Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів обміну даними між платформою Android та серверною частиною
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-4

Кондратьєв М. А.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник доц. Голян В.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кондратьєву Микиті Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів обміну даними між платформою Android та серверною частиною»
Затверджена наказом по університету від 29.03. 2024р. № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 19.06.2024
3. Вихідні дані до роботи опис досліджуваних методів взаємодії між платформою Android та сервером, вимоги до розробки схеми бази даних для проведення досліджень за обраною предметною областю, мови програмування Java та Kotlin, СУБД MySQL, середовища розробки IntelliJ IDEA та Android Studio
4. Перелік питань, що потрібно опрацювати в роботі
аналіз та порівняння існуючих методів обміну даними між платформою Android та серверною частиною, вибір підходящих методів для дослідження, написання програмних рішень, проведення експериментів та аналіз отриманих результатів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	23.01 – 14.02.24	*виконано*
2	Аналіз та вибір API для дослідження	15.02 – 24.02.24	*виконано*
3	Аналіз та моделювання предметної області	17.02 – 28.02.24	*виконано*
4	Планування експериментів	25.02 – 28.02.24	*виконано*
5	Програмна реалізація кожного з обраних для дослідження API	25.02 – 01.04.24	*виконано*
6	Експериментальні дослідження	02.04 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04 – 23.04.24	*виконано*
8	Написання та оформлення статті та тез доповіді	17.04 – 23.04.24	*виконано*
9	Підготовка пояснювальної записки	01.04 – 26.04.24	*виконано*
10	Підготовка презентації та доповіді	26.04 – 1.05.24	*виконано*
11	Нормоконтроль	3.05 – 09.05.24	*виконано*
12	Рецензування	08.05 – 14.05.24	*виконано*
13	Занесення диплома в електронний архів	15.05.2024	*виконано*
14	Попередній захист	15.05.2024	*виконано*
15	Допуск до захисту у зав. кафедри	17.05.2024	*виконано*

Дата видачі завдання 30 березня 2024р.

Студент (ка) _____ Кондратьєв М. А.
(підпис)

Керівник роботи _____ доц. Голян В.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 57 с., 31 рис., 7 табл., 8 джерел.

ДОСЛІДЖЕННЯ, КЛІЄНТ-СЕРВЕРНА ВЗАЄМОДІЯ, ОБМІН ДАНИМИ, СЕРВЕР, ANDROID, HTTP, JSON.

Об’єктом дослідження є сучасні способи обміну даними між платформою Android та серверною частиною.

Метою роботи є проведення дослідження способів обміну даними, платформи Android та її взаємодії з сервером.

Методами розробки та проектування є аналіз проблемної області дослідження, вибір Android API для проведення дослідження шляхом вирішення багатокритеріальної задачі прийняття рішень.

У результаті кваліфікаційної роботи було розроблено серверну частину та Android-застосунок, та налаштовано взаємодію між ними.

RESEARCH, CLIENT-SERVER INTERACTION, DATA EXCHANGE, SERVER, ANDROID, HTTP, JSON.

The object of research is modern ways of data exchange between the Android platform and the server part.

The aim of this paper is to study the methods of data exchange, the Android platform and its interaction with the server.

Development and design methods are the analysis of the problem area of research, the selection of the Android API for conducting research by solving a multi-criteria decision-making problem.

As a result of the qualification work, the server part and the Android application were developed, and the interaction between them was configured.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Кондратьєв Микита Андрійович, студент гр. ІПЗм-22-4, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів обміну даними між платформою Android та серверною частиною», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

В наш час більша частина програм, які люди звикли використовувати повсякденно, є Android застосунками, якими дуже зручно користуватися за допомогою телефона.

Із розвитком технологій, застосунки, поступово, змінювали свою архітектуру та методи збереження даних. Зараз велика кількість застосунків використовують клієнт-серверну архітектуру.

Клієнт-серверна архітектура, дозволила вирішити багато проблем, які раніше виникали під час реалізації різних функцій.

Використання даної архітектури, також дозволило синхронізувати данні між різними клієнтами.

Однак з’явилася проблема передачі даних між платформою Android та серверною частиною.

У даному дослідженні було вирішено дослідити різні сучасні способи взаємодії між Android та сервером, та способи обміну даними.

Сьогодні існує досить велика кількість методів та бібліотек, які допомагають облегшити цей процес.

Усі вони відрізняються як технологією передачі даних, так і структурними особливостями.

Вибрати правильний спосіб передачі даних може бути досить складною задачею.

Тому це дослідження спрямоване на визначення сильних та слабих сторін усіх сучасних способів обміну даними між платформою Android та сервером.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз платформи Android

Android є найпоширенішою операційною системою та платформою для додатків, з більш ніж двома мільярдами активних користувачів. Вона підтримує різні пристрої, починаючи від розумних годинників та телевізорів до ноутбуків і автомобілів. Але найбільш часто використовується на смартфонах і планшетах.

Android являє собою відкритий проект. Більшість вихідного коду поширюється під вільною ліцензією Apache 2.0.

Компанія Android Inc. була заснована в 2003 році і придбана Google в 2005 році.

Публічна бета-версія Android вийшла в 2007 році, а перша стабільна версія – в 2008 році. З тих пір значні оновлення випускаються приблизно раз на рік.

Операційна система Android базується на зміненій версії ядра Linux LTS, яке безпосередньо взаємодіє з обладнанням.

Виробники обладнання розробляють драйвери, необхідні для роботи пристроїв, і впроваджують їх в ядро. Цей підхід дозволяє їм створювати драйвери для широко відомого ядра, тоді як розробники операційної системи можуть ігнорувати різноманітність обладнання.

Додатково, особливості обладнання приховані виробниками з використанням рівнів апаратної Абстракції. Ці рівні забезпечують стандартні інтерфейси для структур високого рівня, забезпечуючи доступ до апаратного забезпечення пристрою без необхідності дбати про реалізацію драйверів.

Android Runtime (ART) являє собою віртуальну машину, яка виконує код додатків, що міститься в файлах Dalvik Executable (DEX). ART керує компіляцією коду, налагодженням та управлінням пам'яттю.

Кожна програма працює у власному екземплярі ART, забезпечуючи ізоляцію коду. ART замінив Dalvik як віртуальну машину Java для Android у 2013 році завдяки компіляції Ahead-of-Time, що забезпечило більш високу продуктивність порівняно з компіляцією Just-in-Time.

Для створення додатків для Android можна використовувати мови програмування Kotlin, Java і c++. Інструменти Android SDK компілюють ваш код разом із необхідними даними та ресурсами у формат APK або Android app Bundle.

В системі Android реалізований принцип найменших привілеїв, що означає, що кожен додаток спочатку має доступ тільки до тих компонентів, які необхідні для його функціонування, і не має доступу до більш широким системних ресурсів без відповідного дозволу. Це забезпечує високий рівень безпеки, запобігаючи доступу програми до частин системи без відповідних повноважень.

Життєвий цикл програми в Android суворо регулюється системою і залежить від запитів користувачів, наявності доступних ресурсів та інших факторів.

Життєвий цикл програми в Android представлено на рисунку 1.1.

