на ньому такий екран (рис. 1.20).

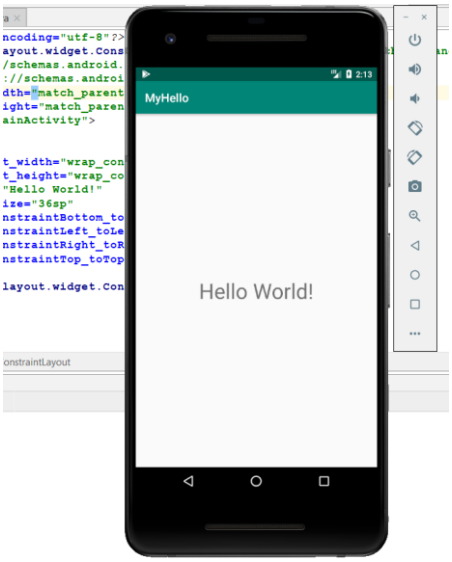



Рисунок 1.20 – Вигляд застосунку MyHello
на віртуальному пристрої

Тепер розберемо покроково, що відбувалося після виконання команди run app:

– файли із вихідним кодом Java компілюються у байт-код;

– створюється android-застосунок у вигляді apk-файлу, в який включаються всі необхідні бібліотеки та ресурси;

– запускається емулятор (якщо він ще не запущений);

– після запуску емулятора apk-файл передається на пристрій;

– віртуальний пристрій запускає активність, що пов’язана із застосунком;

– активність визначає макет та відображає її на екран.

## 1.2. Взаємозв'язок макета та активності

Запускаємо Android Studio та створюємо новий проєкт empty activity «Students», ім'я пакета com.chnulabs.students, API – 19:4.4, мова – Java (рис. 1.21).

Відкриємо макет activity_main.xml у текстовому редакторі, змінимо «Hello world!» на «Hello students» та збільшимо розмір шрифту (рис. 1.22).

Запустимо застосунок (run app) та переконаємось, що текст у макеті змінився (рис. 1.23).

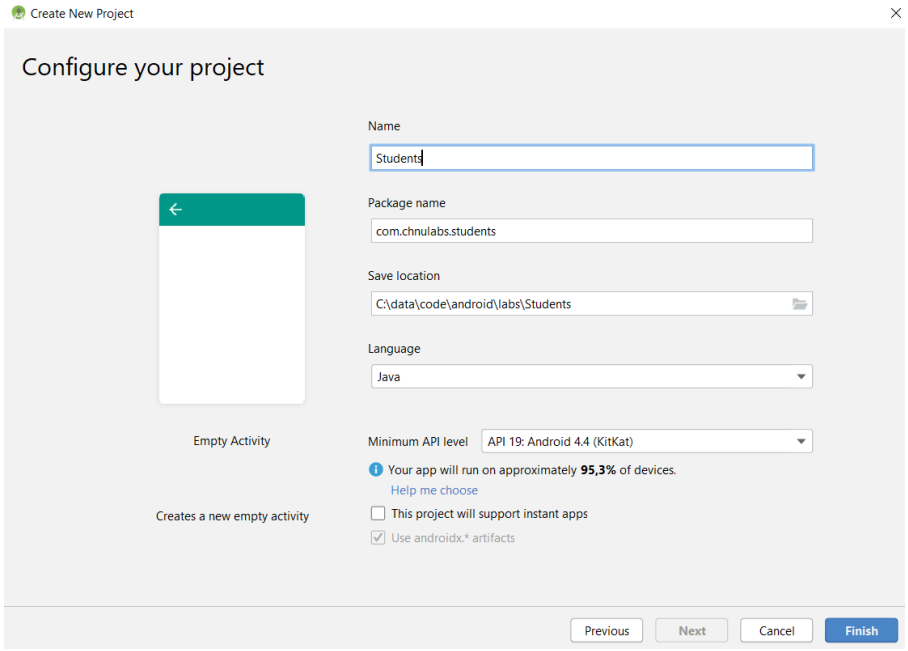



Рисунок 1.21 – Створення нового проєкту Students

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Students!"
        android:textSize="36sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 1.22 – Зміна тексту в TextView

Однак розміщення тексту безпосередньо у макеті не є гарним підходом. Це обумовлено декількома причинами, серед яких – необхідність зміни ідентичного тексту у різних макетах при можливій зміні користувацького інтерфейсу, або підтримка застосунком декількох мов.

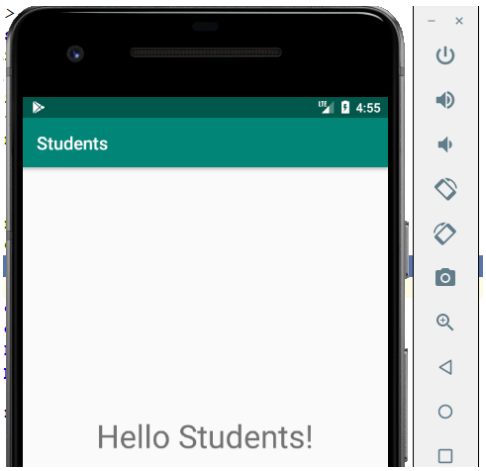



Рисунок 1.23 – Оновлений текст макета активності

Більш правильно додати змінну до файлу strings.xml та використати її у макеті. У загальному випадку strings.xml включає всі строкові ресурси, необхідні макету, такі як заголовки, написи на кнопках, текстові повідомлення та ін.

Перейдемо до редагування файлу res/values/strings.xml та додамо туди рядок hello_text (рис. 1.24). Також змінимо назву застосунку на «Список студентів».

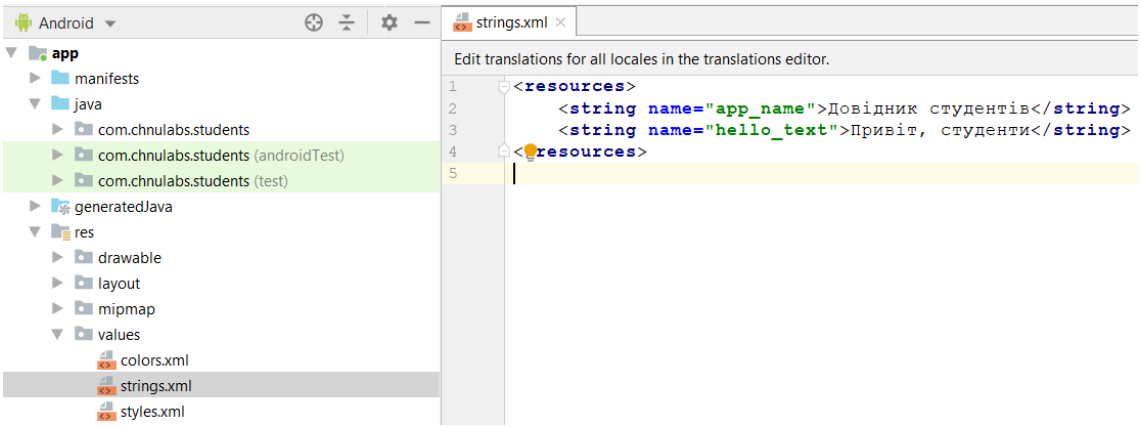



Рисунок 1.24 – Зміни у файлі strings.xml

Далі необхідно внести зміни у макет активності activity_main.xml для того, щоб він використовував дані hello_text, збережені у strings.xml (рис. 1.25).

Запустимо застосунок та переконаймося, що текст макета змінився відповідно до змін у файлі strings.xml (рис. 1.26).

```
10      <TextView
11          android:layout_width="wrap_content"
12          android:layout_height="wrap_content"
13          android:text="@string/hello_text"
14          android:textSize="36sp"
15          app:layout_constraintBottom_toBottomOf="parent"
16          app:layout_constraintLeft_toLeftOf="parent"
17          app:layout_constraintRight_toRightOf="parent"
18          app:layout_constraintTop_toTopOf="parent" />
```

Рисунок 1.25 – Зміни у макеті activity_main.xml

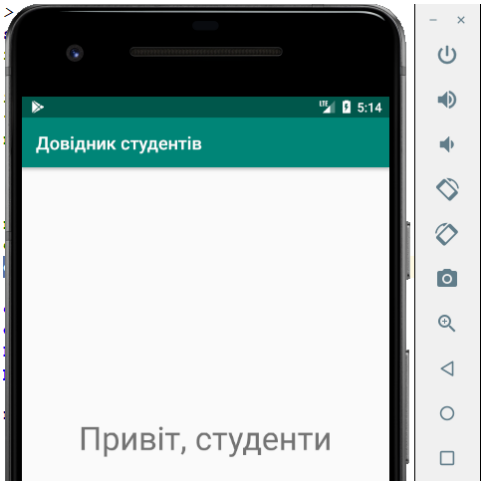



Рисунок 1.26 – Перевірка зміни тексту у макеті активності

Далі змінимо наш макет та активність для відображення списку студентів за студентськими групами. Почнемо із реалізація макета. У макеті активності activity_main.xml у текстовому редакторі змінимо «ConstraintLayout», запропонований Android Studio за замовчуванням, на більш простий «LinearLayout». Він використовується для виведення компонентів графічного інтерфейсу поруч один з одним, по вертикалі або по горизонталі. Якщо компоненти шикуються по вертикалі, вони утворюють стовпець, а якщо по горизонталі – рядок. Також приберемо із TextView частину атрибутів, характерних для використання у контексті «ConstraintLayout» (рис. 1.27).

Використаємо візуальний редактор та додамо до макета випадний список Spinner. Для цього перейдемо на вкладку «design», у вікні «Palette» оберемо розділ «containers» та елемент управління Spinner. Перетягнемо його мишею до екрана макета, розмістивши перед текстом TextView (рис. 1.28).


Рисунок 1.27 – Встановлення LinearLayout для макету активності

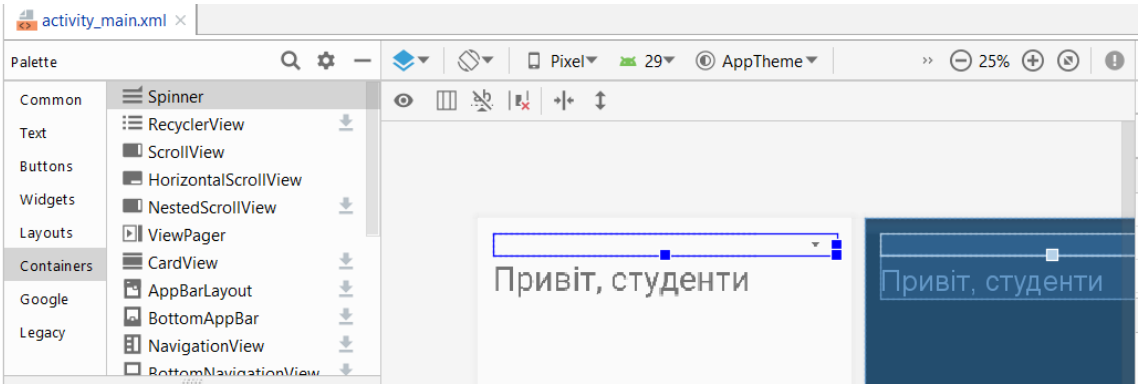



Рисунок 1.28 – Додавання елементу Spinner

Таким самим чином додаємо кнопку Button із розділу «buttons». Розмістимо її між елементом Spinner та TextView (рис. 1.29).

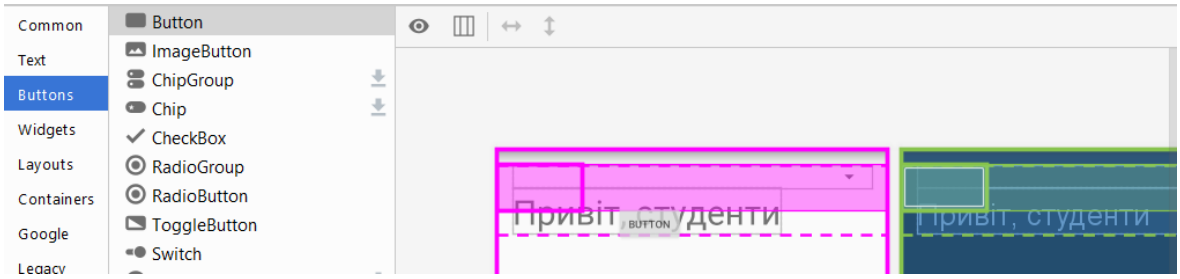



Рисунок 1.29 – Додавання кнопки

Перейдемо до вкладки «Text» редактора макета та виконаємо деякі налаштування властивостей елементів макета. Так, для Spinner змінимо android:layout_width="wrap_content" для того, щоб елемент займав стільки місця, скільки необхідно для вміщення контенту, та додамо відступи зверху та знизу, а також вирівнювання по центру (рис. 1.30).

```
11      <Spinner
12          android:id="@+id/spinner"
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:layout_gravity="center"
16          android:layout_marginTop="40dp"
17          android:layout_marginBottom="16dp"/>
18
```

Рисунок 1.30 – Зміни властивостей елементу управління Spinner

Для кнопки додаємо відступ знизу та візьмемо текст напису із strings.xml (рис. 1.31).

```
19      <Button
20          android:id="@+id/button"
21          android:layout_width="match_parent"
22          android:layout_height="wrap_content"
23          android:text="@string/btn_text"
24          android:layout_marginBottom="16dp"/>
```

Рисунок 1.31 – Зміни властивостей кнопки

Зараз значення android:text підсвічено червоним кольором, оскільки ми ще не додали відповідне значення до strings.xml. Виправимо це, також додавши значення тексту за замовчуванням для TextView. Текст «Привіт, студенти» надалі не знадобиться, тому приберемо його (рис. 1.32).

strings.xml × activity_main.xml ×

Edit translations for all locales in the translations editor.

```
1   <resources>
2       <string name="app_name">Довідник студентів</string>
3       <string name="default_text">Список студентів</string>
4       <string name="btn_text">Показати список</string>
5   </resources>
```

Рисунок 1.32 – Зміни у res/values/strings.xml

Для TextView аналогічно вкажемо android:layout_width="wrap_content". Також змінимо значення тексту із hello_text на default_text та трохи зменшимо розмір шрифту (рис. 1.33).

```
26      <TextView
27          android:layout_width="wrap_content"
28          android:layout_height="wrap_content"
29          android:text="@string/default_text"
30          android:textSize="18sp" />
31
```

Рисунок 1.33 – Зміни в атрибутах TextView

Той факт, що кнопки, написи та інші елементи мають так багато спільних властивостей, цілком логічний – всі ці компоненти успадковують від одного класу Android View.

Виконаємо застосунок та переглянемо результат в емуляторі (рис. 1.34).

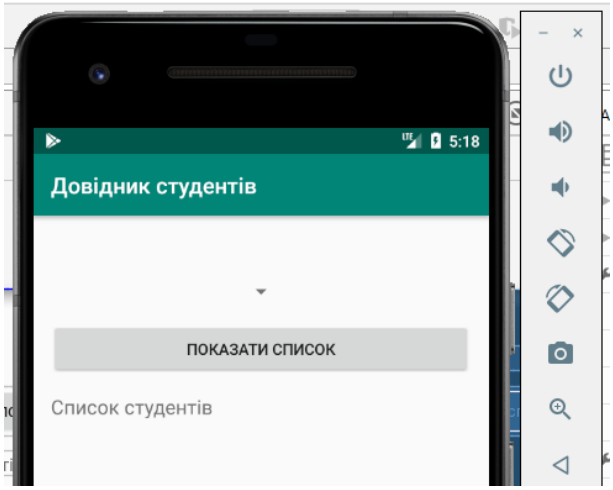



Рисунок 1.34 – Зовнішній вигляд макету активності

Далі необхідно наповнити елемент Spinner значеннями студентських груп. Дана задача буде реалізована через використання можливості збереження масивів рядків у strings.xml. Перейдемо до res/values/strings. xml та додамо список студентських груп (рис. 1.35).

Використаємо створений масив, підключивши його до елемента Spinner у макеті активності activity_main.xml. Для цього у Spinner додаємо атрибут android:entries (рис.1. 36).

Перевіримо результат виконаних змін, запустивши застосунок (рис. 1.37).

strings.xml

Edit translations for all locales in the translations editor.

```
<resources>
    <string name="app_name">Довідник студентів</string>
    <string name="default_text">Список студентів</string>
    <string name="btn_text">Показати список</string>
    <string-array name="students_groups">
        <item>301</item>
        <item>302</item>
        <item>308</item>
        <item>309</item>
    </string-array>
</resources>
```

Рисунок 1.35 – Збереження масиву рядків в strings.xml

```
11      <Spinner
12          android:id="@+id/spinner"
13          android:layout_width="wrap_content"
14          android:layout_height="wrap_content"
15          android:layout_gravity="center"
16          android:layout_marginTop="40dp"
17          android:layout_marginBottom="16dp"
18          android:entries="@array/students_groups"/>
```

Рисунок 1.36 – Зв’язок Spinner із string-array

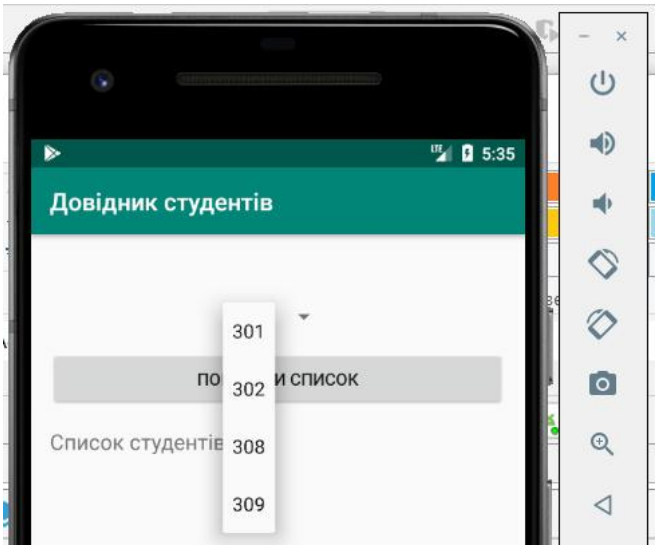



Рисунок 1.37 – Вибір групи в елементі Spinner

Реалізуємо обробку події натискання на кнопку. Для цього перейдемо до файлу активності MainActivity.java та створимо метод onBtnClick, що буде викликатись по натисканню на кнопку. Цей метод має вхідний параметр типу View (рис. 1.38).

```
5   import android.os.Bundle;
6   import android.view.View;
7
8   public class MainActivity extends AppCompatActivity {
9
10      @Override
11      protected void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.activity_main);
14      }
15
16      public void onBtnClick(View view) {
17
18      }
19  }
```

Рисунок 1.38 – Додавання методу onBtnClick до активності

Пов’яжемо цей метод із подією натискання на кнопку. Для цього у макеті activity_main.xml у компоненті Button пропишемо атрибут «android:onClick="onBtnClick"» (рис. 1.39).

```
20      <Button
21          android:id="@+id/button"
22          android:layout_width="match_parent"
23          android:layout_height="wrap_content"
24          android:text="Показати список"
25          android:layout_marginBottom="16dp"
26          android:onClick="onBtnClick"/>
```

Рисунок 1.39 – Прив'язка методу активності до події макета

Для перевірки роботи необхідно виконати у методі активності onBtnClick якусь дію. Використаємо клас-віджет Toast (рис. 1.40). Це просте спливаюче повідомлення, яке з'являється на екрані. Повідомлення виконують чисто інформаційні функції, користувач не може з ними взаємодіяти. Поки повідомлення знаходиться на екрані, активність залишається видимою і доступною для взаємодії з користувачем. Повідомлення автоматично закривається після закінчення часу очікування. Щоб створити повідомлення, викличте метод Toast.makeText() і передайте йому три параметра: Context (зазвичай для поточної активності), CharSequence (виведене повідомлення) та int (тривалість). Після того, як об'єкт повідомлення буде створений, його можна вивести на екран викликом методу show().

```
6   import android.view.View;
7   import android.widget.Toast;
8
```

```
17      public void onBtnClick(View view) {
18          CharSequence text = "Ви натиснули на кнопку!";
19          int duration = Toast.LENGTH_LONG;
20          Toast toast = Toast.makeText( context: this, text, duration);
21          toast.show();
22      }
```

Рисунок 1.40 – Виведення повідомлення по натисканню на кнопку

Перевіримо отриманий результат, запустивши застосунок в емуляторі (рис. 1.41)

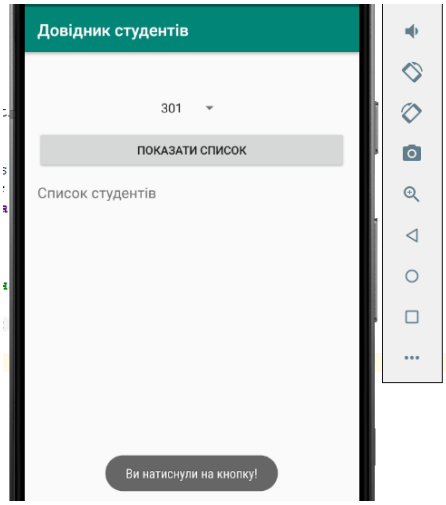



Рисунок 1.41 – Виведення повідомлення по натисканню на кнопку

Тепер залишилось по натисканню на кнопку Button реалізувати виведення в компонент TextView списку студентів обраної в Spinner групи. Почнемо зі створення класу Student, що визначатиме список студентів та набір базових методів по роботі зі студентом. Для цього у пакеті com.chnulabs.students створимо новий java-класс Student New–>Java class (рис. 1.42–1.43).

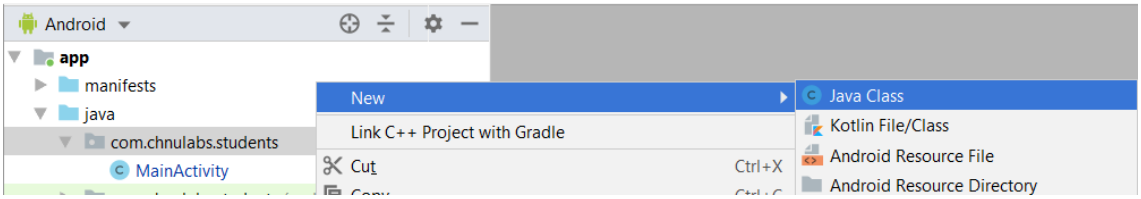



Рисунок 1.42 – Створення нового java-класу Student

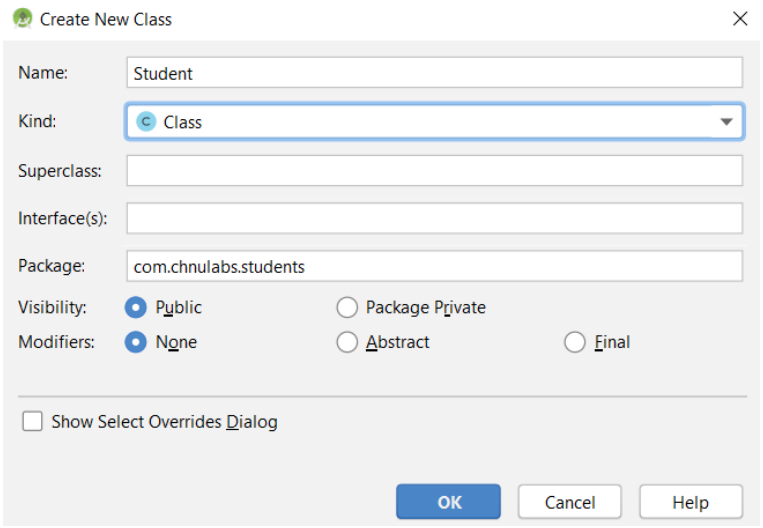



Рисунок 1.43 – Вказання назви класу

Створений клас Student міститиме 2 приватних поля: name – ім’я студента та groupNumber – номер групи; конструктор для створення екземпляра класу з 2-ма відповідними параметрами та методи-гетери для отримання значень цих полів. Крім того, у класі визначено статичну колекцію студентів та метод для отримання із неї студентів відповідної групи (рис. 1.44).

Student.java

```
1  package com.chnulabs.students;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5
6  public class Student {
7      private String name;
8      private String groupNumber;
9
10     public Student(String name, String groupNumber) {
11         this.name = name;
12         this.groupNumber = groupNumber;
13     }
```

```
14
15      public String getName() {
16          return name;
17      }
18
19      public String getGroupNumber() {
20          return groupNumber;
21      }
22
23      private final static ArrayList<Student> students = new ArrayList<Student>(
24          Arrays.asList(
25              new Student( name: "Іванов Роман",  groupNumber: "301"),
26              new Student( name: "Петров Федір",  groupNumber: "301"),
27              new Student( name: "Осадча Оксана",  groupNumber: "302"),
28              new Student( name: "Максимов Руслан",  groupNumber: "302"),
29              new Student( name: "Смірнов Василь",  groupNumber: "308"),
30              new Student( name: "Потапова Марія",  groupNumber: "309"),
31              new Student( name: "Гонський Іван",  groupNumber: "309"),
32              new Student( name: "Васильев Максим",  groupNumber: "309")
33          )
34      );
35
36      public static ArrayList<Student> getStudents(String groupNumber) {
37          ArrayList<Student> stList = new ArrayList<>();
38          for(Student s: students) {
39              if (s.getGroupNumber().equals(groupNumber)) {
40                  stList.add(s);
41              }
42          }
43          return stList;
44      }
45
46  }
```

Рисунок 1.44 – Клас Student

Перейдемо до макета та перевіримо наявність ідентифікаторів у компонент. Це необхідно для можливості звернення до них із коду активності. На цей момент компонент TextView не має ідентифікатора, виправимо це, додавши до нього атрибут android:id="@+id/text" (рис. 1.45).

```
28      <TextView
29          android:id="@+id/text"
30          android:layout_width="wrap_content"
31          android:layout_height="wrap_content"
32          android:text="Список студентів"
33          android:textSize="18sp" />
```

Рисунок 1.45 – Додавання ідентифікатора до textView

Залишається змінити метод активності onBtnClick для виведення у макет списку студентів обраної групи.

Для отримання посилання на компонент графічного інтерфейсу можна скористатися методом findViewById(). Метод findViewById() отримує ідентифікатор компонента у вигляді параметра і повертає об’єкт View. Далі залишається привести значення, що повертається до правильного типу компонента (наприклад, TextView або Button). Передаючи ідентифікатор компонента, буде використано спеціальний клас R, який генерується інструментарієм Android під час створення або побудови програми. Він знаходиться в папці app/build/generated/source/r/debug вашого проєкту –

всередині папки, ім'я якої збігається з ім'ям пакета програми. Android використовує R.java для відстеження ресурсів, використовуваних у застосунку; серед іншого, цей клас дозволяє отримувати посилання на компоненти графічного інтерфейсу з коду активності.

Нижче, на рис. 1.46 наведено оновлений код методу onBtnClick.

```
19      public void onBtnClick(View view) {
20  //          CharSequence text = "Ви натистули на кнопку!";
21  //          int duration = Toast.LENGTH_LONG;
22  //          Toast toast = Toast.makeText(this, text, duration);
23  //          toast.show();
24
25              Spinner spinner = (Spinner) findViewById(R.id.spinner);
26              String grpNumb = (String) spinner.getSelectedItem();
27
28              String txtStudents = "";
29              for(Student s: Student.getStudents(grpNumb)) {
30                  txtStudents += s.getName() + "\n";
31              }
32
33              TextView textView = (TextView) findViewById(R.id.text);
34              textView.setText(txtStudents);
35      }
```

Рисунок 1.46 – Код методу onBtnClick

Ми коментуємо попередній варіант коду, де виводилось повідомлення про натискання на кнопку, та додаємо новий. Тут, за допомогою методу findViewById отримується компонент spinner, з якого через метод getSelectedItem у змінну grpNumb записується значення обраної групи. Далі у циклі перебираємо всіх студентів отриманої групи, що повертає getStudents, та додаємо їх імена (отримані через getName()) до змінної txtStudents. Після цього, через findViewById отримуємо компонент textView та встановлюємо йому текст – сформований список студентів обраної групи.

Перевіримо роботу застосунку, запустивши app run (рис. 1.47). У випадному списку виберемо якусь групу та натиснемо кнопку «Показати список».

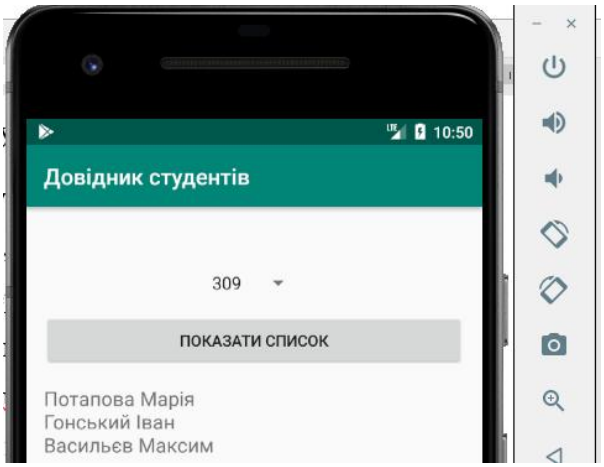



Рисунок 1.47 – Перевірка роботи застосунку «Довідник студентів»

## 1.3. Інтенти та передача даних між активностями

Активність – одна чітко визначена операція, яка може виконуватися користувачем, наприклад, відображення списку студентських груп. У дуже простому застосунку може бути достатньо однієї активності. Але зазвичай користувачеві потрібно виконувати більше однієї операції – наприклад, не тільки виводити список студентів, але і додавати до нього нових. У таких випадках в застосунку використовуються різні активності: одна, наприклад, для відображення списку студентських груп, а інша – для перегляду студентів.

Розберемо, що необхідно зробити для реалізації цієї задачі:

– додати до застосунку другу активність та макет;

– організувати виклик другої активності із першої;

– організувати передачу даних щодо поточної студентської групи із першої активності до другої;

– перенести логіку щодо отримання та виведення студентів із першої активності до другої.

Під час старту застосунку пристрій завантажуватиме головну активність, яка відображатиме на екрані свій макет. Далі по натисканню на кнопку завантажуватиметься друга активність, що замінюватиме користувацький екран власним макетом. Повернення до головної активності буде можливим за допомогою використання стандартної кнопки пристрою «назад».

Для додавання нової активності виділимо пакет com.chnulabs.students та оберемо New –> Activity –> Empty activity (рис. 1.48).

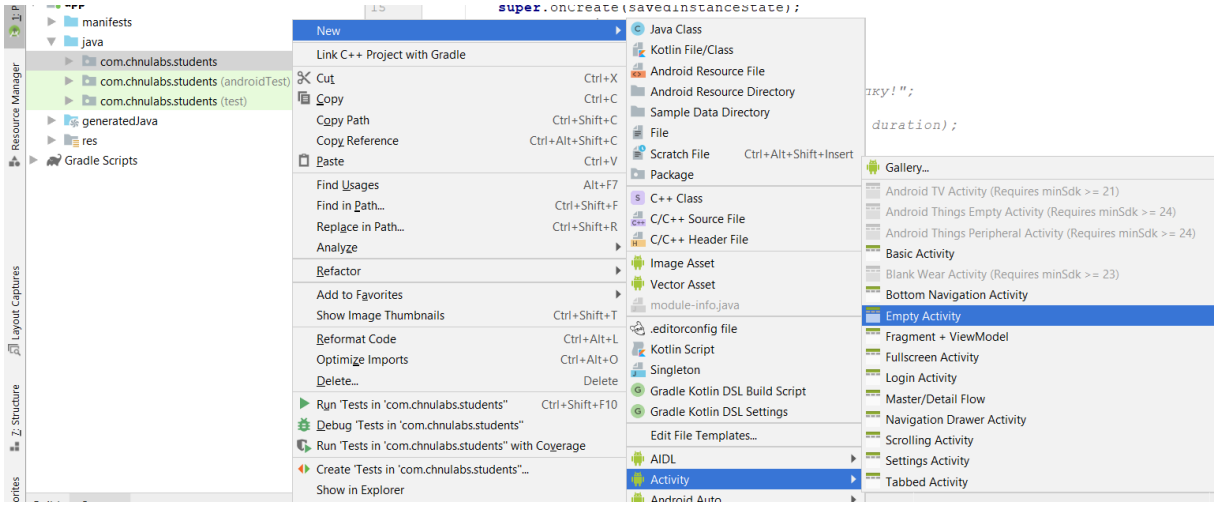



Рисунок 1.48 – Створення Empty activity

Далі вводимо ім’я активносні StudentsListActivity та залишаємо прапорець «Create layout file» (рис. 1.49).

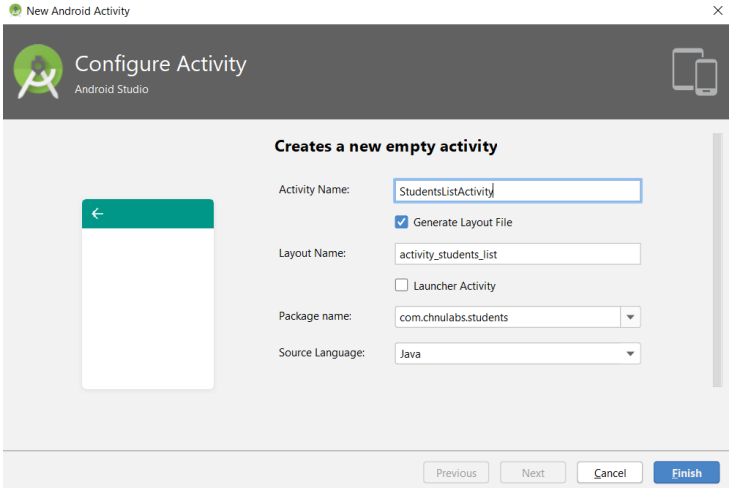



Рисунок 1.49 – Створення Empty activity (продовження)

Оновимо макет активності – встановимо LinearLayout та перенесемо туди компонент TextView із макета головної активності (рис. 1.50.)

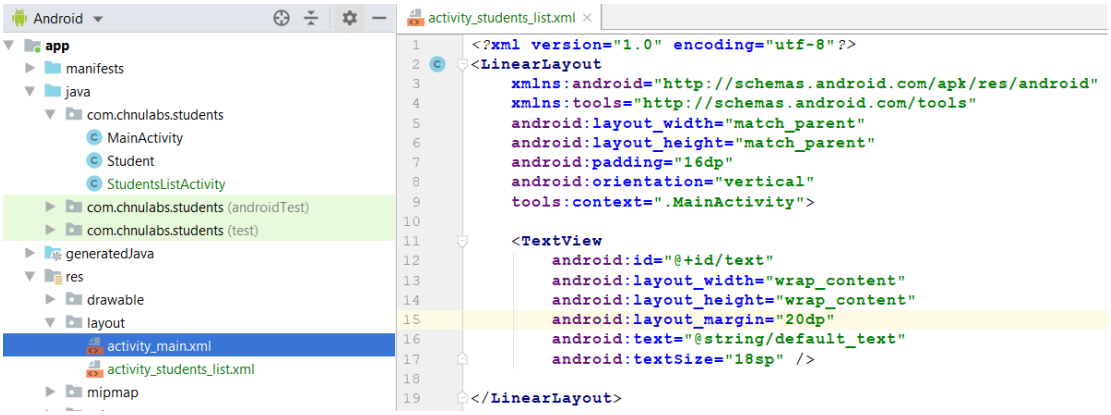



Рисунок 1.50 – Макет активності StudentsListActivity activity_students_list.xml

Після видалення компонента TextView із макета головної активності він матиме такий вигляд (рис. 1.51):

```
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Показати список"
        android:layout_marginBottom="16dp"
        android:onClick="onBtnClick"/>

</LinearLayout>
```

Рисунок 1.51 – Макет головної активності

Кожен Android-застосунок повинен містити файл з ім'ям AndroidManifest.xml. Ви знайдете його в папці app/src/main свого проєкту. Файл AndroidManifest.xml містить найважливішу інформацію про програму: які активності вона містить, які бібліотеки їй необхідні та інші оголошення. Android створює цей файл під час створення програми. Наш екземпляр AndroidManifest.xml виглядає таким чином (рис. 1.52):

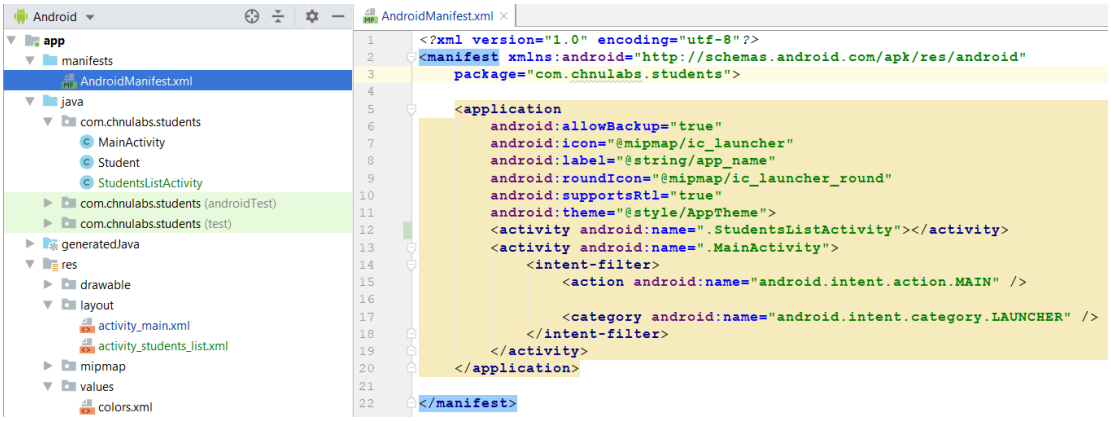



Рисунок 1.52 – AndroidManifest.xml нашого застосунку

Усі активності повинні бути оголошені у файлі AndroidManifest.xml. Якщо активність не оголошена у файлі, то система не знатиме про його існування. А якщо система не знає про активність, то активність не буде виконуватися. Активності оголошуються в маніфесті включенням елемента <activity> в елемент <application>.

На цей момент наш проєкт містить дві активності, одна з яких (MainActivity) зазначена у AndroidManifest.xml як головна та як launcher нашого застосунку. Наступний крок – змусити MainActivity викликати StudentsListActivity, коли користувач натисне на кнопку «Показати список».

Щоб запустити одну Активність з іншої, будемо використовувати інтент. Інтент можна розглядати як свого роду «намір виконати якусь операцію». Це різновид повідомлення, що дозволяє зв'язати різнорідні об'єкти (наприклад, активності) на стадії виконання. Якщо одна активність хоче запустити іншу, вона відправляє для цього інтент системі Android. Android запускає другу активність і передає їй інтент.

Перейдемо до коду головної активності, а саме – методу onBtnClick. Видалимо весь попередній код, попередньо зберігши його десь у іншому місці (він нам знадобиться надалі у процесі реалізації другої активності). Замість нього наберемо такий код (рис. 1.53).

```
20      public void onBtnClick(View view) {
21          Intent intent = new Intent( packageContext: this, StudentsListActivity.class);
22          startActivity(intent);
23      }
24  }
```

Рисунок 1.53 – Запуск активності StudentsListActivity
у методі onBtnClick

Конструктор інтенту має два параметри. Перший параметр повідомляє Android, від якого об'єкта надійшов інтент; для позначення поточної активності використовується ключове слово this. У другому параметрі передається ім'я класу активності, яка повинна отримати інтент.

Після того, як інтент буде створений, він передається Android викликом startActivity. Цей виклик наказує Android запустити активність, яка визначається інтентом. Під час отримання інтенту Android переконується в тому, що все правильно, і наказує активності запуститися. Якщо знайти активність не вдалося, ініціюється виключення ActivityNotFoundException.

Перевіримо роботу застосунку, запустивши run app (рис. 1.54).

Спочатку запускається головна активність та відображає макет із списком студентських груп та кнопкою, але вже без списку студентів. Далі по натисканню на кнопку «Показати список» завантажується друга активність та відображає макет із TextView для списку студентів. На цьому етапі він містить текст за замовчуванням, вказаний у strings.xml.

Для того, щоб реалізувати у другій активності механізм отримання та відображення списку студентів обраної групи, необхідно повідомити другій активності, яка група буда обрана у макеті першої.

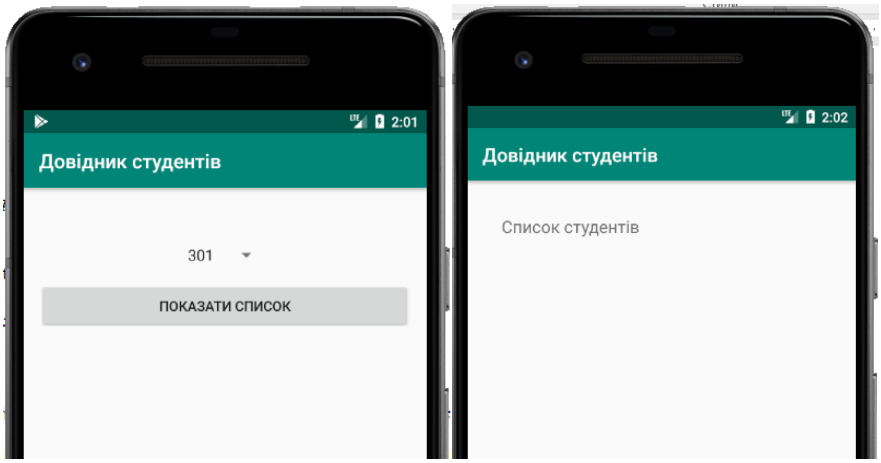



Рисунок 1.54 – Перехід між активностями

У інтент можна додати додаткову інформацію, яка повинна передаватися одержувачу. В цьому випадку активність, яка отримала інтент, зможе на нього якось зреагувати. Для цього використовується метод putExtra(), що має 2 параметри. Перший – ім'я ресурсу для переданої інформації, а другий – його значення. Перевантаження методу putExtra() дозволяє передавати значення багатьох можливих типів. Наприклад, це може бути примітив (скажімо, boolean або int), масив примітивів або String. Багаторазові виклики putExtra() дозволяють включити в інтент набір даних.

Для отримання переданих даних з боку другої активності необхідно спочатку отримати інтент за допомогою методу getIntent(), після чого до отриманого екземпляра можна застосувати методи getStringExtra (або getIntExtra чи ін.). Цей метод має один параметр – ім'я ресурсу.

Реалізуємо передачу значення обраної групи із першої активності в другу. Почнемо із визначення назви ресурсу, що буде передаватись. Зробимо це у класі другої активності StudentsListActivity (рис. 1.55).

