

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка
до дипломного проєкту (роботи)
магістра
(ступінь вищої освіти (освітній ступінь))

на тему КОМП'ЮТЕРНА СИСТЕМА ДИСТАНЦІЙНОГО ЗБИРАННЯ
ANDROID-ДОДАТКІВ

Виконав: студент 2 курсу, групи КНТ-612м
спеціальності _____
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Спеціалізовані комп'ютерні системи

РУДЕНКО В.В.
(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.
(ПРИЗВИЩЕ та ініціали)

Рецензент СТЕПАНЕНКО О.О.
(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій

Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти магістерський

Спеціальність 123 Комп'ютерна інженерія

(код і найменування)

Освітня програма (спеціалізація) «Спеціалізовані комп'ютерні системи»

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Зав. кафедри Кудерметов Р.К.

“12” листопада 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

РУДЕНКО Віталій Володимирович

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Комп'ютерна система дистанційного збирання android-додатків

керівник проєкту (роботи) к. т. н., доцент, СКРУПСЬКИЙ Степан Юрійович

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “24” жовтня 2023 року № 400

2. Строк подання студентом проєкту (роботи) 10 грудня 2023 року

3. Вихідні дані до проєкту (роботи) існуючі методи та алгоритми збирання

Android-додатків, наявні варіанти локальної та дистанційної збірки

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Аналіз технічного завдання;

2) Розробка системи;

3) Дослідження системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проекту (роботи)

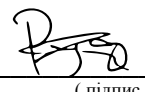
Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	СКРУПСЬКИЙ С.Ю., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.10.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз наявних методів та алгоритмів збирання Android-додатків	10.10.2023 р.	
2	Розробка структури системи	15.10.2023 р.	
3	Розробка алгоритму роботи системи	20.10.2023 р.	
4	Реалізація системи дистанційного збирання android-додатків	05.11.2023 р.	
5	Дослідження системи дистанційного збирання android-додатків	20.11.2023 р.	
6	Оформлення отриманих результатів у ПЗ	25.11.2023 р.	
7	Оформлення графічного матеріалу	01.12.2023 р.	
8	Оформлення допоміжного матеріалу	10.12.2023 р.	

Студент(ка)


(підпис)

Керівник проекту (роботи)


(підпис)

Віталій РУДЕНКО
(Ім'я ПРИЗВИЩЕ)

Степан СКРУПСЬКИЙ
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 81 с., 8 рис., 2 табл., 16 джерел.

KOTLIN, ANDROID STUDIO, JAVA, UBUNTU, ANDROID-ДОДАТКИ.

Об'єкт дослідження – процес дистанційного збирання Android-додатків.

Предмет дослідження – це комп'ютерна системи дистанційного збирання Android- додатків.

Метою магістерської роботи — створення системи для дистанційної збірки Android-додатків, що забезпечує ефективну, гнучку та зручну роботу розробників.

В цій роботі було розроблено інтегровану систему, здатну збирати Android-додатки віддалено, що дозволить істотно поліпшити робочий процес і знизити завантаження на локальні машини розробників.

Дипломна робота містить детальний основ систем збирання Android-додатків, розглянуті варіанти локальної та дистанційної збірки для реалізації системи обрано мову програмування Kotlin та використано інструментарій Android Studio.

Наукова новизна полягає у запропонованому підході, заснованому на використанні власної дистанційної системи збирання Android-додатків, що має можливість збирання додатків різної архітектури з різною кількістю модулів, комплексне рішення, яке відповідає різноманітним вимогам розробки програм для Android.

ABSTRACT

PP: 81 p. , 8 fig., 2 table., 16 sources.

KOTLIN, ANDROID STUDIO, JAVA, UBUNTU, ANDROID APPS.

The object of research is the process of remote Assembly of Android applications.

The subject of the research is a computer system for remote Assembly of Android applications.

The purpose of the master's thesis is to create a system for remote Assembly of Android applications, which ensures efficient, flexible and convenient work of developers.

In this work, an integrated system was developed that can build Android applications remotely, which will significantly improve the workflow and reduce loading on local developer machines.

The thesis contains a detailed description of the basics of Android application assembly systems, local and remote assembly options are considered the Kotlin programming language was chosen to implement the system and the Android Studio toolkit was used.

The scientific novelty lies in the proposed approach based on the use of its own remote system for collecting Android applications, which has the ability to collect applications of different architectures with a different number of modules, a complete solution that meets various requirements for developing applications for Android.

ЗМІСТ

1 Аналіз технічного завдання	6
1.1 Аналіз систем збирання Android-додатків	8
1.1.1 Локальна збірка	8
1.1.2 Дистанційна збірка	11
1.1.3 Дистанційне збирання та Continuous Integration	11
1.1.4 Переваги та недоліки дистанційної збірки	12
1.2 Огляд аналогів систем дистанційного збирання додатків	15
1.3 Аналіз систем дистанційного збирання додатків	18
1.4 Висновок по розділу	20
2 Розробка системи	21
2.1 Вибір програмного забезпечення	21
2.1.1 Вибір мови програмування	22
2.1.2 Вибір операційної системи	32
2.1.3 Вибір середовища розробки	36
2.2 Вибір апаратного забезпечення	39
2.3 Розробка дистанційної системи	42
2.4 Висновок по розділу	49
3 Дослідження системи	52
3.1 Основні характеристики системи	54
3.2 Порівняння використання системи для модульної та монолітної архітектури	55
3.2.1 Монолітна архітектура	55
3.2.2 Модульна архітектура	58
3.3 Дослідження часу збирання додатку	61
3.4 Висновки по розділу	67
Висновки	69
Перелік джерел посилання	70
Додаток А	72

ВСТУП

Метою магістерської роботи — створення системи для дистанційної збірки Android-додатків, що забезпечує ефективну, гнучку та зручну роботу розробників. Для досягнення цієї мети необхідно виконати наступні задачі:

- детальний аналіз існуючих систем збирання Android-додатків для визначення їх переваг і недоліків;
- розробка власної дистанційної системи збирання Android-додатків, використовуючи мову програмування Kotlin та інструментарій Android Studio;
- розгляд варіантів архітектур Android-додатків та вибір оптимального підходу до розробки;
- аналіз програмного та апаратного забезпечення для ефективної роботи машини для збирання додатків;
- порівняння розробленої системи з існуючими аналогами для визначення її переваг та можливостей.

Основна частина дослідження складається з введення, 3 розділів, висновків і списку джерел.

У першому розділі виконується огляд сучасних систем збирання додатків, які можуть бути використані в Android-розробці. Розглядаються переваги та недоліки кожного з методів.

У другому розділі вибір відповідного програмного забезпечення при розробці та збиранню додатків Android. Розглядається різні аспекти цього вибору, включаючи мови програмування та інші ключові компоненти.

У третьому розділі проектування налаштованої системи, призначеної для бездоганної інтеграції в сучасні робочі процеси розробки Android.

Практична цінність отриманих результатів – розроблено систему, використання якої підвищило ефективність розробки та роботи Android-додатків в порівнянні з аналогами.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

В епоху, яка визначається постійно прискореними темпами технологічного прогресу та дедалі більшою розповсюдженістю мобільних пристроїв, розробка додатків для Android стала однією з основних сфер розробки програмного забезпечення. Оскільки вимоги до інноваційних та ефективних додатків продовжують зростати, також зростає потреба в нових парадигмах розробки, які можуть оптимізувати процес розробки додатків і оптимізувати використання ресурсів. У цьому контексті концепція створення додатків Android на віддаленому комп'ютері набула значного інтересу, пропонуючи багатообіцяючий шлях для вирішення проблем і можливостей, які представляє ця динамічна екосистема. Ця дипломна робота досліджує критичну важливість і невід'ємну оптимальність збірки додатків Android на віддалених машинах, з'ясовуючи переваги, проблеми та наслідки цієї зміни парадигми у сфері розробки мобільних додатків.

Значення розробки додатків для Android:

Операційна система Android, розроблена Google, стала домінуючою силою в мобільній екосистемі. З мільйонами пристроїв Android у руках користувачів по всьому світу, додатки Android є важливими інструментами для виконання різноманітних завдань, від спілкування та продуктивності до розваг та навчання. Як наслідок, роль розробників додатків для Android ніколи не була настільки важливою. Перед цими розробниками поставлено завдання створити програмне забезпечення, яке не лише відповідає вимогам користувачів щодо функціональності та зручності використання, але й не відстає від екосистеми Android, що швидко розвивається [1].

Однак розробка додатків для Android — це багатогранна робота, повна складностей і викликів. Розробники повинні враховувати різноманітні типи пристроїв, розміри екранів і роздільну здатність, а також варіації версій Android, не кажучи вже про необхідність безпеки, масштабованості та швидкості реагування. Крім того, створення додатків для Android потребує значної

обчислювальної потужності та ресурсів, що може бути вимогливим до окремих машин розробника.

Останніми роками в розробці додатків Android відбулася зміна парадигми, зросла потреба у віддаленій розробці, відхід від традиційних середовищ розробки, розміщених на локальних машинах, до використання платформ віддаленої розробки. Цей перехід був зумовлений збігом факторів:

- оптимізація ресурсів. Процес розробки додатків для Android може бути ресурсоемним, вимагаючи значних ресурсів процесора та пам'яті, особливо під час компіляції великих кодових баз, запуску симуляцій або проведення масштабного тестування. Переносячи ці ресурсомісткі завдання на віддалені машини, розробники можуть оптимізувати свої локальні машини для кодування та проектування, що призводить до більш ефективного процесу розробки;

- співпраця та командна робота: співпраця є основою сучасної розробки програмного забезпечення, а платформи віддаленої розробки сприяють безперебійній командній роботі. Працюючи на спільній віддаленій машині, учасники команди можуть співпрацювати в реальному часі, спільно використовувати ресурси та колективно налагоджувати та оптимізувати код. Це не тільки підвищує продуктивність, але й заохочує обмін знаннями та інновації;

- хмарні послуги та масштабованість: хмарні середовища розробки стають дедалі доступнішими та доступнішими. Хмарні платформи пропонують можливість масштабувати ресурси за вимогою, дозволяючи розробникам працювати над більшими проектами без необхідності інвестувати у дороге оновлення апаратного забезпечення;

- гнучкість платформи: віддалена розробка не обмежена специфікаціями локальної операційної системи розробника чи обладнання. Розробники можуть отримати доступ до потужних середовищ розробки з широкого діапазону пристроїв, у тому числі недорогих машин, планшетів і навіть

смартфонів. Ця гнучкість розширює охоплення розробки додатків для Android на ширшу аудиторію розробників [2].

Хоча перехід до віддаленої розробки додатків для Android дає значні переваги, він не позбавлений проблем і наслідків. Питання безпеки, затримка мережі та крива навчання, пов'язана з інструментами віддаленої розробки, – лише деякі з проблем, з якими можуть зіткнутися розробники. Крім того, розробники повинні орієнтуватися у складності вибору правильної платформи віддаленої розробки або хмарної служби для задоволення своїх конкретних потреб.

У цій дипломній роботі буде детально розглянуто ці проблеми та наслідки, щоб забезпечити всебічне розуміння важливості та оптимальності створення додатків Android на віддалених машинах. Досліджуючи ці аспекти, зрозуміємо, як віддалена розробка може покращити процес розробки додатків Android, зрештою сприяючи створенню більш ефективних, масштабованих і функціональних мобільних додатків для екосистеми Android, що постійно розвивається.

1.1 Аналіз систем збирання Android-додатків

Оскільки розробка Android розвивалася, розробники тепер мають вибір збирати та створювати свої програми локально чи віддалено. В дипломній роботі ми обидва варіанти і оберемо оптимальний.

1.1.1 Локальна збірка

Локальна збірка програм для Android починається з налаштування середовища розробки на локальній машині розробника. Це включає такі ключові компоненти:

- Java Development Kit (JDK) – розробка Android в основному покладається на Java, тому потрібно встановити сумісну версію JDK. Офіційне середовище розробки Android, Android Studio, зазвичай містить JDK;

- Android Studio – це офіційне інтегроване середовище розробки (IDE) для розробки програм Android. Він надає інструменти для проектування, кодування та створення програм Android [2];

- SDK і інструменти платформи – Android Studio включає Android SDK і інструменти платформи, необхідні для створення програм Android. Менеджер SDK в Android Studio дозволяє розробникам завантажувати компоненти платформи Android і керувати ними.

Після налаштування середовища розробки, розробник може створити новий проект Android. Під час створення проекту вони можуть налаштувати різні аспекти програми, такі як її назва, ім'я пакета та мінімальний рівень Android API, який він підтримуватиме. Крім того, розробники можуть вибирати між різними шаблонами проектів, які визначають початкову структуру програми.

Розробники пишуть код програми, використовуючи Java або Kotlin для серверної логіки. XML-файли використовуються для визначення макета інтерфейсу користувача, а інші ресурси, такі як зображення, шрифти та рядки, зберігаються у спеціальних папках у межах проекту [3].

Після написання коду та ресурсів розробник може розпочати процес компіляції та збірки [4]. Це включає кілька кроків:

- компіляція. Вихідний код компілюється в байт-код, який сумісний із віртуальною машиною Java (JVM). Зібраний код зберігається в каталозі збірки;
- компіляція. Файли макетів XML та інші ресурси компілюються у двійковий формат XML і зберігаються в каталозі res;
- упаковка. Скомпільований код і ресурси упаковані у файл APK (пакет Android). Цей файл містить усі необхідні компоненти програми, що робить її готовою до встановлення на пристрої Android;

- підпис. Файл APK має мати цифровий підпис, щоб гарантувати його автентичність і цілісність. Розробники можуть використовувати ключ налагодження під час розробки або створити ключ випуску для публікації;
- вирівнювання. Щоб оптимізувати APK для розповсюдження, розробники можуть використовувати інструмент zipalign, щоб переконатися, що внутрішня файлова структура APK добре організована;
- тестування. Розробники можуть запускати програму на емуляторах або фізичних пристроях Android, щоб перевірити її функціональність і переконатися, що вона відповідає бажаним специфікаціям;
- розгортання. Після ретельного тестування програму її можна поширювати серед користувачів через Google Play Store або інші канали розповсюдження.

Переваги:

- повний контроль над середовищем. Розробник має повний контроль над середовищем збирання на своїй локальній машині. Це може бути корисним для налаштування та оптимізації процесу складання;
- відсутність затримок під час передачі файлів. При локальному збиранні немає затримок під час передачі файлів між локальною машиною та віддаленим сервером, що сприяє більш швидкому збиранню маленьких проектів;
- повний доступ до ресурсів локальної машини. Локальна збірка дозволяє використовувати ресурси локальної машини, включаючи процесор та пам'ять без додаткових обмежень;
- мобільність. Можливість працювати з будь-якої локації що забезпечує гнучкість та доступність у роботі.

Недоліки:

- обмежені ресурси. Локальна машина може мати обмежені обчислювальні ресурси, що може сповільнити збирання великих проектів;
- навантаження на локальну машину. Складання програм може навантажувати локальну машину, що призводить до зменшення продуктивності під час складання;
- залежності та конфлікти. Управління залежностями та їх конфліктами може бути складним завданням на локальній машині, особливо під час роботи з великою кількістю бібліотек та компонентів.

1.1.2 Дистанційна збірка

Дистанційне збирання Android-додатків забезпечує альтернативний підхід до процесу розробки. Цей метод особливо корисний у середовищах спільної роботи та під час вирішення ресурсномістких завдань.

Щоб дистанційно створювати програми Android, розробники можуть використовувати хмарні середовища розробки або виділені сервери збірки. Основні етапи налаштування середовищ віддаленої розробки включають:

- вибір платформи. Доступно кілька хмарних платформ розробки, як-от Google Cloud, Amazon Web Services (AWS) і Microsoft Azure. Розробники можуть вибрати платформу, яка найкраще відповідає їхнім вимогам;
- налаштування інструментів розробки. На віддаленій машині або сервері розробники повинні встановити необхідні інструменти розробки, включаючи JDK і Android SDK;
- доступ і автентифікація. Розробники повинні забезпечити безпечний доступ до віддаленого середовища через SSH або інші безпечні протоколи. Механізми автентифікації повинні бути на місці для захисту коду та створення артефактів [5].

Віддалена збірка Android-додатків є процесом складання додатків на сервері або віддаленій машині, а не на локальній машині розробника. Це процес, при якому розробник відправляє вихідний код та ресурси на віддалений сервер, де складання програми виконується на віддаленій машині. Результати збирання можуть бути передані назад на локальну машину розробника [6].

1.1.3 Дистанційне збирання та Continuous Integration

Інтеграція контролю версій: розробники використовують системи контролю версій, такі як Git, для керування проектом і співпраці над ним. Інтеграція з хмарними репозиторіями (наприклад, GitHub, Bitbucket) може полегшити безперервну інтеграцію (CI).

Автоматизовані сценарії збірки: сценарії збірки, часто написані в таких інструментах, як Gradle або Maven, налаштовані для автоматизації компіляції, упаковки та підписання програми Android. Ці сценарії можна виконувати на віддаленому сервері.

Безперервна інтеграція: такі інструменти CI, як Jenkins, Travis CI або CircleCI, можна налаштувати на автоматичний ініціатор збірок на основі змін коду, забезпечуючи створення та тестування програми щоразу, коли надходять оновлення.

Керування артефактами: артефакти, створені під час процесу збірки, наприклад файл APK та інші ресурси, зберігаються та керуються на віддаленому сервері.

Розповсюдження та розгортання: після успішних збірок файли APK можна автоматично розповсюджувати на різні канали розгортання, включаючи Google Play Store або внутрішні системи розповсюдження.