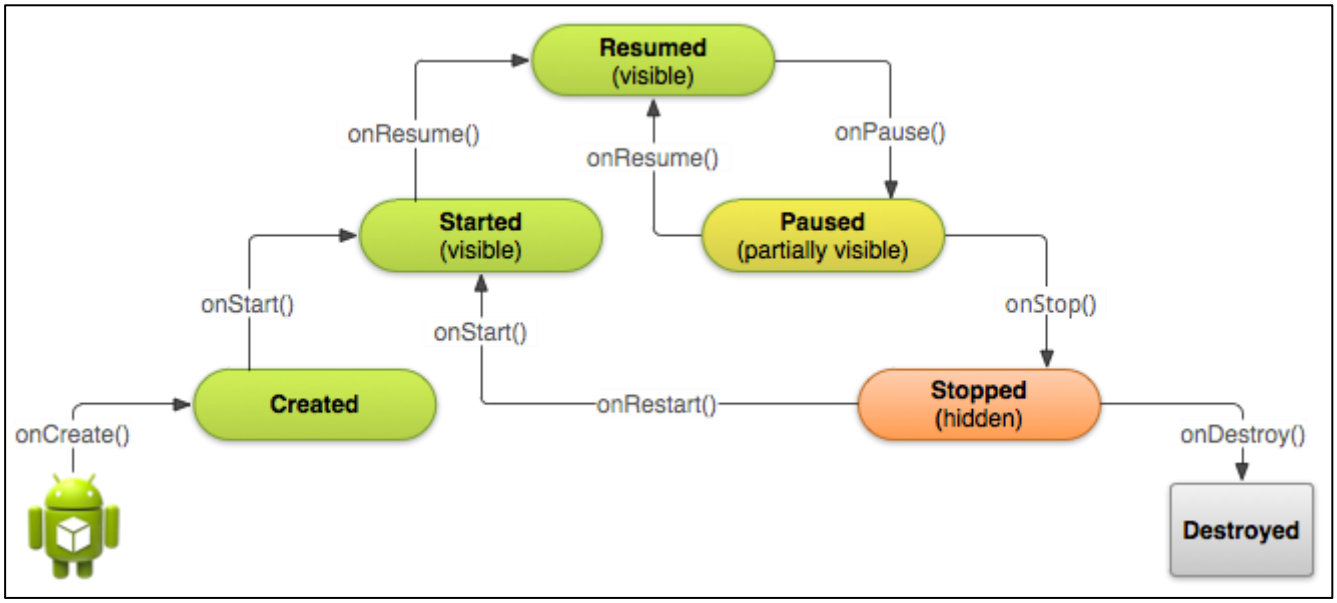

Рисунок 1.1 – Життєвий цикл програми в Android

Приведемо приклад. Якщо користувач висловлює бажання запустити веб-браузер, рішення про запуск програми приймається системою.

Незважаючи на те, що остаточне рішення залишається за системою, вона діє відповідно до певних логічних правил, що визначають, чи можна завантажити, призупинити або завершити роботу програми.

Якщо Користувач зараз взаємодіє з певним вікном, система надає пріоритет цій програмі. Навпаки, якщо вікно стає невидимим, і система визначає, що необхідно призупинити роботу програми для вивільнення додаткових ресурсів, додаток з більш низьким пріоритетом може бути завершено. Через обмеженість ресурсів в Android система більш строго управляє виконанням додатків.

Компоненти програми являють собою основні будівельні блоки Android-Додатки. Кожен компонент є точкою входу, через яку система або Користувач можуть взаємодіяти з вашим додатком.

Ці компоненти включають в себе:

– activities (активності);
– services (сервіси);
– broadcast receivers (приймачі широкомовних повідомлень);
– content providers (постачальники контенту).

Кожен тип компонента служить певній меті і має власний життєвий цикл, який визначає, як компонент створюється та знищується.

1.2 Головні відомості про клієнт-серверну архітектуру

Архітектура клієнт-сервер - це організація обчислювальних або мережевих процесів, при якій завдання або мережеве навантаження розподіляються між постачальниками послуг, відомими як сервери, та одержувачами послуг, яких називають клієнтами.

Сутності "клієнт" і "сервер" в даному контексті являють собою програмне забезпечення.

Зазвичай ці програми розміщуються на різних обчислювальних пристроях і взаємодіють між собою через обчислювальні мережі, використовуючи мережеві протоколи.

Однак вони також можуть бути розміщені на одному пристрої.

Програми-сервери готові обробляти запити від клієнтських програм і надавати їм свої ресурси у формі даних або сервісних функцій.

Оскільки одна серверна програма може обслуговувати запити від безлічі клієнтських програм, вона часто розміщується на спеціально виділеному обчислювальному пристрої, оптимізованому для ефективної роботи з іншими серверами.

В силу цього і особливостей її обладнання та програмного забезпечення, таку машину прийнято називати сервером, а пристрої, що виконують клієнтські програми, відповідно, клієнтами.

Архітектура "клієнт-сервер" являє собою один з ключових принципів функціонування інтернет-мережі.

Будь-який веб-сайт або додаток в Інтернеті оперує на сервері, виступаючи в ролі постачальника послуг, в той час як користувачі є клієнтами.

Соціальні мережі, інтернет-магазини, мобільні додатки і пристрої Інтернету речей функціонують на основі клієнт-серверної архітектури.

Прикладом ефективної роботи системи "клієнт-сервер" є автомобільний навігатор.

Додаток навігації, що працює на сервері, збирає дані від безлічі смартфонів користувачів, на яких встановлені клієнтські додатки. Крім того, воно використовує дані з сервера геоінформаційної системи, такі як інформація про поточні ремонти доріг і появі нових дорожніх маршрутів.

Інформація від різних клієнтів, така як місце розташування і швидкість, обробляється сервером навігації, після чого надається користувачам у вигляді даних про середню швидкість руху по певних ділянках маршруту на їх смартфони.

Архітектура "клієнт-сервер" є основою більшості корпоративних мереж і бере свій початок від найперших обчислювальних машин, так званих "мейнфреймів".

1.3 Аналіз серверної частини клієнт-серверної архітектури

Серверна частина в клієнт-серверній архітектурі являє собою сегмент системи, який функціонує на сервері і забезпечує обробку запитів від Клієнтів.

Цей компонент виконує ключові функції, необхідні для надання послуг клієнтам та ефективної взаємодії в рамках розподіленого мережевого середовища.

Основні аспекти серверної частини в архітектурі клієнт-сервер включають обробку запитів, зберігання даних, бізнес-логіку, аутентифікацію та безпеку, масштабованість і продуктивність та взаємодію з іншими службами.

Обробка запитів: серверна частина відповідає на запити від клієнтів, надаючи необхідні дані, послуги або ресурси.

Вона є центральним виконавцем, який обробляє дії, запитані клієнтами.

Зберігання даних: Сервер зберігає дані, які використовуються додатком або клієнтами.

Це може включати бази даних, сховища файлів та інші механізми ефективного управління інформацією.

Бізнес-логіка: в серверній частині реалізується бізнес-логіка додатки.

Тут визначаються правила обробки даних, логіка взаємодії та основні функції, необхідні для роботи системи.