Повертаємось до методу onBtnClick та передамо значення обраної групи до інтенту (рис. 1.56).

Тепер реалізуємо прийом переданої групи у другій активності. Для цього розмістимо відповідний код у методі onCreate активності StudentsList Activity. Повний код класу StudentsListActivity наведено на рис. 1.57.

StudentsListActivity.java

```
1  package com.chnulabs.students;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6
7  public class StudentsListActivity extends AppCompatActivity {
8
9      public static final String GROUP_NUMBER = "groupnumber";
10
```

Рисунок 1.55 – Визначення назви ресурсу – номера групи

```
20     public void onBtnClick(View view) {
21         Spinner spinner = (Spinner) findViewById(R.id.spinner);
22         String grpNumb = (String) spinner.getSelectedItem();
23
24         Intent intent = new Intent( packageContext: this, StudentsListActivity.class);
25         intent.putExtra(StudentsListActivity.GROUP_NUMBER, grpNumb);
26
27         startActivity(intent);
28     }
```

Рисунок 1.56 – Передача обраної студентської групи до інтенту

StudentsListActivity.java

```
1  package com.chnulabs.students;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.widget.TextView;
8
```

```
9  public class StudentsListActivity extends AppCompatActivity {
10
11     public static final String GROUP_NUMBER = "groupnumber";
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_students_list);
17
18         Intent intent = getIntent();
19         String grpNumber = intent.getStringExtra(GROUP_NUMBER);
20
21         String txtStudents = "";
22         for(Student s: Student.getStudents(grpNumber)) {
23             txtStudents += s.getName() + "\n";
24         }
25
26         TextView textView = (TextView) findViewById(R.id.text);
27         textView.setText(txtStudents);
28     }
29 }
```

Рисунок 1.57 – Клас StudentsListActivity

Перевіримо роботу нашого коду, запустивши застосунок на виконання (рис. 1.58).

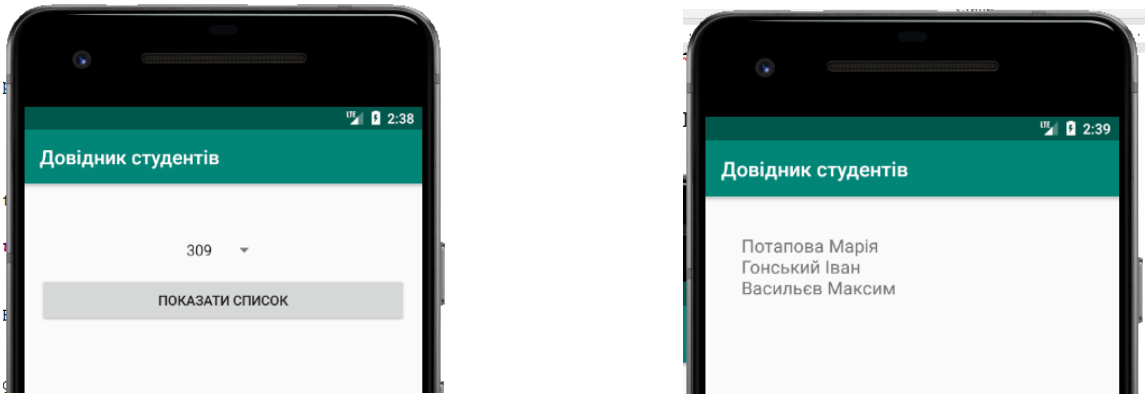



Рисунок 1.58 – Передача даних у інтент та їх обробка

Як вже зазначалося, Android-застосунки складаються з однієї або декількох активностей. Кожна активність представляє одну чітко визначену операцію, яка може виконуватися користувачем. Наприклад, такі застосунки, як Gmail, Viber, Facebook і Twitter, містять активності, що дозволяють відправляти повідомлення, хоча в кожному застосунку ця операція виконується по-своєму.

Ми вже використали інтент для запуску другої активності. Цей принцип стосується і активностей інших застосунків. Активність нашого застосунку передає інтент Android, Android перевіряє його, а потім наказує другій активності запуститися – незважаючи на те, що ця активність знаходиться в іншому застосунку. Наприклад, можна скористатися інтентом для запуску активності Gmail, що відправляє повідомлення, і передати їй текст, який потрібно відправити. Замість того, щоб писати власні активності для відправки електронної пошти, можна скористатися готовим застосунком

Gmail. Це означає, що об'єднуючи активності на пристрої в ланцюжок, ви можете будувати програми, що мають значно більшу функціональність.

Ми не можемо знати, які програми встановлені на пристрої, але ця проблема вирішується за допомогою дій (actions). Дії – стандартний механізм, за допомогою якого Android дізнається про те, які стандартні операції можуть виконуватися активностями. Наприклад, Android знає, що всі активності, зареєстровані для дії send, можуть відправляти повідомлення. Тобто, якщо потрібно виконати деяку дію і нас не цікавить, якою саме активністю вона буде виконана, ми створюємо неявний інтент та повідомляємо Android, яку дію потрібно виконати. Вибір активності, яка виконує цю дію, доручаються Android.

Для створення інтенту із зазначенням дії застосовується синтаксис Intent intent = new Intent(дія), де дія – тип дії, що виконується активністю. Додамо до активності StudentsListActivity кнопку, за натисканням якої список студентів буде відправлятись у вигляді повідомлення. Для цього додаємо до макета activity_students_list.xml кнопку (рис. 1.59).

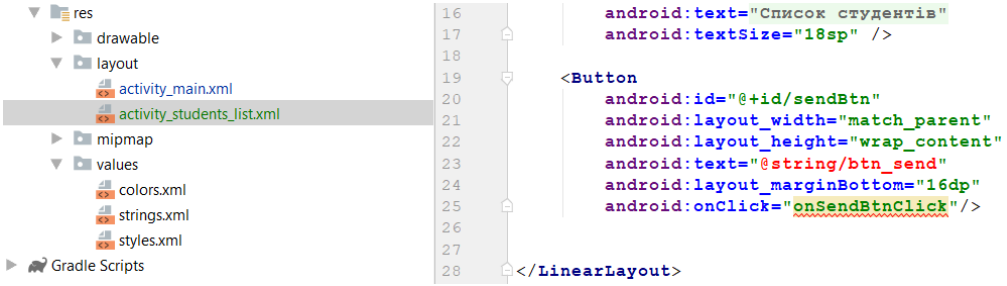



Рисунок 1.59 – Додавання кнопки до макета activity_students_list.xml

Визначимо значення btn_send у strings.xml (рис. 1.60) та пропишемо метод onSendBtnClick в класі активності StudentsListActivity.java (рис. 1.61).

```
9           <item>309</item>
10      </string-array>
11      <string name="btn_send">Відправити повідомленням</string>
12  </resources>
```

Рисунок 1.60 – Додавання значення btn_send в strings.xml

```
StudentsListActivity.java
24              txtStudents += s.getName() + "\n";
25          }
26
27          TextView textView = (TextView) findViewById(R.id.text);
28          textView.setText(txtStudents);
29      }
30
31      public void onSendBtnClick(View view) {
32          TextView textView = (TextView) findViewById(R.id.text);
33
34          Intent intent = new Intent(Intent.ACTION_SEND);
35          intent.setType("text/plain");
36          intent.putExtra(Intent.EXTRA_TEXT, textView.getText().toString());
37          intent.putExtra(Intent.EXTRA_SUBJECT,  value: "Список студентів");
38          startActivity(intent);
39      }
40  }
```

Рисунок 1.61 – Метод onSendBtnClick класу активності StudentsListActivity

Перевіряємо роботу застосунку. Після запуску переходимо до активності списку студентів, натискаємо кнопку «Відправити повідомлення». Далі для відправки повідомлення обираємо Gmail (рис. 1.62).

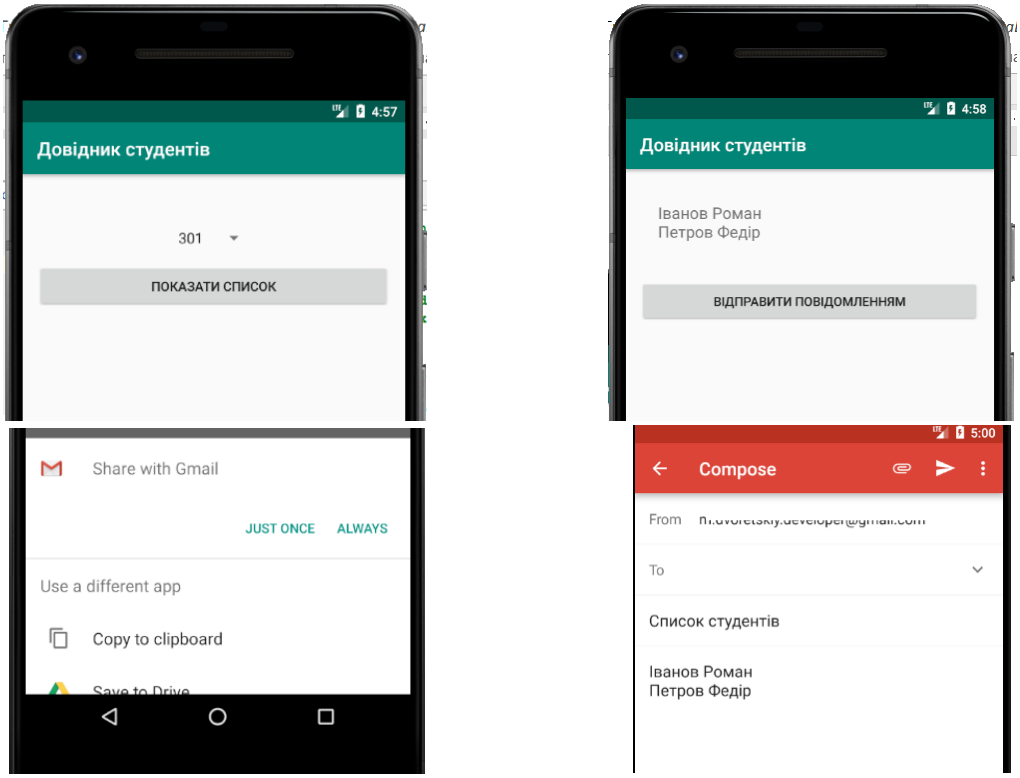



Рисунок 1.62 – Відправка листа із списком студентів через Gmail

## 1.4. Життєвий цикл активності

Додамо до макета другої активності проєкту activity_students_list.xml кнопку, за натисканням на яку розмір шрифту компонентів TextView, в якій відображається список студентів, збільшуватиметься на 20%. Розмістимо цю кнопку між списком студентів та кнопкою «Відправити повідомлення» (рис. 1.63).

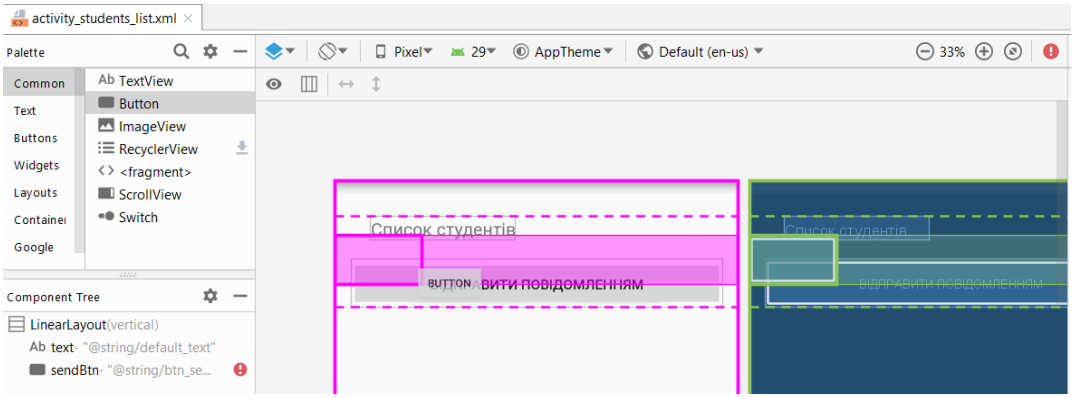



Рисунок 1.63 – Додавання кнопки до activity_students_list.xml у візуальному редакторі

Далі перейдемо на вкладку «text», змінюємо ширину кнопки, її текст, ідентифікатор та додаємо подію onClick для подальшої обробки натискання у коді активності (рис. 1.64).

```
18
19      <Button
20          android:id="@+id/btnPlus"
21          android:onClick="onPlusBtnClick"
22          android:layout_width="wrap_content"
23          android:layout_height="wrap_content"
24          android:text="+" />
25
```

Рисунок 1.64 – Зміна властивостей компонента нової кнопки

Тепер відкриємо код класу другої активності StudentsListActivity та реалізуємо збільшення шрифту відображення списку студентів. Почнімо із того, що визначимо приватне поле для збереження поточного значення розміру шрифту TextView (рис. 1.65).

```
14      private float textSize = 0;
15
```

Рисунок 1.65 – Визначення приватного поля

Далі у методі onCreate виконаємо заповнення змінної поточним значенням розміру шрифту текстового поля для відображення списку студентів (рис. 1.66).

```
16      @Override
17      protected void onCreate(Bundle savedInstanceState) {
18          super.onCreate(savedInstanceState);
19          setContentView(R.layout.activity_students_list);
20
21          Intent intent = getIntent();
22          String grpNumber = intent.getStringExtra(GROUP_NUMBER);
23
24          String txtStudents = "";
25          for(Student s: Student.getStudents(grpNumber)) {
26              txtStudents += s.getName() + "\n";
27          }
28
29          TextView textView = (TextView) findViewById(R.id.text);
30          textView.setText(txtStudents);
31
32          textSize = textView.getTextSize();
33      }
```

Рисунок 1.66 – Ініціалізація textSize

І, нарешті, реалізуємо метод onPlusBtnClick, що оброблятиме натискання на кнопку (рис. 1.67).

```
34
35      public void onPlusBtnClick(View view) {
36          textSize = textSize * 1.1f;
37          TextView textView = findViewById(R.id.text);
38          textView.setTextSize(TypedValue.COMPLEX_UNIT_PX, textSize);
39      }
```

Рисунок 1.67 – Реалізація методу onPlusBtnClick()

Перевіримо роботу нашого застосунку. Запускаємо його на виконання та переходимо до форми списку студентів. Натискаємо декілька разів на кнопку «+», щоб зміна розміру шрифту стала помітною (рис. 1.68).

Але якщо тепер ми спробуємо виконати поворот пристрою, то розмір шрифту повертається до початкового значення (рис. 1.69).

Спробуємо розібратись, чому так відбувається:

– користувач запускає застосунок та переходить до потрібної активності;

– відбувається ініціалізація змінної textSize значенням поточного розміру шрифту;

– користувач натискає на кнопку «+»;

– спрацьовує метод активності, що збільшує змінну textSize та встановлює розмір шрифту = textSize;

– користувач повертає пристрій;

– Android фіксує, що орієнтація екрана змінилася, знищує активність разом з усіма її змінними та створює її заново для нової орієнтації екрана;

– відбувається ініціалізація змінної textSize значенням поточного розміру шрифту, тобто того, що було взято із макета під час створення активності;

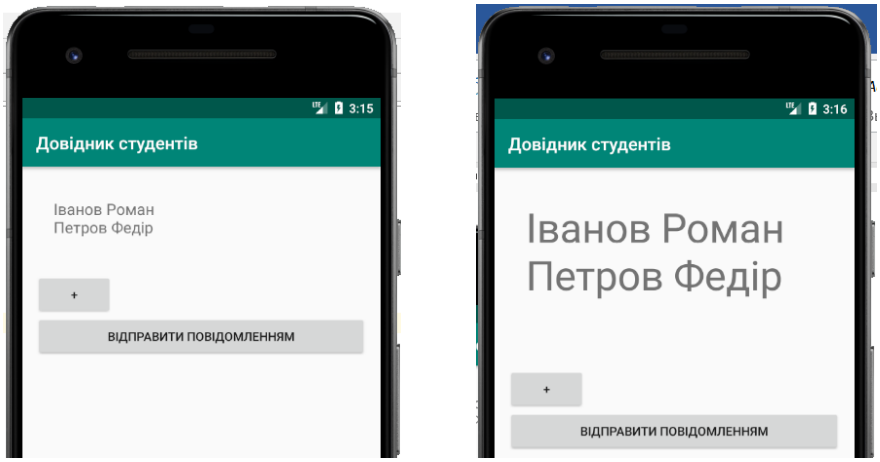



Рисунок 1.68 – Використання кнопки для зміни розміру шрифту

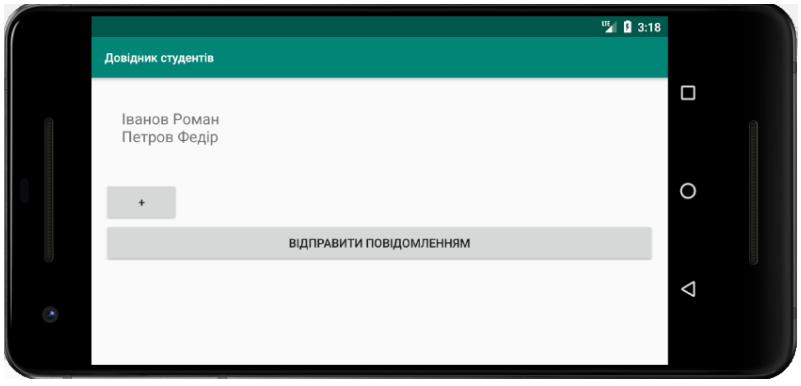



Рисунок 1.69 – Відновлення розміру шрифту під час повороту пристрою

Під час запуску активності на початку роботи програми Android бере до уваги конфігурацію пристрою. Під цим терміном розуміється як конфігурація фізичного пристрою (розмір екрана, орієнтація екрана, наявність клавіатури), так і параметри конфігурації, задані користувачем (наприклад, локальний контекст). Система Android повинна знати конфігурацію пристрою під час запуску активності, тому що ця інформація може вплинути на ресурси, необхідні застосунку. Наприклад, у горизонтальній орієнтації може використовуватися інший макет, а у разі вибору російського чи англійського локального контексту може знадобитися інший набір строкових ресурсів.

У процесі зміни конфігурації пристрою всі компоненти програми, що відображають призначений для користувача інтерфейс, повинні бути оновлені відповідно до нової конфігурації. Якщо повернути пристрій, Android помічає, що орієнтація і розміри екрана змінилися, і інтерпретує цей факт як зміну конфігурації пристрою. Поточна активність знищується і створюється заново, щоб застосунок міг вибрати ресурси, відповідні новій конфігурації.

Активність виконується, коли вона знаходиться на першому плані на екрані. Метод onCreate() викликається під час створення активності; саме тут відбувається основна настройка активності. Метод onDestroy() викликається безпосередньо перед знищенням активності. Щоб зберегти поточний стан активності, також необхідно реалізувати метод onSaveInstanceState(). Метод onSaveInstanceState() викликається перед знищенням активності; це означає, що вам випаде можливість зберегти всі значення, які потрібно зберегти, перш ніж вони будуть безповоротно втрачені.

Метод onSaveInstanceState() отримує один параметр типу Bundle. Тип Bundle дозволяє об’єднати різні типи даних в один об’єкт. Для включення map «ім’я/значення» в Bundle використовуються методи Bundle putInt, putString та ін. Метод onCreate() отримує параметр Bundle, завдяки чому можна виконати читання попередньо збережених значень.

Виконаємо необхідні зміни у коді активності StudentsListActivity. Почнемо із реалізації методу onSaveInstanceState (рис. 1.70).

```
50
51      @Override
52      protected void onSaveInstanceState(Bundle outState) {
53          super.onSaveInstanceState(outState);
54          outState.putFloat("textSize", textSize);
55      }
```

Рисунок 1.70 – Збереження змінної textSize в onSaveInstanceState

Також виконаємо читання збережених даних у методі onCreate (рис. 1.71).

```
29          TextView textView = (TextView) findViewById(R.id.text);
30          textView.setText(txtStudents);
31
32          textSize = textView.getTextSize();
33          if (savedInstanceState != null) {
34              textSize = savedInstanceState.getFloat( key: "textSize");
35          }
36      }
```

Рисунок 1.71 – Читання збережених даних змінної textSize

Перевіримо роботу застосунку. Збільшимо розмір шрифту (рис. 1.72) та повернемо пристрій (рис. 1.73).

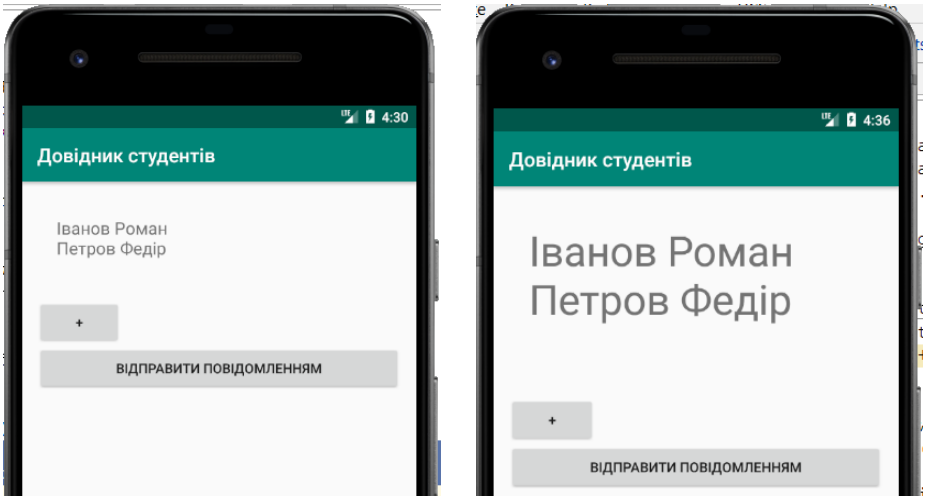



Рисунок 1.72 – Збільшення розміру шрифту

Розглянемо іншу задачу. На головній активності реалізуємо таймер, що фіксуватиме проміжок часу, протягом якого на екрані користувача відображається ця активність. По-перше, для цього нам знадобиться приватне поле seconds, в якому буде зберігатись значення кількості секунд з моменту завантаження активності (рис. 1.74).

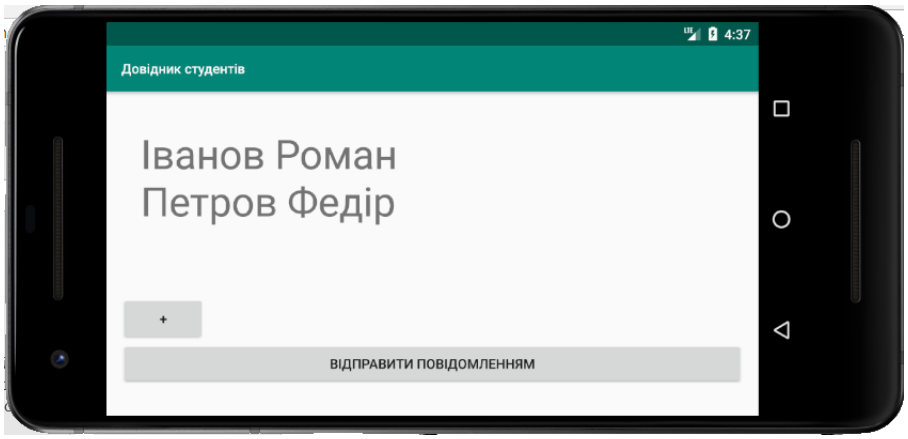



Рисунок 1.73 – Збереження розміру шрифту після повороту екрана

```
15
16      private int seconds = 0;
```

Рисунок 1.74 – Додавання приватного поля seconds в головну активність

Для вирішення задачі накопичення секунд буде використано клас Handler. Handler – клас Android, який може використовуватися для планування виконання коду в певний момент у майбутньому. Також клас може використовуватися для передачі коду, який повинен виконуватися в іншому програмному потоці. У нашому прикладі Handler буде використовуватися для планування виконання коду кожну секунду.

Щоб використовувати клас Handler, упакуйте код, який потрібно запланувати, в об'єкт Runnable, після чого використовуйте методи post() і postDelayed() класу Handler для визначення того, коли повинен виконуватися цей код.

Реалізуємо метод runTimer, що визначатиме код, що буде виконуватись кожну секунду (рис. 1.75).

```
34
35      private void runTimer() {
36          final TextView timeView = (TextView)findViewById(R.id.textView);
37          final Handler handler = new Handler();
38          handler.post(new Runnable() {
39              @Override
40              public void run() {
41                  int hours = seconds/3600;
42                  int minutes = (seconds%3600)/60;
43                  int secs = seconds%60;
44                  String time = String.format(Locale.getDefault(),
45                          format: "%d:%02d:%02d", hours, minutes, secs);
46                  timeView.setText(time);
47                  seconds++;
48                  handler.postDelayed( r: this,  delayMillis: 1000);
49              }
50          });
51      }
```

Рисунок 1.75 – Реалізація класі runTimer

Та під час завантаження активності у методі onCreate запускаємо наш таймер, виконавши runTimer (рис. 1.76).

```
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_main);
21
22          runTimer();
23      }
```

Рисунок 1.76 – Запуск таймера в onCreate

Останній крок – збереження значення змінної seconds в методі onSaveInstanceState та відновлення у методі onCreate для запобігання втрати значення під час повороту пристрою (рис. 1.77–1.78).

```
28
29      @Override
30      protected void onSaveInstanceState(Bundle outState) {
31          super.onSaveInstanceState(outState);
32          outState.putInt("seconds", seconds);
33      }
```

Рисунок 1.77 – Збереження змінної seconds

```
18    protected void onCreate(Bundle savedInstanceState) {
19        super.onCreate(savedInstanceState);
20        setContentView(R.layout.activity_main);
21
22        if (savedInstanceState != null) {
23            seconds = savedInstanceState.getInt( key: "seconds");
24        }
25
26        runTimer();
27    }
```

Рисунок 1.78 – Відновлення значення змінної seconds

Перевіримо роботу застосунку (рис. 1.79).

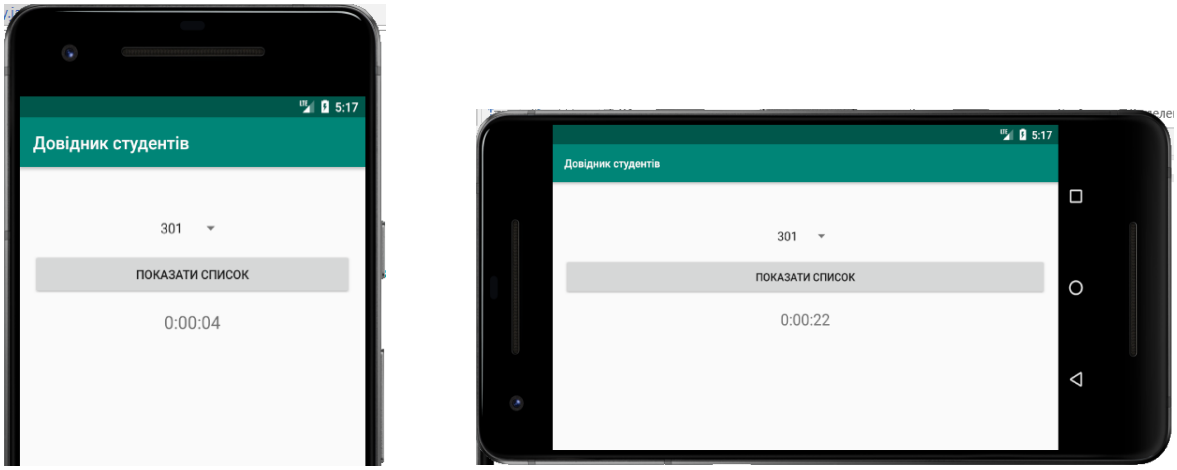



Рисунок 1.79 – Робота таймеру при повороті екрану

Як бачимо, значення таймера зберігається під час повороту пристрою. Але є інша проблема – у разі переходу на іншу активність наш таймер не зупиняється (рис. 1.80).

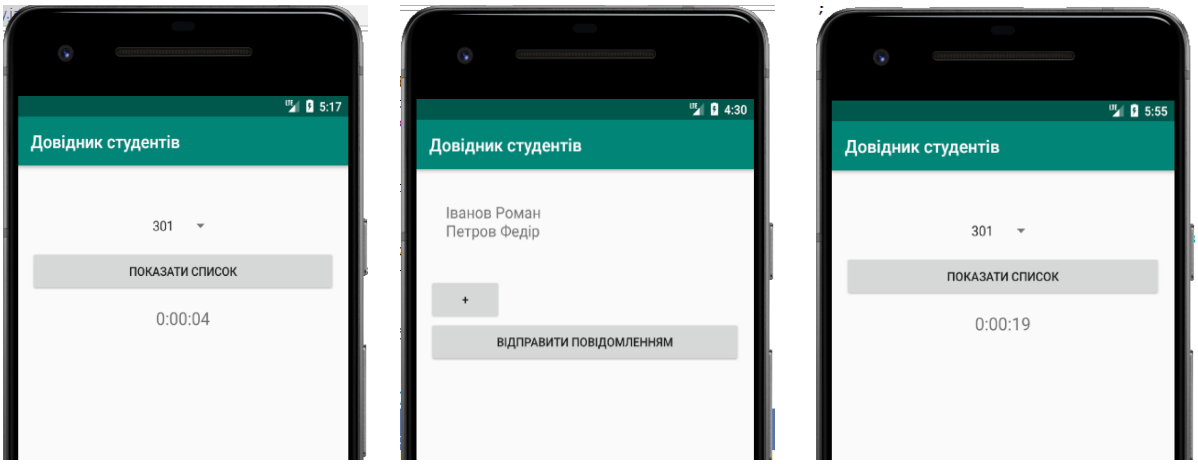



Рисунок 1.80 – Продовження роботи таймеру
під час переходу на іншу активність

Отже, нам потрібно організувати зупинку роботи таймера, якщо активність не видима на екрані. Для реалізації цієї задачі для початку спробуємо розібратися із основними етапами життєвого циклу активності (рис. 1.81).

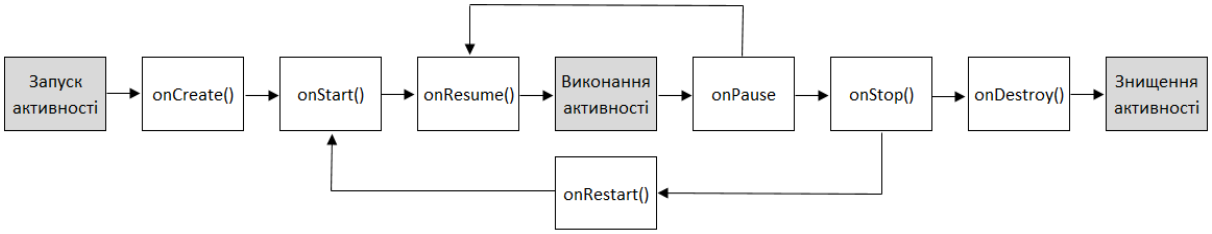



Рисунок 1.81 – Життєвий цикл активності

1. Активність запускається, виконуються її методи onCreate() і onStart(). У цій точці активність видно, але вона ще не володіє фокусом.

2. Викликається метод onResume(). Він виконується тоді, коли активність збирається перейти на перший план. Після виконання методу onResume() активність отримує фокус, а користувач може взаємодіяти з нею.

3. Метод onPause() виконується тоді, коли активність перестає перебувати на першому плані. Після виконання методу onPause() активність залишається видимою, але не володіє фокусом.

4. Якщо активність знову повертається на перший план, викликається метод onResume(). Активність, яка багаторазово втрачає і отримує фокус, може проходити цей цикл кілька разів.

5. Якщо активність перестає бути видимою користувачу, викликається метод onStop(). Після виконання методу onStop() активність не видно користувачеві.

6. Якщо активність знову стане видимою, викликається метод onRestart(), за яким слідують onStart() та onResume(). Активність може проходити через цей цикл багаторазово.

7. Нарешті, активність знищується. Під час переходу від виконання до знищення методи onPause() і onStop() викликаються до того, як активність буде знищена.

Для вирішення нашої задачі виконуємо зупинку таймера, якщо активність перестає бути видимою користувачу, та відновлення його роботи, коли активність знову стане видимою. Скористатися методами onStart та onStop. Почнемо із додавання приватного поля isRunning до класу нашої активності (рис. 1.82).

```
14  public class MainActivity extends AppCompatActivity {
15
16      private int seconds = 0;
17      private Boolean isRunning = true;
18
```

Рисунок 1.82 – Додавання приватного поля isRunning

Реалізуємо onStart та onStop (рис. 1.83)

```
65      @Override
66      protected void onStart() {
67          super.onStart();
68          isRunning = true;
69      }
70
71      @Override
72      protected void onStop() {
73          super.onStop();
74          isRunning = false;
75      }
```

Рисунок 1.83 – Методи onStart та onStop

Та внесемо зміни у код методу runTimer для врахування значення isRunning (рис. 1.84).

```
47      private void runTimer() {
48          final TextView timeView = (TextView)findViewById(R.id.textView);
49          final Handler handler = new Handler();
50          handler.post(new Runnable() {
51              @Override
52              public void run() {
53                  int hours = seconds/3600;
54                  int minutes = (seconds%3600)/60;
55                  int secs = seconds%60;
56                  String time = String.format(Locale.getDefault(),
57                          format: "%d:%02d:%02d", hours, minutes, secs);
58                  timeView.setText(time);
59                  if (isRunning) {
60                      seconds++;
61                  }
62                  handler.postDelayed( r: this,  delayMillis: 1000);
63              }
64          });
65      }
```

Рисунок 1.84 – Зміни у runTimer

Перевіримо роботу застосунку (рис. 1.85).

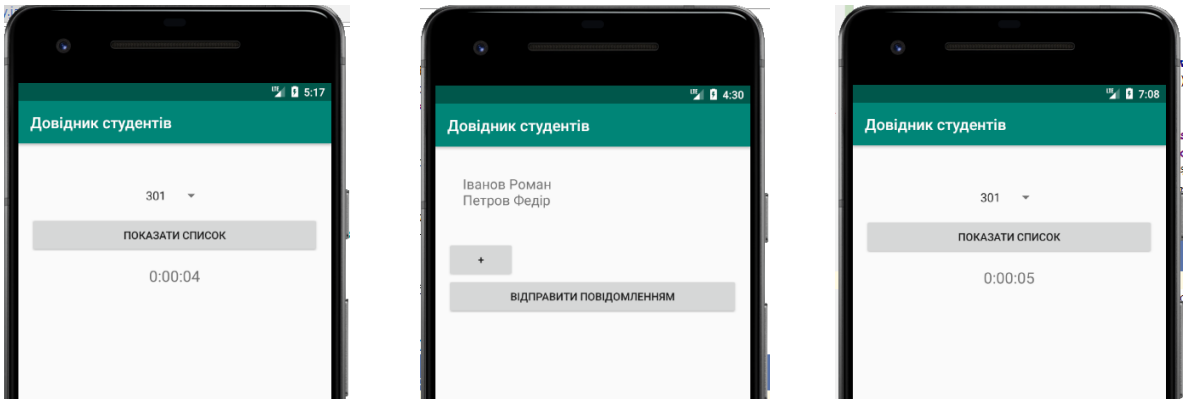



Рисунок 1.85 – Зупинка таймера при зміні активності

Як бачимо, тепер, у разі втрати активністю видимості, таймер зупиняється та продовжує свою роботу під час повернення активності на екран користувача.

# *РОЗДІЛ 2.*
# *МАКЕТИ ТА РОЗРОБКА ІНТЕРФЕЙСІВ*

## 2.1. Макети та зовнішній вигляд

Усі компоненти графічного інтерфейсу розширюють клас представлення – у внутрішній реалізації всі вони є субкласами класу android.view. View. Це означає, що всі компоненти, що використовуються в інтерфейсі застосунку, мають спільні атрибути і поведінку. Наприклад, всі вони можуть відображатися на екрані, а також можуть повідомляти інформацію про свою ширину і висоту. Кожен компонент графічного інтерфейсу, який використовується в інтерфейсі застосунку, наслідує цю базову функціональність і розширює її.

Але не тільки компоненти графічного інтерфейсу є спеціалізаціями класу представлення View. Усі макети також є субкласами класу android.view.ViewGroup. Група представлень – особливий різновид представлень, здатних містити інші представлення (рис. 2.1).

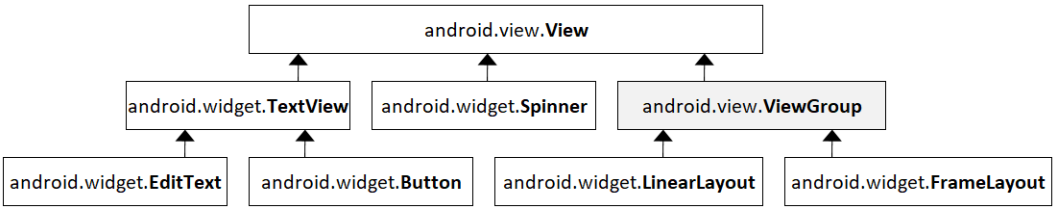



Рисунок 2.1 – Ієрархія наслідування від android.view.View

Об’єкт View займає прямокутну область екрана та включає функціональність, необхідну всім представленням для нормального існування в Android. Нижче перераховані деякі аспекти цієї функціональності:

1. Кожне представлення представлено об’єктом Java; це означає, що ви можете задавати і зчитувати його властивості в коді активності – наприклад, отримувати значення, вибране в списку, або змінювати текст напису. Конкретний набір властивостей і методів, які можуть використовуватися в коді, залежить від типу представлення. Кожному представленню можна призначити ідентифікатор, за яким до нього можна звертатися з коду.
2. За значеннями ширини і висоти, заданими у програмі, Android визначає необхідні розміри представлення. Також можна вказати, чи потрібно забезпечити представлення відступами. Після того, як представлення з’явиться на екрані, ви зможете отримати дані про його позицію, а також визначити фактичні розміри.
3. Android управляє передачею фокусу залежно від дій користувача. Зокрема, під час передачі фокусу враховуються події приховування, видалення або появи представлення.

4. Кожне з представлень може реагувати на події. Також розробник може створювати слухачів (listeners) для реакції на події, що відбуваються у представленні. Наприклад, всі представлення здатні реагувати на отримання або втрату фокусу, а кнопка (а також всі її субкласи) може реагувати на натискання.

Макет, який визначається в розмітці XML, формує ієрархічне дерево представлень і груп представлень. Наприклад, розглянемо лінійний макет головної активності нашого застосунку (рис. 2.2–2.3).

Лінійний макет є групою представлень, а спінер, кнопка і текстове поле – представленнями. Група представлень є батьком текстового поля, а представлення є нащадками відносно групи (рис. 2.4).

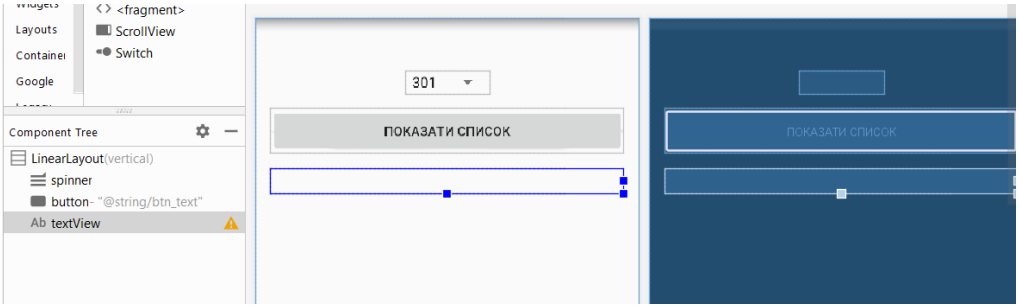



Рисунок 2.2 – Макет головної форми – візуальний редактор

MainActivity.java | activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <Spinner
        android:id="@+id/spinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="40dp"
        android:layout_marginBottom="16dp"
        android:entries="@array/students_groups"/>

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Показати список"
        android:layout_marginBottom="16dp"
        android:onClick="onBtnClick"/>

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="20dp" />

</LinearLayout>
```

Рисунок 2.3 – Макет головної форми – текстовий редактор

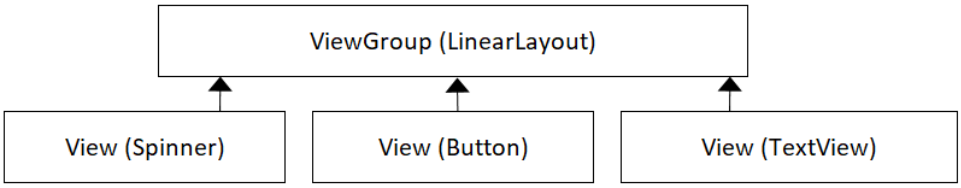



Рисунок 2.4 – Ієрархія представлень у макеті головної активності

Створимо активність для відображення атрибутів студентської групи. Виділивши мишею пакет com.chnulabs.students за допомогою команди new–>activity–>empty activity додаємо нову активність StudentsGroup Activity (рис. 2.5).

Використовуючи знайомий лінійний макет, реалізуємо такий зовнішній вигляд екрана користувача (рис. 2.6).

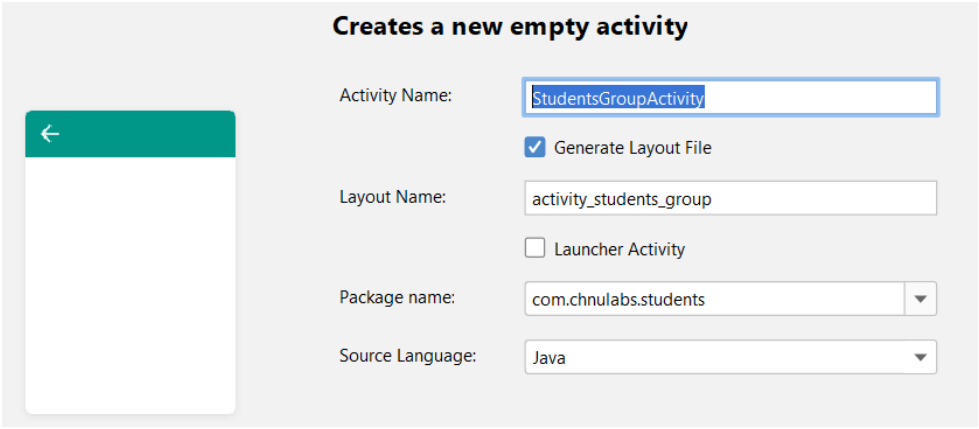



Рисунок 2.5 – Додавання нової активності

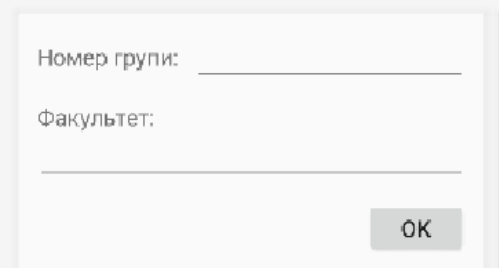



Рисунок 2.6 – Вигляд макета активності

Для початку пропишемо у strings.xml текст для елементів TextView та Button (рис. 2.7).

