МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

БУТІН ВОЛОДИМИР СЕРГІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«_____» _____ 20__ р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ТЕСТУВАННЯ ЗНАНЬ З КОМП'ЮТЕРНИХ НАУК

Спеціальність 122 «Комп'ютерні науки»
Кваліфікаційна (бакалаврська) робота

Керівник:
Бабаков Р.М., професор
кафедри інформаційних технологій,
д. т. н., доцент

_____
(Підпис)

Оцінка _____/_____/_____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(Підпис)

Вінниця 2025

АНОТАЦІЯ

Бутін В.С. Мобільний застосунок для тестування знань з комп'ютерних наук. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2025.

У кваліфікаційній (бакалаврській) роботі розроблено мобільний застосунок для операційної системи Android, призначений для тестування знань у сфері комп'ютерних наук. В рамках виконання роботи було проведено дослідження сучасних підходів до розробки мобільних застосунків, зокрема на платформі Android з використанням декларативного UI-інструментарію Jetpack Compose, а також методів зберігання та синхронізації даних. У застосунку реалізовано систему аутентифікації користувачів, функціонал проходження тестів з різними типами питань, завантаження контенту з локальної бази даних SQLite та збереження результатів у Firebase Firestore. Застосунок створено з використанням Kotlin та Jetpack Compose.

Ключові слова: мобільний застосунок, тестування знань, комп'ютерні науки, Android, Kotlin, Jetpack Compose, Firebase, SQLite.

84 с., 40 рис., 30 джерел.

ABSTRACT

Butin V.S. Mobile application for testing knowledge of computer science. Specialty 122 “Computer Science”, educational program “Computer Science”. Vasyl' Stus Donetsk National University, Vinnytsia, 2025.

In the qualification (bachelor's) work, a mobile application for the Android operating system was developed, designed to test knowledge in the field of computer science. As part of the work, a study was conducted of modern approaches to the development of mobile applications, in particular on the Android platform using the declarative UI toolkit Jetpack Compose, as well as methods of data storage and synchronization. The application implements a user authentication system, the functionality of passing tests with different types of questions, downloading content from a local SQLite database, and saving the results to the Firebase Firestore. The application was created using Kotlin and Jetpack Compose.

Keywords: mobile application, knowledge testing, computer science, Android, Kotlin, Jetpack Compose, Firebase, SQLite.

84 p., 40 figures, 30 sources.

ЗМІСТ

ВСТУП

Стрімкий розвиток інформаційних технологій та їх глибоке проникнення у всі сфери людської діяльності зумовлюють постійно зростаючу потребу у кваліфікованих фахівцях у галузі комп'ютерних наук. Ефективне навчання та об'єктивна оцінка знань стають ключовими факторами підготовки таких спеціалістів. У цьому контексті, мобільні застосунки відкривають нові можливості для організації інтерактивного навчального процесу та самостійної перевірки знань, забезпечуючи доступність, гнучкість та персоналізацію навчання. Використання мобільних платформ для тестування дозволяє студентам, абітурієнтам та фахівцям перевіряти рівень своєї підготовки у зручний час та в будь-якому місці, що сприяє кращому засвоєнню матеріалу та виявленню прогалин у знаннях. Особливо актуальною є розробка спеціалізованих застосунків, адаптованих до специфіки комп'ютерних наук, які охоплюють широкий спектр тем від основ програмування до складних алгоритмів. Таким чином, створення мобільного застосунку для тестування знань з комп'ютерних наук є важливим та своєчасним завданням, що відповідає сучасним освітнім трендам.

Мета дослідження полягає у розробці мобільного застосунку для операційної системи Android, призначеного для інтерактивного тестування знань користувачів у сфері комп'ютерних наук, з реалізацією функціоналу аутентифікації, підтримкою різних типів тестових завдань та механізмом збереження результатів.

Завдання дослідження:

- Проаналізувати предметну область, сучасні тенденції в освітніх технологіях і функціонал аналогічних застосунків.
- Сформувати функціональні та нефункціональні вимоги до мобільного застосунку.
- Обґрунтувати вибір інструментів і технологій для розробки застосунку.
- Спроєктувати архітектуру, модульну структуру та бази даних.

- Реалізувати модулі аутентифікації, вибору категорій, тестування, збереження та аналізу результатів.
- Провести тестування застосунку.
- Узагальнити результати розробки, сформулювати рекомендації щодо використання застосунку.

Об’єктом дослідження є процес розробки та реалізації мобільного застосунку, призначеного для тестування знань користувачів.

Предметом дослідження є архітектура, алгоритми функціонування, програмні засоби та технології, що використовуються для створення мобільного застосунку тестування знань під платформу Android.

Теоретичне значення роботи полягає в узагальненні підходів до проєктування та розробки освітніх мобільних застосунків з використанням сучасних технологій, таких як Kotlin та Jetpack Compose.

Практичне значення полягає у створенні готового до використання мобільного застосунку, який може слугувати ефективним інструментом для самоперевірки або закріплення знань студентами та абітурієнтам у сфері комп'ютерних наук, а також може бути інтегрований в освітній процес як допоміжний засіб.

Структура кваліфікаційної (бакалаврської) роботи.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел. У вступі обґрунтовано актуальність теми, сформульовано мету та завдання дослідження, визначено об’єкт, предмет, теоретичне та практичне значення роботи. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення, на основі цього сформульовано вимоги до власного застосунку та розглянуто інструментарій разом з технологіями для розробки. Другий розділ присвячено проєктуванню мобільного застосунку, включаючи розробку архітектури, модульної структури та баз даних. У третьому розділі описано процес реалізації програмного продукту, використані інструментальні засоби та результати тестування. У висновках підсумовано результати виконаної роботи та окреслено перспективи подальших досліджень.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз системи тестування знань у сфері комп'ютерних наук

Оцінювання знань у галузі комп'ютерних наук дедалі більше покладається на цифрові технології, які роблять навчальний процес більш інтерактивним, персоналізованим і ефективним [1][2]. Мобільне навчання здатне підвищити мотивацію студентів і ефективність засвоєння матеріалу за рахунок частих взаємодій, безперервного зворотного зв'язку та динамічних оцінювань, що долають обмеження традиційних методів викладання [2]. Враховуючи ці властивості, сучасні системи тестування знань у комп'ютерних науках включають широкий спектр інструментів – від класичних систем управління навчанням, що забезпечують онлайнові тести й контролі, до спеціалізованих автоматизованих платформ для оцінювання програмних рішень і симуляцій. Наприклад, у навчальних курсах з програмування використовують автоматизовані системи, що перевіряють код студента заздалегідь визначеними правилами й тестовими сценаріями. Такі системи існують від початку навчання програмуванню й нині широко впроваджені: упродовж останнього десятиліття публікуються тисячі наукових праць з автоматизованого оцінювання в Computer Science-освіті [3].

У той же час різноманіття доступних платформ і методів свідчить про відсутність універсального рішення. Практично кожен курс або заклад іноді розробляє власні інструменти оцінювання, оскільки складно інтегрувати готові рішення під специфічні навчальні потреби [3]. Це означає, що системи тестування знань у комп'ютерних науках охоплюють як стандартизовані тести (тести з вибором відповіді, завдання на встановлення відповідності, питання з короткою відповіддю тощо), так і комплексні практичні завдання (програмні проєкти, симуляції, кейс-завдання). Загалом можна зазначити, що цифрові технології забезпечують не лише автоматизацію перевірки знань, а й нові можливості для учасників освітнього процесу – наприклад, адаптивні

оцінювання з урахуванням індивідуального рівня студента, а також негайний зворотний зв’язок [2][3]. Однак застосування таких технологій у навчанні та тестуванні вимагає обережного підходу: слід враховувати необхідність підготовки викладачів, наявність інфраструктури та потенційні проблеми приватності й зловживання даними [1][4].

Отже, сучасна система тестування знань у сфері комп’ютерних наук являє собою багаторівневий конгломерат рішень – від традиційних контрольних робіт до інтелектуальних мобільних і веб-орієнтованих застосунків, що автоматизують оцінювання. Цей різноплановий підхід дозволяє перевіряти як теоретичні знання (алгоритми, теорія інформатики, комп’ютерна грамотність), так і практичні навички (програмування, моделювання систем), забезпечуючи різні рівні глибинності оцінки. Перспективи впровадження мобільних технологій та інноваційних форматів тестів роблять оцінювання ще більш гнучким і доступним, відповідаючи потребам сучасного динамічного освітнього середовища [1][2]. Саме тому мобільні технології стали невід'ємною складовою сучасної освіти, оскільки зростання поширеності смартфонів і планшетів серед студентів створює передумови для використання мобільних додатків у навчанні та оцінюванні [4]. Так, дослідження показують, що інтеграція мобільних платформ у навчальний процес сприяє підвищенню мотивації та залученості студентів. У мобільному середовищі учні взаємодіють з навчальним контентом у зручний для себе спосіб – практично в будь-який час і в будь-якому місці. Наприклад, українські науковці відзначають, що використання мобільних застосунків у вищій освіті «має позитивний ефект на підвищення рівня умінь і навичок учнів та їх мотивації до навчання», сприяє індивідуалізації навчального процесу та організації продуктивної самостійної роботи поза аудиторією [5]. Аналіз літератури також підтверджує успішність mobile-learning у різних дисциплінах, включаючи комп’ютерні науки, де мобільні інструменти допомагають покращити результати навчання з програмування, алгоритмів та веб-технологій [2].

Сьогодні мобільні додатки дозволяють не лише подавати навчальний матеріал, але й інтерактивно оцінювати знання. Вони здатні формувати адаптивне навчальне середовище з миттєвим зворотним зв’язком і можливістю постійної самоперевірки. Приміром, тестування за допомогою мобільних пристроїв може включати короткі квести, мікроконтенти та ігрові елементи, що розвивають критичне мислення і сприяють кращому засвоєнню матеріалу [2]. Однак успішна імплементація мобільних технологій вимагає системного підходу: важливі планування, підготовка викладачів і адаптація контенту під специфіку мобільних платформ [2][4].

Разом з тим слід враховувати й обмеження. Мобільні пристрої мають менш зручний інтерфейс для довготривалого вводу тексту чи розгортання складних завдань, тому не всі види тестових завдань однаково доречні у мобільному форматі. Крім того, необхідно забезпечити доступ до стабільного Інтернету, а також подолати фактори відволікання і ризики цифрової втоми. Незважаючи на це, переваги мобільного навчання – зокрема гнучкість, індивідуалізація і підвищена зацікавленість студентів – роблять його перспективним засобом підтримки процесу тестування знань [4][5]. Таким чином, роль мобільних технологій полягає у розширенні можливостей традиційного навчання: вони створюють додаткові канали для навчання і оцінювання, дозволяють оперативно фіксувати результати і адаптувати навчальний контент відповідно до індивідуальних потреб учнів.

Сучасні системи оцінювання знань передбачають використання різних типів тестових завдань, кожен з яких має свої особливості, переваги і недоліки. Наприклад, завдання з множинним вибором і завдання «так/ні» є одними з найпоширеніших форм контролю знань. Вони складаються зі ствердження (або запитання) та кількох варіантів відповіді (або двох – «так»/«ні»), що дає змогу автоматизувати оцінювання. Дослідження з комп’ютерних наук свідчать, що питання з множинним вибором можуть слугувати стимулом для студентів та ефективним способом перевірки засвоєних концепцій; зокрема, вони показали здатність таких питань «підвищувати мотивацію та ефективність перевірки

вивчених концепцій програмування» [6]. Однак подібні завдання зазвичай перевіряють здатність відтворювати знання або швидко застосовувати зрозумілі поняття, тоді як вищі когнітивні рівні (аналіз, синтез) можуть залишатися поза межами оцінювання.

Інший тип – завдання із заповненням пропусків і короткі відповіді – вимагає від студента самостійно сформулювати відповідь у кількох словах або реченнях. Такі тести легше моделюють реальні уміння висловлювати поняття, але їх автоматизована перевірка складніша: часто застосовують ключові слова для пошуку чи спеціальні алгоритми (наприклад, аналіз синтаксису відповіді). Есе-розгорнуті відповіді дозволяють оцінити вміння конструктивно аргументувати та інтегрувати знання, проте потребують ручної перевірки або складних алгоритмів лінгвістичного аналізу, що робить їх застосування у великих класах менш масштабованим.

У галузі інформатики особливе значення мають практичні завдання з програмування та моделювання. Ці завдання можуть бути як закритими, так і відкритими. Наприклад, студенту пропонують написати фрагмент коду або алгоритм для вирішення певної задачі. Такі open-ended programming tasks передбачають творчу роботу над кодом, що сприяє глибшому розумінню матеріалу і мотивації [7]. Однак автоматична перевірка програмного коду є нетривіальною проблемою: систему потрібно налаштувати на перевірку правильності виконання програми, стилю коду, безпеки тощо. Найпростіші рішення використовують набір тестових даних і порівняння результатів, але це обмежує типи завдань простими функціональними перевірками . В останні роки розробляють більш складні методи, в тому числі із застосуванням технологій штучного інтелекту (генеративних мовних моделей), щоб аналізувати відкриті завдання і надавати розгорнутий зворотний зв’язок з коду [7].

Ще один поширений формат – задачі на налагодження коду (debugging), де студенти повинні знайти помилку у заданому програмному фрагменті. Такі завдання перевіряють логічне мислення і розуміння синтаксису/семантики мов програмування: студент повинен розпізнати, чому програма не працює, і

запропонувати виправлення. Автоматизована перевірка налагодження може здійснюватися шляхом порівняння з правильним рішенням або аналізу тестів, однак вимагає спеціальних механізмів оцінювання.

Також використовують візуальні та інтерактивні тести – наприклад, перетягування блоків (drag-and-drop) для впорядкування кроків алгоритму чи створення простого коду, побудова блок-схем, робота з емуляціями пристроїв [5]. Такі формати є зручними для мобільних та веб-застосунків, оскільки вони інтуїтивні й залучають до вивчення за допомогою взаємодії з інтерфейсом. Проте інтеграція таких елементів у систему тестування потребує спеціальної реалізації, а повноцінна автоматична оцінка складних інтерактивних завдань може бути обмежена (наприклад, за допомогою звірки фінального стану чи часу виконання).

Розглянемо узагальнено основні типи тестових завдань та їх ключові особливості:

Множинний вибір: один або кілька правильних варіантів з кількох пропозицій. Легко автоматизується, дозволяє тестувати фактичні знання та базові концепції [6]. Ефективний для швидкого оцінювання, але обмежений у виявленні глибинного розуміння.

Питання так/ні (True/False): ствердження із вибором «так» або «ні». Швидке оцінювання простих фактів, часто застосовується у первинному контролі знань.

Заповнення пропусків: студент вписує одне слово, число або коротку фразу на позначене місце. Перевірка може бути автоматизованою (за ключовими словами) або комбінованою. Корисне для перевірки знання термінів чи елементів коду.

Коротка відповідь: кілька слів чи речень у відповідь на запитання. Перевірка вимагає аналізу тексту, однак дозволяє оцінити обізнаність з темою без деталізації.

Проєктні/творчі завдання: більш розгорнуті відкриті завдання, що вимагають описових відповідей або реалізації проєкту. Дає змогу оцінити

аналітичні й синтетичні навички, але потребує ручного аналізу або складних алгоритмів автоматичної перевірки.

Програмні завдання: написання коду чи алгоритму для розв'язання поставленої проблеми. Оцінка проводиться через тестові сценарії, симуляцію або аналіз вихідного коду. Формат сприяє глибокому розумінню, але потребує розширених засобів автоматизованого тестування [7].

Задачі на налагодження: студент знаходить і виправляє помилку у готовому коді. Перевіряє розуміння програмної логіки й синтаксису. Результат автоматично перевіряється шляхом аналізу коректності виправленого коду.

Інтерактивні/візуальні: перетягування елементів, складання блоків коду, етапів алгоритмів чи графічних формул. Створюють інтерактивний досвід, придатні для мобільних тестів. Перевірка може базуватись на правильності зібраної послідовності або стану моделі.

Таким чином, вибір типу тестового завдання залежить від цілей контролю: завдання множинного вибору і «так/ні» підходять для швидкої перевірки фактичних знань та широкого охоплення теми, заповнення пропусків і короткі відповіді дозволяють оцінити конкретні уміння студента формулювати поняття, а проєктні та програмні завдання – глибоке розуміння та практичні навички [6][7]. Інтерактивні формати роблять тестування більш залучаючим, хоч і потребують додаткового програмного забезпечення.

Отже, система тестування знань у комп'ютерних науках повинна використовувати комбінований підхід, підбираючи різні типи завдань відповідно до мети контролю і контексту навчання. Кожен вид завдання має свої переваги й обмеження: правильно поєднуючи їх, можна забезпечити повнішу та об'єктивнішу оцінку як теоретичних знань, так і практичних вмінь студентів. Враховуючи сучасні тренди, перспективним є розвиток інтелектуальних систем оцінювання, що поєднують різні формати завдань та автоматичний аналіз, що дозволяє давати глибокий зворотний зв'язок і підтримувати процес навчання у реальному часі [7].