Аутентифікація та безпека: сервер відповідає за автентифікацію користувачів та забезпечення безпеки даних.

Це включає перевірку облікових даних, управління доступом та шифрування даних для захисту конфіденційності.

Масштабованість і продуктивність: ефективна робота серверної частини забезпечує масштабованість системи, дозволяючи справлятися зі збільшенням навантаження і забезпечуючи високу продуктивність при обробці запитів.

Взаємодія з іншими службами: серверна частина може взаємодіяти з іншими зовнішніми службами, API або системами для отримання додаткових даних або послуг.

Серверна частина є критичною частиною клієнт-серверної архітектури, забезпечуючи стабільність, надійність і ефективність роботи всієї системи.

Головним представленням серверної частини є сервер.

Сервер являє собою спеціальний комп'ютер з високою продуктивністю, зазвичай володіє найбільшими ресурсами, призначений для виконання різних завдань.

Його обов'язки включають зберігання інформації, забезпечення DNS (перетворення "буквених" адрес в IP), виконання DHCP (динамічний розподіл адрес) і так далі.

У контексті архітектури "клієнт-сервер" сервер відповідає на запити клієнта, надаючи йому свої ресурси за потреби.

Розглянемо сервер у контексті мобільних додатків.

Бекенд мобільного додатка, або серверна частина, являє собою сегмент програми, що виконується на віддаленому сервері, а не на пристрої користувача. Цей компонент надає функціональні можливості, необхідні для зберігання та управління даними, аутентифікації користувачів, обробки та аналізу даних, а також взаємодії з іншими системами або службами.

Серверна частина включає різні елементи, такі як сервер у хмарному середовищі, база даних або API.

Її мета-забезпечити додатку необхідну інфраструктуру та інструменти для ефективного функціонування, що в свою чергу дозволяє надавати користувачам багатий і зручний досвід використання.

Бекенд мобільного додатка зазвичай надає наступні функціональні можливості:

- зберігання даних і управління ними;
- обробка та аналіз даних;
- аутентифікація та авторизація;
- синхронізація між багатьма пристроями;
- можливість інтеграції з іншими системами і сервісами;
- управління завданнями на стороні сервера;
- push-повідомлення;
- аналітика та моніторинг.

Через значні переваги, що надаються бекендом, майже у кожного мобільного додатку є своя серверна частина.

Це може бути як для зберігання даних, так і для виконання різних завдань, таких як аналітика, відправка push-повідомлень, моніторинг додатків, збір звітів про аварії і багатьох інших функцій.

1.4 Відомості про HTTP

Протокол HTTP розроблений для передачі даних в мережі Інтернет. Цей простий протокол використовує надійні служби TCP для передачі даних, що надає йому високий ступінь надійності в обміні контентом.

HTTP є одним з найбільш широко використовуваних протоколів додатків, і всі операції в Інтернеті здійснюються з його використанням.

HTTPS являє собою захищену версію протоколу HTTP, де застосовується протокол TLS для забезпечення безпеки базового TCP-підключення. За винятком необхідної конфігурації для впровадження TLS, застосування HTTPS фактично не відрізняється від використання звичайного протоколу HTTP.

Протокол HTTP має простий механізм запиту вмісту в Інтернеті. Доступний стандартний набір команд HTTP, які відправляються клієнтом після успішного встановлення підключення по стандартному TCP-порту 80 (або порту 443 для HTTPS).

Нижче наведено деякі основні команди HTTP:

- GET ресурс HTTP / 1.1: отримання зазначеного ресурсу;
- POST ресурс HTTP / 1.1: отримання зазначеного ресурсу і передача вкладених вхідних даних на HTTP-сервер;
- Head ресурс HTTP / 1.1: виконується так само, як GET, але HTTP-сервер не повертає вміст;
- PUT ресурс HTTP / 1.1: розміщення ресурсу на HTTP-сервері;
- DELETE ресурс HTTP / 1.1: видалення ресурсу з сервера.

Сервер HTTP використовує той самий TCP-порт 80 (або порт 443 для HTTPS) для надсилання відповідей на запити клієнта.

Після завершення обробки запиту сервер повертає рядок відповіді у форматі ASCII, що включає тризначний числовий код стану. Програмне забезпечення клієнта, аналізуючи цей числовий код, визначає, чи успішно була виконана операція або стався збій.

Нижче наведено список відповідей сервера HTTP на команди клієнта:

- 200: запит виконано успішно;
- 400: запит не сформований належним чином;
- 401: несанкціонований запит, клієнт повинен надіслати дані автентифікації;
- 404: вказаний у запиті ресурс не знайдено;
- 500: Внутрішня помилка сервера HTTP;
- 501: запит не реалізований сервером HTTP;
- 502: служба недоступна.

Наприклад, у відповідь на успішно виконаний запит PUT клієнта для файлу test.htm буде повернуто повідомлення "HTTP/1.1 200 OK".

Даний протокол часто використовують для обміну даними між клієнтом та сервером.

1.5 Відомості про обмін даними між сервером та Android клієнтом

Для реалізації взаємодії між сервером і мобільними пристроями під управлінням Android часто використовують різні бібліотеки.

Вибір конкретної бібліотеки залежить від вимог проекту, уподобань розробника та контексту використання в додатку Android.

Нижче наведено деякі з таких бібліотек:

- Retrofit;
- OkHttp;
- Volley;
- AsyncHttpClient.

Розглянимо усі дані бібліотеки докладніше.

Retrofit – це бібліотека від Square, призначена для роботи з мережевими запитами. Retrofit полегшує створення HTTP-запитів та обробку відповідей на стороні клієнта. Вона інтегрується з конвертерами для обробки форматів даних, таких як JSON.

Retrofit надає анотації для визначення кінцевих точок API, управління параметрами запитів і обробку відповідей у вигляді Java-об'єктів. Він також легко інтегрується з бібліотекою Gson для автоматичного перетворення даних JSON.

Retrofit часто використовується в Android-додатках для виконання HTTP-запитів і взаємодії з RESTful API.

OkHttp – також від Square. OkHttp надає потужні можливості для виконання мережевих запитів. Ця бібліотека може використовуватися окремо або в комбінації з Retrofit для більш низькорівневого управління HTTP-з'єднаннями.

OkHttp надає зручний API для надсилання мережевих запитів та обробки відповідей. Він підтримує перехоплювачі (interceptors), які дозволяють маніпулювати запитами і відповідями до і після їх відправки або отримання.

OkHttp може використовуватися в якості низькорівневої бібліотеки для управління HTTP-з'єднаннями, а також в комбінації з Retrofit для більш високорівневої роботи з мережею.

Розроблена Google, бібліотека Volley надає зручний та ефективний спосіб роботи з мережевими запитами. Вона підтримує асинхронні запити, кешування і обробку зображень, що робить її відмінним вибором для взаємодії з сервером на платформі Android.

Volley надає простий API для виконання мережевих запитів, управління кешуванням, а також автоматичну обробку паралельних запитів. Він оптимізований для роботи з частими, але невеликими запитами, такими як запити зображень або даних JSON.

Volley широко використовується для обробки мережевих запитів в Android-додатках, особливо у випадках, коли необхідна ефективна обробка зображень.