Хоча як локальне, так і дистанційне збирання мають свої переваги, є певні міркування та компроміси, про які слід знати:

- використання ресурсів. Локальна розробка може потребувати ресурсів на машині розробника, тоді як віддалена розробка може перекласти ці завдання на більш потужні сервери. Віддалена розробка може бути більш ефективною для великих проєктів;
- співпраця. Віддалена розробка добре підходить для спільних проєктів, оскільки забезпечує централізоване середовище, доступне для членів команди з усього світу. Місцевий розвиток може вимагати складних установок для співпраці;
- затримка. Віддалена розробка може призвести до затримки мережі, що вплине на процес створення. Локальний розвиток пропонує взаємодію в реальному часі з середовищем розробки;

– вартість. Хмарні платформи розробки можуть мати витрати залежно від використання, тоді як локальні інструменти розробки зазвичай безкоштовні. Розробники повинні враховувати фінансові наслідки обраного ними підходу.

1.1.4 Переваги та недоліки дистанційної збірки

Збирання та створення додатків для Android є ключовими етапами процесу розробки додатків, і розробники мають вибір виконувати ці завдання локально або віддалено. Кожен підхід має свої переваги та компроміси залежно від масштабу проекту, вимог до співпраці та наявних ресурсів. Локальна розробка забезпечує контроль і ефективність у режимі реального часу, тоді як віддалена розробка пропонує переваги оптимізації ресурсів і співпраці. Зрештою, вибір між цими двома методами залежить від конкретних потреб і цілей проекту розробки Android. Все ж, для розробки великих додатків командою розробників краще використовувати дистанційну, яка надає ряд істотних переваг:

– ресурсна оптимізація. Один із головних бенефітів віддаленого складання полягає в оптимізації використання ресурсів. Багато Android-додатків потребують значних обчислювальних потужностей для складання, що може навантажувати локальну машину розробника. Віддалена збірка дозволяє виконувати цей процес на потужних серверах, звільняючи ресурси локального комп'ютера інших завдань;

– швидкість та ефективність. Віддалена збірка Android-додатків може бути виконана на високопродуктивних серверах, що забезпечує більш швидке та ефективне виконання збірки. Це особливо важливо під час роботи з великими проектами, прискорюючи цикл розробки;

– масштабованість. Віддалена збірка дозволяє легко масштабувати ресурси за потреби. Це особливо корисно для команд, які працюють над великими проектами або мають сезонне збільшення робочого навантаження;

- спільна робота. Багато віддалених середовищ для збирання надають інструменти для спільної роботи. Розробники можуть легко спільно працювати над проектами, а також виконувати загальні зборки та тестування;
- управління залежностями. Видалені системи збирання дозволяють ефективно керувати залежностями та сторонніми бібліотеками. Це полегшує оновлення та узгодження версій бібліотек, зменшуючи потенційні конфлікти;
- дистанційний доступ. Розробники можуть запускати складання та тестування програм на віддалених пристроях та емуляторах без необхідності фізичного доступу до них. Це спрощує тестування на різних пристроях та конфігураціях;
- покращена безпека. Видалені системи збирання можуть надавати високий рівень безпеки для коду та збирання. Доступ до серверів може бути суворо обмежений і моніториться, що сприяє безпеці програми та коду;
- економія часу та зусиль. Віддалене складання дозволяє заощадити час та зусилля розробників, звільняючи їх від рутинних завдань, таких як встановлення залежностей та складання. Вони можуть зосереджуватися на кодуванні та покращенні функціональності програми.

Звісно, дистанційне збирання має деякі незначні недоліки:

- затримки передачі файлів. При передачі файлів між локальною та віддаленою машиною можуть виникнути затримки, особливо при великих обсягах даних;
- необхідність налаштування та інтеграції. Налаштування та інтеграція віддаленого середовища може вимагати додаткових зусиль та часу;

– залежність від доступності віддаленого сервера. Відсутність віддаленого сервера може призвести до затримок у процесі розробки.

Вибір між локальною та віддаленою збіркою Android-додатків залежить від конкретних потреб проекту та переваг розробників. Віддалена збірка надає безліч переваг, особливо для великих проектів та команд розробників, але потребує налаштування та інтеграції. Локальна збірка забезпечує повний контроль, але може бути обмежена ресурсами локальної машини. Правильний вибір залежить від конкретних потреб та цілей розробників [7].

В цілому, збирання Android-додатків на віддаленій машині надає ефективний спосіб прискорення та оптимізації розробки, що особливо важливо у сучасному світі мобільних додатків [5].

1.2 Огляд аналогів систем дистанційного збирання додатків

Розробка додатків для Android – це багатогранний процес, який часто вимагає значних обчислювальних ресурсів. Вибір системи збирання додатків Android локально чи віддалено може сильно вплинути на ефективність і співпрацю в групах розробників.

Розглянемо один з аналогів систем дистанційного збирання додатків - Mainframer [8]. Хоча Mainframer пропонує унікальні переваги, він також має кілька недоліків, включаючи проблеми, пов'язані з його використанням у програмах Android, роботу на основі консолі та труднощі налаштування.

Віддалене збирання додатків для Android набуло популярності завдяки своєму потенціалу для вдосконалення процесів розробки, оптимізації співпраці та оптимізації використання ресурсів. Віддалені середовища розробки дозволяють розробникам перекладати ресурсомісткі завдання, такі як компіляція, упаковка та тестування, на віддалені сервери або хмарні платформи. Цей підхід стає все більш популярним, особливо у спільних і великомасштабних проектах Android.

Mainframer — це інструмент із відкритим вихідним кодом, призначений для полегшення віддаленого збирання та створення програм Android. Він пропонує засоби для ефективного використання обчислювальної потужності віддалених серверів для виконання завдань збірки Android, зменшуючи ресурсне навантаження на локальні машини розробників. Mainframer працює шляхом перенесення завдань Gradle на віддалений сервер, що забезпечує швидші та ефективніші процеси збирання. Хоча Mainframer був прийнятий через його потенціал для прискорення розробки додатків для Android, він має певні обмеження та проблеми.

Розглянемо переваги та недоліки використання Mainframer.

Переваги Mainframer:

- віддалене збирання. Mainframer дозволяє виконувати збирання Android-додатків на віддалених серверах, забезпечуючи оптимальне використання обчислювальних ресурсів. Це знижує навантаження на локальну машину розробника та прискорює процес збирання.

- ефективна передача файлів. Mainframer вміє ефективно передавати файли між локальною та віддаленою машиною. Він копіює файли, виконує команди і результат передає на локальну машину. А при повторному виклику, копіюються лише змінені файли, що істотно зменшує обсяг даних, що передаються. Це значно покращує продуктивність виконання команд та прискорює цикл розробки;

- прозора інтеграція з Gradle. Mainframer інтегрується із системою складання Gradle, яка широко використовується у розробці Android-додатків. Це забезпечує природну та прозору інтеграцію з існуючими проектами;

- підвищення продуктивності. Оскільки Mainframer виконує збирання на віддаленій машині, він може використовувати потужні сервери, що прискорює збирання та знижує тимчасові витрати;

- масштабованість. Mainframer легко масштабується для роботи з великими проектами та командами розробників. Це робить його ідеальним інструментом для великих та складних проектів;

- контроль версій. Mainframer надає засоби для контролю версій зібраних файлів, що корисно при спільній роботі та підтримці різних версій програми;

- зручність використання. Mainframer легко налаштовується та інтегрується у робочий процес розробки Android-додатків. Його файли

конфігурації зрозумілі і дозволяють розробникам визначити необхідні параметри збирання;

- ефективне обслуговування залежностей. Mainframer дозволяє ефективно керувати залежностями, встановленими на віддаленій машині, що сприяє спрощенню процесу збирання;

- підтримка кешування. Mainframer підтримує кешування залежностей, що дозволяє скоротити час збирання за рахунок повторного використання раніше зібраних даних;

- поліпшення спільної роботи. Mainframer сприяє більш ефективній спільній роботі, оскільки команди розробників можуть виконувати складання на одному віддаленому сервері, забезпечуючи консистентність результатів.

Головним у функціональності Mainframer є його інтеграція з rsync, потужним інструментом синхронізації файлів.

Rsync, відома утиліта синхронізації файлів, відіграє ключову роль в екосистемі Mainframer. Він використовується для синхронізації файлів між локальним середовищем розробки та призначеною віддаленою машиною. Ця синхронізація є вирішальним кроком, який забезпечує передачу лише необхідних файлів, мінімізує накладні дані та оптимізує ефективність процесу віддаленого збирання.

Однією з ключових переваг rsync є його здатність виконувати інкрементні передачі. У контексті Mainframer це означає, що передаються лише ті файли, які було додано або змінено після останньої синхронізації. Цей підхід значно скорочує час і ресурси, необхідні для передачі файлів, сприяючи загальній швидкості віддаленого збирання.

Rsync розроблено для оптимізації використання пропускну здатності під час синхронізації файлів. Передаючи лише відмінності між файлами, він мінімізує обсяг даних, що передаються через мережу. Це особливо вигідно в сценаріях, де

пропускна здатність є обмежуючим фактором, сприяючи більш швидкому та гнучкому віддаленому процесу створення.

Інтеграція rsync у структуру Mainframer є прикладом стратегічного рішення щодо оптимізації процесу віддаленого збирання. Використання rsync для синхронізації файлів забезпечує ефективність, мінімізує передачу даних і сприяє загальній швидкості та оперативності віддаленої системи збирання. Ця синергія між Mainframer і rsync формує міцну основу для гнучкої та ефективної віддаленої розробки.

В цілому, Mainframer є потужним засобом для віддаленого складання Android-додатків, який підвищує ефективність, прискорює процес розробки та знижує навантаження на локальну машину розробника. Це інструмент, який може значно покращити досвід розробки Android додатків.

Недоліки Mainframer:

- складність використання в додатках Android. Одним із головних недоліків Mainframer є його складність і складність інтеграції в робочі процеси розробки додатків Android. Проекти Android часто включають безліч залежностей, налаштованих конфігурацій і файлів ресурсів. Адаптувати Mainframer для бездоганної роботи з цими складнощами може бути складно, вимагаючи глибокого розуміння як розробки Android, так і самого Mainframer;

- робота на основі консолі. Mainframer працює в основному з командного рядка або консолі, що може не узгоджуватися з уподобаннями розробників, які звикли до графічних інтерфейсів користувача (GUI). Робота на основі консолі може бути складною для тих, хто має обмежений досвід роботи з інструментами командного рядка, потенційно створюючи перешкоди для входу для нових користувачів;

- обмежене налаштування. Mainframer, хоча й ефективний у розвантаженні завдань Gradle, пропонує обмежені можливості налаштування для проектів Android. Інструмент може бути непростим для проектів із

вужькоспеціалізованими конфігураціями або складними конвеєрами побудови. Це обмеження може бути значною перешкодою для розробників, яким потрібен детальний контроль над процесами збирання.

1.3 Аналіз систем дистанційного збирання додатків

Процес збирання додатків Android є критичним етапом у розробці мобільного програмного забезпечення. Вибір між локальним і віддаленим збиранням має глибокий вплив на ефективність, масштабованість і потенціал співпраці проектів розробки Android. У цьому розділі дипломної роботи ми розглянули різні системи для збирання додатків Android та аналог віддаленого збирання, зосередивши увагу на Mainframer, інструменті з відкритим вихідним кодом, призначеному для перевантаження збірок Android на віддалені сервери і визначили його недоліки.

Під час нашого аналізу ми виявили інструмент із відкритим кодом, Mainframer. Цей інструмент, хоч і перспективний у віддаленому збиранні, має певні недоліки, які підкреслюють потребу в більш універсальній і зручній альтернативі.

Mainframer — це відомий інструмент у сфері дистанційного збирання програм Android, що пропонує потенціал для оптимізації використання ресурсів і скорочення часу створення. Однак він не позбавлений недоліків. Складність інтеграції Mainframer з проектами Android, його робота на основі консолі та обмежені можливості налаштування створюють проблеми для розробників, які шукають просте та адаптоване рішення для віддаленого збирання. Хоча Mainframer обіцяє, він може бути не найкращим варіантом для всіх сценаріїв розробки Android. Тому розробники повинні ретельно оцінити вимоги до своїх проектів і ознайомитися з інструментами командного рядка, перш ніж використовувати Mainframer у своїх робочих процесах розробки. Оскільки

екосистема розробки Android продовжує розвиватися, усунення цих недоліків могла б підвищити цінність Mainframer як інструменту віддаленого збирання програм Android.

Виявлені недоліки включають:

- комплексна інтеграція. Аналоговий інструмент Mainframer, виявив складність під час інтеграції з проектами Android. Програми для Android часто містять складні конфігурації, спеціальні залежності та файли ресурсів. Труднощі з узгодженням інструменту з цими тонкощами можуть перешкодити прийняттю;
- робота на основі консолі. Розглянутий аналог працює тільки з командного рядка. Цей підхід на основі консолі може стати перешкодою для розробників, які більше звикли до графічних інтерфейсів користувача, ускладнюючи адаптацію до інструментів командного рядка;
- обмежене налаштування. Розглянутий інструмент пропонував обмежені можливості налаштування для проектів Android, особливо тих, які мають спеціалізовані конфігурації або складні конвеєри збірки. Негнучкість інструменту може не задовольнити потреби розробників, які прагнуть до детального контролю над процесом збірки.

1.4 Висновок по розділу

У світлі результатів аналізу було прийнято рішення розпочати розробку власної системи дистанційного збирання додатків для усунення виявлених обмежень, визначених у розглянутого аналогу. Нова система запропонує комплексне рішення, яке відповідає різноманітним вимогам розробки програм для Android. Ключові особливості та міркування запропонованої нами системи включають:

- інтеграція графічного інтерфейсу користувача (GUI). Однією з головних відмінностей нашої системи буде повна інтеграція графічного інтерфейсу користувача в середовище розробки (IDE) Android. Цей графічний інтерфейс забезпечить інтуїтивно зрозумілий і зручний інтерфейс для розробників, усуваючи необхідність покладатися виключно на операції командного рядка. Подолаючи розрив між графічним інтерфейсом користувача та роботою на основі консолі, наша система прагне задовольнити вподобання та знання ширшого кола розробників;

- налаштування та розширення. Наша система збирання додатків надаватиме пріоритет гнучкості та налаштуванню. Розробники матимуть свободу адаптувати процес відповідно до своїх конкретних вимог проекту. Ця гнучкість включатиме можливість визначати користувацькі конфігурації, адаптуватися до складних конвеєрів побудови та легко інтегрувати спеціалізовані компоненти;

- відкритий і безкоштовний. Система дотримуватиметься принципів програмного забезпечення з відкритим кодом, гарантуючи, що воно залишається доступним і прозорим для спільноти розробників Android. Пропонуючи безкоштовне рішення з відкритим кодом, ми маємо намір розширити можливості розробників, зменшити фінансові бар'єри та сприяти співпраці в екосистемі розробки Android;

- розвиток, керований громадою. Бачення системи створення додатків полягає в тому, щоб зробити її проектом, керованим спільнотою. Співпраця та внесок розробників Android з усього світу керуватимуть процесом розробки. Залучаючи спільноту розробників Android, ми прагнемо забезпечити, щоб система відповідала найнагальнішим потребам і адаптувалася до ландшафту розробки Android, що розвивається.

Нова система дозволить паралельно виконувати задачі від різних користувачів.

Одна потужна серверна машина зможе закрити проблему продуктивності цілого відділу (команди), навіть за наявності малопродуктивних локальних машин.

2 РОЗРОБКА СИСТЕМИ

2.1 Вибір програмного забезпечення

В епоху цифрової трансформації та вибухового зростання технологій, вибір програмного забезпечення стає ключовим стратегічним моментом для організацій та розробників. Від ретельно продуманих рішень у цій галузі залежить ефективність бізнес-процесів, конкурентоспроможність продуктів і, зрештою, успіх проектів. У цій роботі ми зосереджуємо увагу на одному з критично важливих аспектів цього вибору – виборі відповідного програмного забезпечення для розробки Android-додатків.

Створення програм для Android-платформи стало невід'ємною частиною сучасної індустрії інформаційних технологій. У процесі розробки різні інструменти програмного забезпечення відіграють вирішальну роль у формуванні результативності, надійності та якості додатків. Від вибору мови програмування до інструментів керування версіями та систем збирання, кожен елемент впливає на процес розробки і, отже, на кінцевий продукт.

У цьому розділі ми зосередимося на важливості вибору відповідного програмного забезпечення при розробці та збиранні додатків Android. Розглянемо різні аспекти цього вибору, включаючи мови програмування та інші ключові компоненти. Щоб надати всебічний огляд того, як ці рішення можуть впливати на процес розробки та результативність проектів.

2.1.1 Вибір мови програмування

В епоху інформаційних технологій, що стрімко розвиваються, вибір мови програмування стає важливим стратегічним рішенням при розробці програмного

забезпечення. Це рішення надає глибоке впливом геть процес розробки, продуктивність, супровід докладання, і зрештою, з його успіх у динамічній сфері інформаційних технологій. В рамках даного дипломного дослідження, фокусуємо увагу на одному з ключових аспектів цього вибору – мовою програмування розробки Android-додатків.

Android, як одна з наймасштабніших і найвпливовіших мобільних платформ, надає розробникам великі можливості і, отже, створює потребу у правильному виборі інструментів. Зокрема, рішення між двома провідними мовами програмування – Java та Kotlin – стає ключовим моментом при старті проекту. Обидві ці мови пропонують унікальні можливості та мають свої сильні та слабкі сторони, що робить вибір між ними значним етапом на початку розробки.

