Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем

Розробка Android застосунку з використанням Firebase

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник курсової роботи

Старший викладач Борозенний С.О.

_____

(підпис)

“____” _____ 2022 р

Роботу виконав студент БП ІПЗ-4

Левчук В. Ю.

“____” _____ 2022 р

Київ 2022

Тема: **Розробка Android застосунку з використанням Firebase**

Календарний план виконання роботи

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової	11.10.2021	
2.	Огляд документації Firebase	11.01.2022	
3.	Дослідження тонкощів побудови Android-додатків на Compose UI	12.02.2022	
4.	Підготовка базового каркасу Android-додатку	10.02.2022	
5.	Додавання функціоналу Firebase у додаток	17.04.2022	
6.	Завершення розробки та тестування додатку	24.04.2022	
7.	Написання першого та другого розділів текстової частини	01.05.2022	
8.	Написання решти роботи	08.05.2022	
9.	Остаточне редагування роботи	15.05.2022	
10.	Перевірка роботи на плагіат	20.05.2022	
11.	Перевірка змісту роботи науковим керівником	26.05.2022	
12.	Захист курсової роботи	27.05.2022	

Студент _____
Керівник _____

“_____”_____

Зміст

Вступ

Смартфони стали невід'ємною частиною життя сучасного суспільства, що створило високий запит на якісні мобільні системи та застосунки. Власники смартфонів є найбільшою аудиторією багатьох ресурсів та застосунків, тож мобільні додатки є затребуваними як ніколи. На сьогодні, найпопулярнішою мобільною операційною системою є Android, перевагами якої є відкритий сирцевий код, відносна зручність розробки та величезна кількість сумісних пристроїв. Це платформа, для якої зручно та варто розробляти, а останні оновлення, як, наприклад, бібліотека Compose UI, підносять Android-розробку до нових висот.

Проте, практично жоден Android-застосунок не обходиться у тій чи іншій формі без Firebase – платформи, яка надає сервіси бекенду та аналітики для мобільних та веб-застосунків. Сюди також входить підтримка сповіщень для мобільних додатків, авторизація, сховища та бази даних. Firebase дозволяє заощадити на бекенд розробці, що є безцінною перевагою для всіх, а особливо маленьких команд розробників.

Саме тому надзвичайно важливими у розробці Android-додатків є навички інтеграції з численними сервісами Firebase.

Метою виконання курсової роботи є дослідження обраних технологій, а також поглиблення знань та навичок мобільної розробки. Якщо говорити про Android, то це перш за все повне опанування бібліотеки Compose UI, яка набагато полегшує розробку та значно покращує стабільність роботи додатків. Щодо Firebase також варто зауважити, що більшість сервісів є кросплатформовими, тож навички користування ними є корисними і при розробці для інших платформ(наприклад iOS).

Об'єкт дослідження – ефективні інструменти для Android-розробки та сервіси платформи Firebase.

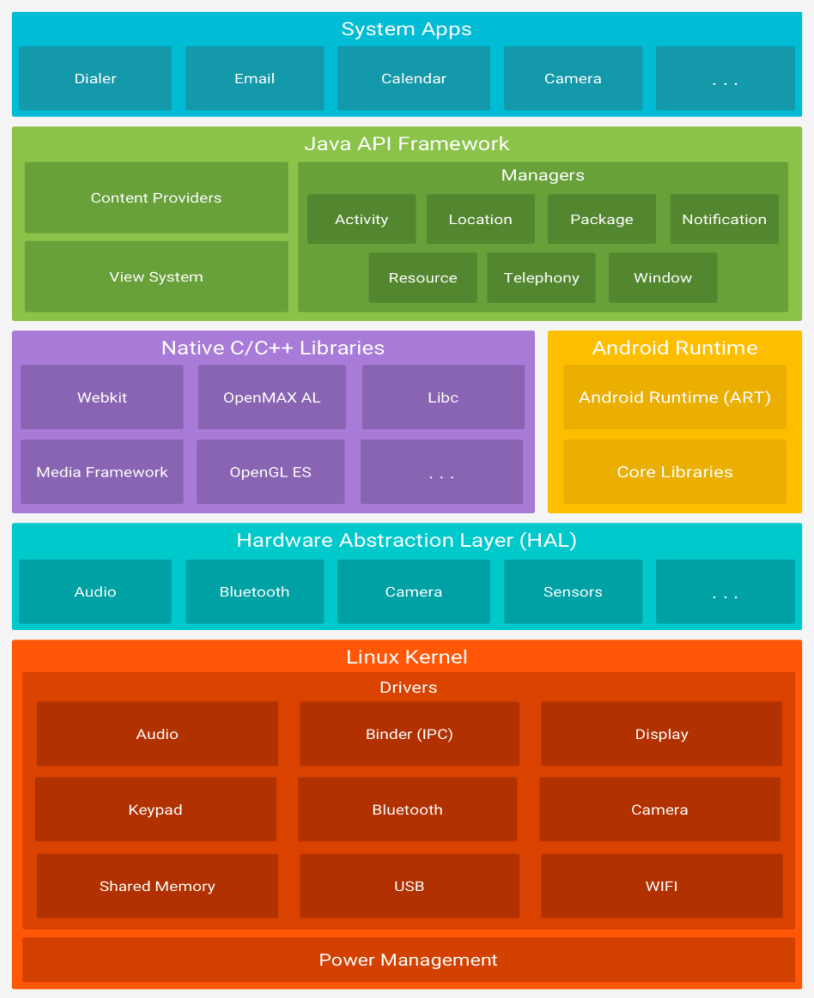
Предмет дослідження – Android, Compose UI framework, Firebase Auth, Cloud Firestore, Cloud Storage, Firebase Analytics.

Оскільки кінцевий продукт має бути гарною ілюстрацією можливостей сервісів Firebase, було прийняте рішення розробити StampNet – додаток та базу даних про поштові марки з усього світу, з реєстрацією користувачів та можливістю поповнення бази даних спільнотою. Для інтерфейсу StampNet було використано винятково можливості Compose UI, а всі завдання з авторизації, збереження та отримання інформації були покладені на сервіси Firebase. StampNet розроблявся перш за все для Android 12 та використовує останні здобутки платформи, тож ця робота стане цікавим прикладом для кожного мобільного розробника.

Розділ 1. Операційна система Android

1.1 Опис та історія системи

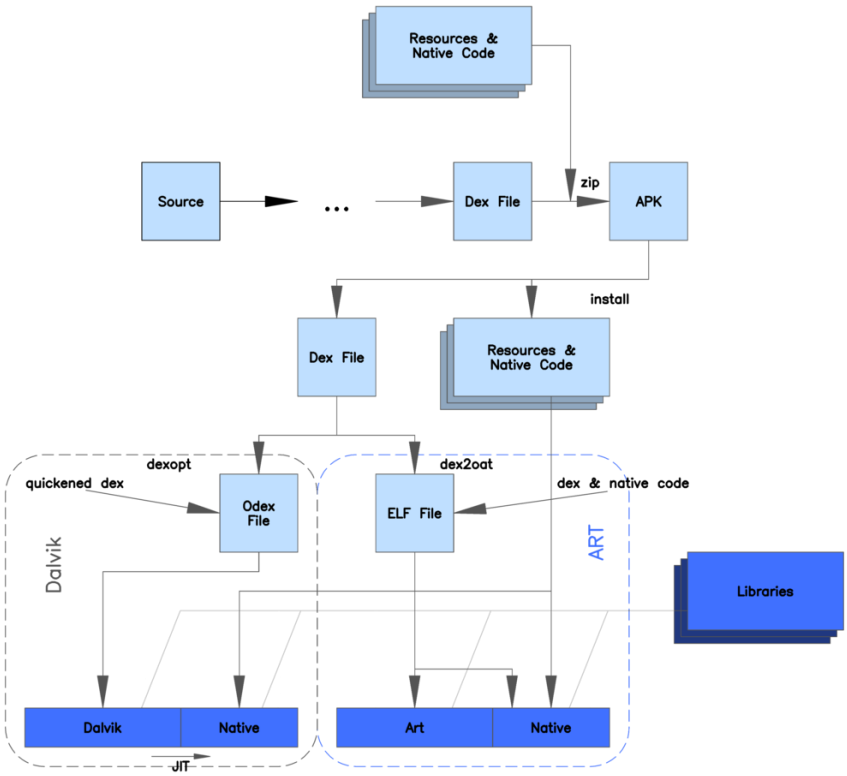
Android – це мобільна операційна система на базі модифікованого ядра Linux та іншого відкритого програмного забезпечення, призначена перш за все для мобільних пристроїв з сенсорними екранами, а саме смартфонів та планшетів. Програмна оболонка Android використовує ППЗ та бібліотеки написані мовою C, а також бібліотеки для застосунків, сумісні з Java.



