Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
(рівень вищої освіти)

_____Система мобільного тестування знань на платформі Андроїд_____

Виконав: студент 2 курсу, групи СКСм-20-1

_____Марчук С. С._____
(прізвище, ініціали)

Спеціальність *123 Комп'ютерна інженерія*

Тип програми ____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____

*Спеціалізовані комп'ютерні системи*_____
(повна назва освітньої прграми)

Керівник роботи доц. Шкіль О. С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія
(шифр і назва)

Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва

ЗАТВЕРДЖУЮ:
Зав. кафедри ____
(підпис)
«__04__»___11____ 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Марчуку Сергію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи __Система мобільного тестування знань на платформі Андроїд

затверджена наказом по університету від «_04_»__11__2021 р. № _1635 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 25.12.2021

3. Вихідні дані до роботи ___Excel файл з інформаціє
Мова програмування Kotlin
Операційна система Android
Середовище розробки Android Studio

4. Перелік питань, що потрібно опрацювати в роботі
аналіз математичного апарату та специфікацій технологій
дослідження існуючих аналогів на ринку
проектування та реалізація програмної частини проекту
тестування отриманого проекту

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 21 слайд

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

7. Дата видачі завдання 20.10.2021

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	20.10.2021 - 25.10.2021	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	25.10.2021 - 10.11.2021	
3	Дослідження існуючих програмних рішень на ринку	10.11.2021 - 20.11.2021	
4	Дослідження побудови файлів формату Excel	20.11.2021 - 30.11.2021	
5	Розробка програми для ОС Android	30.11.2021 - 10.12.2021	
6	Проведення тестування	10.12.2021 - 15.12.2021	
7	Оформлення пояснювальної записки	15.12.2021 - 20.12.2021	
8	Перевірка виконаного проекту керівником,	20.12.2021 - 23.12.2021	
9	Захист проекту	23.12.2021 - 28.12.2021	

Студент _____
(підпис)

Керівник роботи (проекту) _____ доц. Шкіль О. С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 78 сторінок, 27 рисунків

FIREBASE, ANDROID SDK, MVVM, БАЗА ДАНИХ, RETROFIT, UI, МОДЕЛЬ, РЕСУРСИ, МЕСЕНДЖЕР, KOTLIN, БАГАТОПОТОЧНІСТЬ, RXJAVA, ПЗ SDK UI/UX

Об’єкт дослідження – система мобільного тестування знань на платформі «Android».

Предмет дослідження – розробка мобільного застосунку для проведення тестування знань на платформі «Android».

Мета кваліфікаційної роботи – реалізація комп’ютерної системи для проведення автоматизованих тестувань за допомогою мобільного додатку для операційної системи Android та сервісу «Firebase».

В кваліфікаційній роботі проведено аналіз та розроблено мобільний застосунок для проведення тестувань. Програмна реалізація виконується в середовищі «Intellij Android Studio» мовою програмування «Kotlin» з використанням «AndroidSDK».

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Firebase – комплекс платформ, які підтримуються компанією Google, для розробки мобільних та веб застосунків

MVVM – архітектурний патерн розробки Model View View Model

RxJava (Reactive Extensions for the JVM) – бібліотека для компонування асинхронних та базованих на подіях програм з використанням спостереження послідовностей для віртуальної машини Java

SDK (Software development kit) – набір засобів розробки, який дозволяє фахівцям з програмного забезпечення створювати додатки для певного пакету програм

ОЗП – оперативний запам’ятовуючий пристрій

ОС (OS) – операційна система (operating system)

ЗМІСТ

ВСТУП.....7
1 ПОСТАНОВКА ЗАДАЧІ ТА СФЕРА ВИКОРИСТАННЯ.....10
1.1 Характеристики об’єктів проектування 10
1.2 Вимоги до програмної та апаратної сумісності.....11
1.3 Опис предметної області.....12
1.4 Аналіз існуючих аналогів розробок.....14
2 МЕТОДИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ.....16
2.1 Платформа Android.....16
2.2 Переваги мови програмування Kotlin над Java.....20
2.3 Середовище розробки Android Studio.....23
2.4. Система автоматизованої збірки Gradle.....26
2.5 Порівняння Java та Kotlin як засобів для написання додатків під ОС Android.....30
2.6 Порівняння архітектурних патернів проектування MVC, MVP, MVVM 40
3 ПРОЕКТУВАННЯ ДОСЛІДЖУВАНОЇ СИСТЕМИ.....44
3.1 Огляд моделі предметної області.....44
3.2 Огляд структури вхідних даних.....45
4 ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ МОБІЛЬНОГО ТЕСТУВАННЯ.....49
4.1 Загальна архітектура системи.....49
4.2 Розробка візуального інтерфейсу за допомогою засобів Android SDK. .50
ВИСНОВКИ.....63
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....64
ДОДАТОК А.....66
ДОДАТОК Б.....67

ВСТУП

За останні, приблизно, 30 років у системі освіти розвинених країн відбулися істотні структурні зміни, обумовлені усестороннім впливом науково-технічного прогресу на життєдіяльність суспільства, інформаційною революцією.

Вища освіта повинна бути однаково доступною для всіх на основі здібностей кожного. Оскільки здобуття вищої освіти всіма громадянами за традиційною очною формою практично неможливе (цього не витримають бюджети навіть найбільш благополучних країн), широкий розвиток отримали такі форми освіти, як вечірнє та дистанційне навчання.

Багатовіковий педагогічний досвід вчить, що повноцінну освіту можна отримати лише при наявності хороших навчально-методичних матеріалів, підручників та спілкування, або, як зараз прийнято говорити, діалогу студента і викладача.

При вечірній формі навчання на таке спілкування відведено значно менше часу, ніж при очній формі, а у випадку дистанційного навчання контакт студента з викладачем взагалі зведений до мінімуму. З цієї причини велика увага приділяється самостійній роботі студентів. Закономірно постає питання про те, якими мають бути навчальні матеріали, щоб забезпечити якісний процес навчання і компенсувати обмежене спілкування студента з викладачем, який вносить в процес навчання емоційне забарвлення, реалізує зворотний зв’язок і, при необхідності, може провести корегування процесу навчання студента [1].

Рішенням цієї проблеми стає поступова заміна поліграфічних видань на комп’ютерні засоби дистанційного навчання, до складу яких при сучасному розвитку інформаційних технологій легко впроваджуються елементи такого діалогу.

Невід’ємною складовою будь-якого навчального закладу є процес контролю якості засвоєного навчального матеріалу. На сьогоднішній день застосовується методологія перевірки знань здобувачів освіти за допомогою проведення тестувань. Наразі ця система є найефективнішою і використовується в усьому світі, не зважаючи на такі недоліки, як: присутність елементу випадковості, неможливість перевірки рівню знань зв’язаних з творчістю та недостатність часу для глибокого аналізу теми.

Зважаючи на останні події, а саме епідемію коронавірусу SARS-CoV-2, в освітній процес були введені корегування для запобігання нових випадків інфікування. Більшість навчальних закладів були переведені на дистанційну форму навчання, що, також, вплинуло на процес перевірки знань студентів. На даний момент постає питання забезпечення ефективної перевірки знань в дистанційній формі за допомогою програмних засобів.

Технології дозволяють створювати проекти найрізноманітнішого типу складності та цільової спрямованості. Вони, як і будь-які інші розробки, постійно удосконалюються і розвиваються: додаються нові, замінюються застарілі. Іншими словами йде природний процес еволюції сфери інформаційних технологій.

Незважаючи на те, що розроблена достатня кількість програмних продуктів, які дозволяють автоматизувати процес тестування студентів, багато з них мають недоліки, або зайву функціональність. Розробка нового продукту, орієнтованого на конкретного користувача, є важливим і актуальним завданням.

Беручи до уваги перехід суспільства від настільних стаціонарних комп’ютерів до компактних обчислювальних пристроїв, таких як ноутбуки та смартфони постає необхідність зробити процес тестування максимально зручним та забезпечити можливість проходження тесту навіть перебуваючи у транспортному засобі.

Метою даної кваліфікаційної роботи є аналіз та дослідження існуючих систем орієнтованих на тестування знань студентів. В результаті аналізу, було прийняте рішення розробки системи мобільного тестування на базі

операційної системи Андроїд, оскільки дана мобільна ОС є найпоширенішою серед існуючих аналогів [2].

Підготовка тестових матеріалів відбуватиметься за допомогою текстового редактору «Microsoft Excel». Вхідними даними для забезпечення ідентифікації студентів, які тестуються, слугуватиме файл зі списком груп, ФІБ та електронних адрес.

1 ПОСТАНОВКА ЗАДАЧІ ТА СФЕРА ВИКОРИСТАННЯ

1.1 Характеристики об’єктів проектування

Розробці підлягає система автоматизації тестування знань студентів. Програмно-обчислювальний комплекс призначений для автоматизації тестування знань студентів за темами дисциплін навчального плану. Система повинна функціонувати в глобальній мережі Інтернет.

Завдання проектування полягає в:

- обґрунтуванні структури системи;
- виборі програмних продуктів, на основі яких буде створюватися програмна система, і здійснюватися її підтримка;
- розробці програмної системи та її описі.

Система призначена для автоматизації тестування знань студентів. Програмно-обчислювальний комплекс можна поділити на дві великі підсистеми: "Студент" та "Викладач". Мета даної кваліфікаційної роботи – розробка підсистем студента та викладача.

Права доступу до системи автоматизації тестування знань студентів можуть отримувати «Студент» та «Викладач». «Викладач» виступає в ролі «Адміністратора» та повинен завантажити до серверу сформований файл з тестовими питаннями та варіантами відповідей, а також файл з переліком студентів, які будуть проходити тестування. Кожен користувач, який є в списку, що завантажується до серверу, має можливість авторизуватись в системі після підтвердження особистості за допомогою електронної адреси з доменом «nure.ua».

Викладач розміщує тести відповідно до навчального плану і відкриває доступ до них студентів. Студенти мають право на проходження пробного тесту для ознайомлення з функціями системи. Адміністратор має доступ до всіх частин системи і має можливість змінювати, видаляти або створювати тести, проводити перевірку і підтвердження реєстрації студентів.

Для ідентифікації користувачів відбувається обробка даних з файлу, який вивантажується викладачем до серверної частини системи. Після успішного завантаження файлу студент вводить адресу електронної пошти, та отримує на електронну пошту код для підтвердження особистості [3].

Система повинна надавати користувачеві наступні можливості:

- проходження пробного тестування для ознайомлення роботи з системою;
- вибір дисципліни і теми, по якій планується проходження тесту;
- ознайомлення з теоретичним блоком по даній темі;
- проходження тесту при наявності прав, які надаються викладачем;
- отримання звітів про результати пройдених тестів;

Система повинна відповідати певним вимогам:

- безпека - доступ до функцій системи повинен бути відкритий виключно авторизованим користувачам;
- віддаленість - доступ до можливостей системи повинен здійснюватися віддалено, за допомогою мережі інтернет;
- надійність - необхідно дбати про безпеку і доступність даних про проходження студентами тестів.

Керівництво по використанню системи має ділитися на інструкції для викладачів і студентів.

Інструкції для студентів повинні містити роз'яснення щодо авторизації в системі і опис роботи в ній.

