ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АНДРОЇД ЗАСТОСУНКУ ДЛЯ ТЕСТУВАННЯ

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістра

Виконавець роботи Лелюх Олександр Дмитрович
_____ «____»_____ 2025 р.
(підпис)

Науковий керівник к.пед.н., доцент, Кошова Оксана Петрівна
_____ «____»_____ 2025 р.
(підпис)

Рецензент

ПОЛТАВА 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2023 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка програмного забезпечення андроїд застосунку для тестування**»**
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістр
Прізвище, ім'я, по батькові Лелюх Олександр Дмитрович
Затверджена наказом ректора № 206-Н від «27» жовтня 2023 р.
Термін подання студентом роботи «___»_____ 2024 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки застосунків для тестування.

Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Огляд існуючих рішень для проведення тестування.
2.2. Порівняльний аналіз популярних інструментів для тестування.
2.3. Переваги та недоліки існуючих рішень.
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Опис проектних рішень, інструментів та підходів до розробки застосунку.
3.2. Опис методології створення програмного забезпечення та побудова блок схеми застосунку.
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Розробка алгоритму роботи андроїд застосунку для тестування.
4.2. Опис та побудова діаграми класів андроїд застосунку для тестування.
4.3. Інтерфейс користувача та функціональні можливості застосунку.
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Перелік графічного матеріалу: 1 аркуш блок-схем, 16 ілюстрацій.

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Кошова О.П.		
Інформаційний огляд	Кошова О.П.		
Теоретична частина	Кошова О.П.		
Практична частина	Кошова О.П.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
Вступ		
Вивчення методичних рекомендацій та стандартів та звіт керівнику		
Постановка задачі		
Інформаційний огляд джерел бібліотек та інтернету		
Теоретична частина		
Практична частина		
Закінчення оформлення		
Доповідь студента на кафедрі		
Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 2023 р.

Здобувач вищої освіти _____ Лелюх Олександр Дмитрович
(підпис)

Науковий керівник _____ к.пед.н., доц. Кошова О.П.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2025 р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2023 р.

Погоджено
Науковий керівник _____
к.пед.н., Оксана КОШОВА
«____» _____ 2023 р.

План

дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Розробка програмного забезпечення андроїд застосунку для тестування»
Прізвище, ім'я, по батькові Лелюх Олександр Дмитрович

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Огляд існуючих рішень для проведення тестування.
2.2. Порівняльний аналіз популярних інструментів для тестування.
2.3. Переваги та недоліки існуючих рішень.
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Опис проектних рішень, інструментів та підходів до розробки застосунку.
3.2. Опис методології створення програмного забезпечення та побудова блок схеми застосунку.
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Розробка алгоритму роботи андроїд застосунку для тестуваеня.
4.2. Опис та побудова діаграми класів андроїд застосунку для тестування.
4.3. Інтерфейс користувача та функціональні можливості застосунку.
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Здобувач вищої освіти _____ О.Д. Лелюх
(підпис)
«____» _____ 2023 р.

РЕФЕРАТ

Записка: 79 сторінок, 16 рисунків, 1 додаток, 19 літературних джерел.

Мета дипломної роботи є розробка програмного забезпечення андроїд застосунку для тестування.

Об’єкт розробки – андроїд застосунок для тестування.

Методи дослідження – проектування мобільних додатків, зокрема методики UX/UI дизайну. Також використано архітектурний шаблон MVVM (Model-View-ViewModel). Застосунок розроблений із використанням Clean Architecture. Застосовано Navigation Component від Android Jetpack для спрощення реалізації навігації між різними екранами та фрагментами в додатку; View Binding для інтеграції UI компонентів з кодом; Hilt для спрощення процесу управління залежностями в додатку, автоматично забезпечуючи компоненти, такі як класи, сервіси та репозиторії, необхідними ресурсами; Parcelize для ефективної передачі складних даних, як-от об’єктів, між екранами або фрагментами, забезпечуючи легкість та надійність цього процесу. Застосунок розроблений у середовищі розробки Android Studio.

Визначено ключові вимоги до розробки програмного забезпечення андроїд застосунку для проведення тестування; проведено аналіз переваг та обмежень аналогічних застосунків та інструментів для тестування; проаналізовано та описано підходи та інструменти, використані при розробці ПЗ застосунку; описано алгоритм, методологію створення та блок схему розробленого андроїд застосунку для тестування; розроблено мобільний інтерфейс для взаємодії користувача з матеріалом та створено докладну інструкцію користувача; представлено діаграму класів, що ілюструє особливості програмного коду розробленого застосунку; розроблено та протестовано андроїд застосунок для тестування.

ЗМІСТ

ВСТУП

Актуальність. Зростаюча популярність мобільних рішень у галузях освіти і професійного розвитку зумовлює актуальність створення Android-застосунку для тестування у різних категоріях. Поєднання формату тестування з можливостями смартфона, що завжди під рукою, сприяє індивідуалізованій підготовці та оперативному зворотному зв’язку. Крім того, розробка англомовного інтерфейсу розширює коло потенційних користувачів на міжнародному рівні, забезпечуючи доступ до глобальних професійних та академічних спільнот. Такий застосунок можна інтегрувати з сервісами для зберігання даних та аналітики, що значно підвищує ефективність навчального процесу та сприяє вчасному оновленню контенту відповідно до вимог сучасної ринкової кон’юнктури.

Впровадження цього продукту буде корисним для студентів, які прагнуть вдосконалити знання за допомогою оперативного контролю знань, а також для спеціалістів у різних галузях, що вимагають регулярної переатестації чи професійного зростання. Викладачі та тренери зможуть спростити процедуру перевірки результатів та підвищити ефективність навчальних курсів. У свою чергу, розробник, який працюватиме над цим рішенням, отримає унікальний досвід створення масштабованого мобільного ПЗ для масового ринку, розвине компетентності у сфері інтеграції з різноманітними платформами та опанує сучасні підходи до забезпечення безпеки й стабільності застосунків. Це не лише розширить його професійний портфоліо, а й сприятиме науково-дослідним перспективам у галузі інформаційних технологій.

Метою даної роботи є розробка програмного забезпечення андроїд застосунку для тестування. Для досягнення цієї мети передбачено вирішення наступних завдань: аналіз існуючих рішень та технологій у автоматизованого тесування, розробка алгоритму роботи застосунку, створення програмного забезпечення для андроїд застосунку, його тестування та оптимізація.

Об'єктом дослідження є андроїд застосунок для тестування, а *предметом* – розробка програмного забезпечення андроїд застосунку для тестування (англійською мовою). Для досягнення поставлених цілей у роботі буде використано методи збору та аналізу даних, а також програмні засоби для розробки програмного забезпечення.

Перелік методів включає проектування мобільних додатків, зокрема методики UX/UI дизайну. Також використано архітектурний шаблон MVVM (Model-View-ViewModel).

Застосунок розроблений із використанням Clean Architecture , що розділяє бізнес-логіку, дані та представлення. Також використано архітектурний шаблон MVVM (Model-View-ViewModel). Застосовано Navigation Component від Android Jetpack для спрощення реалізації навігації між різними екранами та фрагментами в додатку; View Binding для інтеграції UI компонентів з кодом; Hilt для спрощення процесу управління залежностями в додатку, автоматично забезпечуючи компоненти, такі як класи, сервіси та репозиторії, необхідними ресурсами; Parcelize для ефективної передачі складних даних, як-от об'єктів, між екранами або фрагментами, забезпечуючи легкість та надійність цього процесу. Застосунок розроблений у середовищі розробки Android Studio.

Пояснювальна записка включає чотири основних розділи: постановка задачі, огляд сучасного стану проблеми, теоретична та практична частини. Структура роботи орієнтована на логічне представлення матеріалу та всебічне розкриття теми дослідження.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Мета додатку

Розробка навчального застосунку для тестування знань є актуальним завданням в умовах сучасного розвитку інформаційних технологій та зростання попиту на інтерактивні освітні ресурси. Метою створення такого програмного продукту є забезпечення студентів та викладачів зручним, ефективним і адаптивним інструментом для оцінювання знань. Це дозволяє оптимізувати навчальний процес, підвищити рівень засвоєння матеріалу та забезпечити зворотний зв’язок, необхідний для формування освітньої траєкторії.

Однією з ключових особливостей розробленого застосунку є використання англійської мови для формулювання запитань і відповідей. Цей підхід має кілька суттєвих переваг. По-перше, англійська є універсальною мовою технічних наук, що сприяє підготовці студентів до роботи з міжнародною технічною документацією, алгоритмами та програмними інтерфейсами, які здебільшого представлені англійською. По-друге, такий формат тестування сприяє покращенню мовних навичок, що є особливо важливим для студентів спеціальності «Комп’ютерні науки», адже володіння англійською відкриває доступ до світового освітнього контенту, наукових публікацій та професійних можливостей у глобальному середовищі.

Застосунок є особливо актуальним для студентів спеціальності «Комп’ютерні науки», оскільки тестування знань з дисциплін цієї галузі часто потребує інтеграції теоретичних і практичних аспектів, включаючи алгоритми, структури даних, програмування та математичне моделювання. А при написанні ПЗ цього застосунку ми зможемо використати значну частину знань отриманих під час навчання в університеті. Використання мобільного або десктопного застосунку дозволяє студентам проходити тестування в зручний для них час, що сприяє самоорганізації та покращенню академічної успішності.

Крім того, функції адаптивного тестування дають змогу створювати завдання різного рівня складності, що відповідають поточним знанням студента.

Доцільність розробки такого інструменту обумовлена не лише академічними потребами. Сучасна модель навчання орієнтована на інтерактивність і цифровізацію освітніх процесів, що підвищує ефективність засвоєння знань. У цьому контексті мобільні додатки та веб-рішення стають ключовими елементами цифрових освітніх платформ, що дозволяють адаптувати навчальні програми до потреб студентів і вимог ринку праці.

Цей застосунок буде корисним не лише для студентів, але й для викладачів, які можуть використовувати його як інструмент для моніторингу академічного прогресу. Викладачі можуть швидко оцінювати результати тестів, аналізувати статистику правильних і неправильних відповідей та виявляти складні теми, що потребують додаткового опрацювання.

Окрім академічного середовища, застосунок також може використовуватись у корпоративному навчанні для оцінювання професійних знань співробітників. Його адаптивна архітектура дозволяє створювати спеціалізовані курси та тестові завдання для підвищення кваліфікації в різних галузях.

Використання англійської мови в тестуванні значно розширює аудиторію застосунку, роблячи його придатним для міжнародного використання. Це також сприяє підготовці студентів до глобальної професійної діяльності, де англійська є мовою спілкування та технічної комунікації.

Таким чином, розробка цього застосунку є важливою складовою сучасного освітнього процесу, яка сприяє підвищенню якості навчання, стимулює зацікавленість студентів та забезпечує зручний інструмент для оцінювання знань. Застосування англійської мови для тестування додатково сприяє розвитку мовних навичок студентів, формує готовність до роботи в міжнародному середовищі та відкриває доступ до нових освітніх і професійних можливостей.

Основні функції застосунку

Застосунок для тестування «Quiz Go» - це інтерактивна вікторина для Android-пристроїв, розроблена з використанням сучасних технологій та практик програмування. Він створений на мові програмування Kotlin, яка є популярною для Android-додатків, завдяки своїй чистоті та ефективності. «Quiz Go» підтримується на більшості сучасних Android-пристроїв, починаючи з версії Android 7.0 до 14.0

У додатку використовується спеціальна технологія (ViewBinding), що дозволяє розробникам легко управляти інтерфейсом, роблячи взаємодію з користувачем більш інтуїтивною та зручною. Крім того, в «Quiz Go» впроваджені передові методи для організації коду, включаючи Hilt для управління залежностями, що допомагає підтримувати код чистим і легким для розуміння.

Зовнішній вигляд додатку є стильним та сучасним, завдяки використанню останніх бібліотек для дизайну, таких як Material Design. «Quiz Go» обіцяє бути не тільки корисним, але й візуально привабливим додатком для любителів вікторин.

Серед функціональних можливостей застосунку реалізовано наступні:

1. Вибір категорій тестування.

Застосунок надає користувачу можливість вибору категорії тестування через інтерактивний список у фрагменті MenuFragment. Для кожної категорії (математика, біологія, кіно, історія, програмування) використовується модель ModelQuiz, яка містить інформацію про назву категорії, її іконку та кількість запитань.

У класі RecyclerAdapterQuiz створено адаптер для відображення списку категорій у RecyclerView, що забезпечує інтерактивність.

2. Проведення тестування.

Основна функціональність реалізована у фрагменті QuizFragment, який відповідає за проведення вікторини. Тестування включає відображення запитань, варіантів відповідей і перевірку правильності відповідей.

- Запитання динамічно оновлюються завдяки використанню ViewModel (ViewModelQuiz) та LiveData.
- Відповіді обробляються через метод onClick, який перевіряє правильність відповіді та оновлює статистику.

3. Збереження стану тестування.

Для забезпечення збереження стану тестування використовуються ViewModel та LiveData. ViewModel зберігає поточний список запитань і номер активного запитання, що дозволяє уникнути втрати даних під час змін конфігурації пристрою.

4. Обробка відповідей.

Після вибору відповіді користувачем система визначає, чи є вона правильною.

Реалізовано візуальний зворотний зв’язок (через іконки правильності) для кожного вибору відповіді.

5. Відображення результатів.

Після завершення тестування користувачу надається діалогове вікно з результатами у ResultDialogFragment.

6. Управління запитаннями.

Запитання для кожної категорії зберігаються у репозиторії DefaultRepository, реалізованому класом DefaultRepositoryImp. Цей клас отримує запитання з окремих джерел даних, таких як Biology, Math, Programming тощо.

