МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Ректор, професор

_____ Олександр ШЕВЧЕНКО

(підпис)

«__» _____ **2025 р.**

РОЗРОБЛЕННЯ МОБІЛЬНИХ ДОДАТКІВ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторних робіт
для здобувачів освітнього ступеня «бакалавр»
спеціальності 122 «Комп'ютерні науки»
освітньо-професійної програми
«Комп'ютерні науки»
денної та заочної форм здобуття освіти

Всі цитати, цифровий та фактичний матеріал, бібліографічні відомості перевірені.
Написання одиниць відповідає стандартам

Підпис укладача:

_____ Микола КОСТІКОВ

«__17__» __квітня__2025 р.

СХВАЛЕНО
на засіданні кафедри
інформаційних технологій,
штучного інтелекту і
кібербезпеки
Протокол №11
від 18.04.2025 р.

Реєстраційний номер електронних методичних рекомендацій у НМВ
50.156-2025

КИЇВ НУХТ 2025

Розроблення мобільних додатків [Електронний ресурс]: методичні рекомендації до виконання лабораторних робіт для здобувачів освітнього ступеня «бакалавр» спеціальності 122 «Комп'ютерні науки» освітньо-професійної програми «Комп'ютерні науки» денної та заочної форм здобуття освіти / уклад. : М. П. Костіков. – К.: НУХТ, 2025. – 43 с.

Рецензент: Сергій ГРИБКОВ, д-р техн. наук, проф. _____

Укладач: Микола КОСТІКОВ, канд. техн. наук, доц. _____

Відповідальний за випуск: Сергій ГРИБКОВ, д-р техн. наук, проф. _____

Подано в авторській редакції

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

Лабораторний практикум охоплює усі змістові модулі навчальної програми дисципліни «Розроблення мобільних додатків» та призначений для здобувачів вищих навчальних закладів, що навчаються за спеціальністю 122 «Комп'ютерні науки» освітньо-професійної програми «Комп'ютерні науки» денної та заочної форм здобуття освіти.

Метою виконання лабораторних робіт є закріплення у здобувачів знань і вмінь із дисципліни «Розроблення мобільних додатків», набуття навичок використання сучасних технологій розроблення мобільних додатків для подальшого їх використання у своїй професійній діяльності.

Завданням лабораторного практикуму є допомога здобувачам в опануванні методів розроблення мобільних додатків, які розглядаються в рамках навчальної дисципліни. В цьому практикумі викладено загальні рекомендації щодо використання сучасних технологій створення мобільних додатків, наведено короткі теоретичні відомості, завдання та індивідуальні варіанти для виконання лабораторних робіт, запитання для самоконтролю та рекомендовану літературу.

Лабораторні роботи з дисципліни «Розроблення мобільних додатків» розроблено відповідно до робочої програми навчальної дисципліни.

Методичні вказівки складаються з трьох лабораторних робіт, до яких наведено завдання та індивідуальні варіанти, а також у теоретичній частині послідовно описано різні типи завдань із розроблення мобільних додатків, розглянутих у рамках навчальної дисципліни.

Виконання кожної лабораторної роботи передбачає ознайомлення здобувачів із методичними вказівками та теоретичну підготовку з відповідних розділів дисципліни «Розроблення мобільних додатків».

Для виконання лабораторних робіт при створенні програмних засобів можна використовувати довільні сучасні мови програмування (зокрема Kotlin, Java тощо), а також довільні середовища розроблення програмних засобів (IDE).

Для здачі лабораторної роботи здобувач має оформити звіт, який містить:

- титульний аркуш;
- формулювання завдання;
- індивідуальний варіант;
- опис ходу виконання роботи;
- висновки.

Бали, отримані за виконання лабораторних робіт, підсумовуються та враховуються при виставленні підсумкової оцінки з навчальної дисципліни.

2. ПРАВИЛА ТЕХНІКИ БЕЗПЕКИ ПРИ ВИКОНАННІ ЛАБОРАТОРНИХ РОБІТ

Загальні положення

1. До роботи в комп'ютерному класі допускаються особи, ознайомлені з даною інструкцією з техніки безпеки та правил поведінки.

2. Робота здобувачів у комп'ютерному класі дозволяється лише у присутності викладача (інженера, лаборанта).

3. Під час занять сторонні особи можуть знаходитися в класі лише з дозволу викладача.

4. Під час перерв між парами проводиться обов'язкове провітрювання комп'ютерного кабінету з обов'язковим виходом здобувачів з нього.

Перед початком роботи необхідно:

1. Переконатися у відсутності видимих пошкоджень на робочому місці.

2. Включити комп'ютери та налагодити роботу.

При роботі в комп'ютерному класі забороняється:

1. Знаходитися в класі у верхньому одязі.

2. Класти одяг і сумки на столи.

3. Знаходитися в класі з напоями та їжею.

4. Розташовуватися збоку або ззаду від включеного монітора.

5. Приєднувати або від'єднувати кабелі, чіпати роз'єми, дроти і розетки.

6. Пересувати комп'ютери і монітори.

7. Відкривати системний блок.

8. Вмикати і вимикати комп'ютери самостійно.

9. Намагатися самостійно усувати несправності в роботі апаратури.

10. Перекривати вентиляційні отвори на системному блоці та моніторі.

11. Ударяти по клавіатурі, натискувати безцільно на клавіші.

12. Приносити і запускати комп'ютерні ігри.

Перебуваючи в комп'ютерному класі, здобувачі зобов'язані:

1. Дотримуватись тиші і порядку.

2. Виконувати вимоги викладача та інженера/лаборанта.

3. Дотримуватись режиму роботи.

4. Після роботи завершити всі активні програми і коректно вимкнути комп'ютер.

5. Залишити робоче місце чистим.

Необхідно дотримуватись правил:

1. Відстань від екрану до очей — 70–80 см.

2. Вертикально пряма спина.

3. Плечі опущені і розслаблені.

4. Ноги на підлозі і не схрещені.

5. Лікті, зап'ястя і кисті рук на одному рівні.

Вимоги безпеки в аварійних ситуаціях:

1. При появі програмних помилок або збоях устаткування здобувач повинен негайно звернутися до викладача (інженера/лаборанта).

2. При появі запаху гару, незвичайного звуку негайно припинити роботу і повідомити викладача (інженера/лаборанта).

3. ПЕРЕЛІК ТА ОПИС КОМПЕТЕНТНОСТЕЙ, ЩО ФОРМУЮТЬСЯ У ЗДОБУВАНА ПІД ЧАС ЛАБОРАТОРНИХ ЗАНЯТЬ

Мета дисципліни — надати здобувачам вищої освіти знання, вміння та навички використання сучасних технологій розроблення мобільних додатків. Дисципліна містить розділи, присвячені сучасним технологіям створення мобільних додатків, зокрема під ОС Android із використанням мови Kotlin. Завданнями навчальної дисципліни є набуття теоретичних знань і практичних навичок щодо розроблення мобільних додатків. Серед іншого, вивчаються такі теми: особливості створення дизайну додатка й інтерактивного інтерфейсу користувача, взаємодії з папками, файлами і базами даних, а також опрацювання дотиків користувача.

Технології, що вивчаються в рамках дисципліни: операційна система Android; мови програмування Kotlin і Java; мова розмітки XML; середовища розроблення програмних засобів Android Studio та JetBrains IntelliJ IDEA; системи керування базами даних SQLite і Realm; засіб адміністрування БД SQLite Expert; формати файлів CSV і TSV; хмарне середовище для роботи з електронними таблицями Google Sheets тощо.

Міждисциплінарні зв'язки: навчальна дисципліна відіграє важливу роль у підготовці майбутніх фахівців у галузі проєктування, розроблення та використання програмного забезпечення інформаційних систем. У подальшому знання та навички, набуті на дисципліні «Розроблення мобільних додатків», використовуються при створенні інтерфейсів мобільних програмних засобів під час виконання випускних кваліфікаційних робіт. Пререквізити — дисципліни Об'єктно-орієнтоване програмування, Організація баз даних та знань, Захист та безпека даних. Надалі знання та навички, набуті на дисципліні «Розроблення мобільних додатків», можуть використовуватися при проходженні виробничої практики та написанні випускної кваліфікаційної роботи бакалавра.

Згідно з вимогами освітньо-професійної програми «Комп'ютерні науки» здобувачі повинні набути **здатності** отримувати компетентності:

інтегральна:

- здатність розв'язувати складні спеціалізовані задачі та практичні проблеми у галузі комп'ютерних наук або у процесі навчання, що передбачає застосування теорій та методів інформаційних технологій і характеризується комплексністю та невизначеністю умов.

загальні:

- ЗК 1. Здатність до абстрактного мислення, аналізу та синтезу.
- ЗК 2. Здатність застосовувати знання у практичних ситуаціях.
- ЗК 3. Знання та розуміння предметної області та розуміння професійної діяльності.
- ЗК 6. Здатність вчитися і оволодівати сучасними знаннями.
- ЗК 7. Здатність до пошуку, оброблення та аналізу інформації з різних джерел.

- ЗК 8. Здатність генерувати нові ідеї (креативність).
- ЗК 9. Здатність працювати в команді.
- ЗК 10. Здатність бути критичним і самокритичним.
- ЗК 11. Здатність приймати обґрунтовані рішення.
- ЗК 14. Здатність реалізувати свої права і обов’язки як члена суспільства, усвідомлювати цінності громадянського (вільного демократичного) суспільства та необхідність його сталого розвитку, верховенства права, прав і свобод людини і громадянина в Україні.

фахові:

- СК 2. Здатність до виявлення статистичних закономірностей недетермінованих явищ, застосування методів обчислювального інтелекту, зокрема статистичної, нейромережевої та нечіткої обробки даних, методів машинного навчання та генетичного програмування тощо.
- СК 7. Здатність застосовувати теоретичні та практичні основи методології та технології моделювання для дослідження характеристик і поведінки складних об'єктів і систем, проводити обчислювальні експерименти з обробкою й аналізом результатів.
- СК 8. Здатність проектувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об’єктно-орієнтованого, функціонального, логічного, крос-платформного з відповідними моделями, методами й алгоритмами обчислень, структурами даних і механізмами управління.
- СК 10. Здатність застосовувати методології, технології та інструментальні засоби для управління процесами життєвого циклу інформаційних і програмних систем, версіями програмних продуктів і сервісів інформаційних технологій відповідно до вимог замовника.
- СК 11. Здатність до інтелектуального аналізу даних на основі методів обчислювального інтелекту включно з великими та погано структурованими даними, їхньої оперативної обробки та візуалізації результатів аналізу в процесі розв’язування прикладних задач.
- СК 12. Здатність забезпечити організацію обчислювальних процесів в інформаційних системах різного призначення з урахуванням архітектури, конфігурування, показників результативності функціонування операційних систем і системного програмного забезпечення.

Здобувачі повинні досягти таких **програмних результатів навчання:**

- ПРН 1. Застосовувати знання основних форм і законів абстрактно-логічного мислення, основ методології наукового пізнання, форм і методів вилучення, аналізу, обробки та синтезу інформації, в предметній області комп'ютерних наук, володіти іноземною мовою професійного спрямування.
- ПРН 4. Використовувати методи обчислювального інтелекту, машинного навчання, нейромережевої та нечіткої обробки даних,

генетичного та еволюційного програмування для розв’язання задач розпізнавання, прогнозування, класифікації, ідентифікації об’єктів керування тощо.

- ПРН 12. Застосовувати методи та алгоритми обчислювального інтелекту та інтелектуального аналізу даних в задачах класифікації, прогнозування, кластерного аналізу, пошуку асоціативних правил з використанням програмних інструментів підтримки багатовимірного аналізу даних на основі технологій DataMining, TextMining, WebMining..

4. РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Виконання лабораторної роботи передбачає ознайомлення здобувача з методичними вказівками до відповідної лабораторної роботи, його теоретичну та практичну підготовку з відповідних розділів дисципліни.