Діаграма архітектури Android

Для роботи Android-додатків до версії Android 5 використовувалося середовище виконання Dalvik - регістр-орієнтована віртуальна машина, оптимізована для обмежених ресурсів, програми для якої писалися на Java. Це середовище виконання використовувало JIT(just-in-time) компіляцію, при

якій найчастіше використовувані сегменти байткоду програм динамічно компілювалися у машинний код.



Порівняння архітектур Dalvik та ART

Починаючи з версії Android 5, всі програми виконуються у середовищі ART (Android Runtime), яка використовує AOT(ahead-of-time) компіляцію: програми повністю компілюються у машинний код ще при встановленні. Це значно підвищило швидкодію та енергоефективність застосунків.

Робота над створенням системи велася компанією Android Inc. з 2003 року, і початковою метою було створити просунути операційну систему для цифрових фотоапаратів. Проте ще на етапі пошуку інвесторів стало зрозуміло, що цільовий ринок надто малий, тому було прийняте рішення створити мобільну операційну систему, яка б стала конкурентом тодішнім головним ОС для смартфонів – Symbian та Microsoft Windows Mobile.

Після придбання Android Inc. компанією Google у 2005 році, почалася активна розробка мобільної платформи на основі ядра Linux. Google просував нову платформу серед телефонних виробників та мобільних операторів як гнучку та зручну для оновлень систему.

Проте на той час Android все-одно абсолютно відрізнявся від того, яким його знають тепер. Ранній прототип цільового пристрою на Android нагадував тогочасні BlackBerry: QWERTY клавіатура та жодного сенсорного екрану. Але успішна поява iPhone змусила Google переглянути свої плани та пообіцяти підтримку сенсорних екранів у документації платформи. Та вже у 2007, через успіх iPhone 3G та перехід ключових виробників мобільних телефонів на смартфони з сенсорними екранами, Android став призначеним винятково для сенсорних екранів.

Перша версія Android була випущена 23 вересня 2008 року і мала назву 1.0 Astroboy, а першим комерційним носієм системи став представлений того ж року смартфон HTC Dream.

Всі наступні версії до Android 10 мали назви десертів, починаючи з 10 версії Android версії називаються літерами латинського алфавіту.

З 2008 Android отримав величезну кількість оновлень, які додали чимало функцій та покращень у систему, виправляли помилки, а також, інколи, забирали(і в наступних випусках додавали) доступ до деяких функцій та апаратних можливостей. Часті критичні зміни системи у нових версіях можуь створювати додаткові складнощі розробникам, що детальніше розглядається у наступному підрозділі.

З 2011 року Android є найпопулярнішою у світі ОС для смартфонів, а з 2013 – для планшетів. Станом на травень 2021 року, система налічувала понад 3 мільярди активних користувачів у місяць – найбільше число серед усіх операційних систем, а магазин додатків Google Play Store – більше 3 мільйонів додатків.

Відкритість системи, а також величезне число та різноманітність користувачів є, водночас, як благословенням, так і прокляттям системи, адже накладає на розробників набагато більшу відповідальність та потребує значних ресурсів для тестування. Всі вищезгадані чинники утворюють **особливості розробки застосунків для ОС Android.**

1.2 Особливості розробки застосунків для ОС Android

Нові версії Android поширюються у рамках Android Open Source Project, безкоштовного проєкту з відкритим сирцевим кодом. Через високу універсальність та різноманітний ряд підтримуваних пристроїв, до дистрибутивів Android часто входять не всі потрібні апаратні драйвери та застосунки. Саме тому, зазвичай використовуються модифіковані пропрієтарні версії Android, з індивідуальним набором драйверів та іншого програмного забезпечення.

Саме ця можливість модифікації початково коду під свої потреби призвела до того, що чимало дистрибутивів Android відчуваються як абсолютно різні системи як на користувацькому, так і на рівні розробки.

Нижче наведені приклади найпоширеніших версій та модифікацій Android:

Чистий Android

Android One

OxygenOS

Samsung OneUI

MIUI *EMUI* *Sony Xperia UI* *ZenUI*

Якщо чистий Android та Android One не вносять жодних змін до сирцевого коду та не ускладнюють життя розробникам, то інші версії, наприклад, MIUI, EMUI та інші ОС китайських виробників суттєвим чином змінюють роботу системи та вимагають додаткових зусиль для розробки.

Так, операційні системи телефонів китайських виробників Xiaomi, Huawei, Oppo та інших мають налаштування «Автозапуску», яке фактично блокує роботу фонових процесів додатків. Ба більше, жоден з виробників не надає потрібної документації для розробників, де було б вказана правильна обробка такої поведінки.

У таких випадках, розробники можуть лише показати вікно даного налаштування, попередньо пояснивши користувачу потребу у його вимкненні. Внизу показані рядкові константи, необхідні для запуску вікна з налаштуванням «Автозапуску»

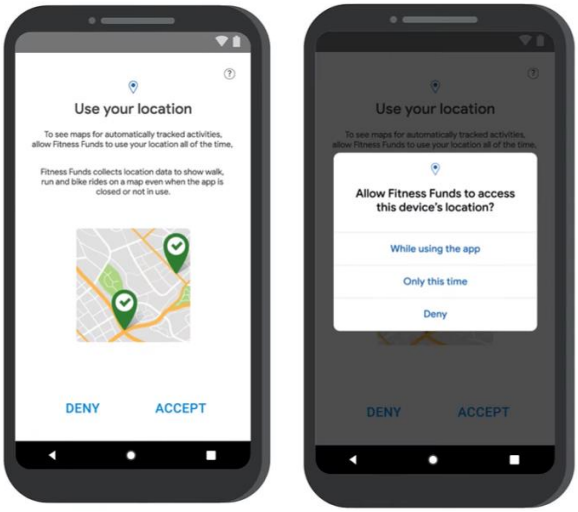
```
<!--Devices, that need "Autostart" option to be turned on-->
<string name="huawei_battery_opt_pkg" translatable="false">com.huawei.systemmanager</string>
<string name="huawei_battery_opt_cls" translatable="false">com.huawei.systemmanager.optimize.process.ProtectActivity</string>
<string name="xiaomi_battery_opt_pkg" translatable="false">com.miui.securitycenter</string>
<string name="xiaomi_battery_opt_cls" translatable="false">com.miui.permcenter.autostart.AutoStartManagementActivity</string>
<string name="redmi_battery_opt_pkg" translatable="false">@string/xiaomi_battery_opt_pkg</string>
<string name="redmi_battery_opt_cls" translatable="false">@string/xiaomi_battery_opt_cls</string>
<string name="oppo_battery_opt_pkg" translatable="false">com.coloros.safecenter</string>
<string name="oppo_battery_opt_cls" translatable="false">com.coloros.safecenter.permission.startup.StartupAppListActivity</string>
<string name="vivo_battery_opt_pkg" translatable="false">com.iqoo.secure</string>
<string name="vivo_battery_opt_cls" translatable="false">com.iqoo.secure.MainGuideActivity</string>
<string name="pkg_battery_opt_sufix" translatable="false">_battery_opt_pkg</string>
<string name="cls_battery_opt_sufix" translatable="false">_battery_opt_cls</string>
<string-array name="battery_optimization_devices">
    <item>vivo</item>
    <item>oppo</item>
    <item>xiaomi</item>
    <item>huawei</item>
    <item>redmi</item>
</string-array>
```

Ідентифікатори програмних пакетів, необхідних для запуску вікна з налаштуванням «Автозапуску»(окремі для кожної ОС)

Цікавим є те, що це налаштування є вибірковим, і для відомих додатків на кшталт Instagram, Facebook тощо, блокування за замовчуванням вимкнене.