Репозиторій забезпечує централізований доступ до даних, дозволяючи масштабувати систему шляхом додавання нових категорій або запитань.

7. Ін’єкція залежностей.

Застосунок використовує Hilt для ін’єкції залежностей. Наприклад, залежності, такі як DefaultRepository, автоматично впроваджуються у фрагменти та адаптери.

РОЗДІЛ 2. ОГЛЯД СИСТЕМ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ

2.1. Огляд існуючих рішень для проведення тестування

Розробка програмного забезпечення для навчальних застосунків і систем тестування є актуальним завданням сучасної освіти, зокрема в умовах цифровізації та дистанційного навчання. У роботах, виконаних студентами спеціальності «Комп'ютерні науки» Полтавського університету економіки і торгівлі, розглядаються різні аспекти створення подібних програмних рішень, які поєднують методи тестування знань із використанням інноваційних технологій.

Зокрема у дослідженні «Розробка десктопного навчального застосунку для тестування на тему 'Рекурсивні алгоритми'» (автор: Тарасюк) розкривається аспект використання цього інструменту для поглибленого вивчення складних концепцій програмування. Програмне забезпечення забезпечує реалізацію інтуїтивно зрозумілого інтерфейсу користувача, який підтримує інтерактивну взаємодію. Застосунок дозволяє студентам не лише проходити тестування, але й відслідковувати помилки та отримувати роз'яснення до кожного запитання. Важливою особливістю є інтеграція анімацій для візуалізації виконання рекурсивних алгоритмів, що значно покращує засвоєння матеріалу [2].

У роботі «Розробка програмного забезпечення тренажеру з теми 'Матриці'» (автор: Волжан) зосереджується увага на створенні інструменту для закріплення теоретичних знань і формування практичних навичок. Програмне забезпечення передбачає інтерактивний режим виконання завдань, який охоплює різні аспекти роботи з матрицями, включаючи операції додавання, множення та обчислення визначників. Система надає миттєвий зворотний зв'язок користувачам, що дозволяє їм аналізувати свої помилки. Високий рівень адаптивності інструменту дозволяє застосовувати його як для самостійної

роботи, так і в рамках дистанційних курсів [5].

У роботі «Розробка програмного забезпечення тренажеру з теми ‘Верифікація програм’» (автор: Сімперович М.М.) розроблено програмне забезпечення, орієнтоване на англомовних студентів. Програмний продукт включає сценарії роботи з верифікаційними моделями, які дозволяють студентам аналізувати процеси перевірки коректності програмного коду. Особливістю є включення інтерактивних завдань, які моделюють реальні приклади використання технік верифікації. Такий підхід сприяє не лише кращому розумінню теоретичних аспектів, але й формуванню навичок їх практичного застосування [6].

У дослідженні «Розробка програмного забезпечення тренажеру з теми ‘Основні методології програмування’» (автор: Банасюкевич Л.С.) основну увагу приділено створенню засобу для ознайомлення з ключовими концепціями різних методологій розробки програмного забезпечення. Програмне забезпечення містить багаторівневу систему завдань, які включають елементи моделювання, аналізу та практичного застосування. Однією з особливостей є використання інтерактивних графічних інструментів, які дозволяють візуалізувати процеси, характерні для таких методологій, як Agile, Waterfall та DevOps. Цей інструмент є універсальним засобом навчання, який може бути інтегрований у курси з теорії програмування [7].

У кваліфікаційній роботі Задворкіна М. О. «Розробка системи тестування знань з курсу ‘Основи наукових досліджень в інформатиці’» представлено розробку програмного забезпечення, яке автоматизує процес оцінювання знань студентів. Система забезпечує створення, редагування та генерування тестів, підтримує багаторівневі питання і ведення звітності про результати. Робота базується на використанні об’єктно-орієнтованого підходу, UML-діаграм для моделювання системи та сучасних технологій для інтеграції з базою даних. Інноваційний підхід включає адаптивне тестування, яке змінює рівень складності питань залежно від результатів студента, а також алгоритми аналізу

помилок для персоналізованих рекомендацій. Розроблена система є актуальною для підвищення ефективності навчального процесу в галузі інформатики [15].

У статті [19] детально розглядаються етапи розробки навчального андроїд-застосунку, який інтегровано в дистанційний курс «Алгоритми і структури даних». Автори обґрунтовують вибір теми «Сортування вставками» як однієї з базових та важливих для розуміння студентами. Вони описують методологію розробки, включаючи аналіз вимог, проєктування інтерфейсу користувача та реалізацію функціоналу додатку.

Особлива увага приділяється інтерактивним елементам застосунку, які сприяють кращому засвоєнню матеріалу студентами. Автори підкреслюють важливість використання мобільних технологій у сучасному освітньому процесі, зазначаючи, що такий підхід підвищує мотивацію та залученість студентів.

Таким чином, кожна з представлених робіт робить внесок у розвиток цифрових освітніх технологій, пропонуючи інноваційні рішення для автоматизації тестування, підвищення мотивації студентів і формування практичних навичок. Усі розглянуті інструменти орієнтовані на інтерактивну взаємодію користувачів із програмним забезпеченням, що є важливим чинником у сучасному освітньому процесі.

Крім застосунків, розроблених здобувачами вищої освіти нашої кафедри існує і чимало більш популярних рішень для тестування відомих у всьому світі.

У сучасному світі тестування знань є важливим компонентом освітнього процесу, який активно використовують як у формальній, так і в неформальній освіті. Завдяки цифровізації навчання, значна частина тестування реалізується за допомогою програмних рішень, які забезпечують автоматизацію процесів оцінювання знань, адаптацію тестів до рівня підготовки користувачів та інтерактивну взаємодію. Серед численних програмних інструментів для тестування знань найбільш популярними є Moodle, Kahoot, Google Forms, Socrative та Edmodo.

Moodle є одним із найпоширеніших рішень для організації тестування та управління навчальними курсами. Ця платформа забезпечує інтеграцію тестування в освітні процеси завдяки розширеним можливостям створення завдань, серед яких багатоваріантні відповіді, відповідність, відкрита відповідь та завдання з обчисленням. Moodle підтримує адаптивне тестування, дозволяючи викладачам налаштовувати складність запитань залежно від результатів студентів. Платформа також пропонує детальні звіти про результати тестування, що сприяє аналізу успішності навчання.

Рисунок 1.1. Лого платформи Moodle

Kahoot є інструментом для інтерактивного тестування, який широко використовується в освітніх закладах різних рівнів. Основна перевага Kahoot полягає в ігровій формі тестування, яка підвищує мотивацію учнів. Використання яскравого дизайну, обмеження часу на відповіді та можливості організації командної гри робить платформу популярною серед молодої аудиторії. Крім того, Kahoot підтримує створення квізів, обмін ними між викладачами та зберігання статистики проходження.

Рисунок 1.2. Лого Kahoot

Google Forms є універсальним і зручним інструментом для швидкого створення тестів. Популярність цього рішення зумовлена простотою використання, доступністю на будь-якому пристрої та безкоштовним доступом. Google Forms підтримує автоматичне оцінювання відповідей, що знижує навантаження на викладачів. Платформа також забезпечує інтеграцію з іншими сервісами Google, такими як Google Sheets, для зберігання і аналізу результатів.

Socrative є спеціалізованою платформою для проведення тестування, яка пропонує різноманітні типи завдань, включаючи відкриті запитання, тести з вибором відповіді та швидкі опитування. Платформа дозволяє проводити тестування в реальному часі, що робить її зручною для інтерактивних занять. Важливою перевагою Socrative є можливість створення індивідуальних звітів для кожного студента, які можна використовувати для формування персоналізованих рекомендацій.

Рисунок 1.3. Лого Socrative

Edmodo є освітньою платформою, що забезпечує інтеграцію навчальних матеріалів із тестуванням знань. Ця платформа використовується як для класичних форм навчання, так і для дистанційного навчання. Edmodo пропонує інструменти для створення тестів, роздачі завдань і обміну матеріалами між учнями та викладачами. Завдяки широкій функціональності платформа дозволяє оцінювати не лише правильність відповідей, але й прогрес кожного студента в рамках курсу.

Популярність цих рішень зумовлена їхньою функціональністю, зручністю використання та адаптивністю до потреб освітнього процесу. Вони

забезпечують викладачів інструментами для автоматизації оцінювання, аналізу результатів і створення індивідуальних підходів до навчання. У сукупності ці рішення сприяють підвищенню якості освітнього процесу та ефективності навчання у сучасному цифровому середовищі.

Тестування знань є важливою складовою як формальної, так і неформальної освіти, а також корпоративного навчання. Завдяки розвитку цифрових технологій, ринок програмного забезпечення пропонує численні рішення для тестування, серед яких особливо популярними є мобільні Android-застосунки та десктопні програми. Вони забезпечують інтерактивність, доступність і високу ефективність оцінювання знань. Нижче розглянуто кілька найбільш відомих програмних рішень у цій сфері.

Мобільні додатки на базі Android забезпечують користувачам гнучкість і доступність у проведенні тестування завдяки їхній портативності та інтеграції з іншими сервісами.

1. Quizlet є одним із найпопулярніших мобільних додатків для навчання та тестування. Застосунок дозволяє створювати набори флешкарт для вивчення матеріалу, а також проходити тести, що автоматично генеруються на основі цих наборів. Quizlet підтримує різні режими, включаючи «Правильні відповіді» і «Написання», які допомагають користувачам адаптувати тестування до своїх потреб.

«Quizlet» надає можливість створювати власні набори карток або використовувати готові набори, які створили інші користувачі. Крім того, сервіс пропонує різноманітні режими навчання: тести, ігри, вивчення та інше. Основна мета – зробити процес навчання цікавим та ефективним.

Рисунок 1.4. Лого інтерактивного сервісу «Quizlet»

2. Kahoot! початково був орієнтований на інтерактивні квізи, доступні через веб-браузери, його мобільна версія забезпечує всі функціональні можливості для проведення тестування. Застосунок дозволяє користувачам брати участь у тестах у реальному часі, змагатися в командних режимах і аналізувати свої результати.
3. Duolingo орієнтований переважно на вивчення мов, його система включає елементи тестування, які допомагають користувачам перевіряти знання граматики, словника та синтаксису. Тестування здійснюється через інтерактивні вправи, які включають заповнення пропусків, вибір правильного варіанту та переклад.
4. Додаток Google Classroom надає можливість організовувати навчальні курси з інтегрованим тестуванням. У ньому реалізовано функції створення запитань із множинним вибором, коротких відповідей та інших типів завдань. Результати тестів автоматично синхронізуються між викладачем і студентами.

Водночас, десктопні програми орієнтовані на складніші середовища, які потребують багатофункціональності та масштабованості.

1. Moodle є платформою з відкритим кодом, яка широко використовується в навчальних закладах для управління навчальними курсами. Її десктопна версія дозволяє створювати складні тестові сценарії, включаючи адаптивне тестування та завдання з інтеграцією мультимедійних ресурсів.
2. Testmoz є інструментом для створення онлайн-тестів, який також підтримується в десктопному середовищі. Його основною перевагою є простота використання та можливість швидкого генерування тестів з різними типами запитань, такими як множинний вибір і відповідність.
3. Respondus LockDown Browser. Ця програма створена для забезпечення академічної чесності під час тестування. Вона блокує доступ до інших

програм, веб-браузерів і файлів під час проходження тесту, забезпечуючи безпечне середовище для проведення екзаменів.

4. Hot Potatoes — це інструмент для створення тестів, який підтримує різні формати, включаючи кросворди, вправи на відповідність і тести з множинним вибором. Ця програма особливо популярна у викладачів завдяки своїй простоті та універсальності.

Застосунки для тестування, як мобільні, так і десктопні, забезпечують широкий спектр можливостей для проведення оцінювання знань. Android-застосунки, такі як Quizlet і Kahoot!, акцентуються на інтерактивності та мобільності, що робить їх популярними серед учнів і викладачів. Десктопні програми, такі як Moodle та Respondus LockDown Browser, орієнтовані на більш складні завдання, включаючи управління великими навчальними курсами та забезпечення академічної чесності. Поєднання цих рішень дозволяє ефективно реалізовувати різноманітні підходи до тестування в сучасному освітньому середовищі.

2.2. Порівняльний аналіз популярних інструментів для тестування

Тестування знань, як важливий компонент освітнього процесу, реалізується за допомогою різноманітних програмних інструментів, які відрізняються функціональними можливостями, методами взаємодії з користувачами та архітектурою програмного забезпечення. Порівняльний аналіз Android-застосунків (Quizlet, Kahoot!, Duolingo, Google Classroom) та десктопних програм (Moodle, Testmoz, Respondus LockDown Browser, Hot Potatoes) дозволяє виявити їхні особливості та ефективність у різних навчальних сценаріях.

Мобільні Android-застосунки для тестування, такі як Quizlet та Kahoot!,

акцентуються на інтерактивності, простоті використання та гейміфікації. Архітектура цих інструментів побудована навколо забезпечення швидкої взаємодії користувача з системою. Наприклад, Quizlet використовує модель флешкарт, що ґрунтується на хмарному зберіганні даних для синхронізації прогресу між пристроями. Kahoot!, у свою чергу, реалізує ігрову форму тестування з елементами реального часу, що забезпечується технологіями WebSocket для швидкої передачі даних між користувачами. Основна відмінність цих застосунків від десктопних рішень полягає у їхній адаптації до мобільного середовища, зокрема, використанні адаптивного дизайну та оптимізації під обмежені ресурси пристроїв.