1.2 Огляд існуючих мобільних застосунків для тестування знань

Сучасний ринок мобільних застосунків пропонує значну кількість рішень для тестування та перевірки знань у різних галузях, включаючи комп'ютерні науки та інформаційні технології. Для визначення ніші та обґрунтування актуальності розроблюваного продукту було проведено аналіз існуючих аналогів. Розглянемо два типових представники: "CS IT - Computer Science MCQs" та "Programming Quiz" [8][9].

Перший застосунок "CS IT - Computer Science MCQs"

Позиціонується як застосунок для підготовки до іспитів та перевірки знань з широкого спектра тем комп'ютерних наук та інформаційних технологій. Він пропонує користувачам набір запитань з множинним вибором.

Рисунок 1.1 – Меню вибору категорій

Застосунок містить значну кількість запитань, що охоплюють різні аспекти IT, такі як операційні системи, комп'ютерні мережі, структури даних, мови програмування, бази даних тощо.

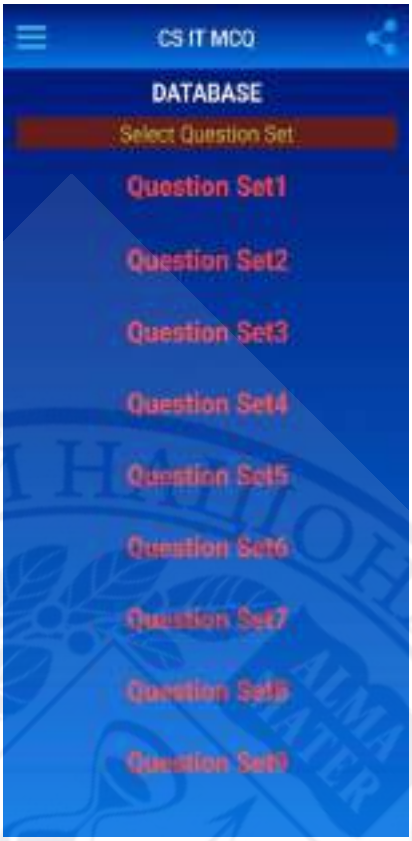

Рисунок 1.2 – Набір питань в категорії

Питання згруповані за тематичними категоріями, що дозволяє користувачеві фокусуватися на конкретних областях знань.

Рисунок 1.3 – Вікно з запитаннями

Тести з одним варіантом відповіді, присутня можливість одразу переглянути правильну відповідь.

Другий застосунок "Programming Quiz":

також є мобільним застосунком, орієнтованим на перевірку знань у сфері програмування. Його основна мета – надати користувачам можливість швидко оцінити свій рівень розуміння різних концепцій програмування.

Рисунок 1.4 – Меню програми

В головному вікні програми містяться одразу декілька функціональних карток, основну цінність представляє картка "Quiz".

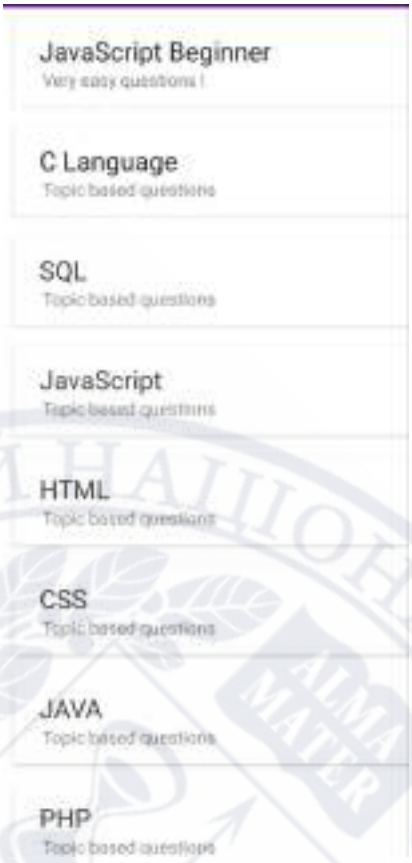

Рисунок 1.5 – Вибір тематично тесту по певній мові програмування

Застосунок концентрується на питаннях, пов'язаних з мовами програмування, парадигмами, інструментами розробки та базовими концепціями.

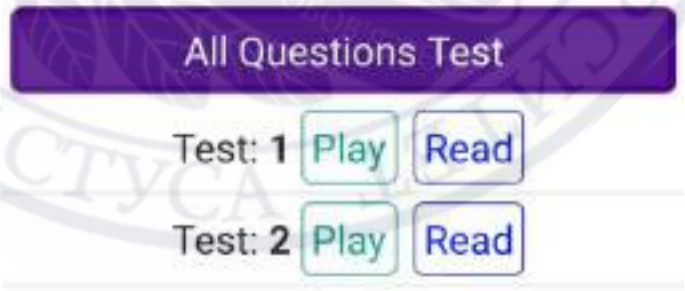

Рисунок 1.6 – Екран початку тестування

Є можливість перегляду тесту та відповідей перед його проходженням.

Рисунок 1.7 – Екран тесту

Дається можливість вибору однієї правильної відповіді, а також варіанту "Жодної правильної відповіді".

Аналіз розглянутих застосунків дозволяє виділити як загальні сильні сторони подібних рішень, так і їхні типові недоліки, особливо в контексті потреб українського користувача.

Загальні переваги аналогів:

Доступність великої бази знань: надають доступ до значного обсягу тестових матеріалів з різних галузей комп'ютерних наук.

Зручний формат самоперевірки: дозволяють користувачам швидко та в будь-який час перевірити свій рівень підготовки.

Категоризація контенту: структурування питань за темами полегшує цілеспрямовану підготовку.

Загальні недоліки та обмеження аналогів:

- Мовний бар'єр: переважна більшість подібних застосунків є англомовними, що створює труднощі для більшості україномовних

користувачів, які не володіють англійською на достатньому рівні, особливо при вивченні складної технічної термінології.

- Обмеженість типів запитань: часто домінує формат запитань з єдиним правильним вибором, що не завжди дозволяє комплексно оцінити глибину розуміння матеріалу. Відсутність підтримки запитань з множинним вибором чи типу "так/ні" обмежує варіативність тестування.
- Проблемний UI/UX дизайн: інтерфейси мають певні графічні недоліки та не можуть запропонувати особливої кастомізації.

Існуючі рішення показують ряд корисних ідей, які слід застосувати чи покращити у власному проекті.

1.3 Формулювання вимог до мобільного застосунку

На основі аналізу предметної області, існуючих рішень та потреб потенційних користувачів, було сформульовано функціональні, нефункціональні вимоги та вимоги до користувацького інтерфейсу та досвіду, які стануть основою для проєктування та розробки мобільного застосунку "Computer Science Testing".

Розглянемо функціональні вимоги:

Система автентифікації та авторизації користувачів: можливість реєстрації нового користувача, вхід існуючого користувача, вихід користувача, керування сесією.

Управління тестовим контентом та вибір тесту: відображення категорій тестів, вибір категорії для тестування, завантаження питань.

Процес проходження тестування: відображення питання, підтримка різних типів питань (одна правильна відповідь, кілька правильних відповідей, питання розряду так/ні), вибір відповіді, навігація між питаннями, реалізація таймера (можливість відстежити час до закінчення тесту), автоматичне завершення тесту через закінчення часу таймера, функція завершення тесту.

Обробка, збереження та відображення результатів: підрахунок результатів, відображення результатів тесту, збереження результатів тесту, детальний перегляд з аналізом помилок.

Перейдемо до нефункціональних вимог:

Продуктивність**:** час відгуку, швидкість завантаження питань та ефективне використання ресурсів телефону – повинно бути добре реалізовано.

Надійність: стабільна робота застосунку, коректне оброблення помилок.

Зручність використання: зрозумілий інтуїтивний інтерфейс, легка навігація, мінімальна кількість дій для збереження концентрації користувача. Безпека: дані повинні безпечно передаватися, облікові записи та їх паролі повинні зберігатися у зашифрованому вигляді.

Сумісність: підтримка версій Android 6.0 та вище, адаптивність інтерфейсу до різних телефонів.

Підтримуваність: код повинен бути читабельним, добре структурованим та модульним, щоб можна було легко вносити зміни, виправляти помилки та розвивати і на далі.

Вимоги до користувацького інтерфейсу (UI) та досвіду користувача (UX):

Візуальний дизайн: стиль повинен бути зручним та привабливим з єдиною кольоровою схемою та типографікою, мати чіткі зручні іконки.

Навігація та інформаційна архітектура: проста та логічна з зрозумілою ієрархією інформації. Акцент на використанні стандартних елементів навігації.

Досвід користувача : дизайн та функціонал орієнтовані на юзера, інтерфейс без перевантаження лишньою інформацією чи елементами, консистентного типу.

1.4 Вибір інструментальних засобів та технологій розробки

Вибір платформи розробки (Android).

Обрана платформа Android обгрунтовується її домінуючим становищем на ринку мобільних ОС і відкритістю для розробників. Android є найпоширенішою ОС для смартфонів – за даними StatCounter, світова частка Android перевищує

72 %, (в Україні – близько 70 %) [10][11]. Відтак застосунок матиме широку аудиторію. Крім того, Android побудовано на відкритому коді Linux (Android Open Source Project) [12], що дає змогу гнучко налаштовувати підсистеми, пристрої та розширювати функціонал. Екосистема Android підтримується потужними інструментами від Google (SDK, емулятор, Google Play тощо) і великою спільнотою розробників.

Рисунок 1.8 – Android

Серед переваг варто відзначити:

- Великий ринок і різноманіття пристроїв – через відкритість, платформу використовують виробники як флагманських, так і бюджетних пристроїв, що розширює охоплення аудиторії [13].
- Гнучкість і налаштування – розробники мають доступ до системних API і можуть адаптувати ОС під власні потреби [12].
- Багатий набір бібліотек – Android підтримує численні Jetpack-бібліотеки (UIKit, LiveData, Navigation тощо) і сервіси (Firebase), що прискорюють розробку.

Недоліком вибору Android є фрагментація середовища: різні пристрої мають відмінні характеристики і версії ОС, що ускладнює тестування і гарантування однакового досвіду. Фрагментація веде до затримок з оновленнями

безпеки і потребує додаткових зусиль під час розробки і тестування, хоча й є логічним наслідком відкритої архітектури Android [13]. Проте ці недоліки компенсуються великими перевагами у вигляді ширшої аудиторії та можливості швидкого залучення готових бібліотек і сервісів.

Вибір мови програмування (Kotlin).

Для розробки клієнтської частини застосунку обрана мова Kotlin. Google офіційно підтримує Kotlin як сучасну мову розробки Android [14], і її використання помітно зросло: понад 95 % найпопулярніших Android-додатків написані на Kotlin [14]. Kotlin є статично типізованою мовою, що дозволяє писати високо виразний код із мінімальним шаблонним наповненням. Вбудована підтримка null-безпечності значно зменшує ризик виникнення NullPointerException у порівнянні з Java, що робить додатки більш стабільними [15]. Наприклад, у документації Google зазначено, що програми з Kotlin на 20 % менш схильні до аварійних збоїв саме завдяки строгій перевірці null [15]. Крім того, Kotlin повністю сумісний з Java: можна поступово використовувати існуючі Java-бібліотеки і легко поєднувати код на обох мовах.

Рисунок 1.9 – Kotlin

Переваги Kotlin включають скорочення коду, підвищену продуктивність розробки та зручні інструменти IDE (автодоповнення, рефакторинг) [15]. Недоліками можна вважати необхідність навчання розробників новій мові та

трохи більшу вагу APK у порівнянні з аналогічним Java-кодом, але ці мінуси з лишком компенсуються поліпшеннями продуктивності і якості коду [14][15].

Вибір середовища розробки (Android Studio) та системи збірки (Gradle).

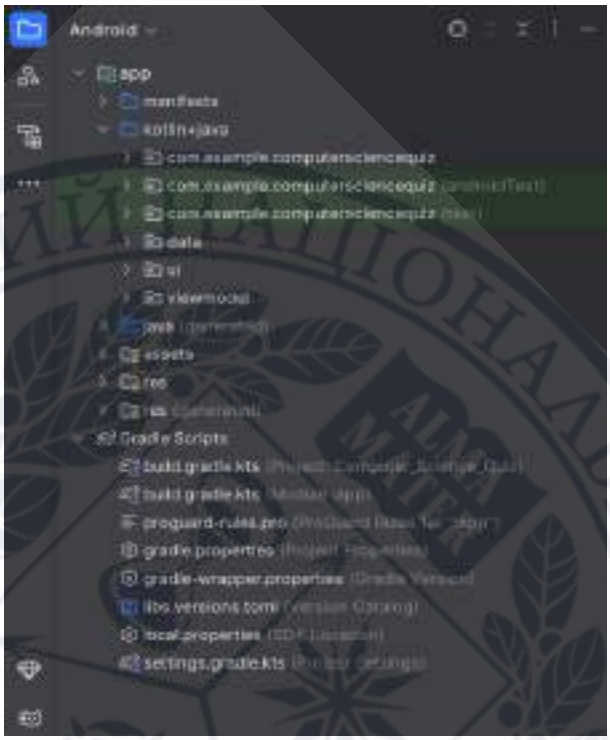

Рисунок 1.10 – Вікно Android Studio

Android Studio — офіційне інтегроване середовище розробки (IDE) для Android, створене на базі IntelliJ IDEA [16]. Воно надає розробнику повноцінний інструментарій: зручний код-редактор з автодоповненням, вбудовані емулятори Android та інструменти профілювання, а також засоби налагодження. Android Studio вже має «з коробки» підтримку Kotlin і Jetpack Compose, що пришвидшує початок розробки.

Проєкт у Android Studio будується за допомогою системи Gradle. Gradle – це гнучка система побудови, що трансформує вихідний код та ресурси у фінальний APK [17].

Рисунок 1.11 – Gradle

Використання Gradle дозволяє налаштувати різні конфігурації збірки (наприклад, відладочні та релізні), керувати залежностями бібліотек і виконувати складні сценарії побудови. Завдяки Gradle, Android Studio автоматично синтезує потрібні файли, виконує об'єднання файлів тощо, що спрощує управління проєктом. Основні можливості Android Studio та Gradle:

- Гнучка система збірки – Gradle поєднаний з Android Studio і керує компіляцією коду, ресурсів та пакуванням APK [16][17].
- Інтеграція інструментів – є швидкий емулятор, Live Edit (миттєва перезагрузка UI при розробці на Compose), візуальні редактори макетів, засоби тестування і аналізу коду.
- Уніфіковане середовище – в одному IDE можна редагувати код, слідкувати за логами, працювати з системою контролю версій і переглядати ієрархію ресурсів.

До недоліків відносять великий розмір IDE та ресурсомісткість: Android Studio потребує достатньо пам'яті і часу на індексацію проєкту. Іноді перша збірка з Gradle може займати значний час, особливо на слабших машинах. Проте оновлення Gradle і механізми кешування з часом значно знизили ці витрати, а інтегровані інструменти виправдовують затрати ресурсів зростанням продуктивності розробки.

Підхід до розробки UI (Jetpack Compose та Material Design 3).

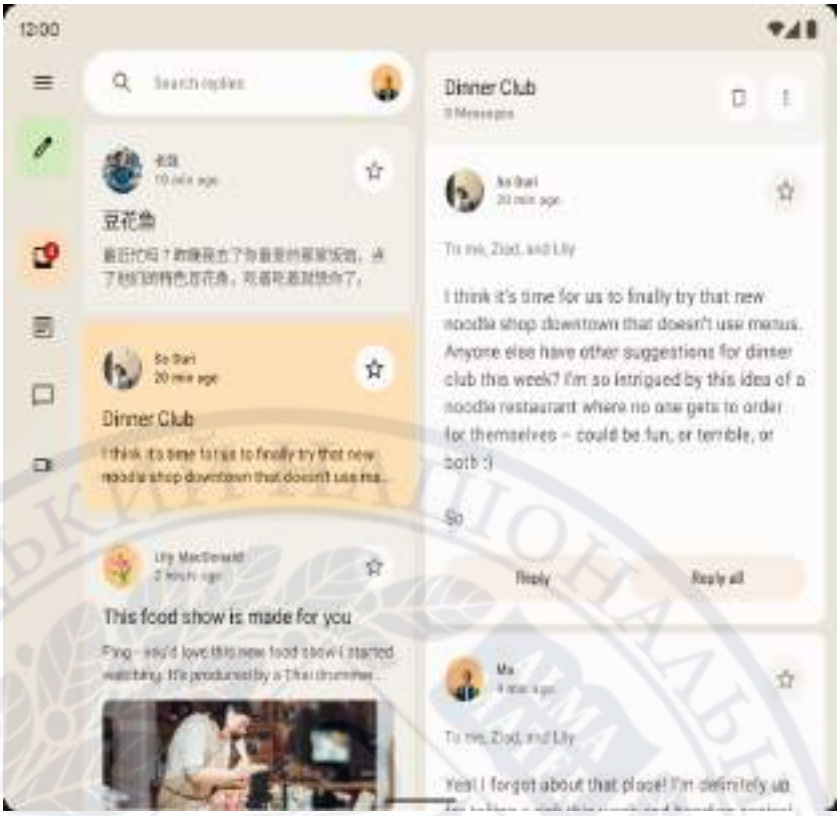

Рисунок 1.12 - Приклад застосування Material Design 3

Для створення інтерфейсу користувача використовується Jetpack Compose — сучасний декларативний UI-фреймворк від Google. Compose описує інтерфейс як набір обчислюваних функцій на Kotlin, що спрощує логіку побудови екрану. Він дозволяє розробникам писати значно менше коду, оскільки один і той самий елемент інтерфейсу можна описати в кілька разів компактніше у порівнянні з традиційними XML-макетами [18]. В документації Google наголошується, що Compose пришвидшує і полегшує розробку UI, даючи змогу швидко оживлювати додатки з мінімальними зусиллями [18]. Завдяки єдиним Kotlin-описам компонентів легше підтримувати послідовність коду й уникати проблем з синхронізацією між кодом та XML.