Також чималий вплив на розробку має подальша публікація додатка у Google Play Store, яка вимагає дотримання усіх політик Google, які часто змінюються та відрізняються для різних версій Android. Найяскравішою ілюстрацією цьому є зміна політики дозволів на відстеження розташування через фонові процеси додатку у 2021 році.

Згідно з новими політиками, додатки, що потребували ці дозволи, тепер мусили заповнити відповідну заявку перед публікацією, а також додати у самі додатки діалоги-пояснення потреби у дозволах. Насамкінець, перед публікацією версію необхідно записати та відправити відео, де демонструвалося дотримання даних вимог у додатку.



Діалоги, необхідні для отримання дозволу на фонове відстеження розташування(може бути і більше)

Врешті-решт, більшість особливостей розробки під Android так чи інакше пов'язані з двома протилежними явищами: відкритістю й універсальністю платформи, та залежністю від сервісів компанії Google.

1.3 Головні нововведення Android 12

Як і кожна нова основна версія, Android 12 вносить чимало змін та нового функціоналу у розробки. Оскільки змін вкрай багато, є сенс розглянути найголовніші:

- **Splash screen API –** відтепер, екран заставки автоматично генерується додатком з використанням значка додатку. Є можливість конфігурувати це екран заставки, але окреме вікно вже не потрібно створювати. Якщо екран заставки до цього створювався вручну, його необхідно перенести на новий API, або ж додати анотацію @CustomSplashScreen.

- **Material You та Material Design 3 –** оновлення бібліотек Material Design 3, які дозволяли налаштувати інтерфейс додатку згідно з API та кращими практиками. Було додано нові можливості налаштувань графічних елементів, а також була додана інтеграція з **Material You** – ексклюзивна опція для користувачів Android 12, яка розширює горизонти для «кастомізації» системи. Найцікавіша особливість цього оновлення для розробників це **Dynamic Color –** функція, яка автоматично генерує кольорову палітру на основі шпалер головного екрану, та розповсюджує її на всі елементи системи та додатки.

- **Дозволи Bluetooth –** відтепер для використання додатком функцій Bluetooth потрібно отримати до 3 нових дозволів: BLUETOOTH_SCAN, BLUETOOTH_ADVERTISE та BLUETOOTH_CONNECT. Кожен відповідає певному типу функції Bluetooth:

BLUETOOTH_SCAN – додаток шукає приладу Bluetooth-периферії поблизу;

BLUETOOTH_ADVERTISE – додаток робить телефон видимим для інших Bluetooth-пристроїв;

BLUETOOTH_CONNECT – додаток обмінюється даними з вже приєднаними Bluetooth-пристроями.

- **AppSearch –** новий вбудований рушій для пошуку у пристрої. Дозволяє індексувати та відображати інформації з усіх частин системи та додатків, які підтримують дану функцію. Тобто, можна налаштувати, яка інформація буде індексована даним пошуковиком.

- **Rich content insertion –** можливість зручно копіювати та отримувати з інших джерел та додатків не лише текст, а й зображення та відео(як через копіюванням, такі через drag-and-drop).

Це далеко не повний список усіх змін, що приносить з собою версія Android 12, але вищезгадані нововведення вже здатні суттєво змінити розробку, як це загалом буває майже з кожною версією.

Таким чином, можна зробити висновок, що Android є надзвичайно динамічною та мінливою платформою. І хоча така особливість розробки часто може виснажувати або відлякувати, це також надихає – інколи.

Розділ 2. Платформа Firebase

2.1 Опис та історія розвитку платформи

Firebase – це платформа для розробки мобільних та веб- застосунків. Фактично, ця платформа є сукупністю серверних сервісів, таких як авторизація, автентифікація, збереження та обмін даними, аналітика тощо.

Firebase є розвитком Evolve – стартапу, заснованого Джеймсом Тампліном та Ендрю Лі у 2011 році. Envolve надавав розробникам API, яке дозволяло додати функціонал онлайн-чату до веб-сайту. Після запуску сервісу, засновники виявили, що сервіс використовувався зазвичай не для передачі повідомлень, а для обміну різноманітними даними застосунку. Наприклад, деякі розробники використовували Evolve для синхронізації ігрових даних між користувачами в режимі реального часу.

Тамплін та Лі зрозуміли справжній потенціал платформи, та запустили окремий сервіс з архітектурою реального часу, яка використовувалася до цього у чатах. Згодом, у тому ж 2011 році, вони заснували нову компанію – Firebase, яка стала публічною у 2012 році.

Першим продуктом компанії став Firebase Realtime Database – API, який синхронізує дані застосунку між iOS, Android і веб-приладами, та зберігає у хмарному сховищі Firebase .

Відтоді Firebase здобула все більше фінансування від інвесторів, і в 2014 році запустила два продукти: Firebase Hosting та Firebase Authentication. Поява цих проєктів остаточно затвердила компанію як постачальника моделі «Мобільний серверний сервіс як послуга»(англ. Mobile backend as a service, скорочено MBaaS).

У жовтні 2014-го, Firebase була поглинута компанією Google. Через рік, у жовтні 2015-го, Google придбала Divshot, платформу для веб-хостингу HTML5, та об’єднала її з командою Firebase.

У травні 2016-го року, Google представила Firebase Analytics та оголосила про наміри розширити спектр доступних сервісів платформи, аби перетворити

Firebase на уніфіковану платформу бекенд-як-послуга для мобільних розробників.

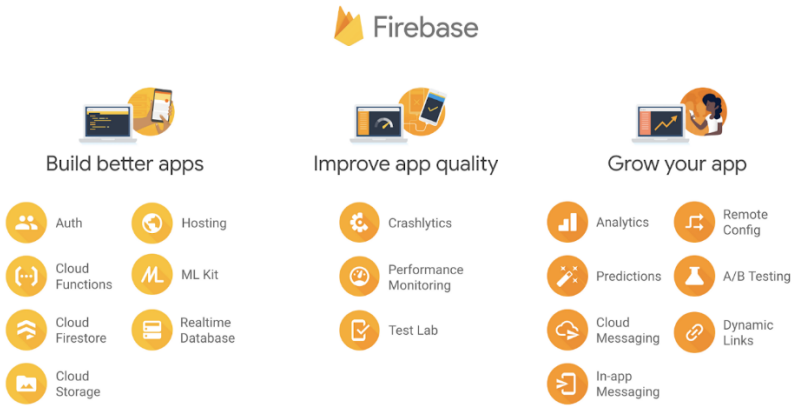
З того часу платформа інтегрувалася з багатьма сервісами Google: Google Cloud Platform, AdMob, Google Ads та інші. Google Clound Messaging, сервіс надсилання push-сповіщень на пристрої Android, був замінений на Firebase Cloud Messaging, який додав можливість надсилати сповіщення також на iOS та веб-пристрої.

У липні 2016-го, Google поглинула платформу мобільної розробки LaunchKit, яка спеціалізувалася на розробці маркетингу застосунків, та з'єднав її з командою Firebase Growth Tools. У січні 2017-го, компанія також купила служби Fabric та Crashlytics у Twitter, та приєднала їх до Firebase.

У жовтні цього ж року, Firebase запустила Cloud Firestore – базу даних реального часу, яка фактично стала наступницею першого продукту компанії, Firebase Realtime Database.

2.2 Основні сервіси та послуги Firebase

За роки свого розвитку, а особливо після поглинання компанією Google, платформа Firebase обросла величезною кількістю послуг та сервісів найрізноманітнішого напрямку. Такий швидкий ріст став можливим не в останню чергу завдяки тому, що Google, маючи відповідні ресурси та правильне бачення, швидко купувала успішні проєкти, які надавали певну послугу. Яскравим прикладом цьому є Firebase Crashlytics, яка була викуплена у Twitter.