У цьому розділі дипломної роботи буде розглянута важливість вибору відповідної мови програмування для розробки Android-додатків. Особливості та переваги кожної з цих мов, будеи проведено аналіз їх застосування та вплив цього вибору на кінцевий результат. Метою аналізу є надання повного огляду аспектів, які слід врахувати при прийнятті рішення про мову програмування для створення Android-додатків та надання рекомендацій, які можуть допомогти розробникам приймати обґрунтовані рішення у своїй практиці.

Java-це класна об'єктно-орієнтована мова програмування високого рівня, розроблена для того, щоб зробити ваш додаток максимально залежним. Це означає, що скомпільований код Java може виконуватися на будь-якій платформі з підтримкою Java без перекомпіляції. Програми Java зазвичай компілюються в байт-код і можуть запускатися на будь-якій віртуальній машині Java (JVM), незалежно від базової архітектури комп'ютера. Синтаксис Java схожий на синтаксис C і C++, але він має менше функцій нижчого рівня, ніж обидва. Час виконання Java надає динамічні функції (такі як перегляд та зміна виконуваного коду), які зазвичай недоступні в традиційних компільованих мовах.

Java набула популярності незабаром після випуску, і з того часу була дуже популярною мовою програмування. За даними GitHub, Java була третьою за популярністю мовою програмування у 2022 році, а станом на жовтень 2023 року

вона посідає четверте місце за індексом TIOBE. Незважаючи на те, що Java все ще широко популярна, останніми роками спостерігається поступове зниження використання Java, а інші мови, які використовують JVM, набувають популярності.

Java була вперше розроблена Джеймсом Гослінгом із Sun Microsystems. Він був випущений в 1995 році 5 травня як основний компонент платформи Sun Microsystems Java. Оригінальний компілятор Java, віртуальна машина та бібліотека класів спочатку були випущені Sun під власною ліцензією. Станом на травень 2007 року, відповідно до специфікацій процесів спільноти Java, Sun повторно ліцензувала більшість технологій Java під GPL-2.0. Хоча Oracle надає власну віртуальну машину Java для гарячих точок, офіційною довідковою реалізацією є віртуальна машина OpenJDK, яка є безкоштовним програмним забезпеченням з відкритим кодом, що використовується більшістю розробників, і є віртуальною машиною за замовчуванням майже для всіх дистрибутивів Linux.

Станом на 2023 рік Java21 є останньою версією, а також довгостроковою версією підтримки (LTS). Java8, 11 і 17-попередні версії lts, які досі офіційно підтримуються.

Мова Java є важливою основою Android, мобільної операційної системи з відкритим кодом. Побудований поверх ядра Linux, Android написаний переважно мовою C, тоді як Android SDK використовує мову Java як основу для додатків Android, але не використовує стандартний графічний інтерфейс, SE, ME або інші вбудовані стандарти Java. Мова байт-коду, що підтримується Android SDK, несумісна з байт-кодом Java і працює на власній віртуальній машині, оптимізованій для пристроїв з невеликим обсягом пам'яті, таких як смартфони та планшетні комп'ютери. Залежно від версії Android, байт-код або інтерпретується віртуальною машиною Dalvik, або компілюється у власний код виконання Android.

Android SDK включає окрему реалізацію великої його підмножини, але Android не надає повну стандартну бібліотеку Java SE. Він підтримує Java 6 та деякі функції Java 7 та забезпечує реалізацію, сумісну зі стандартною бібліотекою (Apache Harmony).

Kotlin – це кросплатформна, статично набрана, універсальна мова програмування високого рівня з виведенням типів. Kotlin розроблений для повної взаємодії з Java, а JVM-версія стандартної бібліотеки Kotlin залежить від бібліотеки класів Java, яка простіша за синтаксисом через визначення типу Kotlin в першу чергу призначений для JVM, але не призначений для машинного коду через JavaScript (для інтерфейсних веб-додатків, що використовують React, наприклад) або LLVM. (Наприклад, витрати на розробку мови несе JetBrains, а фонд Kotlin захищає торгову марку Kotlin.)

У 2019 році Google оголосила, що мова програмування Kotlin стала найкращою мовою для розробників додатків для Android. Після виходу Android Studio 3.0 Kotlin був включений як альтернатива стандартному компілятору Java. Компілятор Android Kotlin за замовчуванням створює байт-код Java8 (який буде працювати в пізніших JVM), але програмісти можуть вибрати оптимізацію до 20 Java9 або дозволити більше функцій. Він підтримує сумісність класів двонаправлених записів JVM, введених у Java 16. Це вважається стабільним у Kotlin 1.5.

Kotlin має підтримку для web за допомогою Kotlin/JS або через класичний бекенд на основі інтерпретатора, який оголошено стабільним з версії 1.3, або через проміжний бекенд на основі представлення, який оголошено стабільним з версії 1.8. Kotlin/Native (наприклад, для підтримки Apple Silicon) вважається бета-версією з версії 1.3.

Коли Kotlin був оголошений офіційною мовою розробки Android в Google I/O У 2017 році, він став 3-ю повністю підтримуваною мовою Android після Java та C++. Станом на 2020 рік Kotlin є найбільш широко використовуваною мовою на Android, і за оцінками Google, 1000 найпопулярніших додатків Play Store написані на Kotlin, 70% з яких написані на Kotlin. Сама Google має 60 програм, написаних на Kotlin, включаючи карти та диски. Багато додатків для Android, таких як Google Home, переходять на Kotlin, тому вони використовують як Kotlin, так і Java. Вважається, що Kotlin для Android є корисним за безпеку нульового вказівника, а також за його функції, які роблять код коротшим і більш читабельним.

У 2020 році під час опитування розробників, які використовують Kotlin, JetBrains виявив, що 56% використовували Kotlin для мобільних додатків, тоді як 47% використовували його для веб-сервера. Трохи більше третини всіх розробників Kotlin сказали, що вони переходять на Kotlin з іншої мови. Більшість користувачів Kotlin орієнтувалися на Android (або на JVM), лише 6% використовували Kotlin Native.

Kotlin, сучасна та експресивна мова програмування, набула популярності як кращий вибір для розробки додатків Android завдяки своїй лаконічності та сумісності з Java. Однак важливо заглибитися в тонкощі процесу компіляції Kotlin, зокрема його перекладу в байт-код Java, і зрозуміти, як цей крок впливає на процес створення програми Android.

Kotlin розроблено для повної взаємодії з Java, що дозволяє розробникам легко інтегрувати код Kotlin і Java в одному проекті. Для досягнення такої сумісності вихідний код Kotlin компілюється в байт-код Java, проміжну мову, яку розуміє віртуальна машина Java (JVM). Потім цей байт-код виконується на JVM, дозволяючи коду Kotlin і Java співіснувати та безперебійно взаємодіяти.

Незважаючи на те, що компіляція Kotlin у байт-код Java сприяє взаємодії мови та сумісності з існуючими бібліотеками Java, вона вводить певні міркування, зокрема в контексті процесу збирання для програм Android.

Одним помітним недоліком є потенційне збільшення часу виготовлення. Трансляція коду Kotlin у байт-код Java додає до процесу збірки додатковий етап компіляції. Цей крок включає не тільки компілятор Kotlin, але й подальшу обробку згенерованого байт-коду компілятором Java. Отже, система збірки повинна впоратися з тонкощами керування вихідними кодами Kotlin і Java, сприяючи більшій тривалості збірки.

Оскільки Kotlin зрештою покладається на байт-код Java для виконання, будь-які затримки в процесі компіляції Java безпосередньо впливають на загальний час збірки проектів Android на основі Kotlin. Потреба в синхронізації між компонентами Kotlin і Java може створити складності у вирішенні залежностей і може призвести до каскадного впливу на ефективність збірки.

Щоб усунути підвищене навантаження на збірку, створене компіляцією Kotlin для байт-коду Java, розробники та системи автоматизації збірки можуть вивчити оптимізацію. Це може включати використання методів інкрементної компіляції, механізмів кешування або прийняття систем збірки, спеціально розроблених для роботи з проектами зі змішаними мовами.

У той час як компіляція Kotlin у байт-код Java покращує сумісність Kotlin і Java у розробці Android, це приносить компроміс через потенційно тривалий час збирання. Розробники повинні усвідомлювати цей вплив і застосовувати стратегії для оптимізації процесу створення. Розуміння тонкощів інтеграції Kotlin з екосистемою Java дає цінну інформацію для планування та виконання проекту, дозволяючи розробникам знайти баланс між функціями мови та ефективністю створення в контексті розробки додатків для Android.

Порівняння Java та Kotlin для розробки віддаленого збирання Android-додатків є важливим аспектом вибору технології в контексті проекту. Обидві мовні платформи пропонують низку переваг, але мають свої особливості. Розглянемо ключові аспекти порівняльного аналізу.

Переваги Java:

- велика спільнота та екосистема. Java має широке співтовариство розробників та багату екосистему бібліотек та інструментів, що спрощує розробку та підтримку проектів;
- широка підтримка інструментів складання. Багато інструментів складання, таких як Gradle та Maven, широко підтримують Java, що полегшує інтеграцію в існуючі системи;
- кросплатформність. Java є кросплатформною мовою програмування, що забезпечує переносимість коду між різними операційними системами.

Недоліки Java:

- великий обсяг коду. Java, як правило, вимагає більше рядків коду для досягнення тих же результатів, ніж Kotlin. Це може збільшити обсяг коду, який потрібно передавати на віддалену машину для збирання.

Переваги Kotlin:

- короткість коду. Kotlin пропонує більш компактний і виразний синтаксис порівняно з Java, що призводить до зменшення обсягу коду і, отже, скорочення часу передачі файлів на віддалену машину;
- 100% сумісність із Java. Kotlin повністю сумісний із Java, що забезпечує можливість інтеграції з існуючими Java-проектами та бібліотеками;
- корутини для асинхронного програмування. Kotlin надає корутини, що робить асинхронне програмування більш простим та зручним, що може бути важливим при розробці віддаленого збирання з використанням асинхронних завдань.

Недоліки Kotlin:

- зменшення швидкості виконання. У певних сценаріях виконання Kotlin-коду може бути трохи повільніше порівняно з Java. Однак, для завдання складання, де швидкість виконання не є першочерговою вимогою, це може мати невелике значення.

Висновки про перевагу Kotlin перед Java для розробки Android-додатків можуть ґрунтуватися на кількох ключових аспектах, враховуючи вимоги проекту та переваги команди розробників:

- сучасний синтаксис і виразність. Kotlin пропонує сучасніший і виразніший синтаксис, що робить код більш читаним і дозволяє розробникам писати лаконічніший код. Це особливо важливо в умовах, коли ясність та стислість можуть підвищити продуктивність та полегшити супровід коду;

- безпека та нульові значення за промовчанням. Kotlin включає деякі інноваційні функції, такі як система типів з підтримкою нульових значень за замовчуванням. Це зменшує кількість помилок, пов'язаних із нульовими показниками, що підвищує безпеку коду;
- робота з асинхронним кодом. В Kotlin присутні корутини, що надають зручний спосіб роботи з асинхронним кодом. Вони надають ефективніший і зрозуміліший механізм обробки асинхронних операцій порівняно з Java;
- повна сумісність з Java. Kotlin повністю сумісний з Java, що забезпечує плавну інтеграцію з існуючим Java-кодом та бібліотеками. Це означає, що команди можуть поступово переходити на Kotlin, не втрачаючи можливості використання коду;
- активна підтримка Google. Google оголосив Kotlin офіційною мовою розробки для Android, що наголошує на його значущості та довгостроковій підтримці з боку ключового гравця у світі мобільних технологій.

З усіх зазначених причин, у контексті розробки Android-додатків, вибір Kotlin може бути кращим. Важливо відзначити, що ці висновки можуть бути контекстуальними та залежати від конкретних вимог проекту та рівня досвіду команди розробників. Зрештою, ухвалення рішення на користь Kotlin зумовлене бажанням використовувати передові технології, покращити якість коду та підвищити продуктивність розробки.

Слід ще зазначити сучасність Kotlin та наявність зручних інструментів.

Jetpack Compose, інноваційний набір інструментів інтерфейсу користувача для створення власних додатків Android, революціонізує спосіб створення розробниками інтерфейсів користувача. В основі Compose полягає його унікальний підхід до створення інтерфейсу користувача, який використовує виразний синтаксис Kotlin і представляє потужну концепцію, відому як доменно-специфічна мова (DSL) для інтерфейсу користувача. Kotlin Compose

використовує генерацію коду та парадигму DSL для полегшення стислого та декларативного визначення інтерфейсів користувача.

Kotlin Compose використовує можливості процесорів анотацій, особливості компілятора Kotlin. Процесори анотацій аналізують анотації коду під час компіляції та генерують додатковий код на основі цих анотацій. У випадку Compose ці процесори відіграють ключову роль у створенні необхідного коду, пов'язаного з інтерфейсом користувача, на основі декларативних специфікацій інтерфейсу розробника.

Compose представляє DSL, який дозволяє розробникам описувати структуру інтерфейсу користувача та поведінку декларативним способом. Цей DSL — це набір функцій і розширень Kotlin, які абстрагуються від складнощів основної структури інтерфейсу користувача. Замість того, щоб обов'язково вказувати, як інтерфейс користувача має змінюватися у відповідь на події, розробники оголошують бажаний стан інтерфейсу користувача, а Compose піклується про відповідне оновлення інтерфейсу користувача.

У Compose інтерфейс користувача структурований як ієрархія складових функцій. Кожна складена функція представляє елемент UI або частину UI. Компоновані об'єкти складаються разом, щоб створити повну ієрархію інтерфейсу користувача. Ієрархія відображає структуру інтерфейсу користувача, що полегшує роздуми про макет і поведінку інтерфейсу користувача.

Compose DSL підтримує реактивність, дозволяючи інтерфейсу користувача автоматично оновлюватися у відповідь на зміни в основних даних. Коли дані змінюються, Compose інтелектуально переоцінює відповідні складові функції та оновлює лише ті частини інтерфейсу користувача, які змінилися. Цей реактивний підхід усуває потребу в ручному оновленні інтерфейсу користувача та покращує загальну швидкість реагування програми.

Парадигма DSL у Compose забезпечує стислий і читабельний код інтерфейсу користувача. Розробники можуть створювати складні структури інтерфейсу користувача за допомогою мінімальних шаблонів, зосереджуючись на декларативному характері визначення інтерфейсу користувача. Це не тільки

покращує підтримку коду, але й покращує співпрацю між членами команди, надаючи чітке та виразне представлення інтерфейсу користувача.

Під час компіляції Compose DSL проходить сувору перевірку під час компіляції. Структура DSL перевіряється, гарантуючи, що вказана ієрархія інтерфейсу користувача відповідає очікуваному формату. Ця рання перевірка мінімізує помилки під час виконання, пов'язані зі складом інтерфейсу користувача та поведінкою.

Процесори анотацій аналізують код Compose DSL і генерують відповідний код, пов'язаний з інтерфейсом користувача. Це включає створення елементів інтерфейсу користувача, обробку обмежень макета та інтеграцію з базовими API платформи. Згенерований код інкапсулює логіку, необхідну для відтворення інтерфейсу користувача та реагування на взаємодії користувача.

Використання Kotlin Compose генерації коду та парадигми DSL знаменує зміну парадигми в розробці інтерфейсу Android. Використовуючи виразну силу Kotlin і застосовуючи декларативний підхід, Compose дає розробникам змогу створювати складні та реактивні інтерфейси користувача з безпрецедентною лаконічністю. Синергія між генерацією коду, Compose DSL і процесом компіляції формує основу для сучасного та ефективного досвіду розробки інтерфейсу користувача у рідних програмах Android.

Щодо зручних та сучасних інструментів Kotlin також варто згадати про ProGuard – інструмент для стиснення коду та обфускації.

Kotlin часто вибирають для розробки Android завдяки його сучасним функціям і повній сумісності з Java. Щоб захистити конфіденційний код, зменшити розмір програми та підвищити продуктивність, розробники часто використовують ProGuard. Kotlin у поєднанні з ProGuard захищає та оптимізує код.

ProGuard спеціалізується на скороченні коду, видаленні невикористаних класів, полів і методів зі скомпільованого коду. В екосистемі Kotlin це особливо вигідно для додатків Android, де зменшення розміру файлу APK має вирішальне значення для ефективного розповсюдження та взаємодії з користувачем.

Однією з основних функцій ProGuard є обфускація імен, яка передбачає перейменування класів, методів і полів на загадкові та неописові імена. Ця обфускація ускладнює зворотне проектування та захищає інтелектуальну власність, приховуючи оригінальну структуру та зміст коду.

ProGuard містить різні методи оптимізації, такі як інлайнінг, постійне згортання та усунення мертвого коду. Ці оптимізації, застосовані до коду Kotlin, підвищують продуктивність програми під час виконання, мінімізуючи надлишкові обчислення та усуваючи недоступні шляхи коду.

Щоб інтегрувати ProGuard із проектом Kotlin, розробники створюють файл конфігурації ProGuard (`proguard-rules.pro`). Цей файл містить правила, які вказують ProGuard щодо того, як зменшувати, обфускувати та оптимізувати код. Специфічні міркування Kotlin, такі як збереження анотацій і відображення, часто включені в цю конфігурацію.

Kotlin представляє додаткові метадані, які допомагають у таких функціях, як відображення та серіалізація. Розробникам потрібно налаштувати ProGuard, щоб зберегти ці метадані Kotlin, щоб гарантувати, що функції, які покладаються на відображення під час виконання, як-от класи даних Kotlin, функціонують правильно після обфускації коду.

Для проектів Android інтеграція ProGuard спрощена через плагін Android Gradle. Розробники вказують файл конфігурації ProGuard у сценарії збірки Gradle проекту, а плагін автоматизує застосування ProGuard до скомпільованого коду під час процесу збирання.