```
13      <string name="grp_number_txt">Номер групи:</string>
14      <string name="faculty_txt">Факультет:</string>
15      <string name="btn_ok_txt">OK</string>
```

Рисунок 2.7 – Доповнення strings.xml

Складність реалізації полягає у тому, що лінійний макет може розміщувати компоненти один за одним або горизонтально, або вертикально. Але нам потрібно, щоб текст «номер групи» та відповідне поле вводу знахо-

дились на одному рядку (горизонтальна орієнтація), а текст «факультет» та його поле вводу – один під одним (вертикальна орієнтація). Виходом є поміщення одного layout-у всередину іншого. Це є можливим виходячи із того, що, по-перше, layout є нащадком ViewGroup, що може включати до свого складу інші представлення. По-друге, ViewGroup, у свою чергу, є нащадком View, а отже може бути включений до іншого ViewGroup як дочірній елемент.

Наведемо дерево ієрархії компонент для нашого макета (рис. 2.8).

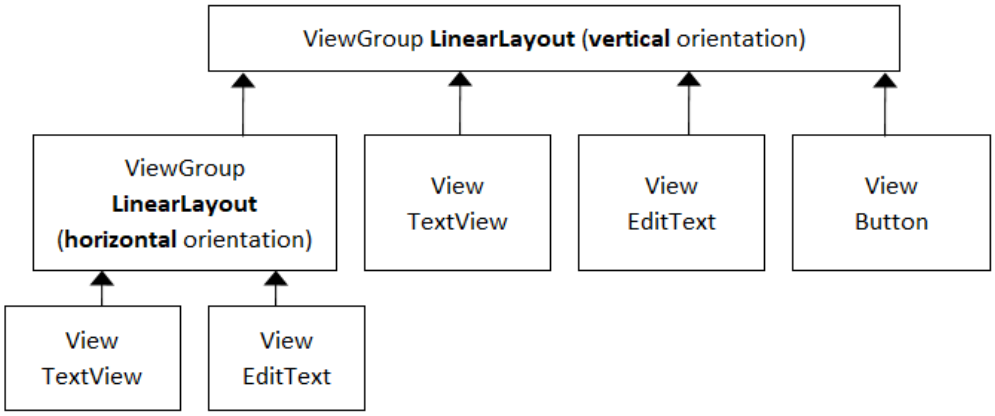



Рисунок 2.8 – Дерево ієрархії компонент для макета activity_students_group.xml

Повний код макета activity_students_group.xml наведено на рис. 2.9.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".StudentsGroupActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="2"
            android:text="@string/grp_number_txt"
            android:textSize="20dp"/>

        <EditText
            android:id="@+id/grpNumberEdit"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="3"
            android:ems="10"
            android:inputType="textPersonName"
            android:gravity="center"/>
    </LinearLayout>
```

```
32      <TextView
33          android:layout_width="match_parent"
34          android:layout_height="wrap_content"
35          android:text="Факультет:"
36          android:textSize="@dimen/default_text_size"
37          android:layout_marginTop="@dimen/default_padding"/>
38
39      <EditText
40          android:id="@+id/facultyEdit"
41          android:layout_width="match_parent"
42          android:layout_height="wrap_content"
43          android:ems="10"
44          android:inputType="textPersonName" />
45
46      <Button
47          android:id="@+id/btnOk"
48          android:layout_width="wrap_content"
49          android:layout_height="wrap_content"
50          android:text="@string/btn_ok_txt"
51          android:layout_gravity="end"
52          android:layout_marginTop="16dp"
53          android:textSize="20dp"/>
54  </LinearLayout>
```

Рисунок 2.9 – Код макету activity_students_group.xml

Звернімо увагу на деякі фрагменти коду. Для макета обов'язково задати значення для атрибутів layout_width та layout_height. Значення match_parent означає, що представлення займатиме всю область батьківського елемента. У випадку wrap_content представлення займе стільки місця, скільки потрібно для розміщення дочірніх елементів. Orientation для LinearLayout визначає спосіб розміщення елементів один за одним – горизонтально або вертикально. Padding визначає відступи по краях компонента.

Для елементів TextView за допомогою інструкції «android:text="@string/…» беремо значення тексту із файлу strings.xml. Для елементів EditText та Button вказуємо інструкцію «android:id="@+id/…"» для можливості звернення до компонент із коду активності.

Для компонентів TextView та EditText горизонтального лінійного представлення виконаємо пропорційне розміщення елементів на екрані: TextView займатиме 2/5 ширини екрана, а EditText – 3/5. Для цього вказуємо «android:layout_width="0dp"» та «android:layout_weight="2"», «android:layout_weight="3"» відповідно.

Для тексту написів TextView та Button вказуємо розмір штифту «android:textSize="20dp"», для лінійного представлення відступи з усіх боків «android:padding="16dp"» та для деяких інших елементів – відступ зверху «android:layout_marginTop="16dp"».

Для компонета EditText для виводу номера групи вказана властивість «android:gravity="center"». Це означає, що текст в середині елемента буде вирівняно по центру. Для кнопки ОК вказано «android:layout_gravity="end"», що говорить про вирівнювання самої кнопки по правому краю layout-у.

Звернімо увагу, що розмір шрифту та відступи мають однакові значення для всіх елементів. Якщо буде потреба змінити це значення, нам

доведеться зробити це в декількох місцях, що не є правильно та може привести до помилок у майбутньому. Для виправлення ситуації у папці res\values за допомогою команди new–>values resource file створимо файл dimens.xml (рис. 2.10) такого вмісту (рис. 2.11).

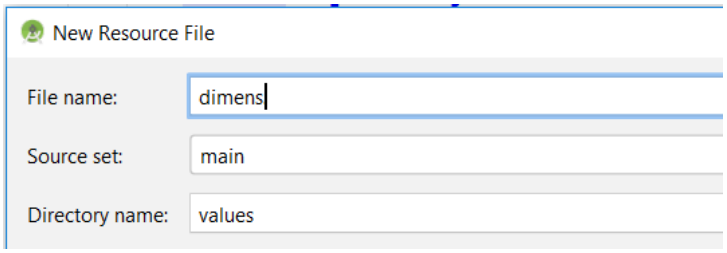



Рисунок 2.10 – Створення файлу ресурсів dimens.xml

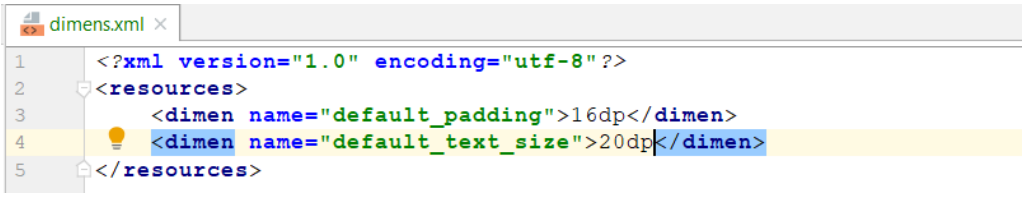



Рисунок 2.11 – Вміст файлу dimens.xml

Змінимо макет activity_students_group.xml для використання значень, доданих до dimens.xml. Для цього змінюємо «android:textSize="20dp"» на «android:textSize="@dimen/default_text_size"», а «android:padding="16dp"» і «android:layout_marginTop="@dimen/16dp"» на «android:padding="@dimen/default_padding"» і «android:layout_marginTop="@dimen/default_padding"».

Створимо клас Groups, в якому зберігатимуться дані щодо студентських груп. На пакеті com.chnulabs.students виконуємо new–>java class, StudentsGroup (рис. 2.12).

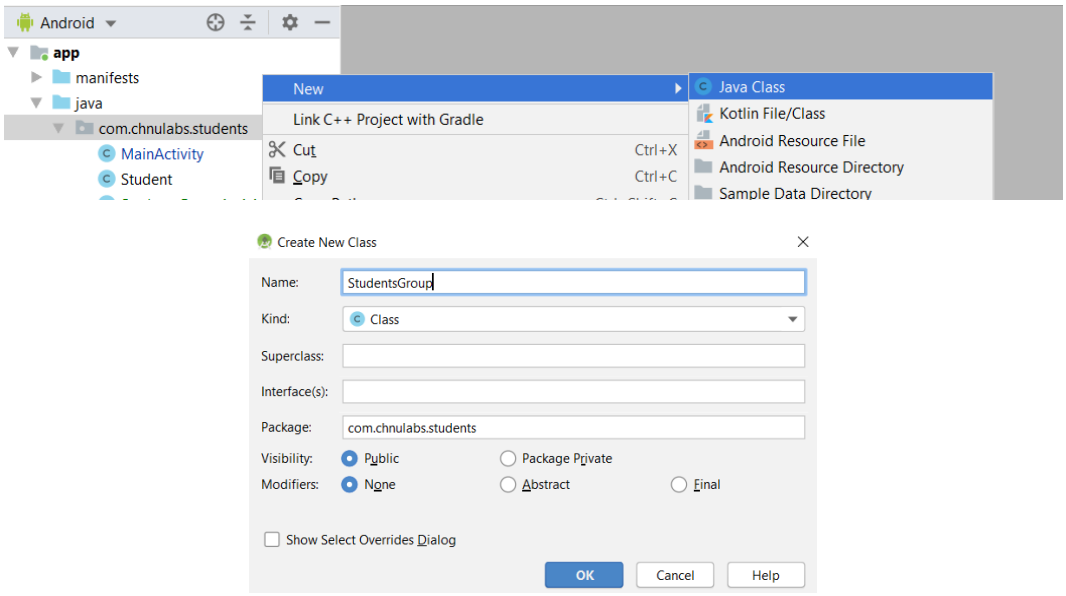



Рисунок 2.12 – Створення класу StudentsGroup

Клас буде дуже схожим на клас Students, створений у попередніх роботах, матиме набір відповідних полів, конструктор, гетери, статичну колекцію груп та метод, що повертає об’єкт-групу за вхідним параметром-рядком – номером групи (рис. 2.13).

StudentsGroup.java

```
package com.chnulabs.students;

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

public class StudentsGroup {
    private String number;
    private String facultyName;

    public StudentsGroup(String number, String facultyName) {
        this.number = number;
        this.facultyName = facultyName;
    }

    public String getNumber() {
        return number;
    }

    public String getFacultyName() {
        return facultyName;
    }

    private final static ArrayList<StudentsGroup> groups = new ArrayList<StudentsGroup>(
            Arrays.asList(
                    new StudentsGroup( number: "301", facultyName: "Комп'ютерних наук"),
                    new StudentsGroup( number: "302", facultyName: "Комп'ютерних наук"),
                    new StudentsGroup( number: "308", facultyName: "Комп'ютерних наук"),
                    new StudentsGroup( number: "309", facultyName: "Комп'ютерних наук")
            )
    );

    public static StudentsGroup getGroup(String groupNumber) {
        for(StudentsGroup g: groups) {
            if (g.getNumber().equals(groupNumber)) {
                return g;
            }
        }
        return null;
    }
}
```

Рисунок 2.13 – Код класу StudentsGroup

У макеті головної активності перед кнопкою перегляду списку студентів додаємо кнопку для переходу до активності StudentsGroupActivity (рис. 2.14) та у коді головної активності реалізуємо метод onGrpBtnClick (рис. 2.15).

activity_main.xml

```
    <Button
        android:id="@+id/btn_grp_detail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/btn_grp_detail_txt"
        android:onClick="onGrpBtnClick"/>
```

Рисунок 2.14 – Додавання кнопки у макет головної активності

```
    public void onGrpBtnClick(View view) {
        Spinner spinner = (Spinner) findViewById(R.id.spinner);
        String grpNumb = (String) spinner.getSelectedItem();

        Intent intent = new Intent( packageContext: this, StudentsGroupActivity.class);
        intent.putExtra(StudentsGroupActivity.GROUP_NUMBER, grpNumb);

        startActivity(intent);
    }
```

Рисунок 2.15 – Реалізація методу onGrpBtnClick у головній активності

Також у strings.xml додаємо напис для доданої кнопки (рис. 2.16).

```
12    <string name="btn_grp_detail_txt">Детальніше ...</string>
13
```

Рисунок 2.16 – Зміни у strings.xml

І, нарешті, реалізуємо заповнення текстових полів номера групи та факультету відповідно до групи, що була обрана в головній активності. Для цього в активності StudentsGroupActivity в методі onCreate додаємо код, що виршує цю задачу (рис. 2.17).

StudentsGroupActivity.java

```
1  package com.chnulabs.students;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.widget.EditText;
8
9  public class StudentsGroupActivity extends AppCompatActivity {
10
11     public static final String GROUP_NUMBER = "groupnumber";
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_students_group);
17
18         Intent intent = getIntent();
19         String grpNumber = intent.getStringExtra(GROUP_NUMBER);
20         StudentsGroup
21         group = StudentsGroup.getGroup(grpNumber);
22
23         EditText txtGrpNumber = (EditText) findViewById(R.id.grpNumberEdit);
24         txtGrpNumber.setText(group.getNumber());
25
26         EditText txtFacultyName = (EditText) findViewById(R.id.facultyEdit);
27         txtFacultyName.setText(group.getFacultyName());
28     }
29 }
```

Рисунок 2.17 – Код активності StudentsGroupActivity

Перевіримо роботу застосунку. На головній активності обираємо групу та натискаємо «Детальніше …» (рис. 2.18).

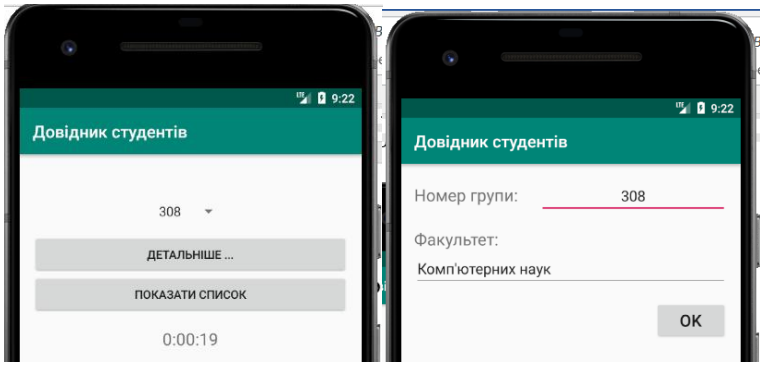



Рисунок 2.18 – Відображення даних по студентській групі

Далі розглянемо ще декілька компонентів та layout-ів. Розмістимо у макеті активності StudentsGroupActivity картинку. Для цього розмістимо файл картинки у директорії «app\src\main\res\drawable» нашого проєкту.

Далі включаємо до складу макета activity_students_group.xml (а саме на його початку) компонент ImageView (рис. 2.19).

```
8       android:padding="16dp"
9       tools:context=".StudentsGroupActivity">
10
11      <ImageView
12          android:layout_width="match_parent"
13          android:layout_height="wrap_content"
14          android:scaleType="fitXY"
15          android:src="@drawable/group"
16          android:layout_marginBottom="20dp"/>
17
18      <LinearLayout
19          android:layout_width="match_parent"
20          android:layout_height="wrap_content"
```

Рисунок 2.19 – Додавання компонента ImageView до activity_students_group.xml

На рис. 2.20 наведемо зовнішній вигляд активності після додавання зображення.

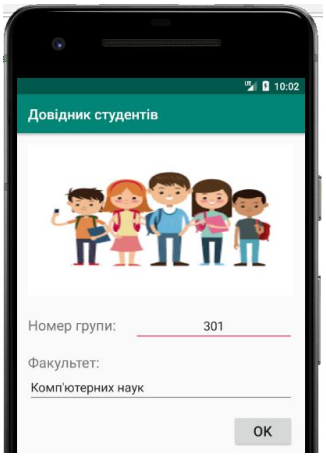



Рисунок 2.20 – Зовнішній вигляд активності StudentsGroupActivity

Наступна задача – вивести номер групи та назву факультету поверх картинки у правому верхньому та лівому нижньому куті екрана відповідно. Для цього поміщуємо ImageView всередину FrameLayout, що дозволяє виводити один елемент поверх іншого, і додаємо туди ще два текстові поля, у які код активності буде виводити групу та факультет (рис. 2.21).

```
8       android:padding="16dp"
9       tools:context=".StudentsGroupActivity">
10
11      <FrameLayout
12          android:layout_width="match_parent"
13          android:layout_height="wrap_content">
14          <ImageView
15              android:layout_width="match_parent"
16              android:layout_height="wrap_content"
17              android:scaleType="fitXY"
18              android:src="@drawable/group"
19              android:layout_marginBottom="@dimen/default_padding"/>
20
21          <TextView
22              android:id="@+id/grpNumberImageTxt"
23              android:layout_width="wrap_content"
24              android:layout_height="wrap_content"
25              android:text="Група"
26              android:layout_gravity="top|end"
27              android:textSize="@dimen/default_text_size"/>
28
```

```
        <TextView
            android:id="@+id/facultyNameImageTxt"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Факультет"
            android:layout_gravity="bottom|start"
            android:textSize="@dimen/default_text_size"
            android:layout_marginBottom="@dimen/default_padding"/>

    </FrameLayout>
```

Рисунок 2.21 – Додавання FrameLayout до activity_students_group.xml

Після додавання FrameLayout, картинки та тексту, дерево ієрархії компонент для макета activity_students_group.xml має такий вигляд (рис. 22).

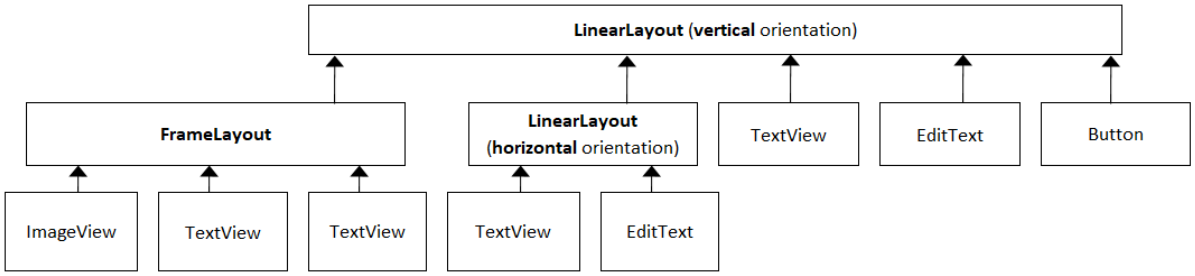



Рисунок 2.22 – Дерево ієрархії компонент для макета activity_students_group.xml

Також до методу onCreate активності StudentsGroupActivity додаємо код, що заповнить додані текстові поля значеннями (рис. 2.23).

```
        TextView txtImgGrp = (TextView) findViewById(R.id.grpNumberImageTxt);
        txtImgGrp.setText(group.getNumber());

        TextView txtImgFaculty = (TextView) findViewById(R.id.facultyNameImageTxt);
        txtImgFaculty.setText(group.getFacultyName());
    }
```

Рисунок 2.23 – Заповнення тестових полів FrameLayout значеннями

Після виконання вказаних змін, користувацький екран активності StudentsGroupActivity матиме такий вигляд (рис. 2.24).

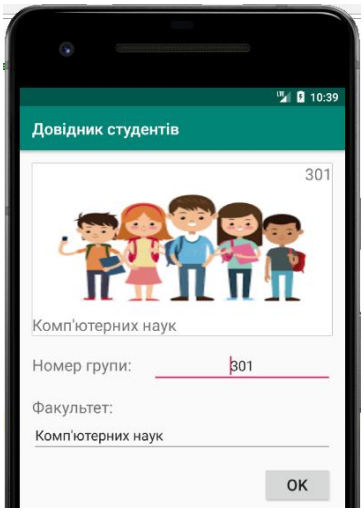



Рисунок 2.24 – Активність StudentsGroupActivity

Розглянемо ще декілька компонентів. Але перед цим додамо відповідні поля до класу StudentsGroup – це будуть рівень освіти та відомості про те, чи є пільговики та контрактники серед студентів групи. Вносимо відповідні зміни до коду класу StudentsGroup (рис. 2.25).

StudentsGroup.java

```
package com.chnulabs.students;

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

public class StudentsGroup {
    private String number;
    private String facultyName;
    private int educationLevel;
    private boolean contractExistsFlg;
    private boolean privilageExistsFlg;

    public StudentsGroup(String number, String facultyName, int educationLevel,
                         boolean contractExistsFlg, boolean privilageExistsFlg) {
        this.number = number;
        this.facultyName = facultyName;
        this.educationLevel = educationLevel;
        this.contractExistsFlg = contractExistsFlg;
        this.privilageExistsFlg = privilageExistsFlg;
    }

    public String getNumber() { return number; }

    public String getFacultyName() { return facultyName; }

    public int getEducationLevel() {
        return educationLevel;
    }

    public boolean isContractExistsFlg() {
        return contractExistsFlg;
    }

    public boolean isPrivilageExistsFlg() {
        return privilageExistsFlg;
    }

    private final static ArrayList<StudentsGroup> groups = new ArrayList<~>(
        Arrays.asList(
            new StudentsGroup( number: "301", facultyName: "Комп'ютерних наук",
                    educationLevel: 0, contractExistsFlg: true, privilageExistsFlg: false),
            new StudentsGroup( number: "302", facultyName: "Комп'ютерних наук",
                    educationLevel: 0, contractExistsFlg: true, privilageExistsFlg: false),
            new StudentsGroup( number: "308", facultyName: "Комп'ютерних наук",
                    educationLevel: 0, contractExistsFlg: true, privilageExistsFlg: true),
            new StudentsGroup( number: "309", facultyName: "Комп'ютерних наук",
                    educationLevel: 0, contractExistsFlg: true, privilageExistsFlg: false),
            new StudentsGroup( number: "501м", facultyName: "Комп'ютерних наук",
                    educationLevel: 1, contractExistsFlg: false, privilageExistsFlg: true)
        )
    );

    public static StudentsGroup getGroup(String groupNumber) {
        for(StudentsGroup g: groups) {
            if (g.getNumber().equals(groupNumber)) {
                return g;
            }
        }
        return null;
    }
}
```

Рисунок 2.25 – Змінений код класу StudentsGroup

Далі додаємо до strings.xml тексти написів, що знадобляться нам у макеті. Також до масиву students_groups додаємо ще одну групу (рис. 2.26).

```
    <string-array name="students_groups">
        <item>301</item>
        <item>302</item>
        <item>308</item>
        <item>309</item>
        <item>501м</item>
    </string-array>
    <string name="edu_level_txt">Рівень освіти: </string>
    <string name="edu_level_bachelor_txt">бакалавр</string>
    <string name="edu_level_master_txt">магістр</string>

    <string name="additional_flg_txt">Додаткові відомості: </string>
    <string name="contract_flg_txt">наявність контрактників</string>
    <string name="privilege_flg_txt">наявність пільговиків</string>
```

Рис. 2.26. Зміни у strings.xml.

У макеті activity_students_group.xml перед кнопкою ОК додаємо TableLayout, в якому будуть розміщені додані до студентської групи поля (рис. 2.27).

```
    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <TableRow
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp">
            <TextView android:text="@string/edu_level_txt"/>
            <RadioGroup>
                <RadioButton
                    android:id="@+id/edu_level_bachelor"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="бакалавр" />
                <RadioButton
                    android:id="@+id/edu_level_master"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="магістр" />
            </RadioGroup>
        </TableRow>
        <TableRow
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <TextView android:text="Додаткові відомості:"/>
            <LinearLayout android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">
                <CheckBox
                    android:id="@+id/contract_flg"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="наявність контрактників" />
                <CheckBox
                    android:id="@+id/privilege_flg"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="наявність пільговиків" />
            </LinearLayout>
        </TableRow>
    </TableLayout>
```

Рисунок 2.27 – Додавання TableLayout до activity_students_group.xml

На рис. 2.28 наведемо дерево ієрархії представлень макета activity_students_group.xml після виконання всіх вищенаведених змін.

Виконаємо необхідні зміни у класі активності StudentsGroupActivity для заповнення доданих компонент-даними із класу StudentsGroup (рис. 2.29).

Перевіримо роботу застосунку. Оберемо групу та натиснемо кнопку «Детальніше …» (рис. 2.30).

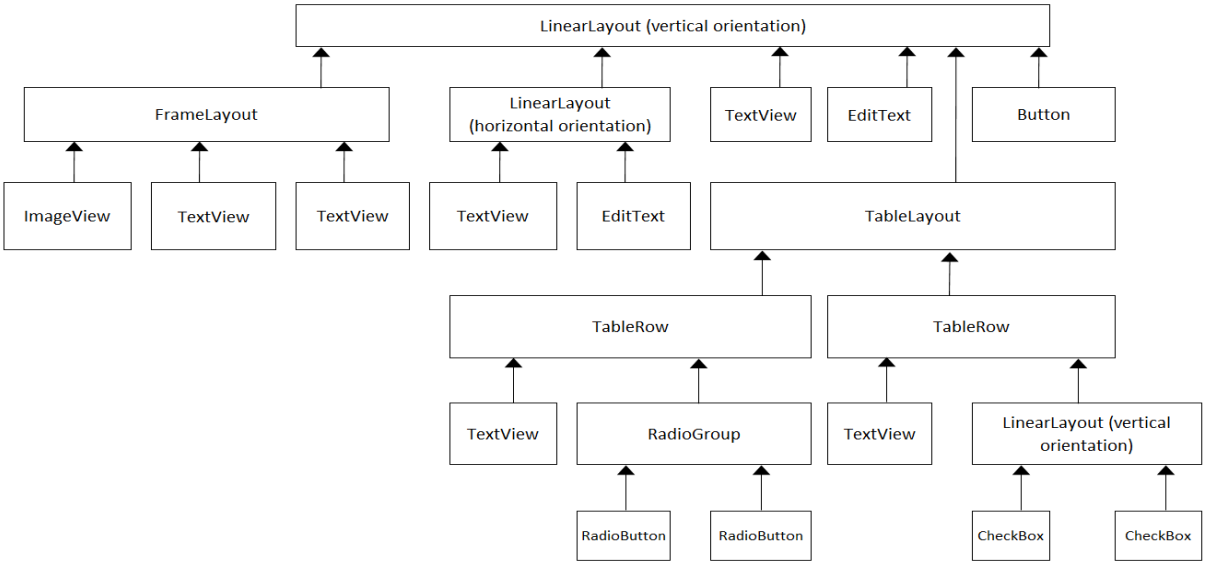



Рисунок 2.28 – Дерево ієрархії для макета activity_students_group.xml

```
38
39          if (group.getEducationLevel() == 0) {
40              ((RadioButton) findViewById(R.id.edu_level_bachelor)).setChecked(true);
41          } else {
42              ((RadioButton) findViewById(R.id.edu_level_master)).setChecked(true);
43          }
44
45          ((CheckBox) findViewById(R.id.contract_flg)).setChecked(
46                  group.isContractExistsFlg()
47          );
48
49          ((CheckBox) findViewById(R.id.privilege_flg)).setChecked(
50                  group.isPrivilageExistsFlg()
51          );
```

Рисунок 2.29 – Заповнення доданих компонент даними у методі on Create активності StudentsGroupActivity

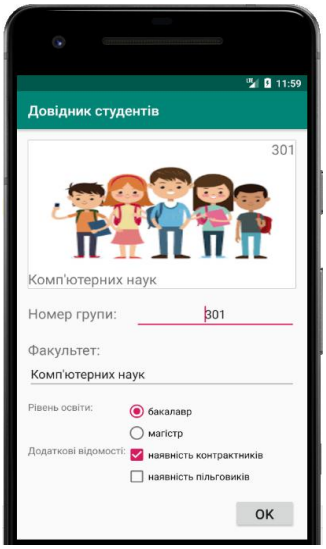



Рисунок 2.30 – Вигляд екрана активності StudentsGroupActivity

І, наостанок, по натисканню на кнопку ОК виведемо повідомлення із значеннями всіх компонент нашої активності. Для цього додаємо подію на компонент Button макету activity_students_group.xml (рис. 2.31).

```
120     <Button
121         android:id="@+id/btnOk"
122         android:layout_width="wrap_content"
123         android:layout_height="wrap_content"
124         android:text="OK"
125         android:layout_gravity="end"
126         android:layout_marginTop="16dp"
127         android:textSize="20dp"
128         android:onClick="onOkBntClick"/>
129 </LinearLayout>
```

Рисунок 2.31 – Додавання події у макет activity_students_group.xml

Реалізуємо зазначений метод onOkBntClick в активності StudentsGroupActivity (рис. 2.32).

```
56  public void onOkBntClick(View view) {
57      String outString = "Група " + ((TextView) findViewById(R.id.grpNumberEdit)).getText() + "\n";
58
59      outString += "Факультет " + ((TextView) findViewById(R.id.facultyEdit)).getText() + "\n";
60
61      if (((RadioButton) findViewById(R.id.edu_level_master)).isChecked()) {
62          outString += "рівень освіти - " + "магістр\n";
63      } else {
64          outString += "рівень освіти - " + "бакалавр\n";
65      }
66
67      if (((CheckBox) findViewById(R.id.contract_flg)).isChecked()) {
68          outString += "контрактники є\n";
69      } else {
70          outString += "контрактників нема\n";
71      }
72
73      if (((CheckBox) findViewById(R.id.privilege_flg)).isChecked()) {
74          outString += "є пільговики\n";
75      } else {
76          outString += "пільговиків нема\n";
77      }
78
79      Toast.makeText( context: this, outString, Toast.LENGTH_LONG).show();
80  }
```

Рисунок 2.32 – Реалізація методу onOkBntClick

Та переглянемо результат. Змінимо деякі поля та натиснемо ОК (рис. 2.33).

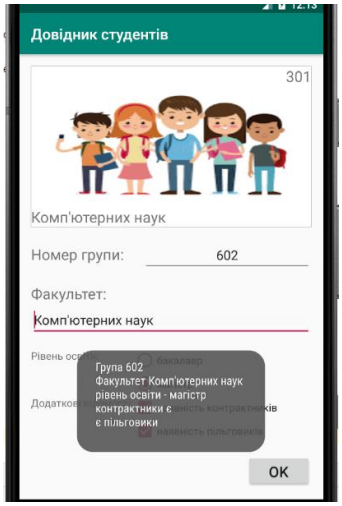



Рисунок 2.33 – Повідомлення по натисканню на кнопку ОК

## 2.2. Макети з обмеженнями

Макети з обмеженнями складніше лінійних, але при цьому вони відрізняються істотно більшою гнучкістю. Вони також значно краще підходять для складних інтерфейсів, тому що скорочення глибини ієрархії представлень означає, що Android доведеться виконувати менше обчислень під час виконання.

У макетів з обмеженнями є ще одна перевага: вони призначені для роботи з візуальним редактором середовища Android Studio. На відміну від лінійних і композиційних макетів, які зазвичай пишуться прямо в розмітці XML, макети з обмеженнями будуються у візуальному режимі. Розробник перетягує компоненти графічного інтерфейсу на панель схеми і задає інструкції щодо того, як має виводитися кожне представлення.

Виконаємо реалізацію макета activity_students_group.xml активності StudentsGroupActivity за допомогою lauout-у ConstraintLayout. Почнімо із додавання нового файлу макета у директорію layout (рис. 2.34).

У наступному вікні вводимо назву макета activity_students_group2 та кореневий елемент androidx.constraintlayout.widget.ConstraintLayout (рис. 2.35).

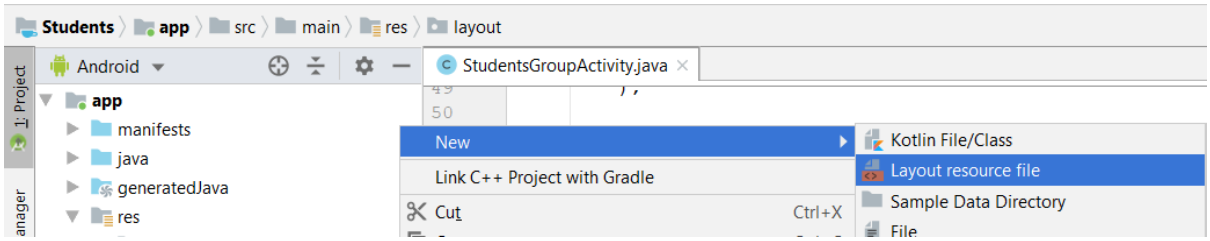



Рисунок 2.34 – Створення нового макету

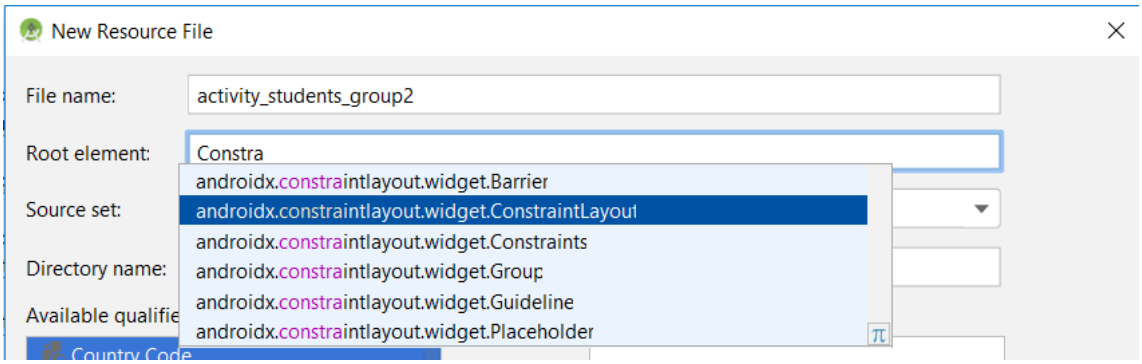



Рисунок 2.35 – Створення макету activity_students_group2.xml

Після створення макет відкривається у візуальному редакторі. Додаємо у макет компонент ImageView (рис. 2.36).

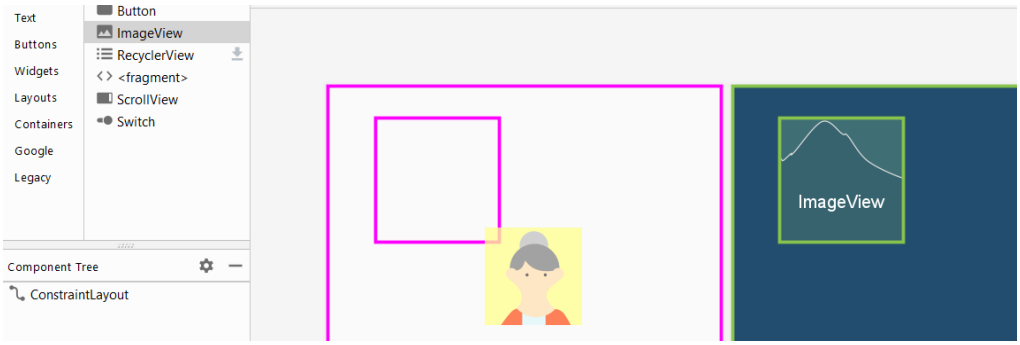



Рисунок 2.36 – Додавання до макету компоненту ImageView

У наступному вікні необхідно обрати джерело картинки. У розділі project обираємо ресурс group (рис. 2.37).

Тепер необхідно задати позицію компонента. Для цього беремо мишею лівий край компонента та тягнемо до лівого краю макета (рис. 2.38). Те ж саме робимо із верхнім та правим краєм (до верху та правого краю макета відповідно).

Після цього у вікні властивостей компонента встановимо властивість layout_width у match_parent (рис. 2.39) для того, щоб компонент розтягувався на всю ширину екрана.

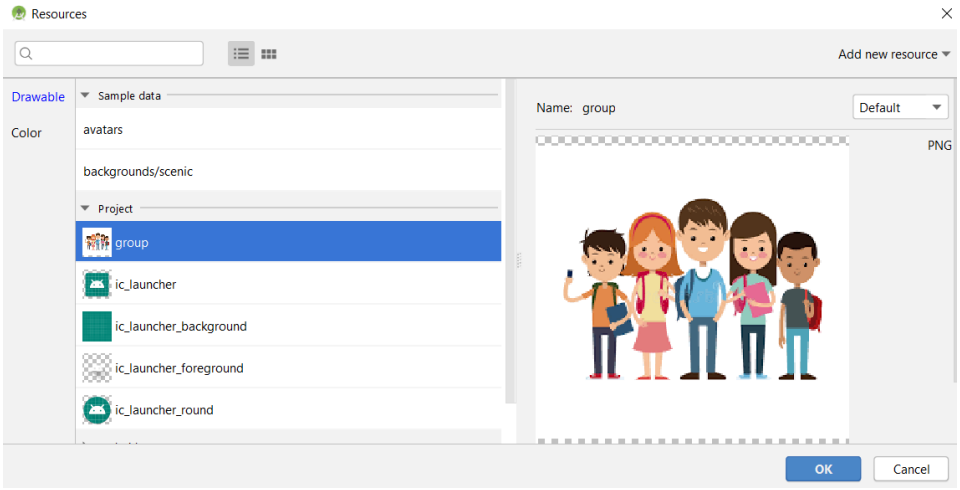



Рисунок 2.37 – Додавання зв’язку із ресурсом

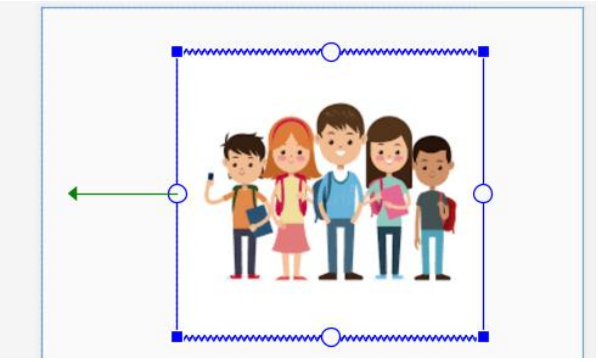


Рисунок 2.38 – Задання позиції компонента

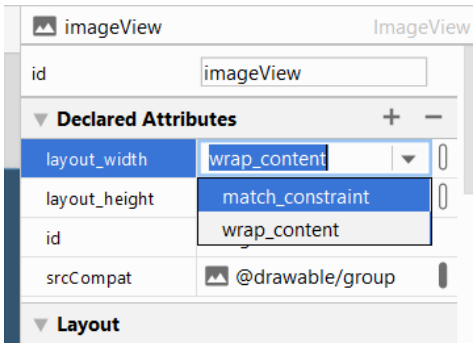



Рисунок 2.39 – Встановлення layout_width = match_parent

Щоб картинка розтягувалась по ширині, змінимо властивість scaleType = fitXY (рис. 2.40).

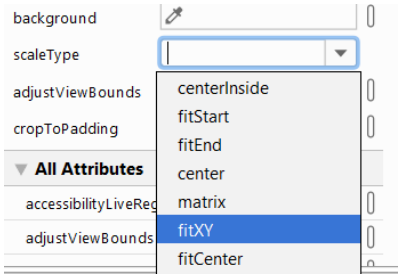



Рисунок 2.40 – Зміна властивості scaleType = fitXY

Також у вікні властивостей вкажемо margin зліва, зверху та справа – 16 (рис. 2.41).

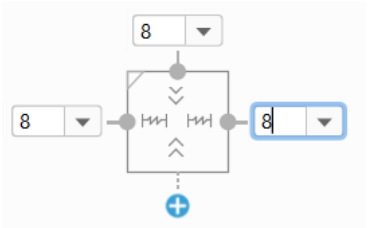



Рисунок 2.41 – Встановлення margin

Далі переходимо до розміщення напису із номером групи у правому верхньому куті зображення. Для цього перетягуємо компонент TextView у приблизне місце, де він має бути (рис. 2.42).

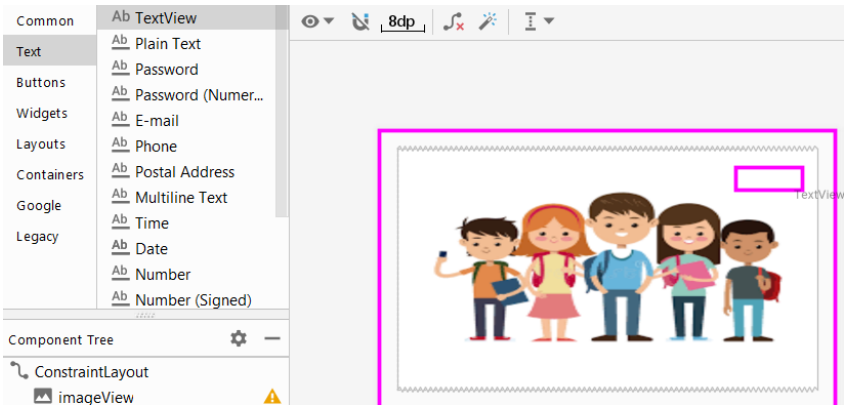



Рисунок 2.42 – Додавання компонента TextView до макету

У вікні властивостей вкажемо новому компоненту id та text (останній не обов'язково) відповідно до рис. 2.43.

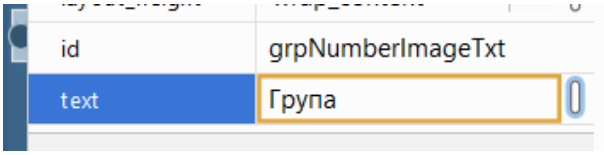



Рисунок 2.43 – Встановлення властивостей id та text

Встановимо позицію для компонента TextView аналогічно до того, як ми це робили для ImageView. Але в цьому випадку нам потрібно прив'язати верхній та правий край TextView до верхнього та правого краю елемента ImageView відповідно (рис. 2.44).