Сервіси Firebase за напрямками використання

Сьогодні можна впевнено сказати, що Firebase є повноцінною бекенд-платформою для мобільних та веб-застосунків, оскільки величезна кількість сервісів охоплює усі найважливіші напрямки розвитку додатку. Цими напрямами є:

❖ Розробка – безпосередньо розробка бекенду для застосунку, включно з усіма необхідними ресурсами. Firebase надає для таких функцій послуги хмарного сховища, динамічної бази даних реального часу, автентифікації, хостингу та навіть комп'ютерного бачення.

❖ Контроль якості – перевірка якості програмного забезпечення додатку. Firebase для цього надає послуги відстеження аварійних зупинок додатку, відстеження продуктивності серед усіх користувачів, а також автоматизовані тести інтерфейсу

❖ Розвиток та масштабування – спектр різних послуг, які допомагають визначити подальший розвиток застосунку, та пристосувати його до збільшення масштабів. Сюди входять аналітика, віддалене конфігурування, хмарний обмін повідомленнями, динамічні посилання, A/B тестування та багато іншого.

Так, кількість сервісів у Firebase справді величезна. Проте у даній роботі увага звертається, перш за все, на сервіси, корисні для Android-розробки, та ті, які найчастіше використовуються. Ними є:

- **Firebase Cloud Messaging –** це крос-платформове хмарне рішення для надсилання повідомлень та сповіщень на Android, iOS та веб-застосунки. Він замінила попередній сервіс Google для надсилання сповіщень на Android-додатки, Google Cloud Messaging, та додав підтримку інших платформ. Наразі, послуга Firebase Cloud Messaging є повністю безкоштовною та є стандартним компонентом сучасних мобільних додатків.

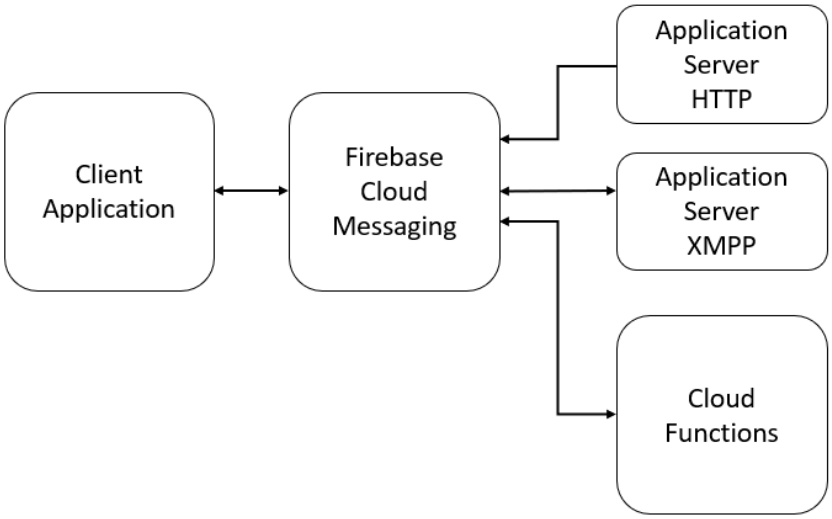
Firebase Cloud Messaging успадкував кореневу структуру Google Cloud Messaging, проте значно спростив розробку клієнтської сторони. Проєкт має 3 важливі якості:

1) він дозволяє користувачу отримувати сповіщення або повідомлення з даними , які можуть бути розшифровані клієнтським застосунком;

2) він дозволяє обирати цілі для сповіщень та повідомлень: це можуть бути індивідуальні пристрої, певні групи пристрої, або ж пристрої, підписані на певні окремі канали;
3) між клієнтським застосунком та сервером встановлюється канал зв'язку, який дозволяє надсилати повідомлення багатьох типів з обраних пристроїв або клієнтських застосунків.

Архітектура Firebase Cloud Messaging складається з трьох компонентів:

1) сервер з'єднання FCM;
2) захищене середовище з сервером застосунків, який використовує протокол HTTP або XMPP, та має хмарні функції:
3) клієнтський додаток.



Архітектура Firebase Cloud Messaging

Надсилання та отримання повідомлень потребує захищене середовище або сервер, аби створювати, спрямовувати та надсилати повідомлення, а також клієнтський застосунок для отримання повідомлень.

Через Firebase Cloud Messaging можна надсилати 2 види повідомлень: сповіщення та повідомлення з даними. Вони мають ключову різницю: повідомлення-сповіщення обробляються SDK FCM, який автоматично виводить їх на екран; повідомлення ж з даними оброблюються власне клієнтським застосунком.

Таким чином, розробники можуть всіляко використовувати ці типи повідомлень: надсилати просто сповіщення, аби Firebase Cloud Messaging їх показав на екрані пристрою; або надсилати повідомлення з даними, щоб опрацювати їх за допомогою власної складнішої логіки всередині додатку.

Також варто відзначити наявність інтеграції Firebase Cloud Messaging з Firebase Analytics, що дозволяє відслідковувати статистику використання сервісу. Ця інтеграція є дуже корисною і зручною, але не є унікальною. Практично усі сервіси Firebase співпрацюють таким або іншим чином.

- **Realtime Database –** перший продукт компанії Firebase, хмарна NoSQL база даних, яка синхронізується в режимі реального часу. У 2017 році фактично була замінена новим удосконаленим продуктом – Cloud Firestore.

Realtime Database є хмарною базою даних, яка надається в одному екземплярі для усіх платформ проєкту. Тобто, кожен iOS, Android чи веб-застосунок при наявності зберігає з'єднання з цим єдиним екземпляром бази даних, який у режимі реального часу отримує оновлення від усіх клієнтів, синхронізує їх та розсилає клієнтам.

Дані у Realtime Database зберігаються у формі одного JSON дерева, що є перевагою для простих даних, але створює проблеми за більших масштабів. Realtime Database також підтримує офлайн-сховище на Android-пристроях.

Realtime Database є помітно простішою та менш пристосованої до великих даних в порівнянні зі своїм наступником (Cloud Firestore), оскільки має меншу підтримку атомарних операцій, не дозволяє одночасно фільтрувати та сортувати дані в одному запиті, запити повертають усі дані документів, а продуктивність бази даних є обернено пропорційною до її розміру.

Плата знімається за пропускну здатність та розмір сховища, за вищими тарифами, ніж Cloud Firestore.

- **Cloud Firestore –** наступник першого продукту компанії, Realtime Database; хмарна NoSQL база даних, яка синхронізується в режимі реального часу.

На відміну від Realtime Database, дані у цій базі більш структуровані: інформація зберігається у документах, які містять поля, що вказують на значення. Ці документи зберігаються у колекціях, які є контейнерами для документів, що можна використовувати для збереження даних та побудови запитів.

Документи підтримують чимало типів даних: від простих чисел та стрічок до складних, вбудованих об'єктів. Існує також можливість зберігати вбудовані колекції всередині документів та розбудовувати ієрархічні структури даних, що органічно масштабуються разом зі зростаючою базою даних.

До того ж, Cloud Firestore значно поліпшив зручність та ефективність запитів. Є можливість створювати «неглибокі» запити – такі, що повертають дані на рівні документів, без повернення усієї колекції, або вбудованих наборів даних. Також, можна одночасно задавати сортування, фільтри та обмеження для запитів, або ж додавати курсори для пагінації результатів.

Для синхронізації клієнтського застосунку з оновленнями бази, додають слухачі реального часу, які дозволяють при оновленні бази стягувати лише нову інформацію, а не всю базу даних.

Cloud Firestore підтримує інтеграцію з сервісом Firebase Authentication для кращого захисту даних, а також додаткові налаштування безпеки.

- **Cloud Storage –** сервіс хмарного сховища для файлів, який використовує потужності схожих сервісів Google. SDK Firebas-у додають також захист Google Security до всіх вивантажень та завантажень застосунків Firebase незалежно від швидкості мережі.

Даний SDK можна використовувати для зберігання зображень, відео, звуків та інших файлів користувачів. Аби отримувати доступ до цих даних на сервері, потрібно використовувати Google Cloud Storage API. Найцікавіше те, що API дозволяє на стороні сервера проводити обробку файлів: додавання фільтрів на зображення, конвертація відео, тощо.

