МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Литвиненко О.Є.
«___»_____2021 р.

ДИПЛОМНИЙ ПРОЄКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Програмний модуль для онлайн перегляду потокового відео на мобільному пристрої

Виконавець: Ахмедова К.М.

Керівник: Нечипорук О.П.

Нормоконтролер: Тупота Є.В.

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Освітнього ступеня бакалавр

Напрям (спеціальність) 123 "Комп'ютерна інженерія"
(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Литвиненко О. Є.

«____»_____2021 р.

ЗАВДАННЯ
на виконання дипломного проекту

Ахмедової Каміли Михайлівни
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема роботи: "Програмний модуль для онлайн перегляду потокового відео на мобільному пристрої "

затверджена наказом ректора від "04" лютого 2021 року № 135 /ст.

2. Термін виконання роботи: з 17.05.2021 до 20.06.2021

3. Вихідні дані до проєкту: постановка задачі до виконання роботи, мова програмування *Dart, фреймворк Flutter,* середовище програмування *IntelliJ IDEA*

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1) аналіз існуючих мов програмування для мобільної розробки;

2) можливості використання фреймворків для створення кросплатформного додатку;

3) розробка програмного модулю для онлайн перегляду потокового відео на мобільному пристрої.

5. Перелік обов'язкового графічного матеріалу:

1) діаграма бази даних;

2) схема взаємодії екранів додатку;

3) дизайн-макет додатку;

4) головне вікно додатку "*Watch Now*" бази даних.

6. Календарний план

№ п/п	Етапи виконання дипломного проекту	Термін виконання етапів	Примітка
1	Ознайомитись з постановкою задачі дипломного проектування	17.05.21-18.05.21	
2	Вивчити спеціальну літературу і технічну документацію	18.05.21-19.05.21	
3	Написати та налагодити програму	19.05.21-30.05.21	
4	Написати пояснювальну записку	31.05.21-10.06.21	
6	Підготовити графічний і демонстраційний матеріали	11.06.21-14.06.21	
7	Захист дипломного проекту	14.06.20-20.06.21	

7. Дата видачі завдання « 17 » травня 2021 р.

Керівник _____ Нечипорук О.П.
(підпис керівника)

Завдання прийняв до виконання _____ Ахмедова К.М.
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Програмний модуль для онлайн перегляду потокового відео на мобільному пристрої”: 52 с., 14 рис., 2 табл., 18 літературних джерел, 3 додатки.

Ключові слова: *DART, FLUTTER, API*, ПОТОКОВЕ ВІДЕО, МОБІЛЬНА РОЗРОБКА, КРОСПЛАТФОРМНІСТЬ

Об’єкт дипломного проєктування – відтворення потокового відео на мобільному пристрої.

Предмет дипломного проєктування – програмний модуль для перегляду потокового відео на мобільному пристрої.

Мета дипломної роботи – дослідження можливостей використання мови програмування *Dart* та технології *Flutter* при створенні програмного модулю для онлайн перегляду потокового відео на мобільному пристрої.

Прогнози та припущення щодо розвитку об’єкта дослідження – створення робочого прототипу програми та використання його для тестування можливостей при перегляді відео з потокових сервісів на мобільному пристрої.

Результати дипломної роботи рекомендується використовувати в навчальному процесі для відтворення навчальних відеоматеріалів з декількох джерел, шляхом агрегації їх з різних сервісів в один додаток.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API – *Application Programming Interface* (інтерфейс прикладного програмування)

IDE – *Integrated Development Environment* (інтегроване середовище розробки)

JVM – *Java Virtual Machine* (віртуальна машина *Java*)

JSON – *JavaScript Object Notation* (запис об’єктів *JavaScript*)

SDK – *Software Development Kit* (набір засобів розробки)

ВСТУП

Близько 4.66 мільярди людей щодня використовують Інтернет [3]. Це приблизно 59.4% населення планети. З них 4.32 мільярди заходять у Всесвітню мережу через смартфон – 92.7% від загальної кількості користувачів. 52.5% населення використовують Інтернет для перегляду відеоконтенту. За час пандемії *Covid-19* кількість нових інтернет-користувачів через мобільні пристрої зросла на 4.3%. Найбільше смартфон використовують молоді люди віком від 18 до 24 років. На рис. 1 наведено статистику середнього часу використання смартфонів різними віковими категоріями населення.