Десктопні програми, такі як Moodle та Testmoz, орієнтовані на комплексне управління навчальними курсами. Moodle, як платформа з відкритим кодом, підтримує розширену архітектуру модулів, що дозволяє інтегрувати додаткові плагіни для розширення функціональності. Особливістю Moodle є його здатність масштабуватися для підтримки великої кількості користувачів, що досягається через розподілене зберігання даних і балансування навантаження на сервери. Testmoz, натомість, відомий своєю мінімалістичною архітектурою, що спрощує створення тестів і аналіз результатів, але обмежує можливості інтеграції з іншими платформами.

Respondus LockDown Browser демонструє вузькоспеціалізований підхід до забезпечення академічної чесності під час тестування. Його архітектура спрямована на ізоляцію середовища користувача, що досягається через обмеження доступу до інших програм, файлів та веб-ресурсів. Цей інструмент базується на принципах забезпечення безпеки, використовуючи методи моніторингу процесів операційної системи та шифрування даних для збереження конфіденційності.

Hot Potatoes вирізняється серед інших десктопних інструментів завдяки простоті розробки та інтеграції освітнього контенту у формати, доступні для публікації в мережі Інтернет. Програма побудована на архітектурі, орієнтованій

на створення статичних файлів (HTML), що забезпечує незалежність від серверних ресурсів, але обмежує можливості динамічного тестування.

Google Classroom, як мобільний і веб-орієнтований інструмент, інтегрує тестування з іншими сервісами Google, такими як Google Forms та Google Sheets. Його архітектура базується на хмарних технологіях, що дозволяє забезпечувати синхронізацію даних, безперервну роботу в реальному часі та аналіз великих обсягів даних.

Таким чином, Android-застосунки демонструють інтерактивність і мобільність, тоді як десктопні програми орієнтовані на складніші освітні середовища, забезпечуючи розширену функціональність і масштабованість. Особливості архітектури кожного інструменту відображають їхню спрямованість на певну категорію користувачів: від індивідуальних учнів до великих навчальних закладів. Аналіз засвідчує, що вибір інструменту має базуватись на специфічних потребах навчального процесу, доступних ресурсах та освітніх цілях.

2.3. Переваги та недоліки розробки існуючих рішень

Програмне забезпечення для тестувальних застосунків відіграє важливу роль у сучасному освітньому процесі, забезпечуючи автоматизацію оцінювання знань та підвищення мотивації користувачів. Попри численні переваги таких рішень, їхня розробка супроводжується низкою складнощів, пов’язаних із вибором технологій, архітектурних рішень та забезпеченням масштабованості.

Основною перевагою таких систем є автоматизація процесу оцінювання знань, яка знижує навантаження на викладачів та оптимізує навчальний процес. Інтерактивність, забезпечена мобільними та десктопними застосунками, сприяє кращому засвоєнню матеріалу студентами через візуалізацію, гейміфікацію та адаптацію складності завдань. Такі платформи, як Moodle, Quizlet та Kahoot!,

демонструють значний вплив на ефективність навчання завдяки підтримці багатокористувацького середовища та доступності на різних пристроях. Крім того, використання хмарних технологій у Google Classroom та Testmoz забезпечує синхронізацію даних і можливість проведення тестування в режимі реального часу.

Попри переваги, існують і недоліки. Залежність від інтернет-з'єднання у багатьох хмарних рішеннях може обмежувати доступність у регіонах із низьким рівнем цифрової інфраструктури. Інструменти, такі як Respondus LockDown Browser, які забезпечують високий рівень безпеки, часто стикаються з проблемами сумісності з різними пристроями та операційними системами. Крім того, складність використання деяких десктопних рішень, таких як Hot Potatoes, може вимагати додаткового навчання викладачів для їх ефективного впровадження.

Розробка таких програмних рішень вимагає врахування різних аспектів, включаючи масштабованість, безпеку, доступність і зручність використання. Складнощі розробки зумовлені необхідністю інтеграції різноманітних компонентів, таких як управління даними, забезпечення взаємодії в реальному часі та підтримка різних типів завдань. Наприклад, для реалізації адаптивного тестування необхідно створювати алгоритми, які динамічно змінюють складність запитань залежно від результатів користувача.

Розробка тестувальних застосунків базується на використанні сучасних мов програмування, фреймворків та бібліотек. Для мобільних застосунків, таких як Quizlet і Kahoot!, активно використовуються Java та Kotlin, які є стандартами для розробки Android-додатків. Інструменти гейміфікації часто реалізуються через бібліотеки OpenGL ES для створення графічних елементів. Хмарні рішення, такі як Google Classroom, базуються на JavaScript та TypeScript у поєднанні з фреймворками Angular і Node.js, що забезпечує інтерактивність і підтримку великої кількості користувачів.

Десктопні програми, такі як Moodle, побудовані на мові PHP із

використанням баз даних MySQL або PostgreSQL для зберігання інформації про тести та користувачів. Testmoz використовує простіші технології, такі як HTML, CSS і JavaScript, у поєднанні з Ruby on Rails для серверної частини. Для створення захищених середовищ у Respondus LockDown Browser застосовуються C++ і API операційної системи для обмеження доступу до небажаних ресурсів.

Розробка тестувальних застосунків є багатогранним процесом, який потребує врахування численних технічних і користувацьких аспектів. Незважаючи на складнощі, такі як забезпечення безпеки, сумісності та масштабованості, ці інструменти продовжують залишатися важливими складовими освітнього процесу. Використання передових мов програмування та технологій дозволяє створювати потужні, адаптивні рішення, які відповідають вимогам сучасного цифрового навчання.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Опис проектних рішень, інструментів та підходів до розробки застосунку

Для розробки програмного забезпечення андроїд застосунку було використано наступні технології:

- мова програмування Kotlin;
- архітектурний шаблон MVVM (Model-View-ViewModel), який допомагає відокремлювати бізнес-логіку додатку від його інтерфейсу;
- Navigation Component від Android Jetpack для спрощення реалізації навігації між різними екранами та фрагментами в додатку;
- View Binding для інтеграції UI компонентів з кодом;
- Clean Architecture для розділення різних аспекти програми на окремі шари;
- Hilt для спрощення процесу управління залежностями в додатку, автоматично забезпечуючи компоненти, такі як класи, сервіси та репозиторії, необхідними ресурсами;
- Parcelize для ефективної передачі складних даних, як-от об’єктів, між екранами або фрагментами, забезпечуючи легкість та надійність цього процесу;
- середовище розробки Android Studio.

Мова програмування Kotlin.

Kotlin - це сучасна мова програмування, спеціально розроблена для створення мобільних застосунків на платформі Android. Вона здобула швидке визнання завдяки своїй лаконічності, надійності й тісній інтеграції з існуючими засобами Android. Як статично типізована мова, що працює на Java Virtual Machine (JVM), Kotlin повністю сумісна з Java й дає змогу використовувати обидві мови в одному проєкті. Офіційна рекомендація Google застосовувати

Kotlin для розробки Android-застосунків стала важливою віхою в еволюції платформи (Рисунок 3.1).

Рисунок 3.1. Логотип Kotlin.

Однією з головних переваг Kotlin порівняно з Java є його лаконічність. Завдяки автоматичному виведенню типів (type inference), можливості розширення методів (extension functions) та більш компактному синтаксису, мова дає змогу значно скоротити кількість коду. Це не лише підвищує ефективність розробки, а й зменшує ризик появи помилок, роблячи код зрозумілішим і зручнішим для супроводу. Крім того, Kotlin реалізує механізм контролю null-значень на етапі компіляції, що запобігає виникненню NullPointerException і сприяє підвищенню надійності застосунків.

Важливою особливістю Kotlin є підтримка об’єктно-орієнтованої та функціональної парадигм програмування, що надає розробникам гнучкість у виборі підходу до проєктування програмного забезпечення. Мова підтримує функції вищого порядку, лямбда-вирази та має розширені можливості роботи з колекціями, завдяки чому чудово пасує для вирішення складних завдань та імплементації розвиненої бізнес-логіки.

Інтеграція Kotlin з Android Studio відбувається бездоганно: розробники можуть за потреби швидко переходити з Java на Kotlin, не втрачаючи продуктивності чи функціональності. Інструменти автоматичної конвертації коду з Java у Kotlin спрощують міграцію на нову мову, дозволяючи

вбудовувати Kotlin до вже створених проєктів поступово, без кардинальної перебудови. Це робить Kotlin зручним вибором для розробників, звичних до Java, не змушуючи їх докорінно змінювати підхід до розробки.

Завдяки офіційній підтримці Google та органічній інтеграції з компонентами Android Jetpack, Kotlin є ключовим елементом сучасного процесу створення застосунків. Постійні оновлення та активна спільнота сприяють подальшому розвитку мови й відкривають нові можливості для проєктування більш ефективних, надійних і гнучких мобільних застосунків. Зрештою, переваги Kotlin — лаконічність, надійність і сумісність із Java — роблять його одним із найважливіших інструментів розробників Android [9, 10] .

View Binding

View Binding - це частина Android Jetpack, яка суттєво спрощує взаємодію з UI-компонентами у застосунках та робить цей процес безпечнішим, позбавляючи розробників необхідності вручну викликати findViewById() чи використовувати небезпечні перетворення типів [3].

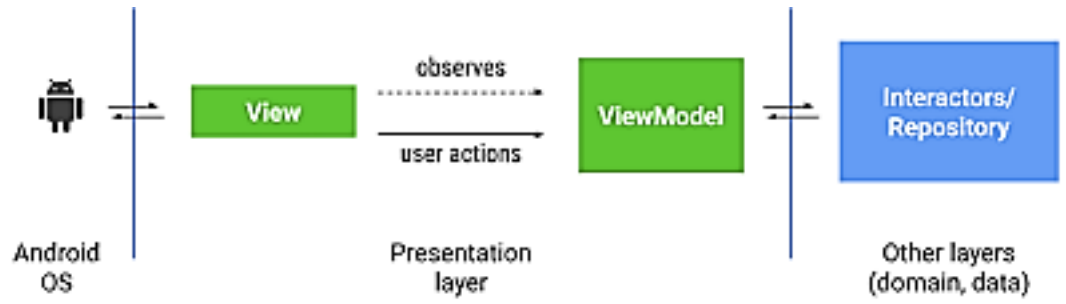

Рисунок 3.1. Процес отримання доступу до переглядів на Android

Завдяки автоматичному створенню спеціальних класів прив’язки для кожного XML-макета, View Binding забезпечує безпосередній доступ до візуальних елементів без зайвого шаблонного коду. Кожен згенерований клас

охоплює всі елементи з унікальними ідентифікаторами, що гарантує коректне посилання на кожен із них і виключає помилки, пов’язані з неправильним доступом до Views або некоректними кастами типів.

Інструмент працює на етапі компіляції, відтак заздалегідь перевіряє, чи всі елементи наявні у відповідних макетах. Це дозволяє уникнути помилок під час виконання, коли програма намагається звернутися до неіснуючого ресурсу або використати неправильний тип. Крім цього, View Binding легко інтегрується з фрагментами, діалогами та іншими ключовими компонентами Android, зокрема RecyclerView, що робить його універсальним рішенням для більшості сценаріїв розробки інтерфейсу. У фрагментах, наприклад, View Binding спрощує роботу з життєвим циклом та знижує ризик витоків пам’яті.

Ця технологія також гармонійно поєднується з іншими складовими Android Jetpack, зокрема Data Binding, що надає можливості двостороннього зв’язку даних чи роботи зі спостерігачами (observers). Водночас, на відміну від Data Binding, View Binding не вимагає змін у XML-макетах та додаткових анотацій, залишаючись простим і легким у використанні.

Отже, View Binding - це ефективний і безпечний спосіб організувати роботу з інтерфейсом користувача в Android-застосунках, який прискорює розробку, підвищує надійність коду та зменшує ймовірність виникнення помилок.

Clean Architecture

Clean Architecture — це спосіб проєктування програмного забезпечення, спрямований на створення чітко розмежованих та гнучких систем, які не прив’язані до конкретних технологій. Цей підхід, запропонований Робертом С. Мартіном (Uncle Bob), здобув популярність у складних і масштабних проєктах. Головна ідея полягає у забезпеченні простоти змін та довговічності програмного продукту шляхом чіткого розподілу відповідальностей між різними складовими.

Ключовим принципом Clean Architecture є ізоляція бізнес-логіки та доменних правил від усіх деталей реалізації на кшталт користувацьких інтерфейсів, баз даних та бібліотек. Це реалізується через концентричну структуру, де в самому центрі розміщені компоненти, максимально незалежні від зовнішніх впливів, а у периферійних шарах перебувають адаптери та рішення, які можуть змінюватися без шкоди для ядра. Найглибший шар зазвичай складається з так званих сутностей (entities), що описують бізнес-об’єкти й визначають їхні основні правила. Ці сутності не залежать від інших компонентів системи, тож залишаються стабільними навіть за зміни технологічного середовища [12].

На наступному рівні знаходяться так звані use cases (інтерфейси використання), які формалізують бізнес-логіку та виступають координаторами між системою і користувачами. Головна мета цього шару — дотримання доменних правил, незалежно від конкретних технічних деталей реалізації. Завдяки такому підходу можна швидко змінювати чи розширювати функціонал застосунку без ризику порушити внутрішні процеси. Після шару use cases розміщуються рівні інтерфейсів та інфраструктури, що охоплюють адаптери для роботи з базами даних, API та інтерфейсами користувача. Вони можуть вільно замінюватися або модифікуватися, не зачіпаючи ключову бізнес-логіку, і саме це є однією з вагомих переваг Clean Architecture.