1.2 Вимоги до програмної та апаратної сумісності

Для проектування системи персональний комп’ютер повинен відповідати наступним характеристикам:

- операційна система: «Microsoft Windows 7/8/10»;
- ЦП новіше «Intel Core» 2-го покоління, або AMD з підтримкою «Windows Hypervisor»
- не менше 8 ГБ ОЗП (також 1 ГБ для віртуального пристрою);
- не менше 8 ГБ доступного місця на жорсткому диску (також 1.5 ГБ для віртуального пристрою);
- стандартну клавіатуру та мишу;
- 1200 x 800 розширення екрану;
- встановлене середовище розробки ПЗ «Android Studio»;

Користувач повинен мати смартфон на базі з операційною системою «Андроїд» не нижче «Lollipop 5.0».

1.3 Опис предметної області

Одним із напрямів удосконалення процесу навчання є розробка оперативної системи контролю знань, умінь і навичок, що дозволяє об’єктивно оцінювати знання учнів.

На даний момент існує велика кількість різноманітних способів проведення контролю і оцінки знань. Найбільш широко застосовується тестування, як один з методів контролю засвоєння учнями знань, з дисципліни, що має низку певних переваг перед традиційними методами контролю знань (контрольна робота, усна відповідь і т. д.). Інструментом для вимірювання за шкалою досягнень учня є правильно сконструйований тест, який відповідає не тільки предмету навчання, але і його завданням і служить розвитку системного підходу до вивчення навчальної дисципліни.

Тестування в освітніх установах використовується, як засіб об’єктивного контролю знань навчальної програми в учнів. Систематичний контроль рівня знань учнів з одного боку визначає успішність учня, а з іншого боку є показником ефективності методики навчання і організації

навчального процесу. Включення різних форм тестових завдань в процес навчання мотивує учнів до активізації роботи по засвоєнню навчального матеріалу і формує прагнення розвивати свої здібності [4].

На даний момент в навчальних закладах використовують два види тестувань комп'ютерне та у вигляді тестових бланків.

Комп'ютерне тестування має ряд переваг:

- швидке формування результатів тестування;
- звільнення викладача від трудомісткої роботи з обробки результатів тестування;
- об'єктивність оцінювання;
- економія тимчасових трудовитрат.

При тестуванні знань учнів у вигляді тестових бланків використовується занадто багато ресурсів, як фінансових, так і трудових з боку викладача (пошук відповідних тестів, їх перевірка, оцінювання тощо).

Щоб провести тестування учнів у вигляді тестових бланків викладачеві необхідно:

- роздрукувати всі варіанти тестів на папері;
- забезпечити учнів бланками з варіантами тестів;
- зафіксувати на паперовому носії або в електронній записній книжці яка категорія учнів тестується, в даному випадку це можуть бути школярі вивчають елективний курс «Економіка 10-11 класи», студенти всіх напрямків, а також ті хто навчається за програмою додаткової освіти;
- зафіксувати на паперовому носії або в електронній записній книжці все прізвища і варіанти осіб, які тестуються;
- перевірити кожен тест і позначити неправильні відповіді;
- зафіксувати на паперовому носії або в електронній записній книжці оцінки учнів.

Якщо порівняти процедуру проведення тестування у вигляді тестових бланків та комп'ютерне, то очевидно, що комп'ютерне тестування є більш ефективним.

Отже, розробка мобільного додатку для комп’ютерного тестування, яке знизить витрати як фінансові, так і трудові покладені на викладача, є розумним рішенням.

1.4 Аналіз існуючих аналогів розробок

В даний час існує безліч систем, призначених для автоматизації тестування знань учнів.

Розглянемо програми, орієнтовані на специфіку навчання у вітчизняних вузах. Розглядати будемо три параметри: можливість роботи через інтернет; платформу і вид ліцензії. Решта необхідні параметри в більшості своїй схожі у всіх системах.

Система «СИнТеЗ»:

- має можливість роботи через мережу Інтернет;
- платформа: PHP, MySQL;
- ліцензія даної програми поширюється на платній основі.

Є можливість створювати питання різних типів, розбивати їх на теми, в текст питання і відповіді вставляти картинки, звук, фільми, списки, таблиці тощо.

Система «Конструктор тестів»:

- має можливість роботи через мережу Інтернет;
- платформа C++;
- ліцензія даної програми поширюється на безкоштовній основі.

У тестах є можливість використовувати музичний та звуковий супровід, графічні зображення, відеоматеріали. Тестування учнів проводиться на одному комп'ютері по черзі. Кожен тестований входить в програму під своїм іменем.

Система «OpenTest 2.0»:

- має можливість роботи через мережу Інтернет;
- платформа: HTML, PHP, JavaScript;
- ліцензія даної програми поширюється на безкоштовній основі.

При проектуванні модулів системи основна увага була приділена високою ергономічності системи, а також орієнтації на роботу з великим потоком користувачів [5].

Розглянуті системи є хорошими способами вирішення поставленого завдання, однак у кожної з них є недоліки, які зайві функції.

Нажаль, з’ясувати, які конкретно засоби застосовувалися при розробці вищевказаних систем, не вдалося, однак загальні принципи розробки та впровадження програмних продуктів були вивчені досить добре.

2 МЕТОДИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

2.1 Платформа Android

Платформа «Android» є продуктом групи «Open Handset Alliance», яка ставить перед собою ціль створити найбільш досконалу мобільну систему. З точки зору розробки програмного забезпечення «Android» знаходиться в самому центрі розробки відкритого програмного забезпечення. Перший пристрій під керуванням цією системою був випущений у 2008 році і єдиним інструментом розробки програм для нього були послідовні випуски пакету розробки «SDK».

За шириною можливостей платформа «Android» не поступається місцем операційним системах персональних комп'ютерів. Це багаторівнева система на основі ядра «Linux» з широкими функціональними можливостями. В підсистему користувацького інтерфейсу входять вікна, представлення та віджети. «Android» володіє широким спектром можливостей підключення, таких як «Wi-Fi», «Bluetooth» та протоколи передачі даних через стільникову мережу. В стек програмного забезпечення «Android» входить і підтримка сервісів, заснованих на визначенні місцеположення та акселерометрів, хоча не всі пристрої на цій платформі устатковані необхідним обладнанням. Також наявна підтримка відеокамери. Історично двома областями, де мобільні системи відставали від настільних були графіка та способи збереження даних. «Android» вирішує проблему графіки завдяки вбудованій підтримці засобів графічної обробки, включаючи бібліотеку "OpenGl". Задача збереження даних спрощується завдяки наявності в платформі «Android» популярної бази даних з відкритим кодом «Sqlite» [6].

Операційна система «Android» працює поверх ядра «Linux». Додатки пишуться мовами програмування «Java» або «Kotlin» і виконуються в віртуальній машині. Важливо уточнити, що віртуальна машина це не «JVM»,

як можна було б очікувати, а відкрита технологія «Dalvik Virtual Machine». Кожний додаток «Android» запускається всередині екземпляра «Dalvik VM», який в свою чергу укладений в рамках контрольованого ядром «Linux» процесу.

Компанія «Google» вклала багато ресурсів в скорочення платформи та зростання її ефективності. Оптимізація в першу чергу направлена на зменшення розміру, збільшення швидкості, економію заряду акумулятора та зниження витрат пам'яті. Вони провели роботу на декількох рівнях стеку. Віртуальна машина «Dalvik» була ретельно розроблена з метою задоволення цих вимог до продуктивності та має декілька незвичайних характеристик. На відміну від звичайного виконуваного середовища «Java», «Dalvik» заснований на регістрі, а не на стеку і не підтримує «JIT» компіляцію. Кожен окремий додаток виконується в окремому екземплярі віртуальної машини «Dalvik» і пам'ять розподіляється між цими екземплярами для зменшення витрат. Для того щоб скоротити час запуску додатку, «Dalvik» має компонент під назвою «Zygote», який попередньо ініціалізує екзмепляри віртуальних машин і розгалужує їх, якщо в цьому виникає необхідність.

Радикально інший підхід «Android» до розробки мобільних додатків пропонує деякі унікальні переваги, але він також створює багато проблем для сторонніх розробників. Найбільшою перевагою в такому підході є те, що він забезпечує високий рівень однорідності. Переважна більшість додатків «Android» можуть без проблем працювати практично на будь-якому пристрої на базі «Android», не потребуючи при цьому подальших змін.

До складу «Android» входить комплект базових додатків: клієнти електронної пошти, календар, різноманітні карти, браузер, програма для керування контактами тощо.

«Android» дозволяє використовувати всю потужність архітектурних компонентів, що використовуються в додатках ядра. Архітектура побудована таким чином, що будь який додаток може використовувати уже реалізовані можливості іншого додатку, при умові, що останній відкриє доступ до використання своєї функціональності. Таким чином архітектура реалізує

принцип багаторазового використання компонентів і додатків. На рисунку 2.1 зображена архітектура системи.

Рисунок 2.1 – Архітектура системи «Android»

Платформа «Android» надає повний та добре організований асортимент високорівневих прикладних інтерфейсів «API» для створення додатків та використовування основних функцій платформи. «API» забезпечують надзвичайно високий рівень абстракції, що робить їх відносно інтуїтивно зрозумілими і простими у використанні в процесі розробки.

Сторонні додатки можуть реплікуватися або взаємодіяти практично з усіма основними компонентами платформи. Наприклад, «Android» надає методи для отримання інформації з списку контактів користувача та для розширення системи контактів новими полями даних. Також легко зробити новий інтерфейс дзвінка, або реалізувати користувацьку поведінку для системних подій, таких як вхідні повідомлення. Зокрема, «API» дійсно дозволяють створювати додатки, які повністю інтегруються з рештою платформи. На рисунку 2.2 проілюстровано приклад використання «API».

Рисунок 2.2 – Приклад використання «Android API»

Набір інструментів для віджетів «Android» надає велику кількість дуже корисних компонентів відразу. Окремі віджети розроблені спеціально для зручної взаємодії з пальцями, із вже вбудованими функціями, такими як кінетична прокрутка. Розробники використовують мову опису користувацького інтерфейсу на основі «XML» для визначення макету та атрибутів віджетів, при цьому описи «XML» завантажуються в програму через систему ресурсів «Android» [7].

На окремі віджети, описані в макеті «XML», можна посилатися по ідентифікатору в програмі. Також є можливість створення віджетів програмно і маніпулювати ними користувацьким інтерфейсом в процесі виконання програми. Найбільш правильним способом створення макетів є написання описів «XML» вручну. Пакет «Android SDK» надає вбудований інструмент візуального макета, проте він не підтримує всі віджети і не завжди працює злагоджено.

Додатки «Android» складаються з провайдерів , служб, приймачів та активностей. Всі ці компоненти мають окрему задачу в стеку додатків «Android». Спосіб, яким вони можуть взаємодіями один з одним це те, що наповнює «Android» його модульністю.

Провайдери контенту слугують рівнем абстракції для взаємодії з різноманітними джерелами даних і для обміну постійними даними між додатками. Вони надають потрібну інформацію через стандартизований інтерфейс запитів. Запити описуються за допомогою синтаксису «URI», але розробники додатків зазвичай використовують багаторівневі класи оболонки, які генерують рядки запитів і керують ними. Коли програма надсилає запит провайдеру контенту, він повертає відповідь у вигляді об’єкту-курсору, який надає програмний доступ до базового контенту. Також можна підключити механізми моніторингу до провайдерів контенту, для того щоб ваш додаток міг отримувати повідомлення про зміну даних.