AsyncHttpClient – це легка бібліотека для виконання асинхронних запитів HTTP в Android. Вона надає зручний API і підтримує функції, такі як обробка куки, Автоматичне перепідключення і аутентифікація.

AsyncHttpClient надає простий API для виконання асинхронних HTTP-запитів. Він підтримує безліч функцій, таких як Автоматичне перепідключення, управління куками і підтримка аутентифікації.

Ця бібліотека часто використовується в Android-додатках, де потрібно виконання асинхронних мережевих запитів без блокування основного потоку.

Данні бібліотеки, найчастіше використовуються для обміну даними між сервером та мобільним застосунком на стороні Android. Однак, окрім даних бібліотек, є ще багато інших способів та методів.

У нашому дослідженні ми плануємо дослідити використання різних методів та бібліотек для обміну даними за різних умов, щоб визначити доцільність використання тих чи інших методів для досягнення різних цілей.

1.6 Постановка задачі

На сьогоднішній день клієнт-серверна архітектура є одною з найпопулярніших архітектур для створення програмного забезпечення, а платформа Android однією з найпопулярніших платформ для користувачів. При розробці Android застосунку, дуже велику увагу потрібно приділяти технологіям зв’язку між застосунком та серверною частиною проекту. Неправильний вибір способу обміну даними може призвести до збільшення часу очікування користувача та знизити швидкість роботи застосунку.

Способів обміну даними між платформою Android та серверною частиною, та їх реалізацій існує досить багато. Тому цікаво дослідити їх та визначити їх сильні та слабкі сторони. У даному дослідженні, ми хочемо визначити який із способів передачі даних та яку з реалізацій цих способів доцільно буде використовувати при різних обставинах.

Метою дослідження є порівняння різних сучасних способів та методів для обміну даними між платформою Android та сервером, та визначити доцільність їх використання у різноманітних обставинах.

Основна суть дослідження полягає в перевірці різних сучаних способів та методів обміну даними між платформою Android та серверною частиною на однакових даних.

Нашою метою є визначення, чи є виправданим використання тих чи інших способів та методів для такого обміну даними.

Задачі магістерського дослідження можна розкласти на наступні пункти:

- визначити способи обміну даними, які відповідають вимогам магістерського дослідження;
- підготувати серверну частину до проведення дослідження;
- підготувати Android застосунок до проведення магістерського дослідження;
- задокументувати процес розробки та підготовки кожної частини у текстовій записці магістерського дослідження;
- підготувати дані для обміну між сервером та Android застосунком;
- порівняти раніше визначені способи обміну даними на однакових даних з метою виявлення виправданості їх використання для обміну між платформою Android та серверною частиною;
- обґрунтувати висновки, отримані у результаті виконаного магістерського дослідження.

Перейдемо до обирання засобів проведення дослідження, аналізу предметної галузі та підготовки до проведення.

Магістерське дослідження буде проводитись за участю застосунку створеного на платформі Android та серверу із базою даних.

Android застосунок, буде написано з використанням мови програмування Kotlin.

Kotlin – це статично типізована мова програмування, яка працює на віртуальній машині Java та є офіційною мовою розробки для платформи Android.

Розроблена компанією JetBrains, Kotlin поєднує в собі експресивність та короткість коду з високою безпекою та розширеною функціональністю.

Ця мова отримала підтримку від Google для розробки Android-додатків, і її популярність продовжує зростати серед розробників.

Код буде написано у середовищі розробки Android Stodio.

Android Studio – це інтегроване середовище розробки для створення Android-додатків. Розроблене компанією JetBrains у співпраці з Google, Android Studio надає розробникам повноцінний набір інструментів для швидкого та ефективного створення мобільних застосунків для платформи Android.

Android Studio є офіційним інструментом для розробки на платформі Android та використовується розробниками для створення інноваційних та високоякісних мобільних додатків.

Сервер буде написано з використанням мови програмування Java.

Java – це універсальна, об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems. Її головною особливістю є можливість виконання на будь-якій платформі, яка підтримує віртуальну машину Java.

У якості середовища розробки буде використовуватися IntelliJ IDEA.

IntelliJ IDEA – це потужна інтегрована середовища розробки для роботи з різними мовами програмування, зокрема Java, Kotlin, Scala, Groovy та іншими. Розроблена компанією JetBrains, IntelliJ IDEA славиться своєю високою продуктивністю та набором інтелектуальних інструментів для підтримки розробки.

IntelliJ IDEA є популярним інструментом серед програмістів завдяки своїм продуктивним можливостям та інноваційним рішенням в галузі розробки програмного забезпечення.

Також для реалізації серверу буде використано фреймворк Spring.

Spring – це потужний фреймворк для розробки застосунків на мові Java. Створений для спрощення розробки та підтримки корпоративних Java-застосунків, Spring надає компроміс між ефективністю та гнучкістю, сприяючи створенню модульних, масштабованих та ефективних додатків.

Spring широко використовується в корпоративному середовищі для розробки різноманітних застосунків, і відомий своєю гнучкістю, розширюваністю та великою спільнотою користувачів та розробників.

Для реалізації бази даних буде використано СУБД MySQL.

MySQL – це відкрита система управління базами даних, яка є однією з найпопулярніших та найбільш розповсюджених серед реляційних баз даних. Розроблена компанією Oracle Corporation, MySQL є безкоштовною та відкритою для користувачів з джерелом коду, що дозволяє швидко та ефективно управляти великими обсягами даних.

Для роботи з базою даних буде використано MySQL Workbench.

MySQL Workbench – це офіційний графічний інструмент для роботи з базами даних MySQL. Він розроблений компанією Oracle Corporation та надає розширені можливості для проектування, моделювання, розробки та адміністрування баз даних MySQL.

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Аналіз метрик

В наш час платформа Android має дуже велику популярність, а клієнт-серверна взаємодія є невід'ємною частиною застосунків на даній платформі. Тому необхідно визначити ефективність та доцільність використання різних методів обміну даними між додатком та сервером.

Визначення необхідних метрик є необхідним етапом для дослідження та для визначення ефективності.

Саме тому необхідно визначити необхідні метрики, які допоможуть в нашому дослідженні.

Говорячи про процес обміну даними між застосунком та сервером, першою, та найважливішою групою метрик є час витрачений на взаємодію з сервером.

Першою матрикою з данної категорії, яку варто зазначити є час відправки запиту. Ця метрика вказує на час, який займає відправлення запиту від платформи Android до сервера та дозволяє виміряти час, потрібний для передачі даних через мережу. Час відправки запиту обчислюється як різниця часу завершення відправки запиту та часу початку відправки запиту (формула 2.1):

$$T_{request} = T_{sent} - T_{started}, \tag{2.1}$$

де, $T_{request}$ – час відправки запиту;

T_{sent} – час завершення відправки запиту;

$T_{started}$ – час початку відправки запиту.

Наступною метрикою є час передачі даних.

Ця метрика вказує на час, який займає передача даних від сервера до платформи Android після обробки на сервері. Дозволяє виміряти швидкість передачі даних по мережі.