Підтримка відображення та динамічних функцій у Kotlin може створити проблеми під час обфускації коду. Розробники повинні ретельно налаштувати ProGuard, щоб зберегти необхідні метадані та анотації, щоб уникнути проблем під час виконання.

Використовуючи бібліотеки сторонніх розробників у проекті Kotlin, розробники повинні переконатися, що конфігурації ProGuard для цих бібліотек вказано правильно. Це вкрай важливо для підтримки сумісності та запобігання проблемам, що виникають через стиснення коду та обфускацію.

Kotlin і ProGuard разом утворюють надійний дует для захисту та оптимізації коду в проектах Android. Здатність ProGuard скорочувати, обфускувати та оптимізувати код бездоганно узгоджується з сучасними функціями мови Kotlin. Інтегрувавши ProGuard у процес збірки та відповідним чином налаштувавши його, розробники можуть досягти балансу між захистом інтелектуальної власності, зменшенням розміру програми та підвищенням продуктивності під час виконання своїх програм для Android на основі Kotlin.

2.1.2 Вибір операційної системи

Розробка програмного забезпечення для мобільних пристроїв, створення ефективних та надійних Android-додатків потребує ретельного планування та оптимізації кожного етапу процесу. Одним із ключових компонентів цього процесу є вибір операційної системи для окремої машини, призначеної для збирання Android-додатків.

Операційна система на цій окремій машині виступає в ролі фундаментального будівельного блоку, що забезпечує стабільність, продуктивність та сумісність із інструментами розробки. У цьому розділі дипломної роботи буде розглянута важливість цього вибору та вплив операційної системи на процес складання Android-додатків.

Буде проведено аналіз різних операційних систем, розглянуті їх переваги та обмеження, а також визначені фактори, які слід враховувати під час ухвалення рішення. Мета цього аналізу – надати повне розуміння того, як вибір операційної системи може вплинути на ефективність та якість складання Android-додатків, а також надати рекомендації для оптимального вибору, що відповідає конкретним потребам розробників та проекту.

Розглянемо найпопулярніші варіанти операційних систем.

Windows – одна з найпопулярніших операційних систем, розроблена корпорацією Microsoft. Windows широко використовується в різних сферах, включаючи домашні комп'ютери, офісні робочі станції та сервери. Вона пропонує великі можливості, підтримує безліч додатків та ігор.

Переваги Windows:

- широка сумісність. Windows підтримує більшість програм та драйверів, що робить її універсальним вибором для користувачів;
- простота використання. Інтуїтивний інтерфейс та звичне оточення роблять Windows доступним для широкого кола користувачів.

Недоліки Windows:

- безпека. Windows піддається вірусам та іншим шкідливим програмам, що потребує регулярних оновлень та антивірусних програм;
- платна. Windows є комерційною операційною системою, що може створювати додаткові витрати.

Особливості для програмістів:

- обмежена сумісність з інструментами Linux. деякі інструменти, популярні серед програмістів, краще працюють в середовищі Linux;
- не ідеально для розробки мобільних програм. Windows не завжди є кращою платформою для розробки мобільних програм, особливо в порівнянні з MacOS.

Mac OS – операційна система, розроблена Apple, яка використовується на комп'ютерах Mac. Mac OS відома своїм стильним інтерфейсом, високою продуктивністю та інтеграцією з іншими продуктами Apple.

Переваги Mac OS:

- екосистема Apple. Mac OS добре взаємодіє з іншими пристроями Apple, створюючи єдину екосистему;
- стабільність та продуктивність. Оптимізована робота з апаратними засобами Mac забезпечує високу продуктивність.

Недоліки Mac OS:

- висока вартість. Пристрої з Mac OS, зокрема, комп'ютери Mac, часто коштують дорожче за аналогічні пристрої з Windows;

- обмежена сумісність. Mac OS призначена в основному для використання на пристроях Apple, що може створювати складнощі при взаємодії з не Apple обладнанням.

Особливості даної задачі:

- обмеженість у виборі пристроїв. Mac OS призначена виключно для пристроїв Apple, що робить її менш універсальною для використання на не Apple комп'ютерах;

- не оптимальність для розробки не-Apple додатків. У випадку, коли розробка додатків не передбачає широкої екосистеми Apple, використання Mac OS може бути зайвим і неефективним з фінансової точки зору.

Ubuntu – один із найпопулярніших дистрибутивів операційної системи Linux. Він відомий своєю безплатністю, відкритим вихідним кодом та широкою спільнотою розробників.

Переваги Ubuntu:

- безкоштовна. Ubuntu є безкоштовною операційною системою, що знижує витрати на початкове налаштування та супровід;

- відкритий код Ubuntu надає доступ до вихідного коду, що дозволяє користувачам модифікувати систему під свої потреби;

- широке співтовариство. Наявність широкого співтовариства розробників та користувачів забезпечує підтримку та вирішення можливих проблем.

Особливості для даної задачі:

- підходить для програмістів. Ubuntu надає широкий набір інструментів розробника, включаючи термінал, можливість встановлення та налаштування пакетів програмного забезпечення, а також доступ до багатьох мов програмування;
- відмінна сумісність з інструментами Linux. Для завдань, пов'язаних із розробкою та складанням Android-додатків, Ubuntu є відмінним вибором, тому що підтримує необхідні інструменти та бібліотеки.

Вибір відповідної операційної системи (ОС) є критичним рішенням під час створення спеціальної машини для віддаленого створення додатків Android. Цей висновок виступає за прийняття Ubuntu як кращої операційної системи з кількох вагомих причин, наголошуючи на її сумісності, безпеці, підтримці спільноти та бездоганній інтеграції з середовищем розробки Android.

Сумісність з Android Development:

- основи відкритого коду. Природа Ubuntu з відкритим вихідним кодом добре узгоджується з духом розробки Android. Він надає розробникам свободу перевіряти, змінювати та поширювати ОС, сприяючи співпраці та прозорості;
- керування пакетами. Advanced Package Tool (APT) на Ubuntu спрощує встановлення пакетів програмного забезпечення та керування ними. Розробка Android покладається на певний набір інструментів, а надійна система керування пакетами Ubuntu забезпечує безпроблемне встановлення та оновлення;
- сумісність. Ubuntu повністю підтримує Java, що є основною вимогою для розробки додатків Android. Сумісність гарантує, що розробники можуть легко працювати з Android Studio та іншими інструментами на основі Java, які є невід'ємною частиною екосистеми розробки Android.

Міркування безпеки:

- регулярні оновлення. Ubuntu дотримується передбачуваного графіка випусків і забезпечує регулярні оновлення безпеки. Ця відданість безпеці гарантує, що операційна система залишається стійкою проти нових загроз, критичних для машини, яка бере участь у життєвому циклі розробки;

- контроль доступу та дозволи. Ubuntu використовує надійну систему дозволів користувача, що знижує ризик несанкціонованого доступу або випадкових змін до критичних системних файлів. Це особливо важливо для підтримки цілісності машини віддаленого збирання;

- аудити безпеки, керовані спільнотою. Ubuntu отримує переваги від великої та активної спільноти, яка активно бере участь у перевірках безпеки. Цей спільний підхід покращує ідентифікацію та вирішення потенційних вразливостей, сприяючи більш безпечному операційному середовищу.

Підтримка спільноти та документація:

- жвава спільнота. Ubuntu може похвалитися яскравою та великою спільнотою користувачів і розробників. Ця підтримка, керована спільнотою, гарантує, що розробники мають доступ до великої кількості знань, форумів і ресурсів, сприяючи швидкому вирішенню проблем і обміну знаннями;

- комплексна документація. Спільнота Ubuntu приділяє велику увагу вичерпній документації. Ця велика кількість документації допомагає розробникам у встановленні та налаштуванні операційної системи для оптимальної продуктивності в контексті збірок додатків Android.

Повна інтеграція з інструментами розробки:

- сумісність з Android Studio. Ubuntu легко інтегрується з Android Studio, офіційною IDE для розробки Android. Ця сумісність гарантує, що розробники можуть використовувати повний набір інструментів розробки Android, не стикаючись із проблемами сумісності;

– ефективність терміналу та командного рядка. Інтерфейс командного рядка Ubuntu забезпечує потужне та ефективне середовище для розробників. Збірки Android часто включають операції командного рядка, а термінал Ubuntu розширює можливості розробника керувати процесом збирання та контролювати його.

Таким чином робимо висновок, що нам підійде саме Ubuntu яка поєднає в собі відкритість і безкоштовність Linux зі зручністю і функціональністю. Це може бути оптимальним вибором для розробників, особливо в контексті Android-додатків. Підсумовуючи, Ubuntu постає як надійна та добре підходяща операційна система для спеціальної машини, на яку поставлено завдання дистанційного створення додатків Android. Його сумісність із інструментами розробки Android, відданість безпеці, активна підтримка спільноти та бездоганна інтеграція роблять його оптимальним вибором. Оскільки екосистема Android продовжує розвиватися, рішення прийняти Ubuntu відображає стратегічний вибір, який узгоджується з потребами розробки сучасних програм для Android. Цей висновок підсилює важливість операційної системи в загальній архітектурі системи віддаленого збирання та виступає за Ubuntu як надійний і ефективний вибір для цього конкретного контексту.

2.1.3 Вибір середовища розробки

Розробка даної системи буде відбуватись в IDE Android Studio, яка є неперевершеним інструментом для розробки Android-додатків і не має аналогів.

Android Studio, розроблена корпорацією Google, виділяється серед інших інтегрованих середовищ розробки завдяки своїй функціональності, зручності використання та безкоштовності. Ця IDE стала безперечним стандартом в індустрії Android-розробки, забезпечуючи широкі можливості та інструменти, необхідні для ефективного створення високоякісних мобільних додатків.

Android Studio замінила плагін ADT для платформи Eclipse. Це середовище базується на вихідному коді продукту IntelliJ IDEA Community Edition, розробленого JetBrains. Android Studio була розроблена з використанням моделі розробки з відкритим вихідним кодом і поширюється під ліцензією Apache2.0.

Бінарні збірки доступні для Linux (для тестування використовувалася Ubuntu), macOS і Windows. Середовище також надає інструменти для розробки застосунків для переносних пристроїв на базі Wear OS, телевізорів (Android TV), Google Glass та автомобільних інформаційно-розважальних систем (Android Auto), а також смартфонів і планшетів. Для додатків, розроблених за допомогою Eclipse і ADT Plugin, доступні інструменти для автоматичного імпорту наявних проєктів в Android Studio [10].

Середовище розробки адаптоване для виконання типових завдань, які вирішуються в процесі розробки додатків для платформи Android. Середовище включає в себе інструменти, що спрощують тестування додатків на сумісність з різними версіями платформи, а також інструменти для проектування додатків, які запускаються на пристроях з екранами різного дозволу (планшети, смартфони, ноутбуки, годинники, окуляри і т.д.). Це гарна ідея. На додаток до функцій, доступних в IntelliJ IDEA, Android Studio включає нову підсистему інтеграції для створення, тестування і розгортання додатків на основі Gradle, що підтримує використання інструментів безперервної інтеграції, доступно низка додаткових функцій.

Для прискорення розробки додатків передбачений візуальний редактор для збору і компонування стандартних елементів інтерфейсу. Він забезпечує зручний попередній перегляд різних станів інтерфейсу програми (наприклад, різних версій Android та різних розмірів екрана), щоб побачити, який інтерфейс існує для різних розмірів екрана). Для створення інтерфейсу користувача у вас є майстер створення власних елементів дизайну, які підтримують використання шаблонів. Це середовище включає можливість завантаження типових прикладів коду з GitHub.

Крім того, в розширений інструментальний редактор був доданий новий режим швидкого редагування для проведення рефакторингу, перевірки сумісності з попередніми версіями, виявлення проблем з продуктивністю, моніторингу споживання пам'яті і оцінки простоти використання. Система виділення, статичного аналізу і виявлення помилок доповнена підтримкою Android API, вбудованою підтримкою Proguard Codeoptimiser і новими інструментами для

генерації цифрових підписів. Вбудований інструмент для генерації цифрових підписів. Є інтерфейс для управління перекладами на інші мови.

Переваги Android Studio:

- безкоштовність. Однією з ключових переваг Android Studio є його безкоштовність. Це робить його доступним для широкого кола розробників, незалежно від їхнього бюджету;
- інтеграція з Kotlin. Android Studio безперервно інтегрується з Kotlin, що робить його ідеальним інструментом для розробки Android-додатків цією мовою програмування. Це підтримує розробку сучасних, виразних та надійних додатків;
- візуальний редактор макетів. Вбудований візуальний редактор макетів полегшує створення інтерфейсу користувача, дозволяючи розробникам наочно проектувати і редагувати макети додатків;
- емулятори пристроїв. Android Studio надає потужні емулятори пристроїв, дозволяючи тестувати програми на різних версіях Android та роздільній здатності екранів;
- потужні засоби налагодження. Інструменти налагодження в Android Studio забезпечують докладні звіти та можливості відстеження помилок, спрощуючи процес виправлення багів та оптимізації програм;
- підтримка усіх Android-платформ. Android Studio забезпечує повну підтримку всіх платформ Android, включаючи телефони, планшети, Android TV та Wearables;
- інтеграція з Firebase. Інтеграція з платформою Firebase спрощує додавання до функціональності, такої як аналітика, хмарні служби та управління користувачами.

Вибір Android Studio як середовище розробки для нашого проекту обумовлений не лише широкою функціональністю та зручністю використання, а й вільним доступом для всіх розробників. Це інтегроване середовище розробки надає все необхідне для того, щоб з легкістю створювати та вдосконалювати Android-додатки, роблячи його оптимальним та незрівнянним вибором для успішного втілення наших розробних завдань.

2.2 Вибір апаратного забезпечення

Ефективність і продуктивність системи віддаленої збірки для програм Android нерозривно пов'язані з апаратним забезпеченням, на якому вона працює. Вибір апаратного забезпечення може значно вплинути на швидкість, масштабованість і надійність процесу віддаленого складання. У цьому розділі розглядаються найважливіші аспекти вибору відповідного апаратного забезпечення для машини, призначеної для віддаленого збирання, підкреслюючи його ключову роль в оптимізації робочого процесу розробки.

Фактори, що впливають на вибір обладнання:

- обчислювальна потужність. Складність створення додатків Android вимагає значної обчислювальної потужності. Вибране апаратне забезпечення має мати багатоядерні процесори, високу тактову частоту та достатню кількість оперативної пам'яті для виконання одночасної компіляції та ресурсомістких процесів, властивих створенню програм для Android;
- рішення для зберігання даних. Швидкий доступ до файлів проекту та залежностей має першочергове значення. Вибір високошвидкісних рішень для зберігання даних, таких як SSD, може значно скоротити час створення за рахунок мінімізації затримок читання та запису;

- можливості мережі. Надійна мережева інфраструктура має вирішальне значення для ефективної віддаленої збірки. Високошвидкісне підключення до Інтернету забезпечує швидку передачу даних між машиною розробника та віддаленим сервером збірки, мінімізуючи затримку та оптимізуючи процес збірки;
- графічний процесор (GPU). Хоча основна увага часто приділяється процесору та оперативній пам'яті, деякі процеси створення Android, особливо ті, що включають графічні ресурси, можуть отримати переваги від потужного графічного процесора. Це особливо актуально для додатків із розширеною графікою або тих, що використовують бібліотеки з інтенсивним використанням GPU;
- масштабованість. Обладнання слід вибирати з урахуванням масштабованості. Оскільки вимоги до проектів Android зростають, система повинна витримувати збільшене робоче навантаження. Масштабовані рішення, такі як хмарна інфраструктура, забезпечують гнучкість адаптації ресурсів відповідно до вимог проекту.

Вплив на продуктивність збірки:

- скорочення часу створення. Добре налаштована машина може значно скоротити час збірки, підвищивши продуктивність розробника. Взаємодія швидкого процесора, достатньої оперативної пам'яті та високошвидкісного накопичувача сприяє швидшій компіляції та складанню програм для Android;
- паралельні збірки. Ефективне апаратне забезпечення забезпечує одночасне збирання, дозволяючи розробникам працювати над кількома функціями або модулями одночасно без шкоди для продуктивності збірки. Це особливо важливо для великомасштабних проектів з кількома учасниками;

– надійність і послідовність. Надійне апаратне забезпечення забезпечує узгодженість результатів збірки. Постійний час створення та результати є критично важливими для створення стабільного та передбачуваного середовища розробки.

Вибір між локальними та хмарними рішеннями:

– локальна інфраструктура. Пропонує повний контроль над конфігурацією обладнання та протоколами безпеки. Ідеально підходить для організацій із особливими вимогами відповідності або для організацій із наявною локальною інфраструктурою;

– хмарні рішення. Забезпечує масштабованість і гнучкість. Розробники можуть використовувати хмарні служби для динамічного розподілу ресурсів на основі попиту. Це особливо вигідно для проектів із різним навантаженням.

Підсумовуючи, вибір апаратного забезпечення для машини, призначеної для віддаленого збирання додатків Android, є стратегічним рішенням, яке глибоко впливає на ефективність і результативність процесу розробки. Ретельно враховуючи такі фактори, як обчислювальна потужність, рішення для зберігання, мережеві можливості та масштабованість, розробники можуть створити оптимізоване середовище, яке сприяє швидкому та надійному створенню додатків Android. Цей розділ закладає основу для подальшого вивчення конфігурацій апаратного забезпечення, порівнюючи локальні та хмарні рішення, а також надає практичне розуміння аспектів апаратного забезпечення для надійної системи віддаленого збирання.