Звіт до лабораторної роботи має бути оформлено відповідно до вимог: титульна сторінка роботи має бути оформлена за шаблоном, наведеним у додатку А, містити завдання та дані індивідуального варіанту, повний код програм(и) та знімки екранів результатів виконання.

Звіти лабораторних робіт виконуються в текстовому редакторі та надсилаються в електронній формі на платформу дистанційного навчання університету.

Робота, виконана з порушеннями наведених вимог, не зараховується і повертається здобувачу для доопрацювання. Робота, що виконана (повністю або частково) за неправильним варіантом, не зараховується.

ЛАБОРАТОРНА РОБОТА 1.

Створення базового інтерфейсу мобільного додатка

Мета: навчитися працювати з середовищем розроблення мобільних додатків, створювати та запускати найпростіші проєкти, застосовувати сучасні підходи до реалізації інтерфейсу та дизайну мобільних додатків, наповнювати екран додатка стандартними візуальними елементами, а також налаштовувати їхній зовнішній вигляд.

Завдання

Створити інтерфейс мобільного додатка-анкети (з використанням Android Studio та мови Kotlin / Java або інших технологій) зі стандартними візуальними елементами.

Теми мобільних додатків є індивідуальними для кожного здобувача і не повинні повторюватись у межах потоку здобувачів (усіх груп, які разом вивчають дисципліну в поточному навчальному році). Зразки тем і формулювань питань для анкет наведено нижче в таблиці 1.1. Можна взяти одну з поданих тем або придумати нову самостійно, узгодивши її з викладачем. Головне, аби всі теми були різними.

1. При створенні дизайну реалізувати можливість **вертикальної прокрутки екрану** (напр., через *ScrollView*), аби користувач міг побачити анкету повністю.
2. У **заголовку екрану** вивести інформацію про автора (прізвище, ім’я, група).
3. Згори екрану помістити **1 напис** із заголовком (темою) анкети.
4. Змінити **колір, розмір** і **накреслення** (наприклад, курсив / жирний) напису. Шрифт заголовку має бути більшим від усіх інших елементів на екрані.
5. Під написом помістити **3 порожні текстові поля** (без тексту):
   - *Ім’я*
   - *E-mail*
   - *Пароль*

   При створенні полів по можливості скористатися спеціальними елементами: зокрема для пароля взяти поле, що **автоматично приховує** введені символи.
6. Вписати в усі три поля **підказки** (*hint*), що саме в них має ввести користувач.
7. Під текстовими полями помістити **1 напис** із питанням № 1 із табл. 1.
8. Під написом розмістити мінімум **4 радіокнопки** для вибору одного з варіантів відповіді на це питання. Варіанти придумати на власний розсуд.
9. Пересвідчитися, що в радіокнопках одночасно можна обрати **лише 1 варіант.**
10. Під радіокнопками помістити **1 напис** із питанням № 2 із табл. 1.
11. Під написом розмістити мінімум **6 галочок** (чекбоксів) для вибору кількох варіантів відповіді на це питання. Варіанти придумати на власний розсуд.
12. Під галочками помістити **1 кнопку** з довільним текстом на ній («*ОК*», «*Надіслати*» тощо).

13. Змінити **колір тексту та фону** кнопки, а також **розмір шрифту,** аби він був більшим за шрифт текстових полів, радіокнопок і галочок.

14. Налаштувати **ширину** кнопки, аби вона не займала весь екран, а була трохи ширшою за розміщений на ній текст.

15. Вирівняти кнопку по горизонталі **по центру** екрану.

Додаткові (бонусні) бали:

За використання технологій, реалізацію яких не було описано в лекціях (у т.ч. ***Java***).

Таблиця. 1.1 − Зразки індивідуальних тем анкет і питань до них

Тема анкети	***Питання № 1 (радіокнопки)***	***Питання № 2 (галочки)***
Розробник ПЗ	рівень [Junior і т.д.]	мови програмування
Пошук роботи	спеціальність	компанії
Резюме	рівень освіти	ділові якості
Вступ до ВНЗ	спеціальність	пріоритетні ВНЗ
Навчання в школі	тип закладу [ліцей і т.д.]	улюблені предмети
Вибір ноутбука	операційна система	сфера застосування (розроблення ПЗ, відео і т.д.)
Вибір смартфона	об’єм пам’яті	виробники
Мобільний зв’язок	оператор	необхідні послуги
Соціальні мережі	вік користувача	мережі, в яких є акаунт
Відеоігри	консоль	улюблені жанри
Опитування геймерів	як часто граєте	улюблені ігри
Відвідування кіно	фільм для перегляду	кінотеатри міста
Мультфільми	вік глядача	улюблені мультфільми
Театр	назва вистави	актори
Фітнес	досвід тренувань	цілі тренувань
Спорт	маса тіла	види спорту
Футболіст	країна збірної	футбольні клуби
Художня література	улюблений автор	улюблені жанри
Вакансія перекладача	рідна мова	володіння іноземними мовами
Оренда квартири	кількість кімнат	райони міста
Дизайн інтер’єру	стиль	бажані кольори
Візит до лікаря	прізвище лікаря	зручні години
Відгук водію таксі	оцінка поїздки	що сподобалось
Доставка їжі	ресторан	замовлені страви
Отримання доставки	місто	зручні дні тижня
Купівля будматеріалів	вартість покупки	придбані товари
Автомобілі	ціновий діапазон	виробники
Іграшки	віковий діапазон	види іграшок
Взуття	розмір взуття	види взуття
Замовлення в ательє	розмір [S, M і т.д.]	предмети одягу

Тема анкети	*Питання № 1 (радіокнопки)*	*Питання № 2 (галочки)*
Готель	кількість зірок	зручності в номері
Кав’ярня	напій	десерти
Торт на замовлення	маса виробу	складові / смаки
Аптеки	назва мережі	категорії ліків
Туристична поїздка	країна	вид відпочинку
Закордонні подорожі	громадянство	відвідані країни
Авіаквиток	клас	додаткові послуги
Громадський транспорт	місто	наявні види транспорту
Організація концертів	гурт	міста гастролів
Музична школа	музичний інструмент	улюблені композитори
Відкриття депозиту	валюта	банки
Міжнародна компанія	галузь	країни-партнери
Коти	ставлення до котів	улюблені породи
Планер	час пробудження	ранкова рутина
Турист	країна, місто	види активності

Теоретичні відомості

Ринок мобільних операційних систем (ОС) бере свій початок наприкінці 2000-х років, коли існувало відразу декілька конкуруючих технологій. Із часом ситуація змінювалась, і станом на початок 2025 р. явними лідерами лишаються тільки дві ОС — Android OS та iOS. Зокрема в 2024 р., за даними ElectroIQ та Statista, їхні частки ринку складали відповідно 71,65% і 27,62%, що в сумі покриває понад 99% пристроїв у сучасному світі. Таким чином, розроблення мобільних додатків у нинішніх умовах передбачає проєктування програмних засобів саме під ці дві платформи. При цьому більшу увагу зазвичай приділяють Android як більш поширеній у світі ОС.

Тим не менше, важливо розуміти, що цей розподіл суттєво відрізняється в різних точках Землі. Наприклад, якщо проаналізувати ринок мобільних ОС і додатків за країнами світу, побачимо, що в таких державах, як США, Канада, Австралія, Японія, Норвегія, Швеція та деяких інших iOS, навпаки, виграє порівняно з Android OS (за даними DataWrapper станом на червень 2024 р.).

Далі, навіть якщо взяти країни, де Android є лідером, його перевага може бути більш чи менш суттєвою. Наприклад, за даними Statcounter на червень 2023 р., у більшості країн Європи дійсно переважав Android, однак різниця з iOS варіювалась від +7% у Бельгії до +76% у Греції. А, скажімо, у Швеції, де лідером була iOS, різниця складала всього 6%, тоді як у Данії — вже 26% на користь iOS.

Більш того, навіть у межах окремих країн ринок може бути неоднорідним. Відомими є приклади з відмінностями між різними штатами в США. Подібна ситуація є і з різними землями в Німеччині, яка протягом тривалого часу була розділена, що досі впливає на велику кількість показників, у тому числі й уподобання користувачів щодо мобільних пристроїв.

Крім того, не варто забувати, що в будь-якій точці вподобання людей і ринок змінюються з часом. Якщо простежити за часткою тих же двох ОС в Україні протягом останнього десятиріччя, можна виявити, що свого часу перевага Android була приголомшливою — понад 90%, тоді як зараз розподіл складає 67% проти 32%, і цей розрив рік у рік зменшується.

Врешті-решт, іще одним чинником, який впливає на вибір технологій для розроблення мобільних додатків IT-фахівцями та компаніями, є потенційні прибутки від створення цих програмних засобів. У той час, як величезна кількість додатків під Android OS є безкоштовними, користувачі iOS відносно частіше купують платні програми та підписки. Також середні витрати на користувача є вищими, ніж у Android OS. Тож із цих міркувань орієнтація на ринок iOS часом може бути вигіднішою фінансово.

Таким чином, попри значні переваги однієї з названих ОС у деяких окремих ситуаціях, у цілому обидві вони лишаються досить популярними. Тож при розробленні мобільних додатків за можливості бажано використовувати кросплатформні технології для створення програм, що працюватимуть і під Android OS, і iOS, або створювати окремі версії під ці ОС.

Якщо говорити про технічний бік питання, то сучасні технології розроблення мобільних додатків передбачають використання мов програмування високого рівня, а також відповідних бібліотек і модулів для них. При цьому нині необхідні засоби доступні для багатьох поширених мов програмування, серед яких Java, Kotlin, Swift, Objective-C, C, C++, C#, Python, JavaScript, Go та інші.

Щодо Android OS, існує два основні підходи до розроблення:

- створення «нативних», «рідних» додатків (від англ. *native apps*):
  - із використанням мови Java;
  - за допомогою мов, похідних від Java (в т. ч. Kotlin);
- використання проміжних ланок для інших мов, наприклад:
  - C# (.NET MAUI, раніше — Xamarin);
  - C і C++ (Android NDK);
  - Python (Kivy тощо);
  - JavaScript + HTML (Apache Cordova).

Варто зауважити, що деякі із вищезазначених підходів є універсальними та дозволяють створити додатки і під iOS. Наприклад, в останні роки одним із лідерів у галузі мобільного розроблення є Kotlin, який дає змогу писати програми під обидві найпопулярніші ОС. І саме цій мові буде приділено основну увагу в нинішньому курсі — на прикладі створення додатків під Android OS.

Мова Kotlin спроєктована таким чином, аби бути повністю сумісною з Java, її бібліотеками та віртуальною машиною (Java Virtual Machine — JVM). Вона розроблялася ще з 2010 року, в 2016-му був перший сталий випуск, із 2017-го її почала офіційно підтримувати корпорація Google, яка розробляла ОС Android. А вже в 2019-му вони оголосили Kotlin рекомендованою мовою для розроблення мобільних додатків під Android OS. Тоді відбувся різкий сплеск популярності Kotlin серед інших компаній та розробників. І якщо раніше більшість засобів під ОС Android писалися на Java, то відтоді саме Kotlin стали масово використовувати для нових проєктів і модулів, а іноді навіть переписувати на ньому вже наявний

Java-код. Тим не менше, завдяки взаємній сумісності Java та Kotlin останнє не є обов'язковим, і в багатьох проєктах досі успішно співіснують відразу дві мови програмування.

Цікаво також, що в 2019–20 рр. Kotlin став лідирувати в деяких опитуваннях щодо пріоритетних мов програмування для розробників. Зокрема він входив до топ-5 відповідей на запитання «Яку мову ви плануєте вивчати наступною?» та «Які мови ви любите?» (за даними HackerRank та Stack Overflow).