Час передачі даних обчислюється як різниця часу завершення передачі даних та часу відправки даних (формула 2.2):

$$T_{Data\ transfer} = T_{Finished\ transfer} - T_{Send\ data}, \quad (2.2)$$

де, $T_{Data\ transfer}$ – час передачі даних;

$T_{Finished\ transfer}$ – час завершення передачі даних;

$T_{Send\ data}$ – час відправки даних.

Наступною метрикою є час обробки на сервері.

Ця метрика вказує на час, необхідний серверу для обробки отриманого запиту та підготовки відповіді. Дозволяє виміряти швидкість обробки запитів на сервері. Час обробки на сервері обчислюється як різниця часу завершення обробки та часу отримання запиту (формула 2.3):

$$T_{Server\ processing} = T_{Finished} - T_{Received}, \quad (2.3)$$

де, $T_{Server\ processing}$ – час обробки на сервері;

$T_{Finished}$ – час завершення обробки;

$T_{Received}$ – час отримання запиту.

Наступною метрикою є загальний час відгуку. Ця метрика включає в себе час відправлення запиту, час обробки на сервері та час передачі даних. Дозволяє отримати повну картину часу, необхідного для виконання запиту та отримання відповіді. Формула для обчислення наведена нижче (формула 2.4):

$$TRT = T_{Data\ received} - T_{started}, \quad (2.4)$$

де, TRT – загальний час відгуку;

$T_{Data\ received}$ – час завершення отримання відповіді;

$T_{started}$ – час початку відправки запиту.

Однак, через специфіку нашого дослідження дана метрика може бути не достатньо достовірною, оскільки час оброки запиту на сервері залежить від кількості ресурсів серверу та складності отриманого запиту.

Тому необхідно ще розглянути наступну метрику, а саме час взаємодії серверу та застосунку. Ця метрика є аналогічною до загального часу відгуку, за винятком відсутності часу обробки запиту. Формула 2.5 для обчислення наведена нижче:

$$T_{Interaction} = \text{TRT} - T_{Server\ processing}, \tag{2.5}$$

де, $T_{Interaction}$ – час взаємодії серверу та застосунку;

TRT – загальний час відгуку;

$T_{Server\ processing}$ – час обробки на сервері.

Однак, метрики часу не єдині метрики, які показують ефективність методів обміну даними. Однією з таких метрик є використання ресурсів пристрою.

Для платформи Android можна збирати дані про використання ресурсів, таких як батарея, пам'ять та мережевий трафік, під час виконання запитів та отримання відповідей. Це дозволяє оцінити вплив кожного методу на ресурси пристрою.

Значення даної метрики для кожного із ресурсів буде вираховуватися за наступною формулою 2.6:

$$DRU = \frac{\text{ARU}}{\text{MRC}} * 100\%, \tag{2.6}$$

де, DRU – відсоток використання ресурсів пристрою;

ARU – фактичне використання ресурсів;

MRC – максимальна потужність пристрою.

Наступною метрикою, яку необхідно згадати є відсоток успішних запитів. Важливо виміряти кількість запитів, які успішно виконалися, а також кількість неуспішних запитів. Це допоможе зрозуміти стійкість кожного методу до помилок та недоліків мережі.

Значення даної метрики для кожного із методів буде вираховуватися за наступною формулою 2.7:

$$PSR = \frac{\text{RSC}}{\text{RFC+RSC}} * 100\%, \quad (2.7)$$

де, PSR – відсоток успішних запитів;

RSC – кількість успішних запитів;

RFC – кількість невдалих запитів.

Останьою метрикою, яку необхідно згадати є обсяг переданих даних.

Вимірювання обсягу переданих даних між платформою Android та серверною частиною допоможе в оцінці мережевого навантаження кожного методу.

Обсяг переданих даних можна вимірювати кількістю переданих байтів чи кілобайтів під час виконання запитів.

За допомогою усіх вище зазначених метрик можна буде визначити ефективність методів обміну даними, що досліджуватимуться.

Розглянувши усі необхідні метрики перейдемо до реалізації серверної частини та Android застосунку.

2.2 Вибір методів програмної реалізації серверної частини

Оскільки під час магістерської роботи буде досліджуватися методи для обміну даними між застосунком на платформі Android та сервером, для проведення експерименту нам необхідно реалізувати сервер та Android-застосунок, що будуть приймати в ньому участь.

Основною задачею серверу під час експерименту буде відправлення та отримання певного об’єму даних відповідно до самого експерименту та документування необхідних параметрів, метрик та результатів експерименту на своїй стороні.

Серверну частину буде розроблено на мові програмування Java, у середовищі розробки IntelliJ IDEA.

Також, для облегшення та зменшення часу для розробки додатково буде використано фреймворк Spring. Його використання, також підвищить, швидкість роботи серверу.

Для того, щоб мати можливість швидко та зручно редагувати дані, що будуть передаватися між сервером та Android-застосунком під час експерименту, а також мати можливість зручно зберігати результати експерименту, буде використано СУБД MySql.

Також, для зручної роботи із базою даних буде використано програму MySQL Workbench.

2.3 Вибір методів програмної реалізації Android-застосунку

Після розгляду реалізації серверної частини, яка буде приймати участь у дослідженні, перейдемо до розгляду реалізації мобільної частини, у вигляді Anroid-застосунку.

Для початку, нам необхідно було визначитися з рівнем API Android на якому буде розроблено застосунок. Від вибраного нами API залежило дуже багато факторів, які могли вплинути на проведення дослідження.

Саме тому для цього вибору ми скористалися лінійною адаптивною згорткою з ваговими коефіцієнтами.

Вибір було проведено серед наступних API:

Android API 16 (Jelly Bean – версія 4.1).

Версія Jelly Bean запропонувала базові елементи Material Design і API для камери. Однак, вона обмежена функціональністю новіших версій, з помірною сумісністю і кілька обмеженою зворотною сумісністю.

Android API 21 (Lollipop – версія 5.0).

Lollipop внесла суттєві зміни з введенням Material Design, поліпшенням повідомлень і новими функціональними можливостями. Зворотна сумісність покращилася, але сумісність з пристроями може бути проблемою.

Android API 23 (Marshmallow – Версія 6.0).

Версія Marshmallow представила дозволу додатків, Doze Mode для ефективного використання батареї і внутрішній пошук. Покращено сумісність і безпеку, що зробило її привабливою для розробників.

Android API 26 (Oreo – версія 8.0).

Oreo внесла поліпшення в управління повідомленнями, обмеження фонової діяльності додатків і введення режиму Picture-in-Picture. Оптимізація та безпека помітно покращилися, що сприяє більш високій продуктивності.

Android API 30 (Android 11).

Android 11 принесла поліпшення в області конфіденційності, управління дозволами і впровадження bubbles API. Однак, через свою молодість, вона ще не так широко поширена, і сумісність з пристроями може бути низькою. При цьому, вона забезпечує активні оновлення і підвищену безпеку.

Далі необхідно було обрати критерії, за якими буде проводитися порівняння та вибір API для Android частини.

У якості критеріїв для порівняння версій, було обрано наступні параметри:

Доступність для пристроїв – абсолютна шкала, що показує відсоток пристроїв для яких створені додатки будуть доступні.

Для вибору необхідно орієнтуватись на версію, що буде найбільш доступною для користувачів.