Система провайдера контенту пропонує розробникам «Android» додатків декілька суттєвих переваг. Найбільш явною перевагою є те, що система забезпечує високий рівень взаємодії, дозволяючи додаткам обмінюватися даними досить уніфікованим методом. Декілька ключових джерел даних платформи, включаючи список контактів та системи збереження мультимедіа, за замовчуванням доступні через інтерфейси провайдерів контенту, тому їх легко використовувати з сторонніх додатків для реалізації певних функцій.

Важливою особливістю, яка відрізняє «Android» від інших платформ є те, що «Android» офіційно підтримує фонові процеси в сторонніх додатках.

Вони реалізовані за допомогою сервісного компоненту. Сервіси – це операції без заголовку, які виконуються в фоновому режимі протягом тривалого часу.

Система повідомлень «Android» приблизно схожа з аналогічною системою сигналів в «Linux». Розробники вбудовують широкомовні приймачі, які можуть визначати, коли з'являються визначені системні повідомлення чи події, і виконувати певні дії у відповідь. Багато базових системних подій можна відслідковувати за допомогою системи мовлення, тому її можна використовувати для відстежування таких речей, як зміна стану акумуляторної батареї, отримання вхідних «SMS» повідомлень, натискання апаратних кнопок, зміна з'єднання, встановлення сховища та затемнення екрану [8].

Система «Android Intents» є ключем до розуміння того, як всі ці частини використовуються разом. Об'єкти «Intents», які містять ідентифікатор дії та «URI» даних, використовуються для виклику дій, служб та отримувачів. Ідентифікатор дії вказує бажану поведінку, а необов'язковий «URI» даних надає місце розташування цих даних, над якими має виконуватися дія. На рисунку 2.3 зображено загальну схему компонентів «Android» додатку.

Одна із головних переваг системи «Android» – її відкритість. Операційна система «Android» побудована на основі відкритого висхідного коду і розповсюджується на вільній основі. Це дозволяє розробникам отримати доступ до висхідного коду і зрозуміти яким чином реалізовані властивості і функції додатків. Кожен користувач може прийняти участь в удосконаленні операційної системи.

Рисунок 2.3 – Загальна схема компонентів «Android» додатку

2.2 Переваги мови програмування Kotlin над Java

В останні роки назріла потреба в новій мові програмування, яка компілюється в байт-код для віртуальної машини «Java». З'явилося кілька

проектів по створенню таких мов і один з них – «Kotlin», статично типізована об’єктно-орієнтована мова. Розробка проекту «Kotlin» почалася влітку 2010 року, в липні 2011-го проект був офіційно анонсований і на офіційному сайті «Kotlin» було розміщено опис мови. Випуск публічної бета-версії компілятору відбувся на початоку 2012 року. Мова розробляється в компанією «JetBrains». Код проекту доступний під вільною ліцензією «Apache 2». Компанія «JetBrains» створює інструменти для програмістів, які дозволяють прискорити процес написання коду, тестування, супроводження, спілкування з користувачами тощо [9].

Дві основні проблеми, які вирішує «Kotlin» і не вирішує «Java», такі.

1. Превентивна боротьба з перевіркою об’єктів на пустоту. Якщо в «Java» системі «null» значення легко розбігаються по всьому проекту. То в «Kotlin» розробник строго контролює де у нього можуть бути «null» значення і мінімізує їх поширення. Що зводить перевірки на «null» до мінімуму.

2. Властивості і покажчики на функції. Без цього дуже страждає як «ORM» так і багато інших систем на рефлексії – зв’язування даних і т. п. Програмісти змушені всюди описувати властивості рядком, а не прямим посиланням на властивість. Тому немає ні перевірки що ім'я написано правильно, ні перевірки на тип властивості під час компіляції.

Створюючи мову, компанія «JetBrains» керувалася низкою вимог, які здаються найбільш важливими для такого проекту. Розглянемо докладніше кожен з них.

Сумісність з «Java». Платформа «Java» – це перш за все екосистема: крім «офіційних» продуктів компанії «Oracle», в неї входить безліч проектів з відкритим кодом: бібліотек і фреймворків різного профілю, на базі яких будується величезна кількість додатків. Тому для мови, що компілюється, для цієї платформи, дуже важлива сумісність з існуючим кодом, який написаний на «Java». При цьому необхідно, щоб існуючі проекти могли переходити на нову мову поступово, тобто не тільки код на «Kotlin» повинен легко викликати код на «Java», але і навпаки.

Статичні гарантії коректності. Під час компіляції коду статично типізованою мовою відбувається безліч перевірок, покликаних гарантувати, що ті чи інші помилки не відбудуться під час виконання. Наприклад, компілятор «Java» гарантує, що об'єкти, на яких викликаються ті або інші методи, «вміють» їх виконувати, тобто що у відповідних класах ці методи реалізовані. На жаль, крім цієї дуже важливої властивості, Java майже нічого не гарантує. Це означає, що успішно скомпільовані програми завершуються з помилками часу виконання (викликають виняткові ситуації). Яскравим прикладом є розіменування нульового посилання, при якому під час виконання викликається виняток типу NullPointerException. Важливою вимогою до нової мови є посилення статичних гарантій. Це дозволить виявляти більше помилок на етапі компіляції і, таким чином, скорочувати витрати на тестування.

Швидкість компіляції. Статичні перевірки спрощують програмування, але уповільнюють компіляцію, і тут необхідно домогтися певного балансу. Досвід створення мов з потужною системою типів (найбільш яскравим прикладом є «Scala») показує, що такий баланс знайти непросто: компіляція часто стає неприйнятно довгою. Взагалі, така характеристика мови, як час компіляції проекту, може здатися другорядною, однак в умовах промислової розробки, коли обсяги коду що компілюється дуже великі, виявляється, що цей фактор дуже вагомий адже поки код компілюється, програміст часто не може продовжувати роботу. Зокрема, швидка компіляція є одним з важливих переваг «Java» в порівнянні з «C++», і «Kotlin» повинен зберігати цю перевагу.

Лаконічність. Відомо, що програмісти часто витрачають більше часу на читання коду, ніж на його написання, тому важливо, щоб конструкції, доступні в мові програмування, дозволяли писати програми коротко і зрозуміло. «Java» вважається багатослівною мовою (ceremony language - «церемонною мовою»), і завдання «Kotlin» – поліпшити ситуацію в цьому сенсі. На жаль, строгі методи оцінювання мов з точки зору їх лаконічності розвинені досить слабо, але є непрямі критерії; один з них – можливість

створення бібліотек, робота з якими близька до використання предметно-орієнтованих мов (Domain-Specific Language, DSL). Для створення таких бібліотек необхідна певна гнучкість синтаксису в сукупності з конструкціями вищих порядків, які є найбільш поширеними, тобто функції, які отримують інші функції в якості параметрів.

Доступність для вивчення. Складні статичні перевірки, гнучкий синтаксис і конструкції вищих порядків ускладнюють мову і ускладнюють її вивчення, тому необхідно до певної міри обмежувати набір підтримуваних можливостей, щоб мова була доступною для вивчення. При розробці «Kotlin» враховувався досвід «Scala» і інших сучасних мов, і занадто складні концепції в мову не включалися.

Інструментальна підтримка. Сучасні програмісти активно використовують різні автоматизовані інструменти, центральне місце серед яких займають інтегровані середовища розробки (Integrated Development Environment, IDE). Десятирічний досвід, накопичений в компанії «JetBrains», показує, що певні властивості мови можуть істотно ускладнювати інструментальну підтримку. При розробці «Kotlin» враховується цей факт і IDE створюється одночасно з компілятором [10].

За рахунок високої сумісності з «Java» і можливості замінювати старий код поступово, «Kotlin» став хорошою заміною «Java» в великих проектах і зручним інструментом для створення невеликих проектів з перспективою їх розвитку. Простота мови і його гнучкість дає розробнику більше можливостей для написання швидкого, але якісного коду.

2.3 Середовище розробки Android Studio

В наш час Створення програмного забезпечення здійснюється з використанням інтегрованого середовища розробки (IDE). В IDE автоматизований процес компіляції, збірки та запуску додатків, що значно спрощує роботу розробнику програмного забезпечення і дозволяє розробнику початківцю без значних зусиль створювати свої перші додатки.

Інтегроване середовище розробки зазвичай має декілька основних властивостей які спрощують розробку програмного забезпечення. Підсвічення синтаксису та нумерація рядків значним чином допомагають зорієнтуватися в програмному коді. Майже кожна IDE має автоматичний відладник додатків. Також дуже важливою є інтеграція з системою контролю версій. Ця функція використовується, коли над проектом працює не один розробник а ціла команда розробників. Ці системи дозволяють розробникам писати на відстані один і той самий код разом і не переписувати правки один одного. На сьогоднішній день кожна мова програмування має інтегроване середовище розробки [11].

В якості середовища розробки було обрано «Android Studio». Це інтегроване середовище розробки виробництва компанії «IntellijIDEA», за допомогою якого розробникам стають доступні інструменти для створення додатків. Це середовище можна встановити на різні операційні системи. «Android Studio» можна завантажити і користуватись безкоштовно. В ньому присутні макети для створення користувацького інтерфейсу, з чого зазвичай і починається робота над додатком. Середовище містить інструменти для створення додатків на смартфони, планшети, годинники та автомобілі.

Рисунок 2.4 – Екран вибору форм-фактору в Android Studio

Середовище «Android Studio» – це офіційне інтегроване середовище розробки для проектування та розробки додатків «Android». Воно побудоване на основі «IntellijIDEA», інтегрованого середовища розробки для «Java» та «Kotlin», і включає в себе інструменти редагування коду та середовище розробки додатків. Вперше «Android studio» було анонсовано на «Google I/O» в травні 2013 року, а перша стабільна збірка була випущена в грудні 2014 року. Середовище «Android Studio» доступне для таких настільних платформ як «Windows», «Mac» та «Linux». Воно замінило

«Eclipse» в якості основного інструменту для розробки додатків на «Android».

Для того щоб почати роботу в «Android Studio» спочатку необхідно встановити безкоштовний пакет для розробника «JDK» (Java Development Kit). Після встановлення та налаштування цього пакету необхідно завантажити «Android Studio» з офіційного сайту. Далі необхідно знайти завантажений виконуваний файл інсталяції «Android Studio» та запустити процес встановлення. Коли з’явиться майстер налаштування Android Studio необхідно налаштувати встановлення відповідно до вимог користувача з точки зору розташування файлової системи, в яку повинне бути встановлене середовище «Android Studio» та доступу іншими користувачами системи. Після встановлення інтегрованого середовища розробки можна починати створювати додатки (рис. 2.5).

Рисунок 2.5 – Стартовий екран проекту в Android Studio

Інтегроване середовище розробки «Android Studio» є віконним. Для того щоб максимально використати простір екрану і щоб не перевантажувати розробника «Android Studio» відображає лише невелику частину доступних вікон в будь-який час. Деякі вікна є контекстними і з’являються тільки в разі контекстного виклику з певних інструментів. Також «Android Studio» містить приховані вікна, які розробник може активувати або деактивувати з меню налаштувань. Однією ж найважливіших функцій будь якого інтегрована середовища розробки є навігація. Проекти «Android» зазвичай складаються з багатьох тек, каталогів та файлів, що організовані між собою. Програмні додатки з багатьма висхідними файлами коду та ресурсами організовані в рамках структури проекту. Це зроблено для кращої класифікації файлів і визначення компіляції висхідного коду та генерації байт-коду. В цілому ця файлова структура і називається проектом, який є організаційною одиницею, що являє собою повне програмне рішення. Проекти додатків зазвичай

складаються з файлів висхідного коду «Java» або «Kotlin», файлів конфігурації «XML», зображень, стилів та інших ресурсів в організаційній структурі. При створенні програми «Android Studio» допомагає у структуризації проекту. На етапі збірки проекту створюються файли конфігурації та відбувається структуризація каталогів за ієрархією (рис. 2.6).