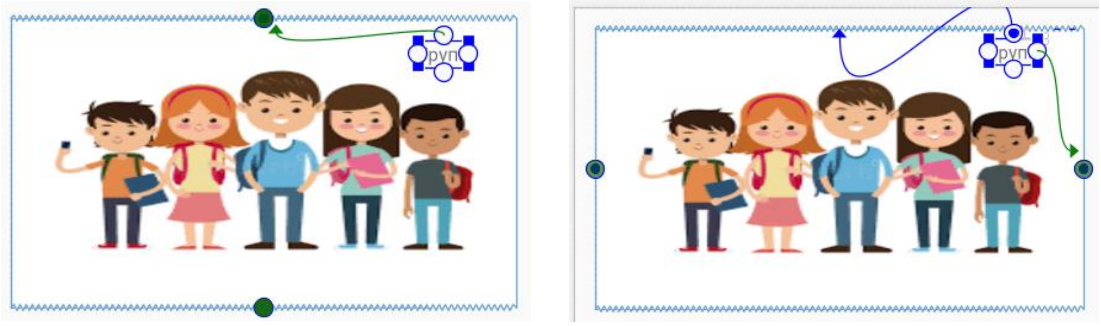


Рисунок 2.44 – Прив'язка верхнього та правого краю TextView до верхнього та правого краю елемента ImageView

Встановлюємо розмір шрифту компонента TextView. У вікні властивостей знаходимо textSize та натискаємо кнопку справа (рис. 2.45).

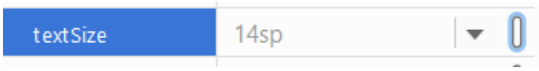



Рисунок 2.45 – Перехід до встановлення розміру шрифту

У наступному вікні (рис. 2.46) обираємо ресурс project/default_text_size. Після виконання всіх вищенаведених дій макет у візуальному редакторі має такий вигляд (рис. 2.47).

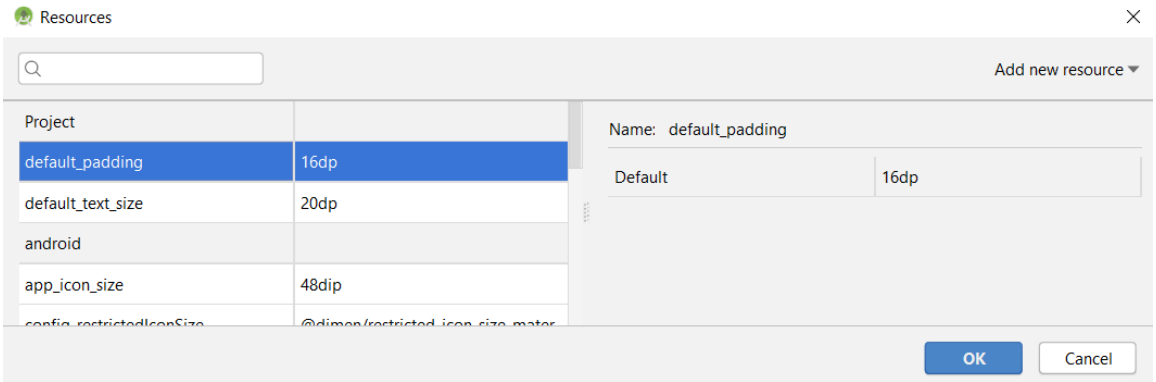



Рисунок 2.46 – Встановлення розміру шрифту

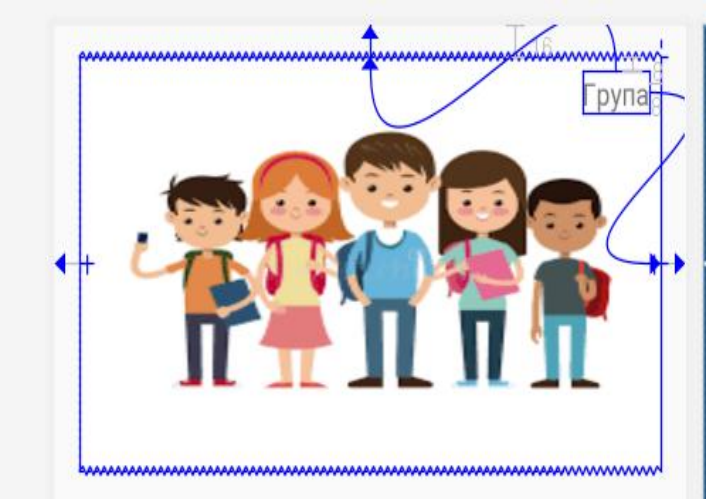



Рисунок 2.47 – Вигляд макету у візуальному редакторі

Перейшовши до редактора тексту, можемо проінспектувати отриманий код макета (рис. 2.48). Додатково пропишемо зв’язок контексту макета із активністю StudentsGroupActivity (рядок 8 на рис. 2.48).

activity_students_group2.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      tools:context=".StudentsGroupActivity">
9      <ImageView
10         android:id="@+id/imageView"
11         android:layout_width="0dp"
12         android:layout_height="wrap_content"
13         android:layout_marginStart="16dp"
14         android:layout_marginTop="16dp"
15         android:layout_marginEnd="16dp"
16         android:scaleType="fitXY"
17         app:layout_constraintEnd_toEndOf="parent"
18         app:layout_constraintHorizontal_bias="0.496"
19         app:layout_constraintStart_toStartOf="parent"
20         app:layout_constraintTop_toTopOf="parent"
21         app:srcCompat="@drawable/group" />
22
23     <TextView
24         android:id="@+id/grpNumberImageTxt"
25         android:layout_width="wrap_content"
26         android:layout_height="wrap_content"
27         android:layout_marginTop="8dp"
28         android:layout_marginEnd="8dp"
29         android:text="Група"
30         android:textSize="@dimen/default_padding"
31         app:layout_constraintEnd_toEndOf="@+id/imageView"
32         app:layout_constraintTop_toTopOf="@+id/imageView" />
33
34 </androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 2.48 – Код макета activity_students_group2.xml

Аналогічним чином додаємо TextView для назви факультету та вказуємо необхідні значення властивостей (рис. 2.49).

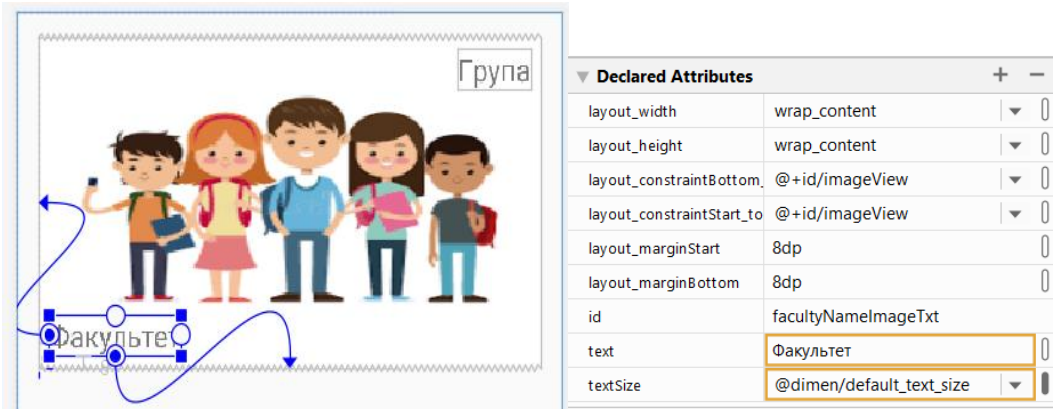



Рисунок 2.49 – Додавання TextView для назви факультету

Додаємо текст-напис «Номер групи:». Перетягуємо до макета TextView та встановлюємо йому у поле text ресурс grp_number_txt із групи project (рис. 2.50)

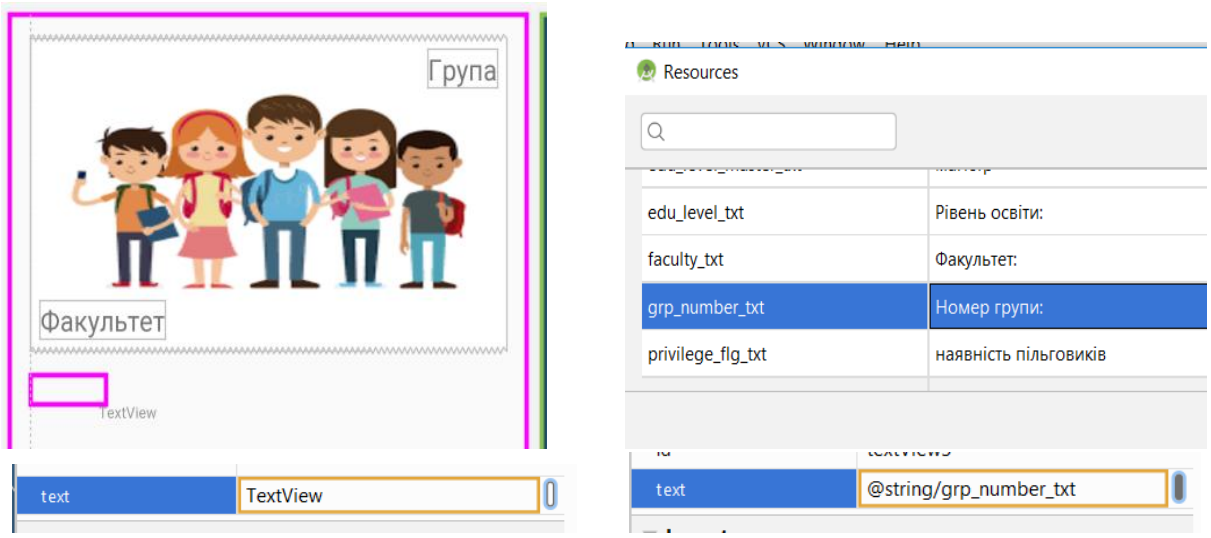



Рисунок 2.50 – Додавання напису «Номер групи:»

Тепер задамо позицію компонента на екрані. Лівий край прив’язуємо до краю макета, верхній – до низу ImageView. Аналогічно до попередніх TextView, вказуємо розмір шрифту.

Тепер вкажемо, що компонент має займати 2/5 ширини екрана. У вікні властивостей знаходимо layout_constraints, розкриваємо його та у властивість layout_constraintWidth_percent встановлюємо 0.4 (рис. 2.51).

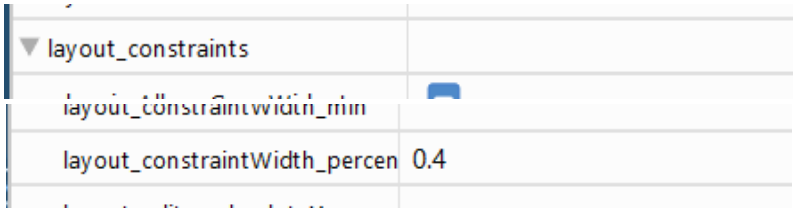



Рисунок 2.51 – Встановлення layout_constraintWidth_percent = 0.4

Додаємо компонент EditText для виводу номера обраної групи. Зв'язуємо лівий край із правим краєм, верхній та нижній із верхнім та нижнім краєм напису «Номер групи», правий край із правим краєм макета (рис. 2.52). Також вказуємо id для доступу із коду активності. Всі необхідні властивості вказані на рис. 2.53.

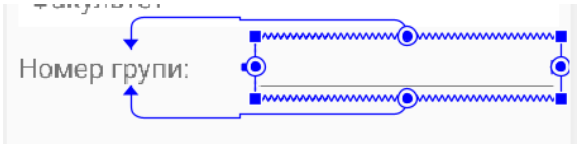



Рисунок 2.52 – Позиціонування EditText для номера групи

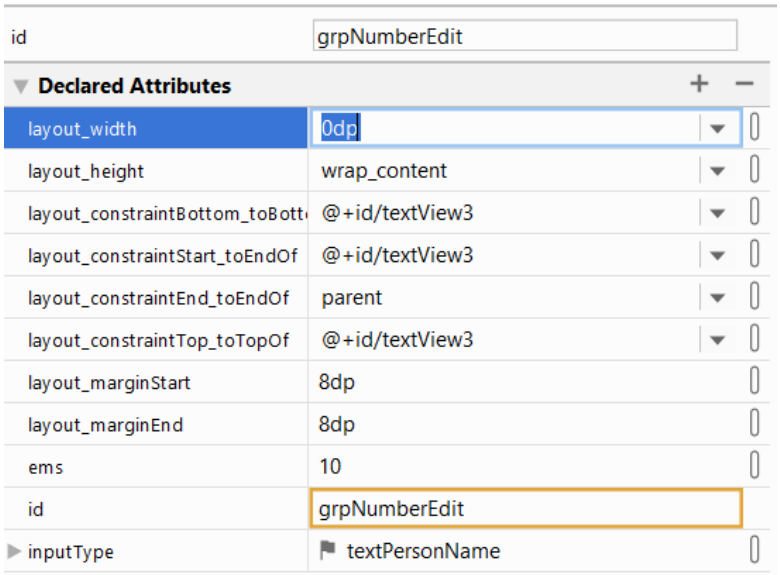



Рисунок 2.53 – Властивості компонента EditText для номера групи

Додаємо іще пару TextView та EditText для назви факультету. Позиціонування та значення властивостей для TextView показано на рис. 2.54, для EditText на рис. 2.55.

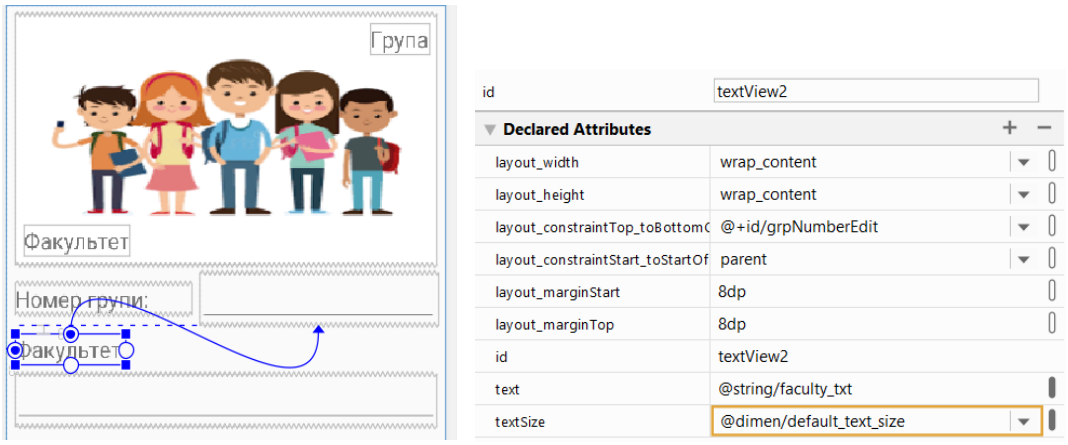



Рисунок 2.54 – Позиціонування та властивості компонента TextView

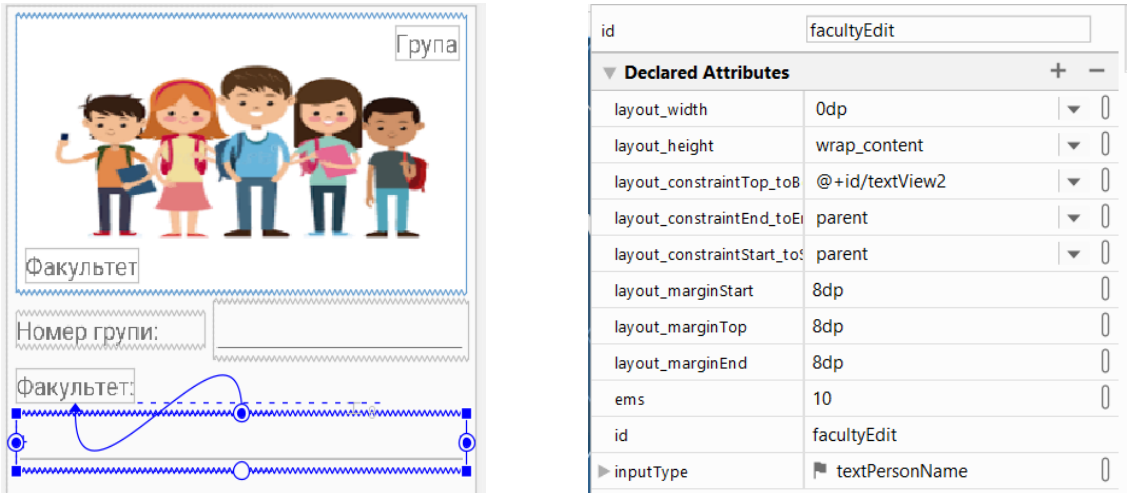



Рисунок 2.55 – Позиціонування та властивості компонента EditText

Додаємо напис «Рівень освіти» (рис. 2.56).

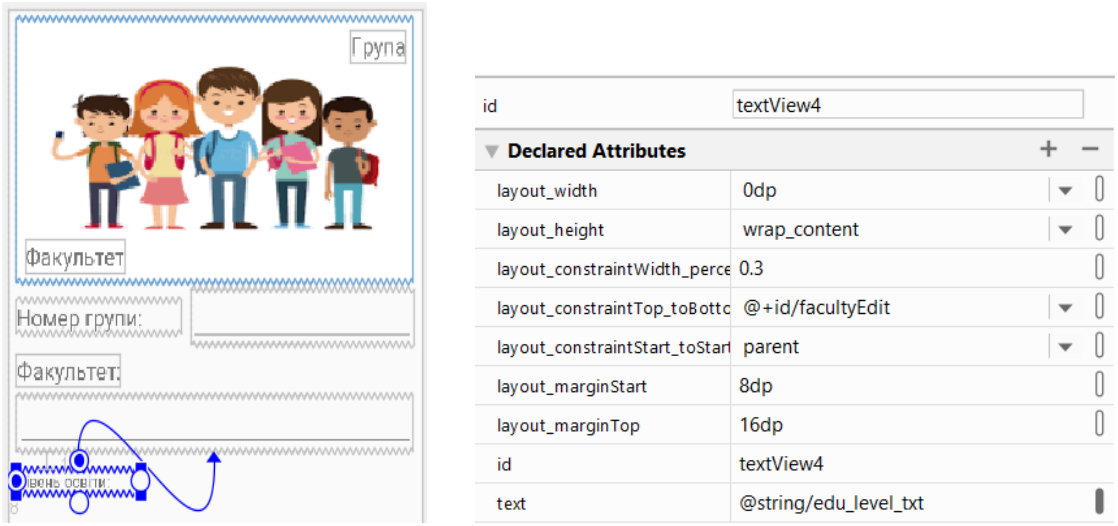



Рисунок 2.56 – Позиціонування та властивості компонента TextView «Рівень освіти»

Далі додаємо RadioGroup (рис. 2.57), а у нього два RadioButton (рис. 2.58).

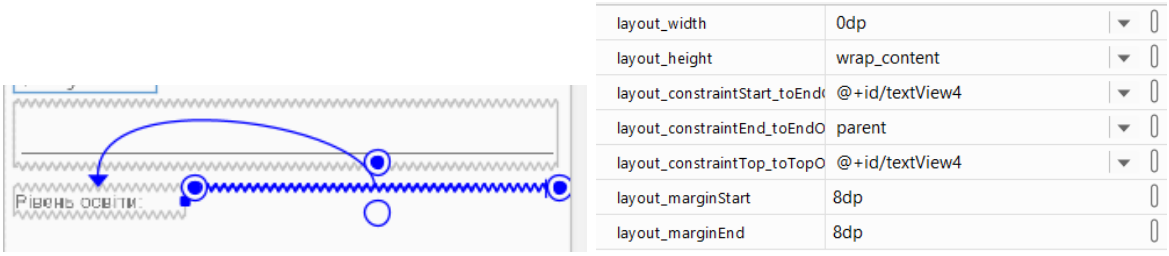



Рисунок 2.57 – Додавання компонента RadioGroup

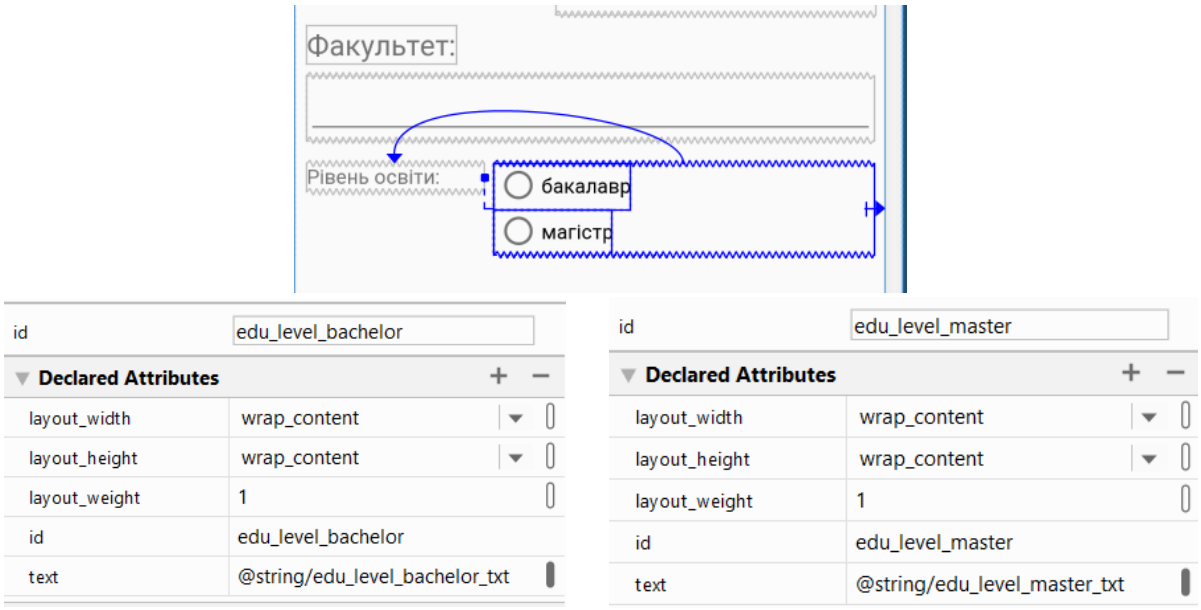



Рисунок 2.58 – Компоненти RadioButton «бакалавр» та «магістр»

Доповнюємо макет написом «додаткові відомості» (рис. 2.59) та компонентами checkbox «наявність контрактників» (рис. 2.60) і «наявність пільговиків» (рис. 2.61).

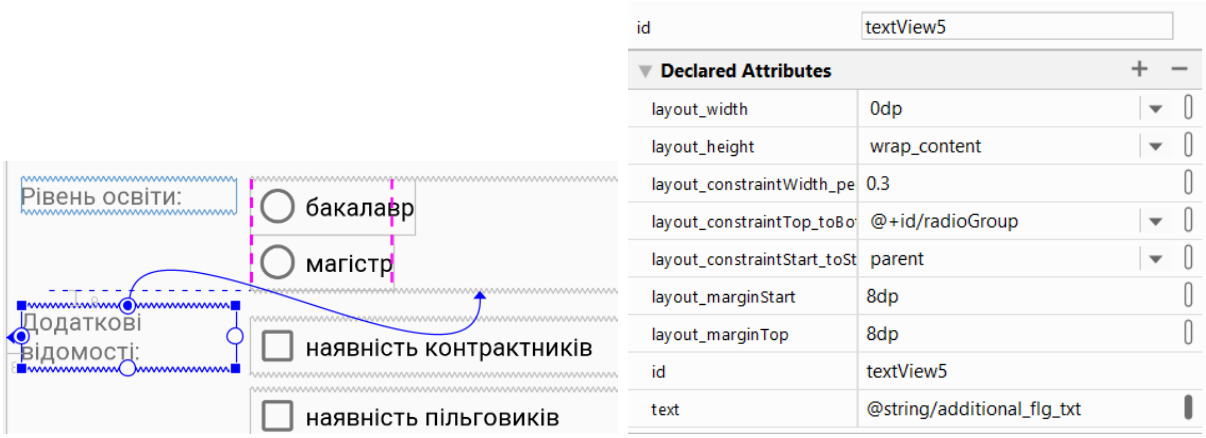



Рисунок 2.59 – TextView «додаткові відомості»

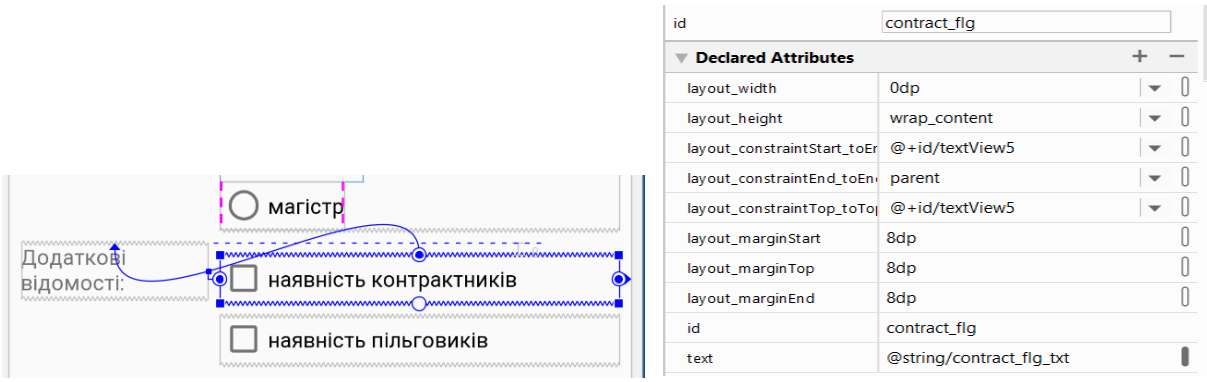



Рисунок 2.60 – CheckBox «наявність контрактників»

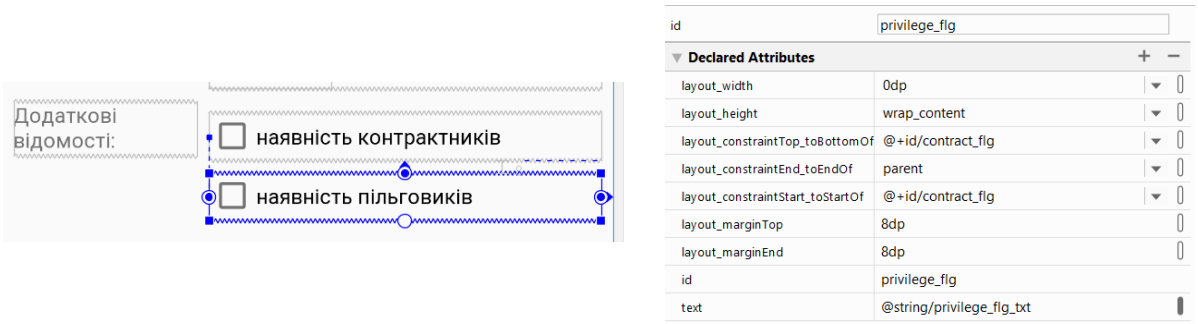



Рисунок 2.61 – CheckBox «наявність пільговиків»

На останок додаємо кнопку «ОК» (рис. 2.62).

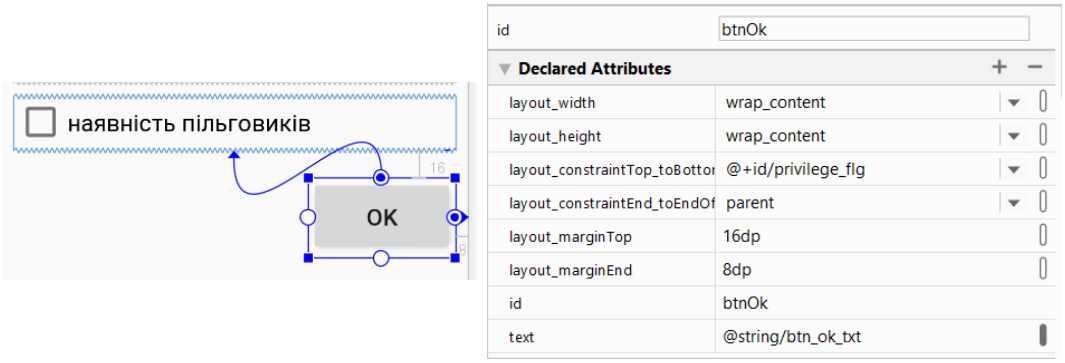



Рисунок 2.62 – Кнопка «ОК»

Також для кнопки пов'яжемо подію onClick із методом активності onOkBtnClick (рис. 2.63).

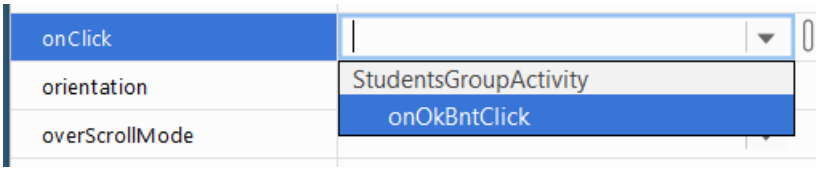



Рисунок 2.63 – Подія натискання на кнопку

Перед перевіркою нового макета необхідно в коді активності прописати, що активність під час завантаження має використовувати саме цей макет, а не створений у попередній роботі. Відкриваємо метод onCreate активності StudentsGroupActivity та змінюємо «R.layout.activity_students_group» на «R.layout.activity_students_group2» (рис. 2.64).

```
15  public class StudentsGroupActivity extends AppCompatActivity {
16
17      public static final String GROUP_NUMBER = "groupnumber";
18
19      @Override
20      protected void onCreate(Bundle savedInstanceState) {
21          super.onCreate(savedInstanceState);
22          setContentView(R.layout.activity_students_group2);
23
```

Рисунок 2.64 – Зміна макета для активності StudentsGroupActivity

Для того, щоб старий та новий макет мали відмінності, перемістимо напис на картинці із назвою факультету із нижнього лівого у нижній правий кут. Для цього видаляємо прив’язку до лівого краю, натиснувши на неї мишею (рис. 2.65), та виконуємо прив’язку до правого краю (рис. 2.66).

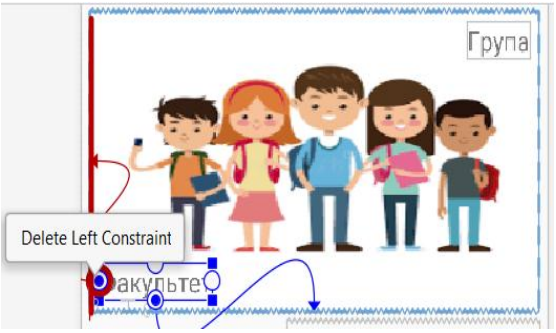



Рисунок 2.65 – Видалення прив’язки напису назви факультету до лівого краю


Рисунок 2.66 – Прив’язка до правого краю

Запускаємо застосунок та перевіряємо роботу нового макета (рис. 2.67).

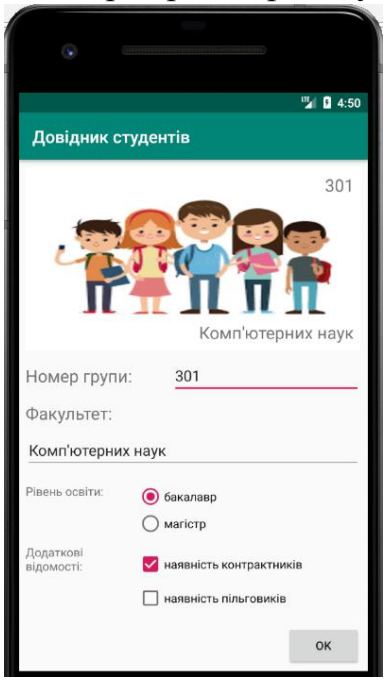



Рисунок 2.67 – Перевірка роботи нового макета

Перевіримо, що подія натискання на кнопку також працює (рис. 2.68).

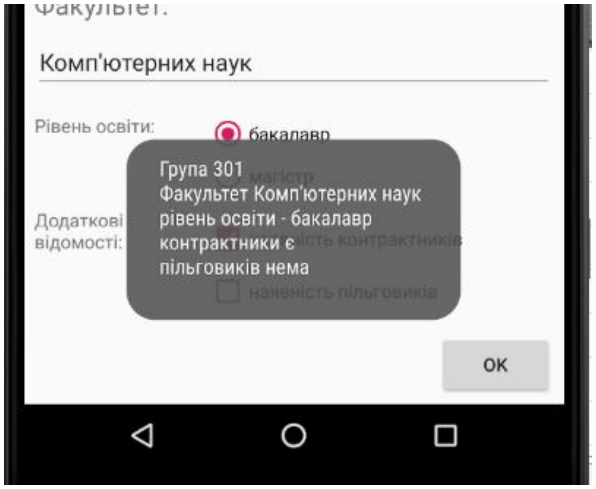



Рисунок 2.68 – Повідомлення по натисканню на кнопку

Нижче наведемо повний код нового макета activity_students_group2. xml (рис. 2.69).

activity_students_group2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".StudentsGroupActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:scaleType="fitXY"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.496"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/group" />

    <TextView
        android:id="@+id/grpNumberImageTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:text="Група"
        android:textSize="20dp"
        app:layout_constraintEnd_toEndOf="@+id/imageView"
        app:layout_constraintTop_toTopOf="@+id/imageView" />

    <TextView
        android:id="@+id/facultyNameImageTxt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:text="Факультет"
        android:textSize="20dp"
        app:layout_constraintBottom_toBottomOf="@+id/imageView"
        app:layout_constraintEnd_toEndOf="@+id/imageView" />

```

```
    <TextView
        android:id="@+id/textView3"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="16dp"
        android:text="@string/grp_number_txt"
        android:textSize="@dimen/default_text_size"
        app:layout_constrainedWidth="false"
        app:layout_constraintDimensionRatio=""
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView"
        app:layout_constraintWidth_percent="0.4" />

    <EditText
        android:id="@+id/grpNumberEdit"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toBottomOf="@+id/textView3"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textView3"
        app:layout_constraintTop_toTopOf="@+id/textView3" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="@string/faculty_txt"
        android:textSize="@dimen/default_text_size"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/grpNumberEdit" />

    <EditText
        android:id="@+id/facultyEdit"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="16dp"
        android:text="@string/edu_level_txt"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/facultyEdit"
        app:layout_constraintWidth_percent="0.3" />

    <RadioGroup
        android:id="@+id/radioGroup"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textView4"
        app:layout_constraintTop_toTopOf="@+id/textView4">

        <RadioButton
            android:id="@+id/edu_level_bachelor"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="бакалавр" />
```

```
        <RadioButton
            android:id="@+id/edu_level_master"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="магістр" />
    </RadioGroup>

    <TextView
        android:id="@+id/textView5"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="Додаткові відомості:"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/radioGroup"
        app:layout_constraintWidth_percent="0.3" />

    <CheckBox
        android:id="@+id/contract_flg"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:text="наявність контрактників"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textView5"
        app:layout_constraintTop_toTopOf="@+id/textView5" />

    <CheckBox
        android:id="@+id/privilege_flg"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:text="наявність пільговиків"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="@+id/contract_flg"
        app:layout_constraintTop_toBottomOf="@+id/contract_flg" />

    <Button
        android:id="@+id/btnOk"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="8dp"
        android:onClick="onOkBntClick"
        android:text="OK"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/privilege_flg" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 2.69 – Повний код нового макета activity_students_group2.xml

## 2.3. Організація застосунків. Спискові представлення

Коли розробник задумує новий застосунок, у нього зазвичай вже є маса ідей щодо того, що повинно бути в цьому застосунку. Безумовно, всі ці ідеї будуть корисні для користувача. Але як побудувати з них інтуїтивно зрозумілий, добре організований застосунок? Корисний спосіб упорядкування таких ідей полягає в їх класифікації на три типи активностей:

- активності верхнього рівня;
- активності категорій;
- активності деталізації / редагування.

Активності верхнього рівня представляють операції, найбільш важливі для користувача, і надають прості засоби для навігації по них. У більшості

застосунків перша активність, яку бачить користувач, є активністю верхнього рівня.

Активності категорій виводять дані, що належать конкретній категорії, – часто у вигляді списку. Такі активності часто допомагають користувачеві перейти до активностей деталізації/редагування. Приклад активності категорії – відображення списку студентських груп.

Активності деталізації/редагування виводять докладну інформацію по конкретному запису, надають користувачеві можливість редагування існуючих записів або введення нових записів. Приклад активності деталізації/редагування – активність, яка виводить детальну інформацію про конкретну студентську групу.

Після того, як ви розділите свої ідеї на активності верхнього рівня, категорій та деталізації/редагування, ця класифікація може використовуватися для планування навігації у застосунку. Як правило, перехід від активностей верхнього рівня до активностей деталізації/редагування повинен здійснюватися через активності категорій.

Для прикладу уявимо ситуацію, що користувач хоче переглянути детальну інформацію по конкретній студентській групі. Для цього він запускає застосунок та обирає команду переходу до списку студентських груп. Далі для відображення даних по конкретній групі користувач обирає її у списку.

У застосунках з такою структурою необхідно організувати навігацію, тобто переходи між активностями. У таких ситуаціях найчастіше застосовуються спискові представленнях. Спискові представлення відображають перелік об’єктів даних, які потім використовується для навігації за застосунком.

Почнімо перетворення нашого застосунку. Для початку змінимо макет activity_main.xml головної активності, що буде відповідати активності верхнього рівня. Прибираємо звідти всі компоненти разом із лінійним layout-ом та поміщуємо ConstraintLayout у якості кореневого елемента (рис. 2.70).

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

</androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 2.70 – Зміна лейауту на ConstraintLayout
в макеті головної активності

У strings.xml додаємо текст для напису на кнопці активності верхнього рівня для завантаження активності категорій списку студентських груп (рис. 2.71).

```
26
27      <string name="btn_groups_txt">Студентські групи</string>
```

Рисунок 2.71 – Додавання даних до strings.xml

Тепер до макета головної активності додаємо ImageView та кнопку для переходу до майбутньої активності категорій – списку студентських груп (рис. 2.72).

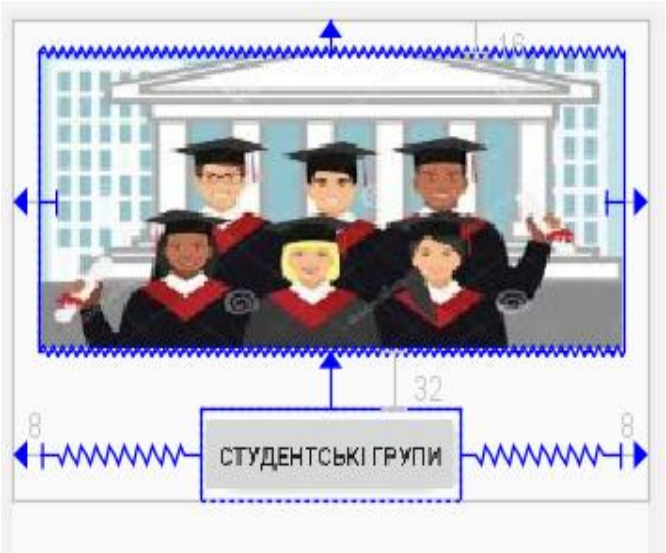



Рисунок 2.72. Активність верхнього рівня

Перелік необхідних властивостей для ImageView та Button наведено на рис. 2.73 та рис. 2.74 відповідно.

| id | imageView2 |
|---|---|
| **Declared Attributes** | |
| layout_width | 0dp |
| layout_height | wrap_content |
| layout_constraintEnd_toEnd | parent |
| layout_constraintStart_toSta | parent |
| layout_constraintTop_toTop | parent |
| layout_marginStart | 16dp |
| layout_marginTop | 16dp |
| layout_marginEnd | 16dp |
| id | imageView2 |
| scaleType | fitXY |
| srcCompat | @drawable/univer |

Рисунок 2.73. Властивості ImageView

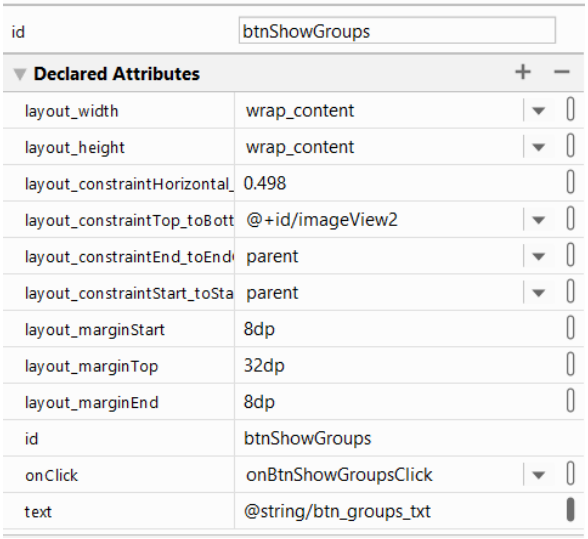



Рисунок 2.74 – Властивості Button

Додаємо нову активність категорій GroupsListActivity New–>Activity –>Empty activity (рис. 2.75).

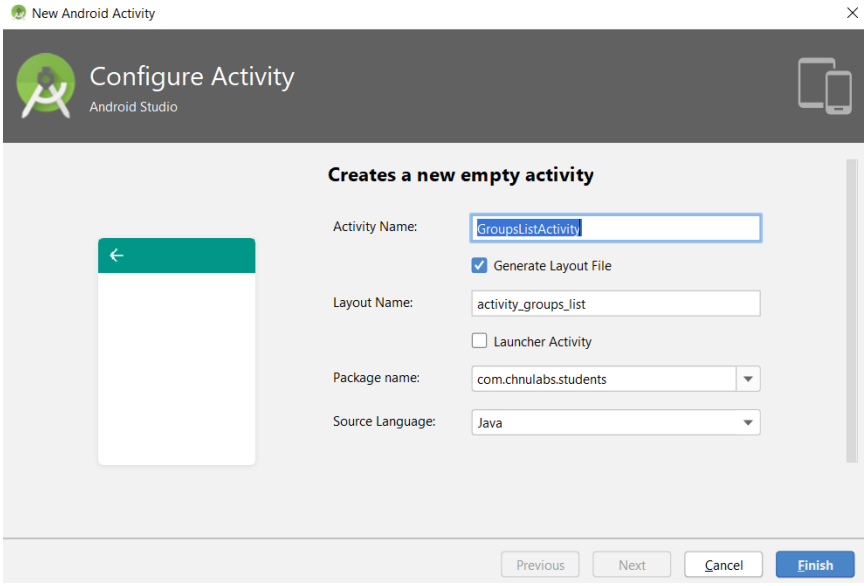



Рисунок 2.75 – Додавання активності категорій GroupsListActivity

У візуальному редакторі перетягуємо на створений макет компонент ListView (рис. 2.76).