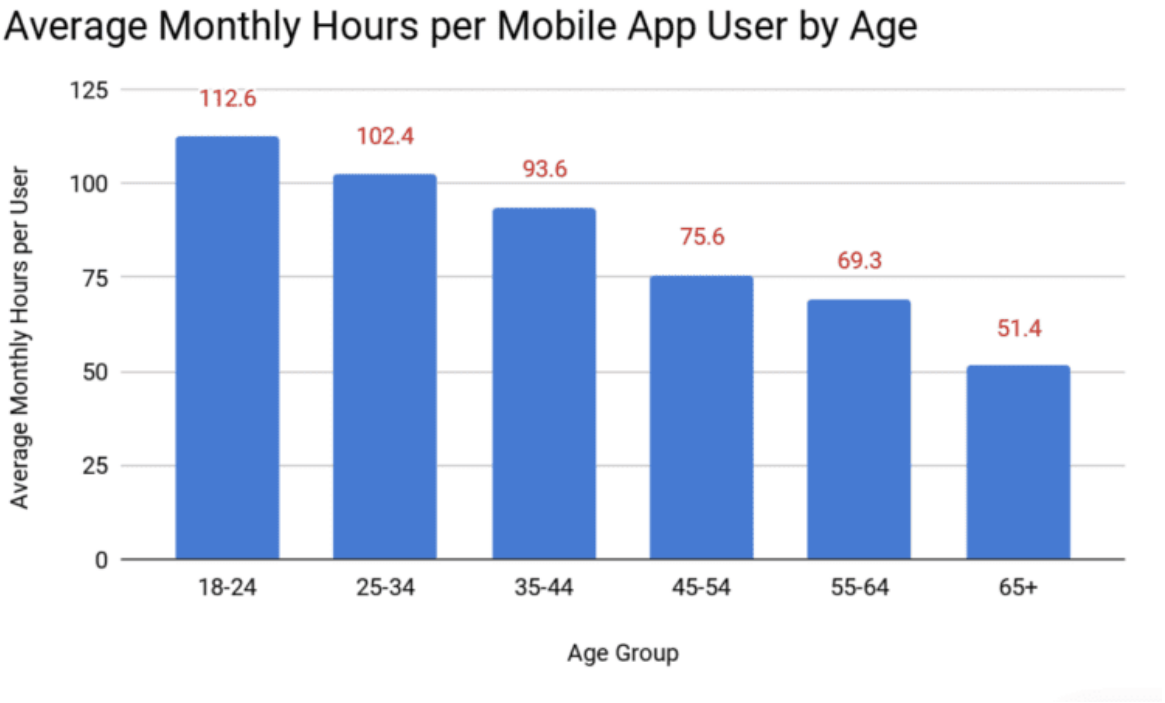

Рис. 1. Середнє використання мобільних додатків за місяць у годинах різними віковими категоріями

Потокові сервіси дозволяють отримувати дані безпосередньо від провайдера. Мультимедіа потоки бувають двох видів:

– прямі трансляції;

– по запиту.

Прямі трансляції доступні до перегляду короткий період часу.

Зазвичай це спортивні або кіберспортивні трансляції, майстеркласи, вебінари тощо. Потоки, що викликаються за запитом користувача, зберігаються на серверах довгий період часу. У даній роботі розглядаються сервіси обох видів.

Компанія *Sandvine* у 2019 році дослідила, яка кількість трафіку передається у мобільних додатках. Підкреслюючи популярність потокових сервісів, на *YouTube* приходиться найбільша кількість мегабайт – 37%, *Netflix* – 2.4%. Також за даними *SIMILARWEB* сайт *YouTube.com* [4] другий за кількістю відвідувань – 33.7 мільярди.

За даними *StatCounter* [5], ринок мобільних операційних систем наступний:

– *Android* – 71.81%;

– *iOS* – 27.43%;

– *Samsung* – 0.38%;

– *KaiOS* – 0.14%;

– *Linux* – 0.02%;

– Інші – 0.14%.

Виходячи з цих даних, було прийняте рішення розробити додаток для двох мобільних операційних систем: *Android* та *iOS*.

На сьогодні для розробки мобільних додатків найпоширенішими мовами програмування є *Swift*, *Kotlin* та *Dart*. *Swift* – мультипарадигмова мова програмування з відкритим кодом, яку презентувала компанія *Apple* безпосередньо для розробки додатків для операційних систем *MacOS* та *iOS*. *Kotlin* – це типізована мова програмування, яка прийшла на зміну *Java* та *Scala*. А з 7 травня 2019 року стала рекомендованою мовою програмування для розробки під *Android*. *Dart* – об’єктно-орієнтована імперативна мова програмування, яка розробляється та підтримується компанією *Google*. Вона була створена під впливом *Java*, *JavaScript* та *C#*. Основними перевагами даної мови є*:*

– кросплатформеність;

– синтаксис, який природний для програмістів на *JavaScript*, *C* і *Java*;

– швидкий запуск та висока продуктивність на сучасних мобільних пристроях;

– можливість створення однорідних систем, які охоплюють як клієнтську так і серверну частину.

Кросплатформені додатки є дуже зручними та економічно вигідними для бізнесів, оскільки нативні додатки для *Android* та *iOS* значно дорожче у розробці та підтримці. Кросплатформа – це набір інструментів (фреймворк), який дозволяє створювати додатки одразу під обидві мобільні системи. Найпоширенішими фреймворками є *ReactNative, Xamarin, PhoneGap, Titanium, Ionic, Flutter*. Найважливіші вимоги у виборі фреймворку:

– економічність у розробці;

– високий рівень досвіду взаємодії.

Економічність досягається не тільки у скороченні бюджету на розробку одного додатку замість двох, а і у підтримці, сервісних зборів, комісією магазинів *PlayMarket* та *AppStore*.