Рисунок 2.6 – Структура проекту в Android Studio

В процесі розробки додатків в інтегрованому середовищі розробки «Android Studio» необхідно буде компілювати та запускати додаток декілька разів. Розроблену програму «Android» можна перевірити, встановити та запускати на фізичному пристрої або на віртуальному пристрої «Android» (AVD – android virtual device), який називають емулятором. Перед тим як використовувати віртуальний пристрій його необхідно завантажити та налаштувати. Емулятор надає практично всі можливості реального пристрою «Android». Користувач може імітувати вхідні дзвінки так текстові повідомлення, вказувати місцезнаходження пристрою, імітувати обертання екрану та інші апаратні датчики, мати доступ до «Google Play» та інше. Тестування додатку на віртуальному пристрої в деякій мірі зручніше та швидше, ніж на фізичному пристрої. Наприклад, передача даних на емулятор відбувається швидше, ніж на підключений фізичний пристрій.

Рисунок 2.7 – Віртуальний девайс в Android Studio

2.4. Система автоматизованої збірки Gradle

Процес створення програмного продукту для розробників та тестувальників програмного забезпечення без засобів автоматизації є дуже повторюваним, нудним та значно підвищує ризик помилок. Кожен крок в процесі розробки програмного продукту, починаючи з джерела компілювання коду до завершальної збірки програмного забезпечення для релізу та перевірки на виробництві потрібно робити вручну. Автоматизація проекту допомагає знизити кількість ручного втручання в проект, робить процес розробки більш ефективним та мінімізує можливість виходу програмного забезпечення з ладу.

Автоматизація проекту має кілька очевидних переваг для процесу розробки програмного продукту. Перша із них це запобігання ручному втручанню в процес збірки. Необхідність вручну виконувати дії для створення та постачання програмного забезпечення вимагає багато часу і підвищує ризик виникнення помилок. Розробник та системний адміністратор мають багато різних задач, які вони могли б вирішувати в той час коли займаються компіляцією вручну. Будь-який крок в процесі розробки, який може бути автоматизований – має бути автоматизований. Наступною перевагою є створення повторюваних збірок. Звичайна побудова програмного проекту відповідає попередньо визначеним і впорядкованим крокам. Наприклад, розробнику необхідно спочатку скомпілювати вихідний код, потім запустити тести і нарешті зібрати проект з отриманням певного результату. Доведеться виконувати ті самі кроки знову і знову кожного дня. Цей процес має бути таким же легким як натискання кнопки. Результат цього процесу має повторюватися для всіх, хто запускає збірку проекту. Також до переваг можна віднести портативні збірки. Можливість запуску збірки з IDE дуже обмежена. По-перше, розробнику програмного продукту необхідно встановити на своєму пристрої певний продукт. По-друге, IDE може бути доступна лише для певної операційної системи. Автоматизована збірка не вимагає певного середовища виконання, незалежно від того, чи це операційна система або середовище розробки. Оптимально, автоматизовані

завдання повинні виконуватися з командного рядка, який дозволяє запускати процес збірки з будь-якого пристрою та в будь-який час [12].

Автоматизація проекту в свою чергу ділиться на декілька основних типів. Перший із них це збірка на вимогу. Типовий випадок використання автоматизації за вимогою, це випадок коли користувач запускає побудову на певному пристрої. Зазвичай система контролю версій VSC керує версифікацією збірок та файлами висхідного коду. У більшості випадків користувач виконує скрипт у командному рядку, який виконує завдання у попередньо визначеному порядку. Наприклад, компіляція вихідного коду, копіювання файлу з одного каталогу до іншого або збірка результату. Зазвичай цей тип автоматизації виконується по кілька разів на день. Наступним типом автоматизації є запущені збірки. Якщо розробник займається гнучкою розробкою програмного забезпечення, його цікавить швидке отримання зворотного зв’язку про стан проекту. Також важлива інформація про те, чи може вихідний код бути зібраним без будь-яких помилок та інформація про наявність в проекті потенційного програмного дефекту поміченого в помилці блоку чи тесті інтеграції. Цей тип автоматизації зазвичай спрацьовує в результаті перевірки коду системою контролю версій. Ще одним типом автоматизації є заплановані збірки. Задумка полягає в плануванні автоматизації як планування завдань на основі часу. Вона виконується в певні проміжки часу або в певний конкретний час. Запланована автоматизація зазвичай виконується на виділеному сервері. Такий вид автоматизації особливо корисний для створення звітів або документації для проекту. Практика реалізації запланованих та запущених збірок, зазвичай визначається як безперервна інтеграція CI.

Рисунок 2.8 – Приклад запланованої збірки

Протягом багатьох років збірки мали прості вимоги до складання та пакування програмного забезпечення. Проте ландшафт сучасної розробки програмного забезпечення змінився і з’явилася потреба в автоматизації побудови проектів. В наш час проекти включають великі та різноманітні програмні стеки, включають велику кількість мов програмування та застосовують широкий спектр стратегій тестування. З підвищенням гнучких практик, збірки повинні підтримувати якнайшвидшу інтеграцію коду, а також швидке та легке постачання продукту в тестові та виробничі середовища. Встановлені інструменти побудови зазвичай не досягають цих цілей в простому вигляді. Оскільки розробнику програмного продукту потрібно чимало часу для того щоб розібратися з «XML» кодом збірки. Також не було можливості задати власну логіку побудови, при додаванні сценарію збірки його неможливо було змінити.

Проте є спосіб зробити ці речі простими і виразними, це – «Gradle». Система автоматизованої збірки «Gradle» вперше була випущена в 2007 році. «Gradle» є наступним еволюційним кроком в інструментах побудови JVM. Ця система спирається на досвід вже існуючих раніше інструментів автоматизованої побудови, таких як «Ant» та «Maven», та переносить їхні найкращі ідеї на наступний рівень. Слідуючи підходу до побудови, «Gradle» дозволяє декларативно моделювати вашу предметну область за допомогою потужної та експресивної доменної мови «DSL», реалізованої в «Groovy» замість «XML». Оскільки «Gradle» є рідним до «JVM», вона дозволяє писати власну логіку мовою «Groovy» або «Java». В мові «Java» існує неймовірно велика кількість бібліотек і фреймворків. Управління залежностями використовується для автоматичного завантаження цих артефактів зі сховища і надання їх до коду програми. Дізнавшись про недоліки існуючих рішень управління залежностями, «Gradle» забезпечує власну реалізацію. Він не тільки добре налаштований, але й прагне бути максимально сумісним з існуючими інфраструктурами управління залежностями, такими як «Maven» та «Ivy». «Gradle» надає потужну підтримку для визначення та організації багато проектних збірок, а також моделювання залежностей між проектами.

Для розробника автоматизація проекту є частиною повсякденної роботи. Сценарії побудови «Gradle» є декларативними, читабельними і чітко виражають свій намір. Написання коду в «Groovy» замість «XML» з елементами філософії «Build-byConvention», дозволяє значно скоротити розмір сценарію збирання і є набагато більше читабельним, що зображено на рисунку 2.9.

Рисунок 2.9 – Порівняння скриптів «Maven» та «Gradle»

Кожна побудова «Gradle» починається з сценарію. Стандартне найменування сценарію для скрипту «Gradle» є build.gradle. При виконанні базової команди в оболонці система збірки шукає файл з саме таким іменем. Якщо його не буде знайдено буде відображено повідомлення про помилку. Після створення цього файлу необхідно визначити одне або декілька завдань.

Підводячи підсумок, можна визначити, що «Gradle» – це автоматизована система збірки, яка походить від декларативного і виразного «Groovy DSL». Вона поєднує в собі гнучкість і простоту розширення з ідеєю про конфігурацію і підтримку традиційного управління залежностями. «Gradle» стає першочерговим вибором для побудови багатьох проектів з відкритим кодом.

Рисунок 2.10 – Набір функцій системи «Gradle»

2.5 Порівняння Java та Kotlin як засобів для написання додатків під ОС Android

Kotlin – це статично типізована мова програмування, розроблена компанією JetBrains. Подібно мові Java, Kotlin став відмінним вибором для розробки додатків на Android. Це можна побачити навіть із того факту, що Android Studio забезпечує вбудовану підтримку Kotlin, як і Java.

Одна з основних відмінностей між Java і Kotlin полягає в тому, що в останньому немає умов для винятків, що перевіряються (checked exception). Отже, немає необхідності відловлювати чи оголошувати якісь винятки. Якщо розробник, що працює на Java, вважає, що використання конструкції try/catch у коді дратує, то спрощення, зроблене Kotlin, можна вважати бажаною зміною. Однак протилежністю буде, якщо розробник вважає, що винятки, які перевіряються, потрібні, тому що вони сприяють відновленню роботи після помилок і створенню надійного коду. У цьому випадку можна вважати дане нововведення плюсом та мінусом, залежно від підходу до розробки.

Лістинг 2.1 – Приклад блоку try/catch мовою програмування Kotlin

```
try {
    log.append(message)
} catch (IOException e) {
    // Must be safe
}
```

Порівняння класу Java з еквівалентним класом Kotlin демонструє лаконічність коду Kotlin (див. лістинги 2.2 та 2.3). Для тієї ж операції, що виконується в класі Java, Kotlin вимагає менше коду.

Лістинг 2.2 – Приклад класу User мовою Java

```
public class User {

    private String name;
    private String surname;
    private String workDepartment;
    private int age;

    public User(String name, String surname, String workDepartment, int
age) {
        this.name = name;
        this.surname = surname;
        this.workDepartment = workDepartment;
```

```
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public String setName(String name) {
        this.name = name;
    }
    public String getSurname() {
        return surname;
    }
    public String setSurname(String name) {
        this.surname = surname;
    }
    public String getWorkDepartment() {
        return workDepartment;
    }
    public String setWorkDepartment(String name) {
        this.workDepartment = workDepartment;
    }
    public String getAge() {
        return age;
    }
    public String setAge(int age) {
        this.age = age;
    }
}
```

Лістинг 2.3 – Приклад класу User мовою Kotlin

```
data class User(
    var name: String,
    var surmane: String,
    var workDepartment: String,
    var age: Int
)
```

Наприклад, конкретний сегмент, де Kotlin може значно скоротити загальний обсяг стандартного коду, – це findViewByIds. Розширення Kotlin до Android дозволяють імпортувати посилання на View у файл Activity. Це

дає можливість працювати з конкретним представленням, так ніби воно є частиною Activity. Дану особливість явно можна зарахувати до плюсів Kotlin.