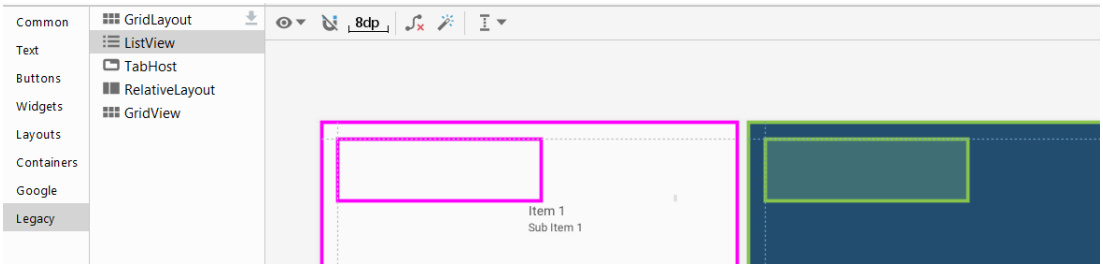



Рисунок 2.76 – Додавання компонента ListView

Найпростіший спосіб додавання даних до списку ListView – через присвоєння властивості android:entries масиву рядків.


Рисунок 2.77 – Встановлення властивості android:entries для ListView

Повний перелік властивостей компонента ListView наведено на рис. 2.78.

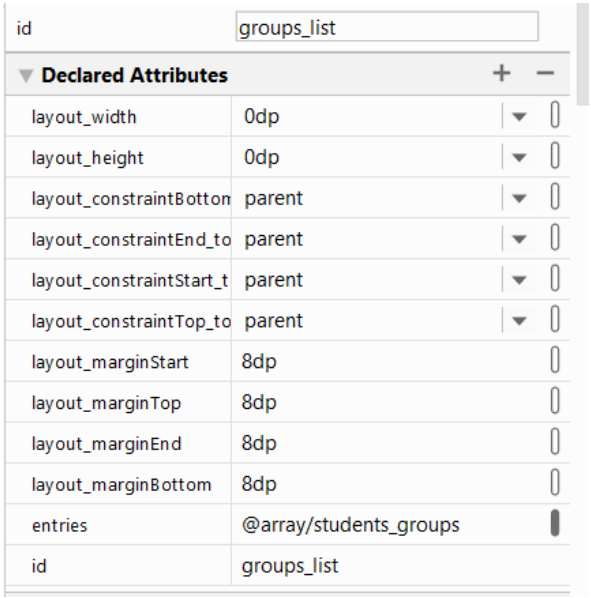
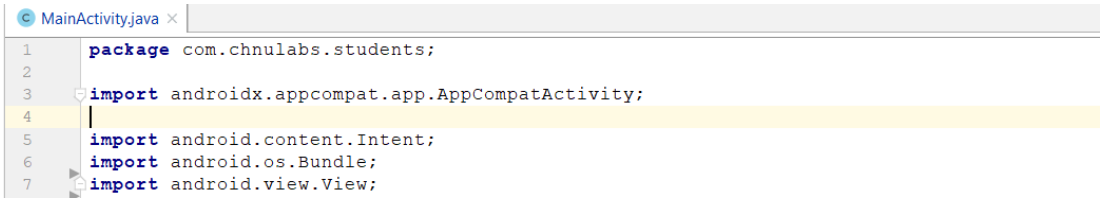



Рисунок 2.78 – Властивості компонента ListView

Пов’яжемо створену активність із головною активністю верхнього рівня. Для цього перейдемо до коду активності MainActivity, видалимо код, що залишився із попередньої роботи, та замінимо його таким (рис. 2.79).



```
8
9   public class MainActivity extends AppCompatActivity {
10
11      @Override
12      protected void onCreate(Bundle savedInstanceState) {
13          super.onCreate(savedInstanceState);
14          setContentView(R.layout.activity_main);
15      }
16
17      public void onBtnShowGroupsClick(View view) {
18          Intent intent = new Intent( packageContext: this, GroupsListActivity.class);
19          startActivity(intent);
20      }
21  }
```

Рисунок 2.79 – Новий код активності MainActivity

Перевірка роботи застосунку (рис. 2.80).

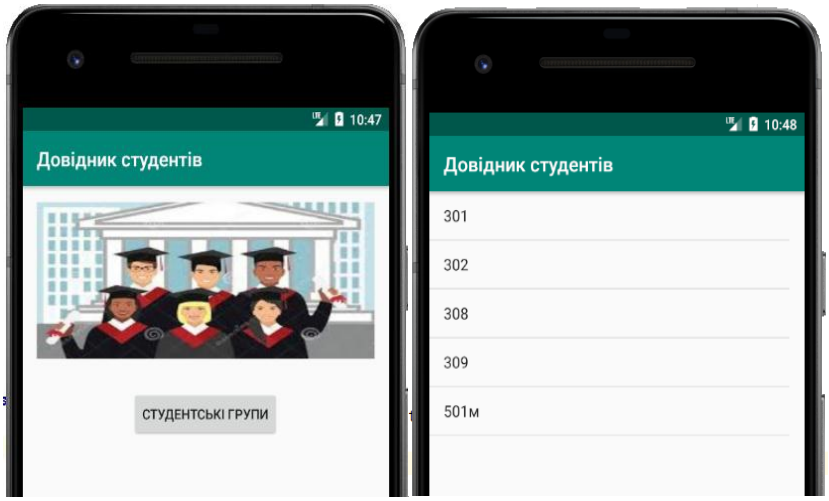



Рисунок 2.80 – Активність верхнього рівня та активність категорій

Щоб пункти списку реагували на клацання, слід реалізувати слухача подій. Слухач подій відстежує події, що відбуваються в застосунку, наприклад, клацання на представленнях, втрату або отримання ними фокуса або натискання фізичної клавіші на пристрої. Реалізація слухача подій дозволить вам виявляти конкретні дії користувача – скажімо, клацання на варіантах списку – і реагувати на них.

Навіщо створювати слухача подій, а не скористатися атрибутом android: onClick в розмітці? Атрибут android:onClick в макетах може використовуватися тільки для кнопок або будь-яких представлень, які є субкласами Button, наприклад CheckBox або RadioButton. Клас ListView не є субкласом Button, тому рішення з атрибутом android:onClick не працює. Саме тому доводиться створювати свою реалізацію слухача.

Якщо ви хочете, щоб елементи списку реагували на клацання, створіть об’єкт OnItemClickListener і реалізуйте його метод onItemClick(). Слухач OnItemClickListener відстежує клацання на елементах списку, а метод onItemClick() визначає реакцію активності на клацання. За параметрами, які передаються методу onItemClick(), можна отримати

додаткову інформацію про подію – наприклад, отримати посилання на елемент зі списку, дізнатися його позицію в списковому представленні (починаючи з 0) та ідентифікатор запису використовуваного набору даних.

Створимо OnItemClickListener для відстеження натискання на елементі списку студентських груп активності GroupsListActivity (рис. 2.81).

GroupsListActivity.java

```
package com.chnulabs.students;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;

public class GroupsListActivity extends AppCompatActivity {

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

        AdapterView.OnItemClickListener listener = new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
                String group = (String) adapterView.getItemAtPosition(i);
                Intent intent = new Intent( packageContext: GroupsListActivity.this,
                        StudentsGroupActivity.class);
                intent.putExtra(StudentsGroupActivity.GROUP_NUMBER, group);
                startActivity(intent);
            }
        };
    }
}
```

Рисунок 2.81 – Створення слухача для ListView

Наш слухач під час натискання на елемент списку отримуватиме його значення та передаватиме його до інтенту запуску активності StudentsGroupActivity за допомогою putExtra. Залишається пов’язати створеного слухача із компонентом ListView. Зробимо це (рис. 2.82).

```
                Intent intent = new Intent( packageContext: GroupsListActivity.this,
                        StudentsGroupActivity.class);
                intent.putExtra(StudentsGroupActivity.GROUP_NUMBER, group);
                startActivity(intent);
            }
        };

        ListView listView = (ListView) findViewById(R.id.groups_list);
        listView.setOnItemClickListener(listener);
    }
}
```

Рисунок 2.82 – Зв’язок ListView із створеним слухачем

Виконуємо перевірку роботи застосунку. Переходимо до активності списку груп та натискаємо на одному із елементів списку (рис. 2.83).

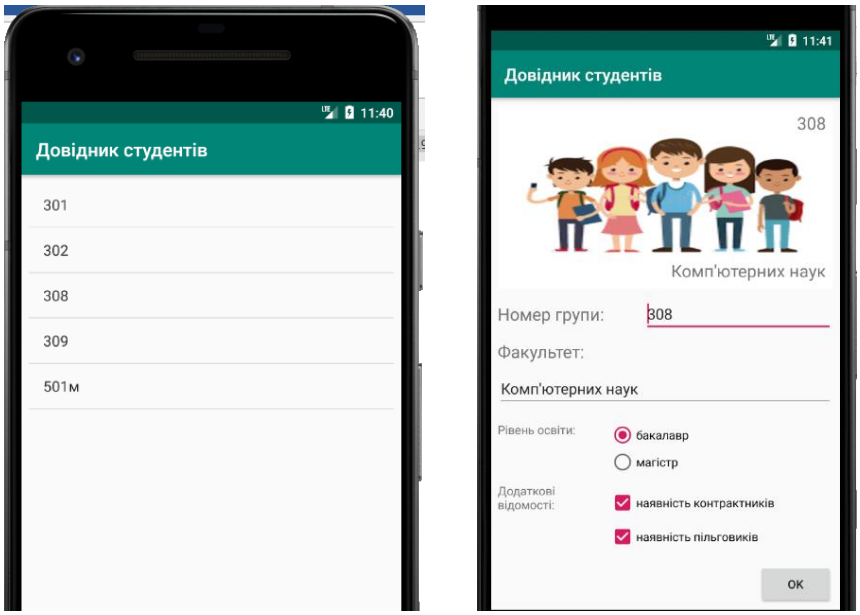



Рисунок 2.83 – Перевірка події натискання на елементі списку

Додавати елементи у список можна не обов'язково із масиву рядків. Як ми пам'ятаємо, у нас реалізований клас StudentsGroup, у якому представлена статична колекція елементів типу StudentsGroup – тобто список спудентських груп. Скористаймося ним.

Але якщо дані спискового представлення повинні надходити з джерела, відмінного від strings.xml (наприклад, з масиву Java або бази даних), необхідно використовувати адаптер. Адаптер грає роль моста між джерелом даних і списковим представленням.

У нашому випадку використаємо адаптер масивів. Щоб використовувати адаптер масиву, слід виконати його форматування і приєднати до спискового представлення. Під час ініціалізації адаптера масиву спочатку вказуємо тип даних масиву, який хочемо зв'язати зі списковим представленням. Потім адаптеру передаються три параметри: Context (зазвичай поточна активність), ресурс макета, який визначає, як повинен відображатися кожен елемент з масиву, і сам масив (або колекція).

На рис. 2.84 наведемо реалізацію адаптеру для студентських груп у методі onCreate активності GroupsListActivity.

```
33          ArrayAdapter<StudentsGroup> adapter = new ArrayAdapter<StudentsGroup>(
34                  context: this,
35                  android.R.layout.simple_list_item_1,
36                  StudentsGroup.getGroups()
37          );
38          listView.setAdapter(adapter);
```

Рисунок 2.84 – Адаптер для відображення колекції студентських груп

У класі StudentsGroup реалізуємо гетер для колекції груп groups та метод toString для строкового представлення елементів класу (рис. 2.85).

```
65
66 @     public static ArrayList<StudentsGroup> getGroups() {
67          return groups;
68      }
69
70      @Override
71      public String toString() {
72          return number;
73      }
```

Рисунок 2.85 – Зміни у класі StudentsGroup

У макеті activity_groups_list.xml активності видаляємо прив'язку ListView до масиву рядків файлу strings.xml (рис. 2.86).

```
9       <ListView
10          android:id="@+id/groups_list"
11          android:layout_width="0dp"
12          android:layout_height="0dp"
13          android:layout_marginStart="8dp"
14          android:layout_marginTop="8dp"
15          android:layout_marginEnd="8dp"
16          android:layout_marginBottom="8dp"
17          android:entries="@array/students_groups"
18          app:layout_constraintBottom_toBottomOf="parent"
19          app:layout_constraintEnd_toEndOf="parent"
20          app:layout_constraintStart_toStartOf="parent"
21          app:layout_constraintTop_toTopOf="parent" />
22  </androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 2.86 – Видалення прив'язки до strings.xml

Після виконаних змін можна видалити масив рядків у файлі strings.xml (рис. 2.87).

```
3       <string name="default_text">Список студентів</string>
4       <string name="btn_text">Показати список</string>
5       <string-array name="students_groups">
6           <item>301</item>
7           <item>302</item>
8           <item>308</item>
9           <item>309</item>
10          <item>501м</item>
11      </string-array>
12      <string name="btn_send">Відправити повідомленням</string>
13      <string name="btn_grp_detail_txt">Детальніше ...</string>
```

Рисунок 2.87 – Видалення непотрібних даних в strings.xml

Перевіримо роботу застосунку (рис. 2.88). Для відображення якихось змін на екрані після попереднього запуску, можемо попередньо додати або видалити якусь групу із колекції у класі StudentsGroup (рис. 2.89).

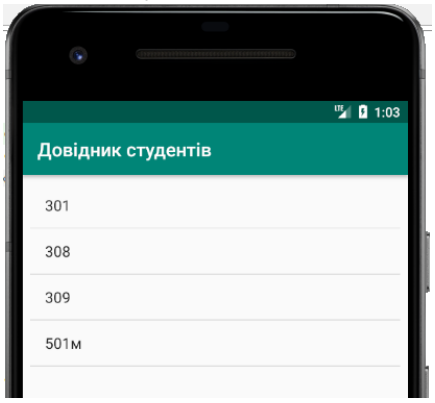



Рисунок 2.88 – Перевірка роботи застосунку після використання адаптера

```
42      private final static ArrayList<StudentsGroup> groups = new ArrayList<~>(
43          Arrays.asList(
44              new StudentsGroup( number: "301", facultyName: "Комп'ютерних наук",
45                      educationLevel: 0, contractExistsFlg: true, privilageExistsFlg: false),
46      //          new StudentsGroup( "302", "Комп'ютерних наук",
47      //                  0, true, false),
48              new StudentsGroup( number: "308", facultyName: "Комп'ютерних наук",
49                      educationLevel: 0, contractExistsFlg: true, privilageExistsFlg: true),
50              new StudentsGroup( number: "309", facultyName: "Комп'ютерних наук",
51                      educationLevel: 0, contractExistsFlg: true, privilageExistsFlg: false),
52              new StudentsGroup( number: "501м", facultyName: "Комп'ютерних наук",
53                      educationLevel: 1, contractExistsFlg: false, privilageExistsFlg: true)
54          )
55      );
56
```

Рисунок 2.89 – Зміна списку студентських груп

Виконаємо перехід від активності студентської групи StudentsGroup Activity до активності списку студентів StudentsListActivity та відображення списку студентів також у вигляді списку ListView. Для початку додаємо кнопку до макета activity_students_group2.xml активності Students GroupActivity, за якою будемо переходити до StudentsListActivity (рис. 2.90).

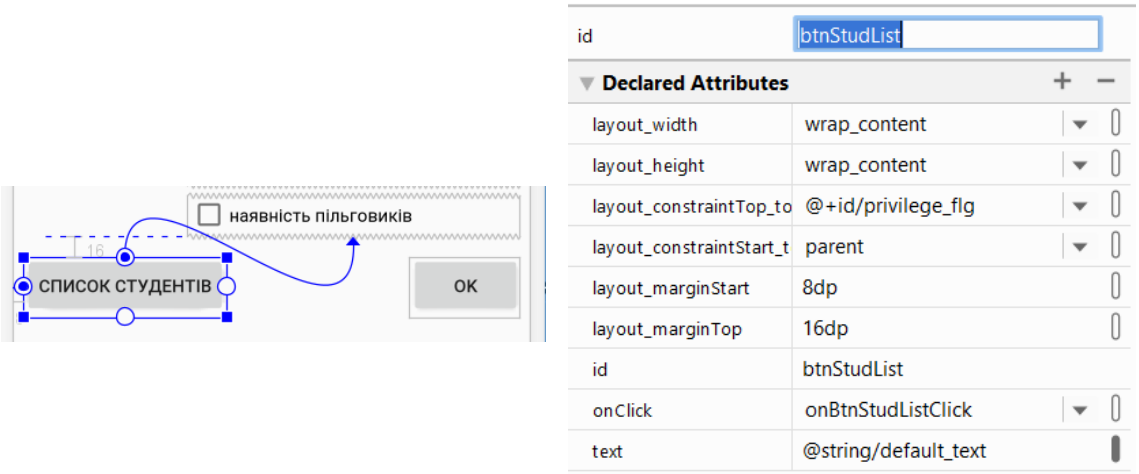



Рисунок 2.90 – Додавання кнопки до активності StudentsGroupActivity

До коду активності StudentsGroupActivity додаємо метод для натискання на кнопку onBtnStudListClick, в якому через інтент запускатимемо активність списку студентів (рис. 2.91).

```
80
81      public void onBtnStudListClick(View view) {
82          Intent localIntent = getIntent();
83          String group = localIntent.getStringExtra(GROUP_NUMBER);
84
85          Intent newIntent = new Intent( packageContext: this, StudentsListActivity.class);
86          newIntent.putExtra(StudentsListActivity.GROUP_NUMBER, group);
87          startActivity(newIntent);
88      }
```

Рис. 2.91. Реалізація методу onBtnStudListClick

Перевіримо роботу застосунку. Під час спроби переходу зі списку груп до конкретної групи маємо помилку (рис. 2.92).

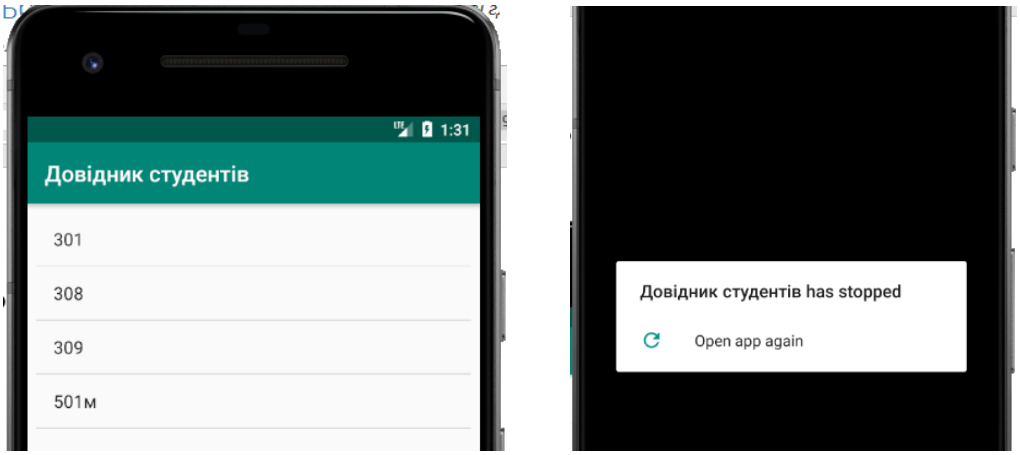



Рис. 2.92. Помилка під час переходу до активності студентської групи

Перейшовши до вкладки Run в AndroidStudio знаходимо нашу помилку та за посиланням переходимо до місця її виникнення у коді (рис. 2.93–2.94).

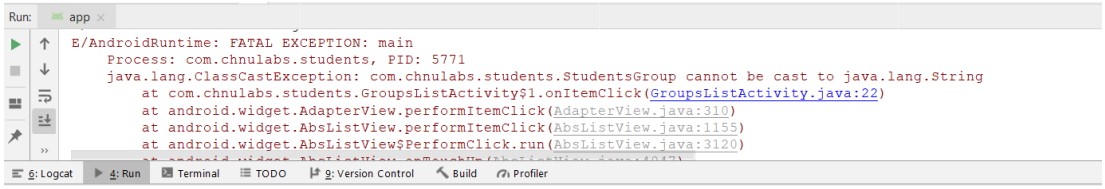



Рисунок 2.93 – Пошук помилки

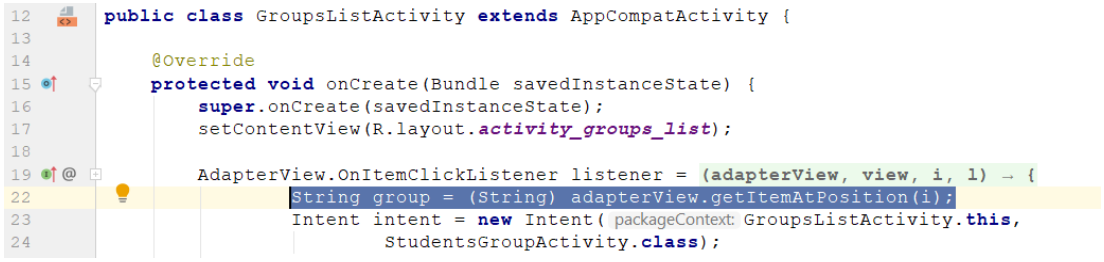



Рисунок 2.94 – Місце виникнення помилки

Маємо помилку «com.chnulabs.students.StudentsGroup cannot be cast to java.lang.String», яка виникла внаслідок того, що раніше ми брали список груп із масиву рядків, а тепер із колекції StudentsGroup, і відповідно adapterView.getItemAtPosition(i) повертає тепер не String, а StudentsGroup. Виправимо цю помилку (рис. 2.95).

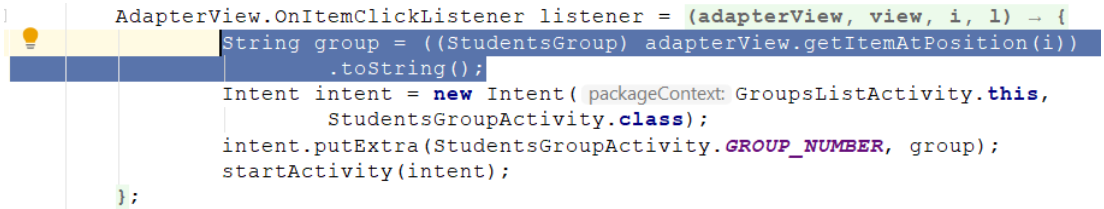



Рисунок 2.95 – Виправлення помилки перетворення типів

Тепер у активності студентської групи натиснемо на кнопку «Список студентів» (рис. 2.96).

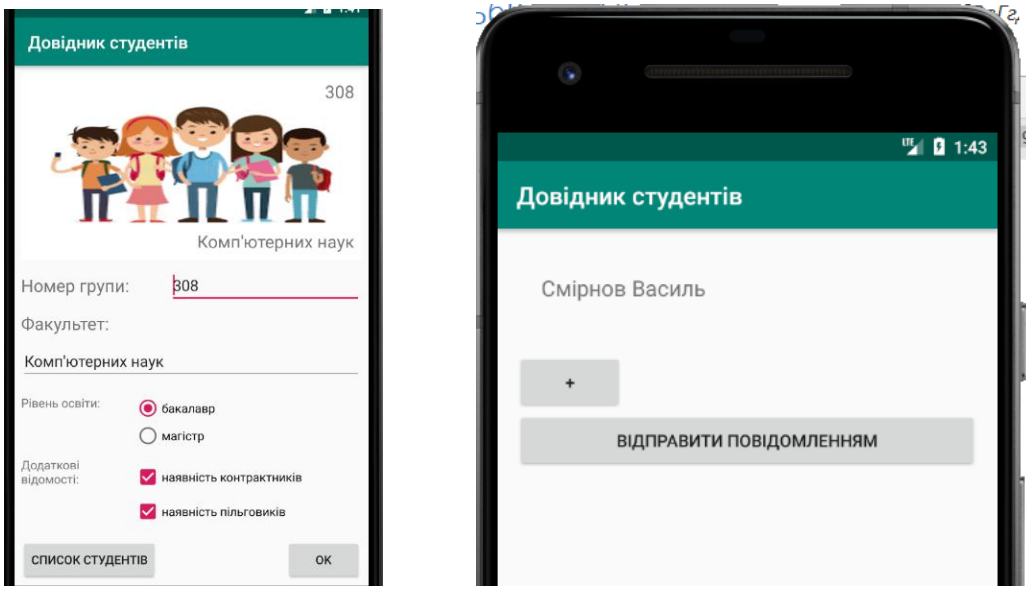



Рисунок 2.96 – Запуск активності списку студентів із студентської групи

Змінимо макет активності списку студентів activity_students_list.xml. Видалимо компоненти TextView і кнопку «+», додаємо компонент ListView (рис. 2.97).

activity_students_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:orientation="vertical"
    tools:context=".StudentsListActivity">

    <ListView
        android:id="@+id/studentsList"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/sendBtn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Відправити повідомленням"
        android:layout_marginBottom="16dp"
        android:onClick="onSendBtnClick"/>

</LinearLayout>
```

Рисунок 2.97 – Код макета activity_students_list.xml

У коді активності StudentsListActivity також прибираємо приватне поле textSize, методи onPlusBtnClick і onSaveInstanceState та більшість коду із методу onCreate. Також у метод onCreate прописуємо код для заповнення списку студентів даними (рис. 2.98).

```
16      @Override
17      protected void onCreate(Bundle savedInstanceState) {
18          super.onCreate(savedInstanceState);
19          setContentView(R.layout.activity_students_list);
20
21          Intent intent = getIntent();
22          String grpNumber = intent.getStringExtra(GROUP_NUMBER);
23
24          ListView listView = (ListView) findViewById(R.id.studentsList);
25          ArrayAdapter<Student> adapter = new ArrayAdapter<~>(
26                  context: this,
27                  android.R.layout.simple_list_item_1,
28                  Student.getStudents(grpNumber)
29          );
30          listView.setAdapter(adapter);
31      }
```

Рисунок 2.98 – Змінений метод onCreate активності StudentsListActivity

Також перевизначаємо метод toString() у класі Students (рис. 2.99).

```
46      @Override
47      public String toString() {
48          return name;
49      }
```

Рисунок 2.99 – Перевизначення методу toString() у класі Students

Після виконання всіх вищенаведених змін, користувацький екран активності матиме такий вигляд (рис. 2.100).

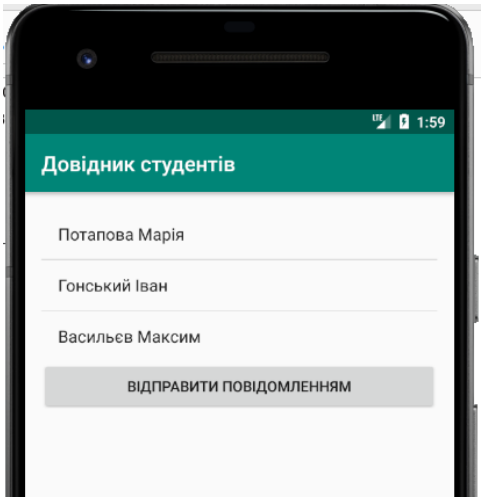



Рисунок 2.100 – Активність списку студентів

## 2.4. Кнопки та панелі інструментів. Провайдер передачі інформації

Створимо нову активність для додавання нової студентської групи. Додаємо нову активність new–>activity–>empty activity AddStudentsGroup Activity. Використовуючи візуальний редактор, приводимо макет activity_ add_students_group.xml до такого вигляду (рис. 2.101).

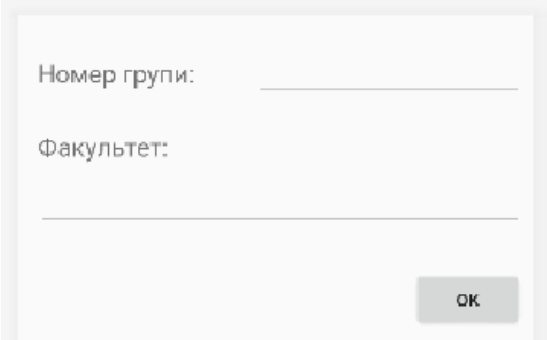



Рисунок 2.101 – Вигляд макета активності додавання нової групи

Далі, на рис. 2.102, наведемо код xml із текстового редактора.

activity_add_students_group.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".AddStudentsGroupActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="32dp"
        android:text="Номер групи:"
        android:textSize="20dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintWidth_percent="0.4" />

    <EditText
        android:id="@+id/addGroupNumber"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toBottomOf="@+id/textView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textView"
        app:layout_constraintTop_toTopOf="@+id/textView" />

    <TextView
        android:id="@+id/textView6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="Факультет:"
        android:textSize="20dp"
        app:layout_constraintStart_toStartOf="@+id/textView"
        app:layout_constraintTop_toBottomOf="@+id/textView" />

    <EditText
        android:id="@+id/addFaculty"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="@+id/textView6"
        app:layout_constraintTop_toBottomOf="@+id/textView6" />

```

```
    <Button
        android:id="@+id/addBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="8dp"
        android:onClick="onGrpAddClick"
        android:text="@android:string/VideoView_error_button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/addFaculty" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 2.102 – Повний код макету activity_add_students_group.xml

Для можливості додавання до масиву студентських груп у класі StudentsGroup потрібно виконати невеликі зміни. Приберемо із поля groups ключове слово final та створюємо метод для додавання до масиву нового значення (рис. 2.103).

```
private static final ArrayList<StudentsGroup> groups = new ArrayList<StudentsGroup>(
        Arrays.asList(
            new StudentsGroup( number: "301", facultyName: "Комп'ютерних наук",

public static void addGroup(StudentsGroup group) { groups.add(group); }
```

Рисунок 2.103 – Зміни у StudentsGroup

Тепер реалізуємо виклик активності додавання із активності списку студентських груп. Починаємо із визначення напису на кнопці у strings.xml (рис. 2.104).

```
    <string name="btn_groups_txt">Студентські групи</string>
```

Рисунок 2.104 – Додавання тексту напису на кнопці

До макета activity_groups_list.xml додаємо кнопку із вищезазначеним написом, ідентифікатором btnGrpAdd та методом onGrpAddClick на подію onClick (рис. 2.105).

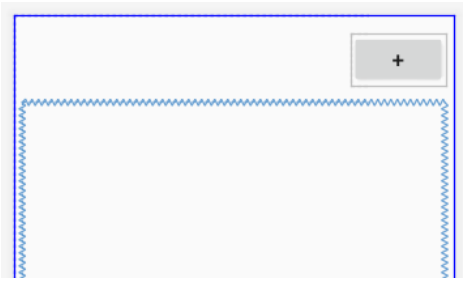



Рисунок 2.105 – Кнопка додавання групи

У коді активності GroupsListActivity реалізуємо метод onGrpAddClick (рис. 2.106).

```
    public void onGrpAddClick(View view) {
        startActivity(
                new Intent( packageContext: this, AddStudentsGroupActivity.class)
        );
    }
```

Рисунок 2.106 – Метод onGrpAddClick активності GroupsListActivity

У коді активності додавання групи AddStudentsGroupActivity також реалізуємо метод onGrpAddClick, що спрацьовуватиме по натисканню на кнопку «OK» (рис. 2.107).

```
18      public void onGrpAddClick(View view) {
19          EditText number = (EditText)findViewById(R.id.addGroupNumber);
20          EditText faculty = (EditText)findViewById(R.id.addFaculty);
21          StudentsGroup.addGroup(
22              new StudentsGroup(number.getText().toString(),
23                  faculty.getText().toString(),
24                   educationLevel: 0,  contractExistsFlg: false,  privilageExistsFlg: false
25              )
26          );
27          NavUtils.navigateUpFromSameTask( sourceActivity: this);
28      }
```

Рисунок 2.107 – Додавання елемента до колекції студентських груп

Останній рядок методу виконує повернення до батьківської активності. Пропишемо у AndroidManifest.xml для активності AddStudentsGroup Activity батьківську активність GroupsListActivity (рис. 2.108).

```
14          <activity android:name=".AddStudentsGroupActivity"
15              android:parentActivityName=".GroupsListActivity">
16          </activity>
```

Рисунок 2.108 – Вказання батьківської активності для AddStudentsGroupActivity

У коді активності GroupsListActivity перенесемо оновлення вмісту списку студентських груп із onCreate в onStart для того, щоб список оновлювався під час відновлення видимості активності, а саме – у процесі повернення до неї від активності AddStudentsGroupActivity (рис. 2.109).

```
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.activity_groups_list);
18
19          AdapterView.OnItemClickListener listener = (adapterView, view, i, l) -> {
22                  String group = ((StudentsGroup) adapterView.getItemAtPosition(i))
23                          .toString();
24                  Intent intent = new Intent( packageContext: GroupsListActivity.this,
25                          StudentsGroupActivity.class);
26                  intent.putExtra(StudentsGroupActivity.GROUP_NUMBER, group);
27                  startActivity(intent);
28          };
30
31          ListView listView = (ListView) findViewById(R.id.groups_list);
32          listView.setOnItemClickListener(listener);
33      }
34
35      @Override
36      protected void onStart() {
37          super.onStart();
38          ListView listView = (ListView) findViewById(R.id.groups_list);
39          ArrayAdapter<StudentsGroup> adapter = new ArrayAdapter<StudentsGroup>(
40                  context: this,
41                  android.R.layout.simple_list_item_1,
42                  StudentsGroup.getGroups()
43          );
44          listView.setAdapter(adapter);
45      }
```

Рисунок 2.109 – Зміни у методах onCreate та onStart активності GroupsListActivity

Переходимо до перевірки роботи застосунку. Завантажуємо список студентських груп, додаємо нову та повертаємося до списку, у якому вже присутня нова група (рис. 2.110). Також звернімо увагу, що під час додавання батьківської активності у AndroidManifest.xml для AddStudents GroupActivity на панелі інструментів з'явилася кнопка Назад.

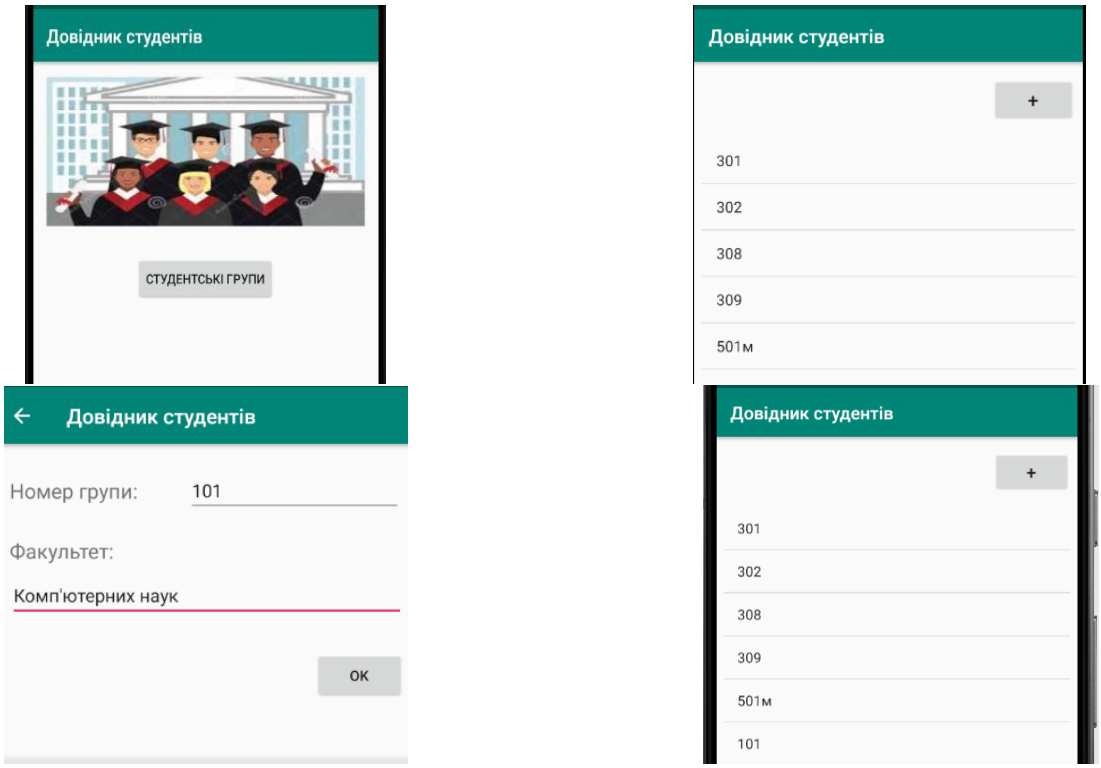



Рисунок 2.110 – Додавання нової групи

Нагадаємо, що у процесі створення програми в основному використовуються екрани трьох видів. Екран верхнього рівня – як правило, це перша активність застосунку, яку бачить користувач. Екрани категорій – на екранах категорій виводяться дані, що відносяться до певної категорії, – часто у вигляді списку. На них користувач може перейти до екранів деталізації/редагування. Екрани деталізації/редагування – на цих екранах виводиться докладна інформація по конкретних записах, користувач може редагувати існуючі або створювати нові записи.

Із активності категорій GroupsListActivity списку студентських груп виконується перехід до двох активностей деталізації/редагування: StudentsGroupActivity – для відображення детальних даних студентської групи та AddStudentsGroupActivity – для додавання нової групи. Якщо перша залежить від обраного елемента списку, то друга не залежить від елементів активності списку студентських груп.

Іншими словами, можна сказати, що за допомогою цієї активності користувач виконує дію. У застосунках Android дії зазвичай додаються на

панель застосунку. Ця панель у більшості випадків розташовується в верхньої частини макета активності; іноді її називають панеллю дій. На ній розміщуються найбільш часто використовувані дії.

Почнімо із створення нового меню у вигляді файлу ресурсів (рис. 2.111).

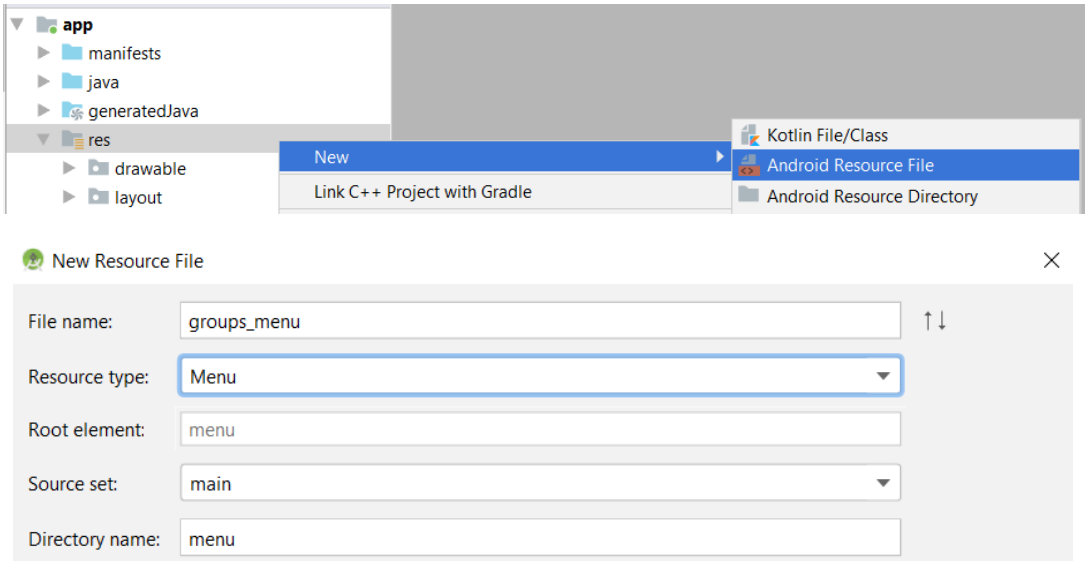



Рисунок 2.111 – Меню для активності списку студентських груп

На рис. 2.112 наведено xml-код ресурсного файлу меню у текстовому редакторі.

```
groups_menu.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto">
4      <item
5          android:id="@+id/action_add_group"
6          android:icon="@android:drawable/ic_input_add"
7          android:orderInCategory="1"
8          android:title="@string/btn_add"
9          app:showAsAction="ifRoom" />
10 </menu>
```

Рисунок 2.112 – Xml-код меню

Далі у коді активності GroupsListActivity виконаємо виведення створеного меню на панелі застосунку. Для цього у класі GroupsListActivity перевизначимо метод onCreateOptionsMenu (рис. 2.113).

```
53
54     @Override
55     public boolean onCreateOptionsMenu(Menu menu) {
56         getMenuInflater().inflate(R.menu.groups_menu, menu);
57         return super.onCreateOptionsMenu(menu);
58     }
```

Рисунок 2.113 – Перевизначення onCreateOptionsMenu в класі GroupsListActivity

Також змінимо текст напису на панелі активності із стандартного для нашого застосунку «Довідник студентів» на «Список груп». Для цього в AndroidManifest.xml у елементі активності GroupsListActivity додаємо свій android:label. Також, для можливості використання кнопки «назад», додаємо в AndroidManifest.xml до активності GroupsListActivity атрибут parentActivityName (рис. 2.114).

```
17      <activity android:name=".GroupsListActivity"
18          android:parentActivityName=".MainActivity"
19          android:label="Список груп"/>
```

Рисунок 2.114 – Зміни в AndroidManifest.xml для активності GroupsListActivity

Виконаємо перевірку змін, запустивши застосунок та перейшовши до активності студентських груп (рис. 2.115).

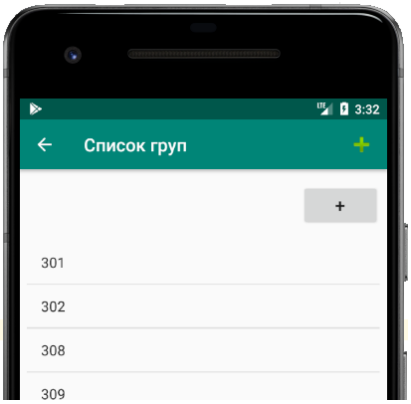



Рисунок 2.115 – Перевірка змін користувацького екрана активності GroupsListActivity

У активності все ще залишається звичайна кнопка «+» – її ми видаляємо із макета. Але тепер необхідно створити метод для обробки події натискання на кнопку «+» в меню панелі застосунку. Для цього у коді активності GroupsListActivity реалізуємо метод onOptionsItemSelected (рис. 2.116). Також видаляємо метод onGrpAddClick, який більше не потрібний.

```
@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_add_group:
            startActivity(
                    new Intent( packageContext: this, AddStudentsGroupActivity.class)
            );
        default:
            return super.onOptionsItemSelected(item);
    }

}
```

Рисунок 2.116 – Реалізація методу onOptionsItemSelected

Перевіряємо роботу застосунку, переходимо до списку груп та пробуємо натиснути пункт меню для додавання нової групи (рис. 2.117).