Не менш важливо щоб користувачі сприймали кросплатформений додаток так, ніби він написаний спеціально для цієї платформи. Це стосується анімації, інтерфейсів та жестів. Саме у користувацькому досвіді *Flutter* обходить своїх конкурентів.

Flutter новий погляд компанії *Google* на кросплатформеність. Фреймворк був реалізований ще у 2018 році, тому ще не набув широкого поширення у розробників. *Flutter* використовує інкапсуляцію, що є зручним для програміста та дешевшим для замовника. Також слід зазначити, що *Flutter* працює швидше і стабільніше за інші фреймворки – на частоті 60 кадрів в секунду. Це дозволяє зробити анімації більш плавними і запускати додатки навіть на старих моделях смартфонів. На веб-сайті *Flutter, Google* підкреслює основні переваги даного фреймворку:

– швидкодія мобільних додатків;

– гнучкий інтерфейс користувача, який досягається через використання віджетів;

– можливість розробки мобільних, настільних, а також веб – додатків.

Наявність автотестів допомагає скоротити кількість помилок, а виправлення помилки за логікою роботи для однієї операційної системи автоматично виправляє її для іншої.

Виходячи з проведених досліджень, було прийняте рішення створити кросплатформений мобільний додаток *VideOn*, який агрегує 3 потокових відео сервісів в одному додатку.

Об’єкт дипломного проєктування – відтворення потокового відео на мобільному пристрої.

Предмет дипломного проєктування – програмний модуль для перегляду потокового відео на мобільному пристрої.

Мета дипломної роботи – дослідження можливостей використання мови програмування *Dart* та технології *Flutter* при створенні програмного модулю для онлайн перегляду потокового відео на мобільному пристрої.

РОЗДІЛ 1
АНАЛІЗ ЗАСОБІВ РОЗРОБКИ СУЧАСНИХ МОБІЛЬНИХ ДОДАТКІВ

Мобільний додаток – це програмне забезпечення, яке виконується на мобільних пристроях. На сьогодні близько 5.7 мільярди людей на планеті мають смартфон, тож популярність розробки для мобільних пристроїв з кожним роком зростає. Збільшується попит на високоефективні та прості у користуванні мобільні додатки. Вибір найкращої мови програмування для розробки додатків багато в чому залежить від операційної системи. Тож для реалізації додатку, було розглянуто 5 найпопулярніших мов програмування для мобільної розробки за версією *DOU* [6]:

– *Swift*;
– *Kotlin*;
– *Dart*;
– *Java*;
– *C#.*

Можна виділити три основних типи мобільних додатків: нативні, кросплатформені та гібридні.

Нативні додатки – додатки, створені для конкретної платформи: *Android, iOS, Windows Phone* та інші. Вони розробляються за допомогою спеціальних інструментів та мов програмування, окремих для кожної платформи. Нативні додатки мають повноцінний функціонал, високу швидкодію та інтерфейс, розроблений відповідно до платформи. Однак вони не є універсальними, так як необхідно розробляти окремі додатки для кожної операційної системи.

Кросплатформені додатки – це рішення, завдяки якому вихідний код компілюється для виконання на декількох мобільних платформах. Це універсаль-

ний підхід який широко використовується для економії часу на розробку та коштів. Кросплатформені додатки прості у обслуговуванні та оновленні, мають широке охоплення та коротший час розміщення в маркетах [7].

Гібридні додатки отримуються за рахунок впровадження *HTML5* додатку усередину нативного контейнеру. Такі додатки прості та швидкі у розробці, на відміну від веб-додатків, є можливість розповсюдження через магазини додатків. Однак, такі додатки нестабільні та складні у масштабуванні та розвитку.

1.1. Аналіз існуючих мов програмування для мобільної розробки

Існує велика кількість мов програмування, які призначені для мобільної розробки. Кожна з них має свої особливості використання, переваги та недоліки. Нижче детально описано про п’ять сучасних мов: *Swift, Kotlin, Dart, Java* та *C#*.

1.1.1. Огляд мови програмування *Swift*

Swift – мультипарадигмова компільована мова програмування загального призначення, яка була представлена у 2014 році компанією *Apple* [8]. Основними сферами її застосування є розробка під *macOS, iOS, tvOS, watchOS*, а також додатків *Linux*. *Swift* порівнянний з основною кодовою базою *Apple*, яка написана на *Objective-C* та працює з фреймворками *Cocoa Touch* і *Cocoa*.

Swift поєднує в собі найкращі мови, зручні для розробників, такі як *JavaScript* та *Python*. Його синтаксис чіткий, стислий, легкий для розуміння та обслуговування. Працює швидше за свого попередника *Objective-C*, оскільки *Swift* вилучає обмеження мови *C* та використовує останні технології для покращення швидкості.

Swift безпечний завдяки набору специфічних функцій, призначених для уникнення помилок та збоїв коду. Також у цій мові програмування покращене управління пам’яттю, оскільки підтримується *ARC (Automatic Reference Counting)* у всіх *API*, що дозволяє розробникам економити час на керування пам'яттю.

Swift має відкритий код, завдяки цьому мова постійно збагачується корисними інструментами.