Процеси, що інтенсивно завантажують процесор і мережеве введення-виведення, зазвичай використовують тривалі операції. Викликаючий потік блокується до завершення всієї операції. Оскільки Android є однопотоковим за замовчуванням, інтерфейс користувача повністю блокується, як тільки блокується основний потік. Традиційне вирішення цієї проблеми Java – створити фоновий потік для тривалої або інтенсивної роботи. Однак керування кількома потоками призводить до збільшення складності, а також помилок у коді. Kotlin також дозволяє створювати додаткові потоки. Тим не менш, є найкращий спосіб управління інтенсивними операціями в Kotlin, відомий як співпрограми або корутини (coroutines). Корутини в Kotlin реалізовані без стека, це означає, що вони вимагають меншого використання пам’яті порівняно із звичайними потоками. Корутини можуть виконувати тривалі та інтенсивні завдання, зупиняючи виконання не блокуючи потік, а потім відновлюючи виконання через деякий час. Це дозволяє створювати неблокуючий асинхронний код, який виглядає як синхронний (лістинг 2.4). Код з використанням корутин не лише зрозумілий, а й лаконічний. Більше того, корутини дозволяють створювати елегантні додаткові стилі асинхронної неблокуючої розробки, такі як async та await. Все це також явно відноситься до плюсів Kotlin [13].

Лістинг 2.4 – Приклад співпрограми (корутини)

```
fun main() = runBlocking { // this: CoroutineScope
    launch { // launch a new coroutine and continue
         delay(1000L) // non-blocking delay for 1 second (default time unit is ms)
        println("World!") // print after delay
    }
    println("Hello") // main coroutine continues while a previous one is delayed
}
```

У повнорозмірних проектах зазвичай є кілька класів, які призначені виключно для зберігання даних. Хоча ці класи практично не мають функціональності, розробнику необхідно написати багато стандартного коду Java. Зазвичай розробник повинен визначити конструктор та кілька полів для зберігання даних, функції гетери та сетери для кожного з полів, а також функції equals(), hashCode() та toString(). Kotlin має дуже простий спосіб створення таких класів. Розробнику достатньо лише включити ключове слово data у визначення класу, і компілятор сам про все подбає. Така зручність створення класів у питаннях для Kotlin «за і проти» явно свідчить на його користь.

Kotlin дозволяє розробникам розширювати клас новими функціями за допомогою функцій розширення. Ці функції, хоч і доступні в інших мовах програмування, таких як C#, але не є доступними в Java. Створити функцію розширення в Kotlin легко. Це робиться шляхом додавання префіксу імені класу, який має бути розширений до імені створюваної функції. Щоб викликати функцію в екземплярах розширеного класу, потрібно використовувати нотацію «.».

Лістинг 2.5 – Приклад використання функції розширення

```
class Example {
    fun printFunctionType() { println("Class method") }
}

fun Example.printFunctionType(i: Int) {
      println("Extension function #$i")
}

Example().printFunctionType(1)
```

Функція вищого порядку – це функція, яка приймає інші функції як параметри або повертає функцію. Крім того, функції Kotlin є функціями першого класу. Це означає, що вони можуть зберігатися в структурах даних та змінних, які можуть передаватися як аргументи та повертатися з інших

функцій вищого порядку. Це означає, що функції можуть працювати всіма можливими способами у взаємодії з іншими нефункціональними значеннями. Як статично типізована мова програмування, Kotlin використовує ряд функціональних типів для представлення функцій. Більш того, він поставляється з набором спеціалізованих мовних конструкцій, таких як лямбда-вираз. Анонімні функції та лямбда-вирази також відомі як функціональні літерали. Це функції, які не оголошені, але передаються як вирази.

Лістинг 2.6 – Приклад функцій вищого порядку Kotlin

```
val repeatFun: String.(Int) -> String = { times -> this.repeat(times) }
val twoParameters: (String, Int) -> String = repeatFun // OK

fun runTransformation(f: (String, Int) -> String): String {
    return f("hello", 3)
}
val result = runTransformation(repeatFun) // OK
```

В Kotlin немає підтримки неявних розширювальних перетворень для даних. Таким чином, менші типи не можуть бути перетворені на великі типи. В той час як Java підтримує неявні перетворення, Kotlin вимагає виконати саме явне перетворення. Низка розробників сприймає це як мінус Kotlin.

Змінні, до яких здійснюється доступ у тілі функції, називаються замиканнями. Використання функцій вищого порядку може значно збільшити час виконання обчислень. Кожна функція Kotlin є об’єктом, і він захоплює замикання. І класи, і функтори вимагають виділення пам’яті. Вони поряд з віртуальними викликами вимагають певних витрат на час виконання. Таких додаткових витрат можна уникнути, вставивши лямбда-вираження у Kotlin. Одним із таких прикладів є функція lock(). На відміну від Kotlin Java не забезпечує підтримку вбудованих функцій. Тим не менш, компілятор Java здатний виконувати вбудовування за допомогою методу final. Це так, тому що методи final можуть бути перевизначені підкласами. З іншого боку,

виклик методу final дозволяється під час компіляції. Такі нововведення також сприймаються як переваги Kotlin.

Лістинг 2.7 – Синтаксис вбудованої функції lock()

```
inline fun <T> lock(lock: Lock, body: () -> T): T { ... }
```

У термінології програмування, делегування – це процес, у якому приймаючий об’єкт делегує свої операції другому об’єкту делегата. Kotlin підтримує шаблон проектування композиції поверх наслодування через делегування першого класу, також відомий як неявне делегування. Делегування Kotlin є альтернативою успадкування. Цей механізм дозволяє використовувати множинне наслідування. Крім того, делеговані властивості Kotlin запобігають дублювання коду. Шаблон делегування є гарною альтернативою наслідування і Kotlin підтримує його нативно, звільняючи від необхідності написання шаблонного коду що є проілюстровано в лістингу 2.8. Ключове слово by в змісті Derived, що знаходиться після типу класу, який делегується, говорить про те, що об'єкт b типу Base буде зберігатися всередині екземпляра Derived, і компілятор згенерує у Derived відповідні методи з Base, які при виклику будуть передані об’єкту b.

Лістинг 2.8 – Приклад делегування в Kotlin

```
interface Base {
    fun print()
}

class BaseImpl(val x: Int) : Base {
    override fun print() { print(x) }
}

class Derived(b: Base) : Base by b

fun main(args: Array<String>) {
    val b = BaseImpl(10)
    Derived(b).print() // prints 10
}
```

Інкапсуляція необхідна у будь-якій програмі для досягнення бажаного рівня керованості. За допомогою інкапсуляції представлення об’єкта може бути встановлене виходячи з того, як викликаючі сторони взаємодіють з ним. Крім того, можна змінити представлення без необхідності зміни викликаючих абонентів, якщо публічний API залишається незмінним. Неприватні поля або public поля в Java корисні в сценаріях, де об’єкти, що викликають, повинні змінюватися відповідно до їх представлення. Це означає, що такі поля надають представлення об’єкта викликаючим об’єктам. Kotlin не має private полів. Це досить цікава відмінність при порівнянні Kotlin та Java.

Однією з найпомітніших проблем для розробників, пов’язаних з Java, є NullPointerExceptions. Java дозволяє розробникам присвоїти значення null будь-якій змінній. Однак, якщо вони намагаються використати посилання на об’єкт із значенням null, виникає виняток NullPointerException! На відміну від Java, Kotlin всі типи за замовчуванням не-nullable. Якщо розробники намагатимуться привласнити або повернути значення null у коді Kotlin, під час компіляції відбудеться збій. Проте є спосіб обійти цей момент. Щоб призначити значення null змінній Kotlin, необхідно явно помітити цю змінну як nullable. Це робиться шляхом додавання запитання після типу, що є проілюстровано в лістингу 2.9. Таким чином, у Kotlin немає винятків NullPointerException. Якщо ви зустрічаєте такий виняток у Kotlin, то, швидше за все, ви або явно надали значення null, або це пов’язано з якимось зовнішнім Java-кодом.

Лістинг 2.9 – Приклад ініціалізації nullable змінної

```
val number: Int? = null
```

Існує 8 примітивних типів даних, включаючи char, double, float та int. На відміну від Kotlin, змінні примітивного типу не є об’єктами Java. Це означає, що вони є об’єктами, створеними з класу чи структури.

Перш ніж об’єкт може бути приведений у Java, обов’язково потрібно перевірити тип. Це також правильно в сценаріях, де явно необхідно приводити об’єкт. На відміну від Java, Kotlin має функцію розумного приведення, яка автоматично обробляє такі надлишкові приведення. Вам не потрібно виконувати приведення всередині оператора, якщо він вже перевірений оператором is.

Лістинг 2.10 – Приклад використання оператору is

```
// x is automatically cast to String on the right-hand side of `||`
if (x !is String || x.length == 0) return

// x is automatically cast to String on the right-hand side of `&&`
if (x is String && x.length > 0) {
    print(x.length) // x is automatically cast to String
}
```

У Kotlin немає статичних елементів. Однак у мові програмування Java ключове слово static відображає те, що конкретний член, з яким використовується це ключове слово, належить типу, а не екземпляру цього типу. Це означає, що один і лише один екземпляр цього статичного члена створюється та використовується всіма екземплярами класу.

Класи Kotlin, на відміну від класів Java, можуть мати один або кілька вторинних конструкторів, крім первинного конструктора. Це робиться шляхом включення цих вторинних конструкторів до оголошення класу (лістинг 2.11).

Лістинг 2.11 – Приклад оголошення конструкторів в Kotlin

```
class Constructors {
    init {
        println("Init block")
    }

    constructor(i: Int) {
        println("Constructor $i")
```

```
    }
}
```

На відміну від Kotlin Java має тернарний оператор. Тернарний оператор Java працює як базовий оператор if. Він складається з умови, яка оцінюється як істинне чи хибне. Крім того, тернарний оператор Java має два значення. Тільки одне з них повертається залежно від того, чи є умова істинною чи хибною. Синтаксис для тернарного оператора Java:

Лістинг 2.12 – Синтаксис тернарного оператора Java

```
(стан) ? (значення 1) : (значення 2)
```

У загальному коді, «?» представляє невідомий тип. Цей символ відомий як знак підстановки. Існує кілька варіантів використання підстановочного знаку, у тому числі як тип поля, локальна змінна або параметр. У той час як система типів Java пропонує використовувати підстановочні знаки, Kotlin їх немає. Тим не менш, у нього є два інші механізми - варіативність на рівні оголошення та проекції типів як альтернатива. Буде корисно враховувати це під час переходу з Java на Kotlin [14].

Крім надання підтримки існуючим Java-фреймворкам та бібліотекам, Kotlin також пропонує розширені Java-фреймворки, що базуються на обробці анотацій. Однак застосування в Kotlin бібліотеки Java, яка використовує обробку анотацій, вимагає додавання її в проект Kotlin трохи іншим способом, ніж потрібно для бібліотеки Java, яка не використовує обробку анотацій. Потрібно вказати залежність за допомогою плагіну kotlin-kapt. Після цього необхідно використовувати інструмент обробки анотацій Kotlin замість annotation Processor.

Очевидно, деякі моменти краще реалізовані в Kotlin, тоді як для інших – вигідно використовувати Java. Для тих, хто не хоче відмовлятися від будь-якої з двох провідних мов програмування для розробки під Android, на щастя, є інший шлях. Незалежно від усіх відмінностей між двома мовами програмування вони повністю сумісні. І Java, і Kotlin компілюються у байт-

код. Це означає, що в питаннях Kotlin vs Java не буде однозначної відповіді, адже можна викликати код Java з Kotlin і навпаки. Ця гнучкість має дві переваги. По-перше, це полегшує початок роботи з Kotlin, поступово додаючи код Kotlin у проект Java. По-друге, обидві мови можуть використовуватися одночасно в будь-якому проекті розробки програм для Android.