Таблиця 2.1 - Апаратне обладнання

Материнська плата	Gigabyte Z790 UD
Процесор	Intel Core i7-14700K BOX, 16 ядер по 3.4GHz
Відеоадаптер	Інтегрований на материнській платі

Оперативна пам'ять	32 DDR5 5600 МГц
Накопичувач	Kingston KC3000 SKC3000D 2 TB
Блок живлення	Chieftec Polaris 1050 Вт
Операційна система	Ubuntu

Продовження таблиці 2.1

Клавіатура/миша	Клавіатура PS/2, миша USB
Мережевий адаптер	Інтегрований адаптер Gigabit Ethernet на материнській платі
Монітор	«32» LG 32MN500M, 1920x1080 (16:9)

2.3 Розробка дистанційної системи

У світі розробки мобільних додатків, що постійно розвивається, ефективність робочого процесу розробки відіграє вирішальну роль в успіху проекту. В цьому розділі, в рамках цього дипломного проекту, починається розробка системи дистанційного збирання для додатків Android.

Система віддаленого збирання задумана як стратегічне рішення для вдосконалення процесу компіляції та збирання програм Android. Переносячи ці ресурсномісткі завдання на спеціальну віддалену машину, мета полягає в тому, щоб оптимізувати час створення, використовувати чудові можливості апаратного забезпечення та оптимізувати робочий процес розробки.

В основі цього проекту лежить ретельний відбір технологій, які відповідають сучасним найкращим практикам і галузевим стандартам. Android Studio, офіційне інтегроване середовище розробки для Android, слугуватиме центром розробки. Рішення реалізувати систему на Kotlin, сучасній та експресивній мові програмування, яка повністю сумісна з Java, забезпечує безперебійну та ефективну розробку.

У міру того, як програми стають складнішими, зростає і час створення, і це може перешкоджати ітераційному циклу розробки, важливого для швидкого створення прототипів і ефективного налагодження. Запровадження системи віддаленого збирання готує безпосередньо вирішити ці проблеми.

Розробка починається з етапу ініціалізації, який передбачає створення спеціального репозиторію, що закладає основу для середовища спільної розробки з керуванням версіями.

Android Studio з надійним набором інструментів і інтеграцій буде налаштовано відповідно до вимог проекту. Це включає налаштування необхідної структури проекту, визначення конфігурацій збірки та встановлення зв'язку з вибраною системою керування версіями.

Kotlin, відомий своєю лаконічністю, нульовою безпекою та повною сумісністю з Java, обрано як мову реалізації. Його виразний синтаксис і сучасні функції дозволяють розробникам створювати кодову базу, яка не тільки відповідає безпосереднім потребам проекту, але й позиціонує його для майбутньої масштабованості та зручності обслуговування.

Основною метою цього проекту є оптимізація процесів створення та збирання додатків Android. Запроваджуючи систему віддаленого збирання, ми прагнемо значно скоротити час компіляції, дозволяючи розробникам виконувати ітерацію швидше та покращуючи загальний досвід розробки.

Під час розвитку проекту наголос буде приділятися створенню середовища, сприятливого для розвитку співпраці. Інтеграція контролю версій у поєднанні з гнучкістю Kotlin спрямована на сприяння духу співпраці серед розробників, забезпечуючи гармонійний і ефективний робочий процес розробки.

Очікуваний вплив цього проекту виходить за межі компіляції коду. Оптимізована система збирання готова підвищити ефективність, масштабованість і співпрацю в групах розробників Android. Ця ініціатива узгоджується з ширшими галузевими цілями прискорення циклів розробки та сприяння інноваціям.

Android Studio - офіційне інтегроване середовище розробки (IDE) для Android, є обраною платформою для розробки проектів. Android Studio, відомий

своїм багатофункціональним середовищем, бездоганною інтеграцією з Gradle і надійними інструментами налагодження, забезпечує ідеальне налаштування для створення, вдосконалення та оптимізації програм Android.

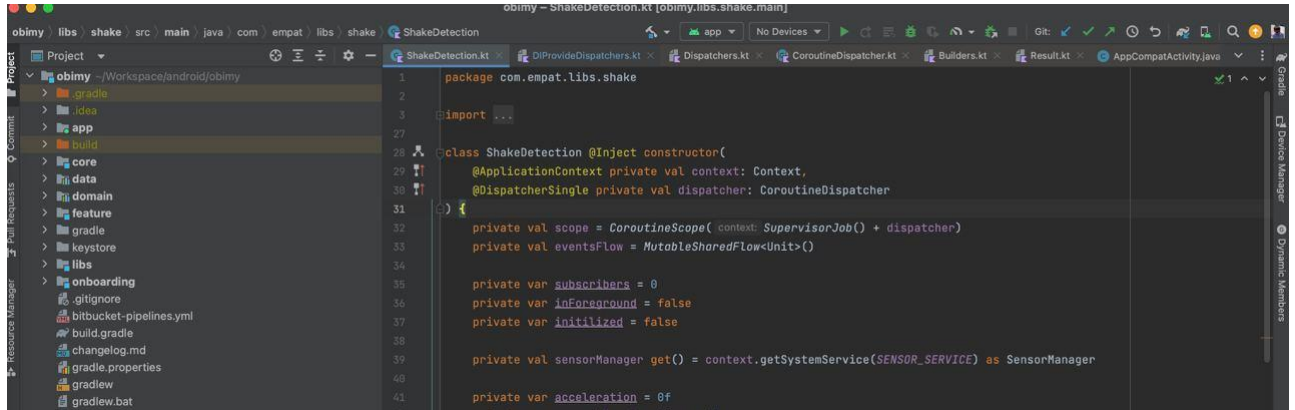


Рисунок 2.1 – Інтерфейс Android Studio

Початковий крок передбачає запуск Android Studio та навігацію через інтуїтивно зрозумілий інтерфейс користувача (рис. 2.1), щоб почати створення нового проекту Android. Цей етап спонукає розробника до прийняття критичних рішень щодо структури проекту, конфігурації та цільових версій платформи Android. Android Studio є зручним інструментом розробки, що дозволяє запускати процеси збірки, компіляції як з терміналу (рис. 2.2), так і з кнопок наявних на панелі користувача.

```

The default interactive shell is now zsh.
To update your account to use zsh, please run 'chsh -s /bin/zsh'.
For more details, please visit https://support.apple.com/kb/HT208950.
MacBook-Pro:obimy ovitaliy$ ./gradlew assembleDebug
Configuration on demand is an incubating feature.
Calculating task graph as no configuration cache is available for tasks: assembleDebug

> Task :app:mergeDevDebugResources
/Users/ovitaliy/Workspace/android/obimy/feature/achievements/ui/build/intermediates/packaged_res/devDebug/values/values.xml:13:4: Multiple substitutional format of string resource string/achievements_unlocked. Did you mean to add the formatted="false" attribute?
/Users/ovitaliy/.gradle/caches/transforms-3/615d8c2a287d45d04d8afc561d960570/transformed/material-1.9.0/res/values/values.xml:1275:4: Multiple substitutional format of string resource string/invite_with_random_message_format. Did you mean to add the formatted="false" attribute?
/Users/ovitaliy/.gradle/caches/transforms-3/615d8c2a287d45d04d8afc561d960570/transformed/material-1.9.0/res/values-es/values-es.xml:17:4: Multiple substitutional format of string resource string/invite_with_random_message_format. Did you mean to add the formatted="false" attribute?
/Users/ovitaliy/.gradle/caches/transforms-3/615d8c2a287d45d04d8afc561d960570/transformed/material-1.9.0/res/values-b+es+419/values-b+es+419.xml:17:4: Multiple substitutional format of string resource string/invite_with_random_message_format. Did you mean to add the formatted="false" attribute?
/Users/ovitaliy/.gradle/caches/transforms-3/615d8c2a287d45d04d8afc561d960570/transformed/material-1.9.0/res/values-uk/values-uk.xml:19:4: Multiple substitutional format of string resource string/invite_with_random_message_format. Did you mean to add the formatted="false" attribute?
  
```

Рисунок 2.2 – Приклад роботи терміналу в Android Studio

Налаштування параметрів проекту є ключовим аспектом створення проекту. Це включає в себе вказівку назви проекту, назви пакета та розташування файлів проекту. Крім того, на цьому етапі приймаються рішення щодо використання Kotlin як мови програмування, інтеграції керування версіями та конфігурації модулів.

Відповідно до сучасної практики розробки, інтеграція контролю версій стає першорядною. Android Studio легко інтегрується з популярними системами контролю версій, такими як Git. Ця інтеграція полегшує співпрацю, забезпечує історію версій і забезпечує систематичний підхід до керування вихідним кодом проекту.

Android Studio пропонує надійну підтримку розробки Kotlin, включаючи такі функції, як доповнення коду, інструменти рефакторингу та розширені можливості налагодження. Синергія між Kotlin і Android Studio прискорює процес розробки, надаючи можливість розробникам писати виразний і стислий код.

Ретельне тестування дуже важливе при розробці системи віддаленого збирання додатків Android для оцінки її ефективності. У цьому розділі розглядається методологія, яка використовується під час тестування системи, де утиліта gradle-profiler займає центральне місце. Gradle-profiler служить потужним інструментом для вимірювання продуктивності процесів збірки Gradle шляхом повторного виконання команд і фіксації точного часу виконання.

Основною метою тестування системи є оцінка ефективності та продуктивності системи віддаленого збирання за різними сценаріями. Використовуючи систему gradle-profiler, можна зібрати кількісні дані про час створення, використання ресурсів і загальну швидкість реагування системи.

Методологія тестування включає в себе організацію процесу збирання через gradle-profiler, виконання відповідних команд Gradle кілька разів і систематичне вимірювання часу, необхідного для кожного виконання. Цей ітераційний підхід забезпечує детальне розуміння продуктивності системи за різних умов.

Gradle-profiler легко інтегрується з системою збірки Gradle, діючи як розширення для покращення можливостей тестування (рис. 2.3). Ця інтеграція забезпечує сумісність із існуючими конфігураціями збірки, дозволяючи проводити ненав'язливий, але комплексний процес тестування.

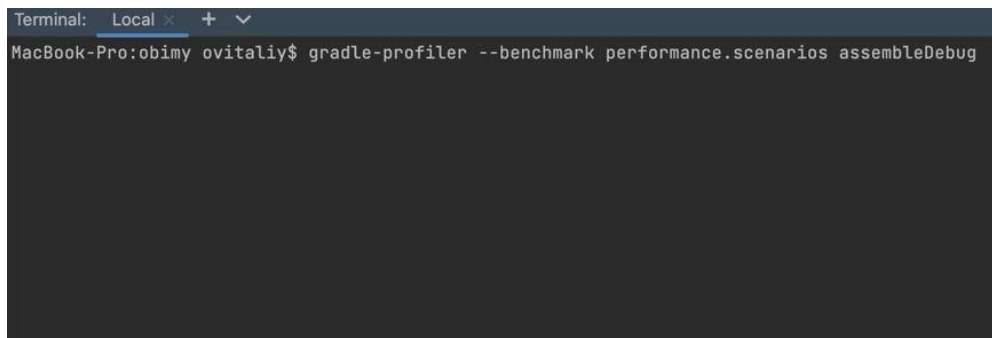


Рисунок 2.3 – Запуск gradle-profiler з терміналу в Android Studio

Утиліта gradle-profiler дозволяє налаштовувати певні команди Gradle, які будуть виконуватися під час тестування. У контексті системи віддаленого збирання націлені команди, пов'язані зі складанням, компіляцією та пакуванням програм Android. Ця можливість налаштування гарантує, що тестування узгоджується з реальними завданнями, які виконуються в реальних сценаріях.

Для тестування системи що розробляється, використовуються налаштування, які наведені у лістингу 2.1. Буде запускатись 3 команди.

Лістинг 2.1 – Тестування системи

```
assembleDebug {
    title = "assembleDebug Task"
    tasks = ["assembleDebug"]
    cleanup-tasks = ["clean"]
    daemon = warm
    warm-ups = 2
}

testDebugUnitTest {
    title = "testDebugUnitTest Task"
    tasks = ["testDebugUnitTest"]
    cleanup-tasks = ["clean"]
    daemon = warm
}
```

```

        warm-ups = 2
    }

    assembleRelease {
        title = "assembleRelease"
        tasks = ["assembleRelease"]
        cleanup-tasks = ["clean"]
        daemon = warm
        warm-ups = 2
    }

```

Однією з відмінних особливостей gradle-profiler є його здатність виконувати команди через декілька ітерацій. Запускаючи процес збирання кілька разів, утиліта фіксує мінливість часу збирання, забезпечуючи точніше представлення характеристик продуктивності системи. Цей ітеративний підхід допомагає визначати тенденції, викиди та потенційну оптимізацію.

Утиліта ретельно вимірює час, витрачений на кожну ітерацію процесу збирання. Потім ці дані адресуються та аналізуються для отримання суттєвої інформації про профіль продуктивності системи. Такі показники, як середній час створення, стандартне відхилення, сприяють повному розумінню поведінки системи.

Окрім часу збирання, gradle-profiler пропонує можливості моніторингу системних ресурсів під час тестування. Це включає відстеження використання ЦП, споживання пам'яті та інших відповідних показників. Кореляція між використанням ресурсів і часом створення надає цінну інформацію для оптимізації конфігурації системи.

Тестові сценарії ретельно розроблені для імітації моделей використання в реальному світі. Це включає варіації в розмірі проекту, складності та структурах залежностей. Сценарії також можуть імітувати одночасні збірки або зміни в специфікаціях обладнання, щоб оцінити адаптивність системи.

Змінність часу створення є природним аспектом розробки програмного забезпечення. Gradle-Profiler допомагає впоратися з цією мінливістю, фіксуючи статистичні заходи та тенденції. Відхилення від очікуваних шаблонів спонукають до подальшого дослідження потенційних вузьких місць або областей, які потрібно вдосконалити в системі віддаленого збирання.

Ітераційне виконання команд Gradle через gradle-profiler відкриває цінну інформацію про мінливість часу збирання. Проводячи кілька ітерацій, ми отримуємо детальну перспективу узгодженості та надійності віддаленої системи збірки. Цей ітеративний підхід є інструментальним у визначенні закономірностей, тенденцій і потенційної оптимізації.

Можливості моніторингу ресурсів, вбудовані в gradle-profiler, пропонують цілісне уявлення про динаміку системи. Кореляція між часом збірки та використанням ресурсів забезпечує глибше розуміння того, як віддалена система збірки взаємодіє з основним обладнанням. Шаблони використання ЦП, споживання пам'яті та інші показники керують рішеннями, пов'язаними з конфігурацією та масштабованістю системи.

Можливість конфігурації утиліти дозволяє проводити оцінки за конкретними сценаріями, віддзеркалюючи шаблони використання в реальному світі. Ця адаптивність має вирішальне значення для оцінки того, як система віддаленого збирання працює в різних розмірах проектів, складності та структурах залежностей. Оцінки на основі сценаріїв дозволяють нам адаптувати оптимізацію на основі конкретних вимог проекту.

Gradle-profiler діє як інструмент діагностики, допомагаючи у виявленні вузьких місць у системі віддаленого збирання. Відхилення від очікуваних показників продуктивності спонукають до глибшого дослідження першопричин, що сприяє цілеспрямованій оптимізації. Цей проєктивний підхід підвищує стійкість системи та адаптивність.

Кількісні дані, згенеровані gradle-profiler, служать дорожньою картою для можливостей оптимізації. Області, де час створення демонструє помітну мінливість або використання ресурсів неоптимальне, стають центрами для вдосконалення. Утиліта керує ітеративним процесом розробки, гарантуючи, що кожна оптимізація сприяє підвищенню загальної ефективності системи.

Інтеграція gradle-profiler у фазу тестування системи збагачує процес оцінювання, забезпечуючи тонкий погляд на продуктивність системи віддаленого збирання (рис. 2.4). Кількісні дані, зібрані за допомогою ітераційного тестування,

у поєднанні з моніторингом ресурсів полегшують прийняття обґрунтованих рішень щодо оптимізації системи, конфігурації та масштабованості.

```
BUILD SUCCESSFUL in 32s
1 actionable task: 1 executed

Publishing build scan...
https://gradle.int.tango.me/s/iypqyqgswswm

* Stopping daemons
Running command /Users/vrudenko/.gradle/wrapper/dists/gradle-8.2-bin/677s1h55ejkykupbl6lhmj6e/gradle-8.2/bin/gradle --stop --gradle-user-home /Users/vrudenko/.gradle
Stopping Daemon(s)
1 Daemon stopped

* Scenarios
Scenario: assembleDev21FullGoogleDebug Task using Gradle 8.2
Gradle 8.2 (/Users/vrudenko/.gradle/wrapper/dists/gradle-8.2-bin/677s1h55ejkykupbl6lhmj6e/gradle-8.2)
Run using: Tooling API
Run: run tasks :tango:assembleDev21FullGoogleDebug
Cleanup: run tasks clean
Gradle args: []
Build changes: []
Warm-ups: 2
Builds: 10

* Running scenario assembleDev21FullGoogleDebug Task using Gradle 8.2 (scenario 1/1)

* Stopping daemons
Running command /Users/vrudenko/.gradle/wrapper/dists/gradle-8.2-bin/677s1h55ejkykupbl6lhmj6e/gradle-8.2/bin/gradle --stop --gradle-user-home /Users/vrudenko/.gradle
No Gradle daemons are running.

* Build details
Gradle version: Gradle 8.2
Java home: /Applications/Android Studio.app/Contents/jbr/Contents/Home
OS: mac os x aarch64 13.5.2
JVM args:
  -noverify
  -XX:+HeapDumpOnOutOfMemoryError
  -XX:+UseParallelGC
  --add-opens=java.base/java.util=ALL-UNNAMED
  --add-opens=java.base/java.lang=ALL-UNNAMED
  --add-opens=java.base/java.lang.invoke=ALL-UNNAMED
  --add-opens=java.prefs/java.util.prefs=ALL-UNNAMED
  --add-opens=java.base/java.nio.charset=ALL-UNNAMED
  --add-opens=java.base/java.net=ALL-UNNAMED
  --add-opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED
  -Xmx16g
  -Dfile.encoding=UTF-8
  -Duser.country=UA
  -Duser.language=en
  -Duser.variant
  -Dfile.encoding=UTF-8
  -Dorg.gradle.profiler.scenario=assembleDev21FullGoogleDebug
Gradle args:
```

Рисунок 2.4 – Приклад результатів роботи gradle-profiler

Ця ретельна методологія тестування є ключовим компонентом у поточному вдосконаленні системи віддаленої збірки для додатків Android. У наступних розділах буде розглянуто аналіз результатів тестування, їх наслідки та ітераційний цикл розробки.