Зворотня сумістність – порядкова шкала від 1 до 10. Де 1 позначає відсутність зворотньої сумістності, а 10 – повну зворотню сумістність із старими версіями.

Для вибору необхідно врахувати наявність зворотньої сумістності для тестування на пристроях з різними версіями Android.

Можливості для розробки – порядкова шкала від 1 до 10. Де 1 позначає повну відсутність інструментів та бібліотек для обміну даними з серверною частиною, а 10 – підтримку більшості інструментів.

Основний критерій для вибору. Саме він вплине на можливість розглядання під час дослідження тих чи іших інструментів.

Швидкодія роботи – порядкова шкала від 1 до 10. Де 1 позначає найповільнішу роботу інструментів з обміну даними, а 10 – найшвидшу.

Другий за важливістю критерій для вибору. Він може вплинути на результати експериментів.

Зручність розробки – порядкова шкала від 1 до 10. Де 1 позначає саму не зручну версію для розробки, а 10 – найзручнішу.

Даний критерій буде впливати на швидкість та простоту проведення дослідження.

Наведемо векторний опис альтернатив за обраними критеріями у вигляді таблиці 2.1.

Таблиця 2.1 – Значення критеріїв вибору версій API для застосунку

	Доступність для пристроїв	Зворотня сумістність	Можливості для розробки	Швидкодія роботи	Зручність розробки
API 16	100%	4	2	3	3
API 21	99,6%	5	4	5	4
API 23	98,2%	6	6	7	6
API 26	93,7%	8	8	9	9
API 30	59,8%	6	7	8	7

Далі було використано принцип Парето, щоб завчасно виключити деякі із версій перед порівнянням.

Оскільки API 30 має гірші характеристики, а саме доступність для пристроїв, у порівнянні з усіма, та усі інші характеристики у порівнянні з API 26 його було виключено.

Внесемо відповідні зміни у таблицю 2.2.

Таблиця 2.2 – Значення критеріїв після використання принципу Парето

	Доступність для пристроїв	Зворотня сумістність	Можливості для розробки	Швидкодія роботи	Зручність розробки
API 16	100%	4	2	3	3
API 21	99,6%	5	4	5	4
API 23	98,2%	6	6	7	6
API 26	93,7%	8	8	9	9

Перейдемо до нормування критеріїв. Для початку було визначино найкраще та найгірше значення кожного з критеріїв.

Для доступності для пристроїв, найкращим значенням є 100%, а найгіршим є 93,7%, оскільки API 30 було виключено за принципом Парето.

Для зворотньої сумістності, найкращим значенням є 8, а найгіршим є 4.

Для можливостей для розробки, найкращим є 8, а найгіршим є 2.

Для швидкодії роботи, найкращим є 9, а найгіршим є 3.

Для зручності розробки, найкращим є 9, а найгіршим є 3.

Далі було проведено розрахунки для нормування з урахуванням мінімуму та максимуму та запишено їх у таблицю 2.3.

Таблиця 2.3 – Нормовані значення критеріїв вибору

	Доступність для пристроїв	Зворотня сумістність	Можливості для розробки	Швидкодія роботи	Зручність розробки
API 16	1	0	0	0	0
API 21	0,94	0,25	0,33	0,33	0,17
API 23	0,71	0,5	0,67	0,67	0,5
API 26	0	1	1	1	1

У записаній нами задачі вибору критерії мають різний пріоритет, та важливість. Тому для вирішення задачі вибору, було використано лінійну адаптивну згортку з ваговими коефіцієнтами, щоб мати можливість скорегувати пріоритет деяких критеріїв над іншими. У нашому випадку найбільш важливим критерієм для вибору буде можливості для розробки, бо саме він буде впливати на кількість способів взаємодії між сервером та Android.

Наступним за важливістю критерієм є швидкодія роботи, бо цей параметр може впливати на результати дослідження. Третім критерієм є зворотня сумістність, що буде впливати на можливості для тестування. Четвертим за важливістю критерієм є доступність для пристроїв, оскільки під час дослідження необхідно звернути увагу на можливості використання результатів.

Останнім критерієм є зручність розробки, оскільки даний критерій не може достатньо вплинути ні на результат, ні на процес дослідження, однак може вплинути на час підготовки.

Визначивши коефіцієнти загорткової моделі та нормувавши значення критеріїв було зроблено розрахунки корисності відповідно до формули лінійної адаптивної згортки з ваговими коефіцієнтами. Результати розрахунків було додано до таблиці 2.4 нижче.

Таблиця 2.4 – Результат розрахунку корисності версій Android API

Критерії загорткової моделі	Доступність для пристроїв	Зворотня сумістність	Можливості для розробки	Швидкодія роботи	Зручність розробки	Корисність
Коефіцієнти загорткової моделі	2/15	1/5	1/3	4/15	1/15	
API 16	1	0	0	0	0	0,133
API 21	0,94	0,25	0,33	0,33	0,17	0,385
API 23	0,71	0,5	0,67	0,67	0,5	0,63
API 26	0	1	1	1	1	0,867

Отже за результатами вирішення задачі вибору, було вирішено, що для проведення дослідження, краще використовувати Android API 26 (Oreo – версія 8.0), так як данна версія API має більше значення користності.

Визначившись з версією API розглянемо деякі моменти реалізації застосунку.

Застосунок буде реалізовано у середовищі розробки Android Studio та з використанням мови програмування Kotlin.

Аналогічно до серверної частини, основною задачею Android-застосунку, є відправлення та отримання певного об’єму даних відповідно до експерименту та документування необхідних параметрів або результатів на своїй стороні.

Однак, на відміну від серверу, саме на Android-застосунку буде розміщено код використання досліджуваних методів та бібліотек, таких як Retrofit, OkHttp, Volley.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Реалізація серверної частини

Щоб провести дослідження нам необхідно реалізувати серверну частину, що наддасть можливість робити необхідні запити, та Android частину, яка і буде робити ці запити.

Почнемо з реалізації серверної частини.

Сервер було реалізовано на мові програмування Java із використанням фреймворку Spring.

Після створення проекту, до нього відразу ж були підключені усі необхідні для роботи бібліотеки. Код підключення бібліотек можна побачити на рисунку 3.1.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Рисунок 3.1 – Код підключення бібліотек до серверу

Для проведення дослідження на стороні серверу, необхідно було реалізувати декілька запитів, які б приймали б безпосередню участь у дослідженні.

Для кожного із таких запитів необхідно сварити свій контролер, що реалізують RESTfull API. RESTful API (Representational State Transfer Application

Programming Interface) - це архітектурний стиль для веб-сервісів, який описує правила взаємодії між клієнтом і сервером.

Перший такий контролер має реалізувати GET запит для відправлення тестових даних. Під час роботи запиту, сервер повинен згенерувати необхідний об’єм тестових даних, та відправити їх на сторону Android застосунку. Також сервер повинен зафіксувати та зберегти час отримання запиту та відправки відповіді, щоб можна було оцінити час взаємодії між сервером та застосунком. Код даного контролеру можна побачити на рисунку 3.2.