За думкою багатьох розробників, ця мова є поєднанням ефективності та надійності, що успадкована з різних компільованих мов програмування. *Swift* має ряд переваг у порівнянні з іншими мовами програмування:

– Стрімке поширення. Простий синтаксис – основна перевага цієї мови. Вилучається непотрібний складний синтаксис, що безпосередньо впливає на швидкість та якість розробки. *Swift* – дозволяє писати стислий та чистий код. Це робить *Swift* популярним серед розробників.

– Постійний розвиток. Мова має відкритий код, її постійно оновлюють та додають нові функції та інструменти.

– Ефективне використання пам’яті. *Swift* використовує динамічні бібліотеки, що є великою перевагою над статичними бібліотеками. Статичні бібліотеки можуть бути причиною збільшення часу завантаження, оскільки вони компілюються в коді програми. Динамічні бібліотеки не інтегровані в код і їх можна викликати за потребою. Статичні бібліотеки зазвичай мають копії всіх файлів у додатку, тоді як динамічні бібліотеки мають лише один файл.

– Автоматичне керування пам’яттю. *Swift* пропонує простий спосіб управління пам’яттю програми. Це допомагає захистити програму від витоку пам'яті завдяки автоматичному підрахунку посилань. За допомогою цієї функції *Swift* визначає, які екземпляри не потрібні. *ARC* відсікає їх, допомагаючи підвищити продуктивність, не впливаючи на пам’ять.

– Мала кількість вбудованих функцій. Більшість бібліотек та фреймворків розроблені для більш ранніх версій мови та неефективні для нових.

– Нестабільність. У нових версіях іноді трапляються помилки.

– Відсутність підтримки для попередніх версій *iOS*. *Swift* підтримується тільки у версіях новіших за *iOS7*.

1.1.2. Огляд мови програмування *Kotlin*

Kotlin – статично типізована мова програмування, яка була представлена у 2011 році компанією *JetBrains*. Основна мета розробки *Kotlin* – створення більш лаконічної мови, ніж *Java* та простішої за *Scala*. *Kotlin* розроблений як об'єктно-орієнтована мова, яка повністю взаємодіє із кодом Java , що дозволяє поступово перевести розробку із *Java* на *Kotlin*.

У травні 2017 року *Kotlin* було оголошено офіційною мовою розробки *Android* на *Google*. Станом на 2020 рік *Kotlin* все ще найбільш широко використовується на *Android*, за оцінками Google, 70% з 1000 найкращих додатків у *PlayStore* написані в *Kotlin*. *Kotlin* на *Android* вигідний завдяки своїй безпеці з нульовими вказівниками, а також завдяки своїм функціям, які роблять коротший і читабельній код.

Kotlin також набирає популярності у розробці на стороні сервера, для цього використовується *Spring Framework*. Для подальшої підтримки *Kotlin, Spring* переклав всю свою документацію на *Kotlin* і додав вбудовану підтримку багатьох специфічних для *Kotlin* функцій, таких як програми. Також компанія *JetBrains* випустила перший для *Kotlin* фреймворк під назвою *Ktor* для створення веб-додатків.

У 2020 році *JetBrains* в опитуванні розробників, які використовують *Kotlin*, виявив, що 56% використовують *Kotlin* для мобільних додатків, тоді як 47% використовують його для веб-сервісу. Трохи більше третини всіх розробників *Kotlin* сказали, що вони мігрують до *Kotlin* з іншої мови.

Усе більше розробників схиляються до *Kotlin*. Основні переваги наведено нижче.

– Швидка розробка. *Kotlin* компактна мова. В неї стислий та інтуїтивний синтаксис. Таким чином, написання програми займає менше часу.

– Відповідність *Java*. *Kotlin* позиціонується як 100% сумісна з *Java* та усіма супутніми інструментами та фреймворками.

– Підтримка. *Kotlin* підтримується переважною більшістю середовищ розробки , включаючи *Android Studio* та інші інструменти *SDK*. Це сприяє

підвищенню продуктивності розробників, оскільки вони можуть продовжувати працювати з набором інструментів, до якого звикли.

– Відстежуваність помилок. Помилки виявляються під час компіляції, це дозволяє виправити їх до часу виконання.

До основних недоліків *Kotlin* можна віднести наступні:

– Швидкість компіляції. У деяких випадках *Kotlin* працює швидше, ніж *Java* – здебільшого при виконанні покрокових збірок. Але *Java* швидша, коли мова заходить про чисті збірки.

1.1.3. Огляд мови програмування *Dart*

Dart – об’єктно-орієнтована мультипарадигмова мова програмування, що була представлена компанією *Google* у 2010 році для створення серверних і настільних додатків. Основними задачами розробки *Dart* були:

– створення структурованої, але гнучкої мови програмування для веб-розробки;

– створення мови, що схожа на існуючі, з метою спрощення навчання;

– висока швидкодія програм, що розробляються.