2.4 Висновок по розділу

У другому розділі пояснювальної записки було здійснено аналіз, присвячений вибору системи для розробки проекту. Основна увага була приділена вибору програмного та апаратного забезпечення.

В ході аналізу та вибору технологічного стеку для розробки системи дистанційного складання Android-додатків, була прийнята низка стратегічних рішень, що відображають оптимальність та ефективність вибраних інструментів.

Вирішено використовувати Kotlin як основну мову програмування для нашої системи. Цей вибір обґрунтований не тільки тим, що Kotlin є офіційною мовою розробки для Android, але і багатьма його перевагами. Kotlin надає сучасний та виразний синтаксис, забезпечує безпеку типів та підтримує асинхронне програмування з використанням корутин. Це робить його ідеальним вибором для розробки надійних та ефективних компонентів нашої системи.

Як інтегроване середовище розробки (IDE) для створення Android-додатків, обрано Android Studio. Ця платформа надає широкий набір інструментів та ресурсів для зручності розробників, включаючи візуальний редактор макетів, емулятори пристроїв та потужні засоби налагодження. Android Studio інтегрована з Kotlin, що додатково полегшує процес розробки, забезпечуючи комфортну взаємодію з обраною мовою програмування.

Як операційна система для окремої машини, призначеної для збирання Android-додатків, обрано Ubuntu. Цей дистрибутив Linux має низку переваг, які роблять його ідеальним вибором для цього завдання. Ubuntu є безкоштовним, має високу стабільність і безпеку, а також надає великий репозиторій інструментів розробника. У поєднанні з відкритим вихідним кодом і легким доступом до багатьох пакетів, Ubuntu створює оптимальне середовище для складання та розробки Android-додатків.

Таким чином, вибір Kotlin, Android Studio та Ubuntu у нашому технологічному стеку обумовлений прагненням до максимальної продуктивності, надійності та зручності розробки. Ці рішення дозволять нам ефективно реалізувати поставлені завдання, забезпечуючи високий рівень якості та функціональності нашої системи дистанційного збирання Android-додатків.

А також було обрано апаратне забезпечення, що буде оптимальним для окремої машини, що використовується для дистанційного збирання Android-додатків.

В цьому розділі також розглядався процес розробки системи віддаленої збірки для додатків Android, наступні розділи розкриють тонкощі роботи проекту, архітектурні міркування та оцінку продуктивності. Цей проект є значним кроком до переосмислення парадигм розробки Android, і його розвиток буде детально задокументовано в наступному розділі.

Також розглядалась інтеграція утиліти `gradle-profiler` у фазу тестування системи, що знаменує важливий прогрес у наших пошуках оцінки ефективності нової системи для віддаленого збирання програм Android, підкреслюючи його ключову роль у наданні всебічного розуміння продуктивності та ефективності системи. У наступному розділі будуть зроблені висновки із застосування `gradle-profiler`.

Включення `gradle-profiler` постає як стратегічне рішення з глибоким значенням. Він служить надійним інструментом, що дозволяє систематично та кількісно оцінювати можливості системи віддаленого збирання. Здатність утиліти виконувати команди ітеративно та вимірювати час їх виконання сприяє процесу оцінювання, керованого даними.

Висновки, зроблені на етапі тестування `gradle-profiler`, підкреслюють ітераційний характер циклу розробки. Отримані висновки сприяють подальшим вдосконаленням і оптимізаціям, сприяючи постійному вдосконаленню. Кожна ітерація базується на отриманих уроках, у результаті чого створюється динамічна та адаптивна система віддаленого збирання.

Успішна інтеграція `gradle-profiler` відкриває можливості для майбутніх удосконалень і розширень. У міру розвитку системи віддаленого збирання цю утиліту можна використовувати для оцінки впливу архітектурних змін, впровадження нових технологій і забезпечення відповідності системи мінливим галузевим стандартам і найкращим практикам.

Підсумовуючи, включення `gradle-profiler` у процес тестування й оцінки системи виявилось стратегічним і глибоким рішенням. Здатність утиліти надавати кількісні показники продуктивності, виявляти вузькі місця та спрямовувати зусилля з оптимізації значно сприяє поточному розвитку системи віддаленої

збірки для додатків Android. Висновки, зроблені в результаті тестування gradle-profiler, формують фундаментальну базу знань, яка формує траєкторію майбутніх розробок і вдосконалень.

3 ДОСЛІДЖЕННЯ СИСТЕМИ

У швидкозмінній сфері розробки мобільних додатків ефективність і продуктивність розробників відіграють ключову роль. Традиційний підхід створення додатків Android локально на окремих машинах пов'язаний із набором проблем, зокрема щодо використання ресурсів, масштабованості та спільної розробки. Усвідомлюючи ці проблеми, у центрі уваги цього дипломного проекту – розробка комплексної системи для віддаленого збирання додатків Android.

Розробка додатків для Android демонструє експоненціальне зростання як кількості програм, так і складності їх функціональних можливостей. Зі збільшенням масштабу проектів розробники стикаються з важким завданням керування залежностями, оптимізації ресурсів і забезпечення безперебійної співпраці всередині команд розробників. Локальні системи побудови, хоч і ефективні для невеликих проектів, часто виявляються обмеженнями для більш масштабних програм і розподілених команд.

У цьому контексті стає очевидною потреба в надійній та ефективній системі дистанційного складання. Система, яка дозволяє розробникам переносити процеси компіляції та збирання на централізований сервер, пропонує безліч переваг, починаючи від покращеного використання ресурсів і закінчуючи спрощеною співпрацею.

Необхідність систем віддаленого збирання:

- оптимізація ресурсів. Локальні процеси збирання можуть навантажувати окремі машини, що призводить до сповільнення циклів розробки

та зниження продуктивності. Система віддаленої збірки полегшує цей тягар, використовуючи обчислювальну потужність виділених серверів, забезпечуючи швидшу та ефективнішу збірку;

- масштабованість. Масштабні проекти Android вимагають масштабованої інфраструктури. Системи віддаленого складання забезпечують гнучкість масштабування ресурсів за потреби, пристосовуючись до зростаючої складності програм без шкоди для продуктивності;

- командна розробка. В епоху розподілених команд розробників співпраця має першорядне значення. Система віддаленої збірки служить централізованим центром, що дозволяє розробникам з різних місць безперешкодно брати участь у проекті без обмежень локальних залежностей збірки;

- зменшення вузьких місць розробки. Традиційні локальні збірки часто призводять до вузьких місць, особливо коли кілька розробників працюють одночасно. Система віддаленого збирання пом'якшує ці вузькі місця, розподіляючи процес збирання, дозволяючи паралельну розробку та швидші цикли ітерації.

Цей дипломний проект спрямований на дослідження, аналіз та реалізацію системи віддаленої збірки, призначеної для розробки додатків Android. Завдяки всебічному дослідженню існуючих рішень буде розроблено нову систему з урахуванням різноманітних потреб розробників, які працюють над проектами різного розміру та складності.

У попередніх розділах було детально проаналізовано існуючі системи віддаленого та локального збирання, виявлені їхні сильні та слабкі сторони. Останнім етапом проекту стане розробка налаштованої системи, призначеної для бездоганної інтеграції в сучасні робочі процеси розробки Android, забезпечуючи зміну парадигми в бік більш ефективних практик співпраці.

Підсумовуючи, розробка системи віддаленого збирання для додатків Android – це не просто технологічний прогрес; це являє собою фундаментальний зсув у бік оптимізації робочих процесів розробників, покращення співпраці та задоволення зростаючих вимог до великомасштабної розробки додатків для Android.

3.1 Основні характеристики системи

В цій роботі було розроблено інтегровану систему, здатну збирати Android-додатки віддалено, що дозволить істотно поліпшити робочий процес і знизити завантаження на локальні машини розробників.

Основні характеристики системи:

- дистанційна збірка. Розробка системи буде орієнтована на можливість видаленої компіляції та збірки Android-додатків;
- підтримка різних архітектур. Система буде спроектована для роботи з різними архітектурами Android-додатків, включаючи як монолітні, так і модульні підходи;
- інтеграція з наявними інструментами. Передбачено інтеграцію з популярними інструментами розробки Android, такими як Android Studio, Gradle та іншими;
- гнучкість і адаптивність. Розроблювана система буде володіти гнучкими налаштуваннями, що дозволяють адаптувати процес дистанційної збірки під вимоги конкретного проекту і переваги команди розробників;
- відповідність і моніторинг. Важливою частиною система буде функціонувати по відстеженню статусу збірки, наданню звітів про можливі помилки, а також моніторингу продуктивності.

Проект спрямований на створення інноваційної системи дистанційної збірки Android-додатків, що охоплює різні архітектурні підходи. Аналіз і тестування системи роботи на різних типах архітектури дозволить переконатися в її універсальності в різних проектах, забезпечуючи команді розробників високий рівень гнучкості, ефективності і зручності в процесі створення Android-додатків.

3.2 Порівняння використання системи для модульної та монолітної архітектури

За допомогою розробленої системи можна збирати проекти різної архітектури. Як монолітні, так і модульні.

Розглянемо особливості кожної з архітектур.

3.2.1 Монолітна архітектура

Монолітна архітектура є наріжним каменем у царині розробки додатків для Android, представляючи традиційний, але поширений підхід. У цьому розділі детально розглянемо тонкощі монолітної архітектури для програм Android, заглиблюючись у її фундаментальні принципи, організаційні стратегії, а також переваги та виклики, які вона представляє.

Основи монолітної архітектури:

- уніфікований модуль. В основі монолітної архітектури лежить концепція єдиного цілісного модуля, що містить усі функціональні можливості програми. Це інкапсулює кодову базу, ресурси, бібліотеки та залежності, об'єднуючи їх в єдину сутність;

- простота та узгодженість. Простота є керівним принципом монолітної архітектури. Розробники працюють в рамках єдиної структури проекту, сприяючи чіткому та прямому розумінню компонентів програми та їх взаємодії.

Структурні компоненти:

- заходи та фрагменти. Монолітні додатки для Android організовують інтерфейс користувача через дії та фрагменти в одному модулі. Діяльність представляє окремі екрани, тоді як фрагменти інкапсулюють повторно використовувані компоненти інтерфейсу користувача;

- послуги та трансляційні приймачі. Керування фоновими процесами та обробкою подій здійснюється в рамках моноліту за допомогою служб і приймачів трансляції. Цей централізований підхід спрощує оркестровку фонових завдань;

- управління ресурсами. Ресурси, включаючи макети, зображення та рядки, містяться в монолітній структурі. Ця консолідація полегшує спрощене керування ресурсами та полегшує посилання в кодовій базі.

Організація та практика розвитку:

- стандарти кодування. Монолітні архітектури часто стимулюють встановлення стандартів кодування. Уніфікована кодова база вимагає узгоджених практик, сприяючи читабельності, зручності обслуговування та спільному розвитку.

- контроль версій. Системи контролю версій відіграють вирішальну роль в управлінні монолітними проектами. Розробники застосовують такі практики, як розгалуження функцій і регулярна інтеграція, щоб підтримувати стабільну кодову базу, що розвивається.

Переваги монолітної архітектури:

- простота та легкість розробки. Консолідована природа монолітних архітектур спрощує розробку, полегшуючи розробникам розуміння, навігацію та зміну кодової бази;
- єдиний підрозділ розгортання. Монолітні програми розгортаються як єдине ціле, що спрощує процес випуску. Це спрощує версії, розгортання та процедури відкату;
- спільне використання ресурсів. Ресурси, бібліотеки та залежності безперешкодно розподіляються в межах моноліту, зменшуючи надмірність і забезпечуючи послідовне використання в усій програмі.

Виклики та міркування:

- проблеми масштабованості. Хоча монолітні архітектури вигідні для невеликих проектів, вони можуть зіткнутися з проблемами масштабованості. У міру зростання проектів моноліт може ставати складнішим і складнішим для масштабування;
- керування залежностями. Управління залежностями в межах моноліту вимагає ретельного розгляду. Зміни в одній частині кодової бази можуть вплинути на інші, що потребує надійного підходу, щоб мінімізувати непередбачені наслідки;
- спільна розробка. Великим командам розробників спільна розробка в монолітній архітектурі може бути складною. Координація стає вирішальною для запобігання конфліктам і забезпечення ефективної співпраці.

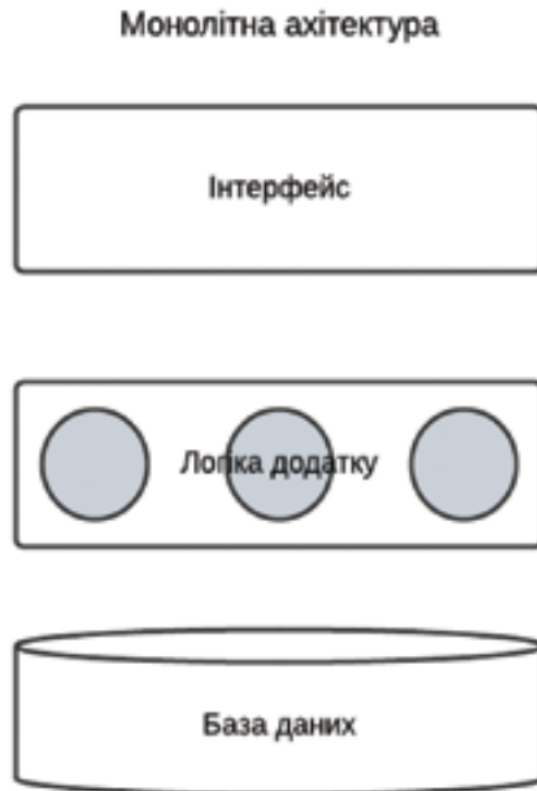


Рисунок 3.1 – Монолітна архітектура Android-додатку

Підсумовуючи, монолітна архітектура додатків Android втілює класичний, але незмінний підхід. Його простота, уніфікована структура та легкість розробки роблять його привабливим вибором для різноманітних проєктів. Однак у міру розвитку та масштабування додатків стають очевидними проблеми, пов'язані з масштабованістю, керуванням залежностями та спільною розробкою. Цей розділ закладає основу для розуміння принципів, компонентів і динаміки монолітних архітектур Android, готуючи основу для комплексного аналізу та порівняння з альтернативними архітектурними парадигмами.

3.2.2 Модульна архітектура

Модульна архітектура представляє сучасну та гнучку парадигму розробки додатків для Android. У цьому розділі детально розглядається модульна

архітектура, з'ясовуються її основні принципи, структурні компоненти, переваги та стратегії впровадження в екосистемі Android.

Основоположні принципи:

- модульність компонентів. В основі модульної архітектури лежить концепція поділу програми на окремі та незалежні модулі. Кожен модуль інкапсулює певний набір функціональних можливостей або особливостей, сприяючи модульному та відокремленому дизайну;

- незалежність і повторне використання. Модулі працюють незалежно, сприяючи повторному використанню коду та ресурсів у різних частинах програми. Ця незалежність полегшує тестування, обслуговування та модифікації без впливу на всю кодову базу.

Структурні компоненти:

- функціональні модулі. Програма поділена на модулі функцій, кожен з яких відповідає за певну функцію чи функціональність. Модулі функцій інкапсулюють усе, що стосується призначеної їм функції, включаючи компоненти інтерфейсу користувача, логіку та ресурси;

- бібліотечні модулі. Модулі бібліотеки містять спільний код, ресурси або утиліти, які використовуються для багатьох функцій. Ці модулі сприяють повторному використанню коду, забезпечуючи послідовність і ефективність керування ресурсами та кодом;

- модуль програми. Модуль програми служить точкою входу та оркестровцем, об'єднуючи різні модулі функцій і бібліотеки. Він забезпечує загальну структуру, навігацію та координацію між різними частинами програми.

Практика розробки:

- відокремлення концернів. Модульна архітектура підкреслює розділення проблем, дозволяючи розробникам зосередитися на конкретних функціях, не обтяжуючись складнощами всієї програми;

- ін'єкція залежності. Для керування залежностями між модулями часто використовуються інфраструктури впровадження залежностей. Це гарантує, що модулі можуть взаємодіяти без створення жорсткого зв'язку, підвищуючи ремонтпридатність і гнучкість.

Переваги модульної архітектури:

- масштабованість. Модульні архітектури відрізняються масштабованістю. У міру зростання програми можна додавати нові функції або змінювати наявні, не впливаючи на всю кодову базу. Ця масштабованість особливо вигідна для великих і складних програм;

- повторне використання коду. Модульний підхід сприяє багаторазовому використанню коду за допомогою створення модулів і бібліотек для окремих функцій. Це не тільки зменшує надлишковість, але й забезпечує послідовність реалізації коду;

- паралельний розвиток. Кілька команд розробників або окремих людей можуть працювати над різними модулями одночасно, сприяючи паралельній розробці. Це прискорює життєвий цикл розробки та забезпечує більш ефективну співпрацю.