Інтерфейс оформляється відповідно до концепції Material Design 3 (Material You) — останньої версії гайдлайнів Google для дизайну додатків. Compose має готові компоненти Material 3, які враховують сучасні стандарти темізації та рухів [19]. Material 3 включає адаптивну (динамічну) палітру кольорів на основі системної теми і спрощене типографічне масштабування, що

дозволяє створювати привабливі та уніфіковані UI за допомогою мінімальної конфігурації [19]. Так, бібліотека androidx.compose.material3 забезпечує автоматичне підставляння основних кольорів, розмірів шрифтів і форм із можливістю легкого налаштування.

Переваги такого підходу: декларативна побудова інтерфейсу, швидкий зворотний зв'язок (через попередній перегляд у IDE), а також готові бібліотеки компонентів із сучасним дизайном. Недоліки: Jetpack Compose — відносно нова технологія, тому для неї ще відсутній великий набір сторонніх плагінів, а компіляція UI-функцій може бути більш ресурсомісткою, ніж у старій системі View (хоча з часом швидкість збірки значно покращується).

Вибір архітектурного патерну (MVVM).

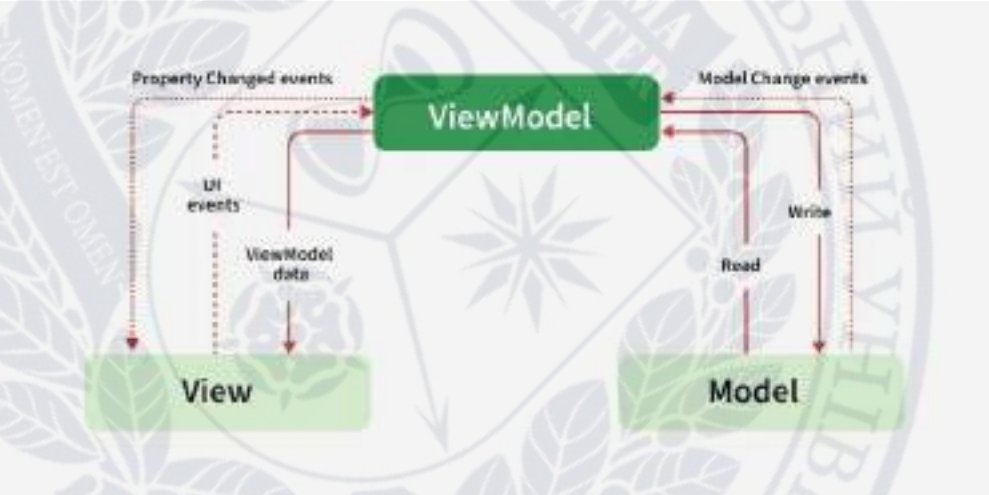

Рисунок 1.13 – Принцип роботи патерну Model – View – ViewModel [20]

У цій моделі View (інтерфейс) залежить від ViewModel, а Model забезпечує доступ до даних (наприклад, до бази даних чи інтернету). ViewModel містить логіку презентації і стан екрану, але не має посилань на елементи UI [21]. Google рекомендує MVVM для Android-розробки, і вона має широку підтримку інструментарію [21]. Завдяки MVVM можна безпечно розподіляти код: View лише підписується на зміни даних у ViewModel, а ViewModel взаємодіє з Model/репозиторієм для отримання або зберігання даних.

Це розділення дозволяє, наприклад, зберігати стан користувацького інтерфейсу через повороти екрану без перевірки під час кожного повторного

створення екрану програми чи його складових частин. Таким чином, MVVM спрощує тестування та покращує надійність. Головні переваги: вища масштабованість і зручність підтримки коду, особливо у великих проєктах. Недоліком MVVM вважається необхідність введення проміжного рівня (ViewModel) навіть для простих екранів, що трохи ускладнює початкову структуру проєкту. Проте ці витрати виправдовуються збільшенням стійкості архітектури й підтримкою сучасних практик (корутинації, залежностей через Hilt тощо).

Локальне зберігання даних (SQLite + Room).

Для збереження даних на пристрої використовуємо SQLite – вбудовану у Android реляційну СУБД. На верху SQLite накладається бібліотека Room Persistence, яка спрощує роботу з базою даних. Room забезпечує абстракцію над SQLite: на основі анотованих класів Entity та DAO автоматично генерується SQL-код, а помилки запитів виявляються на етапі компіляції [22]. Завдяки Room не треба писати вручну багато шаблонного коду для створення й оновлення БД. Модель даних відображається на об’єкти Kotlin, що значно полегшує читання та запис даних. У локальну БД можна зберігати, наприклад, відкладені відповіді користувача, результати тестів або кешовані запитання.

Рисунок 1.14 – SQLITE

Основні переваги SQLite + Room: перевірені реляційні запити, оптимізація під великий обсяг локальних записів, повна робота офлайн. До недоліків можна

віднести необхідність керувати версіями схеми (під час оновлень) та додаткові розміри коду на опис сутностей і DAO. Однак переваги надійності та потужності запитів з лишком компенсують ці витрати.

Хмарні технології (Firebase Authentication, Firestore).

Для зберігання даних у хмарі та аутентифікації користувачів обрано продукти Firebase. Firebase Authentication надає готові бекенд-сервіси й SDK для реалізації безпечного входу та реєстрації користувачів [23]. Він підтримує вхід за паролем, через соціальні мережі (Google, Facebook та ін.), телефону тощо, пропонуючи «end-to-end» рішення з мінімальним кодом [23]. Це дозволяє швидко додати в застосунок можливості керування обліковими записами без розробки власної системи автентифікації. Cloud Firestore – гнучка масштабована NoSQL-база даних від Google. Згідно з офіційними даними, Firestore забезпечує «гнучке, масштабоване зберігання даних в хмарі» з синхронізацією в реальному часі [24]. Firestore зберігає документи в колекціях і автоматично оновлює їх на всіх пристроях під час змін, а також має підтримку офлайн-кешування [24]. У межах реалізації даного застосунку, Firestore використовуватиметься для зберігання результатів користувачів та інших спільних даних.

Рисунок 1.15 - Firebase

Переваги Firebase: швидка розробка (менше часу на налаштування бекенду) та масштабованість (сервіси Google).

Недоліки: залежність від доступу в інтернет (хоча Firestore частково підтримує офлайн), а також потенційні витрати при великому навантаженні на читання/запис. Проте для освітнього застосунку з невеликою кількістю користувачів ці сервіси ідеально підходять завдяки стабільності й інтеграції з Android.

Вибір бібліотеки для навігації (Jetpack Navigation Compose).

Для управління переходами між екранами використовується офіційна бібліотека Navigation Compose. Цей компонент забезпечує декларативне описання навігаційного графа і керування стеком екранів у додатках на Compose [25]. У цій системі NavHost виступає як контейнер, що відображає поточний екран відповідно до поточного маршруту, а NavController є інструментом, який дозволяє керувати цими маршрутами. За допомогою NavController і NavHost легко визначати маршрути між функціями-екранами, передавати аргументи і обробляти повернення назад.

Рисунок 1.16 – Navigation Compose

Головними перевагами бібліотеки є уніфікованість рішення та гарантії цілісності при навігації (наприклад, безпечне передавання даних між екранами). Мінусів практично немає, хіба що доведеться додати залежність і трохи освоїти

API Navigation Compose, але це швидко окупається простотою розробки навігації.

Висновок до першого розділу

У першому розділі даної кваліфікаційної роботи було проведено комплексний аналіз предметної області, що стосується систем тестування знань у сфері комп’ютерних наук, та здійснено огляд існуючих мобільних застосунків-аналогів. На основі цього дослідження було сформульовано ключові функціональні та нефункціональні вимоги до розроблюваного мобільного застосунку, а також обґрунтовано вибір інструментальних засобів та технологій, необхідних для його реалізації.

Аналіз систем тестування знань засвідчив стрімке зростання ролі цифрових, зокрема мобільних, технологій в освітньому процесі. Визначено, що mobile-learning пропонує значні переваги, такі як гнучкість, персоналізація та підвищення мотивації студентів, однак його ефективне впровадження потребує врахування специфіки мобільних платформ та потенційних обмежень. Розглянуто різноманітні типи тестових завдань, що підкреслює необхідність комбінованого підходу для всебічної оцінки знань.

Огляд існуючих мобільних застосунків, дозволив виявити як позитивні аспекти, так і суттєві недоліки. Серед останніх особливо виділяються мовний бар'єр, оскільки більшість якісних аналогів є англомовними, обмеженість у типах пропонованих тестових завдань, де часто домінують запитання з одним правильним варіантом відповіді, та не завжди оптимальний користувацький інтерфейс.

Спираючись на проведений аналіз та потреби потенційної аудиторії, було сформульовано детальні вимоги до власного мобільного застосунку. Ці вимоги охоплюють необхідність створення системи аутентифікації, підтримки кількох категорій тестів з різними типами запитань, реалізації таймера, навігації, відображення та збереження результатів, а також розробки інтуїтивно зрозумілого україномовного інтерфейсу.

На завершення першого розділу, спираючись на проведений аналіз та потреби потенційної аудиторії, було здійснено та обґрунтовано вибір ключових інструментальних засобів та технологій для розробки. Вибір платформи Android, мови програмування Kotlin, середовища розробки Android Studio, системи збірки Gradle, UI-фреймворку Jetpack Compose, бібліотеки Room для локальної бази даних та сервісів Firebase для аутентифікації і хмарного зберігання даних, створює технологічний фундамент для реалізації сучасного та ефективного мобільного застосунку. Ці рішення будуть детально застосовані при проєктуванні архітектури та розробці у наступних розділах.

РОЗДІЛ 2
ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

2.2 Проєктування архітектури

В розробці Android-застосунку було дотримано перевірених принципів архітектури. За основу обрано згадану багатошарову архітектуру **MVVM (Model-View-ViewModel)** з унідирективним потоком даних, що забезпечує чіткий поділ відповідальностей, покращену тестованість та високу супроводжуваність коду [26].

У рамках обраної архітектури, шар UI (View) представлений компонентами Jetpack Compose. Ці декларативні Composable-функції відповідають за візуалізацію даних, що надходять від ViewModel, та за передачу ініційованих користувачем подій (наприклад, вибір категорії тесту або відповідь на питання) до відповідної ViewModel для подальшої обробки [26].

Класи ViewModel (AuthViewModel для аутентифікації та QuizViewModel для логіки тестування) виконують роль утримувачів стану. Вони інкапсулюють стан відповідних екранів (наприклад, список категорій, поточне питання, відповіді користувача, стан таймера, статус завантаження) та всю пов'язану бізнес-логіку. ViewModel обробляють події від View, взаємодіють з шаром даних для отримання або збереження інформації, та оновлюють свій стан, який потім реактивно відображається у View. Таке розділення UI та логіки підвищує читабельність коду, спрощує тестування та управління станом застосунку [26].

Шар Model у контексті MVVM представлений джерелами даних та логікою доступу до них. У розроблюваному застосунку він включає:

- Локальне джерело даних (SQLite/Room): Клас AppDatabase разом з Data Access Objects (DAO), такими як CategoryDao та QuizDataDao, відповідає за надання списку категорій, питань та варіантів відповідей з локальної бази даних SQLite.
- Хмарні джерела даних (Firebase): Сервіс FirebaseAuth використовується AuthViewModel для процесів реєстрації, входу та

управління сесіями користувачів. Сервіс FirebaseFirestore застосовується у QuizViewModel для збереження результатів пройдених тестів авторизованими користувачами.

Таким чином, використання MVVM забезпечує чіткий розподіл відповідальності: UI (екрани, такі як LoginScreen, RegistrationScreen, CategorySelectionScreen, QuizScreen, ResultScreen, DetailedResultsScreen) лише відображає стан та передає події; ViewModel утримує та оновлює стан, керуючи логікою; а DAO та сервіси Firebase слугують джерелами даних. Дана архітектура є особливо доцільною для розроблюваного тестового застосунку, оскільки він вимагає асинхронної роботи з базами даних та хмарними сервісами без блокування користувацького інтерфейсу, а також ефективного збереження стану при змінах конфігурації пристрою.

Для ілюстрації взаємодії основних архітектурних компонентів та напрямків потоків даних у ключових сценаріях роботи застосунку розроблено відповідні схеми.