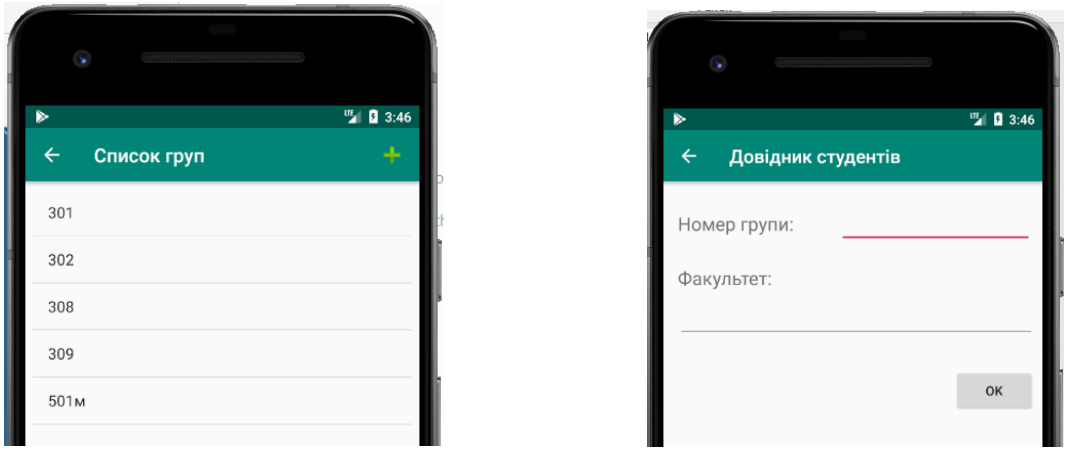



Рисунок 2.117 – Перевірка додавання нової групи

А тепер розглянемо, як використовувати провайдера дій на панелі застосунку. Провайдер дій – елемент, який додається на панель застосунку і сам керує своїм зовнішнім виглядом і поведінкою. Зараз зосередимося на використанні провайдера дії передачі інформації. З його допомогою користувачі можуть передавати інформацію з вашого застосунку в інші застосунки – наприклад, в Gmail. Скажімо, користувач може відправити список студентських груп одному зі своїх контактів. Провайдер дії передачі інформації має власний значок, так що вам не доведеться визначати значок самостійно. Під час натискання провайдер надає список застосунків, яким він вміє передавати інформацію. Він додає окремий значок для програми, яку ви найчастіше обираєте для передачі інформації.

Щоб обмінюватися інформацією через провайдера передачі інформації, слід передати йому інтент. Переданий інтент визначає передану інформацію і її тип. Наприклад, можна визначити інтент, який передає текст з дією ACTION_SEND; дію передачі інформації відкриє список застосунків на пристрої, здатних передавати дані такого типу.

Додаємо до strings.xml новий запис для кнопки (рис. 2.118).

```
25      <string name="btn_share">Поділитися</string>
```

Рисунок 2.118 – Новий напис у strings.xml

Тепер повертаємося до файлу res/menu/groups_menu.xml та додаємо ще один пункт меню (рис. 2.119).

groups_menu.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto">
4      <item
5          android:id="@+id/action_add_group"
6          android:icon="@android:drawable/ic_input_add"
7          android:orderInCategory="1"
8          android:title="@string/btn_add"
9          app:showAsAction="ifRoom" />
```

```
    <item android:id="@+id/action_share"
        android:title="@string/btn_share"
        android:orderInCategory="2"
        app:showAsAction="ifRoom"
        app:actionProviderClass="androidx.appcompat.widget.ShareActionProvider" />
</menu>
```

Рисунок 2.119 – Додавання пункту меню для відправки листа

Далі, у методі onCreateOptionsMenu активності GroupsListActivity реалізуємо отримання списку студентських груп у вигляді рядка, далі за id отримуємо shareActionProvider та передаємо йому інтент ACTION_SEND із рядком – списком студентських груп. Код методу onCreateOptionsMenu після вказаних змін наведено на рис. 2.120.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.groups_menu, menu);

    String text = "";
    for(StudentsGroup group: StudentsGroup.getGroups()) {
        text += group.getNumber() +"\n";
    }

    MenuItem menuItem = menu.findItem(R.id.action_share);
    ShareActionProvider shareActionProvider =
            (ShareActionProvider) MenuItemCompat.getActionProvider(menuItem);

    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_TEXT, text);
    shareActionProvider.setShareIntent(intent);

    return super.onCreateOptionsMenu(menu);
}
```

Рисунок 2.120 – Код методу onCreateOptionsMenu

Перевіряємо роботу, в активності списку груп натискаємо кнопку поділитися та обираємо Gmail (рис. 2.121).

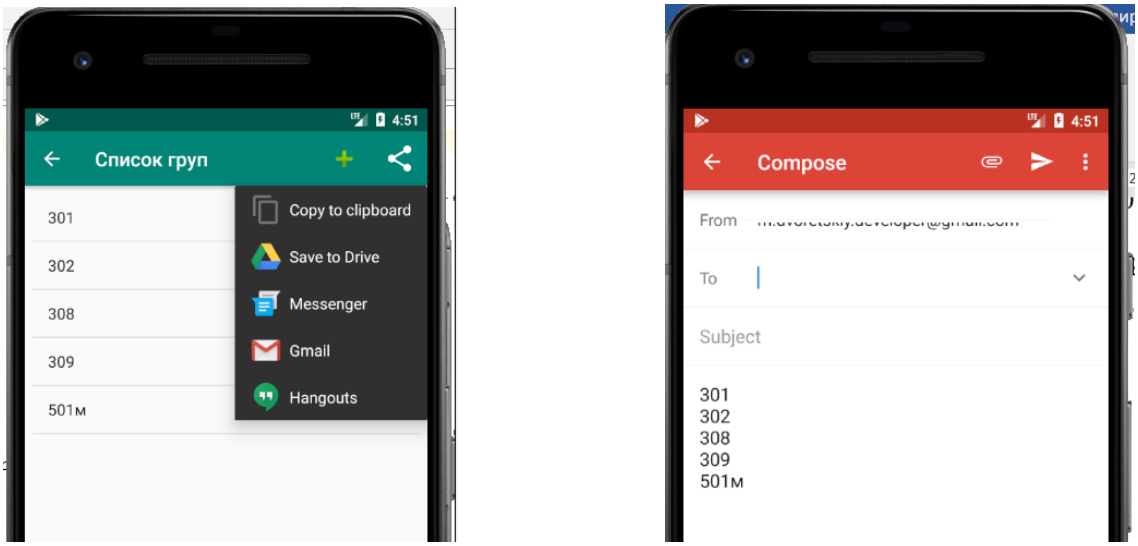



Рисунок 2.121 – Перевірка відправки списку студентських груп

# *РОЗДІЛ 3. РОБОТА ІЗ БАЗОЮ ДАНИХ SQLITE*

## 3.1. Створення БД та отримання даних

Яка б інформація не використовувалася в застосунку – її необхідно десь зберігати. Один із найбільш розповсюджених варіантів – використання баз даних. В Android як локальну БД може бути використано базу даних SQLite. Наведемо декілька переваг використання SQLite.

1. Мінімальні витрати ресурсів. Для роботи більшості систем управління базами даних необхідний спеціальний процес сервера бази даних. SQLite обходиться без сервера; базою даних SQLite є звичайний файл. Коли база даних не використовується, вона не витрачає процесорний час. Це особливо важливо на мобільних пристроях, щоб уникнути розрядки акумулятора.

2. Оптимізація для одного користувача. З базою даних взаємодіє тільки наш застосунок, тому можна обійтися без ідентифікації з іменем користувача і паролем.

3. Надійність і швидкість. Бази даних SQLite підтримують транзакції. Крім того, операції читання і запису даних реалізуються на оптимізованому коді C. Цей код не тільки швидко працює, але і скорочує обсяг необхідних обчислювальних ресурсів.

4. Android автоматично створює для кожної програми папку, в якій зберігаються бази даних цього застосунку. Кожна база даних складається з двох файлів. Перший файл – основний файл баз даних SQLite; в ньому зберігаються всі дані. Другий файл – файл журналу. У файлі журналу зберігається інформація про зміни, внесені до бази даних. Якщо в роботі з даними виникне проблема, Android використовує дані журналу для скасування (або відкату) останніх змін.

Система Android також включає набір класів для управління базою даних SQLite. Основна частина цієї роботи виконується трьома типами об’єктів.

1. Помічник SQLite – створюється розширенням класу SQLiteOpen Helper. Він надає засоби для створення та управління базами даних.

2. Клас Cursor призначений для читання і запису в базу даних. Його можна порівняти з класом ResultSet в JDBC.

3. Клас SQLiteDatabase надає доступ до бази даних. Його можна порівняти з класом SQLConnection в JDBC.

Ми скористаємося об’єктом помічника SQLite для створення бази даних, яка буде використовуватися нашим застосунком. Щоб замінити клас Java StudentsGroup базою даних, помічник SQLite повинен: створити

версію 1 (першу версію) бази даних Students; створити таблицю Groups і заповнити її інформацією про студентські групи. Структура програми майже не змінюється, якщо не брати до уваги того, що файл StudentsGroup.java замінюється об'єктом помічника SQLite і базою даних SQLite Students. Помічник SQLite буде керувати базою даних Students і забезпечувати доступ до її даних із активностей.

Щоб створити помічника SQLite, реалізуємо клас, який розширює SQLiteOpenHelper. При цьому необхідно перевизначити методи onCreate() і onUpgrade(). Ці методи є обов'язковими. Метод onCreate() викликається під час першого створення бази даних на пристрої. Він повинен включати весь код, необхідний для створення таблиць, які використовуються в застосунку. У нашому застосунку буде використовуватися помічник SQLite з ім'ям StudentsDatabaseHelper. Створимо цей клас (рис. 3.1) та помістимо у нього такий код (рис. 3.2).

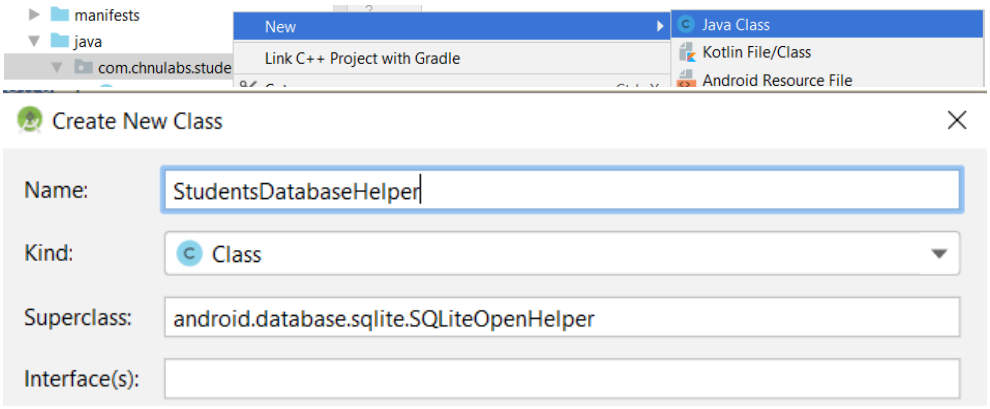



Рисунок 3.1 – Створення класу StudentsDatabaseHelper

StudentsDatabaseHelper.java

```
1  package com.chnulabs.students;
2
3  import android.content.Context;
4  import android.database.sqlite.SQLiteDatabase;
5  import android.database.sqlite.SQLiteOpenHelper;
6
7  public class StudentsDatabaseHelper extends SQLiteOpenHelper {
8
9      private static final String DB_NAME = "students";
10     private static final int DB_VERSION = 1;
11
12     public StudentsDatabaseHelper(Context context) {
13         super(context, DB_NAME,  factory: null, DB_VERSION);
14     }
15
16     @Override
17     public void onCreate(SQLiteDatabase sqLiteDatabase) {
18     }
19
20     @Override
21     public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldV, int newV) {
22     }
23 }
```

Рисунок 3.2 – Код класу StudentsDatabaseHelper

Конструктор задає інформацію про базу даних, але сама база даних в цій точці не створюється. Помічник SQLite очікує, поки застосунок звернеться до бази даних, і створює базу цьому в цій точці. Тепер можна переходити до визначення таблиць. На цьому етапі нам потрібно створити таблицю для збереження даних по студентських групах. У таблиці мають бути представлені такі атрибути: унікальне поле – первинний ключ id, номер групи – number, назва факультету – facultyName, рівень освіти – educationLevel, флаг наявності контрактників – contractExistsFlg та флаг наявності пільговиків – privilageExistsFlg. Наведемо SQL-код для створення таблиці Groups бази даних Students (рис. 3.3).

```
CREATE TABLE Groups (
    id                 INTEGER    PRIMARY KEY AUTOINCREMENT,
    number             TEXT (10)  NOT NULL,
    facultyName        TIME (100),
    educationLevel     INTEGER,
    contractExistsFlg  BOOLEAN,
    privilageExistsFlg BOOLEAN
);
```

Рисунок 3.3 – SQL-код на створення таблиці Groups

Помічник SQLite відповідає за те, щоб база даних SQLite була створена в момент її першого використання. Спочатку на пристрої створюється пуста база даних, після чого викликається метод onCreate(). Метод отримує один параметр – об’єкт SQLiteDatabase, що представляє створену базу даних. Можна скористатися цим об’єктом для виконання команди SQL за допомогою execSQL(String sql). Також у методі onCreate() будемо виконувати заповнення таблиці Groups початковими даними, що представлені зараз у класі StudentsGroup.

Щоб занести нові дані в таблицю бази даних SQLite, спочатку слід створити об’єкт ContentValues. Об’єкт ContentValues описує набір даних. Зазвичай створюється новий об’єкт ContentValues для кожного рядка даних, яку потрібно створити. Для додавання даних в об’єкт ContentValues використовується метод put(). Метод додає дані у вигляді пар «ім’я – значення». Після додавання рядка даних в об’єкт ContentValues для її вставки в таблицю використовується метод insert() класу SQLiteDatabase. Метод вставляє дані в таблицю і повертає ідентифікатор запису після її вставки.

Далі, на рис. 3.4 наведемо код методу onCreate() та додаткових методів populateDB() та insertRow() для заповнення таблиці даними.

```
17      @Override
18      public void onCreate(SQLiteDatabase sqLiteDatabase) {
19          String sqlText = "CREATE TABLE Groups (\n" +
20                  "id                 INTEGER    PRIMARY KEY AUTOINCREMENT,\n" +
21                  "number             TEXT (10)  NOT NULL,\n" +
22                  "facultyName        TIME (100),\n" +
23                  "educationLevel     INTEGER,\n" +
```

```
                "contractExistsFlg  BOOLEAN,\n" +
                "privilageExistsFlg BOOLEAN\n" +
                ");";
        sqLiteDatabase.execSQL(sqlText);

        populateDB(sqLiteDatabase);
    }

    private void populateDB(SQLiteDatabase db) {
        for(StudentsGroup group: StudentsGroup.getGroups()) {
            insertRow(db, group);
        }
    }

    private void insertRow(SQLiteDatabase db, StudentsGroup group) {
        ContentValues contentValues = new ContentValues();
        contentValues.put("number", group.getNumber());
        contentValues.put("facultyName", group.getFacultyName());
        contentValues.put("educationLevel", group.getEducationLevel());
        contentValues.put("contractExistsFlg", group.isContractExistsFlg());
        contentValues.put("privilageExistsFlg", group.isPrivilageExistsFlg());
        db.insert( table: "Groups",  nullColumnHack: null, contentValues);
    }
```

Рисунок 3.4 – Методи для створення таблиці Groups
та заповнення її даними

До активності списку студентський груп додаємо метод, що під'єднається до БД та виконає запит до таблиці Groups. Виконаємо виклик цього методу під час створення активності просто для того, щоб на пристрої створився файл БД (рис. 3.5–3.6).

```
    private ArrayList<StudentsGroup> getDataFromDB() {
        ArrayList<StudentsGroup> groups = new ArrayList<StudentsGroup>();

        SQLiteOpenHelper sqliteHelper = new StudentsDatabaseHelper( context: this);
        try {
            SQLiteDatabase db = sqliteHelper.getReadableDatabase();
            Cursor cursor = db.query( table: "groups",
                    new String[] {"number","facultyName"},
                     selection: null,  selectionArgs: null,  groupBy: null,
                     having: null,  orderBy: null);
        } catch(SQLiteException e) {
            Toast toast = Toast.makeText( context: this,
                     text: "Database unavailable",
                    Toast.LENGTH_SHORT);
            toast.show();
        }
        return groups;
    }
```

Рисунок 3.5 – Метод для підключення до БД

```
    @Override
    protected void onStart() {
        super.onStart();

        getDataFromDB();

        ListView listView = (ListView) findViewById(R.id.groups_list);
        ArrayAdapter<StudentsGroup> adapter = new ArrayAdapter<StudentsG
```

Рисунок 3.6 – Виклик методу під час старту активності

Тепер запустимо застосунок та виконаємо перехід до активності студентських груп, в результаті чого має створитися БД. Перевіримо це, запустивши Device File Explorer у правому нижньому кутку вікна Android Studio (рис. 3.7).

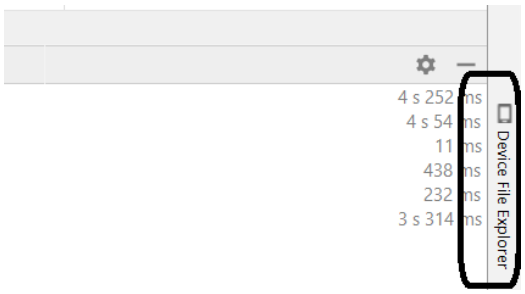



Рисунок 3.7 – Запуск Device File Explorer

Знайдемо файл БД у директорії /data/data/com.chnulabs.students/databases та збережемо його на комп’ютері (рис. 3.8).

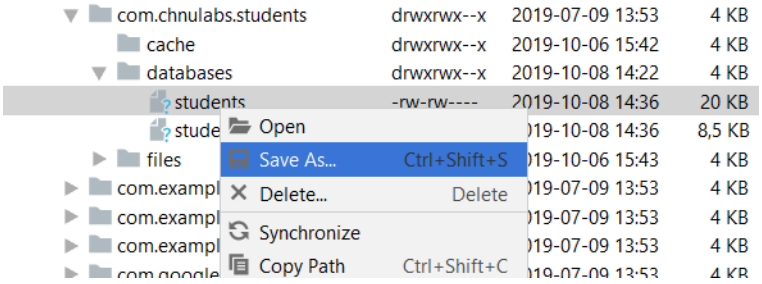
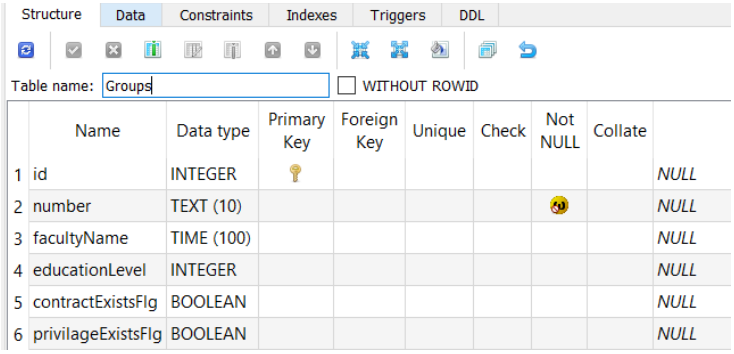


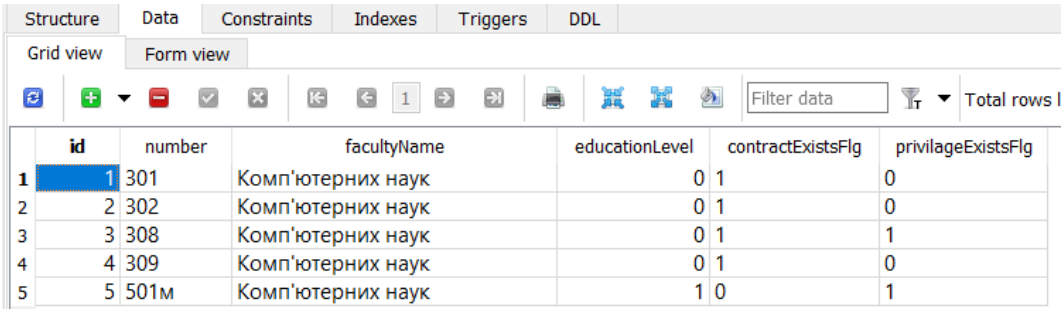

Рис. 3.8. Копіювання файлу БД із пристрою на комп’ютер

Далі відкриємо його, використавши, наприклад, SQLiteStudio (може бути завантажено тут: https://sqlitestudio.pl/index.rvt?act=download) або іншу програму для перегляду даних БД sqlite (рис. 3.9–3.10).




| | Name | Data type | Primary Key | Foreign Key | Unique | Check | Not NULL | Collate | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | id | INTEGER | | | | | | | NULL |
| 2 | number | TEXT (10) | | | | | | | NULL |
| 3 | facultyName | TIME (100) | | | | | | | NULL |
| 4 | educationLevel | INTEGER | | | | | | | NULL |
| 5 | contractExistsFlg | BOOLEAN | | | | | | | NULL |
| 6 | privilageExistsFlg | BOOLEAN | | | | | | | NULL |

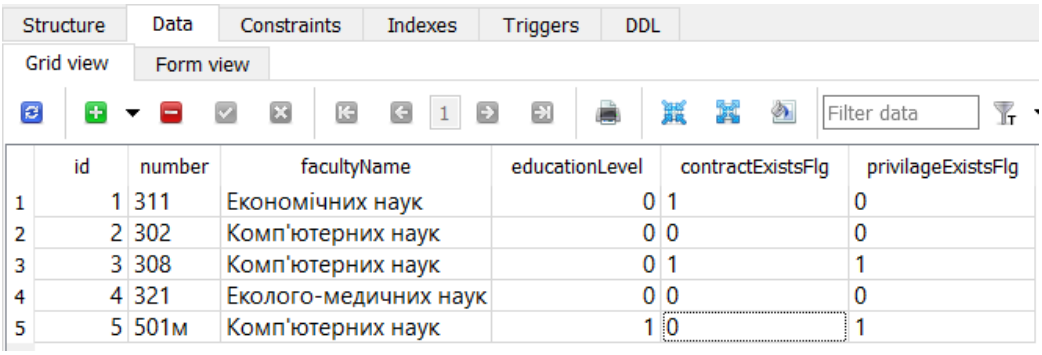
Рисунок 3.9 – Структура таблиці Groups




| | id | number | facultyName | educationLevel | contractExistsFlg | privilageExistsFlg |
|---|---|---|---|---|---|---|
| 1 | 1 | 301 | Комп'ютерних наук | 0 | 1 | 0 |
| 2 | 2 | 302 | Комп'ютерних наук | 0 | 1 | 0 |
| 3 | 3 | 308 | Комп'ютерних наук | 0 | 1 | 1 |
| 4 | 4 | 309 | Комп'ютерних наук | 0 | 1 | 0 |
| 5 | 5 | 501м | Комп'ютерних наук | 1 | 0 | 1 |

Рисунок 3.10 – Дані таблиці Groups

Як бачимо, в результаті виконання нашого коду була створена БД student, а у ній – таблиця Groups. Таблиця була заповнена даними згідно з колекцією студентських груп із класу StudentsGroup. Далі виконаємо читання цих груп із БД та подальше їх відображення у макеті активності. Але для початку трохи змінимо дані БД, щоб у подальшому пересвідчитися у тому, що відображаються саме вони та завантажемо її на пристрій (рис. 3.11–3.12).




| | id | number | facultyName | educationLevel | contractExistsFlg | privilageExistsFlg |
|---|---|---|---|---|---|---|
| 1 | 1 | 311 | Економічних наук | 0 | 1 | 0 |
| 2 | 2 | 302 | Комп'ютерних наук | 0 | 0 | 0 |
| 3 | 3 | 308 | Комп'ютерних наук | 0 | 1 | 1 |
| 4 | 4 | 321 | Еколого-медичних наук | 0 | 0 | 0 |
| 5 | 5 | 501м | Комп'ютерних наук | 1 | 0 | 1 |

Рисунок 3.11 – Зміна даних у БД

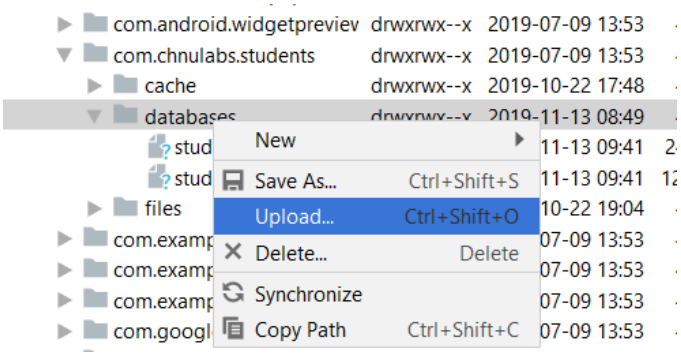



Рисунок 3.12 – Завантаження файлу на пристрій

Повернемося до методу getDataFromDB. На цей момент у ньому створюється екземпляр помічника SQLite та всередині спроби відкривається БД для читання, потім за допомогою курсора виконується запит на читання даних із таблиці Groups. Курсор забезпечує читання і запис інформації в базу даних. Ми вказуємо, які дані нас цікавлять, а курсор вибирає відповідні записи з бази даних. Після цього ми можемо переміщатися між записами, наданими курсором. Виконаємо переміщення по курсору та повернення даних по студентським групам із методу (рис. 3.13).

```
47      private ArrayList<StudentsGroup> getDataFromDB() {
48          ArrayList<StudentsGroup> groups = new ArrayList<~>();
49
50          SQLiteOpenHelper sqliteHelper = new StudentsDatabaseHelper( context: this);
51          try {
52              SQLiteDatabase db = sqliteHelper.getReadableDatabase();
53              Cursor cursor = db.query( table: "groups",
54                      new String[] {"number","facultyName","educationLevel",
55                              "contractExistsFlg","privilageExistsFlg"},
```

```
                selection: null,  selectionArgs: null,  groupBy: null,
                having: null,  orderBy: "number");
        while (cursor.moveToNext()) {
            groups.add(
                    new StudentsGroup(
                            cursor.getString( i: 0),
                            cursor.getString( i: 1),
                            cursor.getInt( i: 2),
                            (cursor.getInt( i: 3) > 0),
                            (cursor.getInt( i: 4) > 0)
                             (cursor.getInt( i: 4) > 0)
                    )
            );
        }
        cursor.close();
        db.close();
    } catch(SQLiteException e) {
        Toast toast = Toast.makeText( context: this,
                text: "Database unavailable",
                Toast.LENGTH_SHORT);
        toast.show();
    }
    return groups;
}
```

Рисунок 3.13 – Доопрацьований метод getDataFromDB()

Також у onStart() змінимо виклик StudentsGroup.getGroups() на getData FromDB() (рис. 3.14).

```
    ArrayAdapter<StudentsGroup> adapter = new ArrayAdapter<~>(
            context: this,
            android.R.layout.simple_list_item_1,
            //StudentsGroup.getGroups()
            getDataFromDB()
    );
```

Рисунок 3.14 – Зміна виклику методу на getDataFromDB() в onStart()

Запустимо застосунок, перейдемо на список груп та перевіримо наявність груп, що ми ввели вручну через SQLiteStudio (рис. 3.15).

Зараз ідентифікація студентської групи відбувається за її номером. Це є не зовсім правильним підходом, оскільки у деяких випадках імена об’єктів у БД можуть співпадати, а ідентифікатори (первинні ключі) при цьому можуть відрізнятися. Почнімо із доопрацювання класу StudetnsGroup, додаємо поле id, конструктор та геттер. Сеттер для цього поля не потрібен, оскільки за логікою нашого застосунку воно є незмінним (рис. 3.16).

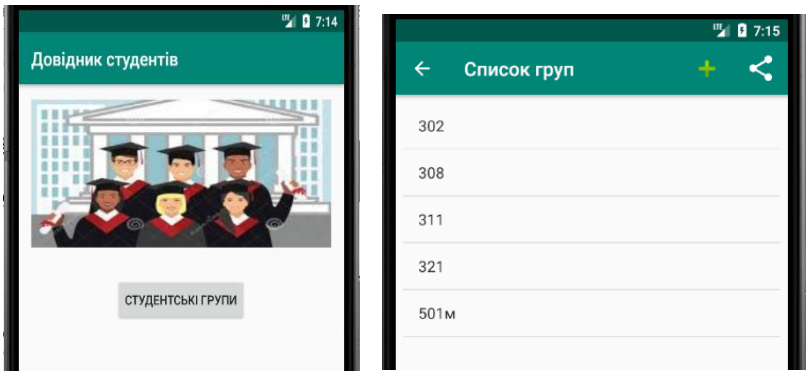


Рисунок 3.15 – Перевірка виведення списку студентських груп із БД

```
6   public class StudentsGroup {
7       private int id;
8       private String number;
14      public StudentsGroup(int id, String number, String facultyName, int educationLevel,
15                           boolean contractExistsFlg, boolean privilageExistsFlg) {
16          this(number, facultyName, educationLevel, contractExistsFlg, privilageExistsFlg);
17          this.id = id;
18      }
19
20      public StudentsGroup(String number, String facultyName, int educationLevel,
21                           boolean contractExistsFlg, boolean privilageExistsFlg) {
27      }
28
29      public int getId() { return this.id; }
```

Рисунок 3.16 – Додавання id у клас StudetnsGroup

У методі getDataFromDB активності GroupsListActivity виконаємо читання поля id із БД та створення об’єкта студентської групи із урахуванням значення цього поля (рис. 17).

```
50          try {
51              SQLiteDatabase db = sqliteHelper.getReadableDatabase();
52              Cursor cursor = db.query( table: "groups",
53                      new String[] {"number","facultyName","educationLevel",
54                              "contractExistsFlg","privilageExistsFlg", "id"},
55                      selection: null,  selectionArgs: null,  groupBy: null,
56                      having: null,  orderBy: "number");
57              while (cursor.moveToNext()) {
58                  groups.add(
59                          new StudentsGroup(
60                                  cursor.getInt( i: 5),
61                                  cursor.getString( i: 0),
62                                  cursor.getString( i: 1),
```

Рисунок 3.17 – Зміни у методі getDataFromDB активності GroupsListActivity

Також у методі onCreate активності GroupsListActivity під час створення слухача OnItemClickListener для ListView будемо передавати до активності StudentsGroupActivity значення поля id замість номера групи (рис. 3.18).

```
26      @Override
27      protected void onCreate(Bundle savedInstanceState) {
28          super.onCreate(savedInstanceState);
29          setContentView(R.layout.activity_groups_list);
30
31          AdapterView.OnItemClickListener listener = new AdapterView.OnItemClickListener() {
32              @Override
33              public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
34                  StudentsGroup group = (StudentsGroup) adapterView.getItemAtPosition(i);
35                  Intent intent = new Intent( packageContext: GroupsListActivity.this,
36                          StudentsGroupActivity.class);
37                  intent.putExtra(StudentsGroupActivity.GROUP_NUMBER, group.getId());
38                  startActivity(intent);
39              }
40          };
41
42          ListView listView = (ListView) findViewById(R.id.groups_list);
```

Рисунок 3.18 – Передача id до активності StudentsGroupActivity

Тепер перейдемо до активності StudentsGroupActivity та у методі onCreate виконаємо читання із БД даних студентської групи та створення на їх основі об’єкта StudentsGroup (рис. 3.19). Наступна частина коду щодо виведення даних у компоненти залишається без змін.

```
23      @Override
24      protected void onCreate(Bundle savedInstanceState) {
25          super.onCreate(savedInstanceState);
26          setContentView(R.layout.activity_students_group2);
27
28          Intent intent = getIntent();
29          int grpNumber = intent.getIntExtra(GROUP_NUMBER,  defaultValue: 0);
30          StudentsGroup group = null;
31          SQLiteOpenHelper sqliteHelper = new StudentsDatabaseHelper( context: this);
32          try {
33              SQLiteDatabase db = sqliteHelper.getReadableDatabase();
34              Cursor cursor = db.query( table: "groups",
35                      new String[] {"number","facultyName","educationLevel",
36                              "contractExistsFlg","privilageExistsFlg", "id"},
37                      selection: "id=?", new String[] {Integer.toString(grpNumber)},
38                      groupBy: null, having: null,  orderBy: null);
39              if (cursor.moveToFirst()) {
40                  group = new StudentsGroup(
41                                  cursor.getInt( i: 5),
42                                  cursor.getString( i: 0),
43                                  cursor.getString( i: 1),
44                                  cursor.getInt( i: 2),
45                                  (cursor.getInt( i: 3) > 0),
46                                  (cursor.getInt( i: 4) > 0)
47                          );
48              } else {
49                  Toast toast = Toast.makeText( context: this,
50                          text: "Can't find group with id " + Integer.toString(grpNumber),
51                          Toast.LENGTH_SHORT);
52              }
53              cursor.close();
54              db.close();
55          } catch(SQLiteException e) {
56              Toast toast = Toast.makeText( context: this,
57                      text: "Database unavailable",
58                      Toast.LENGTH_SHORT);
59              toast.show();
60          }
61
62          if (group != null) {
```

Рисунок 3.19 – Зміни у методі onCreate активності StudentsGroupActivity

Перевіримо роботу нашого коду, завантаживши активність списку студентських груп та перейшовши до перегляду даних окремої групи (рис. 3.20).

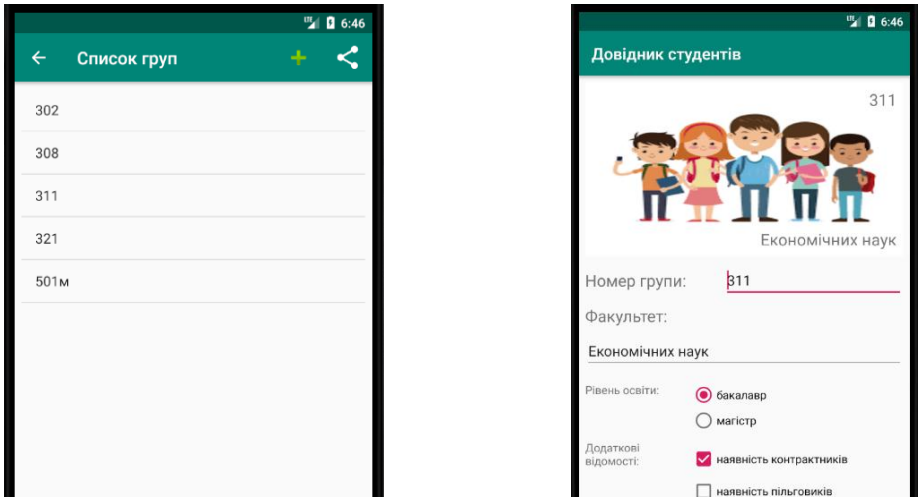



Рисунок 3.20 – Перевірка роботи застосунку

Зауважимо, що на цьому етапі поки що не працюють функції додавання нової групи на зміни атрибутів існуючих.

## 3.2. Версії БД та модифікація даних

Крім створення, оновлення та видалення записів бази даних також може виникнути необхідність у зміні структури бази даних. Так, у нашому застосунку необхідно зберігати дані не тільки щодо студентських груп, а і списки студентів цих груп. Тобто додатково у БД нам знадобиться ще таблиця Students.

Перше, що спадає на думку, – додавання до методу onCreate коду, що створить таблицю студентів. Але тут відразу виникають деякі складності. Якщо на пристрої вже створена БД, метод не буде виконано, а отже не буде створено додаткову таблицю. Якщо ж ми вирішимо видалити БД для того, щоб вона створилась заново із всіма необхідними таблицями, це призведе до втрати всіх раніше занесених до БД даних.

Тут на допомогу приходить здатність помічника SQLite зберігати версію БД та метод onUpgrade(). Реалізуємо метод updateShema у нашому помічнику, змінимо версію БД із першої на другу та виконаємо виклик реалізованого методу в подіях onCreate та onUpgrade. Також метод populateDB розділяється на populiteGroups та populateStudents (рис. 3.21).

StudentsDatabaseHelper.java

```
package com.chnulabs.students;

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

public class StudentsDatabaseHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "students";
    private static final int DB_VERSION = 2;

    public StudentsDatabaseHelper(Context context) {
        super(context, DB_NAME,  factory: null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        String sqlText = "CREATE TABLE Groups (\n" +
                "id                  INTEGER    PRIMARY KEY AUTOINCREMENT,\n" +
                "number              TEXT (10)  NOT NULL,\n" +
                "facultyName         TIME (100),\n" +
                "educationLevel      INTEGER,\n" +
                "contractExistsFlg   BOOLEAN,\n" +
                "privilageExistsFlg BOOLEAN\n" +
                ");";
        sqLiteDatabase.execSQL(sqlText);

        updateShema(sqLiteDatabase,  oldV: 0);

        populateDB(sqLiteDatabase);
    }

    private void populateDB(SQLiteDatabase db) {
        populiteGroups(db);
        populateStudents(db);
    }
```

```
38
39      private void populiteGroups(SQLiteDatabase db) {
40          for(StudentsGroup group: StudentsGroup.getGroups()) {
41              insertRowToGroup(db, group);
42          }
43      }
44
45      private void populateStudents(SQLiteDatabase db) {
46          for(Student student: Student.getStudents()) {
47              insertRowToStudent(db, student);
48          }
49      }
50
51  @   private void insertRowToGroup(SQLiteDatabase db, StudentsGroup group) {
52          ContentValues contentValues = new ContentValues();
53          contentValues.put("number", group.getNumber());
54          contentValues.put("facultyName", group.getFacultyName());
55          contentValues.put("educationLevel", group.getEducationLevel());
56          contentValues.put("contractExistsFlg", group.isContractExistsFlg());
57          contentValues.put("privilageExistsFlg", group.isPrivilageExistsFlg());
58          db.insert( table: "Groups",  nullColumnHack: null, contentValues);
59      }
60
61  @   private void insertRowToStudent(SQLiteDatabase db, Student student) {
62          db.execSQL("insert into students(name, group_id)\n" +
63                  "select '"+student.getName()+"', id\n" +
64                  "from groups\n" +
65                  "where number='" + student.getGroupNumber() +"'");
66      }
67
68      @Override
69  ↑   public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldV, int newV) {
70          updateShema(sqLiteDatabase, oldV);
71      }
72
73      private void updateShema(SQLiteDatabase db, int oldV) {
74          if (oldV < 2) {
75              db.execSQL("CREATE TABLE Students (\n" +
76              "    id       INTEGER    PRIMARY KEY AUTOINCREMENT,\n" +
77              "    name     TEXT (100) NOT NULL,\n" +
78              "    group_id INTEGER    REFERENCES Groups (id) ON DELETE RESTRICT\n" +
79              "                                               ON UPDATE RESTRICT\n" +
80              ");");
81              populateStudents(db);
82          }
83      }
84  }
```

Рисунок 3.21 – Клас StudentsDatabaseHelper – реалізація механізмів модифікації структури БД

Виходячи із того, що таблиця Students має посилання (зовнішній ключ) на таблицю Groups, вставка даних виконується через команду insert із отриманням id студентської групи за її номером.

У класі Student перевизначимо метод getStudents для того, щоб можна було отримати список усіх студентів (рис. 3.22).

```
36      public static ArrayList<Student> getStudents() {
37          return getStudents( groupNumber: "");
38      }
39
40      public static ArrayList<Student> getStudents(String groupNumber) {
41          ArrayList<Student> stList = new ArrayList<>();
42          for(Student s: students) {
43              if (s.getGroupNumber().equals(groupNumber) || (groupNumber == "")) {
44                  stList.add(s);
45              }
46          }
47          return stList;
48      }
```

Рисунок 3.22 Перевизначення методу getStudents класу студентів

Далі запускаємо застосунок та переходимо до активності списку студентів для того, щоб відбулося звернення до БД та помічник створив додаткову таблицю і заповнив її даними. Далі за допомогою Device File Explorer скопіюємо її на комп'ютер та переглянемо за допомогою SQLiteStudio (рис. 3.23).

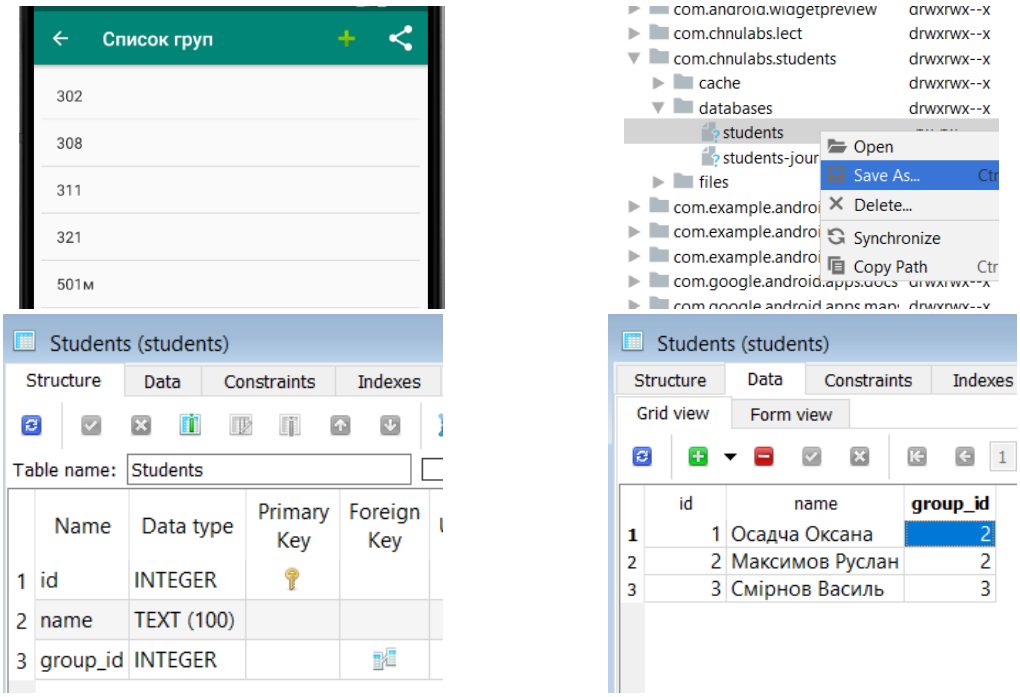



Рисунок 3.23 – Копіювання та перегляд модифікованої БД

Як бачимо, були додані лише ті студенти, групи яких є у БД.