Суттєвою рисою цього підходу є залежність від абстракцій, а не від конкретних реалізацій, що істотно знижує ризики, пов’язані зі зміною технологій або середовища виконання, й забезпечує модульну структуру системи. Розробники мають змогу легко заміняти окремі складові або переносити продукт на нові платформи з мінімальними витратами на рефакторинг. Наприклад, зміна бази даних чи бібліотеки для UI вимагає лише оновлення відповідних адаптерів, тоді як ядро залишається стабільним [12].

Отже, Clean Architecture спрямована на створення систем, що легко адаптуються до змін, є простими у тестуванні та зручними для тривалої

підтримки. Завдяки розмежуванню відповідальностей, орієнтації на абстракції та високому рівню модульності цей підхід підходить як для невеликих застосунків, так і для масштабних корпоративних рішень.

Navigation Component у Android Jetpack

Navigation Component є частиною бібліотеки Android Jetpack, що забезпечує зручні та ефективні засоби для реалізації навігації в мобільних додатках. Цей компонент спрямований на спрощення управління навігацією між екранами, дотримуючись рекомендацій з проектування інтерфейсів, відомих як Material Design. Він надає розробникам набір інструментів для створення структурованих, зрозумілих і масштабованих навігаційних схем, які відповідають сучасним вимогам до програмного забезпечення.

Navigation Component складається з кількох ключових елементів, які взаємодіють між собою для побудови навігаційної архітектури. Навігаційна графіка, визначена у вигляді XML-файлу, виконує роль основного центру керування переходами між екранами. У цьому файлі визначаються пункти навігації (destinations), які представляють фрагменти, активності або навіть діалогові вікна. Інструмент NavHost виступає контейнером для відображення визначених пунктів навігації, тоді як NavController забезпечує керування переходами між цими пунктами відповідно до вказаних дій (actions).

Однією з головних переваг Navigation Component є усунення необхідності вручну писати складний код для навігації. Наприклад, інтеграція NavController дозволяє легко здійснювати переходи між екранами, використовуючи попередньо визначені дії. Крім того, ця технологія підтримує анімації переходів, передачу параметрів між екранами та автоматичне управління стеком повернення, що значно спрощує розробку та підвищує якість програмного забезпечення.

Navigation Component забезпечує підтримку різних типів навігації, включаючи навігацію на основі фрагментів, односторонню навігацію, глибоку

навігацію (deep linking) та глобальні дії, які можна викликати з будь-якого місця програми. Інтеграція з іншими компонентами Jetpack, такими як ViewModel і LiveData, дозволяє передавати дані між екранами без втрати стану під час змін конфігурації пристрою.

У науковому та практичному контексті Navigation Component сприяє створенню архітектурно коректних і підтримуваних додатків, які легко адаптуються до змін вимог і масштабуються без значних витрат на рефакторинг. Завдяки автоматизації багатьох аспектів навігації, ця бібліотека стала невід'ємною частиною сучасного інструментарію розробників Android-додатків [14].

Архітектура MVVM (Model-View-ViewModel).

Model-View-ViewModel (MVVM) є архітектурним патерном, широко використовуваним у розробці мобільних застосунків, зокрема для платформи Android. Він забезпечує чітке розділення відповідальностей між різними компонентами застосунку, що дозволяє створювати модульний, масштабований та легкий для підтримки код. Головна ідея цього патерну полягає у відокремленні бізнес-логіки та логіки відображення від користувацького інтерфейсу, що робить застосунок простішим у підтримці та тестуванні [11, 12].

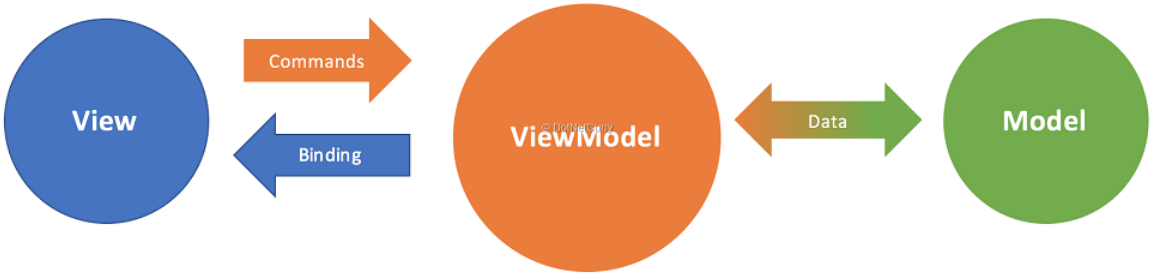

Рисунок 3.3. Архітектура MVVM

У шаблоні архітектури MVVM виокремлюють три головні складові: Model, View і ViewModel.

Model відповідає за управління даними, їх зберігання та взаємодію із зовнішніми джерелами - такими, як бази даних чи мережеві API. Саме тут розміщено бізнес-логіку, яка регламентує порядок обробки інформації. View зосереджена на графічному інтерфейсі та реакції на дії користувача. Вона не містить бізнес-логіки чи алгоритмів роботи з даними, а лише відображає отриману інформацію та реагує на користувацькі події.

ViewModel є центральною ланкою між Model і View. Її завдання — забезпечити отримання даних із Model і підготувати їх до відображення у View. ViewModel містить логіку, пов’язану зі станом інтерфейсу, проте не містить даних про конкретну структуру або зовнішній вигляд елементів інтерфейсу. Таким чином, відбувається чітке розмежування між процесами обробки та подання даних, що спрощує тестування бізнес-логіки без необхідності включати в процес тестування візуальну складову.

Однією з ключових переваг MVVM є сумісність із Android-архітектурними компонентами, зокрема LiveData і ViewModel, завдяки чому розробники можуть ефективно керувати життєвим циклом застосунків і динамічно оновлювати інтерфейс. LiveData дозволяє View автоматично реагувати на зміни в Model через ViewModel, зводячи до мінімуму ручне керування станом екрана. Це дає можливість зосередитися на реалізації бізнес-логіки, не витрачаючи надмірних зусиль на синхронізацію даних із UI.

Ще одна важлива перевага полягає в тому, що ViewModel можна легко тестувати окремо від View. Оскільки ViewModel не має прямого доступу до конкретних графічних елементів, створювати юніт-тести, спрямовані на бізнес-логіку та механізми управління станом, можна без ускладнень, які зазвичай виникають під час тестування візуальних компонентів. Завдяки цьому значно зростає надійність застосунків, а помилки виявляються й виправляються ще до того, як доходитиметься до розробки чи інтеграції інтерфейсу.

Окрім цього, MVVM заохочує чітке розділення обов’язків у командах. Ті, хто займається дизайном та розробкою інтерфейсу, можуть працювати

незалежно від фахівців, що відповідають за бізнес-логіку, оскільки View і ViewModel мають чітку межу розмежування завдань. Це прискорює створення продукту та робить проєкти легше масштабованими й підтримуваними в майбутньому.

Отже, Model-View-ViewModel виступає надійним архітектурним патерном для побудови модульних, зручних у підтримці застосунків. Він чітко розмежовує логіку роботи з даними, бізнес-логіку та компонент відображення, дозволяючи створювати більш гнучкі та надійні рішення, особливо в контексті сучасних інструментів і підходів Android [11, 12].

Hilt: Сучасний інструмент для ін’єкції залежностей у Android.

Hilt є високорівневим фреймворком для ін’єкції залежностей (Dependency Injection) у Android, який є частиною бібліотеки Android Jetpack. Він побудований на основі Dagger 2, забезпечуючи спрощену реалізацію DI у проєктах із фокусом на зручність використання та автоматизацію процесів. Hilt усуває складність ручного керування залежностями та пропонує стандартизований підхід до їхньої інтеграції в Android-застосунки [8].

Основою роботи Hilt є декларативний підхід до визначення залежностей через анотації, що дозволяє розробникам чітко визначати залежності на рівні класів. Наприклад, використання анотацій @Inject та @HiltAndroidApp дозволяє автоматизувати створення об’єктів і їхній життєвий цикл відповідно до потреб додатку. Ця автоматизація забезпечується через граф залежностей, що створюється на основі визначених модулів (@Module) та анотацій @Provides чи @Binds.

Hilt інтегрується з ключовими компонентами Android, такими як Application, Activity, Fragment, ViewModel та Service, що дозволяє використовувати ін’єкцію залежностей у різних частинах програми без складних налаштувань. Наприклад, анотація @AndroidEntryPoint використовується для оголошення компонентів, які отримують залежності

через Hilt, а @ViewModelInject (у сучасних версіях @HiltViewModel) дозволяє зручно інтегрувати ViewModel у проєкт.

Hilt також забезпечує підтримку життєвого циклу залежностей завдяки використанню контейнерів для кожного з компонентів Android. Це гарантує, що об’єкти створюються та знищуються відповідно до їхнього контексту, що значно покращує ефективність управління ресурсами. Наприклад, залежності, визначені в контексті Activity, автоматично знищуються після завершення її життєвого циклу.

Серед важливих переваг Hilt можна виділити його інтеграцію з іншими бібліотеками Android Jetpack. Наприклад, у поєднанні з Navigation Component, Hilt забезпечує передачу параметрів між фрагментами через ViewModel, спрощуючи управління станом даних. Аналогічно, у зв’язці з WorkManager Hilt дозволяє легко впроваджувати залежності у завдання фонової обробки.

Hilt також включає підтримку для модульного програмування, що дозволяє чітко розмежовувати залежності між різними частинами проєкту. Це особливо корисно для великих команд, які працюють над складними додатками, де необхідна ізоляція компонентів і уникнення конфліктів між залежностями [8].

Таким чином, Hilt є сучасним інструментом, який не лише спрощує процес розробки Android-застосунків, але й забезпечує їхню масштабованість, підтримуваність і високу продуктивність. Його інтеграція з Android Jetpack та інші переваги роблять Hilt важливим компонентом екосистеми сучасної розробки програмного забезпечення для Android.

Parcelize.

Parcelize є анотацією, запровадженою в середовищі Kotlin з метою спрощення процесу впровадження інтерфейсу Parcelable у розробці Android-застосунків. У традиційній практиці створення мобільного програмного забезпечення розробники вимушені вручну реалізовувати методи

writeToParcel() і createFromParcel(), що суттєво ускладнює підтримку коду та збільшує ризик появи помилок. Використання анотації Parcelize мінімізує ці недоліки завдяки автоматичному генеруванню допоміжного коду на етапі компіляції, що робить процес серіалізації об’єктів більш гнучким, ефективним і захищеним від типових помилок.

Функціонально Parcelize інтегрується з інструментами Kotlin та Android Studio, що уможливлює взаємодію зі складними структурами даних без написання зайвого шаблонного коду. Важливо, що анотація Parcelize не лише забезпечує коректну серіалізацію об’єктів різної складності (у тому числі колекцій або вкладених об’єктів), а й підтримує відстеження типобезпеки на рівні компіляції. Це суттєво підвищує рівень надійності застосунків і допомагає уникнути прикрих відмов системи у зв’язку з некоректним перетворенням типів.

З огляду на загальні тенденції до зниження складності та збільшення швидкості розробки, Parcelize можна розглядати як одну з ключових технологій у контексті оптимізації процесу створення Android-застосунків. Її використання сприяє зменшенню витрат часу та зусиль команди, надаючи при цьому більш структурований та підтримуваний у подальшому код. Такий підхід відповідає сучасним вимогам до гнучкості й надійності програмних рішень, дозволяючи розробникам зосередитися на проєктуванні логіки високого рівня та створенні інноваційних функціональних можливостей.

Середовище розробки Android Studio.

Android Studio є офіційним інтегрованим середовищем розробки (IDE) для створення мобільних застосунків на платформі Android, розробленим компанією Google [11]. Цей інструмент вирізняється високою ефективністю та надає повний набір можливостей для розробки на всіх етапах - від написання коду до налагодження, тестування й розгортання застосунку. Android Studio збудована на основі IntelliJ IDEA й об’єднує у собі широкий спектр

інструментів та функцій, спеціально створених для Android-розробників.. (Рисунок 3.4).

Рисунок 3.4. Логотип Android Studio

Однією з визначних переваг Android Studio є її глибока інтеграція з Android SDK (Software Development Kit). Завдяки цьому розробники можуть миттєво отримувати доступ до актуальних API, бібліотек та інструментів, необхідних для створення сучасних застосунків. Це середовище спрощує налаштування проєктів під різні версії Android і дає можливість працювати з новітніми функціями платформи. Завдяки вбудованому менеджеру SDK, Android Studio дає змогу швидко інсталювати й оновлювати потрібні компоненти, що дозволяє розробляти та перевіряти застосунки на різноманітних пристроях і емуляторах.

Android Studio підтримує мову програмування Kotlin, яку Google офіційно рекомендує для розробки Android-застосунків, а також Java й C++. Щільна інтеграція з Kotlin забезпечує зручні засоби для написання коду, серед яких автоматичне доповнення, рефакторинг та опція перетворення Java-коду в Kotlin. Це істотно полегшує перехід на Kotlin для тих, хто раніше використовував Java.

Система збірки Android Studio базується на Gradle, що дає змогу гнучко організувати й автоматизувати процес складання застосунків. Gradle дозволяє керувати залежностями, налаштовувати конфігурації для різних сценаріїв (наприклад, дебаг або реліз), а також підбирати оптимальне оточення для

роботи з різними версіями Android чи специфічними функціями. Завдяки цьому стає легше керувати масштабними проєктами, зокрема створювати й підтримувати модульну архітектуру.

У комплекті Android Studio є потужний візуальний редактор макетів, який підтримує технологію drag-and-drop. Він надає можливість швидко створювати інтерфейси користувача й застосовувати ConstraintLayout, що дає змогу будувати гнучкі й адаптивні дизайни для різних діагоналей екранів та орієнтацій. Крім того, цей редактор дає змогу переглядати вигляд застосунку на різних пристроях і в різних режимах без необхідності тестувати його на реальних пристроях.