І якщо відповіді на перше запитання, вочевидь, багато в чому спричинені офіційною підтримкою Google, то на результати другого опитування вплинули особливості самої мови. Річ у тому, що Kotlin з'явився пізніше, ніж Java, тож у ньому було враховано попередній досвід і суттєво поліпшено окремі деталі. Наприклад, звичайне створення класу з трьома властивостями, яке в Java могло займати кілька десятків коду, в Kotlin вкладається всього в три рядки:

```
data class Person(var name: String,
                  var age: Int,
                  var height: Float = 1.8f)
```

Так само базова консольна програма «*Hello, world*» пишеться досить просто:

```
fun main()
{
    println("Hello World")
}
```

До речі, слід сказати, що Kotlin є універсальною мовою програмування, що дозволяє створювати не лише мобільні додатки, а й звичайні консольні програми, додатки з графічним інтерфейсом під настільні ОС та навіть веб-сайти. А на офіційній сторінці мови є онлайн-середовище, де можна експериментувати з її функціями: https://play.kotlinlang.org.

Що стосується середовищ розроблення (IDE), для Android OS зазвичай використовують офіційний засіб Android Studio від JetBrains і Google, який є повністю безкоштовним для всіх користувачів. Крім того, мобільні додатки на Kotlin і Java під цю ОС можна писати в IntelliJ IDEA та інших середовищах.

Розглянемо процес встановлення Android Studio. Інсталяційний файл середовища, який можна знайти на офіційній сторінці IDE, зазвичай не вимагає попереднього завантаження інших програмних засобів. Адже цей файл уже включає в себе всі додаткові компоненти, від яких залежить розроблення та виконання програм на Kotlin і Java, зокрема Java Development Kit (JDK). Тож JDK та інші пакунки можна не скачувати окремо (за винятком випадків, коли Вам потрібні якісь конкретні версії, відмінні від включених до складу Android Studio).

Тим часом важливо правильно налаштувати антивірусні програми, аби вони не блокували та не гальмували процеси інсталяції та використання компонентів. Іноді саме через антивіруси в процесі встановлення та налаштування середовища пропускаються окремі файли завантаження. Через це надалі не працюють окремі функції чи вся система в цілому.

Після встановлення IDE необхідно також скачати та встановити Android Software Development Kit (SDK). Зазвичай цей процес автоматично запускається після встановлення середовища. Проте зауважте, що обсяг цього набору бібліотек значно переважає інсталятор самої Android Studio і може займати 6 ГБ і більше. Тож передбачайте це при виборі мережі для завантаження файлів та диску для їх збереження та встановлення.

Врешті-решт, для процесорів Intel іноді виникає проблема з віртуалізацією, через що буває необхідно додатково встановити компонент Hardware Accelerated Execution Manager (HAXM), зазвичай версії 7.3.2, а також увімкнути можливість віртуалізації через BIOS. Без цих дій у середовищі Android Studio може не запускатися віртуальний пристрій (емулятор), що є досить зручним інструментом для тестування створених додатків.

Якщо процес встановлення IDE та всіх додаткових складових був вдалим, при запуску середовища Ви зможете створити новий проєкт і побачити цілий ряд шаблонів, доступних для вибору. Серед них — порожній екран (*Empty Activity*), базовий екран зі стандартним заголовком і кнопкою (*Basic Activity*), екран авторизації (*Login Activity*), екран із мапою (*Google Maps Activity*), екран налаштувань (*Settings Activity*) та багато інших. Для подальших прикладів будемо використовувати шаблон *Empty Activity with Views.*

На наступному вікні слід задати назву додатка, обрати розташування для збереження проєкту, мову програмування (пріоритетною є Kotlin) та версію SDK. При виборі версії SDK (API) середовище відразу показує приблизну оцінку, на скількох відсотках пристроїв у світі зможе запускатись такий додаток.

Чим нижчу (старшу) версію обираєте, тим більш універсальним буде Ваш засіб, адже базові функції будуть підтримуватись як на старих, так і на сучасних пристроях. І це хороший підхід, якщо першочерговою метою є максимальна доступність програми та розширення кола потенційних користувачів. Крім того, найновіші версії часом не функціонують як слід і можуть спричиняти додаткові труднощі при створенні додатків.

Однак із іншого боку — чим вища (новіша) версія, тим більше нових можливостей зможе включити додаток. Тож при виборі версії SDK слід прагнути до золотої середини, враховуючи складність проєкту та потрібні функції.

Новостворений проєкт міститиме багато різних файлів, однак основних, із якими буде проводитись більшість дій, зазвичай лише три:

- **.kt** або **.java** (залежно від обраної мови) — код, логіка програми;
- **.xml** — дизайн (зовнішній вигляд) екрану;
- **.gradle** (чи інші залежно від компілятора) — конфігурація проєкту.

У файлі конфігурації можна в будь-який момент змінити версію SDK, згадану вище, якщо вже в ході роботи над проєктом із якихось причин було вирішено перейти на новішу чи старішу.

Якщо при створенні проєкту було обрано шаблон *Empty Views Activity* і мову Kotlin, то Ви побачите в основному вікні програми файли **activity_main.xml** та **MainActivity.kt.** У режимі дизайну буде видно майже порожній екран додатка лише з одним елементом — дрібним текстовим написом «*Hello World*» по центру.

Уже такий простий додаток-заготовку можна скомпілювати й запустити на реальному чи віртуальному пристрої. Щодо віртуальних, для цього слід спершу налаштувати емулятор. Це можна зробити через меню *Tools → AVD Manager → Create Virtual Device → Phone,* після чого треба обрати модель телефону чи іншого пристрою та встановити образ системи (*system image*), який відповідає певній ОС, під якою буде перевірятись робота додатка на ньому. При запуску в емуляторі програма виглядатиме саме так, як би це було на реальному пристрої з обраними характеристиками.

Аби перевірити роботу додатка, слід побудувати проєкт (*Make Project*, **Ctrl + F9**) і запустити його (*Run*, **Shift + F10**). При цьому на деяких комп'ютерах під час запуску на віртуальному пристрої виникають проблеми з ADB (*Android debug bridge*), які можна спробувати усунути наступним чином. В емуляторі слід обрати меню *Settings → General → ... → Use detected ADB location.* Далі треба зазначити шлях до файлу **adb.exe,** який зазвичай знаходиться в папці з SDK у підкаталозі **platform-tools.** В іншому разі варто скористатись пошуком і знайти цей файл (або завантажити його на комп'ютер), а тоді задати його розташування, як описано вище.

Крім перевірки додатків у емуляторі, можна також скористатися реальними фізичними пристроями. Для цього слід під'єднати телефон, планшет тощо через USB-кабель до комп'ютера, на якому запущено Android Studio, або побудувати в середовищі файл **.apk** і скинути його собі на пристрій.

Перший варіант може вимагати попереднього встановлення додаткових драйверів (наприклад, для Samsung чи інших виробників телефонів), після чого все працює як слід без особливих труднощів. Це один із найефективніших способів тестування додатків, оскільки по кабелю додаток передається майже миттєво, а також запускається на реальній ОС без зволікань, на відміну від емулятора, який може спричиняти деякі затримки щодо запуску пристрою, образу системи на ньому тощо.

Для другого варіанту нам треба обрати меню *Build → Build Bundle(s) / APK(s) → Build APK(s),* після чого зачекати та відкрити папку з файлом і перекинути його на потрібний пристрій довільним чином. При відкритті такого файлу на телефоні він починає встановлювати додаток (або оновлювати його, якщо раніше вже було встановлено інші версії додатка з тим же іменем) так само, як встановлюються додатки при скачуванні їх із Google Play Store та інших джерел.

Аби додаток запустився та коректно працював, необхідно дозволити його встановлення і використання ним потрібних нам функцій на телефоні (наприклад, роботи з файлами, мережею, камерою, мікрофоном тощо). Також опціональним варіантом у ході встановлення є надсилання додатка на сканування до Play Protect для перевірки на безпеку. Цим інструментом справді доцільно користуватись, якщо додаток створено не Вами, а взято з неперевіреного джерела. Проте для власного додатка цей крок можна спокійно пропустити.

Тепер розглянемо крок за кроком процес створення дизайну нашого додатка, тобто зовнішнього вигляду екрану, розміщення та налаштування елементів на ньому.

На рис. 1.1 наведено інтерфейс середовища Android Studio. Як бачимо, крім основного вікна, де можна переглядати код і дизайн додатка, тут також є багато бічних панелей із різними інструментами. Для зручності можна за потреби згортати та розгортати ці панелі так, аби мати достатньо місця для перегляду речей, потрібних у поточний момент. Серед цих панелей — зокрема дерево проєкту ліворуч (*Project*), атрибути праворуч (*Attributes*), палітра інструментів (*Palette*) і дерево компонентів (*Component Tree*).

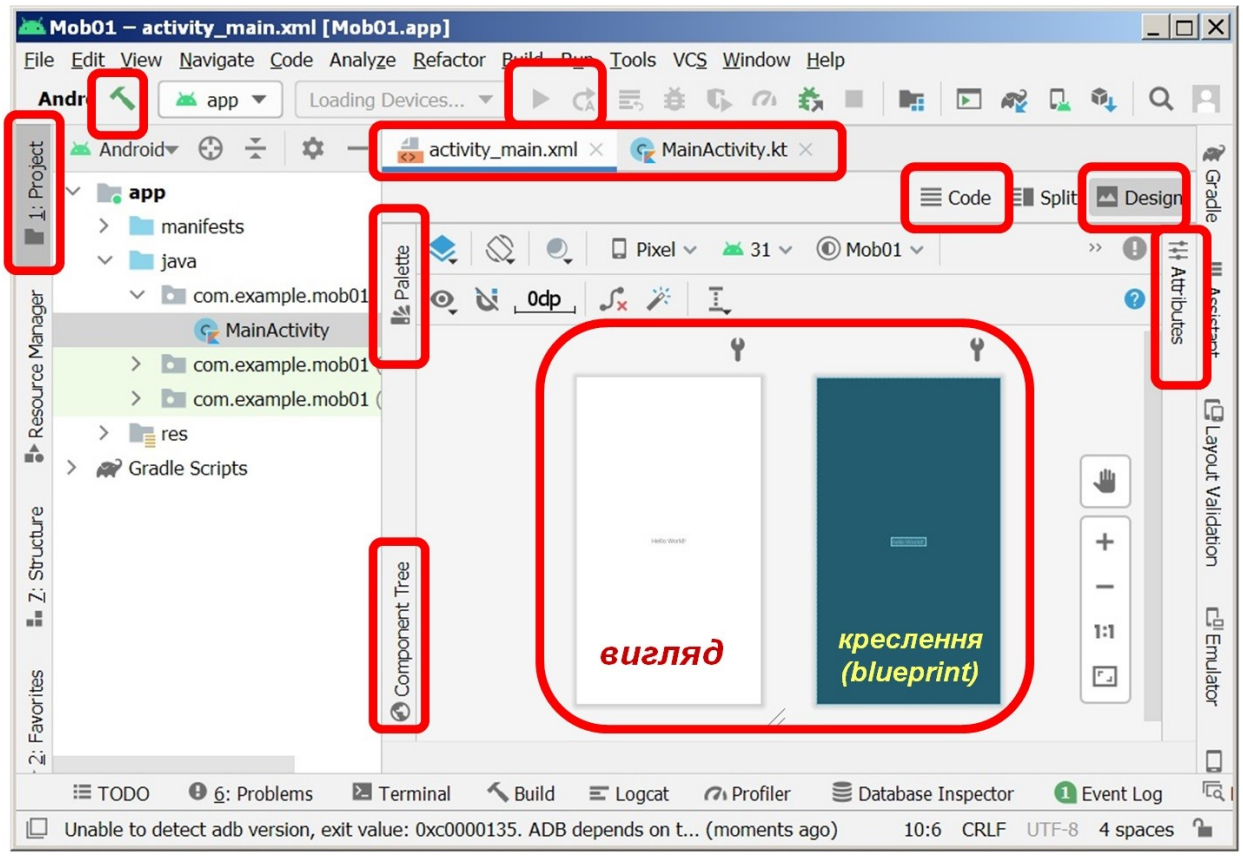

Рисунок 1.1 – Інтерфейс середовища Android Studio

В основному вікні можна перемикатись між окремими файлами, а для дизайну (файл **.xml**) можна обрати режим перегляду XML-коду (**Code**), дизайну (**Design**) чи комбінований (**Split**). Крім того, при роботі з дизайном показуються відразу два екрани поруч один із одним. Лівий є відображенням екрану в тому вигляді, як його бачитимуть користувачі додатка. Правий є ескізом, кресленням (*blueprint*) і показує повністю всі елементи, розміщені на екрані, навіть якщо вони приховані від користувача (постійно чи тимчасово протягом роботи програми).