Мобільні додатки, які написані на *Dart* у поєднанні із *Flutter* являються кросплатфомними нативними додатками, тому вони можуть працювати як на *Android* так і на *iOS*. Кодування в *Dart* стає природним, коли ознайомлено із загальними об'єктно-орієнтованими принципами. *Dart* має вбудовану підтримку для юніт-тестування, немає потреби додавати нові бібліотеки або фреймворки.

Dart має підтримку з боку компанії *Google*. Якщо у розробника є пропозиція щодо покращення *Dart SDK*, її можна подати на розгляд до *Google*. Це саме стосується і помилок. На відміну від патентованого ПЗ, мови із відкритим кодом завжди оновлюються та доповнюються. *Dart* має логічну структуру та інтуїтивний синтаксис, схожий на *C*, *Java* та *C#*. Можливість автоматичного виведення типів полегшує багато завдань програмування, дозволяючи програмісту вільно опускати анотації типів, дозволяючи перевірку типу. Це полегшує розробникам перехід на *Dart* незалежно від рівня їх програмування.

Майже кожен великий редактор тексту та *IDE* мають підтримку мови *Dart*. Можна використовувати важкі *IDE*, такі як *Webstorm*, *IntelliJ IDEA* та *Android Studio*, або прості редактори, такі як *VS Code*, *Sublime text*, *VIM*, *Emacs*, *Atom* тощо. Таким чином, розробники можуть обрати будь-який редактор.

Dart активно заміщує інші мови програмування. Він вирішує багато проблем та має високу ефективність.

Головною, проте не єдиною, перевагою *Dart* є можливість розробки кросплатформеної розробки [9]. Нижче наведено не менш важливі переваги.

– Висока продуктивність. Програми, написані на *Dart*, працюють швидше, ніж програми, створені на *JavaScript*.

– Стабільність. *Dart* дуже стабільний, і його можна використовувати для створення додатків високої якості в режимі реального часу. Це об'єктно-орієнтована мова програмування з підтримкою успадкування, інтерфейсів та додаткових функцій набору тексту.

– Наявність документації. Оскільки *Google* розробляє інтерпретатор для *Dart*, усі особливості мови докладно описані. Це дозволяє швидко отримати відповіді практично на будь-які питання, які можуть виникнути під час навчального процесу або безпосередньо під час написання коду.

– Використання компіляції *AOT* (*Ahead-of-time compilation*) та *JIT* (*Just-in-Time compilation*). У режимі *AOT* код *Dart* компілюється перед виконанням, а у режимі *JIT* – під час роботи програми, що прискорює розробку.

– Багатопоточність. Програма може виконувати багато завдань одночасно.

Звичайно, згадуючи про переваги, слід врахувати і недоліки.

– Обмеженість підтримки. *Dart* має дуже обмежені ресурси в Інтернеті, тому важко знайти відповіді на деякі питання. Причиною цього є відсутність більшої та згуртованої спільноти розробників.

– *Dart* в даний час розробляється. Існує певна ймовірність того, що *API* зміниться, або деякий функціонал ще не задокументований повністю.

1.1.4. Огляд мови програмування *Java*

Java – високорівнева, надійна, об'єктно-орієнтована та безпечна і стабільна мова програмування, яка була представлена компанією *Microsystems* у 1995 році [10]. *Java* є незалежною від платформи мовою, оскільки вона має середовище виконання, тобто *JRE* (*Java Runtime Environment*) та *API*. Тут платформа означає апаратне або програмне середовище, в якому працює додаток.

Коди *Java* компілюються в машинно-незалежний код. Цей код запускається на *JVM*.

Синтаксис *Java* майже такий самий, як *C / C++*. Але *Java* не підтримує такі функції програмування низького рівня, як покажчики. Коди в *Java* завжди пишуться у формі класів та об'єктів.

Станом на 2021 рік, *Java* - одна з найпопулярніших мов програмування [11]. За підрахунками, у світі існує близько дев'яти мільйонів розробників *Java*.

Метою створення мови програмування *Java* було те, що вона повинна бути проста, надійна, портативна, незалежна від платформи, захищена, високопродуктивна, об'єктно-орієнтована, інтерпретована та динамічна. Для мобільних додатків *Java* використовує *ME* (*Micro Edition*) або *J2ME* (*Java 2 Micro Edition*). Цей фреймворк є крос-платформою, яка запускає програми на телефонах та смартфонах. *Java* також забезпечує платформу для розробки додатків на *Android*.

Основні переваги *Java*:

– Простота. *Java* просто використовувати, писати, компілювати, налагоджувати та вивчати, ніж альтернативні мови програмування. *Java* менш складна, ніж *C* ++ [12].

– Об’єктно-орієнтованість. Це дозволяє формувати стандартні програми та код для багаторазового використання.

– Незалежність від платформи. Код *Java* працює на будь-якій машині, яка не потребує встановлення спеціального програмного забезпечення, але на машині має бути присутнім *JVM*.

– Розподілені обчислення. Розподілені обчислення включають кілька комп'ютерів у мережі, що працюють разом. Це допомагає розробляти програми в мережах, які можуть сприяти як передачі даних, так і функціональності додатків.

– Захищеність. *Java* не має явного вказівника. Крім цього, вона має менеджер безпеки, який визначає доступ до класів.