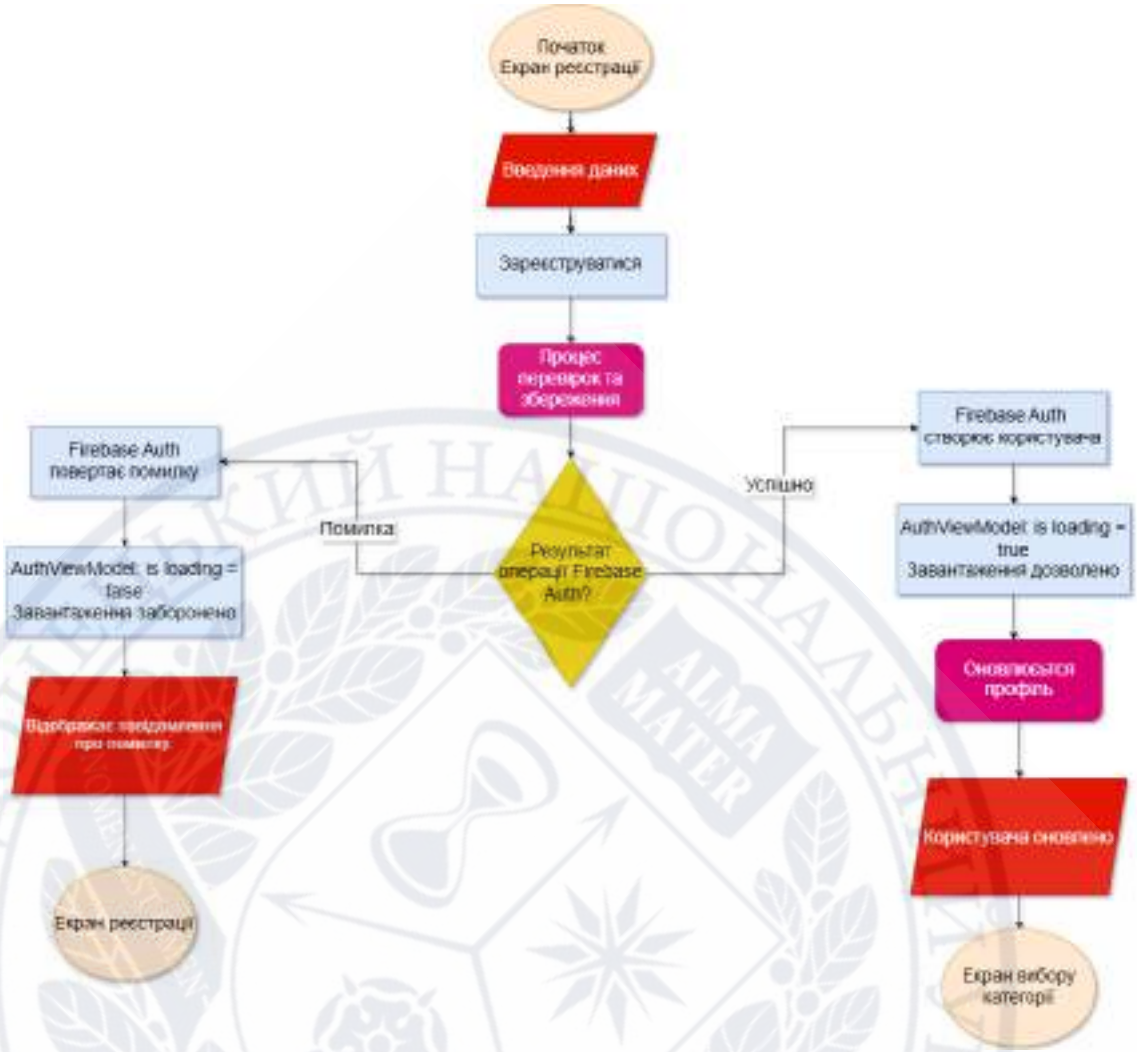

Рисунок 2.1 – Приклад взаємодії компонентів при реєстрації

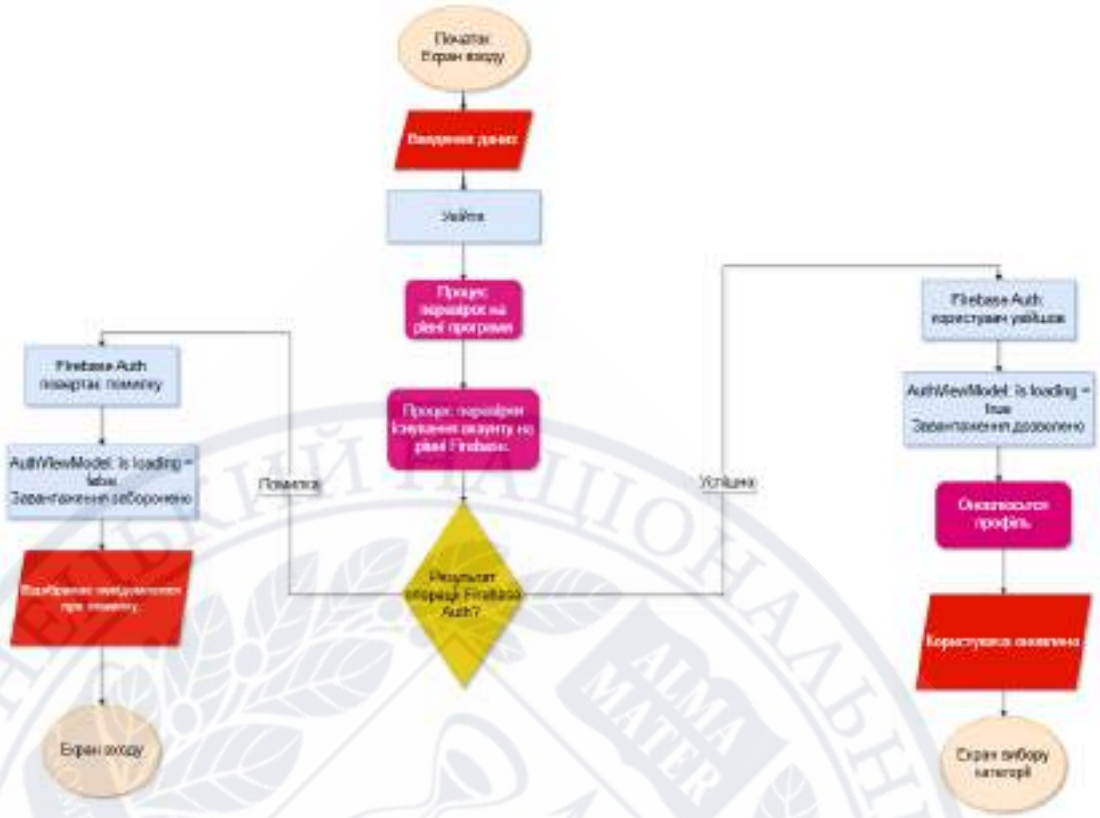

Рисунок 2.2 – Приклад взаємодії компонентів при вході існуючого користувача

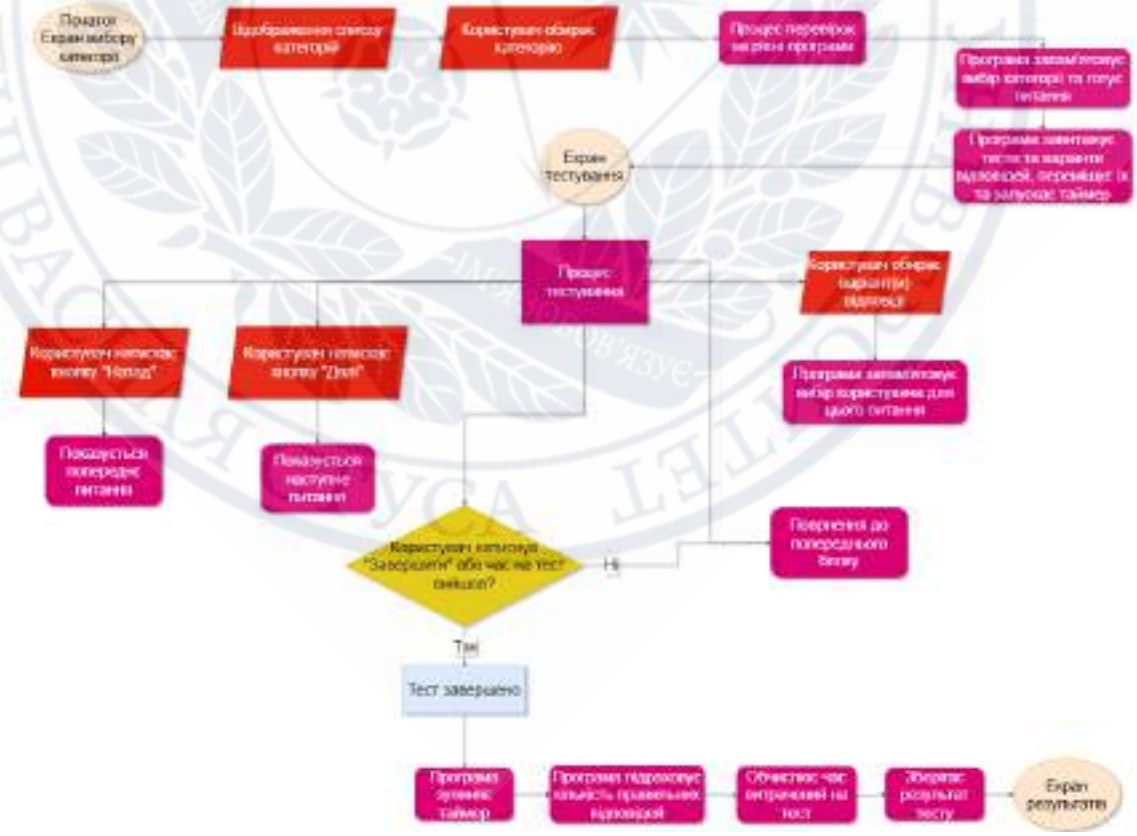

Рисунок 2.3 – Приклад взаємодії компонентів проходженні тесту

2.3 Модульна структура застосунку

Для забезпечення високого рівня організації кодової бази та полегшення подальшої підтримки й розвитку, проєкт структуровано з урахуванням модульного підходу та логічного групування компонентів.

Розподіл за функціональними ознаками. Хоча проєкт реалізовано в рамках одного Android-модуля (:app), його внутрішня структура коду орієнтована на логічне розділення за функціональними блоками. Ключові функціональні області, такі як аутентифікація користувачів та процес тестування, мають свої окремі компоненти (ViewModel, UI-екрани, моделі стану), що сприяє кращій ізоляції логіки та полегшує навігацію по проєкту. У майбутньому, при значному розширенні функціоналу, такий підхід дозволить легко винести ці функціональні блоки в окремі Gradle-модулі.

Організація пакетів. Класи та файли всередині основного модуля організовані за шарами архітектури та функціональним призначенням, що відповідає загальноприйнятим практикам. Основні пакети включають:

Ui: містить підпакети для UI-компонентів:

screens: composable-функції для кожного екрану застосунку (наприклад, LoginScreen.kt, CategorySelectionScreen.kt, QuizScreen.kt).

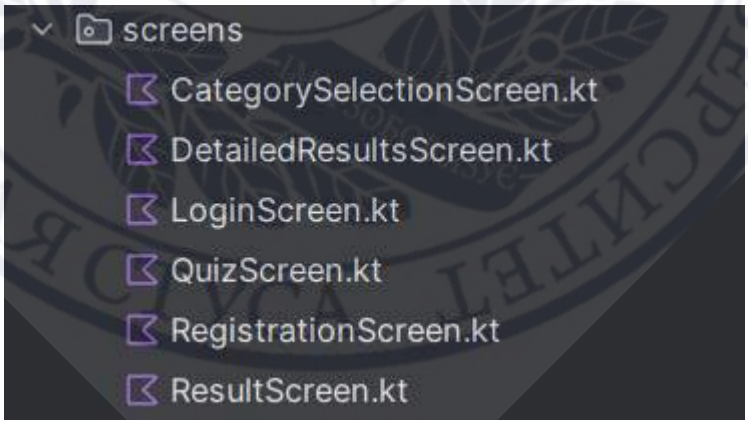

Рисунок 2.4 – Підпакет screens

theme: визначення теми Material Design 3 (Theme.kt, Color.kt, Type.kt, Shape.kt).

navigation: файл AppNavigation.kt для централізованого визначення навігаційних маршрутів.

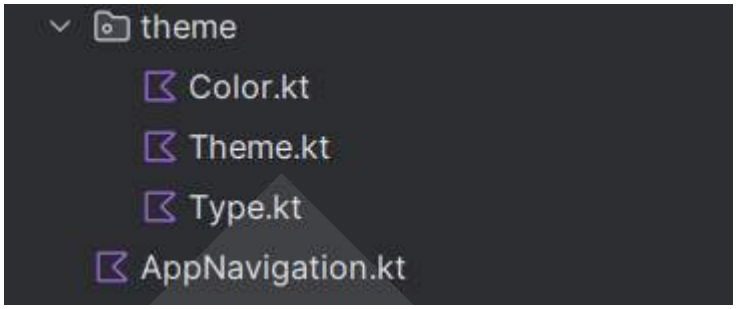

Рисунок 2.5 – Підпакет theme та файл AppNavigation

Окремий пакет viewmodel містить: класи ViewModel (AuthViewModel.kt, AuthUiState.kt, QuizViewModel.kt)

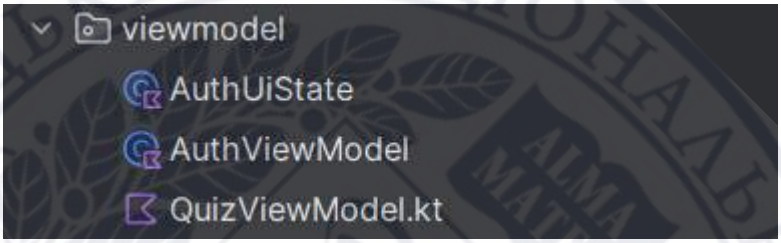

Рисунок 2.6 – Пакет viewmodel

Пакет data: містить компоненти, відповідальні за доступ до даних. А саме database, яка включає підпакети entities (для класів сутностей Room), dao (для Data Access Objects) та клас AppDatabase.kt.

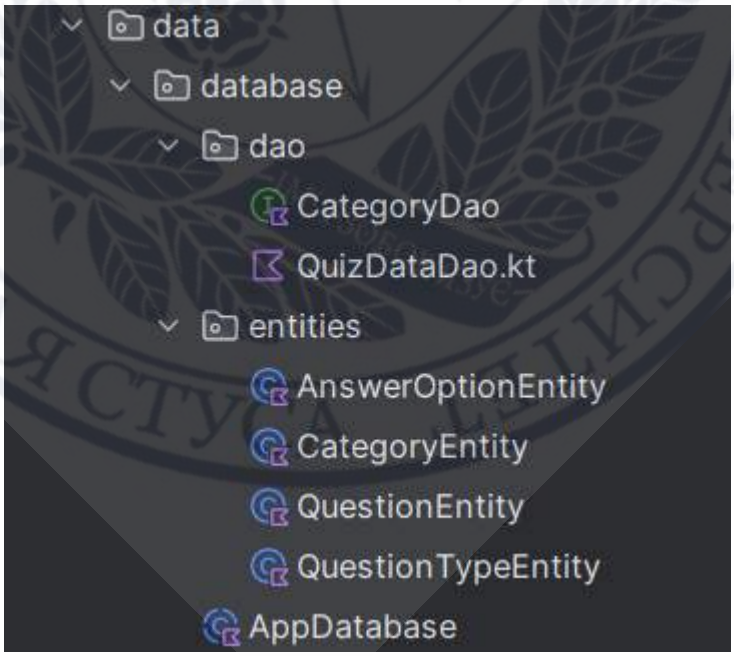

Рисунок 2.7 – Пакет data та його підпакети з класами

Управління залежностями. Для управління зовнішніми бібліотеками та внутрішньомодульними залежностями використовується система збірки Gradle з

конфігураційними файлами, написаними на Kotlin DSL (.gradle.kts). Версії залежностей централізовано управляються через Gradle Version Catalog (файл libs.versions.toml), що підвищує консистентність та спрощує процес оновлення бібліотек.

2.4 Проєктування бази даних

Для локального зберігання основного контенту тестів (категорії, типи питань, самі питання та варіанти відповідей) застосунок використовує бібліотеку Room Persistence Library. Room спрощує роботу з базою даних, забезпечує компіляційну перевірку SQL-запитів [27].

Дані організовуються у сутності (Entities) – класи, позначені @Entity. У нашому застосунку модель містить такі сутності:

- **CategoryEntity**: зберігає інформацію про категорії тестів. Ключові поля: id (INTEGER, первинний ключ, автоінкремент), name (TEXT, назва категорії).
- **QuestionTypeEntity**: зберігає типи питань. Ключові поля: id (INTEGER, первинний ключ, автоінкремент), typeCode (TEXT, унікальний код типу, наприклад,"SINGLE_CHOICE", "MULTIPLE_CHOICE", "TRUE_FALSE").
- **QuestionEntity**: зберігає текст питань та їх зв'язки. Ключові поля: id (INTEGER, первинний ключ, автоінкремент), categoryId (INTEGER, зовнішній ключ до Categories.id), questionTypeId (INTEGER, зовнішній ключ до QuestionTypes.id), text (TEXT).
- **AnswerOptionEntity**: зберігає варіанти відповідей. Ключові поля: id (INTEGER, первинний ключ, автоінкремент), questionId (INTEGER, зовнішній ключ до Questions.id), text (TEXT), isCorrect (INTEGER, 0 або 1).

Між сутностями встановлені зв'язки "один-до-багатьох": одна категорія (CategoryEntity) пов'язана з багатьма питаннями (QuestionEntity); одне питання (QuestionEntity) пов'язане з багатьма варіантами відповідей (AnswerOptionEntity). Ці зв'язки реалізовані за допомогою зовнішніх ключів та

анотацій @ForeignKey в Room, з опцією onDelete = ForeignKey.CASCADE для підтримки цілісності даних [28].

Доступ до бази даних здійснюється через Data Access Objects (DAO):

CategoryDao: надає метод getAllCategories(): Flow<List<CategoryEntity>> для отримання списку всіх категорій

QuizDataDao: надає метод getQuestionsByCategoryId(categoryId: Int): Flow<List<QuestionWithDetails>>. Цей метод використовує анотацію @Transaction та @Relation (всередині допоміжного класу QuestionWithDetails) для ефективного завантаження питання разом з його типом та списком варіантів відповідей.

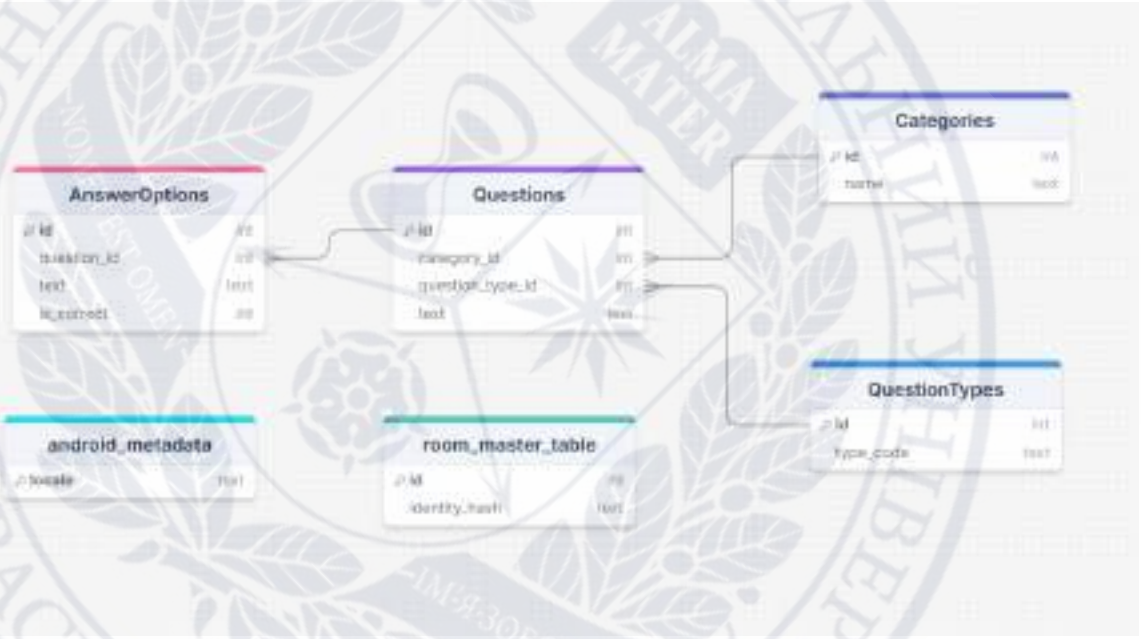

Рисунок 2.8 – діаграма локальної бази даних Sqlite

База даних quiz_database.db попередньо заповнена двома категоріями ("Основи програмування", "Алгоритми та структури даних") та загалом 60 питаннями різних типів з відповідними варіантами. Цей файл розміщено у теці app/src/main/assets/databases/ і копіюється у внутрішнє сховище застосунку при першому запуску за допомогою методу createFromAsset() у конфігурації AppDatabase. Приклади наповнення ключових таблиць наведені на рисунках 2.9-2.12.