При роботі з **.xml** у режимі дизайну, відкривши ліворуч вкладку «Палітра» (**Palette**), можна обрати і перетягнути на екран додатка потрібні нам елементи. Далі праворуч у панелі атрибутів (**Attributes**) можемо задати їм назви, текстовий вміст, кольори, фон тощо через відповідні властивості.

Перед тим, як розглядати окремі елементи, слід сказати, що є принаймні два основні типи макетів, які традиційно використовуються для створення дизайну додатків у Android Studio. Це розмітка за відступами (**ConstraintLayout**) і лінійна розмітка (**LinearLayout**).

У першому випадку для кожного візуального елемента задаються відступи по боках. Розташування будь-якої речі в дизайні визначається відносно або країв екрану, або сусідніх елементів. Цей варіант система встановлює за замовчуванням при створенні нових проєктів. Такий спосіб зручний, якщо в дизайнера є чітке бачення того, як має виглядати екран і є потреба чітко встановити розміщення кожного елемента з точністю до пікселів. Також це зручно, коли елементи незалежні один від одного, коли на екрані має бути деякий вільний простір тощо.

Натомість у лінійній розмітці всі елементи йдуть послідовно один за одним, і система автоматично визначає їхнє розташування на екрані. Зазвичай це вертикальна розмітка, але це можна використовувати і для горизонтальної. Такий підхід зручний, коли нам треба просто розмістити на екрані деякі речі в певній послідовності, але їхнє докладне розташування по пікселях для нас не суттєве. Тоді простим перетягуванням елементів на екран ми можемо дуже швидко створити базовий дизайн нашого додатка. Наступні приклади тут будуть саме на основі лінійної розмітки.

На початку створення макета важливо також урахувати, що звичана нам можливість прокрутки екрану (*scrolling*) не є автоматичною властивістю всіх мобільних додатків. А відтак цю річ необхідно передбачити окремо, додавши спеціальний елемент **ScrollView** (у палітрі він є в категорії **Common**). Цей елемент повинен бути заданим у першу чергу, як батьківський відносно всієї області екрану, яка буде доступною для прокрутки. Зазвичай на місці лишається тільки заголовок додатка, натомість усі інші елементи прокручуються, хоча бувають інші реалізації.

При створенні лінійної розмітки зручним підходом є перетягувати елементи з палітри не напряму на екран, а на панель дерева компонентів під палітрою. Там можна побачити ієрархію та послідовність усіх елементів, за потреби поміняти їх місцями або перемістити рівнем нижче чи вище. Для реалізації прокрутки екрану слід зробити так, аби найвищим рівнем ієрархії був елемент **ScrollView,** а рівнем нижче буде елемент LinearLayout — лінійна розмітка для всіх елементів, що будуть прокручуватись. Ця розмітка зазвичай створюється автоматично при доданні елемента **ScrollView.** І вже всередину неї (ще рівнем нижче) слід тепер поміщати всі основні візуальні елементи, які відображатимуться на екрані.

Розглянемо основні категорії та елементи, з якими ми будемо працювати:

- категорія **Text:**
  - **TextView** (напис);
  - **EditText** (текстове поле введення):
    - **Plain Text** (для звичайного тексту);
    - **E-mail** (для введення електронної пошти);
    - **Password** (для паролів);
- категорія **Buttons:**
  - **RadioButton** (радіокнопка);
  - **CheckBox** (галочка, прапорець);
  - **Button** (звичайна кнопка).

Зауважте, що вибір правильного підтипу елемента, наприклад, для пошти та пароля, може спростити нам роботу з дизайном і валідацією даних. Адже поштове

поле самостійно перевіряє введену інформацію та дозволяє користувачам вводити лише коректні адрес, а поле пароля автоматично приховує символи при введенні, заміняючи їх на крапки.

Слід зазначити, що всі елементи, з якими буде взаємодіяти користувач — текстові поля, радіокнопки тощо — зберігають свої налаштування (введений текст, обрані позначки і т.д.) доти, доки додаток запущено і не закрито. При повторному запуску програми на пристрої всі ці елементи автоматично очищаються. Більш того, в деяких старих версіях Android вміст і налаштування окремих елементів могли збиватись навіть при повороті екрану (з портретної орієнтації в альбомну або навпаки). Про ці речі варто пам'ятати і про всяк випадок тестувати, як додаток працює з даними у візуальних елементах.

Щодо напису (**TextView**), він має наступні основні властивості (атрибути):

- **id** — назва, за якою можна звертатись до елемента в коді;
- **text** — вміст поля;
- **textSize** — розмір шрифту;
- **textColor** — колір шрифту;
- **background** — колір фону напису;
- **textStyle (bold, italic...)** — стиль, накреслення (жирний, курсив...)
- **textAlignment (center, viewStart, viewEnd)** — вирівнювання.

Усі ці налаштування є досить очевидними і подібними до аналогів із інших мов програмування і середовищ, зокрема C# і Visual Studio. Проте вирівнювання трохи відрізняється тим, що замість лівого та правого краю тут зазначено початок і кінець елемента. Таке найменування є більш універсальним, адже в деяких мовах текст пишеться справа наліво, і, отже, початок напису має бути праворуч.

В елемента **EditText** дуже подібні властивості, але він також має одну особливу — підказку (**hint**), яка дає змогу показати певний текст блідим шрифтом ніби під самим текстовим полем для введення даних. Коли користувач торкається поля та починає вводити власний текст, підказка зникає. Таку можливість зручно застосовувати, коли треба пояснити користувачам, що саме вимагається ввести в кожному полі. Без підказки цю інформацію довелося б розміщати в окремих текстових написах, що займало би додатковий простір на екрані. Тим часом саме в мобільних додатках, як ніде інде, особливо важливо заощаджувати місце та не перевантажувати екран будь-якими зайвими візуальними елементами.

Щодо радіокнопок, їх обов'язково треба згрупувати між собою в межах спеціального елемента — радіогрупи (**RadioGroup**). Саме він визначає, що в межах окремої групи користувач зможе одночасно позначити лише одну радіокнопку, а решта позначок будуть автоматично зняті. На екрані можна створити декілька груп, наприклад, для кількох окремих запитань. Отже, спершу перетягуємо на дерево компонентів радіогрупу, а тоді вже всередину нього, рівнем нижче, поміщаємо всі радіокнопки, що стосуються того самого запитання чи налаштування.

Звичайна кнопка має наступні важливі властивості:

- **id** — назва;
- **text** — текст на кнопці;

- **textSize** — розмір шрифту тексту;
- **textColor** — колір тексту;
- **backgroundTint** (чи аналоги в інших версіях) — колір фону кнопки.

Щодо розмірів кнопки, то при використанні вертикальної лінійної розмітки всі елементи за замовчуванням автоматично розширюються до меж екрану по ширині. Проте це можна змінити, задавши параметр layout_width:

- **match_parent** — по ширині батьківського елемента;
- **wrap_content** — по ширині власного вмісту (зазвичай тексту).

Крім того, коли кнопка вже не займає всю ширину екрану, доречно також задати її вирівнювання через властивість layout_gravity, наприклад, по центру: center = true.

Врешті-решт, у ході роботи над додатком можна також змінити його заголовок, який відображається в панелі вгорі. Це можна задати або в дизайні, або у файлі коду. Перший варіант створює заголовок, який відображається при запуску. Щоби змінити його, треба знайти у файлі **AndroidManifest.xml** рядки:

```
<application
//...
  android:label="@string/app_name"
```

Далі слід замінити *"@string/app_name"* на довільний текст у лапках.

Другий варіант передбачає можливість динамічної зміни заголовку в будь-який момент роботи користувача з додатком. Це можна робити багаторазово. Якщо нам для початку так само треба задати текст, який відображатиметься відразу при запуску, слід у файлі **MainActivity.kt** знайти метод **onCreate** і в ньому записати такий код:

```
this.setTitle("app title")
```

Для подальшої зміни будемо застосовувати ту саму функцію **setTitle(),** але вже в потрібні моменти часу (наприклад, як реакцію на певну подію або через певні проміжки часу з допомогою таймера) та з новим текстом.

Таким чином, ми розглянули всі необхідні кроки та візуальні елементи, необхідні для створення додатка-анкети на будь-яку тему.

Висновки

У ході виконання лабораторної роботи визначено, яким чином можна працювати з середовищем розроблення мобільних додатків, створювати та запускати найпростіші проєкти, застосовувати сучасні підходи до реалізації інтерфейсу та дизайну мобільних додатків, наповнювати екран стандартними візуальними елементами, а також налаштовувати їхній зовнішній вигляд.

Запитання для самоперевірки

1. Сучасні технології розроблення мобільних додатків.
2. Операційна система Android та її особливості.

3. Мова програмування Kotlin та її особливості.
4. Шляхи створення макета (дизайну, структури) мобільного додатка.
5. Основні візуальні елементи графічного інтерфейсу користувача (GUI) в мобільних додатках під Android.
6. Налаштування розмірів та шрифтів візуальних елементів мобільного додатка.
7. Налаштування кольорів візуальних елементів мобільного додатка та RGB-модель.

ЛАБОРАТОРНА РОБОТА 2.

Реалізація інтерактивного інтерфейсу мобільного додатка

Мета: навчитися застосовувати сучасні підходи до реалізації інтерактивності в мобільних додатках та реалізовувати відгук інтерфейсу на дії користувача з візуальними елементами на екрані.

Завдання

Реалізувати реакцію на дії користувача в мобільному додатку-анкеті, створеному в лабораторній роботі № 1 (із використанням Android Studio та мови Kotlin / Java або інших технологій).

1. Додати (імпортувати) до проєкту картинку довільного розміру, яка відповідає тематиці анкети.
2. Вставити картинку в кінці анкети (після кнопки) і зробити її невидимою.
3. Змінити розміри картинки так, аби по ширині вона займала приблизно половину екрану, а її висота автоматично підлаштовувалась пропорційно (без спотворень).
4. Налаштувати вирівнювання картинки по центру.
5. Створити іконку додатка. У звіті показати, як виглядає іконка на екрані.
6. Створити метод (функцію), який запускатиметься при натисненні на кнопку наприкінці анкети. В цьому методі:
   - зробити саму кнопку невидимою, а картинку під кнопкою — видимою;
   - показати спливаюче повідомлення (***Toast***) із подякою за заповнення анкети (текст подяки довільний);
   - додати до подяки **ім'я** та **e-mail,** введені користувачем у текстові поля вгорі.
7. Створити один спільний метод (функцію), який запускатиметься при виборі кожної з наявних на екрані радіокнопок.
8. У створеному в ***п. 8.*** методі змінити колір фону кнопки залежно від обраної радіокнопки. Всі кольори — довільні, але вони мають бути різними для кожної з радіокнопок.
9. Створити один спільний метод (функцію), який запускатиметься при натисненні кожної з наявних на екрані галочок (прапорців, чекбоксів). У методі показати спливаюче повідомлення (***Toast***) із назвою саме тієї галочки, яку натиснув користувач, і текстом на кшталт «обрано» / «не обрано» залежно від того, галочку поставили чи зняли.

Або (на нижчий бал):

Для кожної з наявних на екрані галочок (прапорців, чекбоксів) створити окремі методи (функції), які запускатимуться при їх натисненні. У кожному методі показати спливаюче повідомлення (***Toast***) з назвою галочки.