– Розподіл пам'яті. У *Java* пам'ять розділена на дві частини. Це допомагає зберігати інформацію та легко її відновлювати.

– Багатопоточність. Програма може виконувати багато завдань одночасно.

До недоліків можна віднести наступні:

– Швидкодія. *Java* займає багато пам’яті і значно повільніше, ніж компільовані мови, такі як *C* або *C* ++.

– Мова з однією парадигмою. Статичне імпортування було додано в *Java 5.0*. Процедурна парадигма краще застосовується, ніж у попередніх версіях *Java*.

– Керування пам’яттю. У *Java* керування пам’яттю здійснюється за допомогою збору кешу. Кожного разу, коли збирач кешу працює, це впливає на продуктивність програми. Це тому, що всі інші потоки повинні бути зупинені, щоб дозволити потоку збирача працювати.

1.1.5. Огляд мови програмування *C#*

C# – сучасна об'єктно-орієнтована мова програмування, яка була розроблена корпорацією *Microsoft* у 2000 році та відноситься до *C*-подібних мов, найбільш наближений його синтаксис до *C++* та *Java*. *C#* застосовують при створенні складних веб-сервісів, програм для настільних комп’ютерів та мобільних додатків. Компанія *Microsoft* активно підтримує та розвиває цю мову, тому регулярно з'являються оновлення та доповнення [13].

C# є лідером в рейтингах затребуваних мов програмування на ринку праці [14]. Переваги його використання наступні:

– Документація і доступність. *Microsoft* має докладну і розгорнуту документацію. Крім того, існує спільнота в мережі, яка може допомогти.

– Гнучкість. Інструментарій *C#* дозволяє вирішувати широке коло завдань, мова є універсальною. На ньому розробляють додатки для *WEB*, ігрові платформи, програми для *Windows*, додатки для платформ *Android* або *iOS*.

– Єдина система типів. У мові прийнята загальна система роботи з типами, починаючи від примітивів і закінчуючи складними, в тому числі, призначеними для користувача наборами.

C# має ряд недоліків:

– Пріоритетна орієнтованість розробки на платформу *Windows*.

– Вартість. Мова безкоштовна тільки для невеликих фірм, індивідуальних програмістів, стартапів та учнів.

1.2. Аналіз існуючих кросплатформених технології для розробки мобільного додатку

1.2.1. *React Native*

React Native – технологія від компанії *Facebook.* Мета її створення – розробка кросплатформенних додатків, що такі ж продуктивні, як і нативні. Використовується мова програмування *JavaScript* та бібліотека *React.* Фреймворк *React* Native досить популярний, його застосовують такі технологічні компанії, як *Facebook, Instagram, Tesla, Pinterest.*

Перша версія *React Native* була опублікована у березні 2015, тому він підтримується майже всіма провідними *IDE*. Застосування одного коду для різних платформ – головна ідея *React Native*. Також в цьому фреймворку є функція *Hot Reloading* (додавання нового коду без повторного виконання), що дозволяє редагувати код під час виконання – це дуже зручно, коли відбувається налаштування інтерфейсу користувача. Середа поставляється з великим набором готових компонентів, однак вони не завжди адаптуються під різні платформи, що вимагає додаткових коригувань в коді.

1.2.2. *Flutter*

Flutter – це кросплатформений набір інтерфейсів, призначений для повторного використання коду в таких операційних системах, як *iOS* та *Android*, а також дозволяє додаткам взаємодіяти безпосередньо з базовими службами платформи. Мета його створення полягає в тому, щоб дозволити розробникам доставляти високопродуктивні програми, які почуваються природньо на різних платформах, враховуючи відмінності там, де вони існують, одночасно надаючи якомога більше коду. *Flutter* використовується розробниками та організаціями з усього світу і є безкоштовним та відкритим.

Під час розробки програми *Flutter* працюють у віртуальній машині, яка пропонує *Hot Reloading* змін без необхідності повної перекомпіляції. Для випуску програми *Flutter* компілюються безпосередньо до машинного коду, будь то інструкції *Intel* або *ARM*, або до *JavaScript*, якщо націлено на Інтернет. Фреймворк є відкритим, з дозвольною ліцензією *BSD* і має процвітаючу екосистему сторонніх пакетів, які доповнюють основні функціональні можливості бібліотеки.

Для роботи з фреймворком використовується мова програмування *Dart* – об'єктно-орієнтована мова, розроблена *Google*, та наступні *IDE*: *Android Studio, IntelliJ IDEA* і *Visual Studio Code*.

Flutter – нова, проте дуже популярна технологія. Для розробки свої додатків його використовують такі корпорації, як *Google Ads, Alibaba*.

Flutter перевершує конкурентів у продуктивності і демонструє найвищі оцінки завдяки сучасної мови *Dart* і власним двигунам рендеринга.

1.2.3. *Iconic*

З *Ionic* можна створювати кросплатформені гібридні програми. Він працює у поєднанні із фреймворком *Apache Cordova*, який дозволяє з веб-додатків зробити мобільні додатки. Для розробки використовуються мови програмування *JavaScript*, *HTML* та *CSS*.