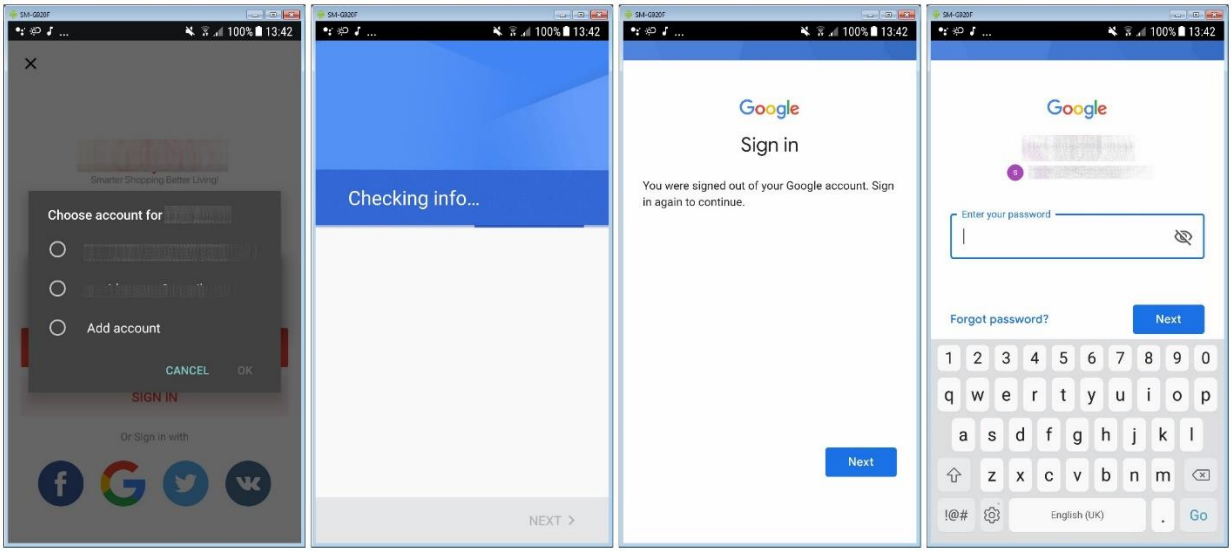
Всі файли доступні як через сервіси Firebase, так і з сервісу Google Cloud. Це додає гнучкості та більше варіантів для завантаження та вивантаження файлів.

SDK для Firebase Cloud Storage підтримує безшовну інтеграцію з сервісом Firebase Authentication, а для безпеки використовується декларативна мова правил безпеки. Таким чином, можна контролювати індивідуальні файли або групи файлів, та змінювати їх налаштування приватності.

- **Firebase Authentication –** сервіс та SDK, які надають послуги автентифікації у клієнтських застосунках**.** Він підтримує автентифікацію з використанням номерів телефону, адреси електронної пошти, паролів, окремих постачальників ідентифікаторів, сервісів Google, Facebook, Twitter, тощо.

Firebase Authentication є тісно інтегрованим у інші сервіси Firebase, та використовує протоколи Oath 2.0 та OpenID Connect. Завдяки цьому, він легко влаштовується у власний бекенд застосунку.

SDK надає кілька можливостей для додавання авторизації у додаток: власна реалізація synthatqce з використанням API Firebase Authentication, або ж використання бібліотеки FirebaseUI, яка містить зручні готові рішення для інтерфейсу входу та авторизації.



Імплементація інтерфейсу автентифікації з використанням бібліотеки Firebase UI

Використання готового рішення для інтерфейсу є рекомендованим способом додавання автентифікації у застосунки.

Також, перевагою Firebase UI є обробка виняткових ситуацій, таких як відновлення та зв’язування облікового запису, які є чутливими у плані безпеки та вразливими до помилок при правильній обробці.

Firebase UI дозволяє зручно змінити вбудований інтерфейс, аби він відповідав решті інтерфейсу усього додатку. А оскільки ця бібліотека розповсюджується за політикою відкритого сирцевого коду, немає обмежень у можливостях для її налаштування.

- **Google Analytics –** це сервіс для провадження аналітики застосунків, який дозволяє створювати та відстежувати власноруч визначені події та відповідно реагувати. Google Analytics є інтегрованим практично у кожен інший сервіс Firebase та є цілком безкоштовним.

Сервіс дозволяє визначити до 500 подій через Firebase SDK, та переглядати аналітику спрацювання даних подій у консолі Firebase. SDK автоматично захоплює спрацювання подій, та зберігає їх кількість разом з параметрами користувачів.

Найкращою рисою сервісу є глибокий аналіз аудиторії та можливість налаштування реакцій на зміни в аналітиці. Так, можна налаштувати спрацювання налаштувань у інших сервісах Firebase.

Наприклад, можна аналізувати кількість користувачів з найчастішими аварійними зупинками додатку через Firebase Crashlytics. Коли число таких користувачів досягає певної межі, можна надіслати їм певні сповіщення через Messaging. І це ще не все: можлива також інтеграція вашої аналітики з Google Tag Manager, або ж її експортування до BigQuerry, задля глибшого аналізу та використання даних з інших джерел.

- **Firebase Crashlytics –** це сервіс моніторингу та звітування у реальному часу аварійних зупинок застосунку. Сервіс є розвитком самостійного проекту під назвою Crashlytics, який був заснований у 2011 році, поглинутий Twitter у 2013 році, та врешті придбаний Google та приєднаний до платформи Firebase у 2017 році.

Crashlytics не лише звітує про аварійні завершення роботи додатків, а й проводить набагато більший обсяг роботи:

1) групує повторювані помилки у списки проблем, з контекстуальною інформацією, критичністю проблеми та частотою її повторення;
2) надає повну інформацію про пристрої, на яких відбулося аварійне завершення роботи, а також трасу стеку, що значно полегшує зневадження;
3) надсилає сповіщення у реальному часу про нагальні потреби, які потребують терміново рішення або уваги.

Також, Firebase Crashlytics дозволяє змінювати та персоналізувати правила звітування та моніторингу, задля найбільшої ефективності аналітики.

- **Firebase Performance Monitoring** – сервіс відстеження продуктивності роботи застосунку на всіх пристроях. Послуга є повністю безкоштовною для iOS та Android-додатків.

Для відстеження основних метрик достатньо просто додати відповідний SDK у додаток, який без жодних додаткових налаштувань починає відстежувати час запуску додатку, швидкість процесів застосунку та HTTP-запитів.

Також, можна налаштовувати відстеження специфічних частин додатку та, як у випадку Crashlytics, отримувати сповіщення при появі масових критичних проблем з продуктивністю.

- **Firebase Test Lab –** сервіс, який надає хмарну інфраструктуру для тестування додатків на найрізноманітніших пристроях та конфігураціях. Є можливість тестувати додатки як на віртуальних пристроях(дешевша опція), так і реальних.

Фізичні пристрої розташовані у дата-центрах Google, мають найостанніші версії API та підтримують зміну налаштувань локалізації.

Тестування проводиться в більшості двома методами:

1) Інструментальні тести прописуються безпосередньо розробниками та дозволяють перевірити усі особливі ситуації з заздалегідь визначеними даними та умовами.

2) «Robo-тест» не потребує прописаного сценарію, а самостійно перевіряє весь додаток на базову функціональність.

Це надзвичайно актуальний сервіс для Android-розробників, оскільки велике різноманіття цільових обгорток Android робить вкрай невигідним купівлю фізичних пристроїв для тестування.

- **Firebase ML Kit –** мобільний SDK, що надає функціонал машинного навчання у додатки. Сюди входять розпізнавання різноманітних типів візуальної інформації розпізнавання мови, переклад та багато інших задач.

Сервіс надає готові навчені моделі для використання, з яких деякі можна використовувати як онлайн, так і офлайн.