Таблиця: Categories

	id	name
	Фі...	Фільтр
1	1	Основи програмування
2	2	Алгоритми та структури даних

Рисунок 2.9 – вміст таблиці Categories

На рисунку 2.9 продемонстровано фрагмент таблиці Categories, яка слугує для зберігання та ідентифікації основних тематичних розділів тестів у застосунку. Кожен запис у цій таблиці представляє окрему категорію і містить унікальний числовий ідентифікатор (id), який використовується як первинний ключ та для зв'язку з таблицею питань, а також текстове поле name, що зберігає повну назву категорії. Така структура дозволяє легко розширювати кількість доступних категорій тестів у майбутньому.

Таблиця: QuestionTypes

	id	type_code
	Фі...	Фільтр
1	1	SINGLE_CHOICE
2	2	MULTIPLE_CHOICE
3	3	TRUE_FALSE

Рисунок 2.10 – вміст таблиці QuestionTypes

Рисунок 2.10 ілюструє вміст таблиці QuestionTypes, яка визначає різні формати запитань, що підтримуються тестовою системою. Кожен рядок таблиці відповідає одному типу питання та включає унікальний числовий ідентифікатор і текстовий код типу (type_code), такий як "SINGLE_CHOICE", "MULTIPLE_CHOICE" або "TRUE_FALSE". Ці коди використовуються програмною логікою для коректного відображення інтерфейсу відповіді та перевірки правильності обраних користувачем варіантів.

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Таблиця: Questions

	id	category_id	question_type_id	text
	Фі...	Фільтр	Фільтр	Фільтр
1	1	1	1	Що таке компіляція в програмуванні?
2	2	1	1	Який оператор використовується для …
3	3	1	1	Що означає абревіатура IDE?
4	4	1	2	Які з наведених є парадигмами …
5	5	1	2	Які ключові слова використовуються …
6	6	1	2	Які типи даних зазвичай відносять до…
7	7	1	3	Чи завжди програма з синтаксичними …
8	8	1	3	HTML - це мова програмування.
9	9	1	3	Git - це система керування базами …
10	10	2	1	Який алгоритм пошуку вимагає, щоб …
11	11	2	1	Яка структура даних найкраще …
12	12	2	1	Яка асимптотична складність алгоритм…

Рисунок 2.11 – вміст таблиці Questions

На рисунку 2.11 наведено приклад записів з таблиці Questions. Ця таблиця є центральною для зберігання тестових завдань і містить текстове формулювання кожного питання, а також зовнішні ключі category_id та question_type_id, які встановлюють зв'язок із відповідними записами в таблицях Categories та QuestionTypes. Кожне питання має унікальний ідентифікатор, що дозволяє однозначно посилатися на нього з інших таблиць, зокрема з AnswerOptions.

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Таблиця: AnswerOptions

	id	question_id	text	is_correct
	Фі...	Фільтр	Фільтр	Фільтр
1	1	1	Процес виконання програми крок за …	0
2	2	1	Процес перетворення вихідного коду н…	1
3	3	1	Процес пошуку помилок у програмі	0
4	4	1	Процес написання документації до коду	0
5	5	2	==	0
6	6	2	:=	0
7	7	2	=	1
8	8	2	===	0
9	9	3	Integrated Development Environment	1
10	10	3	Internal Data Exchange	0

Рисунок 2.12 – вміст таблиці AnswerOptions

Рисунок 2.12 демонструє структуру та приклади даних у таблиці AnswerOptions, яка зберігає всі можливі варіанти відповідей для кожного питання. Кожен запис містить текст варіанту відповіді, зовнішній ключ question_id, що вказує на відповідне питання в таблиці Questions, та булеве поле is_correct (зберігається як INTEGER 0 або 1), яке позначає, чи є даний варіант відповіді правильним. Для питань з одним правильним варіантом або типу "так/ні" лише один запис для конкретного question_id матиме is_correct = 1, тоді як для питань з множинним вибором таких записів може бути декілька.

Для реалізації системи аутентифікації та зберігання результатів проходження тестів авторизованими користувачами використовуються сервіси платформи Firebase.

1) Firebase Authentication забезпечує механізми реєстрації, входу та управління сесіями користувачів [29]. У проєкті реалізовано аутентифікацію за допомогою електронної пошти та пароля. При реєстрації також зберігається ім'я користувача. Кожному зареєстрованому користувачеві Firebase присвоює унікальний ідентифікатор (UID), який надалі використовується для зв'язування результатів тестування з конкретним обліковим записом.

Рисунок 2.13 – приклад даних користувачів

На рисунку 2.13 представлено інтерфейс консолі Firebase Authentication, що ілюструє зберігання облікових записів користувачів. Для кожного користувача система автоматично генерує унікальний ідентифікатор (UID), який використовується для зв'язування з іншими даними, наприклад, результатами тестів. Також відображається електронна пошта, використана при реєстрації, та може зберігатися ім'я користувача.

2) Для зберігання результатів проходження тестів авторизованими користувачами використовується документо-орієнтована NoSQL база даних Firebase Cloud Firestore [30]. Це забезпечує централізоване зберігання даних, доступність з різних пристроїв (потенційно) та можливості масштабування.

Спроектовано наступну структуру колекції:

Колекція user_results: кожен документ у цій колекції представляє результат одного пройденого тесту одним користувачем і має автоматично згенерований Firestore ID.

Таблиця 2.1 Структура документа в колекції user_results

Назва поля	Тип даних у Firestore	Опис/Приклад
userId	String	UID користувача (наприклад, "O0a2Oz..")
username	String	Ім'я користувача (наприклад, "Volodymyr275")
categoryId	Number (Integer)	ID категорії тесту (наприклад, 1)
categoryName	String	Назва категорії (наприклад, "Основи програмування")
score	Number (Integer)	Кількість правильних відповідей (наприклад, 15)
totalQuestions	Number (Integer)	Загальна кількість питань (наприклад, 30)
timeSpent	String	Витрачений час, формат "MM:SS" (наприклад, "12:35")
timestamp	Timestamp	Дата та час проходження тесту

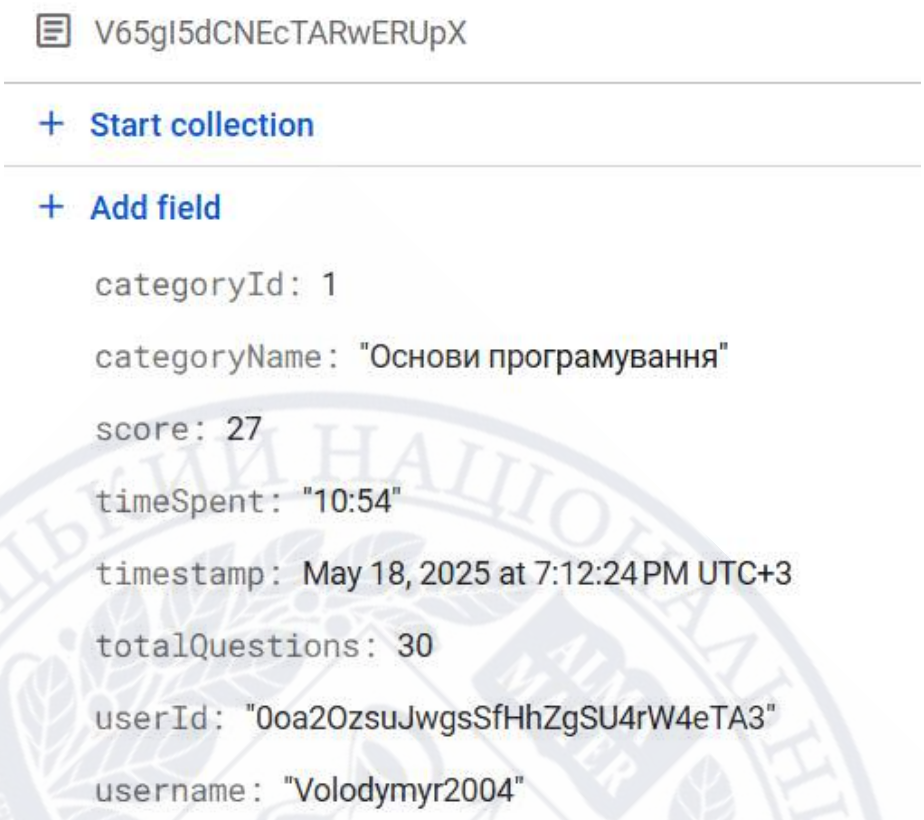

Рисунок 2.14 – приклад структури документа з колекції user_results

Таким чином, поєднання локальної бази Room і хмарного Firestore дозволяє забезпечити швидкий доступ до питань/відповідей та безпечне зберігання результатів і даних користувачів. Local DB відповідає за швидку роботу в офлайні і перевірку знань, тоді як Firestore гарантує централізоване сховище даних і можливість масштабування.

Висновок до другого розділу

У другому розділі кваліфікаційної роботи було детально розглянуто етап проєктування мобільного застосунку для тестування знань з комп'ютерних наук. Ключова увага була приділена розробці надійної архітектури, проєктуванню модульної структури коду та визначенню структури локальної та хмарної баз даних, що є фундаментом для подальшої успішної реалізації програмного продукту.

Для побудови застосунку було спроєктовано багатошарову архітектуру на основі патерну MVVM. Такий підхід забезпечує чіткий поділ відповідальності між компонентами UI, логікою презентації та управлінням станом, та шаром даних, який включає локальну базу даних SQLite/Room та хмарні сервіси Firebase. Було деталізовано взаємодію цих компонентів у ключових сценаріях роботи застосунку. Для покращення організації кодової бази застосовано підхід до структурування проєкту за пакетами, що відповідають функціональним областям та шарам архітектури, а управління залежностями централізовано за допомогою Gradle Version Catalog.

Проєктування баз даних охопило як локальне, так і хмарне сховища. Для локального зберігання основного контенту тестів (категорії, типи питань, самі питання та варіанти відповідей) було обрано бібліотеку Room Persistence Library. Детально спроєктовано структуру сутностей (CategoryEntity, QuestionTypeEntity, QuestionEntity, AnswerOptionEntity) та об'єктів доступу до даних (DAO), що забезпечує ефективне завантаження питань та їх компонентів. Для зберігання результатів проходження тестів авторизованими користувачами та їхніх облікових даних було спроєктовано структуру колекції user_results у документо-орієнтованій NoSQL базі даних Firebase Cloud Firestore, що забезпечить централізоване зберігання, доступність з різних пристроїв та можливість масштабування.

Таким чином, у другому розділі було закладено міцне теоретичне та проєктне підґрунтя для створення функціонального та надійного мобільного застосунку. Визначені архітектурні рішення, організаційна структура коду та спроєктовані структури даних створюють чіткий план та основу для успішної реалізації та подальшого тестування мобільного застосунку.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Розробка модуля аутентифікації користувачів

Фундаментальним компонентом розробленого мобільного застосунку є модуль аутентифікації користувачів. Його основне призначення полягає в ідентифікації користувачів, що є передумовою для персоналізації взаємодії з системою та забезпечення можливості збереження індивідуальних результатів тестування. Для реалізації даного модуля було обрано хмарний сервіс Firebase Authentication, що надається компанією Google, завдяки його надійності, простоті інтеграції та широкому набору інструментів для управління обліковими записами. У рамках даного проєкту було імплементовано механізм аутентифікації за допомогою комбінації електронної пошти та пароля, а також передбачено можливість введення користувачем унікального імені (нікнейма) під час процедури реєстрації.

Користувацький інтерфейс для процесів аутентифікації було розроблено з використанням декларативного підходу Jetpack Compose та компонентів системи дизайну Material Design 3. Створено два ключові екрани: LoginScreen, призначений для входу існуючих користувачів, та RegistrationScreen, що забезпечує створення нових облікових записів. Екран входу містить текстові поля для введення електронної пошти та пароля, кнопку для ініціювання входу та текстову кнопку для переходу на екран реєстрації. Аналогічно, екран реєстрації надає поля для введення email, імені користувача, пароля та його підтвердження, а також кнопку для завершення реєстрації та посилання для переходу на екран входу. Для полів введення пароля на обох екранах реалізовано функцію перемикання видимості символів за допомогою іконки "ока", що покращує користувацький досвід. Валідація введених даних, така як перевірка заповненості полів, коректність формату email (з використанням стандартних засобів Android android.util.Patterns.EMAIL_ADDRESS), відповідність довжини імені користувача встановленим обмеженням (від 3 до 20 символів) та пароля

(мінімум 6 символів), а також співпадіння паролів при реєстрації, здійснюється на стороні клієнта перед відправкою запиту до Firebase. У випадку виявлення помилок валідації або отримання помилки від сервісу Firebase, користувачеві відображаються відповідні інформаційні повідомлення. Під час виконання асинхронних операцій аутентифікації на екранах відображається індикатор завантаження.

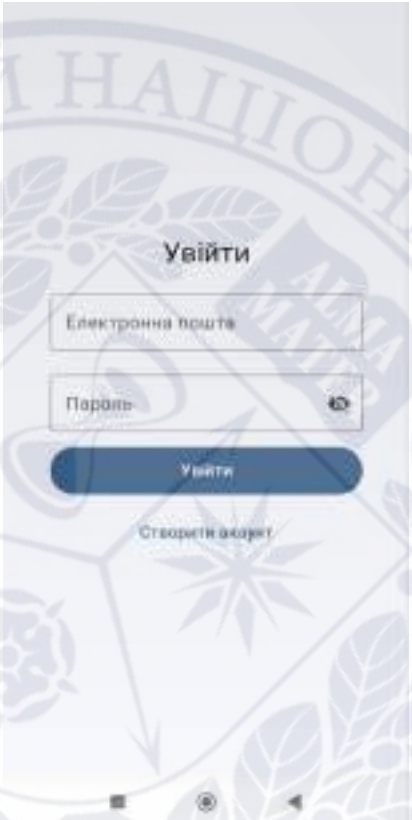

Рисунок 3.1 – Екран входу

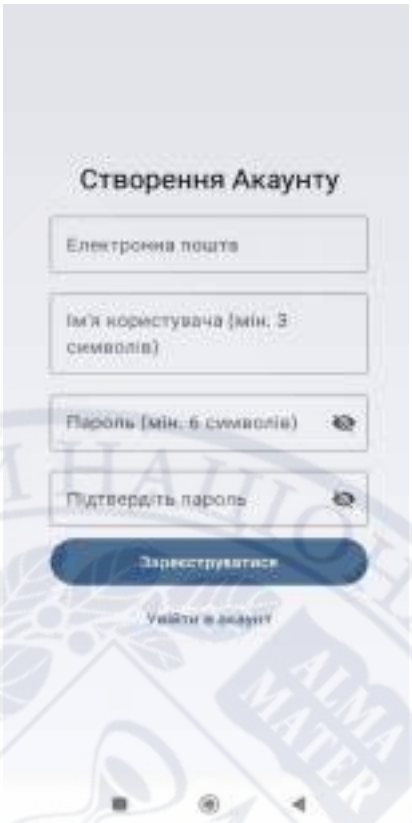

Рисунок 3.2 – Екран реєстрації

Для управління процесами входу та реєстрації було створено спеціальний компонент AuthViewModel. Цей компонент відповідає за всю логіку, пов'язану з акаунтами користувачів. Він зберігає актуальну інформацію про те, чи йде зараз процес завантаження, чи увійшов користувач до системи, а також будь-які повідомлення про помилки, що можуть виникнути. Для взаємодії з сервісом аутентифікації Firebase (який перевіряє правильність логінів та паролів і створює нові акаунти) AuthViewModel використовує стандартні інструменти, надані Firebase.