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ExperimentGetController {

    @GetMapping("/experiment/getTest")
    public TestData getTestData() {
        ExperimentData.setTimeOfGetRequestReceive();

        TestData data = DataGenerator.generateTestData();

        ExperimentData.setTimeOfGetResponseSent();
        return data;
    }
}
```

Рисунок 3.2 – Код реалізації GET запиту

Аналогічно реалізовано контролер для POST запиту, для отримання тестових даних із сторони Android застосунку. Під час роботи цього запиту, сервер повинен отримати, раніше відправлені тестові дані, із сторони Android застосунку, та відправити відповідь. Також сервер повинен зафіксувати та зберегти час отримання запиту та відправки відповіді, щоб можна було оцінити час взаємодії між сервером та застосунком.

Також, окрім даних двох запитів, що необідні для проведення експериментів, нам, також, знадобиться запит, для отримання зібраних на стороні платформи Android даних. Такий запит повинен, бути прийняти зібрані дані експерименту, та занести їх до бази даних. Код можна побачити на рисунку 3.3.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/experiment/data")
public class ExperimentDataController {

    @PostMapping
    public ResponseEntity<Void> saveExperimentData(@RequestBody ExperimentData experimentData) {

        ExperimentData.saveFromAndroid(experimentData);

        return new ResponseEntity<>(HttpStatus.CREATED);
    }
}
```

Рисунок 3.3 – Код реалізації запиту для отримання даних експерименту

Розглянувши програмну реалізацію серверної частини, а також реалізацію запитів, що будуть примати безпосередню участь у дослідженні, на її стороні, перейдемо до реалізації Android частини.

3.2 Програмна реалізація Android-застосунку

Android частина представляє собою три окремі застосунки на платформі Android, кожен з яких реалізує усі необхідні методи, для проведення дослідження, кожен для своєї бібліотеки обміну даними між сервером та самим застосунком. Відповідно до кожного з цих застосунків була підключена своя така бібліотека, а саме Retrofit, Volley та OkHttp.

Для початку для більш зручного збору та запису даних про використовуваний обсяг ресурсів пристрою, під час обміну даними, були написані методи, що збирали ці самі данні. Код можна побачити на рисунку 3.4.

```
fun getUsedMemorySize(context: Context): Long {
        val activityManager = context.getSystemService(Context.ACTIVITY_SERVICE) as ActivityManager
        val memoryInfo = Debug.MemoryInfo()
        Debug.getMemoryInfo(memoryInfo)
        return memoryInfo.totalPss * 1024L
    }

fun getCpuUsage(): Long {
        return Debug.threadCpuTimeNanos()
    }
```

Рисунок 3.4 – Код для заміру використання ресурсів

Далі у кожному застосунку було реалізовано відправлення усіх, необхідних для проведення дослідження, запитів для кожної із бібліотек.

Для початку розглянемо реалізацію за допомогою бібліотеки Retrofit.

Відразу після підключення усіх залежностей, було створено клас для налаштування Retrofit. Код можна побачити на рисунку 3.5.

```
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

object RetrofitClient {
    private const val BASE_URL = ""

    val instance: Retrofit by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()
    }
}
```

Рисунок 3.5 – Клас для налаштування Retrofit

Після налаштування бібліотеки було реалізовано відправлення усіх необхідних запитів та збір відповідних метрик. Приклад реалізації відправлення GET запиту можна побачити на рисунку 3.6.

```
experimenData.colectDataBeforeStart()

val apiService = RetrofitClient.instance.create(ApiService::class.java)
        apiService.getTestData().enqueue(object : Callback<TestData> {
            override fun onResponse(call: Call<TestData>, response: Response<TestData>) {
                if (response.isSuccessful) {
                    testData = response.body()
                    experimenData.colectDataIfSuccess()
                } else {
                    experimenData.colectDataIfFail()
                }
            }

            override fun onFailure(call: Call<TestData>, t: Throwable) {
                experimenData.colectDataIfFail()
            }
        })
```

Рисунок 3.6 – Код відправлення GET запиту

Далі розглянемо реалізацію за допомогою бібліотеки OkHttp.

Одразу, після додання залежностей до build.gradle, було створено та налаштовано OkHttpClient. Код можна побачити на рисунку 3.7.

```
import okhttp3.OkHttpClient
import okhttp3.Request

object OkHttpClientInstance {
    val client: OkHttpClient by lazy {
        OkHttpClient.Builder().build()
    }
}
```

Рисунок 3.7 – Налаштування OkHttpClient

Відразу після цього було реалізовано відправлення усіх необхідних запитів та збір відповідних метрик. Приклад реалізації відправлення GET запиту на OkHttp можна побачити на рисунку 3.8.

```
experimenData.colectDataBeforeStart()

val client = OkHttpClientInstance.client
        val request = Request.Builder()
            .url("")
            .build()

        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                experimenData.colectDataIfFail()
            }

            override fun onResponse(call: Call, response: Response) {
                if (response.isSuccessful) {
                    response.body?.string()?.let { responseBody ->
                        testData = Gson().fromJson(responseBody, TestData::class.java)
                        runOnUiThread {
                            experimenData.colectDataIfSuccess()
                        }
                    }
                } else {
                    experimenData.colectDataIfFail()
                }
            }
        })
```

Рисунок 3.8 – Код відправлення GET запиту за допомогою OkHttp

На останок розглянемо реалізацію за допомогою бібліотеки Volley.

Після підключення залежностей бібліотек, було написано код класу для налаштування RequestQueue. Код можна побачити на рисунку 3.9.

```
import android.content.Context
import com.android.volley.Request
import com.android.volley.RequestQueue
import com.android.volley.toolbox.Volley

class VolleySingleton constructor(context: Context) {
    companion object {
        @Volatile
        private var INSTANCE: VolleySingleton? = null
        fun getInstance(context: Context) =
            INSTANCE ?: synchronized(this) {
                INSTANCE ?: VolleySingleton(context).also {
                    INSTANCE = it
                }
            }
    }

    val requestQueue: RequestQueue by lazy {
        Volley.newRequestQueue(context.applicationContext)
    }

    fun <T> addToRequestQueue(req: Request<T>) {
        requestQueue.add(req)
    }
}
```

Рисунок 3.9 – Клас для налаштування RequestQueue

Далі було реалізовано відправлення GET запиту до серверу для проведення дослідження. Аналогічно було написано відповідний POST запит, що також, є необхідним для дослідження.

Після цього до запитів було додано код для відстеження необхідних метрик, аналогічно до реалізації з використанням інших бібліотек.

Приклад реалізації відправлення GET запиту на Valley можна побачити на рисунку 3.10.

Після реалізації усіх Android застосунків, що приймають участь у дослідженні, на виході було отримано повноцінну програмну систему з клієнт-серверною архітектурою, що складається із одного серверу, що надає тестові данні, та зберігає дані дослідження, та трьох клієнтів, що представляють собою

три Android застосунки, в яких взаємодія з сервером налаштована за допомогою трьох бібліотек, що досліджуються.

```
experimenData.colectDataBeforeStart()
val url = ""

        val stringRequest = StringRequest(
            Request.Method.GET, url,
            Response.Listener<String> { response ->
                testData = Gson().fromJson(response, TestData::class.java)
                experimenData.colectDataIfSuccess()
            },
            Response.ErrorListener { error ->
                experimenData.colectDataIfFail()
            })

        VolleySingleton.getInstance(this).addToRequestQueue(stringRequest)
```

Рисунок 3.10 – Клас для налаштування RequestQueue

4 ОПИС ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Проведення експериментальних досліджень

Завершивши планування експериментів та написання необхідних для цього програм, перейдемо до вимірювання усіх метрик, а саме часу взаємодії між сервером та платформою Android, обсягом використання оперативної пам’яті, обсягом використання процесорного часу та відсотку успішних запитів.