Feature	On-device	Cloud
Text recognition	✓	✓
Face detection	✓	
Barcode scanning	✓	
Image labeling	✓	✓
Object detection & tracking	✓	
Landmark recognition		✓
Language identification	✓	
Translation	✓	
Smart Reply	✓	
AutoML model inference	✓	
Custom model inference	✓	

Доступний функціонал на пристроях і хмарі

За потреби, розробник може завантажити власну TensorFlow Lite модель на хмару Firebase, та використовувати її у рамках сервісу Firebase ML Kit.

- **Firebase Remote Config –** це сервіс хмарних конфігурацій, який можна використовувати для динамічної міни конфігурації мобільних застосунків, не примушуючи користувачів завантажувати оновлення.

Використовуючи консоль Firebase, розробник створює мапу значень конфігурації(включно зі значеннями за замовчуванням). При запуску, додаток

перевіряє оновлення конфігурації та за потреби зберігає нові значення, які буде використовувати при відсутності з’єднання.

Firebase Remote Config дозволяє зберігати різні типи даних, а також має тісну інтеграцію з Google Analytics.

Як бачимо, Firebase покриває практично усі потреби мобільних розробників своїми численними сервісами. Багато з них безкоштовні, або мають безкоштовний тарифний план «Spark», роблячи Firebase вкрай привабливим з погляду фінансів та ресурсів.

А це викликає запитання, яке розглядається у наступному підрозділі.

2.3 Firebase чи власна серверна частина?

Firebase це справді надзвичайно функціональна, зручна та масштабна платформа, яка здатна у рази пришвидшити розробку повноцінного додатку. Її користь важко заперечити, а оскільки чимало її сервісів є безкоштовними, то інтеграція з Firebase у тому чи іншому ступені є ледь не обов’язковою умовою для якісних та функціональних мобільних додатків.

Наприклад, Firebase Cloud Messaging є стандартним рішенням для реалізації сповіщень на мобільних додатках. А також, всі сервіси аналітики (Analytics та Crashlytics) є безкоштовними, а тому використовуються також використовуються майже у всіх сучасних мобільних додатка.

Більшість платних послуг Firebase мають доволі щедрі квоти в рамках тарифного плану «Spark», які, якщо дуже спростити та узагальнити для усіх сервісів, розраховані приблизно на 10 000 користувачів. При більших обсягах, використовується тарифний план «Blaze», який знімає місячну плату залежно від використаних об’ємів даних, кількостей операцій, записів, користувачів, тощо.

Саме тому, необхідне чітке розуміння перспектив розвитку, розміру аудиторії та монетизації додатку. Firebase є чудовим вибором для швидкого запуску додатку, а також сервіси та архітектури платформи добре масштабуються зі зростанням проєктів.

Проте не варто забувати, що деякі сервіси Firebase все-одно потребують власний сервер, або захищене середовище роботи, тож розробникам все-одно потрібно буде налаштовувати не лише клієнтські застосунки, а й сервери.

Також, Firebase не надає можливості виконувати власну логіку на серверах платформи, тож будь-який більш-менш складний додаток потребує налаштування власного бекенду.

У підсумку, можна зробити висновок, що бекенд на основі Firebase найбільше підходить для простих додатків, які потребують швидкого масштабування та надійності структури. Подальше використання сервісу мусить вирішуватися на основі оцінки аудиторії, фінансових ресурсів та складності реалізованої логіки.

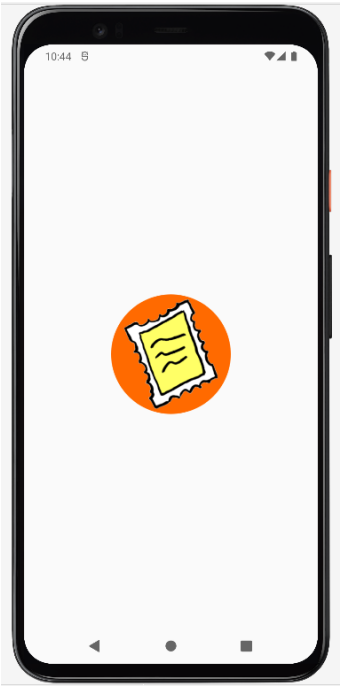
Розділ 3. Огляд Android-додатку з використанням сервісів Firebase

3.1 Реалізація інтерфейсу додатку

Інтерфейс додатку є доволі простим і в більшості випадків не відходить від стандартних бібліотечних налаштувань. Його головною метою є продемонструвати використання фреймворку Compose UI та певних функцій, доданих у Android 12.

3.1.1 Використання SplashScreen API

Так, одним і нововведень Android 12 є SplashScreen API, який дозволяє налаштовувати екран заставки додатку, не створюючи при цьому окремого вікна заставки. Так, за замовчуванням для екрану заставки використовується круглий ярлик додатку, з фоном, визначеним для цього ярлика. У даній роботі використано реалізацію за замовчуванням.



Екран заставки, створений за допомогою SplashScreen API

Можливості для налаштування екрану заставки насправді вкрай широкі, від зображень та кольорів до анімацій та взаємодії.

При спробі додати окреме вікно заставки, Android Studio буде відображати попередження про необхідність позбутися даного вікна, оскільки перед ним все-

одно буде показаний екран заставки зі SplashScreen API. Якщо за певних причин прибирати вікно немає змоги, а два екрани заставки не потрібні, до Activity екрану заставки потрібно додати анотацію @CustomSplashScreen.

3.1.2 Використання Compose UI

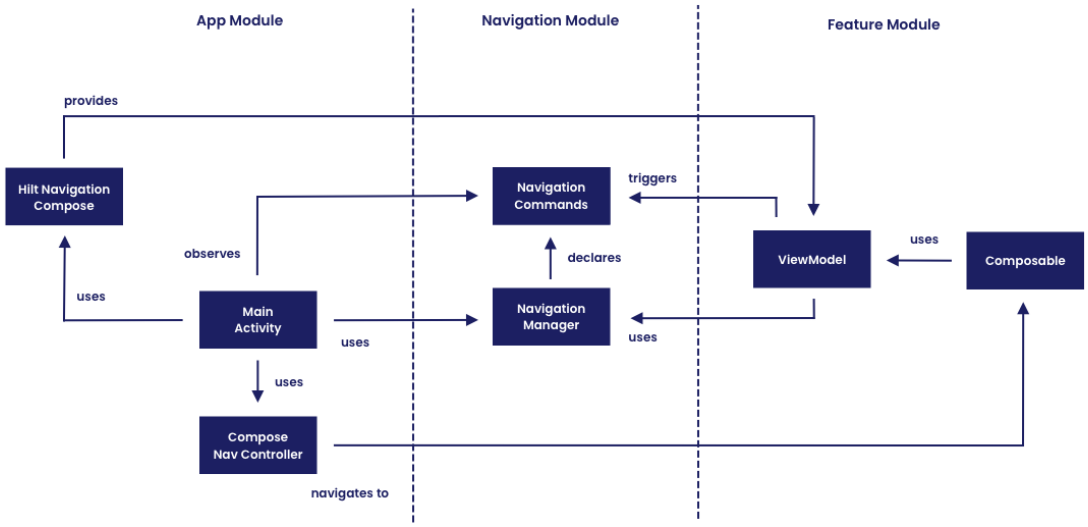
Jetpack Compose – це набір інструментів від Google для створення користувацького інтерфейсу додатків Android. Jetpack Compose надає принципово новий API під назвою Compose UI, який дозволяє писати інтерфейс декларативним способом, повністю мовою Kotlin.

Раніше, основними одиницями інтерфейсу Android були активності(Activity) та фрагменти(Fragment), що складалися з Kotlin/Java коду зв’язаного з розміткою xml. Активності фактично були основними суцільними вікнами на весь екран, які за потреби створювали ланцюжки фрагментів. З Compose UI , кожне вікно тепер є @Composable функцією, яка промальовує інтерфейс залежно від наявних даних та реагує на зміни в цих даних.

Для написання даного інтерфейсу треба значно менше коду, ніж для верстки xml дизайну; він є значно зрозумілішим для розробників, має кращу структуру та можливості для зневадження; і ще вкрай важливе – спрощується життєвий цикл вікон додатку, та підвищується стабільність роботи.