Окремою перевагою Android Studio є вбудований емулятор Android. Він дозволяє розробникам тестувати свої застосунки на віртуальних пристроях із різноманітними апаратними та програмними характеристиками. Завдяки підтримці GPS, камери, сенсорів, а також симуляції дзвінків і повідомлень, емулятор створює повноцінне середовище для відтворення широкого спектра реальних ситуацій використання.

Android Studio пропонує й розвинені інструменти для налагодження та профілювання. Завдяки їм розробники можуть виявляти й виправляти помилки на різних етапах створення застосунку, а також аналізувати його продуктивність і відстежувати споживання ресурсів: процесора, пам’яті чи батареї. Зокрема, вбудований профайлер продуктивності надає дані в реальному часі, що дає змогу оперативно оптимізувати застосунки та покращувати їхню ефективність і енергоощадність.

Таким чином, Android Studio слугує одним із головних інструментів для Android-розробників, надаючи все необхідне для ефективної роботи над сучасними мобільними застосунками. Завдяки комплексній інтеграції з Android-платформою, підтримці Kotlin і Java, а також різноманітним інструментам управління збірками та ресурсами, це середовище забезпечує високий рівень продуктивності й зручності протягом усього циклу розробки

[11].

3.2. Опис методології створення програмного забезпечення та побудова блок схеми застосунку

Для розробки ПЗ нашого застосунку було обрано спіральну модель. Спіральна модель розробки програмного забезпечення є ітеративною методологією, яка поєднує особливості водоспадної моделі (Waterfall Model) та прототипування. Вона дозволяє адаптивно реагувати на зміни вимог і поступово вдосконалювати продукт. Модель складається з чотирьох основних фаз: аналізу вимог і планування, проєктування, реалізації та тестування, а також оцінки результатів і ризиків. Кожен цикл спіралі завершується створенням робочої версії програмного продукту або його компонентів [18].

Спіральна модель є доцільною для розробки навчального застосунку для тестування знань, оскільки передбачає поступове додавання функціональності, що дозволяє швидко адаптувати систему до змінних потреб користувачів і освітнього середовища [18].

Рисунок. 3.5. Модель розробки застосунку для тестування

Опишемо більш докладно етапи розробки навчального застосунку за спіральною моделлю.

Перша ітерація. Розробка базової функціональності

На початковому етапі визначаються ключові вимоги до системи, серед яких основними є створення базового інтерфейсу, функціоналу вибору категорій та навігації між екранами.

1. Аналіз вимог і планування. Було визначено, що застосунок має забезпечувати інтерактивний інтерфейс із можливістю вибору категорій тестування. Ризики включали можливість некоректної роботи навігації між екранами.
2. Проєктування. Було спроєктовано базову архітектуру, що включає класи MainActivity, MenuFragment і QuizFragment. Основний акцент зроблено на забезпеченні гнучкості для подальшого розширення функціоналу.

3. Реалізація:
    - Реалізовано навігацію між екранами.
    - Створено прототип функції вибору категорій.

```
class MenuFragment : Fragment(), OnClickAdapter {
    override fun callMethod(category: ModelQuiz) {
        findNavController().navigate(
            R.id.action_menuFragment_to_quizFragment,
            bundleOf(«category» to category)
        )
    }
}
```

4. Оцінка результатів і ризиків. Було проведено тестування прототипу, яке підтвердило коректність навігації та простоту інтерфейсу.

Друга ітерація. Інтеграція запитань і відповідей.

На цьому етапі було додано функцію відображення запитань для вікторин і перевірки відповідей користувачів.

1. Аналіз вимог і планування. Визначено, що система має динамічно оновлювати інтерфейс під час виконання тестування. Ризики включали можливість затримок при завантаженні запитань.
2. Проєктування. Спроєктовано модель даних для запитань (ModelQuizQuestion) і ViewModel (ViewModelQuiz) для управління станом тестування.
3. Реалізація. Реалізовано методи setQuestion і onClick у QuizFragment для відображення запитань і обробки відповідей.

```
private fun setQuestion(modelQuizQuestion: ModelQuizQuestion) {
    binding.questionText.text = modelQuizQuestion.question
    binding.questionAnswer1.text = modelQuizQuestion.listAnswers[0]
    binding.questionAnswer2.text = modelQuizQuestion.listAnswers[1]
```

```
    binding.questionAnswer3.text = modelQuizQuestion.listAnswers[2]
}
```

4. Оцінка результатів і ризиків. Система продемонструвала коректну роботу навіть при великій кількості запитань, що підтвердило її ефективність.

Третя ітерація. Додавання результатів.

Метою цієї ітерації було створення функції для відображення результатів тестування.

1. Аналіз вимог і планування. Було вирішено додати екран результатів (ResultDialogFragment), який демонструватиме кількість правильних відповідей. Ризики включали можливість некоректної передачі даних між компонентами.
2. Проєктування. Розроблено інтерактивний інтерфейс для діалогового вікна з результатами.
3. Реалізація:
    - Реалізовано передачу результатів із QuizFragment до ResultDialogFragment.

```
findNavController().navigate(
    R.id.action_quizFragment_to_resultDialogFragment,
    bundleOf(
        «data-result» to guessedQuestion,
        «category» to arguments?.getParcelable<ModelQuiz>(«category»)
    )
)
```

4. Оцінка результатів і ризиків. Тестування підтвердило коректність передачі результатів і відображення інтерфейсу.

Четверта ітерація. Розширення функціональності.

На останньому етапі було реалізовано масштабованість системи та

адаптацію до різних пристроїв.

1. Аналіз вимог і планування. Визначено потребу в оптимізації інтерфейсу для мобільних пристроїв і додавання нових категорій.
2. Проєктування. Спроєктовано механізм динамічного завантаження категорій через репозиторій.
3. Реалізація. Було оновлено клас DefaultRepositoryImp для підтримки нових джерел даних.

Застосування спіральної моделі для розробки навчального застосунку дозволило поступово впроваджувати функціональність із детальним аналізом ризиків і тестуванням на кожному етапі. Такий підхід забезпечив модульність, адаптивність та масштабованість системи, що відповідає вимогам сучасного освітнього програмного забезпечення.

Блок-схема (див. рисунок 3.6) відображає архітектуру навчального застосунку для проведення вікторин, розробленого на основі фрагментарної структури з використанням сучасних підходів до ін’єкції залежностей та управління станом даних.

У верхній частині блок-схеми розташований компонент App, який відповідає за ініціалізацію бібліотеки Dagger Hilt. Цей компонент забезпечує налаштування залежностей, необхідних для роботи інших елементів системи.

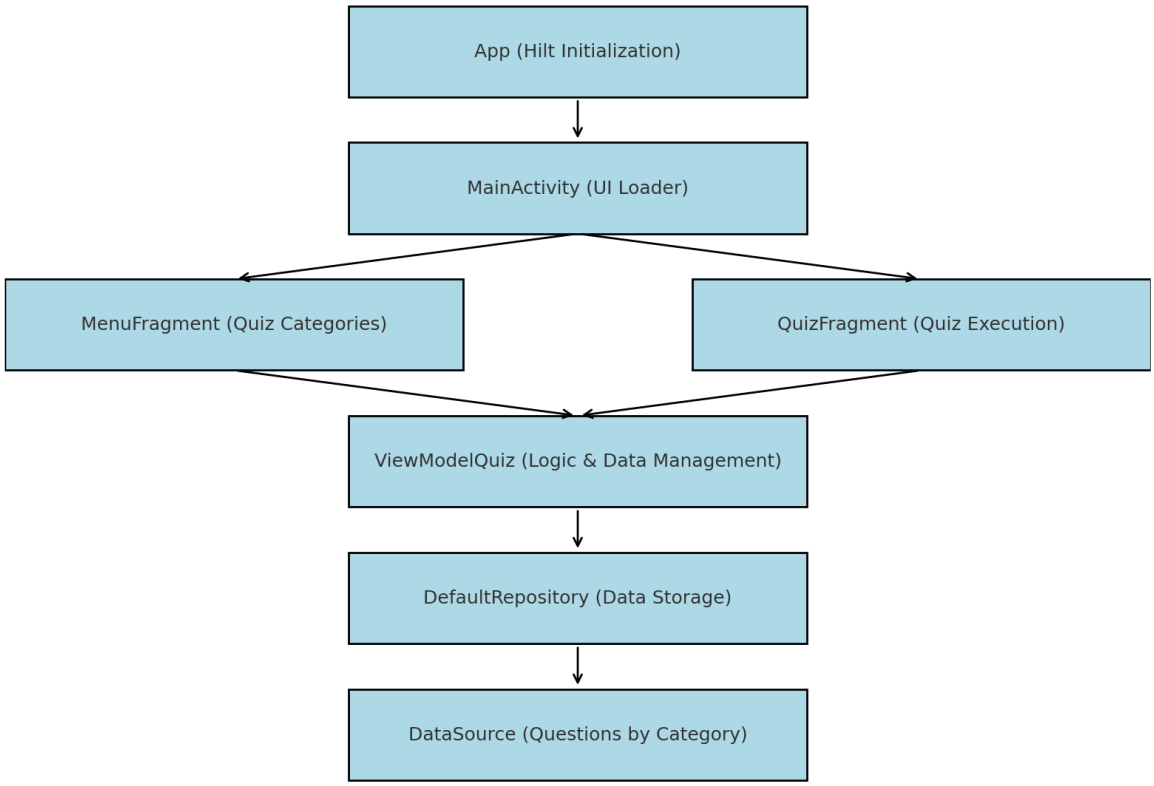

Рисунок 3. 6. Блок схема ПЗ розробленого андроїд застосунку

Другим ключовим елементом є MainActivity, який виступає в якості основного контейнера для користувацького інтерфейсу. Він завантажує відповідні фрагменти залежно від стану програми та забезпечує взаємодію користувача із системою.

MenuFragment і QuizFragment є основними представленнями інтерфейсу. MenuFragment відповідає за відображення списку категорій вікторин, надаючи користувачу можливість вибору певної тематики. QuizFragment виконує функцію проведення вікторини, відображаючи запитання, варіанти відповідей і поточний прогрес тестування.

Центральним елементом, що забезпечує логіку роботи додатку, є ViewModelQuiz. Цей компонент реалізує управління станом даних та взаємодіє з репозиторієм для отримання необхідної інформації. Використання ViewModel дозволяє зберігати дані під час змін конфігурації та зменшувати залежність інтерфейсу від джерел даних.

Джерелом даних для системи є DefaultRepository, який абстрагує доступ до бази даних із запитаннями. Репозиторій реалізує інтерфейс для отримання списків запитань відповідно до обраної категорії, забезпечуючи легкість масштабування та підтримки.

У нижній частині схеми розташовано DataSource, що представляє собою джерело збереження тестових запитань для різних категорій. Цей компонент відповідає за зберігання, організацію та надання даних для роботи репозиторію.

Блок-схема демонструє модульність системи, яка досягається завдяки чіткому розподілу відповідальності між компонентами, використанню архітектурного патерну MVVM та ін’єкції залежностей. Це дозволяє забезпечити простоту розширення функціоналу, тестування окремих модулів та адаптації системи до нових вимог.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Розробка алгоритму роботи андроїд застосунку для тестування

Алгоритм роботи розробленого навчального застосунку для тестування знань побудовано на основі сучасних принципів проєктування програмного забезпечення, таких як ін’єкція залежностей, патерн MVVM (Model-View-ViewModel) та модульний підхід. Додаток забезпечує інтуїтивно зрозумілу взаємодію користувача з системою, автоматизацію тестування знань і адаптивність до обраних категорій тестування. Нижче представлено основні етапи роботи застосунку.

1. Ініціалізація додатку.

На початковому етапі роботи застосунку виконується ініціалізація залежностей за допомогою бібліотеки Dagger Hilt, що здійснюється в класі App. Цей клас виконує функцію глобального налаштування ін’єкції залежностей, забезпечуючи автоматичне створення об’єктів для подальшого використання в інших компонентах.

```
@HiltAndroidApp
class App : Application()
```

Даний етап є фундаментальним для забезпечення узгодженості роботи компонентів програми.

2. Завантаження головного інтерфейсу.

Головний екран додатку представлений класом MainActivity, який слугує контейнером для фрагментів, відповідальних за взаємодію користувача з системою. Основна ініціалізація інтерфейсу відбувається у методі onCreate(Bundle).

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

MainActivity забезпечує управління навігацією між фрагментами, що відображають категорії тестування та сам процес виконання тестів.

3. Відображення списку категорій.

Фрагмент MenuFragment відповідає за генерацію та відображення списку категорій тестування. Список формується через взаємодію з репозиторієм даних, що передається за допомогою адаптера RecyclerAdapterQuiz. Після вибору категорії викликається метод callMethod(ModelQuiz), який передає обрану категорію у наступний фрагмент.

```
@AndroidEntryPoint
class MenuFragment : Fragment(), OnClickAdapter {
    override fun callMethod(category: ModelQuiz) {
        findNavController().navigate(
            R.id.action_menuFragment_to_quizFragment,
            bundleOf(«category» to category)
        )
    }
}
```

Цей етап забезпечує динамічну інтерактивність інтерфейсу для користувача.

4. Проведення тестування.

Фрагмент QuizFragment реалізує логіку проведення тестування. При завантаженні фрагмента виконується ініціалізація списку запитань через ViewModel, яка здійснює взаємодію з репозиторієм даних.

```
viewModelQuiz.quizQuestion.observe(viewLifecycleOwner) { listQuestion ->
    viewModelQuiz.currentQuestion.observe(viewLifecycleOwner) { question -
```

```
>
            setQuestion(listQuestion[question])
        }
    }
```

Метод setQuestion(ModelQuizQuestion) динамічно оновлює інтерфейс відповідно до поточного запитання та можливих варіантів відповідей.