Далі виконаємо необхідні перетворення для відображення списку студентів із БД. Змінимо клас активності студентської групи для передачі до активності списку студентів id групи замість її номера. Для цього створимо приватне поле group та виконаємо запис групи у нього у методі onCreate. Потім, скориставшись цим полем, у методі onBtnStudListClick передамо до активності StudentsListActivity id цієї групи (рис. 3.24).

```
19  public class StudentsGroupActivity extends AppCompatActivity {
20
21      public static final String GROUP_NUMBER = "groupnumber";
22
23      private StudentsGroup group;
24
25      @Override
26      protected void onCreate(Bundle savedInstanceState) {
31          int grpNumber = intent.getIntExtra(GROUP_NUMBER,  defaultValue: 0);
32          group = null;
33          SQLiteOpenHelper sqliteHelper = new StudentsDatabaseHelper( context: this);
119     public void onBtnStudListClick(View view) {
120         Intent newIntent = new Intent( packageContext: this, StudentsListActivity.class);
121         newIntent.putExtra(StudentsListActivity.GROUP_NUMBER, group.getId());
122         startActivity(newIntent);
123     }
```

Рисунок 3.24 – Передача id групи до активності списку студентів

Тепер у активності списку студентів у методі onCreate реалізуємо отримання із БД студентів відповідної групи. Спискове представлення в активності списку студентів у цей момент використовує адаптер масиву для отримання імен студентів. Це пояснюється тим, що дані зберігаються в масиві в класі Student. Такий самий підхід був використаний під час реалізації відображення списку студентських груп, за рахунок поміщення прочитаних із БД даних у масив StudentsGroup. Але є і інший підхід – звернутися напрям до даних курсора за допомогою адаптера курсора. Так, в активності списку студентів StudentsListActivity реалізуємо метод getData FromDB, що повертатиме SimpleCursorAdapter із курсором, що вибиратиме студентів необхідної групи (рис. 3.25).

StudentsListActivity.java

```
package com.chnulabs.students;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.view.View;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import android.widget.TextView;
import android.widget.Toast;

public class StudentsListActivity extends AppCompatActivity {

    public static final String GROUP_NUMBER = "groupnumber";
    private Cursor cursor;
    private SQLiteDatabase db;

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

        Intent intent = getIntent();
        int grpNumber = intent.getIntExtra(GROUP_NUMBER,  defaultValue: 0);

        ListView listView = (ListView) findViewById(R.id.studentsList);
        SimpleCursorAdapter adapter = getDataFromDB(grpNumber);
        if (adapter != null) {
            listView.setAdapter(adapter);
        }
    }

    private SimpleCursorAdapter getDataFromDB(int groupId) {
        SimpleCursorAdapter listAdapter = null;

        SQLiteOpenHelper sqliteHelper = new StudentsDatabaseHelper( context: this);
        try {
            db = sqliteHelper.getReadableDatabase();
            cursor = db.rawQuery( sql: "select s.id _id, name, number\n" +
                            "from students s inner join groups g on s.group_id = g.id\n" +
                            "where g.id = ?", new String[] {Integer.toString(groupId)});
            listAdapter = new SimpleCursorAdapter( context: this,
                    android.R.layout.simple_list_item_1,
                    cursor,
                    new String[]{"name"},
                    new int[]{android.R.id.text1},
                     flags: 0);     }
```

```
53          catch(SQLiteException e) {
54              Toast toast = Toast.makeText( context: this,
55                      text: "Database unavailable",
56                      Toast.LENGTH_SHORT);
57              toast.show();
58          }
59          return listAdapter;
60      }
61
62      @Override
63      protected void onDestroy() {
64          super.onDestroy();
65          cursor.close();
66          db.close();
67      }
68
69      public void onSendBtnClick(View view) {
70          TextView textView = (TextView) findViewById(R.id.text);
71
72          Intent intent = new Intent(Intent.ACTION_SEND);
73          intent.setType("text/plain");
74          intent.putExtra(Intent.EXTRA_TEXT, textView.getText().toString());
75          intent.putExtra(Intent.EXTRA_SUBJECT,  value: "Список студентів");
76          startActivity(intent);
77      }
78  }
```

Рисунок 3.25 – Клас активності списку студентів

Виконаємо перевірку роботи активності та переглянемо список студентів якоїсь групи (рис. 3.26).

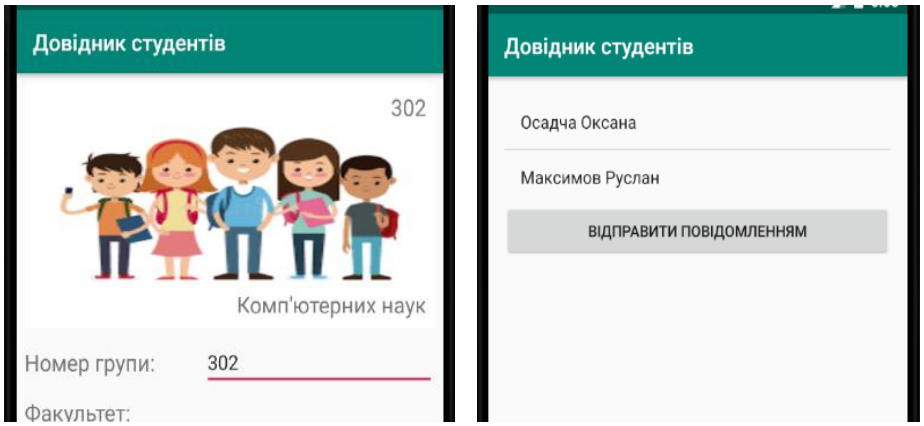



Рисунок 3.26 – Перегляд списку студентів

Реалізуємо функціонал редагування та додавання нової студентської групи у БД SQLite. Почнімо із редагування. Для цього змінимо метод onOkBntClick у класі активності StudentsGroupActivity (рис. 3.27).

```
95      public void onOkBntClick(View view) {
96          SQLiteOpenHelper sqliteHelper = new StudentsDatabaseHelper( context: this);
97
98          ContentValues contentValues = new ContentValues();
99          contentValues.put("number",
100                 ((TextView) findViewById(R.id.grpNumberEdit)).getText().toString()
101         );
102         contentValues.put("facultyName",
103                 ((TextView) findViewById(R.id.facultyEdit)).getText().toString()
104         );
105         contentValues.put("educationLevel",
106                 ((RadioButton) findViewById(R.id.edu_level_master)).isChecked()?1:0
107         );
108         contentValues.put("contractExistsFlg",
109                 ((CheckBox) findViewById(R.id.contract_flg)).isChecked()
110         );
```

```
111             contentValues.put("privilageExistsFlg",
112                     ((CheckBox) findViewById(R.id.privilege_flg)).isChecked()
113             );
114
115             Intent intent = getIntent();
116             int grpNumber = intent.getIntExtra(GROUP_NUMBER,  defaultValue: 0);
117
118             try {
119                 SQLiteDatabase db = sqliteHelper.getReadableDatabase();
120                 db.update( table: "groups",
121                         contentValues,
122                         whereClause: "id=?",
123                         new String[] {Integer.toString(grpNumber)}
124                         );
125                 db.close();
126                 NavUtils.navigateUpFromSameTask( sourceActivity: this);
127             } catch(SQLiteException e) {
128                 Toast toast = Toast.makeText( context: this,
129                         text: "Database unavailable",
130                         Toast.LENGTH_SHORT);
131                 toast.show();
132             }
133     }
```

Рисунок 3.27 – Реалізація зміни даних
та повернення до активності категорій

Також змінимо код методу onGrpAddClick у класі активності AddStudentsGroupActivity для реалізації механізму додавання нового рядку до БД (рис. 3.28).

```
23      public void onGrpAddClick(View view) {
24          EditText number = (EditText)findViewById(R.id.addGroupNumber);
25          EditText faculty = (EditText)findViewById(R.id.addFaculty);
26
27          SQLiteOpenHelper sqliteHelper = new StudentsDatabaseHelper( context: this);
28          try {
29              SQLiteDatabase db = sqliteHelper.getReadableDatabase();
30              ContentValues contentValues = new ContentValues();
31              contentValues.put("number", number.getText().toString());
32              contentValues.put("facultyName", faculty.getText().toString());
33              contentValues.put("educationLevel", 0);
34              contentValues.put("contractExistsFlg", 0);
35              contentValues.put("privilageExistsFlg", 0);
36              db.insert( table: "groups",  nullColumnHack: null, contentValues);
37              db.close();
38              NavUtils.navigateUpFromSameTask( sourceActivity: this);
39          } catch(SQLiteException e) {
40              Toast toast = Toast.makeText( context: this,
41                      text: "Database unavailable",
42                      Toast.LENGTH_SHORT);
43              toast.show();
44          }
45      }
46  }
```

Рисунок 3.28 – Реалізація додавання даних до таблиці БД

Для завершення реалізації операцій CRUD, виконаємо реалізацію механізму видалення студентської групи. Для цього у макеті активності студентської групи activity_students_group2 додаємо кнопку для видалення (рис. 3.29–3.30).

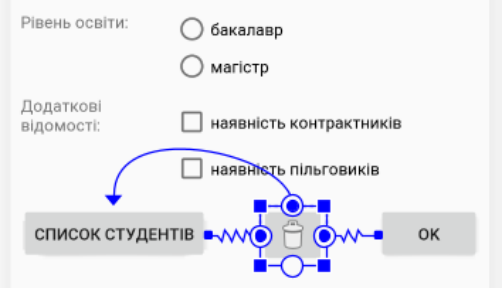



Рисунок 3.29 – Зміни макета activity_students_group2 – візуальний редактор

```
189     <ImageButton
190         android:id="@+id/btnRem"
191         android:layout_width="wrap_content"
192         android:layout_height="wrap_content"
193         android:src="@android:drawable/ic_menu_delete"
194         app:layout_constraintEnd_toStartOf="@+id/btnOk"
195         app:layout_constraintStart_toEndOf="@+id/btnStudList"
196         app:layout_constraintTop_toTopOf="@+id/btnStudList"
197         android:onClick="onDelete"/>
198
```

Рисунок 3.30 – Зміни макета activity_students_group2 – текстовий редактор

Також у класі активності StudentsGroupActivity реалізуємо метод onDelete, що оброблятиме подію натискання та кнопку видалення (рис. 3.31).

```
140
141     public void onDelete(View view) {
142         SQLiteOpenHelper sqliteHelper = new StudentsDatabaseHelper( context: this);
143
144         Intent intent = getIntent();
145         int grpNumber = intent.getIntExtra(GROUP_NUMBER,  defaultValue: 0);
146
147         try {
148             SQLiteDatabase db = sqliteHelper.getReadableDatabase();
149             db.delete( table: "groups",
150                     whereClause: "id=?",
151                     new String[] {Integer.toString(grpNumber)}
152             );
153             db.close();
154             NavUtils.navigateUpFromSameTask( sourceActivity: this);
155         } catch(SQLiteException e) {
156             Toast toast = Toast.makeText( context: this,
157                     text: "Database unavailable",
158                     Toast.LENGTH_SHORT);
159             toast.show();
160         }
161     }
```

Рисунок 3.31 – Реалізація методу onDelete

Перевірити роботу механізмів додавання, зміни та видалення студентської групи. Самостійно реалізувати додавання, зміну та видалення студентів.

# *РОЗДІЛ 4. КЛІЄНТ-СЕРВЕРНІ ЗАСТОСУНКИ*

## 4.1. Виконання GET http-запитів

Сьогодні майже всі програми використовують HTTP / HTTPS-запити як своєрідний транспорт для своїх даних. Навіть якщо ви безпосередньо не використовуєте ці протоколи, багато SDK, які із високою вірогідністю можуть бути включені у застосунок (наприклад, метрика, статистика падінь, реклама), використовують HTTP / HTTPS для роботи з мережею. Існує набір бібліотек, що дозволяють працювати із http-запитами, але у рамках цієї роботи будемо використовувати засоби, доступні в стандартній бібліотеці Java 8, а саме – імплементацію java.net.URLConnection.

Щоб отримати екземпляр URLConnection, потрібно використовувати об’єкт класу java.net.URL, його конструктор приймає тип String, де також повинен бути вказаний протокол. Після цього змінна connection буде зберігати посилання на об’єкт HttpsUrlConnectionImpl. За замовчуванням формуватиметься GET-запит, для того, щоб додати Header, використовуємо метод setRequestProperty(), який приймає key і value.

Для початку реалізуємо виконання GET-запиту, що повертає як результат json-об’єкт. Для тестування скористаємось сервісом http://jsonplaceholder.typicode.com. Так, наприклад, за адресою http://jsonplaceholder.typicode.com/posts/10 сервіс повертає публікацію (post) із id = 10 у json-форматі (рис. 4.1).

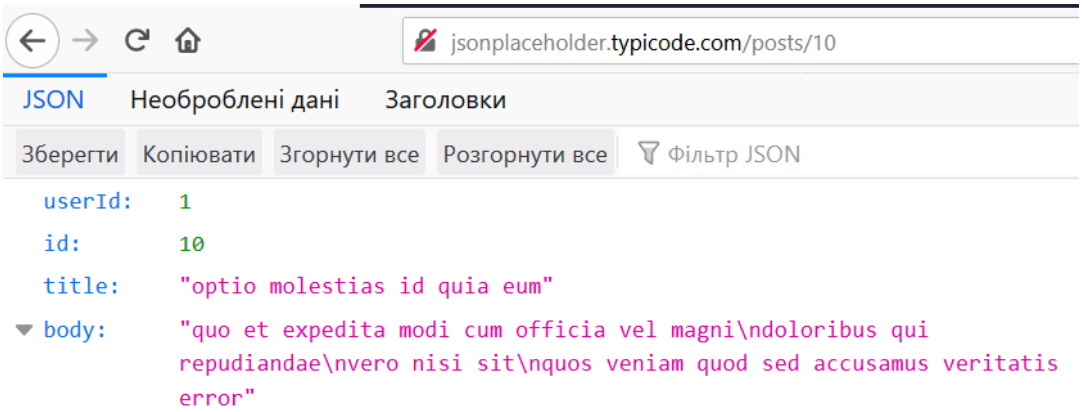



Рисунок 4.1 – Повернення публікації у json-форматі

Виконаємо цей запит із android-застосунку. Починаємо із внесення змін до AndroidManifest.xml. Додаємо дозвіл на підключення до Інтернету android.permission.INTERNET та usesCleartextTraffic="true" (рис. 4.2).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.chnulabs.students">

    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:usesCleartextTraffic="true"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
```

Рисунок 4.2 – Зміни у файлі AndroidManifest.xml

Додаємо нову активність, у якій будемо працювати із http-запитами (рис. 4.3) та кнопку у макеті головної активності, за якою будемо переходити до нової активності (рис. 4.4–4.5).

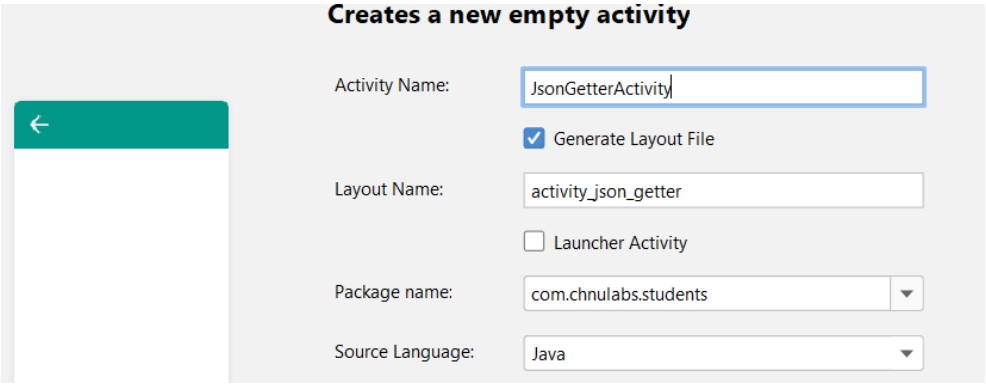



Рисунок 4.3 – Додавання нової активності JsonGetter

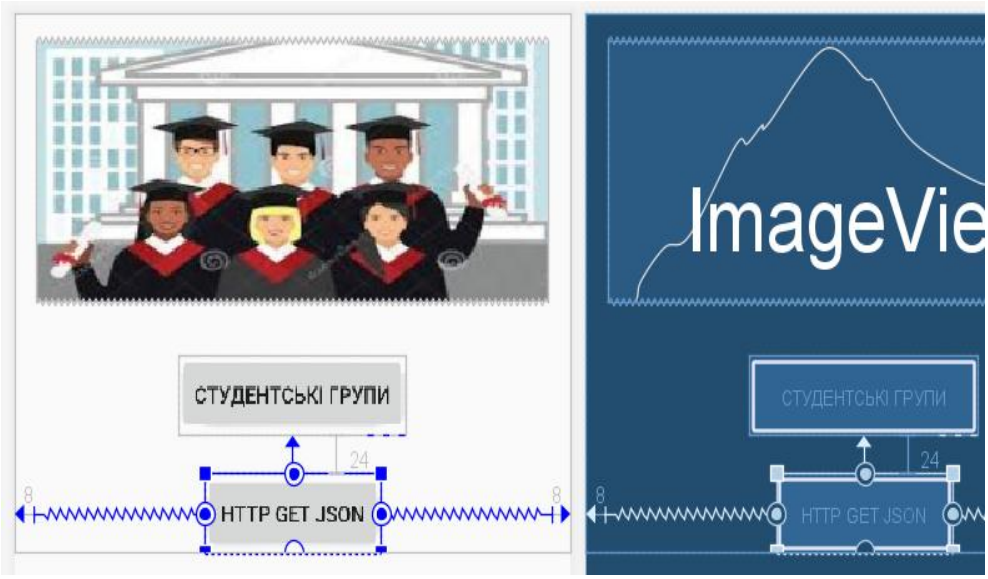



Рисунок 4.4 – Кнопка для переходу на нову активність JsonGetter

```
    public void onGetJsonBtnClick(View view) {
        Intent intent = new Intent( packageContext: this, JsonGetterActivity.class);
        startActivity(intent);
    }
```

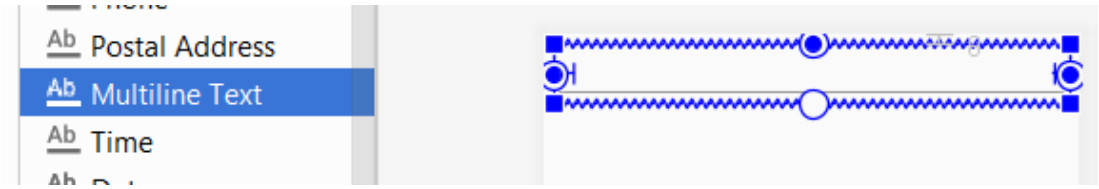
Рисунок 4.5 – Метод головної активності
для переходу до активності JsonGetter

Додаємо клас HttpDataGetter, що буде відповідати за отримання даних за вказаним URL (рис. 4.6). Цей клас імплементуватиме інтерфейс Runnable для можливості подальшої його передачі до виконання в окремий потік. У методі run реалізуємо запит на вказаний URL та повернення результату до приватного поля data нашого класу. Також реалізуємо публічний метод getData, що запускатиме окремий потік та виконуватиме у ньому реалізований у методі run http-запит. Поле data повертатиметься як результат.

```
package com.chnulabs.students;

import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;

public class HttpDataGetter implements Runnable {
    private String data;
    private String url;

    public HttpDataGetter(String url) {
        this.url = url;
    }

    @Override
    public void run() {
        try{
            URL obj = new URL(url);
            HttpURLConnection con = (HttpURLConnection) obj.openConnection();
            con.setRequestMethod("GET");
            con.setRequestProperty("User-Agent", "Mozilla/5.0");
            con.setRequestProperty("Accept-Charset", "UTF-8");
            InputStream response = con.getInputStream();
            Scanner s = new Scanner(response).useDelimiter("\\A");
            data = s.hasNext() ? s.next() : "";
        }
        catch (Exception e) {
            data = e.getMessage();
        }
    }

    public String getData() {
        Thread thread = new Thread( target: this);
        thread.start();
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return data;
    }
}
```

Рисунок 4.6 – Клас HttpDataGetter

Під час завантаження активності JsonGetterActivity виконаємо звернення до URL та отримаємо результат. Для виведення результату додаємо до активності EditText із підтримкою виведення тексту у декілька рядків (рис. 4.7).




```
    <EditText
        android:id="@+id/httpResponse"
        android:layout_width="0dp"
```

Рисунок 4.7 – Додавання EditText для відображення результату запиту

У метод onCreate додаємо код для виконання запиту та виведення результату (рис. 4.8).

```
package com.chnulabs.students;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.EditText;

public class JsonGetterActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_json_getter);
        String res = new HttpDataGetter(
                url: "http://jsonplaceholder.typicode.com/posts/10"
        ).getData();
        EditText editText = findViewById(R.id.httpResponse);
        editText.setText(res);
    }
}
```

Рисунок 4.8 – Виконання запиту та виведення результату

Перевіримо результат, запустимо застосунок та перейдемо до активності JsonGetterActivity (рис. 4.9).

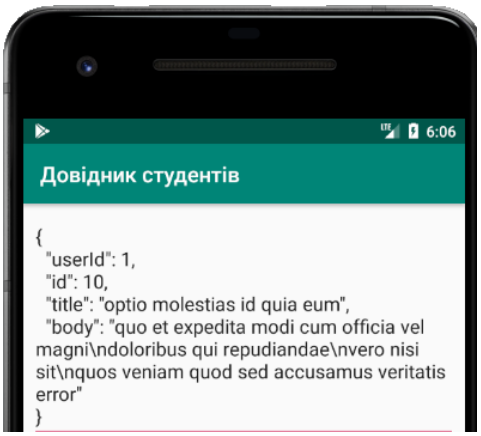



Рисунок 4.9 – Перевірка роботи http-запиту

Далі виконаємо виділення окремих полів публікації та виведення їх на активність. Для цього додаємо до макету чотири окремих поля – користувач, ідентифікатор, тема та текст (рис. 4.10).

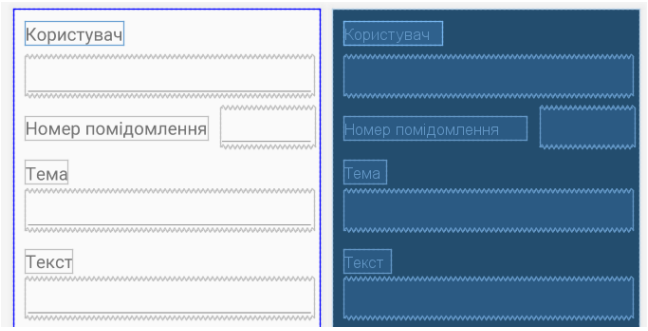



Рисунок 4.10 – Додавання до макету полів для виведення окремих атрибутів

Змінимо метод onCreate активності JsonGetterActivity для відображення даних доданих полів (рис. 4.11).

```
13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_json_getter);
17          String res = new HttpDataGetter(
18                  url: "http://jsonplaceholder.typicode.com/posts/10"
19          ).getData();
20          try {
21              JSONObject obj = new JSONObject(res);
22              EditText txtUser = findViewById(R.id.txtUser);
23              txtUser.setText(obj.getString( name: "userId"));
24              EditText txtId = findViewById(R.id.txtId);
25              txtId.setText(obj.getString( name: "id"));
26              EditText txtTheme = findViewById(R.id.txtTheme);
27              txtTheme.setText(obj.getString( name: "title"));
28              EditText txtText = findViewById(R.id.txtText);
29              txtText.setText(obj.getString( name: "body"));
30          } catch (JSONException e) {
31              e.printStackTrace();
32          }
33      }
```

Рисунок 4.11 – Змінений метод onCreate активності JsonGetterActivity

Перевіримо результат (рис. 4.12).

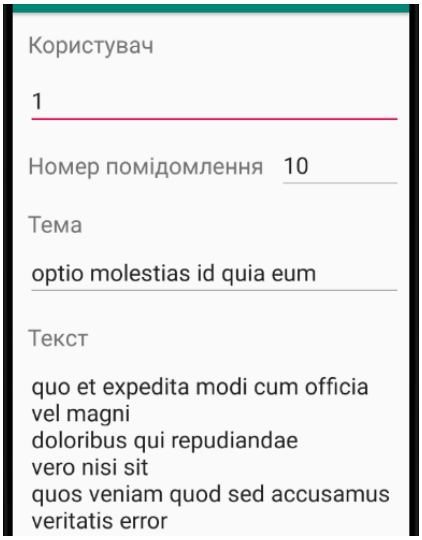



Рисунок 4.12 – Перевірка виведення окремих атрибутів у поля активності

Замість номера користувача, що опублікував повідомлення, непогано було б виводити його ім'я. Для цього отримаємо дані по користувачу із кодом 1, що можна зробити, звернувшись за url http://jsonplaceholder.typicode.com/users/1 (рис. 4.13).

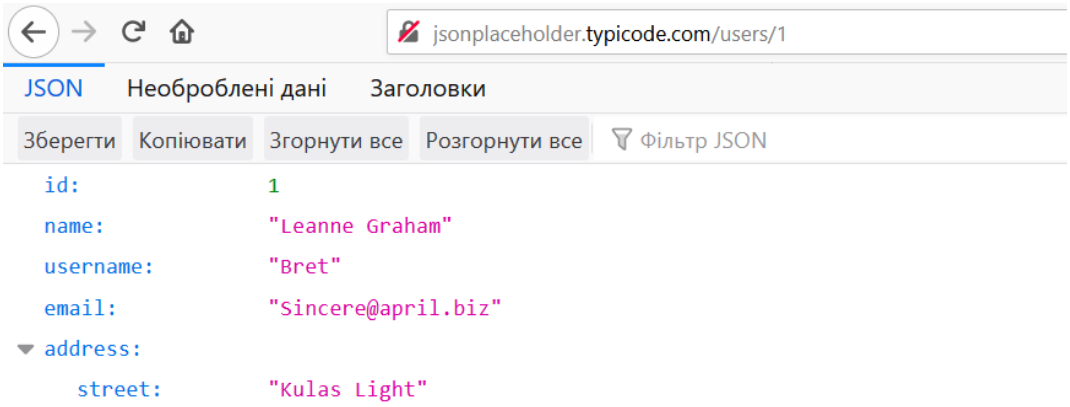



Рисунок 4.13 – Отримання даних користувача

Виконаємо додатковий запит для отримання даних користувача із активності JsonGetterActivity (рис. 4.14) та перевіримо результат (рис. 4.15).

```
28
29          String user_id = obj.getString( name: "userId");
30          String resUser = new HttpDataGetter(
31                  url: "http://jsonplaceholder.typicode.com/users/" + user_id
32          ).getData();
33
34          JSONObject objUser = new JSONObject(resUser);
35          EditText txtUser = findViewById(R.id.txtUser);
36          txtUser.setText(objUser.getString( name: "name"));
```

Рисунок 4.14 – Отримання та виведення імені користувача

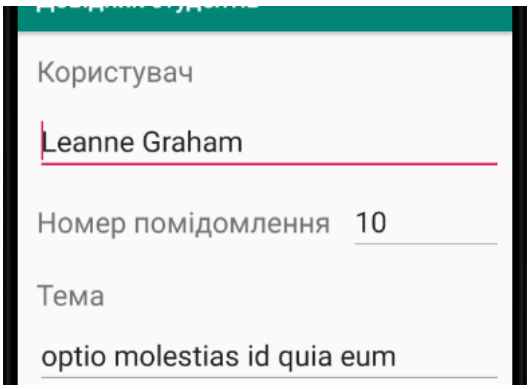
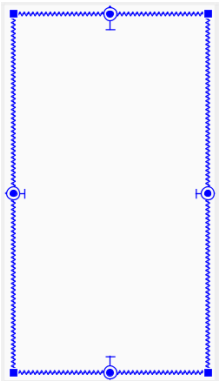



Рисунок 4.15 – Перевірка виведення імені користувача

Тепер перед виводом окремих публікацій реалізуємо вивід списку публікацій, із якого будемо переходити до перегляду детальних даних. Додаємо нову активність PostListActivity, у макеті якої розміщуємо компонент ListView (рис. 4.16).



```
<ListView
    android:id="@+id/postList"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="16dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Рисунок 4.16 – Макет активності activity_posr_list.xml

У класі активності розміщаємо внутрішній клас Post для представлення публікації, список яких буде отримано із json. Реалізуємо метод getListData, що отримуватиме список публікацій через виконання http-запиту на url http://jsonplaceholder.typicode.com/posts та повертатиме результат у вигляді колекції типу Post. І, нарешті, у методі onCreate створимо ArrayAdapter за допомогою якого передамо дані від getListData до ListView (рис. 4.17).

```
1  package com.chnulabs.students;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6  import android.widget.ArrayAdapter;
7  import android.widget.ListView;
8
9  import org.json.JSONArray;
10 import org.json.JSONException;
11 import org.json.JSONObject;
12
13 import java.util.ArrayList;
14
15 public class PostListActivity extends AppCompatActivity {
16
17     class Post {
18         public int id;
19         public String title;
20
21         public Post(int id, String title) {
22             this.id = id;
23             this.title = title;
24         }
25
26         @Override
27         public String toString() {
28             return title;
29         }
30     }
31
32     @Override
33     protected void onCreate(Bundle savedInstanceState) {
34         super.onCreate(savedInstanceState);
35         setContentView(R.layout.activity_post_list);
36
37         ListView listView = (ListView) findViewById(R.id.postList);
```

```
38          ArrayAdapter<Post> adapter = new ArrayAdapter<Post>(
39                  context: this,
40                  android.R.layout.simple_list_item_1,
41                  getListData()
42          );
43          listView.setAdapter(adapter);
44
45      }
46
47      private ArrayList<Post> getListData() {
48          ArrayList<Post> arr = new ArrayList<>();
49          String res = new HttpDataGetter(
50                  url: "http://jsonplaceholder.typicode.com/posts"
51          ).getData();
52          try {
53              JSONArray jsonArray = new JSONArray(res);
54              for(int i=0; i<jsonArray.length(); i++){
55                  JSONObject obj = jsonArray.getJSONObject(i);
56                  arr.add(
57                    new Post(
58                            obj.getInt( name: "id"),
59                            obj.getString( name: "title")
60                    )
61                  );
62              }
63          } catch (JSONException e) {
64              e.printStackTrace();
65          }
66          return arr;
67      }
68  }
```

Рисунок 4.17 – Клас активності PostListActivity

Також у головній активності за натисканням на кнопку «http get json» змінимо код методу для переходу до активності PostListActivity (рис. 4.18). Виконаємо перевірку результату (рис. 4.19).

```
75
76      public void onGetJsonBtnClick(View view) {
77          Intent intent = new Intent( packageContext: this, PostListActivity.class);
78          startActivity(intent);
79      }
```

Рисунок 4.18 – Зміни к класі MainActivity

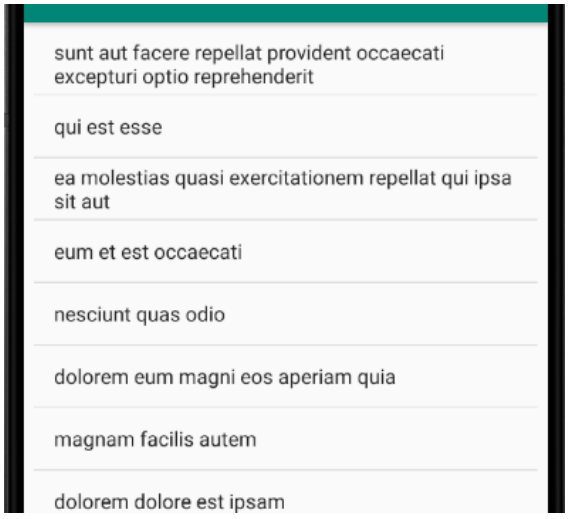



Рисунок 4.19 – Перевірка роботи активності PostListActivity

Наступний крок – зв’язати активність списку публікацій та активність відображення детальних даних однієї публікації. Для цього на активності PostListActivity додаємо слухача для натискання на елементі списку, що ініціюватиме перехід до активності JsonGetterActivity із передачею id обраної публікації (рис. 4.20).

```
AdapterView.OnItemClickListener listener = new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        Post post = (Post) adapterView.getItemAtPosition(i);
        Intent intent = new Intent( packageContext: PostListActivity.this,
                JsonGetterActivity.class);
        intent.putExtra(PostListActivity.POST_ID, post.id);
        startActivity(intent);
    }
};
listView.setOnItemClickListener(listener);
```

Рисунок 4.20 – Додавання слухача у методі onCreate класу PostListActivity

У активності JsonGetterActivity приймаємо переданий id публікації та додаємо його до url http-запиту (рис. 4.21).

```
Intent intent = getIntent();
int postId = intent.getIntExtra(PostListActivity.POST_ID,  defaultValue: 0);

String res = new HttpDataGetter(
        url: "http://jsonplaceholder.typicode.com/posts/"
                + Integer.toString(postId)
).getData();
```

Рисунок 4.21 – Зміни у методі onCreate класу JsonGetterActivity

Перевіримо роботу вищенаведених змін, виконавши перехід від активності PostListActivity до PostListActivity (рис. 22).

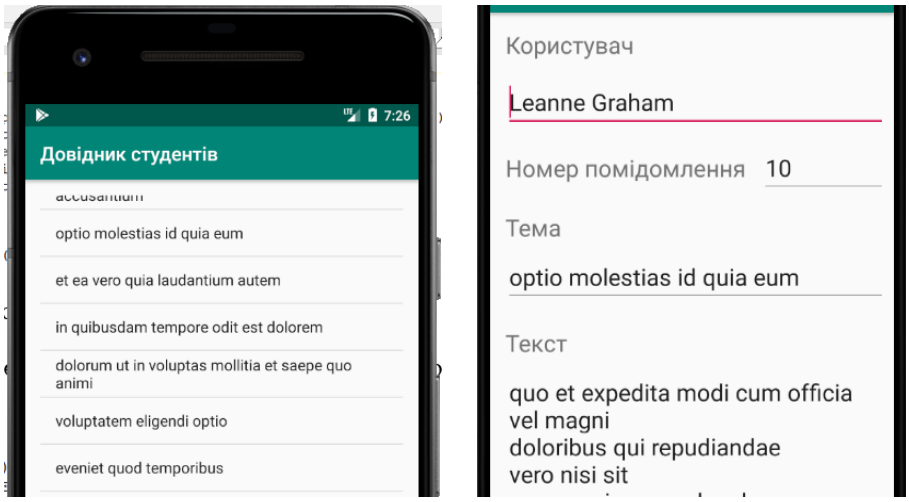



Рисунок 4.22 – Перевірка переходу від списку піблікації до перегляду даних обраної

## 4.2. Робота із власним API-сервісом

У цьому розділі буде розглянуто перенесення БД студентів із SQLite на MySQL server, реалізацію бек-енд для виконання команд БД та взаємодію Android-застосунку із бек-енд (вебсервером) через http-запити.

Буде наведено реалізацію back-end частини на PHP із використанням open server panel для роботи із Apache та MySQL server.

Почнемо із створення БД MySQL. Запустимо phpMyAdmin (рис. 4.23), створимо БД (рис. 4.24) та таблиці Students та Groups (рис. 4.25) тієї ж структури, що і в попередній роботі з SQLite.

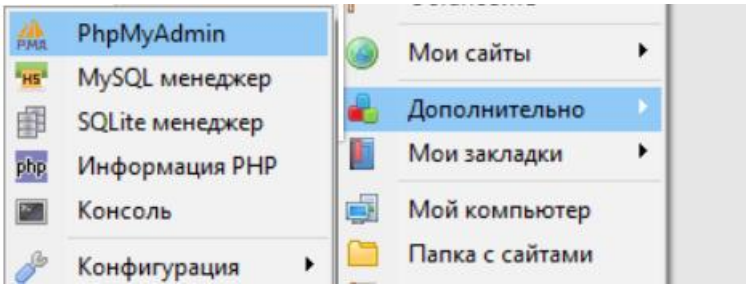



Рисунок 4.23 – Запуск phpMyAdmin із пакета OS panel

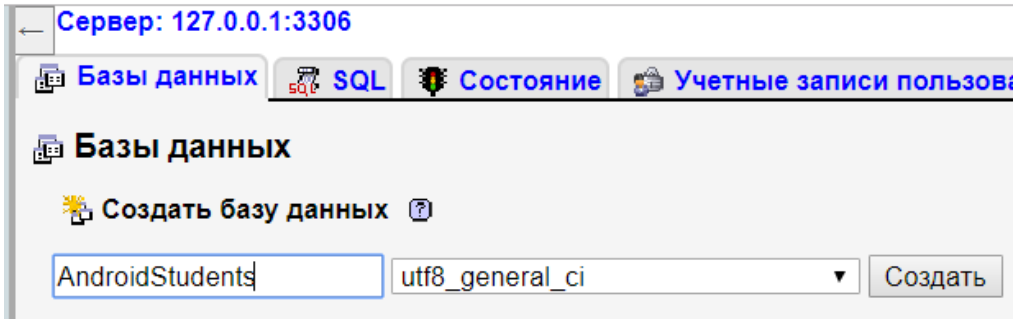



Рисунок 4.24 – Створення БД AndroidStudent

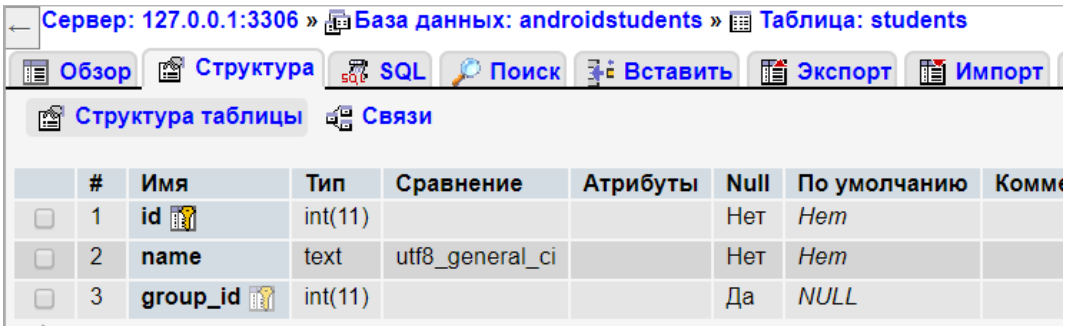
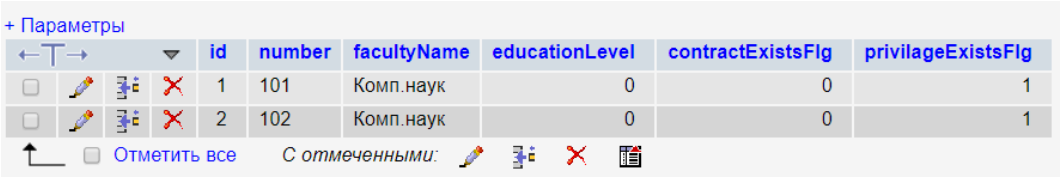




Рисунок 4.25 – Створення таблиць Students та Groups

Додаємо до БД декілька студентських груп для того, щоб простіше було виконувати подальше тестування (рис. 4.26).



+ Параметры

| id | number | facultyName | educationLevel | contractExistsFlg | privilageExistsFlg |
|---|---|---|---|---|---|
| 1 | 101 | Комп.наук | 0 | 0 | 1 |
| 2 | 102 | Комп.наук | 0 | 0 | 1 |

Отметить все  С отмеченными:

Рисунок 4.26 – Результат додавання студентських груп

Виконаємо початкову реалізацію back-end частини для отримання списку студентських груп із БД. Для цього у директорії open server panel domains/localhost створимо папку api, у якій розмістимо 4 файли: index.php, model.php, db.php та config.php. У config.php збережемо дані для підключення до БД MySQL (рис. 4.27).

config.php

```php
<?php
class Config {
    public static $server = "localhost:3306";
    public static $user = "root";
    public static $pwd = "";
    public static $db = "AndroidStudents";
}
```

Рисунок 4.27 – Файл config.php

Файл db.php відповідає за підключення до БД, виконання запиту або команди та повернення результату (рис. 4.28).

db.php

```php
<?php

class DB {
    private $link;
    public $err;
    public function connect() {
        $this->link = new \mysqli(
            \Config::$server, \Config::$user, \Config::$pwd, \Config::$db
        );
        if (!$this->link) {
            return false;
        }
        $this->runQuery( sql: "SET NAMES 'uft-8'");
        return true;
    }
    public function disconnect() {
        $this->link->close();
        unset($this->link);
    }
    public function runQuery($sql) {
        if (!$this->link) {
            $this->connect();
        }
```

```
        $res = $this->link->query($sql);
        if (!$res) {
            $this->err = $this->link->error;
        }
        return $res;
    }
    public function getArrFromQuery($sql) {
        $res_arr = [];
        $rs = $this->runQuery($sql);
        while($row = $rs->fetch_assoc()) {
            $res_arr[] = $row;
        }
        return $res_arr;
    }
}
```

Рисунок 4.28 – Файл db.php

Файл model.php відповідає за вибірку конкретних даних із БД чи виконання конкретних команд, наприклад, отримання списку студентських груп або додавання нового студента (рис. 4.29). На цьому етапі ми реалізуємо лише один метод для отримання списку груп, але надалі цей клас буде розширюватися.

model.php

```
<?php

class Model {
    public static function getGroupsList() {
        return (new DB())->getArrFromQuery(
            sql: "SELECT number, facultyName, educationLevel,
                    contractExistsFlg, privilageExistsFlg
                FROM groups
                order by number");
    }
}
```

Рисунок 4.29 – Файл model.php

І, нарешті, розглянемо index.php, що є своєрідною точкою входу до back-end api. У ньому ми визначаємо, яку саме дію хоче виконати користувач, делегуємо виконання відповідному методу класу model та повертаємо результат користувачу у вигляді json (рис. 4.30). Цей файл також буде надалі розширюватись в міру додавання нових типів запитів від користувача (у нашому випадку – android-застосунку).

index.php

```
<?php
require 'config.php';
require 'db.php';
require 'model.php';

header( string: 'Content-type: application/json');
header( string: 'Access-Control-Allow-Origin: *');
header(
    string: 'Access-Control-Allow-Methods: GET, PUT, POST, DELETE, OPTIONS');
header(
    string: 'Access-Control-Allow-Headers: Origin, Content-Type, X-Auth-Token , Authorization'
);
```

```php
$action = $_REQUEST['action'];
if ($action == 'get_groups_list') {
    $data = Model::getGroupsList();
} else {
    $data = ['err' => 'no action was sent'];
}
echo json_encode($data);
```

Рисунок 4.30 – Файл index.php

Виконаємо тестове отримання даних через інтернет-браузер. Наберемо у адресному рядку localhost/api. Бек-енд повертає відповідь «no action was sent», оскільки ми не передали ніякої дії у запит (рис. 4.31).