Оскільки вікна тепер промальовуються через функції, зникла потреба у численних взаємопов’язаних Activity та Fragment. Потрібна лише одна Activity, яка буде утримувати в собі інтерфейс, відображений через @Composabe функції.

Для навігаціями між цими вікнами-функціями використовується тепер функція NavigationHost та об’єкт NavigationController. NavigationHost є фактично оболонкою та каркасом для ієрархії вікон-функцій Compose, та дозволяє задати початкове вікно, всі вікна в ієрархії та їх теги для навігації, а також об’єкт NavigationController для збереження стану навігації.



Архітектура та навігація між вікнами Compose UI

NavHost потрібно задавати як корінь головної активності(Activity) додатку, що і було виконано у StampNet.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
        val navController = rememberNavController()
        NavHost(navController = navController, startDestination = "stampsMain") {  this: NavGraphBuilder
            composable( route: "stampsMain") { StampsMainView(navController = navController) }
            composable( route: "profileView") { ProfileView(navController = navController) }
            composable( route: "profileView") { StampItemCreationView(navController = navController) }
            composable( route: "profileView") { SettingsView(navController = navController) }
        }
    }
}
```

Вікна та навігація у StampNet

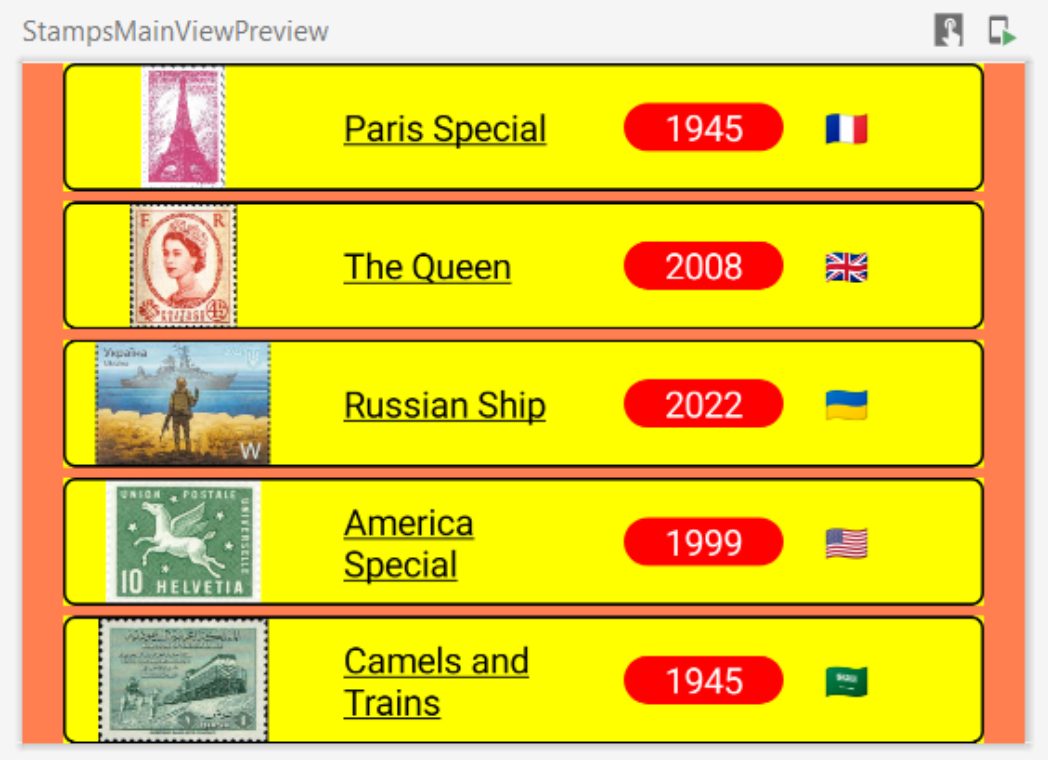
Кожному вікну передається екземпляр navController, аби за допомогою методу navigate(ТЕГ_ВІКНА) переходити до інших вікон.

Для зберігання значень змінних у функціях Compose використовуються делегати з лінивими змінними та функція rememberSaveable, яка відповідає за збереження значення змінної навіть коли функція обчислюється знову.

```
var login by rememberSaveable { mutableStateOf( value: "") }
```

Приклад використання змінних у вікнах-функціях Compose UI

Compose UI дозволяє дуже зручно відображати списки з багатьох елементів, оскільки не потребує створення окремих адаптерів та не вимагає керуванням їх життєвого циклу. Нижче показано список марок з бази даних, який реалізовано дуже швидко завдяки Compose UI.



Реалізація списку марок на Compose UI

Для цього була створена функція StampsMainView, яка приймає список марок для відображення.

```
@Composable
fun StampsMainView(navController: NavController, stamps: List<StampItem>) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .background(color = Color(0xFFFF7F50))
            .padding(horizontal = 16.dp),
        verticalArrangement = Arrangement.spacedBy(4.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        stamps.forEach {
```

```
                StampItemEntity(stamp = it)
            }
        }
}
```

Інформація про марку зберігається у об’єкті класу StampItem(зображення, назва, рік випуску та код країни):

```
data class StampItem(
    @DrawableRes
    val imageRes: Int,
    val name: String,
    val year: Int,
    val country: Locale
)
```

Код країни використовується, аби у списку марок показувати емодзі – прапорцем країни походження марки. Для цього створюємо розширення для класу Locale - flagEmoji:

```
val Locale.flagEmoji: String
    get() {
        val firstLetter = Character.codePointAt(country, 0) - 0x41 + 0x1F1E6
        val secondLetter = Character.codePointAt(country, 1) - 0x41 + 0x1F1E6
        return String(Character.toChars(firstLetter)) +
String(Character.toChars(secondLetter))
    }
```

Врешті, вся інформація про марку відображається за допомогою функції StampItemEntity:

```
@Composable
fun StampItemEntity(stamp: StampItem) {
    Row(
        modifier = Modifier
            .height(50.dp)
            .fillMaxWidth()
            .border(1.dp, Color.Black, RoundedCornerShape(5.dp))
            .background(Color.Yellow), verticalAlignment =
Alignment.CenterVertically,
        horizontalArrangement = Arrangement.spacedBy(16.dp,
Alignment.CenterHorizontally)
    ) {
        Image(
            painter = painterResource(id = stamp.imageRes),
            contentDescription = "",
            modifier = Modifier.weight(3f)
        )
        Text(
            text = stamp.name,
            modifier = Modifier.weight(3f),
            textDecoration = TextDecoration.Underline
        )
        Text(
            text = stamp.year.toString(),
```

```
                textAlign = TextAlign.Center,
                color = Color.White,
                modifier = Modifier
                    .weight(2f)
                    .background(
                        Color.Red,
                        RoundedCornerShape(10.dp)
                    )
            )
            Text(text = stamp.country.flagEmoji, modifier = Modifier.weight(2f))
        }
    }
```

Як бачимо, реалізація інтерфейсу інструментами Compose UI є надзвичайно зручною, функціональною, а тому і надійною. Також, деякий інтерфейс було реалізовано методами Firebase UI, про що буде згадано у наступних рядках.

3.2 Реалізація використання сервісів Firebase у додатку

Як було згадано у попередніх розділах роботи, Firebase Authentication не лише надає API для здійснення реєстрації та автентифікації користувачів, а ще й бібліотеку Firebase UI з готовими вікнами авторизації.

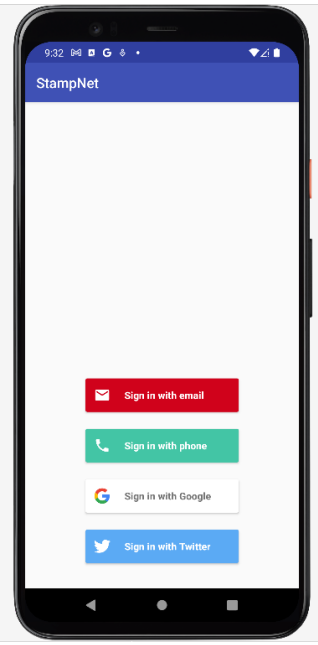
Оскільки використання готових рішень є рекомендованою Firebase практикою, StampNet для авторизації використовує потужності бібліотеки FirebaseUI. Використання бібліотеки є надзвичайно простим.

Для початку, необхідно запустити вікно автентифікації. Для цього створено метод launchSignInActivity(), який спрацьовує одразу після запуску додатку.