Незважаючи на значні плюси Kotlin, у питаннях розробки загального призначення Java перемагає. З іншого боку, все більше розробників та організацій впроваджують Kotlin для швидкої розробки Android додатків. І Java, і Kotlin мають переваги один перед одним. Дискусія про те, який з них підходить для розробки краще, тільки почалася і навряд чи вона закінчиться найближчим часом. Питання вибору: чому Java, чому Kotlin досі залишаються. У будь-якому випадку перехід із Java на Kotlin не буде болючим.

2.6 Порівняння архітектурних патернів проектування MVC, MVP, MVVM

Розробка ПЗ схожа з процесом будування будинку. Перша та найважливіша річ, яку потрібно зробити, – закласти фундамент. Довголіття та надійність, серед багатьох інших речей, залежить від міцності фундаменту. В сервісах розробки мобільних застосунків архітектура відіграє ключову роль і впливає на розмір, структуру, масштабованість та здатність підтримки проекту.

Звісно, немає універсального правила для розробки мобільних застосунків, так як і універсального фундаменту. Кожен проект має свій набір вимог та потреб в унікальному підході. Три найкращі та найвідоміші типи розробки ПЗ: MVC, MVP та MVVM. Кожен з них має підмножину варіантів використання. Найкраще, порівняти їх перед обранням одного з них.

Ці фактори застосовуються при розробці ПЗ для ОС Android та IOS:

– тип проекту;

– операційна система;

– стек технологій;

– набір інструментів для розробки та SDK;

– серверна інфраструктура та бази даних;

– хмарні технології;

– сторонні сервіси (наприклад для оплати);

– UI/UX, вміст та складність навігації;

– форматування даних;

– плани на масштабування;

– рівень навичок команди;

– час та бюджетні обмеження.

Навіщо це потрібно та чому варто ретельно обирати?

Розподіл інтересів. Принцип, який визначає окремі сутності що повинні бути розділеними в коді. Він допомагає з перевикористанням тих сутностей, систематичною розробкою, простим налагодженням та ізоляцією частин, які часто змінюються та не повинні впливати на інші.

Тестопридатність. З відповідною архітектурою це відбувається частіше. Тестувальники знайдуть корисним написати умови для тестування та мати можливість тестування функціоналу окремо. Розробник позбавиться знаходження проблем під час виконання, що зазвичай означає виправлення довжиною в тиждень.

Надійність. Вдало обрана архітектура має вирішальне значення в розробці стабільних додатків без важливих несумісностей, тому що вона визначає, які частини коду взаємодіятимуть між собою.

Масштабованість. Додаток повинен мати стійкий базис для додавання нового функціоналу та зміни існуючого, відповідно до бізнес вимог. Також він повинен підходити для подальших реновацій (нові бібліотеки, операційні системи) в програмних технологіях.

Підтримка та легкість у використанні. Хороша архітектура спрощує написання та читання коду. Також в результаті зменшується вартість підтримки проекту.

Яка відмінність між MVC, MVVM та MVP? Розглянемо кожен з них та визначимо випадки яким вони відповідають.

MVC (model-view-controller) – вважається одним з найпопулярніших в розробці мобільних застосунків. Патерн MVC дозволяє створювати прості кросплатформенні застосунки з простою логікою навігації, не складним вмістом, більш стандартизованим дизайном додатків та досить прямим користувацьким досвідом. Він може просто та швидко реалізовувати завдання типу «завантаження-відображення».

Рисунок 2.11 – Діаграма архітектури MVC

В яких випадках варто використовувати даний патерн?

– Коли необхідний бистрий процес розробки;

– Якщо потрібно забезпечити нескладну логіку навігації;

– Коли потрібно створити простий застосунок типу «клієнт-сервер»;

– Коли передбачена робота з простими даними;

– Якщо потрібні неформатовані дані в результаті;

– Якщо потрібно створити застосунок зі зручним пошуком;

– Якщо немає непрямих інструкцій навігації на екрані.

Наступний патерн, який найчастіше використовується – MVVM (model-view-view-model).

На відміну від попереднього патерну, він дозволяє відокремлювати логіку дизайну від бізнес логіки. Що дозволяє реалізовувати більш складні функціонально завдання та дає користувачу ширший вибір дій та взаємодій з мобільним застосунком. Однією з переваг є зручна реалізація автотестів оскільки бізнес логіка, що тестується, є відокремленою від логіки користувацького інтерфейсу.

Рисунок 2.12 – Діаграма архітектури MVVM

В яких випадках використовувати MVVM? Даний патерн є більш універсальним та підходящим для написання складних застосунків. MVVM – це хороший вибір для розробки малих проектів з перспективою подальшого розширення функціоналу.

MVP (model-view-presenter) – патерн паралельний у використанні MVC. Вибір використання між цими патернами залежить від навантаженості користувацького інтерфейсу та відмінності відображення даних. Тестопридатність даних в MVP обмежена моделями та представленнями.

Рисунок 2.13 – Діаграма архітектури MVP

В яких випадках використовується? Даний патерн використовується, якщо є обмежений набір компонентів користувацького інтерфейсу та проста навігація в застосунку.

Підсумовуючи розглянуту інформацію, що була наведена вище, можна дійти висновку: якщо необхідно розробити додаток з простими навігацією, вмістом та користувацьким інтерфейсом, тоді, ймовірно, MVC правильний вибір. Якщо проект малий, але більш складний, з перспективою подальшого розширення, слід подумати про MVVM або MVC. Вибір між ними залежить від навантаженості користувацького інтерфейсу та відображення даних.

Враховуючи той фактор, що в поточному додатку буде присутня, доволі, складна логіка для парсингу файлу з тестовими даними, логіка вивантаження даних до серверу, логіка зчитування даних та логіка формування звітності, було прийнято рішення використовувати патерн MVVM.

3 ПРОЕКТУВАННЯ ДОСЛІДЖУВАНОЇ СИСТЕМИ

3.1 Огляд моделі предметної області

Будь-яка система, в якій виконується маніпуляція даними, повинна мати свою модель структурних даних. Дана система не є виключенням.

Рисунок 3.1 – UML діаграма класів системи мобільного тестування

На рисунку 3.1 продемонстрована UML діаграма класів, які використовуються в даній системі. Розглядаючи наведений рисунок, можна побачити, що на ньому присутні два види зв’язків. Перший вид – із заповненою позначкою. Даний тип означає обов’язкову присутність конкретного поля в класі. Інший тип – з порожніми позначками зв’язку. Цей тип вказує на необов’язкову присутність конкретних полів в класах.

Клас User являє собою модель користувача, які містяться основні дані, такі як: ідентифікатор, ім’я, прізвище, адреса електронної пошти та ідентифікатор активного тесту. Всі поля є обов’язковими, окрім поля ідентифікатору активного тесту, тому що користувач може бути в стані проходження тесту, або ні.

Модель Test є найголовнішою в даній системі, оскільки в ній міститься важлива інформація про конкретний тест. В ній зберігаються такі поля, як: ідентифікатор тесту, опис, тема, масив питань та масив користувачів. Масив користувачів є опціональним, тому що існує випадок, коли тест не проходить жоден студент. А от масив питань є обов’язковим, тому що тест не може існувати без запитань.

Question – відповідно модель питання. В ній зберігаються такі дані: ідентифікатор, опис, тип, варіанти відповіді та посилання на зображення.

Поле зображення є необов’язковим, тому що не кожне питання містить зображення.

Answer – відповідь. Невеличкий об’єкт, який зберігає такі поля: ідентифікатор, опис та логічне поле яке визначає, чи є дана відповідь правильною.

Також присутня модель QuestionType, яка є допоміжною. Вона була створена для того, щоб ідентифікувати тип конкретного питання. Може містити значення трьох типів: питання з однією правильною відповіддю (RADIO), питання з декількома правильними відповідями (CHECKBOX) та питання з розгорнутим варіантом відповіді (TEXT).

Дані що містяться в даній моделі отримуються шляхом парсингу excel файлу.

3.2 Огляд структури вхідних даних

Вхідними даними для даної системи слугує excel таблиця з даними відповідного формату, який можна побачити на рисунку 3.2.

Рисунок 3.2 – Приклад заповнення excel файлу

Для того, щоб ввести дані про тест до системи, необхідно заповнити excel файл за зразком, який є проілюстровано вище. Поля з жовтим фоном підлягають редагуванню, з білим – повинні залишатися без змін.

В першому рядку, користувачеві необхідно вказати дані про тест. Комірка «B1» міститиме атрибути тесту та дані авторів, наприклад «СКСм-20-1, Шкіль. О. С.». В комірці «D1» необхідно вказати тему тесту, наприклад «ЛМОД – Тестування за темою 1». Обидві комірки в першому рядку є обов’язковими до заповнення.

В другому рядку міститься інформація про тривалість тестування. В комірці «B2» необхідно вказати час, який буде відведено на тестування. Час необхідно вказати в хвилинах. Заповнення даної комірки є обов’язковим.

Починаючи з рядка під номером 3 відбувається внесення даних про запитання. Кожне запитання, повинно починатись з ключового слова «Запитання (тип запитання)», яке завжди повинне знаходитись в «A» стовпці. Користувачеві необхідно явно вказати тип конкретного запитання. Після ключового слова, в комірці «B» необхідно ввести текст запитання. Текст запитання є обов’язковим. Після зазначення тексту, в стовпці «C» можна розмістити зображення. При проектуванні системи, розглядалась можливість вставки формул, але через відсутність підтримки, текстовим редактором «Excel», формул з середовища «Word», було прийнято рішення використовувати формули в форматі зображень. Користувач має можливість створити формулу в текстовому редакторі «Word», скопіювати її та вставити до «Excel» файлу і редактор автоматично переформатує скопійовану формулу в зображення. Зображення в питанні не є обов’язковим.

Після введення основної інформації щодо запитання, необхідно внести інформацію про відповіді. Кількість відповідей на конкретне запитання повинна перевищувати 1. Система підтримує три типи питань:

1. Запитання з однією правильною відповіддю (Radio);
2. Запитання з декількома правильними відповідями (Checkbox);
3. Запитання з текстовим форматом відповіді (Text).

В стовпці з літерою «B» необхідно вказати правильність відповіді знаком «+» або «–». Після зазначення правильності відповіді необхідно вказати текст варіанту відповіді в сусідній комірці під буквою «C».

Тип запитання вказується явно після ключового слова «Запитання». Наприклад, якщо було вказано тип «Radio» запитання вважається запитанням 1-го типу (рис. 3.3).

Рисунок 3.3 – Приклад відображення запитання з одним варіантом відповіді

При вказанні типу «Checkbox» запитання вважається запитанням 2-го типу з декількома варіантами відповіді (рис. 3.4).

Рисунок 3.4 – Приклад відображення запитання з декількома варіантами відповіді

Для того щоб зазначити запитання запитанням 3-го типу потрібно вказати тип «Text» без варіантів відповіді. Відображення даного типу запитань можна побачити з рисунку 3.5.

Рисунок 3.5 – Приклад відображення запитання з текстовим варіантом відповіді

Після заповнення файлу потрібно помістити його на смартфон та перейти на екран вибору файлу і натиснути на кнопку «Обрати файл» (рис. 3.6).

Рисунок 3.6 – Екран вибору файлу

Після натискання кнопки вибору відкривається системний екран вибору файлу, коли файл було обрано, здійснюється парсинг файлу та відбувається відправка даних до серверу. Після успішної відправки даних тест починає відображатись в клієнт версії додатку. Для відкриття тесту

потрібно авторизуватись в додатку як студент та відкрити екран з доступними тестами, який проілюстровано на рисунку 3.7.