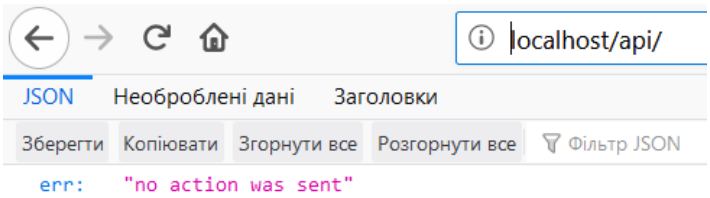



Рисунок 4.31 – Запит на URL localhost/api

Виправимо це, передавши в url action=get_groups_list (рис. 4.32).

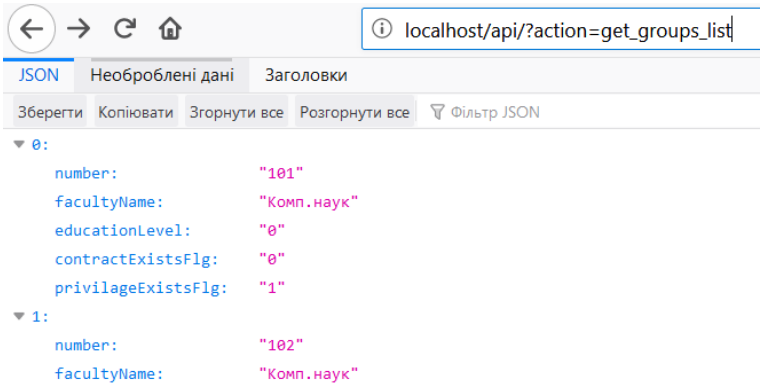



Рисунок 4.32 – Запит на URL localhost/api?action=get_groups_list

Спробуємо виконати аналогічну дію в емуляторі, запустивши інтернет-браузер та звернувшись за вищенаведеним url (рис. 4.33).

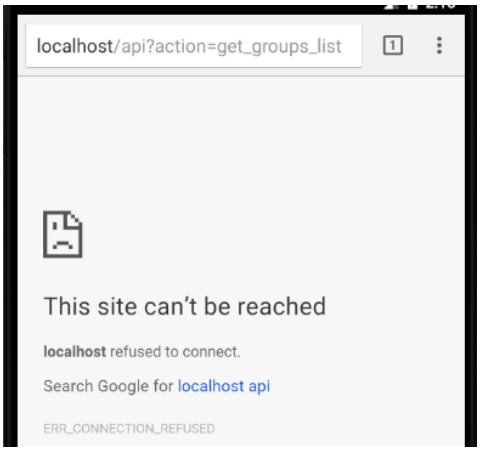



Рисунок 4.33 – Запит на URL localhost/api?action=get_groups_list із браузера на емуляторі

Річ у тім, що спершу в емуляторі звернення до localhost має виконувались через IP 10.0.2.2. Якщо ж ми звертаємось до localhost OSPanel із реального пристрою, замість localhost має бути вказана IP комп'ютера, що у випадку використання OS Windows може бути визначена за допомогою команди ipconfig (рис. 4.34).

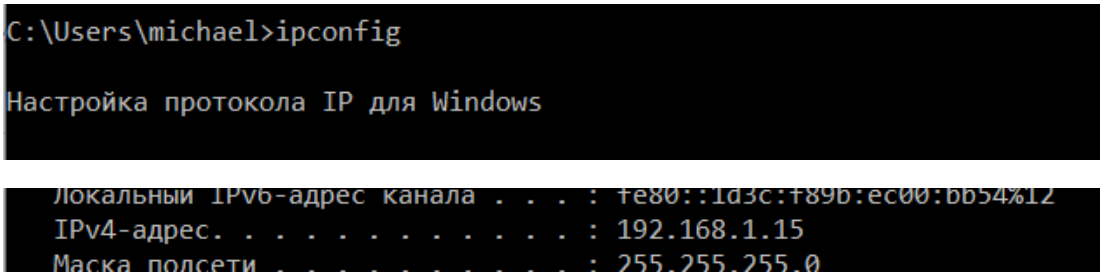



Рисунок 4.34 – Визначення IP комп'ютера

Крім того, необхідно виконати додаткові налаштування open server. Відкриємо вікно налаштувань (рис. 4.35) та встановимо на вкладці «сервер» IP-адреса сервера = Всі доступні IP (рис. 4.36), і на вкладці «аліаси» додамо 2 псевдоніми для localhost – для IP 10.0.2.2 та IP пристрою, який було визначено вище, у нашому випадку 192.168.1.15 (рис. 4.37).

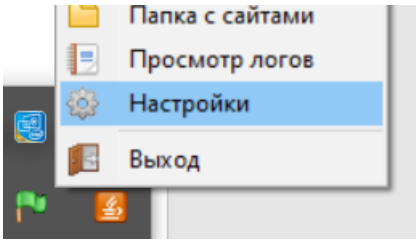



Рисунок 4.35 – Вхід до налаштувань OSPanel

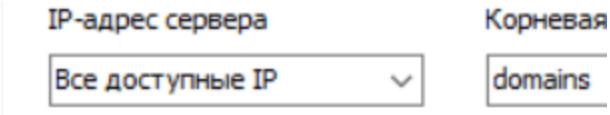



Рисунок 4.36 – Встановлення IP-адреси сервера

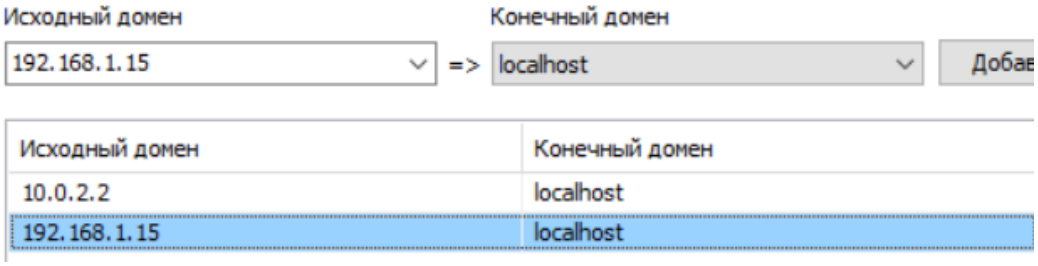



Рисунок 4.37 – Додавання псевдонімів для localhost

Тепер виконавши на емуляторі запит до http://10.0.2.2/api/?action =get_groups_list, отримуємо необхідний результат (рис. 4.38).

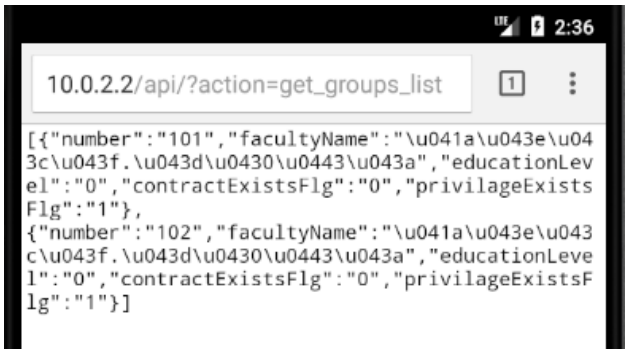



Рисунок 4.38 – Отримання результуючого json в емуляторі

Бек-енд частина також може бути розміщена на хостингу, що дещо спростить звернення до нього із android-застосунку, але реалізація цього підходу виноситься на самостійне виконання.

Повернімося до нашого android-застосунку та реалізуємо отримання даних із БД MySQL під час виведення списку студентських груп. Починаємо із реалізації у класі StudentsGroup методу httpGetGroups, який звертатиметься до бек-енд та привертатиме список студентських груп (рис. 4.39).

```
92      public static ArrayList<StudentsGroup> httpGetGroups() {
93          ArrayList<StudentsGroup> arr = new ArrayList<>();
94          String res = new HttpDataGetter(
95                  url: "http://10.0.2.2/api/?action=get_groups_list"
96          ).getData();
97          try {
98              JSONArray jsonArray = new JSONArray(res);
99              for(int i=0; i<jsonArray.length(); i++){
100                 JSONObject obj = jsonArray.getJSONObject(i);
101                 arr.add(
102                         new StudentsGroup(
103                                 obj.getInt( name: "id"),
104                                 obj.getString( name: "number"),
105                                 obj.getString( name: "facultyName"),
106                                 obj.getInt( name: "educationLevel"),
107                                 (obj.getInt( name: "contractExistsFlg") != 0),
108                                 (obj.getInt( name: "privilageExistsFlg") != 0)
109                         )
110                 );
111             }
112         } catch (JSONException e) {
113             e.printStackTrace();
114         }
115         return arr;
116     }
```

Рисунок 4.39 – Реалізація методу httpGetGroups класу StudentsGroup

Виконаємо звернення до реалізованого методу у методі onStart() активності GroupsListActivity (рис. 4.40).

```
87          ArrayAdapter<StudentsGroup> adapter = new ArrayAdapter<~>(
88                  context: this,
89                  android.R.layout.simple_list_item_1,
90                  StudentsGroup.httpGetGroups()
91          );
```

Рисунок 4.40 – Зміни у методі onStart() активності GroupsListActivity

Перевіримо результат, перейшовши до активності списку студентських груп (рис. 4.41).

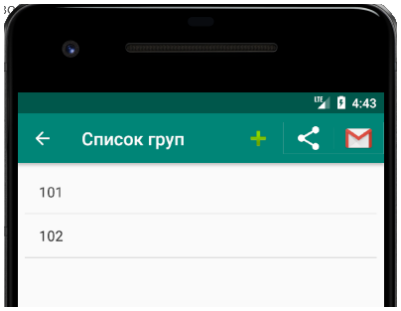



Рисунок 4.41 – Перевірка виведення студентських груп із БД MySQL

Реалізуємо читання детальних даних щодо студентської групи під час переходу до активності StudentsGroupActivity. Починаємо із реалізації відповідного функціоналу з боку back-end. Відкриваємо файл model.php та реалізовуємо метод для повернення студентської групи за її id (рис. 4.42).

```
11
12      public static  function getGroup($id) {
13          return (new DB())->getArrFromQuery(
14                sql: "SELECT id, number, facultyName, educationLevel,
15                      contractExistsFlg, privilageExistsFlg
16                  FROM groups where id=$id");
17      }
```

Рисунок 4.43 – Додавання методу getGroup() у клас Model

Також виконуємо обробку запиту, у відповідь на який маємо повертати студентську групу, у файлі index.php (рис. 4.43).

```
13  $action = $_REQUEST['action'];
14  if ($action == 'get_groups_list') {
15      $data = Model::getGroupsList();
16  } elseif($action == 'get_group' && isset($_REQUEST['group_id'])) {
17      $group = Model::getGroup($_REQUEST['group_id']);
18      if (count($group) > 0) {
19          $data = $group[0];
20      } else {
21          $data = ['err' => 'no group found'];
22      }
23  } else {
24      $data = ['err' => 'no action was sent'];
25  }
26  echo json_encode($data);
```

Рисунок 4.43 – Обробка http-запиту
для отримання даних студентської групи

Тепер повертаємось до android-застосунку та реалізовуємо метод для отримання даних щодо студентської групи у класі StudentsGroup (рис. 4.44).

```
117
118     public static StudentsGroup httpGetGroup(int $id) {
119         StudentsGroup group = null;
120         String res = new HttpDataGetter(
121                 url: "http://10.0.2.2/api/?action=get_group&group_id=" + Integer.toString($id)
122         ).getData();
123         try {
124             JSONObject obj = new JSONObject(res);
```

```
            group = new StudentsGroup(
                                obj.getInt( name: "id"),
                                obj.getString( name: "number"),
                                obj.getString( name: "facultyName"),
                                obj.getInt( name: "educationLevel"),
                                (obj.getInt( name: "contractExistsFlg") != 0),
                                (obj.getInt( name: "privilageExistsFlg") != 0)
                    );
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return group;
    }
```

Рисунок 4.44 – Реалізація методу httpGetGroup у класі StudentsGroup

У методі onCreate() активності StudentsGroupActivity теж виконаємо необхідні зміни. Замість блоку, що відповідає за читання даних із SQLite, просто виконаємо звернення до методу httpGetGroup класу StudentsGroup (рис. 4.45).

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_students_group2);

    Intent intent = getIntent();
    int grpNumber = intent.getIntExtra(GROUP_NUMBER,  defaultValue: 0);
    group = StudentsGroup.httpGetGroup(grpNumber);

    if (group != null) {
        EditText txtGrpNumber = (EditText) findViewById(R.id.grpNumberEdit);
```

Рисунок 4.45 – Зміни у методі onCreate активності StudentsGroupActivity

Запустимо застосунок та виконаємо перевірку переходу до активності детального перегляду даних студентської групи (рис. 4.46).

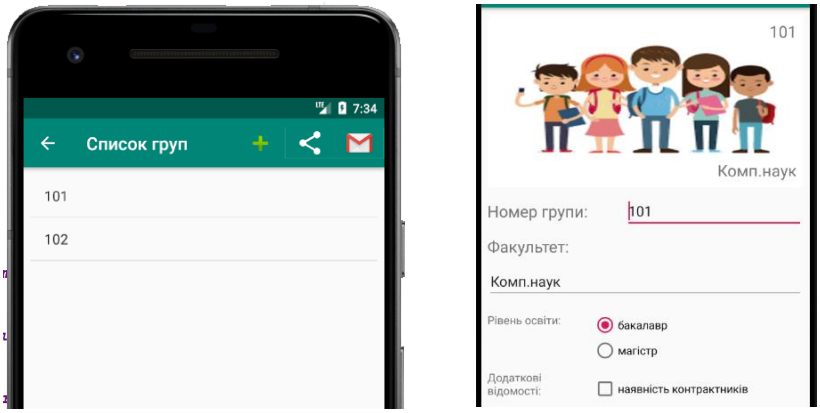



Рис. 4.46. Перехід до активності StudentsGroupActivity.

## 4.3. Post-запити. Організація зміни даних

Продовжимо роботу із даними, що були перенесені до СКБД MySQL та виконаємо реалізацію операції додавання нової студентської групи. Почнемо із модифікації back-end, у клас Model додаємо відповідний метод addGroup (рис. 4.47), у index.php реалізуємо обробку дії action=add_group (рис. 4.48). Відразу зауважимо, що запропонована реалізація back-end частини є максимально спрощеною та не містить валідації даних або перевірки на CSRF-атаки чи SQL-ін’єкції.

```php
    public static function addGroup($group) {
        return (new DB())->runQuery(
                sql: "insert into groups(number, facultyName, educationLevel,
                    contractExistsFlg, privilageExistsFlg)
                    values('".$group['number']."', '".$group['facultyName']."',
                    ".$group['educationLevel']."', ".$group['contractExistsFlg'].",
                    ".$group['privilageExistsFlg']."')");
    }
```

Рисунок 4.47 – Метод addGroup класу Model

```php
} elseif($action == 'add_group' && $_POST) {
    $input = json_decode(file_get_contents( filename: 'php://input'), assoc: true);
    $res = Model::addGroup($input);
    $data = [
       $res
    ];
} else {
```

Рисунок 4.48 – Зміни у index.php

Тепер в андроід-застосунку реалізуємо клас HttpDataPoster, що здійснюватиме виконання post-запиту до нашого back-end із передачею додаткового масиву даних (рис. 4.49).

HttpDataPoster.java

```java
package com.chnulabs.students;

import org.json.JSONObject;

import java.io.InputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;

public class HttpDataPoster implements Runnable {
    private String url;
    private String data;
    private JSONObject sendData;

    public HttpDataPoster(String url, JSONObject sendData) {
        this.url = url;
        this.sendData = sendData;
    }

    @Override
    public void run() {
        try{
            URL obj = new URL(url);
            HttpURLConnection con = (HttpURLConnection) obj.openConnection();
            con.setRequestMethod("POST");
            con.setRequestProperty(
                    "Content-Type", "application/x-www-form-urlencoded; charset=utf-8"
            );
            con.setRequestProperty("Accept", "application/json");
            con.setDoOutput(true);

            OutputStream os = con.getOutputStream();
            os.write(sendData.toString().getBytes( charsetName: "UTF-8"));
            os.close();

            InputStream response = con.getInputStream();
            Scanner s = new Scanner(response).useDelimiter("\\A");
            data = s.hasNext() ? s.next() : "";
        }
        catch (Exception e) {
            data = e.getMessage();
        }
    }

    public String getData() {
        Thread thread = new Thread( target: this);
        thread.start();
        try {
            thread.join();
```

```
52          } catch (InterruptedException e) {
53              e.printStackTrace();
54          }
55          return data;
56      }
57  }
```

Рисунок 4.49 – Клас HttpDataPoster

Переходимо до класу StudentsGroup та реалізуємо метод, що додаватиме поточну студентську групу до БД (рис. 4.50).

```
138     public void httpAddGroup() {
139         StudentsGroup group = null;
140
141         JSONObject data = new JSONObject();
142         try {
143             data.put( name: "number", number);
144             data.put( name: "facultyName", facultyName);
145             data.put( name: "educationLevel", educationLevel);
146             data.put( name: "contractExistsFlg", contractExistsFlg ? 1 : 0);
147             data.put( name: "privilageExistsFlg", privilageExistsFlg ? 1 : 0);
148         } catch (JSONException e) {
149             e.printStackTrace();
150         }
151
152         String res = new HttpDataPoster(
153                 url: "http://10.0.2.2/api/?action=add_group", data
154         ).getData();
155         System.out.println(res);
156     }
```

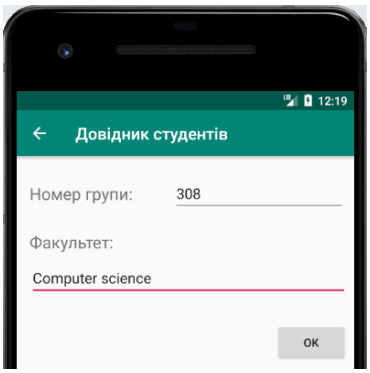
Рисунок 4.50 – Метод httpAddGroup класу StudentsGroup

І, нарешті, виконаємо звернення до цього методу під час натискання на кнопку «ок» у активності AddStudentsGroupActivity (рис 4.51).

```
24      public void onGrpAddClick(View view) {
25          StudentsGroup group = new StudentsGroup(
26                  ((TextView) findViewById(R.id.addGroupNumber)).getText().toString(),
27                  ((TextView) findViewById(R.id.addFaculty)).getText().toString(),
28                  educationLevel: 0,  contractExistsFlg: false,  privilageExistsFlg: false);
29          group.httpAddGroup();
30      }
```

Рисунок 4.51 – Звернення до методу httpAddGroup із активності AddStudentsGroupActivity

Перевіримо роботу функціоналу додавання нової групи (рис. 4.52) та пересвідчимось у її наявності у БД MySQL, переглянувши вміст таблиці groups за допомогою утиліти phpMyAdmin (рис. 4.53).



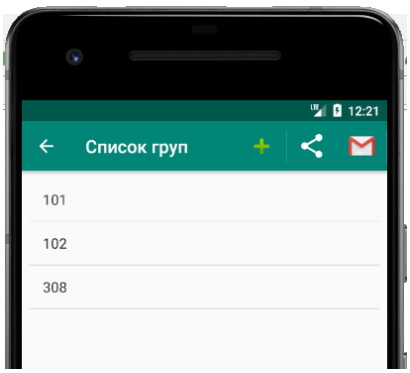
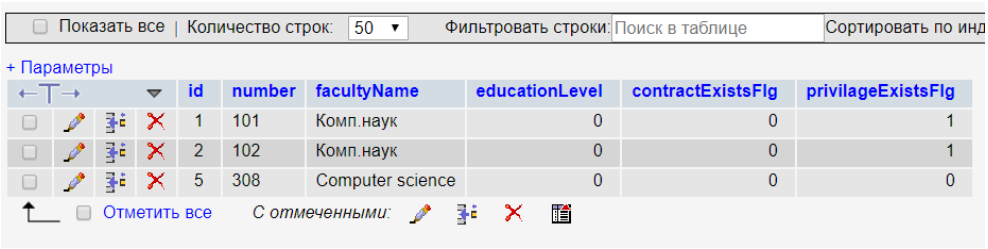




Рисунок 4.52 – Додавання нової групи



Показать все | Количество строк: 50 | Фильтровать строки: Поиск в таблице | Сортировать по инд

+ Параметры

| id | number | facultyName | educationLevel | contractExistsFlg | privilageExistsFlg |
|---|---|---|---|---|---|
| 1 | 101 | Комп.наук | 0 | 0 | 1 |
| 2 | 102 | Комп.наук | 0 | 0 | 1 |
| 5 | 308 | Computer science | 0 | 0 | 0 |

Отметить все С отмеченными:

Рисунок 4.53 – Вміст таблиці groups

Реалізуємо операцію зміни даних студентської групи. Аналогічно попередньому прикладу із додаванням, починаємо із розширення функціоналу back-end у файлах model.php (рис. 4.54) та index.php (рис. 4.55).

```
28      public static function editGroup($group) {
29          return (new DB())->runQuery(
30              sql: "update groups set number = '".$group['number']."',
31                  facultyName = '".$group['facultyName']."',
32                  educationLevel = ".$group['educationLevel'].",
33                  contractExistsFlg = ".$group['contractExistsFlg'].",
34                  privilageExistsFlg = ".$group['privilageExistsFlg']."
35                  where id = " . $group['id']);
36      }
```

Рисунок 4.54 – Зміни у model.php

```
29  } elseif($action == 'edit_group' && $_POST) {
30      $input = json_decode(file_get_contents( filename: 'php://input'), assoc: true);
31      $res = Model::editGroup($input);
32      $data = [
33          $res
34      ];
35  }else {
```

Рисунок 4.55 – Зміни у index.php

У андроід-застосунку додаємо до класу StudentsGroup метод httpSaveGroup (рис. 4.56).

```
156
157     public void httpSaveGroup() {
158         StudentsGroup group = null;
159
160         JSONObject data = new JSONObject();
161         try {
162             data.put( name: "id", id);
163             data.put( name: "number", number);
164             data.put( name: "facultyName", facultyName);
165             data.put( name: "educationLevel", educationLevel);
166             data.put( name: "contractExistsFlg", contractExistsFlg ? 1 : 0);
167             data.put( name: "privilageExistsFlg", privilageExistsFlg ? 1 : 0);
168         } catch (JSONException e) {
169             e.printStackTrace();
170         }
171
172         String res = new HttpDataPoster(
173                 url: "http://10.0.2.2/api/?action=edit_group", data
174         ).getData();
175     }
```

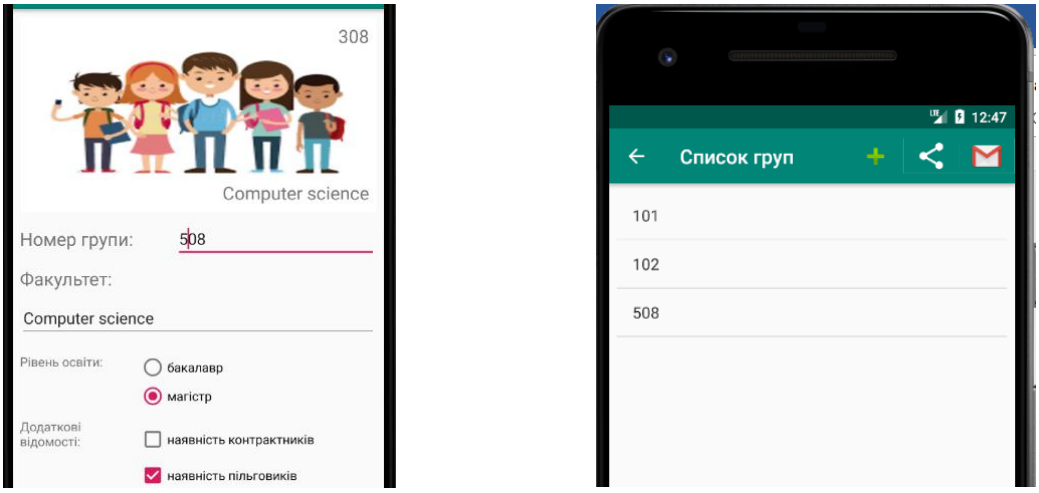
Рисунок 4.56 – Метод httpSaveGroup класу StudentsGroup

Виклик нового методу поміщуємо у активності StudentsGroupActivity натисканням на кнопку «ок» (рис. 4.57).

```
65      public void onOkBntClick(View view) {
66          Intent intent = getIntent();
67          int grpNumber = intent.getIntExtra(GROUP_NUMBER,  defaultValue: 0);
68
69          StudentsGroup group = new StudentsGroup(grpNumber,
70                  ((TextView) findViewById(R.id.grpNumberEdit)).getText().toString(),
71                  ((TextView) findViewById(R.id.facultyEdit)).getText().toString(),
72                  ((RadioButton) findViewById(R.id.edu_level_master)).isChecked()?1:0,
73                  ((CheckBox) findViewById(R.id.contract_flg)).isChecked(),
74                  ((CheckBox) findViewById(R.id.privilege_flg)).isChecked());
75          group.httpSaveGroup();
76
77          NavUtils.navigateUpFromSameTask( sourceActivity: this);
78      }
```

Рисунок 4.57 – Виклик методу httpSaveGroup у класі активності StudentsGroupActivity

Перевіримо роботу зміни групи, змінивши декілька полів та переглянувши результат (рис. 4.58).




| id | number | facultyName | educationLevel | contrac |
|---|---|---|---|---|
| 1 | 101 | Комп.наук | 0 | |
| 2 | 102 | Комп.наук | 0 | |
| 5 | 508 | Computer science | 1 | |

Рисунок 4.58 – Зміна полів студентської групи

Аналогічним чином реалізуємо видалення студентської групи. Спочатку на back-end у model.php (рис. 4.59) та index.php (рис. 4.60), а потім у застосунку у класі StudentsGroup (рис. 4.61) та в активності StudentsGroupActivity (рис. 4.62). Після виконання наведених нижче змін, перевірити роботу операції видалення студентської групи.

```
38      public static function revomeGroup($group) {
39          return (new DB())->runQuery(
40              sql: "delete from groups
41                  where id = " . $group['id']);
42      }
```

Рисунок 4.59 – Зміни у model.php

```php
} elseif($action == 'remove_group' && $_POST) {
    $input = json_decode(file_get_contents( filename: 'php://input'), assoc: true);
    $res = Model::revomeGroup($input);
    $data = [
        $res
    ];
```

Рисунок 4.60 – Зміни у index.php

```
public static void httpRemoveGroup(int id) {
    JSONObject data = new JSONObject();
    try {
        data.put( name: "id", id);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    String res = new HttpDataPoster(
            url: "http://10.0.2.2/api/?action=remove_group", data
    ).getData();
}
```

Рисунок 4.61 – Метод httpRemoveGroup класу StudentsGroup

```
public void onDelete(View view) {
    Intent intent = getIntent();
    int grpNumber = intent.getIntExtra(GROUP_NUMBER,  defaultValue: 0);

    StudentsGroup.httpRemoveGroup(grpNumber);

    NavUtils.navigateUpFromSameTask( sourceActivity: this);
}
```

Рисунок 4.62 – Метод активності StudentsGroupActivity

Реалізуємо просту аутентифікацію користувачів у нашому застосунку. Починаємо як завжди із back-end. До БД MySQL додаємо табличку users (рис. 4.63).

| | # | Имя | Тип | Сравнение | Атрибуты | Null | По ум |
|---|---|---|---|---|---|---|---|
| ☐ | 1 | id | int(11) | | | Нет | *Нет* |
| ☐ | 2 | username | varchar(150) | utf8_general_ci | | Нет | *Нет* |
| ☐ | 3 | passwd | varchar(150) | utf8_general_ci | | Нет | *Нет* |
| ☐ | 4 | token | varchar(150) | utf8_general_ci | | Нет | *Нет* |

Рисунок 4.63 – Таблиця users БД MySQL

Реалізуємо клас Auth, що відповідатиме за авторизацію користувачів. Зробимо це в окремому файлі auth.php (рис. 4.64).

auth.php

```php
<?php

class Auth {
    public static function getUserToken($user) {
        $res = (new DB())->getArrFromQuery(
            sql: "select id from users where username='".$user['username']."'
            and passwd=md5('".$user['passwd']."')"
        );
        if (count($res) > 0) {
            $id = $res[0]['id'];
            $token = bin2hex(random_bytes( length: 64));
            $res2 = (new DB())->runQuery(
                sql: "update users set token='$token' where id=$id"
            );
```

```php
            if ($res2) {
                return $token;
            }
        }
        return null;
    }

    public static function checkToken($user) {
        $res = (new DB())->getArrFromQuery(
            sql: "select id from users where token='".$user['token']."'"
        );
        if (count($res) > 0) {
            return true;
        }
        return false;
    }
}
```

Рисунок 4.64 – Клас авторизації користувачів

Клас містить два методи – getUserToken та checkToken. Перший перевіряє, чи існує у БД користувач із вказаним ім’ям користувача та паролем. Якщо так, генерується унікальний ключ, записується у БД та повертається як результат роботи методу. Інакше повертається null. У подальшому авторизація користувача під час виконання запитів до back-end буде виконуватись із використанням згенерованого ключа. Другий метод слугує для перевірки переданого користувачем ключа, і повертає істину, якщо ключ валідний, а інакше – хибність.

Також доопрацьовуємо index.php, у якому додаємо обробку для action=login та виконуємо перевірку отриманого від користувача ключа на валідність (рис. 4.65).

index.php

```php
<?php
require 'config.php';
require 'db.php';
require 'model.php';
require 'auth.php';

header( string: 'Content-type: application/json');
header( string: 'Access-Control-Allow-Origin: *');
header(
    string: 'Access-Control-Allow-Methods: GET, PUT, POST, DELETE, OPTIONS');
header(
    string: 'Access-Control-Allow-Headers: Origin, Content-Type, X-Auth-Token , Authorization'
);
$action = $_REQUEST['action'];
$input = json_decode(file_get_contents( filename: 'php://input'), assoc: true);
if ($action == 'get_groups_list') {
    if (Auth::checkToken($_REQUEST)) {
        $data = Model::getGroupsList();
    } else {
        $data = ['err' => 'permission denied'];
    }
} elseif($action == 'get_group' && isset($_REQUEST['group_id'])) {
    if (Auth::checkToken($_REQUEST)) {
        $group = Model::getGroup($_REQUEST['group_id']);
    } else {
        $data = ['err' => 'permission denied'];
    }
    if (count($group) > 0) {
        $data = $group[0];
    } else {
        $data = ['err' => 'no group found'];
    }
```

```php
} elseif($action == 'add_group' && $_POST) {
    if (Auth::checkToken($input)) {
        $res = Model::addGroup($input);
    } else {
        $data = ['err' => 'permission denied'];
    }
    $data = [
       $res
    ];
} elseif($action == 'edit_group' && $_POST) {
    if (Auth::checkToken($input)) {
        $res = Model::editGroup($input);
    } else {
        $data = ['err' => 'permission denied'];
    }
    $data = [
        $res
    ];
} elseif($action == 'remove_group' && $_POST) {
    if (Auth::checkToken($input)) {
        $res = Model::revomeGroup($input);
    } else {
        $data = ['err' => 'permission denied'];
    }
    $data = [
        $res
    ];
} elseif($action == 'login' && $_POST) {
    $token = Auth::getUserToken($input);
    $data = [
        "token" => $token
    ];
} else {
    $data = ['err' => 'no action was sent'];
}
echo json_encode($data);
```

Рисунок 4.65 – Зміни у index.php

Виконаємо необхідні зміни в android-застосунку. Почнімо із додавання нової активності для логін-форми (рис. 4.66).

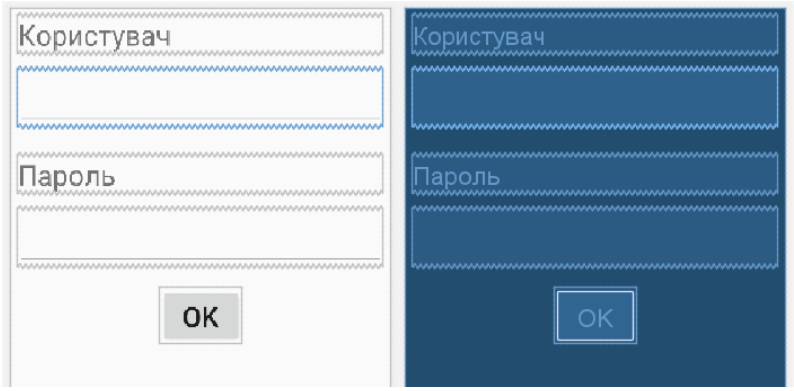



Рисунок 4.66 – Макет активності логін-форми

Натисканням на кнопку ok виконуватимемо відправку на бек-енд імені користувача та пароля для виконання аутентифікації. Після отримання відповіді перевіряємо, якщо token не порожній, запам’ятовуємо його та переходимо до головної активності застосунку (рис. 4.67).

```
23      public void onLogin(View view) {
24          JSONObject data = new JSONObject();
25          try {
26              data.put( name: "username", ((EditText)findViewById(R.id.user)).getText());
27              data.put( name: "passwd", ((EditText)findViewById(R.id.pass)).getText());
28          } catch (JSONException e) {
29              e.printStackTrace();
30          }
31          String strToken = new HttpDataPoster(
32                  url: "http://10.0.2.2/api/?action=login", data
33          ).getData();
34          JSONObject tokenData = null;
35          String token = null;
36          try {
37              tokenData = new JSONObject(strToken);
38              token = tokenData.getString( name: "token");
39          } catch (JSONException e) {
40              e.printStackTrace();
41          }
42          if (token.toLowerCase() == "null") {
43              Toast.makeText( context: this,  text: "Password is incorrect !",
44                      Toast.LENGTH_LONG).show();
45           } else {
46              UserData.token = token;
47              Intent intent = new Intent( packageContext: this, MainActivity.class);
48              startActivity(intent);
49              finish();
50           }
51      }
```

Рисунок 4.67 – Метод onLogin класу LoginActivity

Реалізуємо клас UserData, у якому зберігатиметься значення ключа token для можливості його подальшого використання в інших активностях (рис. 4.68).

```
3   public class UserData {
4       public static String token;
5   }
```

Рисунок 4.68 – Клас UserData

Тепер під час кожного звернення до back-end нам необхідно передавати збережений у класі UserData ключ token. Для цього змінимо класи Http DataGetter (рис. 4.69) та HttpDataPoster (рис. 4.70), а саме конструктори цих класів, де під час формування запиту будемо додавати до нього ключ token.

```
12      public HttpDataGetter(String url) {
13          this.url = url + "&token=" + UserData.token;
14      }
```

Рисунок 4.69 – Конструктор класу HttpDataGetter

```
18      public HttpDataPoster(String url, JSONObject sendData) {
19          this.url = url;
20          this.sendData = sendData;
21          try {
22              this.sendData.put( name: "token", UserData.token);
23          } catch (JSONException e) {
24              e.printStackTrace();
25          }
26      }
```

Рисунок 4.70 – Конструктор класу HttpDataPoster

Також виконаємо зміни у AndroidManifest.xml для того, щоб першою завантажувалася активність LoginActivity (рис. 4.71).

```
17      <activity android:name=".LoginActivity">
18          <intent-filter>
19              <action android:name="android.intent.action.MAIN" />
20              <category android:name="android.intent.category.LAUNCHER" />
21          </intent-filter>
22      </activity>

36      <activity android:name=".MainActivity">
37      </activity>
```

Рисунок 4.71 – Зміни у AndroidManifest.xml

Запускаємо застосунок і виконуємо перевірку входу та виконання подальших операцій перегляду, а також зміни даних.

# *ЗАВДАННЯ ДЛЯ ІНДИВІДУАЛЬНОГО ВИКОНАННЯ*

***Завдання №1.*** Створити новий проєкт, реалізувати під час запуску застосунку виведення у макеті активності вашого прізвища та номера групи.

***Завдання №2.*** Реалізувати обробку події onClick для кнопки у макеті активності. Реалізувати виведення у компонент spinner списку рядків із strings-array файлу strings.xml. Реалізувати колекцію елементів власного класу та організувати пошук елемента за обраним значенням у spinner, виведення до макета активності значень його атрибутів.

***Завдання №3.*** Додати до власного проєкту другу активність та реалізувати її виклик із першої (натисканням кнопки) із використанням інтенту. Організувати передачу даних із першої активності до другої із подальшим їх отриманням і відображенням у макеті другої активності. Реалізувати запуск активності стороннього застосунку та передачу даних із власного застосунку до них (наприклад, відправку пошти або повідомлення).

***Завдання №4.*** Реалізувати застосунок, у якому продемонструвати можливість втрати значень локальних змінних під час повороту пристрою. Реалізувати таймер, що буде залежно від відповідного прапорця налаштувань зупинятися (або продовжувати роботу) у разі втрати активністю фокусу. Забезпечити збереження значення таймера під час повороту пристрою.

***Завдання №5.*** Реалізувати активність із використанням layout-ів LinearLayout, FrameLayout та TableLayout. Також використати компоненти ImageView, EditText, RadioButton Checkbox. Натиснувши на кнопку, вивести всі введені значення у макеті активності у повідомлення toast.

***Завдання №6.*** Реалізувати макет для активності із попередньої роботи, аналогічний за виглядом макета завдання 5, але із використанням layout-у ConstraintLayout. Переключити активність на використання нового макета.

***Завдання №7.*** Реалізувати у своєму застосунку активності верхнього рівня, категорій та деталізації. Для відображення категорій використати компонент ListView. Реалізувати OnItemClickListener для компонента ListView. Використати ArrayAdapter під час відображення колекції елементів у компоненті ListView.

***Завдання №8.*** Реалізувати активність додавання елемента списку (екземпляра колекції). У власному класі реалізувати для цього відповідний метод. Створити меню та розмістити у ньому елемент для виклику створеної активності. Додати меню до панелі застосунку у активності категорій. Обробити подію натискання на елемент меню для завантаження активності. Використати атрибут parentActivityName AndroidManifest для виведення кнопки повернення до батьківської активності. Створити елемент меню із shareActionProvider, що відправлятиме повідомлення.

***Завдання №9.*** У власному застосунку реалізувати збереження даних у БД та подальшу роботу із ними (читання та відображення у макетах активностей списку та елемента) для однієї сутності (таблиці) на вибір.

***Завдання №10.*** У власному застосунку реалізувати всі операції CRUD мінімум для двох зв'язаних таблиць БД SQLite.

***Завдання №11.*** Реалізувати активність категорій зі списком користувачів та організувати перехід до списку справ користувача (отримати список справ користувача із кодом 1 можна за посиланням https://jsonplaceholder.typicode.com/users/1/todos). Організувати фільтрацію за статусом справи – виконана або невиконана (поле completed).

***Завдання №12.*** Для власного застосунку реалізувати БД під управлінням клієнт-серверної реляційної СКБД (MySQL або ін.) та бек-енд (мова програмування – на вибір) для обробки клієнтських http-запитів і взаємодії із БД. З боку андроїд-застосунку реалізувати отримання даних із БД та їх відображення у макетах активностей.

***Завдання №13.*** Для власного застосунку реалізувати операції додавання, зміни та видалення даних під час роботи із БД під управлінням клієнт-серверної реляційної СКБД (MySQL або ін.) для 2-х таблиць БД. Реалізувати аутентифікацію та авторизацію користувачів у процесі виконання операцій із даними.

***Завдання №14.*** Реалізувати мінімум 3 ролі користувачів – читач, редактор та адміністратор. Перевіряти права доступу користувача під час виконання операцій зміни даних. Реалізувати механізм зміни пароля для поточного користувача. Для ролі «адміністратор» реалізувати активність (активності) для редагування даних користувачів (перегляд, додавання, зміна ролі, видалення).

# *РЕКОМЕНДОВАНА ЛІТЕРАТУРА*


1. Goransson A. Efficient Android Threading: Asynchronous Processing Techniques for Android Applications. O'Reilly Media : 1-st ed. June 13, 2014. 280 p.
2. Meier R. Professional Android, 4-th ed. Wrox, 2018. 928 p.
3. Phillips B., Stewart C., Marsicano K. Android Programming: The Big Nerd Ranch Guide : 3-rd ed. February 9, 2017. 624 p.
4. Zechne M. Beginning Android Games : 3-rd ed. Apress, 2016. 619 p.
5. Голощапов А. Л. Google Android. Создание приложений для смартфонов и планшетных ПК, БХВ-Петербург, 2014. 928 с.
6. Гриффитс Д. Head First. Программирование для Android : 2-е изд. Питер, 2018, 912 с.
7. Дарвин Я. Ф. Android. Сборник рецептов: задачи и решения для разработчиков приложений. Диалектика-Вильямс, 2018. 768 с.
8. Дейтел П., Дейтел Х. Андроид для разработчиков : 3-е изд. Питер, 2016. 512 с.
9. Филлипс Б., Стюарт К., Марсикано К. Android. Программирование для профессионалов : 3-е изд. Питер Пресс, 2017. 688 с.
10. Хорстманн К., Корнелл Г. Библиотека профессионала Java. Т. 1. Вильямс, 2014. 864 с.
11. Хорстманн К., Корнелл Г. Библиотека профессионала Java. Т. 2. Вильямс, 2014. 1008 с.
12. Шилдт Г. Java Полное руководство. 10-е изд. Диалектика, 2018. 720 с.

## *Для нотаток*

## *Для нотаток*

## *Для нотаток*

**Навчальне видання**

**Михайло Леонідович**
**ДВОРЕЦЬКИЙ,**

**Юрій Олексійович**
**НЕЗДОЛІЙ,**

**Світлана Володимирівна**
**ДВОРЕЦЬКА,**

**Ігор Олександрович**
**КАНДИБА**

# РОЗРОБКА МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ OS Android

*Навчальний посібник*

*Для підготовки бакалаврів у галузі знань*
*«Інформаційні технології»*

---

Редактор *А. Бурмус.*
Технічний редактор, комп'ютерна верстка *Н. Кардаш*
Дизайн обкладинки *М. Дворецький*.
Друк *С. Волинець*. Фальцювально-палітурні роботи *О. Мішалкіна.*

Підп. до друку 11.02.2021
Формат 60x84$^{1}/_{16}$. Папір офсет.
Гарнітура «Times New Roman». Друк ризограф.
Ум. друк. арк. 8,14. Обл.-вид. арк. 29,97.
Тираж 300 пр. Зам. № 6194.

54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50–03–32, 8 (0512) 76–55–81, e-mail: rector@chmnu.edu.ua.
Свідоцтво суб'єкта видавничої справи ДК № 6124 від 05.04.2018.