Додаткові (бонусні) бали:

За використання технологій, реалізацію яких не було описано в лекціях (у т.ч. ***Java***).

Теоретичні відомості

Робота із зображеннями в Android Studio подібна до розміщення будь-яких інших візуальних елементів. Зокрема на палітрі в категорії **Common** слід знайти елемент **ImageView,** який створює прямокутну заготовку для розташування картинки на екрані. Якщо перетягнути його на екран або в дерево компонентів, то зазвичай автоматично відкривається вікно імпорту, де можна обрати одне із зображень, включених до проєкту. Якщо потрібна нам картинка ще не додавалась, тут можна натиснути «+», обрати **Import Drawables,** знайти шлях до зображення, далі натиснути ***OK → Next → Import.*** Після цього ми побачимо картинку серед доступних варіантів, можемо обрати її та натиснути OK. Тепер вона з'явиться на екрані, а також буде доступна в дереві проєкту в папці **app\res\drawable.**

Після того, як зображення з'явилось на екрані, можемо налаштувати його розміри, вирівнювання та інші властивості аналогічно до кнопки чи інших речей.

За замовчуванням картинка, як і кнопка, займатиме по ширині весь екран, а висота автоматично підлаштовується пропорційно. Аби змінити ці розміри, слід знайти в атрибутах ImageView властивості **scaleType** (має бути **fitXY**), а також ширину та висоту — **layout_width** та **layout_height** — і задати потрібні значення в пікселях. Пам'ятайте, що при ручному заданні розмірів треба завжди стежити за пропорціями зображення, аби воно не виглядало спотвореним (витягнутим або стиснутим за одним із вимірів). Одним зі способів досягнення цього є зафіксувати або ширину, або висоту картинки в пікселях, а для другої величини задати параметри **layout_width = wrap_content** і **adjustViewBounds = true.** Тоді висота підлаштується під ширину (або навпаки) автоматично, зберігаючи оригінальні пропорції зображення.

Щодо розмірів елементів на екрані, варто зауважити, що в Android Studio використовуються дві одиниці вимірювання:

- **dp** (*density-independent pixels*) — пікселі, незалежні від щільності (роздільності) екрану;
- **sp** (*scalable pixels*) — масштабовані пікселі.

Перша застосовується до зображень і подібних візуальних елементів. 1 dp відповідає пікселю на екрані середньої роздільності (160 dpi — *dots per inch*).

Натомість sp використовують для шрифтів і текстів. Ці розміри адаптуються до користувацьких налаштувань розміру шрифту в системі. Завдяки цьому тексти, оформлені з використанням sp, залишаються зручними для читання навіть при зміненому масштабі шрифтів, що є важливим аспектом доступності інтерфейсу.

Видимість картинок, як і інших візуальних елементів, задається значенням атрибута **visibility.** Якщо при проєктуванні дизайну присвоїти зображенню значення **invisible,** його не буде показано на екрані при запуску додатка. Також у режимі дизайну воно зникне з лівого екрану (білого), але залишиться на правому, де відображається ескіз проєкту (див. лабораторну роботу № 1).

Якщо ж нам треба змінити видимість картинки, кнопки тощо в ході роботи програми, тобто показати або приховати її в певний момент, можна звернутись до цього ж атрибута в коді, попередньо визначивши потрібний елемент через функцію **findViewById().**

Цей метод (подібно до його аналогу getElementById() у JavaScript) шукає на екрані елемент за його ID, тобто за назвою в проєкті. Та щоби це було можливо, слід спершу переконатися в панелі атрибутів, що ця назва задана в явному вигляді через властивість id. Якщо з цим усе гаразд, тоді можна шукати елемент і задавати йому будь-які параметри, зокрема й видимість, таким чином:

```
var btn: Button = findViewById(R.id.btn_ok)
btn.visibility = View.VISIBLE

var img: ImageView = findViewById(R.id.img_thank)
img.visibility = View.INVISIBLE
```

Тут var — це позначення змінної (від англ. *variable*). Для звернення до елементів можна також використовувати і val — константи (від англ. *value* — значення), якщо в подальшому не планується перекеровувати посилання на інший об’єкт. Після var або val пишеться назва, а через двокрапку — тип даних.

Зверніть увагу, що назва змінної може збігатись або не збігатись із назвою елемента на екрані. З одного боку, в нас є назви елементів (**id**), які визначаються дизайном (файл **.xml**), а з іншого — імена змінних, визначені в коді (**.kt / .java**). Їх можна називати довільним чином. І саме функція **findViewById()** прив’язує всі ці назви між собою, аби з коду можна було посилатись на дизайн.

Між іншим, при перейменуванні будь-яких елементів, змінних тощо зручно користуватись вікном рефакторингу коду. Воно може відкритись автоматично при спробі змінити назву або бути викликане власноруч. При роботі з кодом, коли курсор перебуває на потрібному елементі, слід натиснути **Shift + F6** або обрати правою кнопкою мишки ***Refactor → Rename...*** Після введення нової назви система автоматично пройде по всьому проєкту та замінить застарілу назву на актуальну. Це допоможе уникнути проблем із посиланням на старі назви без необхідності перевіряти ці речі вручну.

Іконка додатка створюється подібно до інших картинок у проєкті через процес імпорту зображення. Якщо не змінювати її, додаток матиме стандартну іконку Android, і на екрані, у списках додатків тощо виглядатиме так (рис. 2.1):

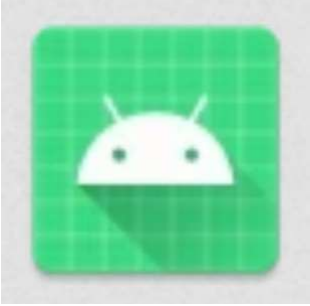

Рисунок 2.1 – Іконка додатка під Android за замовчуванням

Щоби змінити це зображення, можна завантажити власну картинку, з якої система автоматично створить багато різних варіантів іконки — велику, малу, квадратну, заокруглену, круглу тощо, кожна з яких буде використовуватись у різних ситуаціях. Розроблення власної іконки допоможе користувачам швидко знайти саме Ваш додаток у списку. Для цього слід обрати зображення або створити логотип, що якомога більш відповідатиме темі додатка. Для іконок не варто брати фотографії чи графіку з великою кількістю деталей і текстом, адже зазвичай розміри іконок досить малі, і розібрати літери та дрібні фрагменти буде досить важко.

Для створення власної іконки додатка треба знайти в дереві проєкту папку **app\res\mipmap,** натиснути правою кнопкою мишки та обрати ***New → Image Asset.*** Після цього відкривається вікно **Asset Studio,** яке дозволяє обрати та задати налаштування нової іконки. На першому вікні слід обрати шлях до зображення (**Path**). Далі вона з'являється праворуч у полі попереднього перегляду. Тепер ліворуч можна змінити її розмір (**Resize**).

Якщо Ваше зображення збережено з білим фоном у форматі **.jpg** чи іншому, який не підтримує прозорість, то картинка виглядатиме як білий прямокутник на зеленому фоні. При розміщенні такої іконки на екрані пристрою вона матиме вигляд білого прямокутника, що зазвичай не дуже гарно. Тож радимо попередньо видалити фон і зберегти файл в одному з форматів, що підтримують прозорість (наприклад. **.png**).

На останньому кроці імпорту відображається попередження про те, що всі нинішні іконки додатка буде замінено на нові. При цьому іноді стається конфлікт через однакові імена файлів — наприклад, якщо Ваше зображення називається так само, як стара іконка або імпортовано більш ніж одну картинку з тією самою назвою. Характерно, що навіть якщо два файли мають різні розширення, але ту саму назву (скажімо, **icon.png** і **icon.jpg**), це все одно спричиняє помилку *«Duplicate resources»*. Тож варто видалити всі зайві файли, серед яких старі та не використані іконки, з підпапок у **app\src\main\res.**

Аби побачити нову іконку додатка, його слід повторно скомпілювати та встановити на пристрій. Майте на увазі, що при повторному встановленні додатка з тією самою назвою нова версія автоматично заміняє стару. Тож якщо маєте потребу встановити на телефон дві різні версії тієї самої програми для тестування, порівняння тощо, необхідно змінити назву додатка в налаштуваннях проєкту.

Сповіщення (спливаючі повідомлення) в Android Studio називаються **Toast.** Тут розробниками було використано аналогію з хлібом, який на мить вистрибує з тостера. Справді, такі сповіщення подібним чином показують користувачам деякі важливі повідомлення та швидко зникають. При цьому вони не зачіпають решти інтерфейсу та не заважають працювати з додатком. У коді для показу такого повідомлення скористаємось відповідним класом і функціями **makeText()** і **show():**

```
val msg = Toast.makeText(this, "Thank you!", Toast.LENGTH_SHORT)
msg.show()
```

Якщо це перше використання сповіщень у проєкті, нам може знадобитись імпорт класів по роботі з **Toast.** Для цього можна прописати це вручну на початку коду:

```
import android.widget.Toast
```

Якщо не зробити цього, ми можемо побачити попередження про посилання на елемент, що не був підключений до проєкту (*Unresolved reference*). У віконці, яке з'являється, є варіант натиснути кнопку Import і розв'язати цю проблему.

Проте іншим варіантом є подбати про імпорт усіх необхідних компонентів заздалегідь. Це можна зробити, якщо обрати в меню ***File → Settings*** і знайти слово *import*. У пункті налаштувань ***Editor → General → Auto-Import*** є прапорець ***Optimize imports on the fly,*** який слід позначити. Після цього кожного разу при посиланні на нові класи та функції система намагатиметься автоматично знайти їх серед наявних бібліотек і самостійно підключити до проєкту. Це може спростити розроблення додатка та усунути типові помилки й попередження.

Тепер розглянемо опрацювання подій у Android Studio. При взаємодії користувача з мобільним додатком ми можемо відстежувати багато речей, але одними з найпростіших є натиснення на кнопки, радіокнопки та прапорці.

Якщо ми хочемо відреагувати на натиснення кнопки в додатку-анкеті, показуючи сповіщення-подяку, створену вище через **Toast,** нам треба помістити цей код до окремої функції-обробника кнопки. Такі функції в Kotlin схожі на будь-які інші, проте зазвичай вони повинні мати принаймні один параметр — елемент, який викликає їх виконання. Таким чином, код обробника кнопки «OK» може виглядати приблизно так:

```
fun OK(view: View)
{
    val msg = Toast.makeText(this, "Thank you!", Toast.LENGTH_SHORT)
    msg.show()
}
```

Усі обробники, що будуть запускатись із головного екрану додатка, слід поміщати у файлі **MainActivity.kt** у класі **MainActivity,** та паралельно з методом **onCreate,** тобто на тому ж рівні вкладення (а не всередині його).

Тепер ми маємо функцію-обробник, але поки що вона ще не прив'язана до жодної події в нашій програмі, тож фактично не буде запускатись жодного разу при роботі мобільного додатка. Для зв'язку з конкретним візуальним елементом, натиснення якого повинно викликати цю функцію, треба відкрити файл дизайну **(.xml)** у режимі коду **(Code).** Там слід знайти потрібний елемент і додати новий рядок із посиланням на функцію-обробник у довільне місце всередині його опису. Якщо це звичайна кнопка, яка реагує на натискання, маємо знайти елемент **Button** і вписати туди наступне:

```
<Button
    android:onClick="OK"
```

Шлях створення обробників подій для інших візуальних елементів додатка є аналогічним. У коді описуємо необхідні дії, а в дизайні прив'язуємо елемент до цієї функції. Технічно можна робити ці два кроки в довільному порядку.