Коли користувач намагається зареєструватися, AuthViewModel надсилає запит до Firebase для створення нового акаунту з введеними даними (email, пароль та ім'я користувача). Якщо реєстрація успішна, ім'я користувача зберігається у його профілі. При спробі входу, AuthViewModel перевіряє введені email та пароль через Firebase. У разі виникнення помилок, таких як вже існуючий email при реєстрації або неправильний пароль при вході, AuthViewModel готує зрозуміле повідомлення для відображення користувачеві.

AuthViewModel також автоматично відстежує, чи увійшов користувач до системи чи вийшов з неї, і відповідно оновлює стан застосунку. Це забезпечує, наприклад, що користувач бачить своє ім'я після входу або перенаправляється на екран входу після виходу з акаунту.

Приклад обробки результату реєстрації у AuthViewModel:

```
fun registerUser(email: String, password: String, username: String) {
    _authUiState.update { it.copy(isLoading = true, errorMessage = null, registrationSuccess = false) }
    auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val firebaseUser = auth.currentUser

                if (firebaseUser != null && username.isNotBlank()) {
                    val profileUpdates = UserProfileChangeRequest.Builder()
                        .setDisplayName(username.trim())
                        .build()

                    firebaseUser.updateProfile(profileUpdates)
                        .addOnCompleteListener { profileTask ->
                            _authUiState.update { it.copy(isLoading = false, registrationSuccess = true, currentUser = auth.currentUser) }
                        }
                } else {
                    _authUiState.update { it.copy(isLoading = false, registrationSuccess = true, currentUser = firebaseUser) }
                }
            } else {
                val exception = task.exception
                val customErrorMessage = when (exception) {
                    is FirebaseAuthUserCollisionException -> "Користувач з таким email вже зареєстрований."
```

```
                    else -> exception?.localizedMessage ?:
"Помилка реєстрації. Спробуйте пізніше."
                }
                _authUiState.update { it.copy(isLoading = false,
errorMessage = customErrorMessage) }
            }
        }
}
```

Навігація між екранами входу, реєстрації та основним функціоналом застосунку реалізована за допомогою компонента Navigation Compose. У MainActivity визначено NavHost, де стартовий екран динамічно визначається на основі поточного стану авторизації користувача. Успішний вхід або реєстрація призводять до переходу на екран вибору категорії з очищенням стеку навігації від попередніх екранів аутентифікації. Аналогічно, вихід з акаунту повертає користувача на екран входу. Маршрути, що вимагають авторизації, захищені перевіркою стану currentUser, і у випадку неавторизованого доступу відбувається автоматичне перенаправлення на екран входу.

Таким чином, розроблений модуль аутентифікації забезпечує необхідний рівень безпеки та персоналізації, створюючи основу для подальшої взаємодії користувача з основними функціями тестувального застосунку.

3.2 Розробка модуля тестування знань

Центральним функціональним блоком мобільного застосунку є модуль тестування знань. Його реалізація охоплює взаємодію користувача з інтерфейсом для вибору категорії та проходження тесту, а також внутрішню логіку управління даними та станом тесту. Основою для цього модуля слугує QuizViewModel, який координує отримання тестових завдань з локальної бази даних та керує усім процесом тестування.

Розробка користувацького інтерфейсу даного модуля здійснена за допомогою Jetpack Compose. Першим екраном, з яким взаємодіє користувач після аутентифікації, є CategorySelectionScreen.

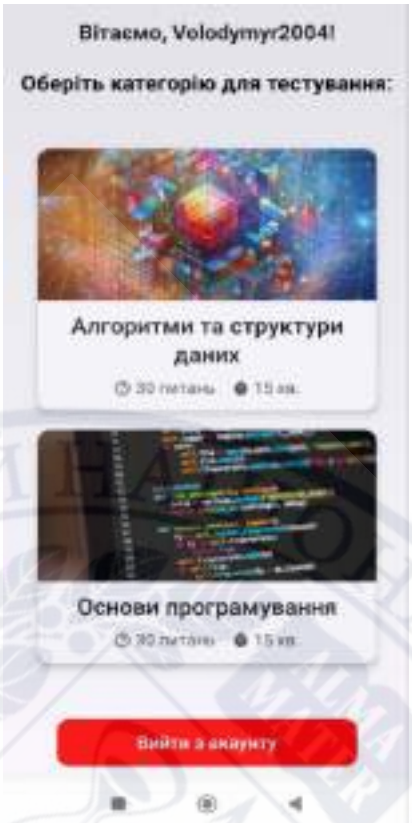

Рисунок 3.3 – Екран вибору категорії

На цьому екрані відображається список доступних категорій тестів, завантажених з локальної бази даних SQLite. Кожна категорія представлена візуально привабливою карткою з тематичним зображенням, назвою, інформацією про кількість питань та часом тестування, що полегшує користувачеві вибір. При натисканні на картку категорії ініціюється процес завантаження відповідних питань.

Після вибору категорії та успішного завантаження тестових завдань, користувач переходить на екран QuizScreen. Цей екран є основним інтерфейсом для безпосереднього проходження тесту. На ньому відображається текст поточного питання, варіанти відповідей, номер поточного питання та загальний таймер, що відраховує час, відведений на тест. Важливою особливістю є підтримка трьох різних типів запитань, що забезпечує варіативність та глибину перевірки знань:

Питання з одним правильним варіантом: користувачеві пропонується обрати один варіант з кількох за допомогою елементів, що імітують радіокнопки.

Кожен варіант відповіді представлений Composable-функцією SingleChoiceOptionRow, яка містить радіокнопку, буквене позначення варіанту (A, B, C, D), виділене жирним чорним шрифтом, та текст самого варіанту. Весь рядок є клікабельним для зручності вибору.

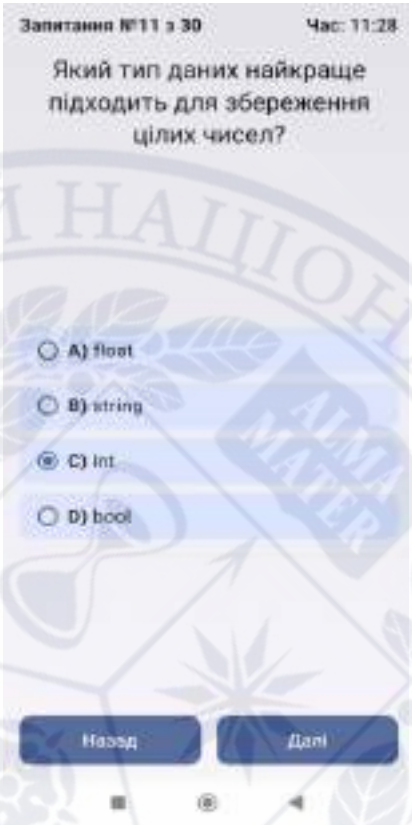

Рисунок 3.4 – Питання з одиничним вибором

Лістинг Composable-функції SingleChoiceOptionRow:

```
@Composable
fun SingleChoiceOptionRow(
    optionLetter: String,
    text: String,
    isSelected: Boolean,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Row(
        modifier = modifier
            .fillMaxWidth()
            .background(

MaterialTheme.colorScheme.primaryContainer.copy(alpha = 1f),
                shape = RoundedCornerShape(12.dp)
            )
            .selectable(
                selected = isSelected,
```

```
                onClick = onClick,
                role = Role.RadioButton
            )
            .padding(horizontal = 16.dp, vertical = 12.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        RadioButton(
            selected = isSelected,
            onClick = null
        )
        Spacer(modifier = Modifier.width(8.dp))
        Text(text = buildAnnotatedString {
            withStyle(style = SpanStyle(fontWeight =
FontWeight.Bold, color = Color.Black)) {
                append("$optionLetter) ")
            }
            append(text)
        },
            style = MaterialTheme.typography.bodyLarge
        )
    }
}
```

У даному фрагменті selectable модифікатор робить весь рядок інтерактивним, а RadioButton візуально відображає стан вибору. buildAnnotatedString використовується для стилізації буквеного позначення варіанту.

Питання з декількома правильними варіантами: користувач може обрати декілька варіантів відповідей за допомогою чекбоксів. Кожен варіант представлений Composable-функцією MultipleChoiceOptionRow, яка містить чекбокс, буквене позначення та текст варіанту. Весь рядок також є клікабельним.

Рисунок 3.5 – Питання з множинним вибором

Лістинг Composable-функції MultipleChoiceOptionRow:

```
@Composable
fun MultipleChoiceOptionRow(
    optionLetter: String,
    text: String,
    isSelected: Boolean,
    onCheckedChange: () -> Unit,
    modifier: Modifier = Modifier
) {
    Row(
        modifier = modifier
            .fillMaxWidth()
            .background(

MaterialTheme.colorScheme.primaryContainer.copy(alpha = 1f),
                shape = RoundedCornerShape(12.dp)
            )
            .selectable(
                selected = isSelected,
                onClick = onCheckedChange,
                role = Role.Checkbox
            )
            .padding(horizontal = 16.dp, vertical = 12.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Checkbox(
```

```
            checked = isSelected,
            onCheckedChange = null
        )
        Spacer(modifier = Modifier.width(8.dp))
        Text(

            text = buildAnnotatedString {
                withStyle(style = SpanStyle(fontWeight =
FontWeight.Bold, color = Color.Black)) {
                    append("$optionLetter) ")
                }
                append(text)
            },
            style = MaterialTheme.typography.bodyLarge
        )
    }
}
```

Логіка вибору та зняття вибору для кожного чекбоксу обробляється у QuizViewModel при виклику onCheckedChange.

Питання типу "так/ні": відповідь надається шляхом вибору однієї з двох відповідних кнопок. Це реалізовано у Composable-функції TrueFalseButtonOption. Для візуального розрізнення та кращого користувацького досвіду, кнопка "Так" має зелений фон, а кнопка "Ні" – червоний, коли вони не обрані. При виборі однієї з кнопок її фон змінюється на відповідний темніший відтінок, а текст на кнопці стає білим для кращого контрасту.

Рисунок 3.6 – Питання так/ні

Лістинг Composable-функції TrueFalseButtonOption:

```
@Composable
fun TrueFalseButtonOption(
    text: String,
    isSelected: Boolean,
    onClick: () -> Unit,
    buttonColor: Color,
    modifier: Modifier = Modifier
) {

    if (isSelected) {
        Button(
            onClick = onClick,
            modifier = modifier
                .fillMaxWidth()
                .defaultMinSize(minHeight = 56.dp),
            colors = ButtonDefaults.buttonColors(
                containerColor = buttonColor,
                contentColor = Color.White
            ),
            shape = MaterialTheme.shapes.medium
        ) {
            Text(text = text, fontSize = 18.sp, textAlign =
TextAlign.Center)
        }
    } else {
```

```
            OutlinedButton(
                onClick = onClick,
                modifier = modifier
                    .fillMaxWidth()
                    .defaultMinSize(minHeight = 56.dp),
                colors = ButtonDefaults.outlinedButtonColors(
                    contentColor = buttonColor
                ),
                border = BorderStroke(2.dp, buttonColor),
                shape = MaterialTheme.shapes.medium
            ) {
                Text(text = text, fontSize = 18.sp, textAlign =
    TextAlign.Center)
            }
        }
    }
```

Питання в межах тесту подаються у випадковому порядку, що забезпечується перемішуванням списку питань у QuizViewModel перед початком кожного нового тесту.

Обробка відповідей користувача та підрахунок балів відбувається у QuizViewModel всередині функції finishQuiz(), яка викликається при завершенні тесту. QuizViewModel зберігає відповіді користувача у userAnswers: MutableMap<Int, List<Int>>, де ключ – це id питання, а значення – список індексів обраних користувачем варіантів (для типів "SINGLE_CHOICE" та "TRUE_FALSE" цей список міститиме один елемент, а для "MULTIPLE_CHOICE" – декілька). Модель QuestionUI містить поле correctOptionIndices: List<Int>, яке зберігає індекси всіх правильних варіантів для даного питання.

Логіка підрахунку балів реалізована наступним чином:

```
var finalScore = 0
        val currentQuestions = _uiState.value.questions
        val currentUserAnswers = _uiState.value.userAnswers

        currentQuestions.forEach { questionUi ->
            val selectedIndices =
currentUserAnswers[questionUi.id] ?: emptyList()
            val correctIndices = questionUi.correctOptionIndices

            if (selectedIndices.isNotEmpty()) {
                when (questionUi.typeCode) {
                    "SINGLE_CHOICE", "TRUE_FALSE" -> {
                        if (selectedIndices.size == 1 &&
correctIndices.contains(selectedIndices.first())) {
```

```
                        finalScore++
                    }
                }
                "MULTIPLE_CHOICE" -> {
                    if (selectedIndices.toSet() ==
correctIndices.toSet()) {
                        finalScore++
                    }
                }
            }
        }
    }
```

Такий підхід до обробки відповідей дозволяє коректно оцінювати знання користувача для кожного з реалізованих типів запитань, забезпечуючи точність підсумкового результату тесту.

Отже, керування логікою тестування повністю інкапсульовано у QuizViewModel. Цей компонент є дуже важливим та відповідає за:

- Завантаження питань та варіантів відповідей для обраної категорії з локальної бази даних SQLite за допомогою QuizDataDao та їх перетворення у UI-сумісні моделі.
- Управління поточним станом тесту, включаючи поточне питання, обрані користувачем відповіді, підрахунок правильних відповідей та загальний прогрес.
- Реалізацію загального таймера тесту. При старті тесту фіксується початковий час і корутина в viewModelScope періодично оновлює час, що залишився. Якщо час вичерпується, тест автоматично завершується.
- Обробку навігації між питаннями ("Вперед", "Назад") та завершення тесту при натисканні відповідної кнопки або по закінченню часу.

3.3 Розробка модуля відображення та збереження результатів

Завершальним етапом взаємодії користувача з тестувальною системою є отримання та аналіз результатів пройденого тесту, а також збереження цих даних для подальшого використання або відстеження прогресу. У мобільному застосунку цей функціонал реалізовано через екран результатів (ResultScreen),

екран детального аналізу відповідей (DetailedResultsScreen) та інтеграцію з хмарним сховищем Firebase Firestore.

Після завершення тесту, ініційованого дією користувача або закінченням відведеного часу, QuizViewModel виконує фінальні обчислення. Припиняється робота загального таймера, і на основі зібраних відповідей користувача та інформації про правильні варіанти для кожного питання розраховується підсумковий бал. Одночасно обчислюється загальний час, витрачений користувачем на проходження тесту, на основі різниці між часом старту тесту та часом його завершення. Отримані дані – кількість правильних відповідей, загальна кількість питань та форматований рядок витраченого часу – передаються для відображення на екран ResultScreen.

Екран ResultScreen, реалізований за допомогою Jetpack Compose, надає користувачеві стислий огляд його успішності. Центральними елементами є чітко відображений підсумковий бал та загальний витрачений час. Для подальшої взаємодії користувачеві пропонуються три кнопки: "Детальні результати тесту", "Спробувати ще" та "Вихід". Кнопка "Спробувати ще" ініціює скидання поточного стану тесту у QuizViewModel та повертає користувача на екран вибору категорії для нового проходження. Кнопка "Вихід" забезпечує закриття застосунку.