Фреймворк заснований на мовах програмування на *ECMAScript6* і *TypeScript*, тому його можна використовувати у будь-якій *IDE*.

Ionic також пропонує концепцію єдиного коду для різних операційних систем, але на новому рівні. Всі його компоненти автоматично адаптуються до операційної системи, на якій запускається додаток.

По продуктивності *Ionic* сильно відстає від вище розглянутих фреймворків. Для візуалізації використовуються веб-технології, а не нативні, як у вище розглянутих фреймворках. Але у такому підході є і плюси *Ionic* дозволяє проводити швидке тестування, яке можна запустити прямо в браузері.

1.2.4. *Xamarin*

Xamarin – технологія від компанії *Microsoft*, яка призначена для створення мобільних додатків, та додатків для *Windows*. Для розробки використовується мова програмування *C#* та програмна технологія *.NET*.

Як IDE можна використовувати, наприклад, *Visual Studio* або *Rider*. У *Xamarin* використовується два основні інструменти: Xamarin.Forms та *Xamarin.Android/iOS*. Для кросплатформеної розробки використовується *API Xamarin.Essentials*.

Xamarin.Android і *Xamarin.iOS* надають додаткам такий самий функціонал і зовнішній вигляд, який є у нативних рішень. У разі *Xamarin.iOS* відбувається *AOT*-компіляція, тоді як в *Xamarin.Android* спочатку компілюється байт-код, який потім інтерпретується віртуальною машиною.

Продуктивність *Xamarin* близька до нативної, але залежить від того, який інструмент використовується. Інструмент *Xamarin.Android/iOS* має хорошу оптимізацію завдяки нативним компонентам.

1.3. Мобільні операційні системи *Android* та *iOS*

Мобільні програми часто служать для надання користувачам послуг, подібних до тих, що доступні на ПК. Користуватися такими додатками на мобільних пристроях. Всі його компоненти автоматично адаптуються до

операційної системи, на якій запускається додаток. значно зручніше, ніж веб-сайтом, тому цей вид програмного забезпечення активно розвивається.

Популярність мобільних додатків обумовлена глибокою взаємодією користувача із додатком, а також широка сфера потреб, яку вони покривають: фінансові та державні послуги, розваги, спілкування та інші.

Найпоширеніші категорії додатків, що завантажувалися у 2020 році, наведено у таблиці 1.1.

Таблиця 1.1

Найпоширеніші категорії додатків, що завантажувались у 2020 році

Android	*iOS*
Соціальні мережі	Розваги
Розваги	Соціальні мережі
Продуктивність	Спосіб життя
Спосіб життя	Фото і відео
Здоров’я та фітнес	Музика

Найпоширенішими мобільними операційними системами є *Android* та *iOS,* вони покривають майже дев’яносто дев’ять відсотків ринку.

Мобільна операційна система – це операційна система, на якій працюють смартфони та планшетні комп’ютери. Для успішного розвитку операційних систем, потрібна різноманітність програмних продуктів. Окрім цього, потрібна постійна підтримка та оновлення версій, створення та розвиток інструментів для розробки ПЗ. Це вимагає великих фінансових та людських ресурсів, тому розробку власних операційних систем можуть дозволити собі тільки компанії з великими фінансовими активами.

Android – це мобільна операційна система, розроблена *Google*. Операційна система *Android* (ОС) базується на ядрі *Linux*. На відміну від *iOS* від *Apple*, *Android* є відкритим кодом, тобто розробники можуть змінювати та налаштовувати ОС для кожного телефону. Тому різні телефони на базі *Android* часто мають різні графічні графічні інтерфейси користувача, навіть якщо вони використовують одну і ту ж ОС.

Телефони *Android* зазвичай постачаються з декількома вбудованими програмами, а також підтримують сторонні програми. Розробники можуть створювати програми для *Android* за допомогою *SDK* для *Android*. Програми *Android* написані на *Java* та працюють через *Java JVM*, оптимізовану для мобільних пристроїв.

iOS – це мобільна операційна система, розроблена *Apple*. Інтерфейс *iOS* заснований на концепції прямої взаємодії з використанням жестів.

Оновлення ОС виходять щоквартально, а номерні версії приурочені зазвичай до виходу нового пристрою в лінійці *iPhone* і з'являються щороку. У нових версіях зазвичай додаються деякі нові функції і прибираються старі. З'являється підтримка нових можливостей для телефонів, наприклад, поліпшена камера. Підтримка збільшеної кількості ОЗУ і т.д. Також, виходять і бета версії, які при бажанні можна протестувати і самому.

1.4. Висновки до розділу

У даному розділі було проведено аналіз предметної області та поставлені задачі проектування.

Під час аналізу актуальності предметної області було виявлено збільшення кількості користувачів інтернету щороку, що в свою чергу призводить до збільшення кількості інтернет-ресурсів та стрімкого поширення мобільних пристроїв призвело до появи великої кількості мов програмування для мобільних операційних систем та технологій, із якими вони працюють. Кросплатформені додатки наразі переважають над нативними та гібрідними додатками, оскільки вони є зручнішими у розробці та економічно вигіднішими.