Якщо ми хочемо зчитати вміст текстових полів на екрані, тут усе теж подібно до вже розглянутих вище способів взаємодії з візуальними елементами. Спершу знаходимо елемент на екрані, а тоді звертаємось до нього та зчитуємо потрібні властивості, серед яких і текст, у змінні чи куди завгодно:

```
var et_name: EditText = findViewById(R.id.et_name)
var et_email: EditText = findViewById(R.id.et_email)
var name = et_name.text
var email = et_email.text
```

Таким чином із додатку-анкети можна зчитати ім'я людини, її поштову адресу, пароль та інші дані, які користувач вніс у відповідні поля введення. Надалі можна буде зберегти цю інформацію до файлу чи бази даних. Однак окрім цього ми можемо також використати ці речі при показі сповіщення про успішне заповнення анкети, розширивши його текст. Наприклад:

```
var pers_text = "Дякуємо, " + name + "!"
pers_text += "\nМи надішлемо відповідь на Вашу пошту: " + email
val msg_pers = Toast.makeText(this, pers_text, Toast.LENGTH_SHORT)
msg.show()
```

Зверніть увагу, що у сповіщеннях, як і в більшості інших текстових написів, можна використовувати спеціальні символи для перенесення рядка **\n,** а також табуляцію **\t** тощо.

Для опрацювання натиснення на радіокнопку так само створюється функція в коді з довільною назвою, а в дизайні прописується параметр **android:onClick.**

Якщо плануємо за натисненням радіокнопки змінювати фон екрану, слід пригадати, що раніше ми вже заповнили майже весь його простір елементом **ScrollView.** І тепер саме його фон треба змінити, аби налаштувати колір екрану. Наприклад:

```
var scrn_main: ScrollView = findViewById(R.id.scrn_main)
scrn_main.setBackgroundColor(Color.GREEN)
```

Крім того, можна задати колір не через назву англійською, а RGB-модель:

```
scrn_main.setBackgroundColor(Color.parseColor("#DDFFDD"))
```

Варто також зазначити, що замість писати окремі обробники на кожну радіокнопку, набагато зручніше створити один загальний обробник для всіх них (у межах групи). З цією метою нам слід спершу звернутись до радіогрупи (так само, як і до інших елементів). Далі ми зможемо зчитати її властивість, що відповідає за визначення тієї радіокнопки, яка натиснута в поточний момент:

```
var rg: RadioGroup = findViewById(R.id.rg_q1)
val crb: RadioButton = findViewById(rg.checkedRadioButtonId)
```

Тепер змінна **crb** міститиме посилання на натиснуту радіокнопку. Далі створимо змінні, що посилаються на всі радіокнопки в цій групі, а потім зробимо в коді розгалуження: якщо **crb** збігається зі змінною від першої радіокнопки, замінимо фон на перший колір, якщо з другою — на другий, і т.д. У таких випадках зручно користуватись оператором **when,** який може перевіряти відразу багато різних значень (подібно до switch ... case в інших мовах програмування):

```
var rb1: RadioButton = findViewById(R.id.rb1)
var rb2: RadioButton = findViewById(R.id.rb2)
// ...
when (crb)
{
    rb1 -> scrn_main.setBackgroundColor(Color.parseColor("#FFFFDD"))
    rb2 -> scrn_main.setBackgroundColor(Color.parseColor("#DDDDFF"))
    // ...
}
```

Аналогічно до зміни фону екрану, ми можемо при бажанні змінити будь-які налаштування довільного елементу на екрані — його колір, фон, розміри і т.д. Скажімо, можна створити функцію-обробник кнопки, радіокнопки тощо, який буде змінювати ширину кнопки, щоразу додаючи до неї по 20 пікселів:

```
btn.width+=20
```

Цікаво, що якщо багаторазово запускати таку функцію, в підсумку кнопка може розширитись настільки, що її край навіть вийде за межі екрану. Тож при діях із дизайном треба враховувати деякі обмеження та взаємодію елементів між собою. Приклад із розширенням кнопки, певно, є досить відірваним від реальних мобільних додатків. Однак він наочно показує, що в ході роботи програми будь-який елемент на екрані може впливати на будь-який інший, змінюючи його властивості та зовнішній вигляд.

Нарешті розглянемо, як працюють прапорці (галочки, чекбокси). Вони так само реагують на натиснення через подію **android:onClick.**

І хоча, на відміну від радіокнопок, такі елементи є незалежними між собою, все одно є можливість створити один спільний обробник для всіх них. Якщо всі варіанти відповідей стосуються одного запитання, можна зробити одну спільну функцію та прив’язати її відразу до всіх прапорців. А визначати, який саме з них викликав функцію, можна через її параметр **view** наступним чином:

```
fun q2_choice(view: View)
{
    val cb: CheckBox = view.findViewById(view.id)
    //...
}
```

При цьому функція, прив’язана до цієї події, буде викликатись і запускатись і тоді, коли прапорець позначено, і коли його знято. Тож слід передбачити це в коді, також зробивши певне розгалуження (наприклад, через **if**). Скажімо, якщо галочку поставили, показується повідомлення про те, що певний варіант обрано, а якщо знято — не обрано. Перевірити це можна через властивість **IsChecked:**

```
if (cb.IsChecked)
    //...
else
    //...
```

Спільним кодом у функції буде виведення певних загальних фраз на кшталт «Обрано:» і «Не обрано:» (залежно від поточного стану галочки). Натомість далі вже як параметр буде підставлятися текст по конкретному прапорцю, який викликав цю функцію. Цей текст можна зчитати з підпису до галочки так само, як і зі звичайного текстового поля. Проте часом слід приводити його до рядкового типу для уникнення проблем із сумісністю. Це можна зробити так:

```
var cb_text: String = cb.text.toString()
```

Таким чином, ми розглянули всі необхідні кроки та аспекти, важливі для роботи з картинками, іконками, створення обробників подій і реалізації реакції на дії користувача в додатку-анкеті.

Висновки

У ході виконання лабораторної роботи визначено, яким чином можна застосовувати сучасні підходи до реалізації інтерактивності в мобільних додатках та реалізовувати відгук інтерфейсу на дії користувача з візуальними елементами на екрані.

Запитання для самоперевірки

1. Налаштування видимості елементів на екрані.
2. Робота із зображеннями в додатку.
3. Створення іконок додатка.
4. Візуальні елементи інтерфейсу, які дозволяють реалізовувати інтерактивність.
5. Реалізація реакції візуальних елементів інтерфейсу на дії користувача в мобільному додатку.
6. Реалізація обробників кнопок.
7. Реалізація обробників радіокнопок.
8. Реалізація обробників галочок (прапорців, чекбоксів).

ЛАБОРАТОРНА РОБОТА 3.

Збереження та завантаження даних

Мета: навчитися застосовувати сучасні підходи до збереження та завантаження даних у мобільних додатках, працювати з файлами, папками, базами даних і хмарними сховищами, реалізовувати навігацію між екранами та опрацювання дотиків користувача.

Завдання

Реалізувати відображення даних із Google-таблиці, навігацію, а також збереження, завантаження та виведення даних про користувачів, введених ними в додатку-анкеті з лабораторних робіт №№ 1–2 (з використанням Android Studio та мови Kotlin / Java або інших технологій).

1. У коді програми при запуску додатка створити нову папку **Data** (у внутрішньому або зовнішньому сховищі, в довільній підпапці). Відразу після цього всередині цієї папки створити текстовий файл із довільною назвою.
2. Доповнити обробник натиснення кнопки (***Button***) наприкінці анкети таким чином, аби до цього файлу записувались усі введені користувачем дані:
   - ім’я;
   - e-mail;
   - пароль;
   - обраний варіант відповіді на питання з радіокнопками;
   - всі обрані варіанти відповідей на питання з галочками (через кому).

   Вміст кожного з цих пунктів має записуватись у файлі з нового рядка.
3. Реалізувати доповнення вмісту файлу: аби при кожному наступному запуску додатка нові анкетні дані записувались у кінець файлу, не затираючи вже наявних. При цьому записи про окремих користувачів мають розділятись одне від одного 1–2 порожніми рядками.
4. У коді програми згенерувати новий текстовий напис (***TextView*** або інший подібний елемент) і помістити його в самому кінці після анкети (останній елемент унизу екрану). Колір і шрифт налаштувати довільним чином.
5. Зчитати весь наявний вміст текстового файлу до змінної та вивести цей текст у згенерованому в п.4 написі.
6. Створити в мобільному додатку, крім головного екрану з анкетою, другий і третій екрани (як Activity або Fragment тощо — довільним чином).
7. Реалізувати можливість переходу користувача з головного екрану на другий і третій та повернення з них на головний (через спеціальні кнопки або іншим чином).
8. На другому екрані додатка реалізувати опрацювання дотиків користувача (довільним чином).
9. Створити Google-таблицю з даними за темою Вашої анкети з **ЛР 1–2.** Таблиця повинна містити дві колонки. У першій мають бути певні елементи

(наприклад, товар, послуга, модель пристрою, ім'я особи, назва компанії, місто тощо — можна взяти це з питань із радіокнопками або галочками з анкети). У другій має бути певна додаткова інформація про елемент із першої колонки (наприклад, ціна, будь-які характеристики, країна, категорія, галузь). У першій колонці помістити 15–20 різних елементів. У другій має бути стільки ж записів. Експортувати (опублікувати) таблицю як TSV-файл і зберегти посилання в коді програми.

10. При запуску додатка з'єднатися з таблицею та вичитати з неї весь вміст у довільну структуру даних (наприклад, масив масивів / 2 списки тощо).

11. На третьому екрані додатка створити 2 порожні текстові написи. При запуску додатка завантажити вміст першого рядка таблиці в написи: в перший — із першої колонки, в другий — із другої.

12. Створити під написами 2 кнопки — «Вперед» і «Назад» (або з подібними назвами). При натисненні «Вперед» оновлювати вміст написів, показуючи наступний запис, «Назад» — попередній.

13. При переході користувача до останнього та першого запису показувати сповіщення (наприклад, як Toast) відповідно про кінець та початок даних.

Додаткові (бонусні) бали:

За використання технологій, реалізацію яких не було описано в лекціях (у т.ч. ***Java***).

Теоретичні відомості

Збереження даних у мобільних додатках під Android OS можливе:

- в тимчасовій пам'яті додатка:
- у файлах на пристрої;
- у хмарних сховищах;
- у базах даних (реляційних, об'єктних тощо).

Папки та файли в Android поділяються на внутрішні та зовнішні. При цьому внутрішніми вважаються лише ті, що знаходяться у так званому обмеженому сховищі (англ. *scoped storage*) — всередині службової папки з програмою, яка створюється у внутрішньому сховищі пристрою в папці **Android\data\...** Натомість файли, розташовані на SD-картці або навіть у внутрішньому сховищі поза межами цієї службової папки, вважаються зовнішніми відносно програми.

З розвитком ОС Android особливості доступу до зовнішніх файлів також змінювались. До 9-ї версії включно додатки мали вільний доступ до файлів поза межами службової папки. В 10-й версії ОС програмі для цього став потрібен спеціальний дозвіл. І починаючи з 11-ї версії, додатки мають вільний доступ лише до службової папки.

Нині для створення папки та файлу можна скористатись таким кодом:

```
val folder = File(this.getExternalFilesDir(String()), "Data")
folder.mkdirs()
var file_name = "user_data.txt"
val new_file = File(folder, file_name)
```

Вищенаведений фрагмент створить у службовій папці підпапку **Data,** а в ній — текстовий файл **user_data.txt.** Після запуску додатка з таким кодом можна переконатися, що все пройшло успішно, відкривши службову папку. Наприклад, якщо наш проєкт називається **mob01,** її розташування у внутрішньому сховищі може бути таким: **Android\data\com.example.mob01\files\.** У разі чого, назва проєкту міститься та може бути змінена у файлі **AndroidManifest.xml.**

Щойно було створено файл, із ним можна працювати, наповнюючи вмістом. Існує багато способів запису та читання файлів у Android OS. Одним із них є використання класу **FileOutputStream.** Візьмемо новостворений файл **new_file** та запишемо в нього текст «*Hello World*»:

```
try
    {
      val fos = FileOutputStream(new_file)
      new_file.appendText("Hello World")
    }
catch (e: FileNotFoundException)
    {}
```

Зверніть увагу, що операції з папками та файлами про всяк випадок слід загортати в конструкцію **try ... catch,** адже можлива ситуація, коли файл не знайдено або до нього немає доступу.