5. Обробка відповіді користувача.

Після вибору користувачем варіанту відповіді викликається метод onClick, який перевіряє правильність відповіді та оновлює відповідні дані у ViewModel.

```
override fun onClick(p0: View?) {
    val button = p0 as Button
    if (button.text == currentQuiz?.rightAnswer) guessedQuestion++
    viewModelQuiz.nextQuestion(viewModelQuiz.currentQuestion.value!! + 1)
}
```

Обробка відповіді включає підрахунок кількості правильних відповідей (guessedQuestion) та перехід до наступного запитання через метод nextQuestion.

6. Завершення тестування.

Після завершення тестування (досягнення кінця списку запитань) виконується навігація до фрагмента ResultDialogFragment, де відображаються результати тесту.

```
findNavController().navigate(
    R.id.action_quizFragment_to_resultDialogFragment,
    bundleOf(
        «data-result» to guessedQuestion,
        «category» to arguments?.getParcelable<ModelQuiz>(«category»)
    )
)
```

Дані про кількість правильних відповідей передаються як аргумент до

результативного діалогового вікна.

7. Відображення результатів.

Результати тестування відображаються у ResultDialogFragment, де інформація про кількість правильних відповідей надається у текстовій формі.

binding.resultPoints.text = arguments?.getInt(«data-result»).toString()

Цей етап завершує цикл взаємодії користувача із системою, забезпечуючи прозорий зворотний зв’язок.

Загальна структура ПЗ розробленого андроїд застосунку:

1. Ініціалізація - налаштування ін’єкції залежностей через Dagger Hilt.
2. Завантаження інтерфейсу - MainActivity керує фрагментами.
3. Меню категорій: MenuFragment надає доступ до тестів.
4. Тестування - QuizFragment реалізує логіку запитань.
5. Обробка відповідей - ViewModel оновлює стан даних.
6. Результати - ResultDialogFragment виводить результат тесту.

Алгоритм розробленого застосунку демонструє модульний підхід із використанням передових технологій, що сприяє масштабованості та простоті його подальшого розвитку. Застосування архітектурного патерну MVVM забезпечує чіткий розподіл обов’язків між компонентами, що підвищує зручність підтримки системи та ефективність її роботи.

4.2. Опис та побудова діаграми класів андроїд застосунку для тестування

Діаграма класів, що розроблена для навчального застосунку(див. рисунок 4.1.), відображає основні компоненти системи, їхні атрибути, методи та зв’язки між класами. Вона побудована відповідно до архітектурного патерну MVVM,

який забезпечує розділення логіки, управління даними та представлення користувацького інтерфейсу.

Ключовим класом є App, який відповідає за ініціалізацію інструменту ін’єкції залежностей Dagger Hilt. Цей клас виконує базове налаштування додатку, забезпечуючи взаємодію між іншими компонентами.

MainActivity є основним контейнером для користувацького інтерфейсу. У класі визначений метод onCreate(Bundle), який завантажує початковий екран та забезпечує взаємодію з фрагментами MenuFragment і QuizFragment.

MenuFragment представляє список категорій вікторин, надаючи користувачу можливість вибору теми. У цьому класі реалізовані методи onCreateView() та onViewCreated() для ініціалізації інтерфейсу. Метод callMethod(ModelQuiz) обробляє вибір категорії користувачем.

QuizFragment відповідає за проведення вікторини. Цей клас містить атрибути currentQuiz, guessedQuestion і counterQuestion, які використовуються для управління станом тестування. Реалізовані методи setQuestion(ModelQuizQuestion), onClick(View) і setReaction(Button) забезпечують динамічну зміну інтерфейсу під час проходження вікторини.

Додатково реалізований клас ResultDialogFragment, який відображає результати тестування у вигляді діалогового вікна. Його методи onCreateView() та onViewCreated() забезпечують відповідне відображення результатів.

Клас ViewModelQuiz виконує роль управління станом даних. Його атрибути quizQuestion і currentQuestion є об’єктами LiveData, що дозволяє відслідковувати зміни даних у реальному часі. Методи setNewQuestionList(CategoryQuiz) і nextQuestion(int) забезпечують оновлення списків запитань та перехід між ними.

Система використовує репозиторій для управління даними, представлений інтерфейсом DefaultRepository. У ньому визначені атрибути, які містять списки запитань для різних категорій вікторин. Клас DefaultRepositoryImp реалізує цей інтерфейс, використовуючи залежності

biology, history, programming, math та cinema для доступу до конкретних даних. Метод getQuestions() забезпечує отримання запитань за категоріями.

Для опису категорій вікторин і запитань у системі використовуються моделі ModelQuiz та ModelQuizQuestion. Клас ModelQuiz містить атрибути title, img і count, що описують категорію. Клас ModelQuizQuestion зберігає інформацію про запитання, варіанти відповідей та правильну відповідь.

Клас ModuleRepository використовується для ін’єкції залежностей через Hilt. Метод provideUserRepository() забезпечує створення та передавання екземпляру репозиторію в систему.

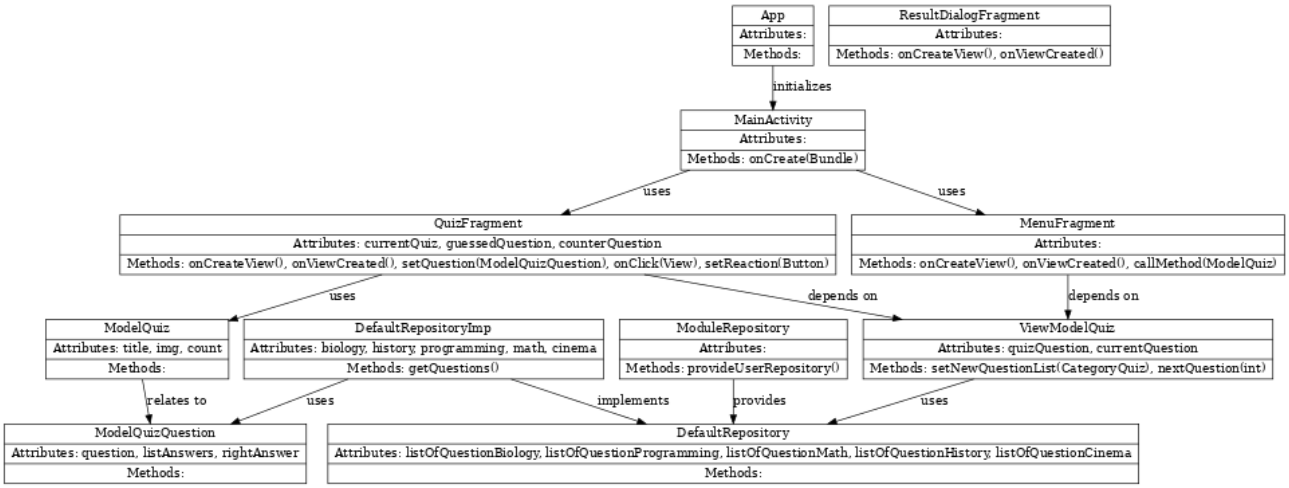

Рисунок 4.1. Діаграма класів ПЗ розробленого застосунку для тестування

Зв’язки між класами включають:

- Використання класу MenuFragment та QuizFragment у MainActivity.
- Залежність MenuFragment і QuizFragment від ViewModelQuiz.
- Взаємодію ViewModelQuiz з DefaultRepository, що реалізується через DefaultRepositoryImp.
- Відношення між ModelQuiz і ModelQuizQuestion, що описують структуру даних.

Представлена діаграма класів демонструє модульність системи, її адаптивність і масштабованість, що робить її зручною для подальшого розвитку та інтеграції в освітні процеси.

4.3. Інтерфейс користувача та функціональні можливості застосунку

Розроблений нами андроїд застосунок є сучасним мобільним інструментом для інтерактивного тестування знань у різних категоріях. Застосунок реалізований на платформі Android з використанням передових технологій, таких як Android Jetpack, Hilt, LiveData та Navigation Component, що забезпечує його масштабованість, ефективність і зручність використання.

Основна мета застосунку - надання користувачам можливості перевіряти свої знання у вибраних категоріях, таких як математика, біологія, історія, програмування та кіно. Інтерфейс програми інтуїтивно зрозумілий: користувач обирає категорію у головному меню, після чого система пропонує серію запитань із варіантами відповідей. Завдяки інтеграції ViewModel і LiveData, тестування відбувається динамічно, із постійним оновленням екрану залежно від прогресу користувача. Після завершення тесту користувач отримує діалогове вікно з підсумковими результатами, яке містить кількість правильних відповідей та надає опцію повторного проходження тесту або повернення до меню.

Однією з ключових особливостей застосунку є використання англійської мови для формулювання запитань і відповідей. Це має значну перевагу для користувачів, оскільки сприяє не лише перевірці їхніх знань у вибраній тематиці, але й удосконаленню володіння англійською мовою. Такий підхід є особливо актуальним у сучасному глобалізованому середовищі, де знання англійської мови відкриває широкі можливості для професійного та особистого розвитку.

З технічної точки зору застосунок реалізовано з використанням архітектури MVVM (Model-View-ViewModel), що дозволяє відокремити бізнес-логіку від користувацького інтерфейсу. Репозиторій даних централізовано зберігає запитання для кожної категорії, забезпечуючи легкість масштабування:

система дозволяє швидко додавати нові категорії або оновлювати запитання без порушення існуючої архітектури. Ін’єкція залежностей через Hilt спрощує підтримку коду та підвищує стабільність роботи системи, що є особливо важливим для проєктів з перспективою довгострокового розвитку.

Застосунок також орієнтований на покращення користувацького досвіду. Адаптивний дизайн інтерфейсу забезпечує коректну роботу програми на пристроях з різними розмірами екранів. Крім того, візуальні елементи, такі як іконки категорій та індикатори правильності відповідей, створюють привабливий і зручний для користувача інтерфейс.

Таким чином, розроблений навчальний застосунок є ефективним інструментом для тестування знань і розвитку англійської мови. Завдяки своїй гнучкій архітектурі, інтерактивності та адаптивності він може використовуватись у широкому спектрі освітніх середовищ, задовольняючи потреби сучасних користувачів.

На рисунку відображено список категорій з певною кількістю питань або завдань у кожній категорії. Кожна категорія має іконку, яка відповідає її тематиці, та цифрове значення поруч, яке вказує на кількість питань (Рисунок 4.2).

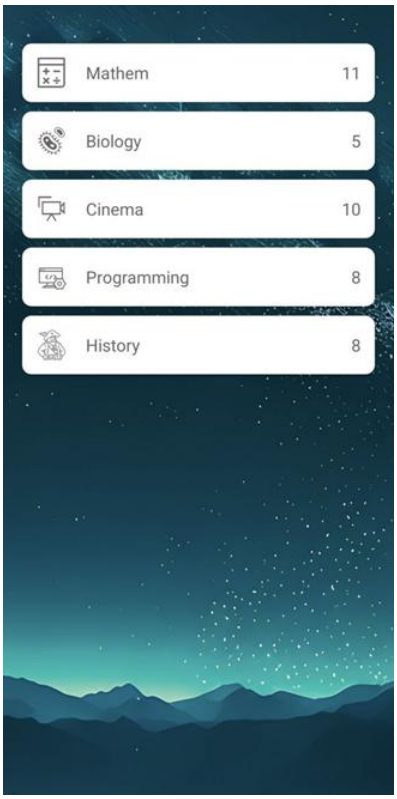

Рис. 4.2. Основний екран застосунку

На цьому скріншоті ми бачимо екран вікторини з питанням із математики. На верхній частині екрана вказано номер питання («Питання 1 з 11»). Текст питання: «A statement which is expressed as an equivalence of two ratios is known as» вказує на те, що користувачу потрібно вибрати правильну відповідь із запропонованих варіантів: «proportion», «ratio» або «variation» (Рисунок 4.3).

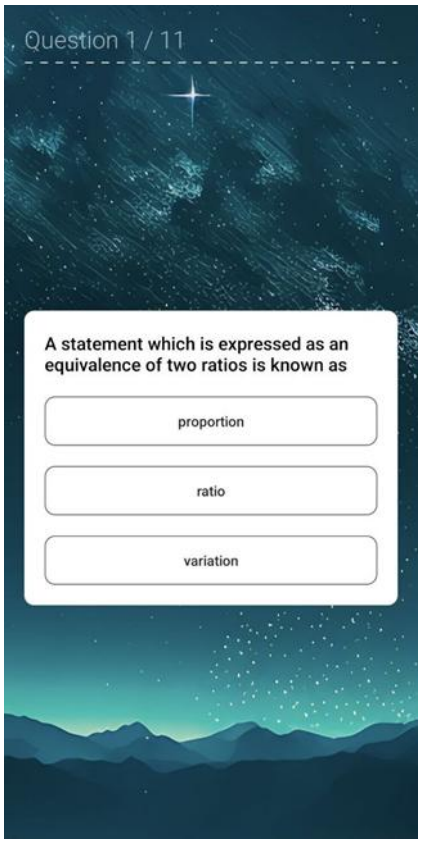

Рисунок 4.3. Приклад питання тестування

Користувач вибирає один із варіантів, натискаючи на відповідну кнопку. Після вибору відповіді користувач має можливість перейти до наступного питання і перевірити правильність вибраної відповіді.

Фонове зображення з зірками на нічному небі додає естетичний вигляд інтерфейсу, що робить досвід користувача більш приємним.