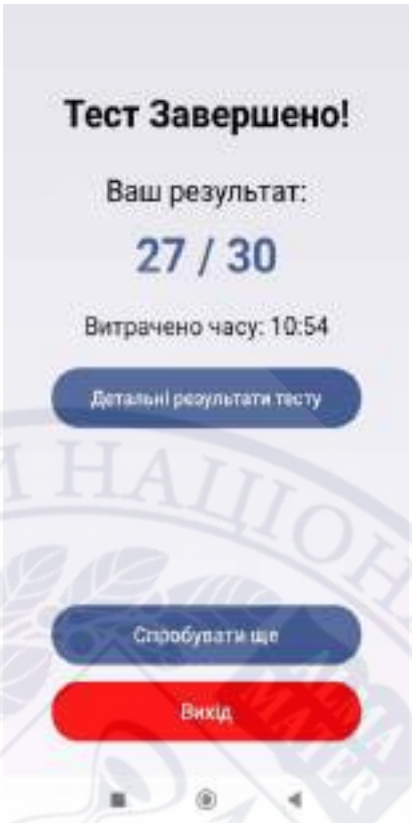

Рисунок 3.7 – Результат тесту

Код Composable-функції ResultScreen для відображення результату успішності тестування:

```
@Composable
fun ResultScreen(
    score: Int,
    totalQuestions: Int,
    timeSpentFormatted: String,
    onRetryClick: () -> Unit,
    onExitClick: () -> Unit,
    onDetailedResultsClick: () -> Unit
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .systemBarsPadding()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        // Верхня частина з результатами та кнопкою "Детальні
результати"
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .weight(1f),
```

```
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            Text(
                text = "Тест Завершено!",
                style = MaterialTheme.typography.displaySmall,
                fontWeight = FontWeight.Bold,
                modifier = Modifier.padding(bottom = 32.dp)
            )

            Text(
                text = "Ваш результат:",
                style = MaterialTheme.typography.headlineMedium,
                modifier = Modifier.padding(bottom = 16.dp)
            )

            Text(
                text = "$score / $totalQuestions",
                style = MaterialTheme.typography.displayMedium,
                fontWeight = FontWeight.Bold,
                color = MaterialTheme.colorScheme.primary,
                modifier = Modifier.padding(bottom = 24.dp)
            )

            Text(
                text = "Витрачено часу: $timeSpentFormatted",
                style = MaterialTheme.typography.titleLarge,
                textAlign = TextAlign.Center,
                modifier = Modifier.padding(bottom = 24.dp)
            )

            // Кнопка "Детальні результати тесту"
            Button(
                onClick = onDetailedResultsClick,
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 32.dp)
                    .height(56.dp),
            ) {
                Text("Детальні результати тесту", fontSize =
18.sp)
            }
        }

        // Нижня частина з кнопками "Спробувати ще" та "Вихід"
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 16.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.spacedBy(16.dp)
        ) {
            Button(
```

```
                onClick = onRetryClick,
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 32.dp)
                    .height(56.dp)
            ) {
                Text("Спробувати ще", fontSize = 18.sp)
            }

            // Кнопка "Вихід"
            Button(
                onClick = onExitClick,
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 32.dp)
                    .height(56.dp),
                colors = ButtonDefaults.buttonColors(
                    containerColor = Color.Red,
                    contentColor = Color.White
                )
            ) {
                Text("Вихід", fontSize = 18.sp)
            }
        }
    }
}
```

Натискання кнопки "Детальні результати тесту" здійснює навігацію на екран DetailedResultsScreen. Цей екран призначений для поглибленого аналізу користувачем своїх відповідей. Для кожного питання з пройденого тесту відображається його текст, усі запропоновані варіанти відповідей, а також візуальні позначки, що вказують, який варіант (або варіанти) обрав користувач, і який варіант (або варіанти) був правильним. Таке представлення дозволяє користувачеві не тільки побачити свої помилки, але й зрозуміти, в яких саме аспектах теми його знання потребують покращення. Кожен елемент списку містить інформацію по одному питанню, з візуальним виділенням правильних відповідей, неправильних відповідей користувача та правильних відповідей, які користувач не обрав. Для забезпечення чіткого візуального розмежування та інтуїтивного розуміння правильності відповідей, кожен варіант відповіді у списку має специфічне оформлення. Це реалізовано у Composable-функції QuestionResultItem.

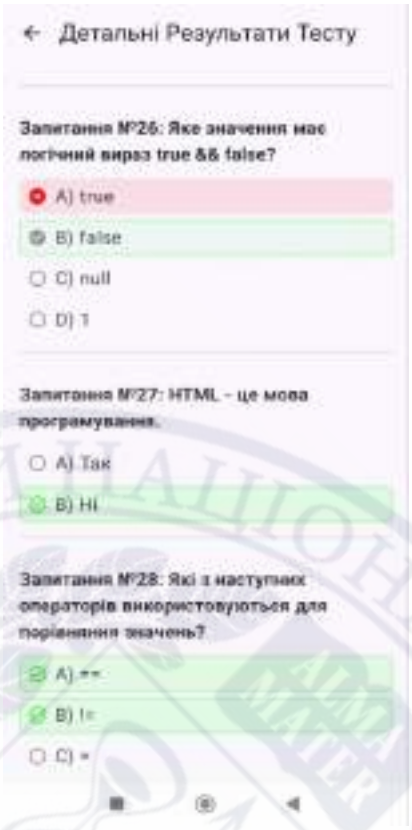

Рисунок 3.8 – Детальні результати

Код Composable-функції QuestionResultItem для відображення детального результату кожного окремого питання:

```
@Composable
fun QuestionResultItem(
    questionNumber: Int,
    question: QuestionUI,
    userSelectedIndices: List<Int>
) {
    Column {
        Text(
            text = "Запитання №$questionNumber: ${question.text}",
            style = MaterialTheme.typography.titleMedium,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(bottom = 12.dp)
        )

        question.options.forEachIndexed { optionIndex, option ->
            val isCorrectOption =
question.correctOptionIndices.contains(optionIndex)
            val wasSelectedByUser =
userSelectedIndices.contains(optionIndex)

            var rowColor = Color.Transparent
            var icon: ImageVector? = null
            var iconColor = MaterialTheme.colorScheme.onSurface
```

```
            when {
                // Користувач обрав правильний варіант
                wasSelectedByUser && isCorrectOption -> {
                    rowColor = Color.Green.copy(alpha = 0.1f)
                    icon = Icons.Filled.TaskAlt
                    iconColor = Color.Green
                }
                // Користувач обрав неправильний варіант
                wasSelectedByUser && !isCorrectOption -> {
                    rowColor = Color.Red.copy(alpha = 0.1f)
                    icon = Icons.Filled.Cancel
                    iconColor = Color.Red
                }
                // Користувач не обрав, але це був правильний
варіант
                !wasSelectedByUser && isCorrectOption -> {
                    rowColor = Color.Green.copy(alpha = 0.05f)
                    icon = Icons.Filled.CheckCircle
                    iconColor = Color.Gray
                }
                // Не обраний і не правильний
                else -> {
                    icon = Icons.Filled.RadioButtonUnchecked
                    iconColor =
MaterialTheme.colorScheme.onSurface.copy(alpha = 0.6f)
                }
            }
```

Важливою складовою функціоналу після завершення тесту є збереження результату користувача. Для цього, як було спроєктовано в підрозділі 2.4, використовується хмарна база даних Firebase Firestore. Імплементація даного механізму зосереджена у QuizViewModel. Після фінального підрахунку балів та визначення витраченого часу, за умови що користувач авторизований, ViewModel формує об'єкт даних, що містить всю необхідну інформацію про результат тесту. Цей об'єкт включає унікальний ідентифікатор користувача (UID), його ім'я, ідентифікатор та назву пройденої категорії, набраний бал, загальну кількість питань, витрачений час у форматі "MM:SS" та мітку часу завершення тесту.

Сформований об'єкт даних асинхронно надсилається на збереження до колекції user_results у базі даних Firestore. Для кожного результату створюється новий документ з автоматично згенерованим Firestore ID. Це забезпечує централізоване та надійне зберігання прогресу користувачів.

Приклад запису результату тесту до Firebase Firestore у QuizViewModel:

```
val resultData = hashMapOf(
        "userId" to currentUser.uid,
        "username" to username,
        "categoryId" to currentSelectedCategoryId,
        "categoryName" to categoryName,
        "score" to finalScore,
        "totalQuestions" to currentQuestions.size,
        "timeSpent" to finalTimeSpentFormatted,
        "timestamp" to Timestamp.now()
    )

    firestore.collection("user_results")
        .add(resultData)
```

3.4 Тестування розробленого програмного продукту

Важливим етапом життєвого циклу розробки будь-якого програмного забезпечення є його всебічне тестування, спрямоване на виявлення помилок, перевірку відповідності функціональним вимогам та оцінку загальної якості продукту. Було проведено функціональне тестування мобільного застосунку, ключових сценаріїв його використання, що охоплюють весь реалізований функціонал.

Методика тестування базувалася на ручному проходженні основних користувацьких сценаріїв на реальних Android-пристроях та емуляторі для перевірки коректності роботи всіх модулів застосунку. Основна увага приділялася перевірці логіки роботи системи аутентифікації, процесу тестування для різних категорій та типів питань, коректності підрахунку результатів, роботи таймера, а також взаємодії з хмарним сховищем Firebase Firestore.

Коментар до таблиці 3.1. Тестування сценарію реєстрації нового користувача з валідними даними продемонструвало коректну роботу системи створення облікових записів. Усі етапи, від введення даних до переходу на екран вибору категорії та перевірки створення запису в Firebase Authentication, пройшли успішно. Це підтверджує працездатність основного потоку залучення нових користувачів.

Основні сценарії функціонального тестування:

Таблиця 3.1 Тест реєстрації з коректними даними

Дія користувача	**Очікуваний результат**	**Фактичний результат**
Відкрити застосунок. Натиснути "Створити акаунт" на екрані входу	Відкривається екран реєстрації	Відповідає
Ввести коректний унікальний Email, ім'я користувача (3-20 симв.), пароль (мін. 6 симв.) та підтвердження пароля	Поля заповнюються, помилки валідації відсутні	Відповідає
Натиснути кнопку "Зареєструватися"	Після успішної реєстрації відбувається перехід на екран вибору категорії	Відповідає
Перевірити консоль Firebase Authentication	Новий користувач з вказаним email створений	Відповідає
На екрані вибору категорії	Відображається привітання та картки категорій	Відповідає

Таблиця 3.2 Тест реєстрації з некоректними даними

Дія користувача	**Очікуваний результат**	**Фактичний результат**
Відкрити застосунок. Натиснути "Створити акаунт" на екрані входу	Відкривається екран реєстрації	Відповідає
Ввести наприклад існуючий email та пароль	Поля заповнюються	Відповідає
Натиснути кнопку "Зареєструватися"	Відображається повідомлення про помилку ("Користувач з таким email вже зареєстрований "). Користувач залишається на екрані реєстрації	Відповідає
Перевірити консоль Firebase Authentication	Нова сесія для користувача не створена	Відповідає

Коментар до таблиці 3.2. Даний тест перевіряв обробку некоректних даних під час реєстрації, зокрема використання вже існуючої електронної пошти. Система коректно ідентифікувала конфлікт та відобразила відповідне повідомлення про помилку, запобігши створенню дубліката акаунту. Це свідчить про належну реалізацію валідації на стороні Firebase та обробки помилок у застосунку.

Таблиця 3.3 Тест входу існуючого користувача

Дія користувача	**Очікуваний результат**	**Фактичний результат**
Запустити застосунок та ввести дані входу(email та пароль)	Текст вводиться у поле	Відповідає
Натиснути кнопку "Увійти"	Після успішного входу відбувається перехід на екран вибору категорії	Відповідає
Перевірити консоль Firebase Authentication	Користувач існує в системі	Відповідає
На екрані вибору категорії	Відображається привітання та картки категорій	Відповідає

Коментар до таблиці 3.3. Тестування сценарію входу для існуючого користувача підтвердило коректну роботу механізму аутентифікації. Застосунок успішно перевіряє облікові дані через Firebase Authentication та надає доступ до основного функціоналу після успішної ідентифікації. Відображення персоналізованого привітання також працює згідно з очікуваннями.

Коментар до таблиці 3.4. Основний функціонал тестування, включаючи завантаження питань з локальної бази даних, обробку різних типів відповідей, роботу таймера, навігацію та підрахунок балів, функціонує належним чином. Важливо, що результати тестування успішно зберігаються в хмарному сховищі Firebase Firestore, що підтверджує коректну інтеграцію з даним сервісом.

Отже, за результатами проведеного функціонального тестування можна зробити висновок, що розроблений мобільний застосунок "Computer Science Testing" в цілому відповідає поставленим функціональним вимогам. Основні сценарії використання працюють коректно, система аутентифікації функціонує належним чином, логіка тестування та підрахунку балів реалізована згідно з проєктом, а результати зберігаються у хмарному сховищі. Виявлені в процесі

тестування дрібні недоліки та помилки були оперативно виправлені. Стабільність роботи застосунку та коректність обробки даних підтверджені.

Таблиця 3.4 Тест проходження тесту

Дія користувача	**Очікуваний результат**	**Фактичний результат**
Увійти в систему, обрати категорію тесту	Відкривається екран тестування, запускається таймер	Відповідає
Послідовно відповісти на всі 30 питань	Для кожного питання відповідь приймається. Відбувається перехід між питаннями	Відповідає
На останньому питанні натиснути "Завершити"	Відбувається перехід на екран результатів	Відповідає
На екрані результатів	Відображається рахунок, наприклад "27 / 30". та коректний витрачений час	Відповідає
Перевірити консоль Firebase Firestore	Створено новий документ з результатом тесту, де все відображається вірно	Відповідає

Висновок до третього розділу

У третьому розділі даної кваліфікаційної роботи було детально описано процес програмної реалізації та подальшого тестування ключових модулів мобільного застосунку. Розробка велася згідно з архітектурними рішеннями та проєктними специфікаціями, визначеними у попередньому розділі, з використанням обраного стеку технологій.

Було успішно імплементовано модуль аутентифікації користувачів за допомогою сервісу Firebase Authentication. Розроблено користувацькі інтерфейси для екранів реєстрації та входу на основі Jetpack Compose, реалізовано логіку валідації введених даних, взаємодію з Firebase для створення нових облікових записів та авторизації існуючих користувачів. Забезпечено обробку основних помилок аутентифікації та управління станом сесії користувача.

Центральний модуль тестування знань було реалізовано з використанням локальної бази даних SQLite, керованої бібліотекою Room, для зберігання та надання тестового контенту. Розроблено екран вибору однієї з двох доступних категорій. На екрані проходження тесту забезпечено динамічне відображення трьох типів запитань, випадковий порядок їх подачі, роботу загального таймера та навігацію між питаннями. Логіка тестування, включаючи обробку відповідей користувача та підрахунок балів для різних типів запитань.

Також було розроблено модуль відображення та збереження результатів. Після завершення тесту користувачеві надається інформація про його успішність (кількість правильних відповідей, загальна кількість питань, витрачений час) на екрані результату. Реалізовано можливість переходу до екрану детальних результатів для поглибленого аналізу відповідей, де візуально позначаються правильні та обрані користувачем варіанти. Результати кожного пройденого тесту авторизованими користувачами зберігаються у хмарній базі даних Firebase Firestore, що включає ідентифікатор користувача, його ім'я, інформацію про категорію, набраний бал, витрачений час та мітку часу.

Проведено функціональне тестування всіх реалізованих модулів та ключових сценаріїв використання застосунку, включаючи реєстрацію, вхід, вибір категорії, проходження тесту з усіма типами питань, роботу таймера, відображення та збереження результатів, а також навігацію між екранами. Тестування підтвердило працездатність та стабільність розробленого програмного продукту та відповідність його функціоналу поставленим вимогам.

Таким чином, на етапі реалізації було успішно втілено всі спроєктовані компоненти та механізми, що дозволило створити завершений програмний продукт, готовий до демонстрації та потенційного подальшого розвитку.

ВИСНОВКИ

У ході виконання дипломної бакалаврської роботи на тему "Мобільний застосунок для тестування знань з комп’ютерних наук" було успішно досягнуто поставлену мету та виконано всі визначені завдання. Результатом роботи є повноцінний програмний продукт – мобільний застосунок "Computer Science Testing", призначений для операційної системи Android, який надає користувачам інтерактивний інструмент для перевірки та поглиблення знань у сфері комп'ютерних наук.

На першому етапі роботи було проведено детальний аналіз предметної області, що включав вивчення сучасних підходів до систем тестування знань та огляд існуючих мобільних застосунків-аналогів. Цей аналіз підтвердив актуальність розробки україномовного продукту з розширеним функціоналом, адаптованого під потреби студентів, зокрема, з можливістю інтеграції специфічного контенту освітніх програм. На основі проведеного дослідження було сформульовано вичерпний перелік функціональних та нефункціональних вимог до розроблюваного застосунку, а також обґрунтовано вибір сучасного стеку технологій, що включає мову програмування Kotlin, UI toolkit Jetpack Compose, архітектурний патерн MVVM, бібліотеку Room для локального зберігання даних та сервіси Firebase (Authentication, Firestore) для управління користувачами та їхніми результатами.

На другому етапі було здійснено проєктування архітектури застосунку. Використання патерну MVVM дозволило чітко структурувати код, розділивши відповідальність між шарами представлення (View), логіки та стану (ViewModel) і даних (Model). Було детально спроєктовано структуру локальної бази даних SQLite за допомогою Room, що включає таблиці для категорій, типів питань, самих питань та варіантів відповідей, а також механізм її початкового наповнення. Для зберігання результатів тестів авторизованих користувачів було спроєктовано структуру колекції в хмарній базі даних Firebase Firestore. Також було розроблено модульну структуру проєкту на рівні пакетів та визначено навігаційну модель застосунку.

Третій розділ роботи був присвячений безпосередній реалізації спроєктованого функціоналу. Було розроблено користувацький інтерфейс для всіх екранів застосунку за допомогою Jetpack Compose з дотриманням принципів Material Design 3. Імплементовано логіку аутентифікації користувачів (реєстрація, вхід, вихід) через Firebase Authentication, включаючи збереження імені користувача. Реалізовано модуль тестування, що підтримує дві категорії питань та три типи запитань із загальним таймером та можливістю навігації між питаннями. Результати кожного тесту зберігаються у Firebase Firestore. Проведено функціональне тестування ключових сценаріїв використання, яке підтвердило працездатність та стабільність розробленого застосунку.