Очевидно, що до файлу можна без проблем записати будь-які текстові дані — не лише рядкові змінні, створені у програмі, а й вміст текстових полів, куди вносили свої дані користувачі додатка. Для зручності ми можемо спершу сформувати блок тексту з переносами, де в окремих рядках міститиметься ім'я, поштова адреса, пароль, а далі записати цей блок до файлу так само, як і раніше:

```
var info = "User data:\n" + name + "\n" + email + "\n" + pwd + "\n\n"
new_file.appendText(info)
```

Нагадаємо, що при збереженні інформації з текстових полів дані можна читати напряму, а для радіокнопок і прапорців краще явним чином перевести їх у текстовий формат через функцію **.toString()** (див. лабораторну роботу № 2).

Також, якщо спробувати записати інформацію до того самого файлу другий, третій раз і т.д., наступні дані можуть затерти ті, що були наявні раніше. Аби цього не сталося, слід при використанні **FileOutputStream** зазначати додатковий опціональний параметр **append,** який відповідає за спосіб доповнення файлу. Якщо задати значення **true,** то нові дані додаватимуться в кінець файлу, якщо ж **false** — нова інформація затиратиме собою попередній вміст:

```
val fos = FileOutputStream(new_file, true)
```

Читання файлу також не становить особливих труднощів. Одним зі способів прочитати вміст текстового файлу є функція **readText().** Однак як і при записі, важливо переконатися, що потрібний нам файл існує та загорнути всю процедуру в **try ... catch:**

```
var file_text = ""
try
    {
      file_text = new_file.readText()
    }
catch (e: FileNotFoundException)
    {}
```

Тепер розглянемо генерацію елементів графічного інтерфейсу додатка в коді програми. Це може бути корисно, якщо нам у певний момент роботи програми треба створити новий елемент і наповнити його деяким вмістом. Для прикладу візьмемо звичайний текстовий напис **(TextView)** і спробуємо помістити в нього дані, зчитані з нашого текстового файлу.

Для цього спершу згенеруємо напис-заготовку та налаштуємо його вигляд:

```
val tv_gen: TextView = TextView(this)
tv_gen.text = "згенерований напис"
tv_gen.setTextSize(TypedValue.COMPLEX_UNIT_SP, 24F)
tv_gen.setTextColor(Color.BLUE)
```

Далі розмістимо цей напис на екрані в межах наявної розмітки:

```
val lin_lay: LinearLayout = findViewById(R.id.lin_lay)
lin_lay.addView(tv_gen)
```

Як бачимо, до розмітки можна звертатись так само, як і до інших елементів на екрані, що логічно, адже вона наявна в дереві компонентів на одному з рівнів ієрархії. Важливо, що елемент **LinearLayout** має спеціальну функцію **addView(),** яка дозволяє додати будь-який новий елемент у кінець цієї розмітки, в нашому випадку — внизу екрану.

Зрозуміло, що після створення напису в нього можна помістити довільну інформацію рядкового типу, наприклад, вміст текстового файлу. Скористаємося для цього функцією **append(),** яка доповнює напис текстом:

```
tv_gen.append(file_text)
```

Для створення додаткових екранів у додатку (другого, третього і т.д.) в Android Studio є два основні підходи.

- **Activity** — створення нових файлів діяльностей поряд із **Main;**
- **Fragments** — розбиття вже наявної діяльності на фрагменти.

Розглянемо перший спосіб. У дереві проєкту в папці з назвою проєкту, що містить файл із кодом головної діяльності **MainActivity.kt,** натискаємо правою кнопкою мишки та обираємо ***New → Activity → Empty Activity*** (або будь-який інший тип екрану, який хочемо створити). Далі задаємо назву новому екрану, наприклад, **SecondActivity,** ставимо галочку для генерації файлу розмітки (*Layout File*) та задаємо назву йому — скажімо, **activity_second.** Слід дотримуватися того ж шаблону найменування файлів, який був на головному екрані.

У результаті отримаємо нові **.kt** та **.xml** поряд із файлами головного екрану. Тепер ми маємо заготовку нового екрану, але поки що між ним і головним іще немає зв'язку. Виправимо це. В коді головного екрану **(.kt)** — наприклад, у межах обробника певної кнопки, напишемо такий код:

```
val intent = Intent(this, SecondActivity::class.java)
startActivity(intent)
```

Цей фрагмент створює посилання на новий екран за його іменем, а тоді запускає його відображення через функцію **startActivity().** Тепер, якщо натиснути потрібну кнопку, користувач перейде з головного екрану на новий. Однак якщо на новому не передбачено жодного способу повернення назад, користувач опиниться у глухому куті. Слід уникати таких ситуацій і обов'язково для кожного нового екрану задавати можливість виходу.

Навігацію можна реалізовувати не лише через звичайні кнопки, а й через панелі меню. Ми можемо створити на новому екрані панель інструментів (*toolbar*), а на ній розмістити кнопку повернення. Панель додається у файлі **.xml:**

```
<androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar_menu"
    //...
    />
```

Кнопку ж можна додати таким чином. У файлі **.kt** імпортуємо панель інструментів, знаходимо її на екрані, а тоді задаємо можливість повернення назад на головний екран через стандартну кнопку:

```
import androidx.appcompat.widget.Toolbar
//...
val toolbar_menu: Toolbar = findViewById(R.id.toolbar_menu)
setSupportActionBar(toolbar_menu)
supportActionBar?.apply()
    {
      title = "Back"
      setDisplayHomeAsUpEnabled(true)
      setDisplayShowHomeEnabled(true)
    }
```

Також важливо визначити зв'язок між екранами додатка. Адже кнопка повернення автоматично переводить користувача на екран, який є батьківським відносно поточного. Тим часом, якщо в додатку є декілька екранів, батьківським технічно може бути будь-який із них. Тож пропишемо це явним чином у файлі **AndroidManifest.xml:**

```
<activity
    android:name=".SecondActivity"
    android:parentActivityName=".MainActivity"
</activity>
```

Опрацювання дотиків у Android реалізовується через клас **MotionEvent.** Коли користувач торкається екрану та робить рухи, це відстежується, і система визначає, що саме відбулось. Зокрема є константи, які відповідають за такі події:

- ***ACTION_DOWN*** — дотик;
- ***ACTION_UP*** — підняття;
- ***ACTION_MOVE*** — рух;
- та інші.

Стандартні візуальні елементи на екрані автоматично реагують на типові дотики та опрацьовують їх через свої методи. Зокрема якщо на екрані маємо елемент **ScrollView,** немає необхідності писати власний обробник для прокрутки, адже ця можливість уже вбудована в самий елемент.

Натомість якщо нам необхідно реалізувати в додатку щось нове і нетипове, для цього можна написати свій обробник, відстежуючи окремі події та реагуючи на них довільним чином. Для прикладу, на додатковому екрані будемо зчитувати рухи користувачів по екрану та виводити поточні координати в один із елементів, скажімо, панель заголовку.

На початку у файлі з потрібною діяльністю (наприклад, **SecondActivity.kt**) створимо змінні, які прийматимуть значення поточних координат. Ініціалізуємо їх, задавши початкові значення (0, 0), що відповідають лівому верхньому краю екрану:

```
var px = 0
var py = 0
```

А далі в методі **onCreate** створимо обробник дотику, який зчитуватиме нинішні координати та записуватиме їх у ці змінні:

```
val scrn_main: ScrollView = findViewById(R.id.scrn_main)
scrn_main.setOnTouchListener()
{v, event ->
  val action = event.action
  when (action)
  {
    MotionEvent.ACTION_MOVE ->
    {
      px = event.x.toInt()
      py = event.y.toInt()
      this.setTitle("x: " + px.toString() + ", y: " + py.toString())
    }
    else -> {}
  }
  true
}
```

Тепер щоразу, коли користувач рухатиметься по екрану, в панелі заголовку будуть виводитись поточні координати відносно верхнього лівого краю. Проте зверніть увагу, що якщо вищенаведений обробник створено для **ScrollView,** то і

рахувати координати він буде саме щодо нього. Таким чином, якщо панель заголовку в нас не прокручується, а міститься окремо над цим елементом, то при дотику до самої панелі координата **y** стане від'ємною, оскільки ми вийшли за межі **ScrollView.**

Крім того, якщо на екрані є інші значущі елементи, які передбачають взаємодію з користувачем — скажімо, кнопки, то написаний нами обробник буде перекривати їхні методи реакції на дотики. І відповідно при натисненні на кнопку додаток лише фіксуватиме координати на екрані, а не запускатиме функцію-обробник кнопки. Тож треба пам'ятати про необхідність узгодження всіх обробників між собою, аби не ставалось таких конфліктів.

Щодо збереження даних, то, крім файлів на самому пристрої, ми можемо скористатися хмарними сховищами або базами даних. Переваги баз даних очевидні — швидкість роботи, цілісність даних, надійність і підтримка роботи з багатьма користувачами одночасно (для клієнт-серверних СКБД). Однак часто є ситуації, в яких може бути простіше та зручніше скористатися файлом у хмарному сховищі.

Для прикладу розглянемо таблиці Google Sheets і їхні основні особливості:

- мають інтерфейс і формат файлів, подібний до Microsoft Excel і аналогів, що звично для багатьох користувачів;
- є повністю безкоштовними для створення та використання;
- для роботи потрібен лише Google Accout, який теж є безкоштовним;
- є доступними з будь-якої точки світу (хмарне сховище даних);
- можна надати права не лише для перегляду, а й для редагування багатьом користувачам і спільно працювати над документом у межах команди, проєкту;
- простіші у використанні, ніж класичні бази даних, адже електронні таблиці знайомі навіть не фахівцям у IT;
- не вимагають написання інтерфейсу для взаємодії з даними;
- так само, як і з баз даних, дані таблиць можна зчитати і опрацювати програмно через API і бібліотеки різних мов програмування.

Тож для невеликих проєктів, а також коли вимагається спільна робота над даними (наприклад, доповнення навчальних матеріалів у електронному засобі навчання викладачами або доповнення інформації про товари працівниками торговельної мережі тощо) можна скористатись таким хмарним сховищем для збереження даних у додатку.

Створити таблицю Google Sheets можна зі свого облікового запису в розділі «Документи» → «Таблиці» (*Docs → Sheets*). Обираємо базовий шаблон «порожня таблиця» (*blank spreadsheet*), задаємо їй назву та наповнюємо даними. Після цього поширюємо файл через меню *File → Share → Publish to web.* Далі відкривається вікно, де можна налаштувати параметри експорту документа. Зокрема можна поширити або цілий документ (*Entire Document*), або лише один із аркушів. Якщо маємо всього 1 таблицю на одному аркуші, це не суттєво. Проте важливо обрати формат експорту, серед яких: веб-сторінка **.html; .csv; .tsv; .pdf; .xlsx; .ods.**

Тут можна порекомендувати формати **.csv** або **.tsv,** які по суті є текстовими файлами, але з іншим розширенням, а тому можуть бути відкриті як у звичайному Блокноті, так і багатьма іншими засобами. При експорті таблиці в такі формати кожен запис (рядок таблиці) стає рядком текстового файлу, а дані по полях запису (колонки таблиці) відділяються між собою комами (в **.csv**) або табуляцією (в **.tsv**). При цьому **.tsv** зазвичай зручніший, оскільки в межах комірок можуть бути коми, які у форматі **.csv** автоматично вважаються за перехід до наступної колонки.

Після налаштування параметрів експорту ми отримуємо посилання такого формату: **https://docs.google.com/spreadsheets/d/e/.....&output=tsv.** Тепер ми можемо зберегти його в коді та спробувати через нього отримати доступ до даних із хмарної таблиці.