Виклики та міркування:

- зв'язок модуля. Ефективний зв'язок між модулями має вирішальне значення. Розробники повинні ретельно керувати залежностями та взаємодіями, щоб уникнути небажаних наслідків і забезпечити взаємодію користувача;

– початкова складність налаштування. Створення модульної архітектури може спричинити додаткову складність на початкових етапах. Однак довгострокові переваги з точки зору зручності обслуговування та масштабованості часто переважають ці початкові інвестиції.

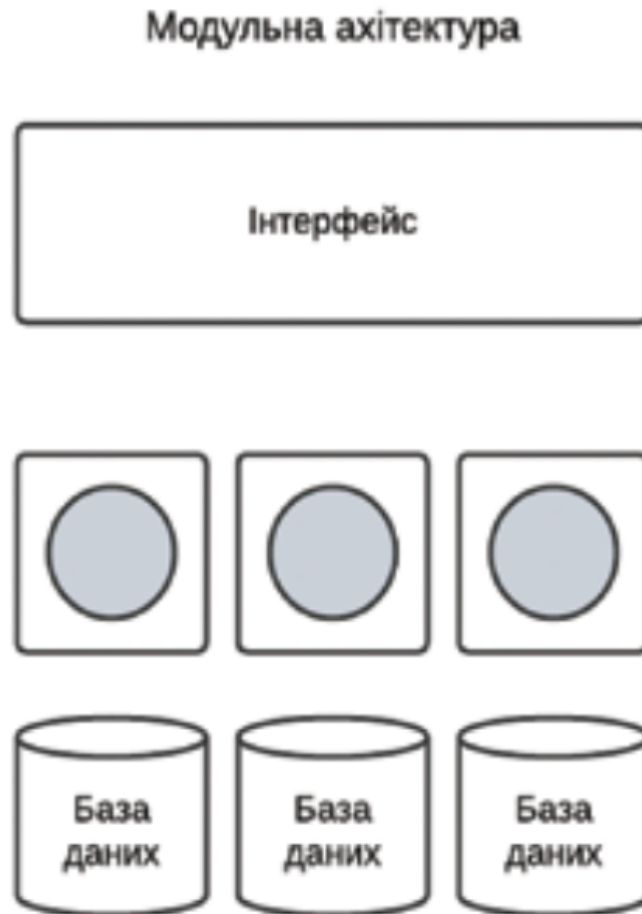


Рисунок 3.2 – Модульна архітектура Android-додатку

Підсумовуючи, модульна архітектура додатків Android втілює сучасний та адаптований підхід до розробки програмного забезпечення. Його наголос на незалежності, багаторазовому використанні та масштабованості узгоджується з еволюцією розвитку Android. Хоча проблеми існують, переваги модульної

архітектури позиціонують її як переконливий вибір для програм різного розміру та складності.

Розгляд стратегії вибору в розробці мобільних додатків є активною та широко дослідженою темою у сучасній галузі. Багато вчених проаналізували переваги, недоліки та компроміси різних архітектурних підходів.

Враховуючи вищезазначене, стає зрозумілим значущість правильного вибору архітектурного підходу, який враховує розмір, складність і вимоги до масштабованості конкретного проекту. Минулі дослідження висвітлюють переваги модульних додатків у зв'язку зі зручністю обслуговування, можливістю багаторазового використання, або визнає простоту та легкість монолітних додатків. Таким чином, розробникам слід уважно взважувати вибір стратегії розробки програмного забезпечення.

3.3 Дослідження часу збирання додатку

Сфера розробки додатків для Android постійно розвивається, керуючись безліччю архітектурних рішень, які мають зробити розробники. Серед вирішальних факторів, що впливають на успіх проекту, час створення та час збирання є ключовими показниками ефективності. Ефективність процесу збирання безпосередньо впливає на робочий процес розробки, швидкість ітерації та загальну продуктивність. В цьому розділі буде проведено дослідження варіацій часу збірки додатку на Android, реалізованого в трьох різних варіантах: модульній архітектурі з великою кількістю модулів, модульній архітектурі зі зменшеною кількістю модулів і монолітній архітектурі.

Для дослідження системи було взято вихідний код одного реального багатомодульного проекту. На його основі було сформовано три варіанти:

- багато модулів;
- менше модулів;

— лише один модуль.

Мета цього експерименту полягає в тому, щоб дослідити, як кількість модулів впливає на час збирання проєкту.

Порівнявши продуктивність і характеристики кожної конфігурації буде визначено оптимальний баланс між модульністю та монолітністю додатку.

Базовий проєкт має 59 модулів, які складаються з 864 файлів Kotlin і 1 файлу Java. Крім того, він включає 65 файлів активів, 1 файл прототипу та 161 файл ресурсів.

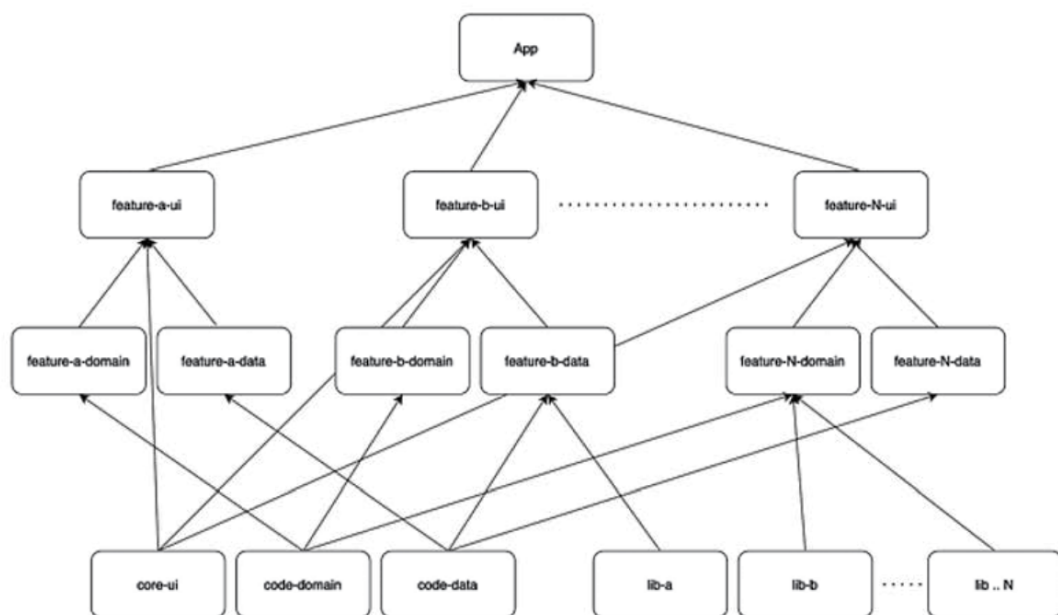


Рисунок 3.3 – Проєкт із 59 модулів

Для кожного з трьох варіантів буде виконано по 5 запусків в різних режимах і проаналізовано час збирання.

Перший етап - тестування базового проєкту. Далі була зроблена його копія та об'єднані під модулі функцій в один модуль. Наприклад, функція А має модулі даних, домену та інтерфейсу користувача. Було зроблено переміщення даних та класів домену в модуль інтерфейсу користувача, що дозволило отримати один

модуль замість 3. Користуючись таким алгоритмом вдалось зменшити кількість модулів до 33.

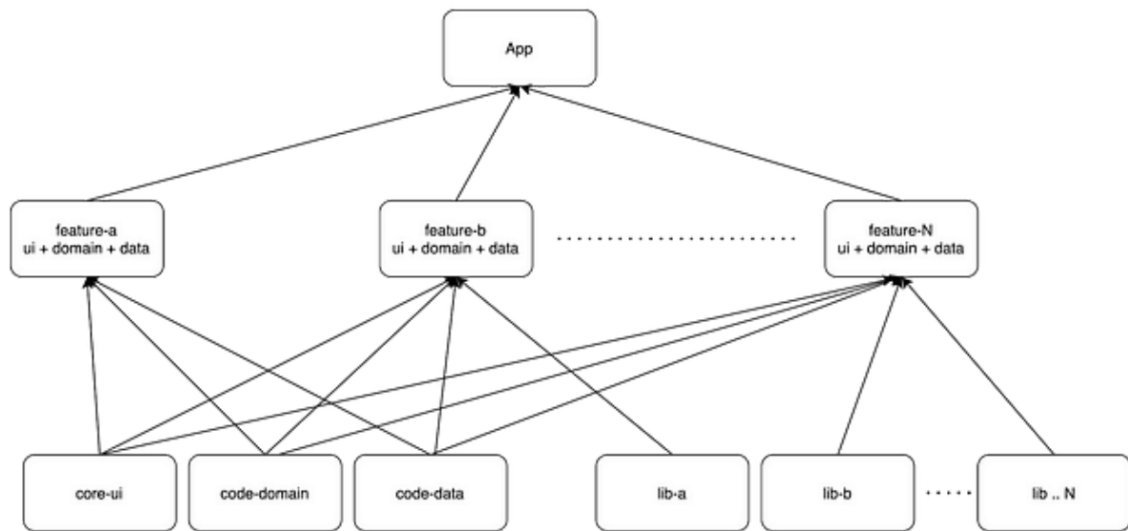


Рисунок 3.4 – Проєкт із 33 модулів

Останній етап - об'єднуємо всі модулі в один і отримуємо проєкт з монолітною структурою.

Операції, які будуть вимірюватись при збиранні проєкту:

- `./gradlew clean` – важливе завдання при перемиканні гілки проєкту;
- `./gradlew assembleDevDebug` – операція виконується кожного разу, коли натискається «Створити проєкт» або «Запустити програму» в Android Studio;
- `./gradlew assembleDevDebug –offline` – використання офлайн-режиму;
- `./gradlew assembleDevRelease` – перевірка якості збірки;
- `./gradlew assembleDevDebug` – але без очищення проєкту, щоб перевірити, як кількість модулів впливає на інкрементні збірки.

Команда `gradlew assembleDevDebug` зазвичай використовується в розробці Android із системою збірки Gradle. Ця команда ініціює процес створення програми Android для варіанту DevDebug.

Сценарій `gradlew` є частиною Gradle Wrapper, інструменту, який дозволяє розробникам запускати збірки Gradle без необхідності вручну встановлювати

Gradle на своїх машинах. Це гарантує, що збірка використовує певну версію Gradle, указану в проекті.

У контексті розробки Android за допомогою Gradle, завдання збирання є частиною стандартного набору завдань, які надає Android Gradle Plugin. Він відповідає за збирання ресурсів і коду у вказаний вихідний формат (APK у випадку проектів Android).

DevDebug представляє окремий варіант збірки в проектах Android. Варіанти складання дозволяють розробникам створювати різні версії своєї програми з різними конфігураціями або функціями. У цьому випадку DevDebug, ймовірно, є комбінацією стилю «Dev» і типу збірки «Debug».

Коли запускається `gradlew assembleDevDebug` з командного рядка, Gradle виконає необхідні завдання для створення програми Android для вказаного варіанту. Це включає такі завдання, як компіляція коду, обробка ресурсів і пакування програми в APK.

Ця команда часто використовується в процесі розробки, коли розробники хочуть створити певний варіант для тестування або налагодження. Отриманий APK можна розгорнути на пристрої Android або емуляторі для подальшого тестування. Фактична структура та конфігурація можуть відрізнятися залежно від специфіки проекту Android, як-от варіантів, типів збірок і використовуваних плагінів.

Команда `gradlew assembleDevRelease` — це спеціальна команда збірки, яка використовується в розробці Android із системою збірки Gradle. Ця команда ініціює процес створення програми Android для варіанту DevRelease.

DevRelease представляє окремий варіант збірки в проектах Android. Варіанти складання дозволяють розробникам створювати різні версії своєї програми з різними конфігураціями або функціями. У цьому випадку DevRelease, ймовірно, є комбінацією стилю «Dev» і типу збірки «Release».

Коли ви запускається `gradlew assembleDevRelease` з командного рядка, Gradle виконає необхідні завдання для створення програми Android для вказаного

варіанту. Це включає такі завдання, як компіляція коду, обробка ресурсів і пакування програми в APK.

Команда `gradlew assembleDevDebug` без очищення зазвичай використовується, коли розробники хочуть створити конкретну випускную версію програми, налаштовану для цілей розробки. Отриманий APK можна підписати ключем налагодження, що робить його придатним для внутрішнього тестування, але не для розповсюдження в магазинах програм.

Пропуск чистого завдання в команді (`gradlew assembleDevDebug`) означає, що процес збирання є поступовим. Gradle намагатиметься створити лише компоненти, які змінилися з часу останньої збірки, використовуючи кешовані результати з попередніх збірок, щоб пришвидшити процес. Це на відміну від виконання очищення перед збиранням, яке видалило б усі раніше створені артефакти та розпочало процес збирання з нуля.

Ця команда часто використовується під час ітераційної розробки, коли розробники хочуть швидко перебудувати та протестувати свою програму, не відкидаючи всі попередньо скомпільовані файли. Це забезпечує швидший час виконання збірки на етапах розробки та налагодження.

Варто зазначити, що в певних ситуаціях може знадобитися чиста збірка, особливо коли виникають проблеми, пов'язані із застарілими або пошкодженими артефактами збірки. Однак під час регулярної розробки використання `assembleDevDebug` без очищення є звичайною практикою для прискорення процесу збирання.

Таблиця 3.1 - Результати тестування операцій

Показники	Операція <code>assembleDevDebug</code>			Операція <code>assembleDevRelease</code>			Операція <code>assembleDevDebug</code> без очищення		
	1 модуль	33-модуля	59-модулів	1 модуль	33-модуля	59-модулів	1 модуль	33-модуля	59-модулів
1 запуск (с)	80	98	102	185	261	331	2	7	13
2 запуск (с)	74	95	104	174	229	249	2	7	11

3 запуск (с)	74	94	102	194	238	260	2	7	10
4 запуск (с)	75	94	106	216	268	283	2	7	11
5 запуск (с)	74	94	104	225	336	368	2	7	10
Середнє значення	75,4	95	103,6	198,8	266,4	298,2	2	7	11

Продовження таблиці 3.1

Показники	Операція assembleDevDebug			Операція assembleDevRelease			Операція assembleDevDebug без очищення		
	1 модуль	33-модуля	59-модулів	1 модуль	33-модуля	59-модулів	1 модуль	33-модуля	59-модулів
Дельта (с)	0	19,6	28,2	0	67,6	99,4	0	5	9
Дельта (%)	0	26	37	0	34	50	0	250	450

З результатів (табл. 3.1) можна зробити висновки, що монолітна архітектура дає нам деякі переваги при збиранні додатку. Операція assembleDevDebug в монолітній архітектурі показує найкращий результат. Середній час на 5 запусках – 75,4 с. Другий варіант, з меншою кількістю модулів працював на 19,6 секунди довше, тобто на 26% довше. А базовий проєкт, що містить 59 модулів, працював на 37% довше. З цього робимо висновок, що монолітний проєкт показує найкращий час збирання.

На тривалість операції assembleDevRelease найбільше впливає кількість модулів. Чим більша кількість модулів, тим більший час збирання додатку.

Збірка для проєкту з 59 модулів займає на 50% більше часу, порівняно з проєктом, що містить 1 модуль. Операція assembleDevDebug, без очищення після кожного виконання, показує, що поетапне збирання значно довше в базовому проєкті.

Результати даного експерименту показали, що монолітна архітектура продемонструвала найменший час збирання, тоді як підхід із багатьма модулями призвів до довшої тривалості компіляції. Загалом експеримент дав цінну інформацію про компроміси між монолітною та модульною архітектурами,

підкреслюючи важливість вибору найбільш прийняттого підходу на основі вимог до проєкту та бажаних результатів.

У випадку експерименту з тестуванням різних підходів, проєкт з монолітною архітектурою демонстрував найкращі результати, з найменшим часом збирання порівняно з багатомодульним проєктом. Таким чином, дана робота надає цінну інформацію про компроміси між монолітною та модульною архітектурами та підкреслює важливість вибору оптимального підходу залежно від вимог до проєкту та бажаних результатів

3.4 Висновки по розділу

Інтеграція gradle-profiler у фазу тестування нашої системи для віддаленого створення програм Android дала цінну інформацію про продуктивність системи в різних архітектурних конфігураціях. У цьому розділі було представлено докладні висновки, зроблені на основі даних, згенерованих gradle-profiler, підкреслюючи ефективність різних структур проєкту та важливість оптимізації.

Застосування gradle-profiler допомогло розкрити нюанси продуктивності нашої системи віддаленого збирання. Ця утиліта надає кількісну лінзу, через яку ми можемо ретельно перевіряти час створення, оцінювати використання ресурсів і приймати обґрунтовані рішення щодо оптимізації системи.

Таблиця (табл. 3.1) в якій зазначено час збирання для проєктів із різною кількістю модулів, служить орієнтиром для архітектурних міркувань. Ці результати підкреслюють вплив структури проєкту на ефективність побудови та закладають основу для стратегічних рішень щодо монолітної та модульної архітектур.

Процес тестування висвітлив сценарії, коли монолітна конструкція продемонструвала високу ефективність з точки зору часу збирання. У деяких випадках консолідований характер монолітного проєкту призводив до більш

швидкої роботи порівняно з модульними аналогами. Це підкреслює важливість урахування факторів, специфічних для проекту, під час вибору оптимальної архітектури.

Протиставляючи переваги монолітної структури, дані також підкреслюють важливість оптимального розподілу модулів. Приклад, який демонструє проект з 59 модулями, час збирання якого займає на 50% більше часу, ніж проект з 1 модулем, служить яскравою ілюстрацією. Цей різкий контраст підкреслює необхідність стратегічного планування модулів для пом'якшення вузьких місць продуктивності.