Таким чином, розроблений мобільний застосунок є завершеним програмним продуктом, який відповідає всім поставленим вимогам. Він демонструє застосування сучасних підходів до розробки Android-застосунків, забезпечує зручний та інтерактивний спосіб перевірки знань і має потенціал для подальшого розвитку та вдосконалення. Серед можливих напрямків подальшої роботи можна виділити розширення бази питань та категорій, впровадження детальної статистики та історії проходження тестів для користувачів, додавання рівнів складності, інтеграцію пояснень до відповідей та локалізацію іншими мовами.

Виконання даної дипломної роботи дозволило закріпити теоретичні знання та набути практичних навичок у проєктуванні, розробці, тестуванні та документуванні мобільних застосунків на платформі Android з використанням сучасного стеку технологій.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Technology in education. UNESCO GEM Report 2023. URL: https://gem-report-2023.unesco.org/technology-in-education (дата звернення: 10.04.2025).
2. Juan A. Gómez-Pulido, Liliana Pedraja – Rejas, Camila Muñoz-Fritis, Emilio Rodríguez-Ponce, David Laroze. Mobile Learning and Its Effect on Learning Outcomes and Critical Thinking: A Systematic Review. URL: https://doi.org/10.3390/app14199105 (дата звернення: 10.04.2025).
3. José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. ACM Trans. Comput. Educ. 22, 3, Article 34 (June 2022), 40 pages. https://doi.org/10.1145/3513140 (дата звернення: 10.04.2025).
4. Краснопольський Володимир Едуардович, Поліщук Олена Анатоліївна, Демченко Ольга Миколаївна. Інтеграція мобільних додатків у освітній процес: аналіз ефективності та можливостей для здобувачів освіти. URL: https://doi.org/10.5281/zenodo.11559587 (дата звернення: 18.04.2025).
5. Palshkov, K., Shetelya, N., Khlus, N., Vakulyk, I., & Khyzhniak, I. (2024). Impact of mobile apps in higher education: Evidence on learning. Amazonia Investiga, 13(74), 115-128. https://doi.org/10.34069/AI/2024.74.02.10 (дата звернення: 18.04.2025).
6. Pedro Henriques Abreu, Daniel Castro Silva, and Anabela Gomes. 2018. Multiple-Choice Questions in Programming Courses: Can We Use Them and Are Students Motivated by Them? ACM Trans. Comput. Educ. 19, 14, Article 6 (November 2018), 16 pages. https://doi.org/10.1145/3243137 (дата звернення: 25.04.2025).
7. Knipp, Franz Helmut ; Winiwarter, Werner. / AI-AFACT: Designing AI-Assisted Formative Assessment of Coding Tasks in Web Development Education. Proceedings of the 17th International Conference on Computer Supported Education (CSEDU 2025). Band 2 2025. DOI: 10.5220/0013430500003932.
8. CS IT - Computer Science MCQs. [Мобільний застосунок]. URL: https://play.google.com/store/apps/details?id=com.gopinathmurmu.cse_it_q&hl=en_US (дата звернення: 26.04.2025).
9. Programming Quiz. [Мобільний застосунок]. URL: https://play.google.com/store/apps/details?id=com.ak.tathtatgujarat (дата звернення: 26.04.2025).
10. StatCounter. Mobile Operating Systems Market Share Worldwide. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата звернення: 26.04.2025).

11. StatCounter. Mobile Operating Systems Market Share Ukraine. URL: https://gs.statcounter.com/os-market-share/mobile/ukraine (дата звернення: 26.04.2025).
12. Android Open Source Project. About Android OS. URL: https://source.android.com/docs/setup/about (дата звернення: 02.05.2025).
13. Will Kelly. Is Android fragmentation still a problem for IT teams? TechTarget. URL: https://www.techtarget.com/searchmobilecomputing/tip/Is-Android-fragmentation-still-a-problem-for-IT-teams (дата звернення: 02.05.2025).
14. Android Developers. Kotlin Overview. URL: https://developer.android.com/get-started/overview (дата звернення: 02.05.2025).
15. Android Developers. Why Kotlin is safer. URL: https://developer.android.com/kotlin (дата звернення: 02.05.2025).
16. Android Developers. Android Studio Introduction. URL: https://developer.android.com/studio/intro (дата звернення: 02.05.2025).
17. Android Developers. Gradle Build Overview. URL: https://developer.android.com/build/gradle-build-overview (дата звернення: 02.05.2025).
18. Android Developers. Why Adopt Jetpack Compose. URL: https://developer.android.com/develop/ui/compose/why-adopt (дата звернення: 02.05.2025).
19. Android Developers. Material 3 in Jetpack Compose. URL: https://developer.android.com/develop/ui/compose/designsystems/material3 (дата звернення: 02.05.2025).
20. GeeksforGeeks. MVVM Architecture in Android. URL: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android (дата звернення: 04.05.2025).
21. Dashlane. Android UI Architecture Migration to MVVM. URL: https://www.dashlane.com/blog/android-ui-architecture-mvvm (дата звернення: 04.05.2025).
22. Andrej Vukelic. Data Persistence With Room. URL: https://www.kodeco.com/41058449-data-persistence-with-room (дата звернення: 04.05.2025).
23. Firebase. Firebase Authentication.URL: https://firebase.google.com/products/auth (дата звернення: 20.05.2025).
24. Firebase. Cloud Firestore. URL: https://firebase.google.com/docs/firestore (дата звернення: 04.05.2025).
25. Android Developers. Navigation in Jetpack Compose. URL: https://developer.android.com/develop/ui/compose/navigation (дата звернення: 04.05.2025).

26. Android Developers. Modern Android App Architecture. URL: https://developer.android.com/topic/architecture (дата звернення: 04.05.2025).
27. Android Developers. Room Persistence Library. URL: https://developer.android.com/training/data-storage/room (дата звернення: 05.05.2025).
28. Android Developers. One-to-Many Relationships in Room. URL: https://developer.android.com/training/data-storage/room/relationships/one-to-many (дата звернення: 05.05.2025).
29. Firebase. Firebase Authentication Documentation. URL: https://firebase.google.com/docs/auth (дата звернення: 05.05.2025).
30. Cloud Firestore Data model Documentation. URL: https://firebase.google.com/docs/firestore/data-model (дата звернення: 05.05.2025).

ДОДАТКИ

ДОДАТОК А

Обробка результату реєстрації (файл AuthViewModel.kt)

```
fun registerUser(email: String, password: String, username: String) {
    _authUiState.update { it.copy(isLoading = true, errorMessage = null,
registrationSuccess = false) }
    auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val firebaseUser = auth.currentUser

                if (firebaseUser != null && username.isNotBlank()) {
                    val profileUpdates = UserProfileChangeRequest.Builder()
                        .setDisplayName(username.trim())
                        .build()

                    firebaseUser.updateProfile(profileUpdates)
                        .addOnCompleteListener { profileTask ->
                            _authUiState.update { it.copy(isLoading = false,
registrationSuccess = true, currentUser = auth.currentUser) }
                        }
                } else {
                    _authUiState.update { it.copy(isLoading = false,
registrationSuccess = true, currentUser = firebaseUser) }
                }
            } else {
                val exception = task.exception
                val customErrorMessage = when (exception) {
                    is FirebaseAuthUserCollisionException -> "Користувач з таким
email вже зареєстрований."
                    else -> exception?.localizedMessage ?: "Помилка реєстрації.
Спробуйте пізніше."
                }
                _authUiState.update { it.copy(isLoading = false, errorMessage =
customErrorMessage) }
            }
        }
}
```

ДОДАТОК Б

Composable-функція SingleChoiceOptionRow для відображення та вибору одного варіанту відповіді в тесті(файл QuizScreen.kt)

```
@Composable
fun SingleChoiceOptionRow(
    optionLetter: String,
    text: String,
    isSelected: Boolean,
    onClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Row(
        modifier = modifier
            .fillMaxWidth()
            .background(
                MaterialTheme.colorScheme.primaryContainer.copy(alpha = 1f),
                shape = RoundedCornerShape(12.dp)
            )
            .selectable(
                selected = isSelected,
                onClick = onClick,
                role = Role.RadioButton
            )
            .padding(horizontal = 16.dp, vertical = 12.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        RadioButton(
            selected = isSelected,
            onClick = null
        )
        Spacer(modifier = Modifier.width(8.dp))
        Text(text = buildAnnotatedString {
            withStyle(style = SpanStyle(fontWeight = FontWeight.Bold, color =
Color.Black)) {
                append("$optionLetter) ")
            }
            append(text)
        },
            style = MaterialTheme.typography.bodyLarge
        )
    }
}
```

ДОДАТОК В

Composable-функція MultipleChoiceOptionRow для відображення та вибору декількох варіантів відповіді в тесті(файл QuizScreen.kt)

```
@Composable
fun MultipleChoiceOptionRow(
    optionLetter: String,
    text: String,
    isSelected: Boolean,
    onCheckedChange: () -> Unit,
    modifier: Modifier = Modifier
) {
    Row(
        modifier = modifier
            .fillMaxWidth()
            .background(
                MaterialTheme.colorScheme.primaryContainer.copy(alpha = 1f),
                shape = RoundedCornerShape(12.dp)
            )
            .selectable(
                selected = isSelected,
                onClick = onCheckedChange, // onClick тут викликає
onCheckedChange
                role = Role.Checkbox
            )
            .padding(horizontal = 16.dp, vertical = 12.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Checkbox(
            checked = isSelected,
            onCheckedChange = null
        )
        Spacer(modifier = Modifier.width(8.dp))
        Text(

            text = buildAnnotatedString {
                withStyle(style = SpanStyle(fontWeight = FontWeight.Bold, color
= Color.Black)) {
                    append("$optionLetter) ")
                }
                append(text)
            },
            style = MaterialTheme.typography.bodyLarge
        )
    }
}
```

ДОДАТОК Г

Composable-функція TrueFalseButtonOption для відображення та вибору варіантів "Так/Ні"(файл QuizScreen.kt)

```
@Composable
fun TrueFalseButtonOption(
    text: String,
    isSelected: Boolean,
    onClick: () -> Unit,
    buttonColor: Color,
    modifier: Modifier = Modifier
) {

    if (isSelected) {
        Button(
            onClick = onClick,
            modifier = modifier
                .fillMaxWidth()
                .defaultMinSize(minHeight = 56.dp),
            colors = ButtonDefaults.buttonColors(
                containerColor = buttonColor,
                contentColor = Color.White
            ),
            shape = MaterialTheme.shapes.medium
        ) {
            Text(text = text, fontSize = 18.sp, textAlign = TextAlign.Center)
        }
    } else {
        OutlinedButton(
            onClick = onClick,
            modifier = modifier
                .fillMaxWidth()
                .defaultMinSize(minHeight = 56.dp),
            colors = ButtonDefaults.outlinedButtonColors(
                contentColor = buttonColor
            ),
            border = BorderStroke(2.dp, buttonColor),
            shape = MaterialTheme.shapes.medium
        ) {
            Text(text = text, fontSize = 18.sp, textAlign = TextAlign.Center)
        }
    }
}
```

ДОДАТОК Ґ

Логіка підрахунку балів всередині функції finishQuiz (файл QuizViewModel.kt)

```
var finalScore = 0
val currentQuestions = _uiState.value.questions
val currentUserAnswers = _uiState.value.userAnswers

currentQuestions.forEach { questionUi ->
    val selectedIndices = currentUserAnswers[questionUi.id] ?: emptyList()
    val correctIndices = questionUi.correctOptionIndices

    if (selectedIndices.isNotEmpty()) {
        when (questionUi.typeCode) {
            "SINGLE_CHOICE", "TRUE_FALSE" -> {
                if (selectedIndices.size == 1 &&
correctIndices.contains(selectedIndices.first())) {
                    finalScore++
                }
            }
            "MULTIPLE_CHOICE" -> {
                if (selectedIndices.toSet() == correctIndices.toSet()) {
                    finalScore++
                }
            }
        }
    }
}
```

ДОДАТОК Д

Composable-функція ResultScreen для відображення результату успішності тестування(файл ResultScreen.kt)

```
@Composable
fun ResultScreen(
    score: Int,
    totalQuestions: Int,
    timeSpentFormatted: String,
    onRetryClick: () -> Unit,
    onExitClick: () -> Unit,
    onDetailedResultsClick: () -> Unit
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .systemBarsPadding()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        // Верхня частина з результатами та кнопкою "Детальні результати"
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .weight(1f),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            Text(
                text = "Тест Завершено!",
                style = MaterialTheme.typography.displaySmall,
                fontWeight = FontWeight.Bold,
                modifier = Modifier.padding(bottom = 32.dp)
            )

            Text(
                text = "Ваш результат:",
                style = MaterialTheme.typography.headlineMedium,
                modifier = Modifier.padding(bottom = 16.dp)
            )

            Text(
                text = "$score / $totalQuestions",
                style = MaterialTheme.typography.displayMedium,
                fontWeight = FontWeight.Bold,
                color = MaterialTheme.colorScheme.primary,
                modifier = Modifier.padding(bottom = 24.dp)
            )

            Text(
                text = "Витрачено часу: $timeSpentFormatted",
                style = MaterialTheme.typography.titleLarge,
                textAlign = TextAlign.Center,
                modifier = Modifier.padding(bottom = 24.dp)
            )

            // Кнопка "Детальні результати тесту"
            Button(
                onClick = onDetailedResultsClick,
                modifier = Modifier
                    .fillMaxWidth()
```

```
                    .padding(horizontal = 32.dp)
                    .height(56.dp),
            ) {
                Text("Детальні результати тесту", fontSize = 18.sp)
            }
        }

        // Нижня частина з кнопками "Спробувати ще" та "Вихід"
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .padding(bottom = 16.dp),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.spacedBy(16.dp)
        ) {
            Button(
                onClick = onRetryClick,
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 32.dp)
                    .height(56.dp)
            ) {
                Text("Спробувати ще", fontSize = 18.sp)
            }

            // Кнопка "Вихід"
            Button(
                onClick = onExitClick,
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 32.dp)
                    .height(56.dp),
                colors = ButtonDefaults.buttonColors(
                    containerColor = Color.Red,
                    contentColor = Color.White
                )
            ) {
                Text("Вихід", fontSize = 18.sp)
            }
        }
    }
}
```

ДОДАТОК Е

Composable-функція QuestionResultItem для відображення детального результату кожного окремого питання(файл DetailedResultsScreen.kt)

```
@Composable
fun QuestionResultItem(
    questionNumber: Int,
    question: QuestionUI,
    userSelectedIndices: List<Int>
) {
    Column {
        Text(
            text = "Запитання №$questionNumber: ${question.text}",
            style = MaterialTheme.typography.titleMedium,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(bottom = 12.dp)
        )

        question.options.forEachIndexed { optionIndex, option ->
            val isCorrectOption =
question.correctOptionIndices.contains(optionIndex)
            val wasSelectedByUser = userSelectedIndices.contains(optionIndex)

            var rowColor = Color.Transparent
            var icon: ImageVector? = null
            var iconColor = MaterialTheme.colorScheme.onSurface

            when {
                // Користувач обрав правильний варіант
                wasSelectedByUser && isCorrectOption -> {
                    rowColor = Color.Green.copy(alpha = 0.1f)
                    icon = Icons.Filled.TaskAlt
                    iconColor = Color.Green
                }
                // Користувач обрав неправильний варіант
                wasSelectedByUser && !isCorrectOption -> {
                    rowColor = Color.Red.copy(alpha = 0.1f)
                    icon = Icons.Filled.Cancel
                    iconColor = Color.Red
                }
                // Користувач не обрав, але це був правильний варіант
                !wasSelectedByUser && isCorrectOption -> {
                    rowColor = Color.Green.copy(alpha = 0.05f)
                    icon = Icons.Filled.CheckCircle
                    iconColor = Color.Gray
                }
                // Не обраний і не правильний
                else -> {
                    icon = Icons.Filled.RadioButtonUnchecked
                    iconColor = MaterialTheme.colorScheme.onSurface.copy(alpha =
0.6f)
                }
            }
```

ДОДАТОК Є

Приклад запису результату тесту до Firebase Firestore(файл QuizViewModel.kt)

```
val resultData = hashMapOf(
    "userId" to currentUser.uid,
    "username" to username,
    "categoryId" to currentSelectedCategoryId,
    "categoryName" to categoryName,
    "score" to finalScore,
    "totalQuestions" to currentQuestions.size,
    "timeSpent" to finalTimeSpentFormatted,
    "timestamp" to Timestamp.now()
)

firestore.collection("user_results")
    .add(resultData)
```

ДОДАТОК Ж

Приклад роботи функції Dynamic Color з бібліотеки Material 3 на різних смартфонах з різними шпалерами екрану

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

_____

_____

_____

Повністю вказується ПІБ та статус (освітня (освітньо-наукова) програма – для здобувачів вищої освіти, назва кваліфікаційної роботи)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)