Проте про всяк випадок перед зчитуванням даних слід іще налаштувати дозволи додатку, аби він міг заходити до мережі та взаємодіяти з веб-даними. На етапі розроблення це можна прописати у файлі **AndroidManifest.xml:**

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

Натомість користувач повинен буде також дати дозвіл додатку через *Apps → Permissions* або *Apps → ... → App Info → App Permissions*. Якщо не надати цей дозвіл, усі інші функції програми працюватимуть, однак потрібні дані просто не будуть завантажуватись. Крім того, така ситуація можлива й тоді, коли в користувача вимкнено інтернет, або сервер не відповідає, або провайдер із певних причин блокує або затримує роботу з певними веб-сервісами.

Також важливо, що в таких випадках додаток самостійно не видаватиме повідомлень про помилку, сповіщень, пояснень тощо. Тож ці ситуації слід передбачити і продумати варіанти дій програми на випадок неможливості роботи з даними. Як мінімум, це може бути кнопка повторного завантаження даних, їх оновлення на випадок, коли на момент запуску програми зв'язку не було, а згодом він з'явився — інакше для оновлення доведеться повністю перезапускати додаток.

Якщо ж зв'язок є і дані завантажуються, нам слід їх зчитати та опрацювати. Для цього створимо глобальну змінну з посиланням на документ, отриманим під час експорту з Google Sheets. Це можна зробити на початку класу **MainActivity,** перед методом **onCreate:**

```
class MainActivity : AppCompatActivity() {
    var tsv_url: String = "https://docs.google.com/spreadsheets/..."
```

Тут же можна створити інші змінні потрібних типів (числові, рядкові тощо), які будуть зберігати власне дані, які ми завантажимо з таблиці. Тут доречно скористатися масивами, списками, словниками та іншими структурами даних, аби зберегти в них набір рядків і колонок, які ми надалі будемо опрацьовувати та виводити на екран додатка.

Важливо, що в Android не можна напряму працювати з інтернетом у межах звичайних функцій, оголошених у певній діяльності (*Activity*). Адже робота з мережею, як ми вже бачили вище, може спричиняти певні затримки, а іноді

відповідь від сервера взагалі відсутня. І якби вся діяльність залежала від роботи з мережею, наш додаток міг би зависати і не відповідати на дії користувачів.

Через це всі подібні дії поміщаються в окремий потік виконання (*Thread*):

```
Thread
{
  try
  {
    val url = URL(tsv_url)
    val uc: HttpsURLConnection = url.openConnection()
            as HttpsURLConnection
    val br = BufferedReader(InputStreamReader(uc.getInputStream()))
    var rline: String?
    var lines_array = emptyArray<Array<String>>()
    br.readLine()
    while (br.readLine().also { rline = it } != null)
    {
      var cur_line: String = rline!!
      var strs = cur_line.split("\t").toTypedArray()
      if (strs[1] != "" && strs[2] != "") lines_array += strs
    }
    for (line in lines_array)
    {
      var wnat: String = line[1]
      var wfor: String = line[2]
      //...
    }
    //...
  }
}
```

Цей фрагмент коду в межах окремого потоку виконання спершу з'єднується з таблицею за посиланням із глобальної змінної **tsv_url,** оголошеної вище. Далі проводиться зчитування з потоку введення в буфер (**BufferedReader**). Створюємо заготовки окремого рядка (**rline**) і масиву рядків (**lines_array**), в які будемо читати дані. Цикл **while** виконується, поки є дані для зчитування і рядок не є порожнім. На кожній ітерації записуємо в проміжну змінну **cur_line** поточний рядок. Далі розбиваємо його функцією **split()** на окремі комірки та записуємо їх у масив **strs.**

У наведеному прикладі для опрацювання взято таблицю словника іноземної мови, в якому перша колонка (індекс 0) містить номер слова, друга — українське слово, третя — іноземне. Тому для виведення слова та перекладу необхідно, аби комірки з індексами 1 і 2 не були порожніми. Для цього призначена перевірка **if,** яка в разі успіху додає поточний рядок до масиву рядків **lines_array.**

Коли цей масив наповнено, ми можемо пройти по ньому та вичитати з нього все, що нам потрібно, так само звертаючись до окремих комірок через індекси стовпців, починаючи з 0.

Зрозуміло, що відколи дані опинилися у змінних програми, надалі можна працювати з ними як завгодно. Зокрема: вивести слово та переклад у текстові

написи на екрані; створити кнопки навігації «Вперед» і «Назад», які змінюють індекс у масиві, зчитують наступне чи попереднє слово і заміняють ним нинішній вміст написів тощо. Для зручності користувачів доцільно також відстежувати, коли в результаті такої навігації масив дійшов до кінця або початку. В таких випадках можна показати сповіщення (через **Toast** або в якомусь іншому написі).

Висновки

У ході виконання лабораторної роботи визначено, яким чином можна застосовувати сучасні підходи до збереження та завантаження даних у мобільних додатках, працювати з файлами, папками, базами даних і хмарними сховищами, реалізовувати навігацію між екранами та опрацювання дотиків користувача.

Запитання для самоперевірки

1. Шляхи збереження інформації в мобільних додатках.
2. СКБД SQLite та її особливості.
3. Робота з файлами та папками в мобільному додатку.
4. Програмна генерація візуальних елементів інтерфейсу.
5. Реалізація панелей додатка для заголовку, меню тощо.
6. Реалізація навігації між екранами мобільного додатка.
7. Реалізація реакції на дотики користувача до екрану.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. **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.
2. **Чичкарьов Є. А., Зінченко О. В., Фесенко М. А.** Програмування мобільних пристроїв на Java : навч. посіб. – К. ДУІКТ: , 2023. – 222 с.
3. **Давидов М. В., Демчук А. Б., Лозинська О. В.** Програмне забезпечення мобільних пристроїв: навч. посіб. – Львів: «Новий Світ-2000», 2020. – 218 с.
4. **Поляков А. О., Федорченко В. М., Шматко О. В.** Аналіз методів технологій розроблення мобільних додатків для платформи Android: навч. посіб. – Х. : ХНЕУ ім. С. Кузнеця, 2017. – 286 с.
5. **Griffiths D., Griffiths D.** Head First Android Development, 3rd ed. O'Reilly, 2021. – 930 p.
6. **Griffiths D., Griffiths D.** Head First Kotlin: A Brain-Friendly Guide. Beijing, Boston, Farnham, Sebastopol, Tokyo : O'Reilly, 2019. – 770 p.
7. **Horton J.** Android Programming with Kotlin for Beginners. – Birmingham: Packt Publishing, 2019. – 669 p.
8. **Google Developers.** Android Studio [online]. URL : https://developer.android.com/studio.
9. **JetBrains s.r.o.** Kotlin docs [online]. URL : https://kotlinlang.org/docs/home.html.
10. **Oracle Corporation.** Java [online]. URL : https://www.java.com.
11. **JetBrains s.r.o.** IntelliJ IDEA. Getting Started [online]. URL : https://www.jetbrains.com/help/idea/getting-started.html.

ДОДАТОК А.
ШАБЛОН ТИТУЛЬНОЇ СТОРІНКИ ЛАБОРАТОРНОЇ РОБОТИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

кафедра інформаційних технологій,
штучного інтелекту і кібербезпеки

ЗВІТ

із лабораторної роботи № __

з дисципліни «Розроблення мобільних додатків»

на тему: «_____»

Варіант ___

Виконав/-ла:

Здобувач(ка) групи КН-_____

_____

Перевірив:

к.т.н., доц. Костіков М. П.

Київ — 2025

ДОДАТОК Б.
КОНТРОЛЬ ТА ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ НАВЧАННЯ

Розподіл балів за окремими елементами змістових модулів та методи поточного контролю успішності навчання здобувачів денної форми здобуття освіти

Таблиця Б.1

№ елем. змістового модуля	Елементи змістового модуля	Кількість балів		Поточний контроль навчальної роботи здобувачів
		мінімальна	максимальна	методи контролю
1	2	3	4	5
Модуль 1				
1.	**Лекційний курс.** Теми 1–8	6	10	Написання модульної контрольної роботи
	Лабораторна робота 1	12	20	Виконання і захист лабораторної роботи
	Лабораторна робота 2	12	20	Виконання і захист лабораторної роботи
	Лабораторна робота 3	12	20	Виконання і захист лабораторної роботи
	Разом за модулем	**42,0**	**70,0**	
	Диференційований залік	**18,0**	**30,0**	
	Усього за семестр	**60**	**100**	

Розподіл балів за окремими елементами змістових модулів та методи поточного контролю успішності навчання здобувачів заочної форми здобуття освіти

Таблиця Б.2

№ елем. змістового модуля	Елементи змістового модуля	Кількість балів		Поточний контроль навчальної роботи здобувачів
		мінімальна	максимальна	методи контролю
1	2	3	4	5
Модуль 1				
1.	**Індивідуальне завдання (контрольна робота)**	25	42	Виконання і захист контрольної роботи
	Лабораторна робота 3	17	28	Виконання і захист лабораторної роботи

1	2	3	4	5
	Разом за модулем	**42,0**	**70,0**	
	Диференційований залік	**18,0**	**30,0**	
	Усього за семестр	**60**	**100**	

Критерії оцінювання програмних результатів навчання здобувачів денної форми здобуття освіти за окремими елементами змістових модулів

Таблиця Б.3

Елемент модуля та критерії його оцінювання	Кількість балів
Лабораторна робота	**20**
Робота відпрацьована та вчасно захищена, надані повні обґрунтовані відповіді	18–20
Робота відпрацьована та вчасно захищена, при відповіді допущені неточності	14–17
Робота відпрацьована, відповіді неповні, допущені помилки	12–13
Робота відпрацьована, відповіді незадовільні, допущені грубі помилки	6–11
Робота не відпрацьована або дані незадовільні відповіді	0–5
Модульна контрольна робота	**10**
У роботі надано повні обґрунтовані відповіді	9–10
Відповіді неповні, допущено помилки	7–8
Відповіді неповні, допущено суттєві помилки	6
Дано незадовільні відповіді	0–5

Критерії оцінювання програмних результатів навчання здобувачів та заочної форми здобуття освіти за окремими елементами змістових модулів

Таблиця Б.4

Елемент модуля та критерії його оцінювання	Кількість балів
Лабораторна робота	**28**
Робота відпрацьована та вчасно захищена, надані повні обґрунтовані відповіді	25–28
Робота відпрацьована та вчасно захищена, при відповіді допущені неточності	21–24
Робота відпрацьована, відповіді неповні, допущені помилки	17–20
Робота відпрацьована, відповіді незадовільні, допущені грубі помилки	9–16
Робота не відпрацьована або дані незадовільні відповіді	0–8
Індивідуальне завдання (контрольна робота)	**42**

Елемент модуля та критерії його оцінювання	Кількість балів
повна відповідь	38–42
неповна відповідь або допущені деякі неточності	32–37
неповна відповідь, допущені окремі помилки	25–31
неповна відповідь, допущені суттєві помилки	15–24
незадовільна відповідь	0–14

Критерії оцінювання програмних результатів навчання здобувачів денної та заочної форм здобуття освіти (форма підсумкового контролю — диференційований залік)

27…30 балів ставиться здобувачу, який демонструє повні і глибокі знання навчального матеріалу з питань дисципліни, вільно володіє науковими термінами та проявляє високу комунікативну культуру;

23…26 балів ставиться здобувачу, який засвоїв основний навчальний матеріал, володіє необхідними знаннями з дисципліни, але при цьому допускає окремі несуттєві помилки та неточності, демонструє достатній рівень комунікативної культури;

18…22 бали ставиться здобувачу, який виявляє дещо обмежені знання навчального матеріалу, не виявляє самостійності суджень, не володіє достатніми знаннями з дисципліни, демонструє недоліки комунікативної культури;

0…17 балів ставиться здобувачу, який не володіє необхідними знаннями, науковими термінами в області дисципліни, демонструє низький рівень комунікативної культури.