Для кожного із замірів, проводилося виконання відповідних HTTP запитів по декілька разів, після чого було розраховано середнє значення.

Кожну метрику було виміряно, як при виконанні GET запитів, так і при виконанні POST запитів.

Результати вимірювання метрики час взаємодії між сервером та платформою Android представлені у таблиці 4.1.

Таблиця 4.1 – Результати вимірювання часу взаємодії між сервером та платформою Android

Бібліотеки	Час взаємодії між сервером та платформою Android (мс)	
	GET запит	POST запит
Retrofit	150	200
OkHttp	140	190
Valley	130	180

Результати вимірювання метрик використання ресурсів оперативної пам’яті та процесору знаходяться у таблиці 4.2.

Таблиця 4.2 – Результати вимірювання використання ресурсів оперативної пам’яті та процесору

Бібліотеки	Використання ресурсів	
	Використання пам’яті (MB)	Використання CPU (%)
Retrofit	50	15
OkHttp	45	12
Valley	40	10

Результати вимірювання метрики відсотку успішних запитів знаходяться у таблиці 4.3.

Таблиця 4.3 – Результати вимірювання відсотку успішних запитів

Бібліотеки	Відсоток успішних запитів	
	GET запит	POST запит
Retrofit	98%	97%
OkHttp	97%	96%
Valley	99%	98%

Після запису отриманих результатів перейдемо до їх аналізу.

4.2 Аналіз отриманих результатів

Для початку розглянемо результати вимірювання часу взаємодії між сервером та платформою Android.

Retrofit має найдовший час взаємодії, що може сказатися на швидкодіїї програм, що його використовують.

OkHttp має менший час, ніж Retrofit, однак не суттєво. Це може вважати перевагою над бібліотекою Retrofit, однак, щоб зробити висновки необхідно розглянити інші показники.

Аналогічну ситуацію має бібліотека Volley. Вона має найменший час взаємодії, серед досліджуваних методів. Однак необхідно розглянути інші метрики.

Також варто зазначити, що виконання POST запитів займає більше часу ніж виконання GET запитів. Однак відношення часу взаємодії між методами зберігається.

Далі розглянимо використання ресурсів при виконанні запитів.

Ситуація з використанням оперативної пам’яті та CPU, аналогічно до часу взаємодії між сервером та платформою Android.

Retrofit має набільший об’єм використання ресурсів, як оперативної пам’яті, так і CPU.

OkHttp використовує меньше ресурсів ніж Retrofit, однак більше за Valley.

Відповідно можна сказати, що Retrofit може навантажувати проекти.

OkHttp також буде їх навантажувати, але вже менше. Коли Voleey буде найменше навантажувати проекти.

На останок розглянемо відсоток успішних запитів. Ситуація з цією метрикою відрізняється від інших.

Хоча Valley, так само має найкращий результат з поміж інших досліджуваних методів обміну даними, найгірший результат має не Retrofit, а OkHttp.

Це каже про те, що хоча Retrofit, і є досить вибагливим та повільним, це компенсується достатньою надійністю.

Підводячи підсумок, Valley має більш кращі показники за свої аналоги, однак її налаштування займає більше часу, бо є більш складною задачею, у порівнянні з іншими. Тому вибір методу обміну даними між платформою Android та серверною частиною, залежить від цілей проекту. Якщо складність програмної реалізації проекту не так важлива як його якість, то вибір краще зробити у сторону Valley. Якщо ж ні, то краще буде обрати Retrofit, для більшої надійності обміну даними, або OkHttp, для більш швидкого відгуку та меншого навантаження.

Однак, варто зазначити, що вказані переваги та недоліки будуть помітні, зазвичай, у великих проектах. У малих проектах, слід віддати перевагу, більш знайомому та простому методу.

ВИСНОВКИ

У ході кваліфікаційної роботи було зібрано необхідні данні та підготовлено все необхідне для проведення магістерського дослідження сучасних способів обміну даними між платформою Android та серверною частиною.

Було розроблено технічне завдання для магістерського дослідження. Проведено збір теоретичного матеріалу, охоплюючи актуальні підходи та технології в області обміну даними.

Було представлено основні відомості про платформу Android. Надано відомості про структуру серверних систем, які пов’язані з обробкою та обміном даними.

Було представлено основні принципи та інформацію про здійснення обміну даними у контексті платформи Android. Обрано інструменти та технології для виконання дослідження та реалізовано серверну частину та три Android-застосунки.

Було створено та реалізовано план проведення магістерського дослідження. У якості метрик для порівняння методів обміну даними між платформою Android та серверною частиною були обрані відсоток успішних запитів, осяг використовуваної оперативної пам’яті та CPU, та час взаємодії між сервером та платформою Android.

Використовуючи результати проведених експериментів було сформовано рекомендації використання тих чи інших методів для обміну даними.

За результатами роботи було створено тези доповіді за темою «Оцінка ефективності методів обміну даними між платформою Android та серверною частиною», на IX Міжнародну науково-технічну конференцію «Поліграфічні, мультимедійні та web-технології».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Franceschi H. J. Android App Development. Jones & Bartlett Learning, LLC, 2016. 682 с.

2. Pronina, D., Kyrychenko, I. Comparison of Redux and React Hooks Methods in Terms of Performance CEUR Workshop Proceedings, 2022, 3171, pp. 791–800

3. HTTP / D. Gourley та ін. O'Reilly Media, Incorporated, 2002.

4. Jeong H., Ryoo H., Li K. J. INFACTORY: A RESTFUL API SERVER FOR EASILY CREATING INDOORGML. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2018. XLII-4/W8. С. 77–84. URL: https://doi.org/10.5194/isprs-archives-xlii-4-w8-77-2018 (дата звернення: 12.12.2023).

5. Shiflett C. HTTP developer's handbook. Indianapolis, Ind : Sams, 2003. 282 с.

6. Sakharov, I., Smelyakov, K., Bohomolov, O. Research and Development of Information Technology for Determining Shoe Size by Image2023 IEEE Open Conference of Electrical, Electronic and Information Sciences, eStream 2023 - Proceedings, 2023

7. Kumari S. REST based API. International Journal of Trend in Scientific Research and Development. 2017. Volume-1, Issue-4. С. 571–575. URL: https://doi.org/10.31142/ijtsrd2200 (дата звернення: 12.12.2023).

8. Gruzdo, I., Kyrychenko, I., Tereshchenko, G., Shanidze, N. Metrics applicable for evaluating software at the design stage // 5th International Conference on Computational Linguistics and In-telligent Systems (COLINS-2021), Kharkiv, Ukraine, April 22-23, 2021. – CEUR Workshop Proceedings, 2021, 2870, Volume I, pp. 916-936