На наступному рисунку 4.4. ми бачимо екран вікторини, на якому зображено друге питання з восьми. Питання стосується програмування або комп’ютерних наук і звучить так: «A set of logically sequenced instructions that allows to find the solution to a problem is:». Це означає, що користувачу потрібно вибрати термін, який описує послідовність інструкцій для вирішення проблеми.

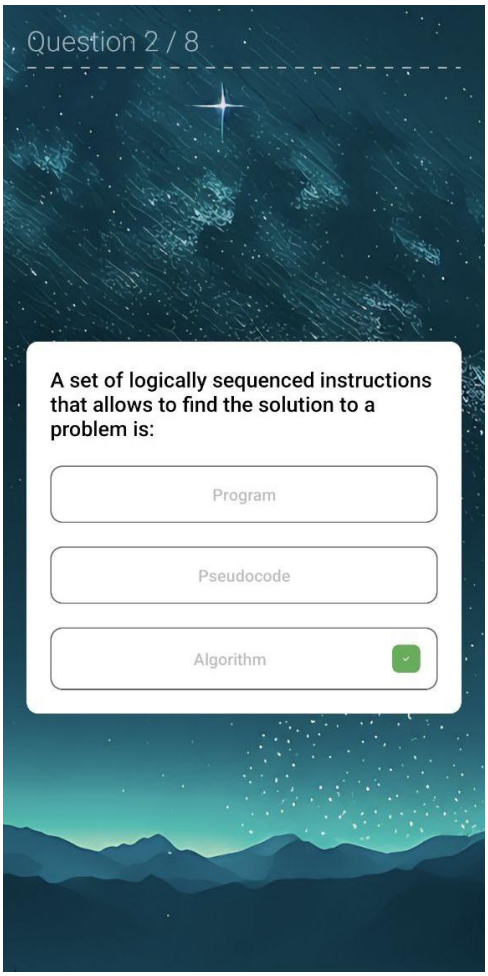

Рисунок 4.4. Приклад питання іншої категорії

З трьох варіантів відповідей («Program», «Pseudocode», «Algorithm») користувач уже обрав «Algorithm», що позначено зеленою галочкою, вказуючи на те, що ця відповідь правильна. Такий інтерфейс показує користувачам, коли вони вибрали правильну відповідь, і може також надавати зворотний зв’язок для неправильних відповідей.

На наступному рисунку ми бачимо третє питання з одинадцяти в контексті вікторини з математики. Питання звучить так: «If 2 ratios a : b and c : d are equal then we can write it as», що в перекладі означає «Якщо два співвідношення a : b і c : d рівні, то ми можемо записати це як». Нижче надано три варіанти відповідей, серед яких користувач вже зробив свій вибір, позначений червоним хрестиком, що вказує на те, що вибрана відповідь є неправильною.

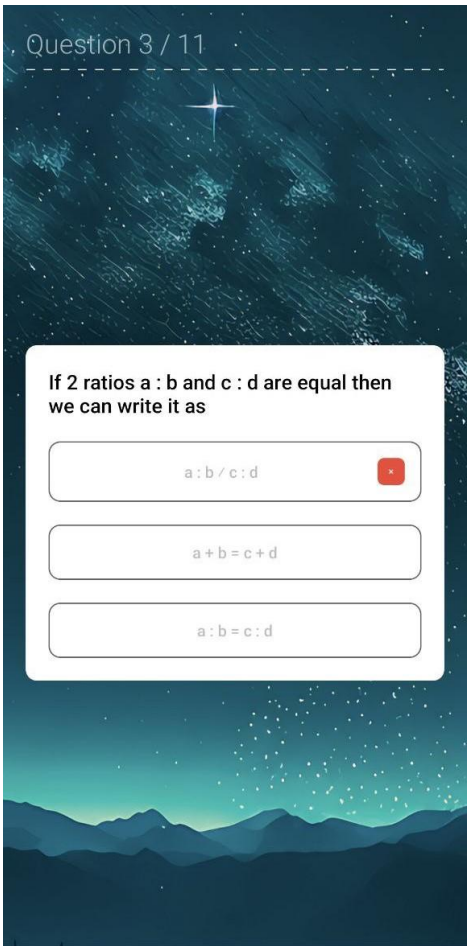

Рисунок 4.5. Приклад питання з математики

На наступному рисунку ми бачимо екран, який відображається після завершення серії питань у вікторині. На екрані вказано «1 point», що означає, що користувач набрав один бал. Під цим показником балів розміщено іконку, яка може символізувати оновлення або перезапуск, що, дає змогу користувачу спробувати пройти вікторину ще раз.

Рисунок 4.6. Результати тестування

Серед функціональних можливостей застосунку реалізовані наступні.

1. Вибір категорій тестування.

Застосунок надає користувачу можливість вибору категорії тестування через інтерактивний список у фрагменті MenuFragment. Для кожної категорії (математика, біологія, кіно, історія, програмування) використовується модель ModelQuiz, яка містить інформацію про назву категорії, її іконку та кількість запитань.

```
private val listQuiz = listOf(
    ModelQuiz(CategoryQuiz.Mathem, R.drawable.ic_math,
repository.listOfQuestionMath.size),
    ModelQuiz(CategoryQuiz.Biology, R.drawable.ic_biology,
repository.listOfQuestionBiology.size),
    ...
)
```

У класі RecyclerAdapterQuiz створено адаптер для відображення списку категорій у RecyclerView, що забезпечує інтерактивність.

2. Проведення тестування.

Основна функціональність реалізована у фрагменті QuizFragment, який відповідає за проведення вікторини. Тестування включає відображення запитань, варіантів відповідей і перевірку правильності відповідей.

```
private fun setQuestion(modelQuizQuestion: ModelQuizQuestion) {
    binding.questionText.text = modelQuizQuestion.question
    binding.questionAnswer1.text = modelQuizQuestion.listAnswers[0]
    binding.questionAnswer2.text = modelQuizQuestion.listAnswers[1]
    binding.questionAnswer3.text = modelQuizQuestion.listAnswers[2]
}
```

- Запитання динамічно оновлюються завдяки використанню ViewModel (ViewModelQuiz) та LiveData.
- Відповіді обробляються через метод onClick, який перевіряє правильність відповіді та оновлює статистику.

3. Збереження стану тестування.

Для забезпечення збереження стану тестування використовуються ViewModel та LiveData. ViewModel зберігає поточний список запитань і номер активного запитання, що дозволяє уникнути втрати даних під час змін конфігурації пристрою.

```
val quizQuestion: LiveData<List<ModelQuizQuestion>> =
mutableQuizQuestion
val currentQuestion: LiveData<Int> = mutableCurrentQuestion
```

4. Обробка відповідей.

Після вибору відповіді користувачем система визначає, чи є вона

правильною. Для цього використовуються методи setReaction і onClick.

```
if (view.text == currentQuiz?.rightAnswer) {
   guessedQuestion++
}
```

Реалізовано візуальний зворотний зв’язок (через іконки правильності) для кожного вибору відповіді.

5. Відображення результатів.

Після завершення тестування користувачу надається діалогове вікно з результатами у ResultDialogFragment. Фрагмент отримує кількість правильних відповідей через передані аргументи.

```
binding.resultPoints.text = arguments?.getInt(«data-result»).toString()
```

Функціональність:

- Відображення кількості правильних відповідей.
- Кнопки для повернення до меню або повторного тестування.

6. Управління запитаннями.

Запитання для кожної категорії зберігаються у репозиторії DefaultRepository, реалізованому класом DefaultRepositoryImp. Цей клас отримує запитання з окремих джерел даних, таких як Biology, Math, Programming тощо.

```
override val listOfQuestionBiology: ArrayList<ModelQuestion>
   get() = biology.questions
```

Репозиторій забезпечує централізований доступ до даних, дозволяючи масштабувати систему шляхом додавання нових категорій або запитань.

7. Ін’єкція залежностей.

Застосунок використовує Hilt для ін’єкції залежностей. Наприклад, залежності, такі як DefaultRepository, автоматично впроваджуються у

фрагменти та адаптери.

```
@Provides
@Singleton
fun provideUserRepository(
    biology: Biology,
    history: History,
    programming: Programming,
    math: Mathem,
    cinema: Cinema
): DefaultRepository {
    return DefaultRepositoryImp(biology, history, programming, math, cinema)
}
```

ВИСНОВКИ

Актуальність розробки програмного забезпечення для Android затстосунку, призначеного для тестування у різних категоріях, зумовлена зростаючими вимогами до мобільних застосунків, що забезпечують швидкий і зручний доступ до навчальних ресурсів. Застосунок, написаний англійською мовою, уможливлює залучення широкої аудиторії користувачів, оскільки англійська є міжнародною мовою спілкування, науки та технологій. Крім того, його багатофункціональність дає змогу проводити оцінювання знань у різних галузях, адаптувати формат і структуру завдань, а також надавати миттєвий зворотний зв’язок, що позитивно впливає на ефективність процесу навчання чи професійної підготовки.

Розробка такого програмного забезпечення буде корисною як для здобувачів вищої освіти, які готуються до іспитів так і для професіоналів, що вдосконалюють власні навички та знання у своїй галузі а також для викладачів і тренерів, які прагнуть підвищити результативність навчальних програм. Крім того, процес розробки ПЗ застосунку відкриває перспективи професійного зростання, підвищення рівня компетентності в галузі розробки мобільних додатків та опанування нових інструментів програмування на Android. Такий проєкт є вагомим кроком у формуванні портфоліо й може слугувати підґрунтям для подальших науково-дослідних і прикладних робіт у сфері мобільних технологій.

Отже, у результати виконання дипломного проєкту нами було повністю виконано поставлені завдання дипломної роботи, а саме:

- ✓ визначено ключові вимоги до розробки програмного забезпечення андроїд застосунку для проведення тестування;
- ✓ проведено аналіз переваг та обмежень аналогічних застосунків та інструментів для тестування;
- ✓ проаналізовано та описано підходи та інструменти, використані при

розробці ПЗ застосунку;

- ✓ описано алгоритм, методологію створення та блок схему розробленого андроїд застосунку для тестування;
- ✓ розроблено мобільний інтерфейс для взаємодії користувача з матеріалом та створено докладну інструкцію користувача;
- ✓ представлено діаграму класів, що ілюструє особливості програмного коду розробленого застосунку;
- ✓ розроблено та протестовано андроїд застосунок для тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Черненко О. О. Методичні рекомендації щодо виконання кваліфікаційної роботи студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня магістра / С. В. Гаркуша, О. В. Ольховська, О. О. Черненко. – Полтава : ПУЕТ, 2023. – 68 с. Режим доступу: URL: http://elib.puet.edu.ua/action.php?kt_path_info=ktcore.SecViewPlugin.actions.document&fDocumentId=824868 - Назва з екрану

2. Тарасюк Назар Анатолійович. Кваліфікаційна робота на тему «Розробка програмного забезпечення десктопного навчального застосунку з теми «Рекурсивні алгоритми» дистанційного навчального курсу «Аналіз алгоритмів». Полтава, 2024. Режим доступу: http://dspace.puet.edu.ua/handle/123456789/13498 - Назва з екрану

3. Antonio Leiva.View Binding: The Definitive way to access views on Android. Режим доступу: URL: https://antonioleiva.com/view-binding-android/ - Назва з екрану

4. Developer workflow basics. Режим доступу: URL: https://developer.android.com/studio/workflow - Назва з екрану

5. Волжан, Є. А. Розробка програмного забезпечення тренажеру з теми “Матриці” дистанційного навчального курсу “Прикладна математика”: кваліфікаційна робота ... бакалавра : 122 Комп’ютерні науки / Євген Андрійович Волжан. - Полтава, 2024. - 55 с. Режим доступу: URL: dspace.puet.edu.ua/handle/123456789/13969 - Назва з екрану

6. Сімперович Микита Михайлович. ДИПЛОМНА РОБОТА на тему: «РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТРЕНАЖЕРУ З ТЕМИ «ВЕРИФІКАЦІЯ ПРОГРАМ» АНГЛОМОВНОГО ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «ТЕОРІЯ ПРОГРАМУВАННЯ». Полтава: ПУЕТ, 2022 р.. Режим доступу: URL: dspace.puet.edu.ua/handle/123456789/12601- Назва з екрану

7. Банасюкевич Л. Кваліфікаційна робота на тему «Розробка програмного забезпечення тренажеру з теми «Основні методології програмування» дистанційного навчального курсу «Теорія програмування». Полтава, 2023 р. Режим доступу: URL: http://dspace.puet.edu.ua/handle/123456789/13053 - Назва з екрану

8. Dependency injection with Hilt. Режим доступу: URL: https://developer.android.com/training/dependency-injection/hilt-android - Назва з екрану

9. Kotlin for Android. Режим доступу: URL: https://kotlinlang.org/docs/android-overview.html - Назва з екрану

10. Kotlin home page. Режим доступу: URL: https://kotlinlang.org/ - Назва з екрану

11. Meet Android Studio. Режим доступу: URL: https://developer.android.com/studio/intro - Назва з екрану

12. Model-View-ViewModel (MVVM). Режим доступу: URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm - Назва з екрану

13. MVVM Clean Architecture Pattern in Android with Use Cases. Режим доступу: URL:https://medium.com/@ami0275/mvvm-clean-architecture-pattern-in-android-with-use-cases-eff7edc2ef76 - Назва з екрану

14. Navigation. Режим доступу: URL: https://developer.android.com/guide/navigation - Назва з екрану

15. Задворкін, М. О. Розробка системи тестування знань з курсу «Основи наукових досліджень в інформатиці»: кваліфікаційна робота ... бакалавра : 122 Комп’ютерні науки / Максим Олександрович Задворкін. - Полтава, 2024. - 44 с. Режим доступу: URL: http://dspace.puet.edu.ua/handle/123456789/13972 Назва з екрану