Рисунок 3.7 – Екран вибору доступних тестів

Даний екран також доступний в хост версії застосунку. Однак при натисненні на тест в клієнт версії відбувається відкриття екрану з питаннями до тесту, а в хост версії відбувається навігація до екрану з інформацією щодо тесту (рис. 3.8). На екрані з деталями тесту відображається список студентів, які проходять, або вже пройшли тест. На даному екрані можна завершити тест та згенерувати звіт.

Рисунок 3.8 – Екран інформації тесту

4 ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ МОБІЛЬНОГО ТЕСТУВАННЯ

4.1 Загальна архітектура системи

Ядром системи що розробляється є архітектура «клієнт-сервер», в якій завдання або мережева навантаження розподілені між постачальниками послуг, які називаються серверами, і замовниками послуг, так званисм клієнтами.

Клієнтом є мобільний додаток користувача, де за допомогою інструментів для розробки андроїд додатків «Android SDK» та мови програмування «Kotlin» забезпечується взаємодія користувача з даними. Як середовище взаємодії клієнта з сервером використовується Інтернет.

Основними перевагами архітектури «клієнт-сервер» є:

– можливість розподілити функції обчислювальної системи між декількома незалежними комп’ютерами в мережі, це дозволяє спростити обслуговування обчислювальної системи, зокрема, заміна, ремонт, модернізація або переміщення сервера, не зачіпають клієнтів;

– всі дані зберігаються на сервері, який як правило, захищений набагато краще за більшість клієнтів. на сервері простіше забезпечити контроль повноважень, щоб дозволити доступ до даних тільки клієнтам з відповідними правами доступу;

– використовувати ресурси одного сервера можуть клієнти з різними апаратними платформами, операційними системами.

Основні недоліки:

– у разі використання централізованої системи, виведення з ладу основного сервера може зробити непрацездатним весь додаток;

– адміністрування даної інформаційної системи вимагає кваліфікованого професіонала;

– висока вартість обладнання.

Для забезпечення роботи «серверної» частини додатку виступатиме платформа «Firebase». «Firebase» дозволяє забезпечити доступ користувачів до системи за допомогою методу авторизації, а також дозволяє безкоштовно створювати базу даних.

Формування тестів здійснюватиметься на ПК у вигляді «Excel» файлів. Після завершення формування тестового файлу, він вивантажується на платформу «Firebase», разом з файлом ПІБ студентів, яка також забезпечує можливість зберігання файлів. Клієнт частина, мобільний додаток, завантажує файли та проводить їх аналіз. Вхід в мобільний додаток забезпечуватиметься через авторизацію в додатку за допомогою поштової адреси з доменом «@nure.ua».

4.2 Розробка візуального інтерфейсу за допомогою засобів Android SDK

Графічний інтерфейс користувача є ієрархією об’єктів android.view.View і android.view.ViewGroup. Кожен об’єкт ViewGroup представляє собою контейнер, який містить і впорядковує дочірні об’єкти View. Зокрема, до контейнерів відносять такі елементи, як RelativeLayout, LinearLayout, GridLayout, ConstraintLayout і ряд інших.

Прості об’єкти View представляють собою елементи керування та інші віджети, наприклад, кнопки, текстові поля і т. д., через які користувач взаємодіє з програмою (рисунок 4.1).

Рисунок 4.1 – Ієрархія віджетів в Android SDK

Зазвичай, при розробці додатків, віджетів, які надає Android SDK, буває недостатньо. Наприклад, при старті додатку «NURE Tests» необхідно

відобразити логотип ХНУРЕ, який поступово анімуватиметься. В даному випадку постає необхідність у створенні власного віджету.

Як і в Activity та Fragment у кожного представлення є методи життєвого циклу, які знаходяться в класі View (рисунок 4.2).

Кожен View-компонент починається з конструктора. І це дає відмінну можливість підготувати його, роблячи різні обчислення, встановлюючи значення за замовчуванням, або взагалі робити все що потрібно для подальшого існування представлення. В класі View є чотири конструктори:

– View(Context);

– View(Context, AttributeSet);

– View(Context, AttributeSet, int);

– View(Context, AttributeSet, int, int).

Кожен конструктор, з перелічених, першим параметром приймає аргумент Context.

Рисунок 4.2 – Методи життєвого циклу View

Context – це абстрактний клас, реалізацію якого забезпечує система Android. Він представляє собою інтерфейс для доступу до глобальної інформації середовища додатку. Context надає доступ до ресурсів, які стосуються конкретного додатку, а також до викликів операцій на рівні додатків, таких як запуск діяльності, трансляції та отримання намірів, тощо [15].

AttributeSet, параметр який є присутнім в трьох останніх конструкторах, є колекцією яка зберігає в собі всі атрибути, які були призначені даному представленню в XML файлі розмітки.

Параметром int в третьому та четвертому конструкторах є код стилю, який зазначається в атрибуті «style» xml файлу розмітки. Останній параметр типу int в четвертому конструкторі це також ідентифікатор стилю, який використовується у випадку, якщо ідентифікатор стилю з атрибутів не був вказаний та дорівнює нулю.

Після того як батьківський View-компонент викличе метод addView(View), викликається метод onAttachedToWindow і View-компонент, який був переданий в аргумент методу addView буде прикріплений до вікна. На цій стадії View-компонент буде знати про інші View-компоненти, які його оточують. Якщо View-компонент працює з View-компонентами користувача, розташованими в тому ж самому xml файлі, то це гарне місце для того щоб знайти їх за ідентифікатором (який можна встановити за допомогою атрибутів) і зберегти їх в якості глобальної посилання (якщо є така необхідність).

Метод onMeasure означає, що Сейчас View-компонент знаходиться на стадії визначення власного розміру. Це дуже важливий метод, так як в більшості випадків потрібно визначити специфічний розмір для View-компонента, щоб той помістився на макеті.

Метод onLayout надає можливість присвоїти розмір і позицію дочернім View-компонентам.

У методі onDraw проводиться рисування View-компонента. Два об’єкти, Canvas і Paint, дозволяють намалювати все що необхідно. Екземпляр об’єкту Canvas приходить як параметр в методі onDraw, і по суті відповідає за малювання різних фігур, в той час як об’єкт Paint відповідає за колір цієї фігури.

Для View-компоненту базовим класом слугуватиме клас View, адже він найкраще підходить для конкретної задачі.

В методі init() (лістинг 4.1), який викликається в конструкторі, виконується ініціалізація таких полів як: strokeWidth (товщина лінії), animationDuration (тривалість анімації) та paint типу Paint. Якщо ж аргумент attrs типу At-tributeSet, не дорівнює null, то виконується витяг значень які

були ініційовані в атрибутах xml файлу розмітки. В поле paint задаються такі параметри, як: колір, товщина лінії, та стиль, в даному випадку використовується стиль Paint.Style.STROKE. Вкінці методу виконується виклик fillLinesInfo().

Лістинг 4.1 – Метод init()

```
private fun init(attrs: AttributeSet?) {

var strokeColor = DEFAULT_STROKE_COLOR


attrs?.apply {

      val typedArray = context.obtainStyledAttributes(this,
      R.styleable.NureLogoView)

      strokeWidth =

      typedArray.getDimension(R.styleable.NureLogoView_stroke_width,
      DEFAULT_STROKE_WIDTH)

      isProgressAnimation =

      typedArray.getBoolean(R.styleable.NureLogoView_progress_animation,
      false)

      strokeColor =

      typedArray.getColor(R.styleable.NureLogoView_stroke_color,
      DEFAULT_STROKE_COLOR)

      animationDuration =

      typedArray.getInteger(R.styleable.NureLogoView_animation_duration,

      DEFAULT_ANIMATION_DURATION.toInt()).toLong()

      typedArray.recycle()

      }

paint.color = strokeColor

paint.strokeWidth = strokeWidth

paint.style = Paint.Style.STROKE


animator.addListener(onEnd = { animationEndListener?.invoke() })


fillLinesInfo()

}
```

Лістинг 4.2 – Метод fillLinesInfo

```
private fun fillLinesInfo() {
```

```
    linesInfo.add(Pair(0, 0.36f))
    linesInfo.add(Pair(60, 0.77f))
    linesInfo.add(Pair(0, 0.6f))
    linesInfo.add(null)
    linesInfo.add(Pair(0, -0.5f))
    linesInfo.add(Pair(-55, -1.1f))
    linesInfo.add(Pair(0, 1.48f))
}
```

В методі fillLinesInfo (лістинг 4.2) здійснюється вставка інформації щодо довжини та куту кожної лінії логотипу. В класі View-компонента, NureLogoView, є поле linesInfo типу ArrayList<Pair<Int, Float>>, яке міс-тить в собі пари зі значеннями довжини та куту кожної лінії і в даному ме-тоді виконується вставка цих значень в колекцію.

Лістинг 4.3 – Метод fillPath

```
private fun fillPath() {
   radius = (min(measuredWidth, measuredHeight) - strokeWidth * 2) / 2f
   val centerPoint = PointF()
   val startRadianAngle = degreeToRadians(214)
   centerPoint.x = measuredWidth / 2f
   centerPoint.y = measuredHeight / 2f
   currentPoint.x = centerPoint.x + (radius * cos(startRadianAngle))
   currentPoint.y = centerPoint.y + (radius * sin(startRadianAngle))
   path.moveTo(currentPoint.x, currentPoint.y)
     for (line in linesInfo) {
         if (line != null) {
             updateCurrentPoint(line.first, radius * line.second)
             path.rLineTo(currentPoint.x, currentPoint.y)
         } else { drawBezier() }
    }
    drawCircle()
    val measure = PathMeasure(path, false)
    measure.setPath(path, true)
```

```
        pathLength = measure.length
    }
```

В методі onLayput виконується виклик методу fillPath (лістинг 4.3), в якому будується шлях для рисування логотипу. На початку методу вираховується довжина радіусу, початкова та центральні точки. Потім викликається метод moveTo(Int, Int), екземпляру класу path типу Path, який є полем представлення, що приймає координати точки. Виклик здійснюється для того, щоб встановити шлях на початкову точку. Після чого здійсню-ється прохід циклом по всім елементам колекції linesInfo та виконується переміщення шляху, поступово будуючи лінії за допомогою методу path.lineTo. Вкінці виконується виклик методу drawCircle() для побудови овалу, а також здійснюється вимір довжини шляху, який знадобиться для анімації.

Лістинг 4.4 – метод onDraw

```
override fun onDraw(canvas: Canvas?) {
      super.onDraw(canvas)
      canvas?.drawPath(path, paint)
}
```

В методі onDraw (лістинг 4.4) здійснюється виклик методу canvas?.drawPath(Path, Paint), в який передаються раніше проініціалізовані поля path та paint.

Рисунок 4.3 – Результат роботи коду, View-представлення логотипу

При авторизації в додатку користувачеві необхідно здійснити вхід в

систему за допомогою акаунту з доменом «nure.ua» (рис. 4.4).

Рисунок 4.4 – Екран авторизації

Рисунок 4.5 – Діалогове вікно вибору акаунту

Авторизація в системі відбувається за допомогою Google OAuth API. Даний інструмент дозволяє інтегрувати до застосунків логіку отримання інформації про користувача, використовуючи акаунти Google, доступні акаунти на пристрої є проілюстровано на рисунку 4.5. Для того, щоб інтегрувати використання API в проекті, необхідно додати залежність до гредл файлу проекту (лістинг 4.5). Також необхідно зареєструвати проект на сайті «https://developers.google.com/».