```
private fun launchSignInActivity() {
    val providers = arrayListOf(
        AuthUI.IdpConfig.EmailBuilder().setAllowNewAccounts(true).build(),
        AuthUI.IdpConfig.PhoneBuilder().build(),
        AuthUI.IdpConfig.GoogleBuilder().build(),
        //   AuthUI.IdpConfig.FacebookBuilder().build(),
        AuthUI.IdpConfig.TwitterBuilder().build()
    )
    val signInIntent = AuthUI.getInstance()
        .createSignInIntentBuilder()
        .setAlwaysShowSignInMethodScreen(true)
        .setAvailableProviders(providers)
        .build()
    signInLauncher.launch(signInIntent)
}
```

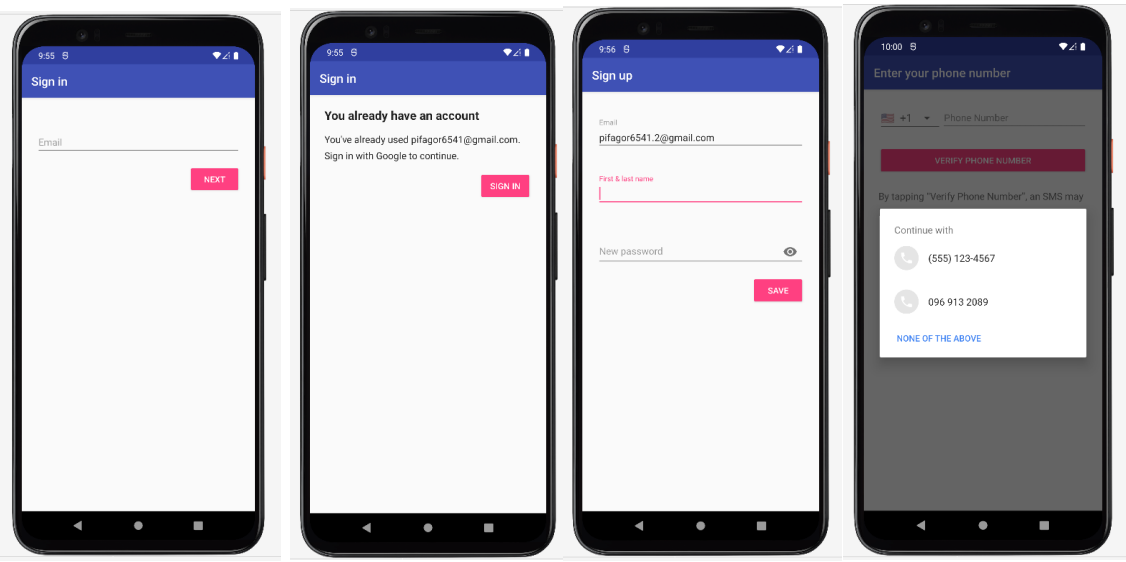
Аби показати можливі способи автентифікації, створюємо змінну provider, де прописуємо список об’єктів типу AuthUI.IdpConfig. Об’єкти даного класу тримають інформацію про спосіб автентифікації, і при побудові можна задавати чимало параметрів для них.

У цьому проєкті дозволена автентифікація через електронну пошту, номер телефону, обліковий запис Google та Twitter. Є також можливість додати автентифікацію через Facebook, проте для цього потрібне додаткове конфігурування проєкту в інструментах Facebook.



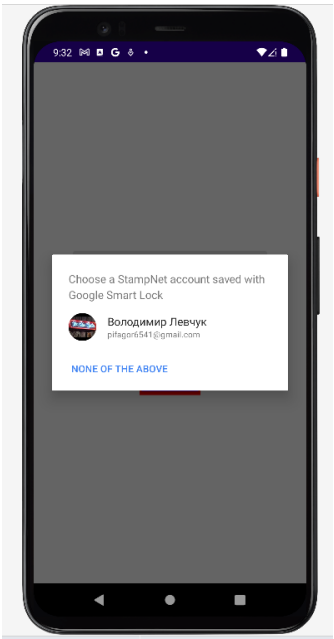
Доступні методи автентифікації

Після натискання на метод автентифікації будуть відкриті вікна входу або реєстрації через відповідний метод. У випадку з Facebook та Twitter можуть бути додатково відкриті відповідні сторінки у веб-браузері або ж їх додатки на даному пристрої.



Робота методів реєстрації

Якщо вхід одним із методів вже було здійснено раніше, авторизований акаунт буде збережено до SmartLock та запропоновано користувачеві при запуску.



Робота SmartLock при запуску

Також важливим є збереження даних, для чого у даному проєкті використовується Cloud Firestore. Cloud Firestore зберігає усі дані у документах, які збираються у колекції.

Дані у документах зберігаються у вигляді мап, тож при збереженні власних об’єктів потрібно врахувати ці перетворення. У проєкті за збереження нових марок відповідає метод saveStampToDataBase, який перетворює марку у мапу, та зберігає її до документа stamps у колекції stampNet.

```
private fun saveStampToDataBase(stampItem: StampItem){
    val database = Firebase.firestore
    val stamp = hashMapOf(
        "pictureId" to stampItem.imageRes,
        "name" to stampItem.name,
        "year" to stampItem.year,
        "locale" to stampItem.country
    )
    database.collection("stampNet").document("stamps")
        .set(stamp)
        .addOnSuccessListener {
            openStampsList()
        }
        .addOnFailureListener { e ->
           Toast.makeText(this,"Failed to save the
stamp!",Toast.LENGTH_LONG).show()
        }
}
```

Це головні інструменти Firebase, які були використані у проєкті StampNet. Загалом, можна зробити висновок, що авторизація та збереження простих даних через Firebase були правильним рішенням для такого проєкту любительського штибу.

Висновки

Метою даної роботи було попрактикуватися у використанні інструментів Firebase, перевірити доцільність використання платформи Firebase, а також поглибити знання з роботи Compose UI.

Було створено невеликий, проте наглядний Android-додаток, який дозволяє продемонструвати деякі особливості нововведень Android 12 та найважливіші сервіси Firebase. Під час роботи було засвоєно чимало нових знань та зроблено кілька висновків.

Перш за все, важливими стали здобуті знання з навігації між вікнами у Compose UI, а також збереження станів між викликами функцій. Це є надзвичайно корисним досвідом, оскільки позбавившись фрагментів у додатку та зменшивши кількість активностей, можна значно поліпшити його стабільність та зменшити розмір.

По-друге, було здобуто досвід роботи з автентифікацією через Firebase Authentication, а особливо корисною стала практика з бібліотекою Firebase UI. Зручні готові реалізації інтерфейсів входу однозначно є корисними функціями для нового додатку.

Врешті, можна зробити два висновки:

1) Compose UI є критично важливим оновленням для існуючих додатків, та обов’язковим для нових, оскільки має величезний вплив на продуктивність та стабільність додатків;
2) Платформа Firebase завжди потрібна в тому чи іншому вигляді майже у всіх мобільних застосунках, а можливості її бекенду, який є гарним рішенням і для промислових масштабів, варто використовувати хоча б для MVP або любительського додатку.

Список використаних джерел

1. https://www.theverge.com/2021/5/18/22440813/android-devices-active-number-smartphones-google-2021

2. https://firebase.google.com/docs/firestore/manage-data/add-data

3. https://firebase.google.com/docs/firestore

4. https://techcrunch.com/2014/10/21/google-acquires-firebase-to-help-developers-build-better-realtime-apps/

5. https://medium.com/google-developer-experts/modular-navigation-with-jetpack-compose-fda9f6b2bef7

6. https://developer.android.com/guide/topics/ui/splash-screen

7. https://developer.android.com/about/versions/12/features

8. https://firebase.google.com/docs