16. Use Navigation actions and Fragments. Режим доступу: URL: https://developer.android.com/guide/navigation/design/actions - Назва з екрану

17. Welcome to our tour of Kotlin. Режим доступу: URL: https://kotlinlang.org/docs/kotlin-tour-welcome.html - Назва з екрану

18. What is Spiral Model in Software Engineering? Режим доступу: URL: https://www.geeksforgeeks.org/software-engineering-spiral-model/- Назва з екрану

19. КОШОВА, О., ЧЕРНЕНКО, О., ОРІХІВСЬКА, О., ТУР, В., & ЯНКО, О. (2024). РОЗРОБКА НАВЧАЛЬНОГО АНДРОЇД-ЗАСТОСУНКУ З ТЕМИ «СОРТУВАННЯ ВСТАВКАМИ» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «АЛГОРИТМИ І СТРУКТУРИ ДАНИХ». *Інформаційні технології та суспільство*, (5 (11), 34-42. https://doi.org/10.32689/maup.it.2023.5.5

ДОДАТОК А
Код програми

```
package gog.quizeog.goeziuq

import android.app.Application
import dagger.hilt.android.HiltAndroidApp

@HiltAndroidApp
class App : Application()



package gog.quizeog.goeziuq

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}



package gog.quizeog.goeziuq.data

import javax.inject.Inject

interface DefaultRepository {

    val listOfQuestionBiology: ArrayList<ModelQuestion>
```

```
    val listOfQuestionProgramming: ArrayList<ModelQuestion>
    val listOfQuestionMath: ArrayList<ModelQuestion>
    val listOfQuestionHistory: ArrayList<ModelQuestion>
    val listOfQuestionCinema: ArrayList<ModelQuestion>

}

package gog.quizeog.goeziuq.data

import gog.quizeog.goeziuq.data.subData.*
import javax.inject.Inject

class DefaultRepositoryImp @Inject constructor(
    private val biology: Biology,
    private val history: History,
    private val programming: Programming,
    private val math: Mathem,
    private val cinema: Cinema
) : DefaultRepository {

    override val listOfQuestionBiology: ArrayList<ModelQuestion>
        get() = biology.questions
    override val listOfQuestionProgramming: ArrayList<ModelQuestion>
        get() = programming.questions
    override val listOfQuestionMath: ArrayList<ModelQuestion>
        get() = math.questions
    override val listOfQuestionHistory: ArrayList<ModelQuestion>
        get() = history.questions
    override val listOfQuestionCinema: ArrayList<ModelQuestion>
        get() = cinema.questions

}

package gog.quizeog.goeziuq.di
```

```
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import gog.quizeog.goeziuq.data.DefaultRepository
import gog.quizeog.goeziuq.data.DefaultRepositoryImp
import gog.quizeog.goeziuq.data.subData.*
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
class ModuleRepository {

    @Provides
    @Singleton
    fun provideUserRepository(
        biology: Biology,
        history: History,
        programming: Programming,
        math: Mathem,
        cinema: Cinema,
    ): DefaultRepository {
        return DefaultRepositoryImp(
            biology,
            history,
            programming,
            math,
            cinema,
        )
    }

}
```

```
package gog.quizeog.goeziuq.domain.recycler

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.LinearLayout
import androidx.core.view.marginBottom
import androidx.core.view.marginLeft
import androidx.core.view.marginRight
import androidx.core.view.marginTop
import androidx.recyclerview.widget.RecyclerView
import gog.quizeog.goeziuq.R
import gog.quizeog.goeziuq.data.DefaultRepository
import gog.quizeog.goeziuq.databinding.ItemQuizBinding
import gog.quizeog.goeziuq.domain.CategoryQuiz
import gog.quizeog.goeziuq.domain.models.ModelQuiz
import javax.inject.Inject


class RecyclerAdapterQuiz @Inject constructor(private val repository: DefaultRepository) :
    RecyclerView.Adapter<RecyclerAdapterQuiz.ViewHolder>() {

    var callBack: OnClickAdapter? = null

    private val listQuiz = listOf(
        ModelQuiz(
            CategoryQuiz.Mathem,
            R.drawable.ic_math,
            repository.listOfQuestionMath.size
        ),
        ModelQuiz(
            CategoryQuiz.Biology,
            R.drawable.ic_biology,
            repository.listOfQuestionBiology.size
```

```
    ),
    ModelQuiz(
        CategoryQuiz.Cinema,
        R.drawable.ic_film,
        repository.listOfQuestionCinema.size
    ),
    ModelQuiz(
        CategoryQuiz.Programming,
        R.drawable.ic_prog,
        repository.listOfQuestionProgramming.size
    ),
    ModelQuiz(
        CategoryQuiz.History,
        R.drawable.ic_history,
        repository.listOfQuestionHistory.size
    ),
)

inner class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
    private val binding = ItemQuizBinding.bind(view)

    fun bindItem(item: ModelQuiz, position: Int) {

        if (position == 0 || position == listQuiz.size) {
            val params = LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.MATCH_PARENT,
                LinearLayout.LayoutParams.WRAP_CONTENT
            )
            params.setMargins(
                binding.root.marginLeft,
                binding.root.marginTop + 100,
                binding.root.marginRight,
                binding.root.marginBottom
            )
```

```
            binding.root.layoutParams = params
        }

        binding.labelCard.text = item.title.toString()
        binding.imgCard.setImageResource(item.img)
        binding.questionCount.text = item.count.toString()

        binding.root.setOnClickListener {
            callBack?.callMethod(item)
        }
    }
}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
    return ViewHolder(
        LayoutInflater
            .from(parent.context)
            .inflate(R.layout.item_quiz, parent, false)
    )
}

override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.bindItem(listQuiz[position], position)
}

override fun getItemCount() = listQuiz.size
}

package gog.quizeog.goeziuq.presenter.ui

import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
```

```
import android.view.ViewGroup
import androidx.core.os.bundleOf
import androidx.fragment.app.Fragment
import androidx.navigation.fragment.findNavController
import dagger.hilt.android.AndroidEntryPoint
import gog.quizeog.goeziuq.R
import gog.quizeog.goeziuq.databinding.FragmentMenuBinding
import gog.quizeog.goeziuq.domain.models.ModelQuiz
import gog.quizeog.goeziuq.domain.recycler.OnClickAdapter
import gog.quizeog.goeziuq.domain.recycler.RecyclerAdapterQuiz
import javax.inject.Inject

@AndroidEntryPoint
class MenuFragment : Fragment(), OnClickAdapter {

    private lateinit var binding: FragmentMenuBinding

    @Inject
    lateinit var adapterQuiz: RecyclerAdapterQuiz

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentMenuBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)



        binding.recyclerQuiz.also {
```

```
            adapterQuiz.callBack = this
            it.adapter = adapterQuiz
        }

    }

    override fun callMethod(category: ModelQuiz) {
        Log.i("myLog", "des = ${findNavController().currentDestination?.label}")
        if (findNavController().currentDestination?.label == "fragment_menu") {
            findNavController().navigate(
                R.id.action_menuFragment_to_quizFragment, bundleOf(
                    "category" to category
                )
            )
        }

    }

}

package gog.quizeog.goeziuq.presenter.ui

import android.annotation.SuppressLint
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import androidx.core.content.ContextCompat
import androidx.core.os.bundleOf
import androidx.fragment.app.Fragment
import androidx.fragment.app.activityViewModels
import androidx.lifecycle.lifecycleScope
```

```
import androidx.navigation.fragment.findNavController
import dagger.hilt.android.AndroidEntryPoint
import gog.quizeog.goeziuq.R
import gog.quizeog.goeziuq.databinding.FragmentQuizBinding
import gog.quizeog.goeziuq.domain.CategoryQuiz
import gog.quizeog.goeziuq.domain.models.ModelQuiz
import gog.quizeog.goeziuq.domain.models.ModelQuizQuestion
import gog.quizeog.goeziuq.presenter.vm.ViewModelQuiz
import kotlinx.coroutines.CoroutineExceptionHandler
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch

@SuppressLint("SetTextI18n")
@AndroidEntryPoint
class QuizFragment : Fragment(), View.OnClickListener {

    private lateinit var binding: FragmentQuizBinding

    private val viewModelQuiz by activityViewModels<ViewModelQuiz>()

    var currentQuiz: ModelQuizQuestion? = null

    var guessedQuestion = 0
    var counterQuestion = 0



    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentQuizBinding.inflate(inflater)
        return binding.root
```

```
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    viewModelQuiz.quizQuestion.observe(viewLifecycleOwner) { listQuestion ->
        viewModelQuiz.currentQuestion.observe(viewLifecycleOwner) { question ->
            binding.questionCounter.text = "Question ${question + 1} / ${listQuestion.size}"
            currentQuiz = listQuestion[question]
            setQuestion(listQuestion[question])

            binding.questionAnswer1.isEnabled = true
            binding.questionAnswer2.isEnabled = true
            binding.questionAnswer3.isEnabled = true
        }
    }

    val args = arguments?.getParcelable<ModelQuiz>("category")

    args?.let {
        viewModelQuiz.setNewQuestionList(it.title)
        counterQuestion = it.count
    }




    binding.questionAnswer1.setOnClickListener(this)
    binding.questionAnswer2.setOnClickListener(this)
    binding.questionAnswer3.setOnClickListener(this)

}

private fun setQuestion(modelQuizQuestion: ModelQuizQuestion) {
    binding.questionText.text = modelQuizQuestion.question
```

```
    binding.questionAnswer1.text = modelQuizQuestion.listAnswers[0]
    binding.questionAnswer2.text = modelQuizQuestion.listAnswers[1]
    binding.questionAnswer3.text = modelQuizQuestion.listAnswers[2]
}

override fun onClick(p0: View?) {
    binding.questionAnswer1.isEnabled = false
    binding.questionAnswer2.isEnabled = false
    binding.questionAnswer3.isEnabled = false

    setReaction(p0 as Button)
}

private fun setReaction(view: Button) {

    lifecycleScope.launch(Dispatchers.Main) {
        view.setCompoundDrawablesWithIntrinsicBounds(
            null,
            null,
            ContextCompat.getDrawable(
                requireContext(),
                if (view.text == currentQuiz?.rightAnswer) {
                    guessedQuestion++
                    R.drawable.ic_true
                } else R.drawable.ic_false
            ),
            null
        )

        delay(500)

        try {
            Log.i("myLog", "counterQuestion = ${counterQuestion}")
            Log.i("myLog", "value + 1 = ${viewModelQuiz.currentQuestion.value!! + 1}")
```

```
            if (viewModelQuiz.currentQuestion.value!! + 1 < counterQuestion) {
                viewModelQuiz.nextQuestion(viewModelQuiz.currentQuestion.value!! + 1)
            } else {
                findNavController().navigate(
                    R.id.action_quizFragment_to_resultDialogFragment,
                    bundleOf(
                        "data-result" to guessedQuestion,
                        "category" to arguments?.getParcelable<ModelQuiz>("category")
                    )
                )
            }
        } catch (e: Exception) {
            Log.i("myLog", e.toString())
        }


        view.setCompoundDrawablesWithIntrinsicBounds(
            null,
            null,
            null,
            null
        )

    }

}

override fun onDestroyView() {
    super.onDestroyView()
    viewModelQuiz.nextQuestion(0)
}

}
```

```
package gog.quizeog.goeziuq.presenter.ui

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.core.os.bundleOf
import androidx.fragment.app.DialogFragment
import androidx.navigation.fragment.findNavController
import dagger.hilt.android.AndroidEntryPoint
import gog.quizeog.goeziuq.R
import gog.quizeog.goeziuq.databinding.FragmentResultDialogBinding
import gog.quizeog.goeziuq.domain.CategoryQuiz
import gog.quizeog.goeziuq.domain.models.ModelQuiz

@AndroidEntryPoint
class ResultDialogFragment : DialogFragment() {

    private lateinit var binding: FragmentResultDialogBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentResultDialogBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val args = arguments?.getInt("data-result").toString()
```

```
        binding.resultPoints.text = args

        binding.restartButton.setOnClickListener {
            findNavController().navigate(
                R.id.action_resultDialogFragment_to_quizFragment, bundleOf(
                    "category" to arguments?.getParcelable<ModelQuiz>("category")
                )
            )
        }

        binding.goMenuButton.setOnClickListener {
            findNavController().navigate(R.id.action_resultDialogFragment_to_menuFragment)
        }


    }
}

package gog.quizeog.goeziuq.presenter.vm

import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import dagger.hilt.android.lifecycle.HiltViewModel
import gog.quizeog.goeziuq.domain.CategoryQuiz
import gog.quizeog.goeziuq.domain.MixQuestion
import gog.quizeog.goeziuq.domain.models.ModelQuizQuestion
import kotlinx.coroutines.CoroutineExceptionHandler
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import javax.inject.Inject
```

```
@HiltViewModel
class ViewModelQuiz @Inject constructor(
    private val mixQuestion: MixQuestion
) : ViewModel() {

    private val mutableQuizQuestion = MutableLiveData<List<ModelQuizQuestion>>()
    val quizQuestion: LiveData<List<ModelQuizQuestion>> = mutableQuizQuestion

    fun setNewQuestionList(category: CategoryQuiz) {
        mutableQuizQuestion.value = (mixQuestion.getQuizQuestionModel(category).shuffled())
    }

    private val mutableCurrentQuestion = MutableLiveData(0)
    val currentQuestion: LiveData<Int> = mutableCurrentQuestion

    fun nextQuestion(num: Int) {
        mutableCurrentQuestion.value = num
    }

}
```