Лістинг 4.5 – Код залежності Google OAuth API

```
apply plugin: 'com.android.application'
...

dependencies {
   implementation 'com.google.android.gms:play-services-auth:20.0.0'
}
```

Google OAuth API також містить в собі елементи представлення, що можна побачити з рисунку 4.4 на прикладі кнопки «Sign in». XML файл розмітки екрану логіну складається з таких елементів як: логотип ХНУРЕ, назва додатку та кнопка логіну Google OAuth (лістинг 4.6). Крім того також присутня логіка програмної складової екрану авторизації.

Лістинг 4.6 – XML розмітка логін екрану fragment_login.xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:background="@drawable/bg_gradient"

    android:orientation="vertical"

    android:padding="24dp">


    <com.google.android.gms.common.SignInButton

        android:id="@+id/signInButton"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        app:layout_constraintBottom_toBottomOf="parent"

        app:layout_constraintTop_toBottomOf="@+id/logoLL"

        app:layout_constraintStart_toStartOf="parent"

        app:layout_constraintEnd_toEndOf="parent"

        tools:layout_editor_absoluteX="24dp"/>


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

В класі LoginFragment присутня властивість типу GoogleSignInClient. Це інструмент, який постачається в бібліотеці Google OAuth API. В методі життєвого циклу onViewCreated, відбувається його ініціалізація (лістинг 4.7). Щоб отримати його екземпляр – необхідно викликати метод GoogleSignIn.getClient, який приймає два параметри: екземпляр активності та екземпляр класу GoogleSignInOptions. GoogleSignInOptions визначає, яку

інформацію необхідно вказати користувачу для здійснення аутентифікації, в даному випадку це опції за замовчуванням GoogleSignInOptions.DEFAULT_SIGN_IN.

Лістинг 4.7 – Метод onViewCreated

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?){
        super.onViewCreated(view, savedInstanceState)
        val gso =
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestEmail()
            .build()

     googleSignInClient = GoogleSignIn.getClient(requireActivity(), gso)
        initListeners()
    }
```

В методі onViewCreated також присутній виклик методу initListeners. В методі initListeners() відбувається звернення до екземпляру представлення кнопки логіну з бібліотеки Google, та відбувається виклик методу setOnClickListener. Даний метод використовується для того, щоб здійснити прослуховування взаємодії користувача з кнопкою. Тобто, у випадку, коли користувач натискатиме кнопку, відбуватиметься виконання коду, який знаходиться в тілі метода, що передається до setOnClickListener. В даному випадку здійснюватиметься виклик методу signIn.

Метод signIn використовується для запуску активності логіну, яка, також, є складовою бібліотеки Google OAuth (лістинг 4.8). Для запуску окремої активності потрібно створити екземпляр класу Intent. Намір (Intent) – це механізм для опису однієї операції – вибрати фотографію, надіслати листа, зробити дзвінок, запустити браузер та перейти за вказаною адресою. В Android-додатках багато операцій працюють через наміри. В методі signIn створення екземпляру наміру здійснюється шляхом виклику

googleSignInClient.signInIntent. Після отримання наміру відбувається виклик методу startActivityForResult.

Лістинг 4.8 – Метод signIn

```
private fun signIn() {
        val signInIntent: Intent = googleSignInClient.signInIntent
        startActivityForResult(signInIntent, RC_SIGN_IN)
    }
```

Метод startActivityForResult здійснює запуск активності від якої очікується повернення результату. Аргументами методу є: намір (intent) та код запиту (request code). Після того, як було обрано акаунт (рис. 4.5) результат отримується в методі onActivityResult. Параметрами даного методу є: код запиту (request code), код результату (result code) та намір (intent) який містить в собі дані про активність, з якої було отримано результат. В методі здійснюється перевірка коду запиту, та якщо він збігається з тим, який було передано до startActivityForResult – відбувається отримання інформації про акаунт за допомогою статичного методу GoogleSignIn.getSignedInAccountFromIntent. Після чого відбувається виклик методу handleSignInResult. В методі handleSignInResult відбувається запуск екрану HomeFragment.

Лістинг 4.9 – Метод onActivityResult

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == RC_SIGN_IN) {
            val task: Task<GoogleSignInAccount> =
GoogleSignIn.getSignedInAccountFromIntent(data)
            handleSignInResult(task)
```

```
            }
        }

        private fun handleSignInResult(completedTask:
Task<GoogleSignInAccount>) {
            try {
                // Signed in successfully, show authenticated UI.
                requireActivity().supportFragmentManager
                    .beginTransaction()
                    .replace(android.R.id.content,
HomeFragment.newInstance())
                    .commit()
            } catch (e: ApiException) {
                Log.w("TAG", "signInResult:failed code=" + e.statusCode)
            }
        }
```

Рисунок 4.6 – Домашній екран HomeFragment

Лістинг 4.10 – XML розмітка домашнього екрану fragment_home.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout android:id="@+id/drawer"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:openDrawer="start"
    android:fitsSystemWindows="true"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
```

```
            <androidx.fragment.app.FragmentContainerView
                android:id="@+id/containerFL"
                android:layout_marginTop="12dp"
                android:background="@color/colorLightGreyBg"
                android:layout_width="match_parent"
                android:layout_height="match_parent"/>

        </LinearLayout>

        <com.google.android.material.navigation.NavigationView
            android:id="@+id/roomsNV"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            app:headerLayout="@layout/layout_drawer_header"
            app:menu="@menu/drawer_menu"
            android:theme="@style/NavigationDrawerStyle"
            app:itemIconTint="@color/colorGrey"
            app:itemTextColor="@color/colorGrey"
            android:layout_gravity="start"
            android:fitsSystemWindows="true" />
    </androidx.drawerlayout.widget.DrawerLayout>
```

Домашній екран є основою функціоналу, який відкривається після логіну. Він складається з бокового меню (drawer layout), яке дозволяє перемикати екрани. Меню містить два пункти: екран активних тестів та екран вивантаження excel файлу до серверу. Перед вивантаженням даних до серверу, відбувається локальний парсинг даних файлу. Для здійснення парсингу використовується бібліотека Apache POI.

Apache POI дозволяє підтримувати API Java для маніпулювання різними форматами файлів на основі стандартів Office Open XML (OOXML) і формату складеного документа Microsoft OLE 2 (OLE2). За допомогою даної бібліотеки є можливість читати та записувати файли MS Excel за допомогою Java або Kotlin. Крім того, вона дозволяє читати та записувати файли MS Word і MS PowerPoint.

Для розділення обов’язків було створено окремий клас для парсингу ExcelFileParser.

Лістинг 4.11 – Реалізація методу parseExcelFile

```
fun parseExcelTestFile(uri: Uri): Test {
        val inputStream =
context.contentResolver.openInputStream(Uri.parse(uri.toString()))
        val currentSheet =
WorkbookFactory.create(inputStream).getSheetAt(0)
        val description = parseTestDescription(currentSheet)
        val topic = parseTestTopic(currentSheet)
        val questions = parseQuestions(currentSheet)

        return Test(description, topic, questions)
    }
```

Точкою входу в клас ExcelFileParser слугує метод parseExcelFile (лістинг 4.11). Його параметром є Uri. Uri – це посилання на файл, який було обрано на екрані вибору файлу. В методі відкривається потік для зчитування даних з файлу, що відбувається за допомогою метода contentResolver.openInputStream(uri). Після отримання потоку даних файлу, здійснюється отримання таблиці з файлу під індексом 0, WorkbookFactory.create(inputStream).getSheetAt(0). Отримана таблиця передається до методів parseTestDescription, parseTestTopic та parseQuestions для парсингу опису тесту, теми та питань відповідно. Результатом виконання методу є повернення об’єкту Test з темою, описом та питаннями тесту.

ВИСНОВКИ

В ході написання дипломної роботи було проведено аналіз доступних рішень для тестування здобувачів освіти на ринку інформаційних технологій. Під час дослідження було виявлено, що велика кількість додатків має недостатній функціонал, такий як: авторизація користувачів в системі, відсутність безоплатної ліцензії для користування додатками, а також зайвий функціонал. В результаті було прийняте рішення розробити систему для тестування у вигляді мобільного застосунку. В якості основної операційної системи було обрано Android, через велику розповсюдженість серед сучасних користувачів. Системою для зберігання та обробки даних щодо груп, ідентифікаторів студентів, які проходитимуть тестування, та файлів тестування було обрано платформу Firebase, яка добре зарекомендувала себе в сфері розробки мобільних додатків.

Для забезпечення безпеки доступу авторизації було розроблено систему верифікації за електронною адресою, за допомогою інструментів аутентифікації користувачів Google OAuth.

Під час розробки додатку були вивчені інструменти для роботи з платформою Firebase, та закріплені вміння роботи з прикладним інтерфейсом REST API, робота з розробкою кастомних представлень для користувацького інтерфейсу та здобуті практичні вміння в проектуванні архітектури MVVM.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Филлипс, Б. Android. Программирование для профессионалов. 3-е издание [Текст] / Б. Филлипс, К. Стюарт, К. Марсикано – СПб.: Питер, 2017. – 688 с.

2. Жемеров, Д. Kotlin в действии [Текст] / Д. Жемеров, С. Исакова – М.: ДМК Пресс, 2016. – 402 с.

3. Шкіль О.С. Комп'ютерна система тестування OpenTEST2 [Текст] / О.С. Шкіль, В.І. Каук, С.В. Напраснік, Є.С. Цимбалюк, О.А. Щербаков // Вісник. Тестування і моніторинг в освіті. – 2008. – № 2. – С.35–41.

4. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения [Текст] / Р. Мартин – СПб.: Питер, 2016. – 352 с.

5. Приемы объектно-ориентированного проектирования [Текст] / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес – СПб.: Питер, 2013. – 368 с.

6. Макконнелл, С. Совершенный код. Мастер-класс [Текст] / С. Макконнелл – М.: Русская редакция, 2013. – 896 с.

7. Офіційний сайт документації платформи Firebase [Електронний ресурс] / Режим доступу: www/ URL: https://firebase.google.com/docs/ – 09.05.2020 р. – Загл. з екрану.

8. Офіційний сайт документації Android SDK [Електронний ресурс] / Режим доступу: www/ URL: https://developer.android.com/ – 09.05.2020 р. – Загл. з екрану.

9. Нуркевич, Т. Реактивное программирование с использованием RxJava [Текст] / Т. Нуркевич, Б. Кристенсен – М.: Дмк-Пресс, 2017. – 358 с.

10. Дарвин, Ян. Android. Сборник рецептов: задачи и решения для разработчиков приложений. 2-е издание [Текст] / Ян Дарвин – К.: Диалектика, 2018. – 768 с.

11. Еванс, Е. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем [Текст] / Е. Еванс – К.: Диалектика, 2016. – 448 с.

12. Вигерс, К. И. Разработка требований к программному обеспечению. стереотипное 3-е издание [Текст] / К. И. Вигерс, Б. Джой – Спб.: БХВ-Петербург, 2017. – 736 с.

13. Клифтон Я. Проектирование пользовательского интерфейса в Android [Текст] / Я. Клифтон – М.: ДМК Пресс, 2017. – 452 с.

14. Ерансон А. Эффективное использование потоков в операционной системе Android [Текст] / А. Ерансон – М.: ДМК Пресс, 2017. – 304 с.

15. Аделекан И. Kotlin. Программирование на примерах [Текст] / И. Аделекан – СПб.: БХВ-Петербург, 2021. – 432 с.