Висновки підкреслюють, що вплив структури проекту на час створення не є універсальним, а залежить від контексту. Масштабованість і складність проекту відіграють ключову роль у визначенні оптимального архітектурного підходу. У той час як монолітна структура може бути більш оптимальною в певних сценаріях, модульна архітектура стає обов'язковою для великомасштабних, складних проектів.

Висновки, зроблені в результаті тестування `gradle-profiler`, скеровують нас у прийнятті стратегічних архітектурних рішень. Замість того, щоб приймати універсальний підхід, ми визнаємо, що оптимальна структура залежить від конкретних факторів проекту. Ця гнучкість дозволяє нам адаптувати архітектуру системи відповідно до унікальних вимог кожного проекту.

Враховуючи такі фактори, як розмір проекту, складність і вимоги до масштабованості, розробники можуть приймати обґрунтовані рішення щодо найбільш прийняттого архітектурного підходу. Ця вказівка покращує ефективність і придатність для обслуговування збірок програм Android.

Ці висновки спонукають до постійних ітерацій оптимізації, гарантуючи, що система розвиватиметься відповідно до змінних вимог проекту. Здатність адаптувати архітектуру та конфігурації системи на основі емпіричних даних залишається наріжним каменем постійного вдосконалення.

ВИСНОВКИ

У рамках дослідження було проведено детальний аналіз обраної сфери, де основною виявленою проблемою стала відсутність якісних та зручних систем дистанційного збирання Android-додатків. Як потенційне рішення цього питання та для підвищення користувацького досвіду було запропоновано розробити власну систему, враховуючи сучасні потреби розробників і недоліки аналогів таких систем.

Далі у роботі було здійснено глибоке дослідження основ систем збирання Android-додатків, розглянуті варіанти локальної та дистанційної збірки. Аналіз підтвердив, що дистанційні системи збирання додатків мають великий потенціал у вирішенні проблем розробки Android-додатків.

Для розробки системи обрано мову програмування Kotlin, яка є сучасною і у порівнянні з Java має багато переваг. Розробка проводилась в IDE Android Studio, яка не має аналогів і як найкраще підходить для розробки Android-додатків та роботи з Kotlin.

Було детально проаналізовано і обрано програмне та апаратне забезпечення для окремо машини для збирання додатків.

Акцентувалась увага на різні варіанти архітектури Android-додатків, детально розглядались переваги та недоліки кожного варіанту.

Завдяки дослідженню стає важливістю вибору архітектури додатку і слідкування за оптимальною кількістю модулів у наявному додатку з модульною архітектурою.

Розроблена система має ряд переваг, яких немає у аналогів, а саме:

- інтеграція графічного інтерфейсу користувача;
- можливість працювати з різними архітектурами;
- відкритість та безкоштовність;

Дозволило зрозуміти деякі переваги монолітної архітектури і важливість оптимізації кількості модулів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Android Studio Release Notes URL:
<https://developer.android.com/studio/releases/index.html>.
2. Adarsh F. Android Studio 4.0 / Fernando Adarsh. URL:
<https://androiddevelopers.googleblog.com/2020/05/android-studio-4.html>
3. Kotlin docs URL: <https://kotlinlang.org/docs>
4. Java Virtual Machine URL:
https://en.wikipedia.org/wiki/Java_virtual_machine
5. Bill Phillips, Brian Hardy. Android Programming: The Big Nerd Ranch Big. NerdRanchGuides, 2013.
6. Recommended app architecture URL:
<https://developer.android.com/jetpack/guide#recommended-app-arch>
7. К.В. Харченко Методи та засоби розробки програмних додатків для операційної системи андроїд.// Збірник наукових праць. Серія: Нові рішення в сучасних технологіях. – Х.: НТУ «ХПІ». – 2014. - № 17. – С. 68-72.
8. Mainframer - Tool for remote builds. Sync project to remote machine, execute command, sync back. URL: <https://github.com/Adambl4/mirakle>.
9. INVESTIGATING BUILD TIMES FOR APPLICATIONS BUILT ON MODERN DEVELOPMENT STRATEGIES / N. A. Kulykovska та ін. Scientific notes of Taurida National V.I. Vernadsky University. Series: Technical Sciences. 2023. № 4. С. 65–70. URL: <https://doi.org/10.32782/2663-5941/2023.4/11>.
10. Учасники проєктів Вікімедіа. Android studio – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Android_Studio.
11. ELARTU – Інституційний репозиторій ТНТУ імені Івана Пулюя: Домівка.URL:https://elartu.tntu.edu.ua/bitstream/123456789/19221/2/FOSSLviv_2016_Iakhnenko_O_V-Rozrobka_Android_dodatviv_114-119.pdf.

12. Moskala M., Wojda I. Android Development with Kotlin: Enhance your skills for Android development using Kotlin. Packt Publishing, 2017. 440 p.
13. Gradle | gradle vs maven comparison. Gradle. URL: <https://gradle.org/maven-vs-gradle/>.
14. Programming android with kotlin / M. Dunn et al. O'Reilly Media, Incorporated, 2021.
15. Leiva A. Kotlin for android developers: learn kotlin the easy way while developing an android app. CreateSpace Independent Publishing Platform, 2016. 240 p.
16. Android's kotlin-first approach | android developers. Android Developers. URL: <https://developer.android.com/kotlin/first>.

Додаток А

```

open class Mirakle : Plugin<Gradle> {
    override fun apply(gradle: Gradle) {
        gradle.rootProject { it.extensions.create("mirakle",
MirakleExtension::class.java) }

        if (gradle.startParameter.taskNames.isEmpty()) return
        if
(gradle.startParameter.projectProperties.containsKey(BUILD_ON_REMOTE))
return

        if
(gradle.startParameter.projectProperties.containsKey(FALLBACK)) return
        if
(gradle.startParameter.excludedTaskNames.remove("mirakle")) return
        if (gradle.startParameter.isDryRun) return
        val gradlewRoot =
findGradlewRoot(gradle.startParameter.currentDir)
        ?: throw MirakleException("gradlew executable file is
not found.")

        if (loadProperties(File(gradlewRoot,
"local.properties"))["mirakle.enabled"] == "false") return
        gradle.assertNonSupportedFeatures()
        val startTime = System.currentTimeMillis()
        val startParamsCopy = gradle.startParameter.copy()
        val breakMode = startParamsCopy.projectProperties.let {
            (it[BREAK_MODE]?.toBoolean() ?: false) ||
(it[BREAK_TASK]?.isNotBlank() ?: false)
        }
        gradle.startParameter.apply {
            currentDir = gradlewRoot
            projectDir = gradlewRoot
            if (breakMode) {
                // pass all the tasks alongside with "mirakle" to
let Gradle calculate taskGraph
                setTaskNames(listOf("mirakle") + taskNames)
            } else {
                setTaskNames(listOf("mirakle"))
            }
        }
    }
}

```

```

        setExcludedTaskNames(emptyList())
    }
    if (!breakMode) {
        //a way to make Gradle not evaluate build.gradle,
settings.gradle and dependency catalog files on local machine
        val settingsFile =
gradlewRoot.listFiles()?.firstOrNull {
it.name.startsWith("settings.gradle") }
        val settingsBackup = settingsFile?.let {
File(gradlewRoot, "backup_${it.name}") }
        val buildFile = gradlewRoot.listFiles()?.firstOrNull
{ it.name.startsWith("build.gradle") }
        val buildFileBackup = buildFile?.let {
File(gradlewRoot, "backup_${it.name}") }
        val versionCatalog = File(gradlewRoot,
"gradle").listFiles()?.filter {
            it.name.endsWith("versions.toml")
        }
        settingsFile?.renameTo(settingsBackup)
        buildFile?.renameTo(buildFileBackup)

        val emptySettings = settingsFile?.let {
            // it makes Gradle stop looking for a
settings.gradle file in parent dir
            File(it.parentFile, "settings.gradle").apply {
createNewFile() }
        }
        val versionCatalogBackup = versionCatalog?.map {
            val backup = File(it.parent,
"${it.name}_backup")
            it.renameTo(backup)
            backup
        }
        gradle.afterMirakleEvaluate {
            emptySettings?.delete()
            settingsFile?.let {
                settingsBackup?.renameTo(it)
            }
            buildFile?.let {

```

```

        buildFileBackup?.renameTo(it)
    }

versionCatalogBackup?.zip(versionCatalog)?.forEach { (backup, original)
->

        backup.renameTo(original)
    }
}

// mirakle.gradle is the only Gradle build file which is
evaluated on local machine.
if (File(gradlewRoot, "mirakle.gradle").exists()) {
    gradle.rootProject { project ->
        project.apply(mutableMapOf("from" to
"mirakle.gradle"))
    }
}

// disable build scan on local machine, but it will be
enabled on remote if flag is set
isBuildScan = false
}

gradle.rootProject { project ->
    project.afterEvaluate {
        val config = kotlin.run {
            val mirakleConfig =
project.extensions.getByType(MirakleExtension::class.java)
                getMainframerConfigOrNull(gradlewRoot,
mirakleConfig)?.also {
                    println("Mainframer config is applied,
Mirakle config is ignored.")
                } ?: mirakleConfig
            }
        if (config.host == null) throw
MirakleException("Mirakle host is not defined.")
        println("Here's Mirakle ${BuildConfig.VERSION}. All
tasks will be executed on ${config.host}.")
        val breakTaskFromCLI =
startParamsCopy.projectProperties[BREAK_TASK]

```

```

        val breakOnTasks =
breakTaskFromCLI?.takeIf(String::isNotBlank)?.let(::listOf) ?:
config.breakOnTasks

        val upload = project.task<Exec>("uploadToRemote") {
            setCommandLine("rsync")
            args(
                fixPathForWindows(gradlewRoot.toString()),
                "${config.host}:${config.remoteFolder}",
                "--rsh",
                "ssh ${config.sshArgs.joinToString(separator
= " ")})",

                "--exclude=mirakle.gradle"
            )
            args(config.buildRsyncToRemoteArgs())
        }
        val execute =
project.task<ExecuteOnRemoteTask>("executeOnRemote") {
            this.config = config
            this.gradlewRoot = gradlewRoot
            this.startParams = startParamsCopy
            if (!breakMode || breakOnTasks.isEmpty()) {
                setTaskList(startParamsCopy.taskNames)
            } // else in break mode tasks arguments will be
set later when Gradle::taskGraph is ready
            isIgnoreExitValue = true
            // disable remote stream modification since it
breaks Idea and Android Studio test result processing
            // https://github.com/Adamb14/mirakle/issues/83
            if (!gradle.containsIjTestInit()) {
                standardOutput = modifyOutputStream(
                    standardOutput ?: System.out,

"${config.remoteFolder}/${gradlewRoot.name}",
                    gradlewRoot.path
                )
                errorOutput = modifyOutputStream(
                    errorOutput ?: System.err,

"${config.remoteFolder}/${gradlewRoot.name}",

```

```

        gradlewRoot.path
    )
}
}.mustRunAfter(upload) as ExecuteOnRemoteTask
val download =
project.task<Exec>("downloadFromRemote") {
    setCommandLine("rsync")
    args(

"${config.host}:${config.remoteFolder}/${gradlewRoot.name}/",
        "./",
        "--rsh",
        "ssh ${config.sshArgs.joinToString(separator
= " ")}",
        "--exclude=mirakle.gradle"
    )
    args(config.buildRsyncFromRemoteArgs())
}.mustRunAfter(execute) as Exec
if (config.downloadInParallel) {
    if (config.downloadInterval <= 0) throw
MirakleException("downloadInterval must be >0")
    val downloadInParallel =
project.task<Task>("downloadInParallel") {
        doFirst {
            val downloadExecAction =
services.get(ExecActionFactory::class.java).newExecAction().apply {
                commandLine = download.commandLine
                args = download.args
                standardOutput =
download.standardOutput ?: System.out
                standardInput =
download.standardInput ?: System.`in`
            }
            //It's impossible to pass this as
serializable params to worker

            //God forgive us

DownloadInParallelWorker.downloadExecAction = downloadExecAction

```

```

DownloadInParallelWorker.gradle = gradle
services.get(WorkerExecutor::class.java).submit(DownloadInParallelWorker:
:class.java) {

    it.isolationMode =

IsolationMode.NONE

it.setParams(config.downloadInterval)

    }

    }

    onlyIf {

        config.downloadInParallel &&
upload.executionResult.get().exitValue == 0 && !execute.didWork

    }

    }

    downloadInParallel.mustRunAfter(upload)
    download.mustRunAfter(downloadInParallel)

gradle.startParameter.setTaskNames(listOf("downloadInParallel",
"mirakle"))

    }

    val mirakle =
project.task("mirakle").dependsOn(upload, execute, download)
    try {
        mirakle.doNotTrackState("Mirakle is never
up-to-date")

    } catch (ignore: NoSuchMethodError) {
        // Gradle <7.3
    }

    execute.onlyIf {
upload.executionResult.orNull?.exitValue == 0 }
        download.onlyIf {
upload.executionResult.orNull?.exitValue == 0 }
        if (!config.fallback) {
            mirakle.doLast {

execute.executionResult.get().assertNormalExitValue()

        }

    } else {

```

```

        val fallback =
project.task<DefaultTask>("fallback") {
            onlyIf {
upload.executionResult.get().exitValue != 0 }
            doFirst {
                println("Upload to remote failed.
Continuing with fallback.")
                val connection =
GradleConnector.newConnector()

                .forProjectDirectory(gradle.rootProject.projectDir)
                    .connect()
                connection.use { connection ->
                    connection.newBuild()

                .withArguments(startParamsToArgs(startParamsCopy).plus("-P$FALLBACK=true"
))

                .setStandardInput(upload.standardInput ?: System.`in`)

                .setStandardOutput(upload.standardOutput ?: System.out)

                .setStandardError(upload.errorOutput ?: System.err)
                    .run()
            }
        }
        upload.isIgnoreExitValue = true
        upload.finalizedBy(fallback)
        mirakle.doLast {
            if (execute.didWork) {

execute.executionResult.get().assertNormalExitValue()
            }
        }
    }
    if (breakMode && breakOnTasks.isNotEmpty()) {
        if (config.downloadInParallel) {

```

```

        throw MirakleException("Mirakle break mode
doesn't work with download in parallel yet.")
    }
    gradle.taskGraph.whenReady { taskGraph ->
        val breakTaskPatterns =
breakOnTasks.filter(String::isNotBlank).map(Pattern::compile)

        val graphWithoutMirakle =
taskGraph.allTasks.filterNot(Task::isMirakleTask)
        val breakTasks = graphWithoutMirakle.filter
{ task ->
            breakTaskPatterns.any {
it.matcher(task.name).find() }
            }
        when {
            breakTasks.isEmpty() -> {

execute.setTaskList(startParamsCopy.taskNames.minus("mirakle"))
                graphWithoutMirakle.forEach {
                    it.enabled = false
                }
            }
            breakTasks.size == 1 -> {
                val breakTask = breakTasks.first()
                val tasksForRemoteExecution =
graphWithoutMirakle
                    .takeWhile { it != breakTask }
                    .onEach { it.enabled = false }
                execute.setTaskList(tasksForRemoteExecution.map(Task::getPath))
                breakTask.apply {
                    enabled = true
                    mustRunAfter(download)
                    onlyIf { execute.didWork &&
execute.executionResult.get().exitValue == 0 }
                }
                if
                (inputs.files.files.isNotEmpty()) {
                    val inputsNotInProjectDir =
inputs.files.files.filter {
!it.path.startsWith(gradlewRoot.path)

```

```

    }
    if
    (inputsNotInProjectDir.isEmpty()) {
        throw MirakleException(
            "${breakTask.toString().capitalize()} declares input not from project
            dir. That is not supported by Mirakle yet.
            Tasks:\n${inputsNotInProjectDir.joinToString { it.path }}"
        )
    }
    // Need to include all of
    the parent directories down to the desired directory
    val includeRules =
    inputs.files.files.map { file ->
        val pathToInclude =
        file.path.substringAfter(gradlewRoot.path).drop(1)
        val split =
        pathToInclude.split("/")
        (1 until
        split.size).mapIndexed { _, i ->
            split.take(i).joinToString(separator = "/")
        }.map {
            "--include=$it"
        }.let {
            val isFile =
            split.last().contains('.')
            if (isFile) {
                it.plus("--include=$pathToInclude")
            } else {
                it.plus("--include=$pathToInclude/**")
            }
        }
    }.flatten().toSet()

    //val excludeAllRule =
    "--exclude=/**" // TODO does it make sense to exclude all except the
    inputs?

```

```

download.setArgs(includeRules /*+ excludeAllRule */ +
download.args!!.toList())

        }
    }

    execute.doFirst {
        println("Mirakle will break
remote execution on $breakTask")
    }

    mirakle.dependsOn(breakTask)
    gradle.logTasks(graphWithoutMirakle
- tasksForRemoteExecution)
    }

    else -> {
        throw MirakleException("Task
execution graph contains more than 1 task to break on. That is not
supported by Mirakle yet. Tasks:\n${breakTasks.joinToString { it.path
}}")
    }
}
}
} else {
    gradle.startParameter.apply {
        if
(taskNames.contains("downloadInParallel")) {

setTaskNames(listOf("downloadInParallel", "mirakle"))
        } else {
            setTaskNames(listOf("mirakle"))
        }
        setExcludedTaskNames(emptyList())
    }
}

gradle.uploadInitScripts(upload, execute, download)
gradle.logTasks(upload, execute, download)

```

```
        gradle.logBuild(startTime, mirakle)
    }
}
}
```