У 2020 році однією і найпопулярніших категорій у магазинах додатків для операційних систем *Android* та *iOS* була категорія розваги. Саме тому, було прийняте рішення створити додаток для перегляду відео, яке носить розважальний характер.

Було проведено огляд вже існуючих програмних рішень. Порівнюючи переваги та недоліки існуючих мов програмування і технологій для мобільної розробки, було виявлено, що оптимальним рішенням є вибір мови *Dart* у поєднанні із фреймворком *Flutter*. Оскільки *Flutter* є лідером серед технологій, швидко розвивається, та дозволяє створити додатки, які схожі на нативні, а *Dart* підтримує кросплатформену розробку, має високу швидкодію та, є більш стабільним.

РОЗДІЛ 2
ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ

2.1. Архітектурний шаблон програмування

Для розробки додатку необхідно визначитися із ключовим архітектурним рішенням – обрати архітектурний шаблон програмування, який передбачає високий рівень абстракції. Вибір архітектури є надзвичайно важливим, оскільки це безпосередньо впливає на швидкість розробки, гнучкість додатку, його стабільність.

Одним із паттернів, який використовується при розробці на *Flutter* є *Model-View-Controller* (*MVC*) (рис. 2.1). Цей шаблон призначений для розробки мобільних та веб-додатків. У цьому паттерні всі об’єкти мають одне з трьох призначень: *model* (модель), *view* (представлення), *controller* (контролер). *MVC* не тільки встановлює призначення для кожного з об’єктів, а й описує взаємодію між ними.

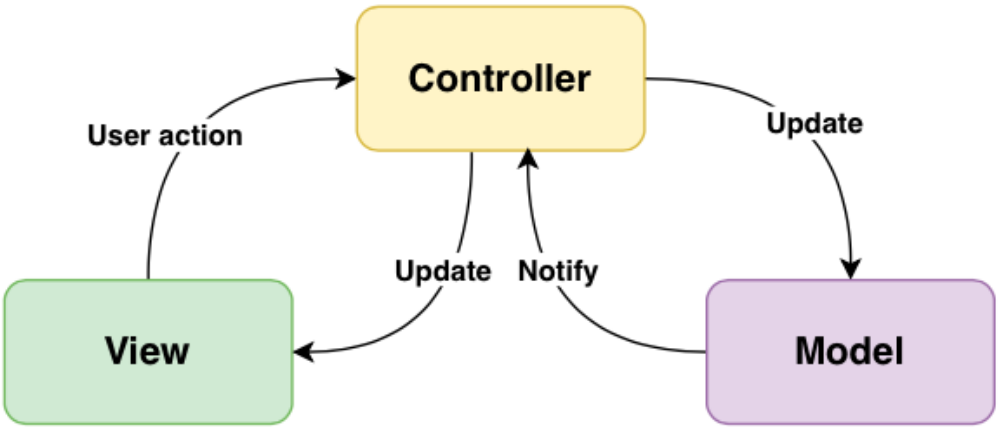

Рис. 2.1.*Model-View-Controller*

MVC працює за шаблоном “Контролер - Модель - Вид”. З використанням цього паттерну враховуються окремо різні елементи:

– логіка інтерфейсу користувача;

– логіка введення;

– бізнес-логіка.

Згідно шаблону, кожен елемент існує в додатку, але вони не взаємопов’язані. Логіка інтерфейсу користувача взаємопов’язана із *View*. Логіка введення має справу із *Controller*, а бізнес-логіка працює із *Model*.

Модель – являє собою модель деякої предметної області, яка зберігає у собі дані та методи їх обробки, реагує на запити із контролера та повертає дані або змінює свій стан. Модель описує функціональну бізнес-логіку проєкту, при цьому вона повністю незалежна від інших частин програми, містить опис лише самої себе.

Представлення – відображення даних моделі користувачу. Цей елемент стосується того, як пов'язати дані моделі, проте не містить жодної логіки щодо того, що з себе представляють ці дані або про те як користувач їх може використовувати.

Контролер – знаходиться між Моделлю та Представленням. Він відстежує всі події, які викликані Представленням та виконує відповідну до цих подій реакцію.

Основною перевагою використання даного патерну є незалежність Представлення від Моделі. Це робить код модульним і дозволяє повторно використовувати компонент, а також змінювати частини кода, не впливаючи при цьому на інші.

2.2. *Google Cloud Platform*

Для створення мобільного додатку використовується сервіс *Google Cloud Platform*. *Google Cloud Platform* – це пакет хмарних обчислювальних послуг, які надає компанія *Google*. *Google Cloud Platform* є частиною *Google Cloud*, яка також включає корпоративні версії ОС *Android* та *Chrome*, а також інтерфейси

прикладного програмування *(API)* для машинного навчання. Платформа пропонує сервіси для обчислення даних, зберігання та розробки додатків, які будуть працювати на операційній системі *Google*. Для використання сервісу необхідно створити акаунт адміністратора та зареєструвати проєкт. На рис.2.2 зображено інтерфейс *Google Cloud Platform*.

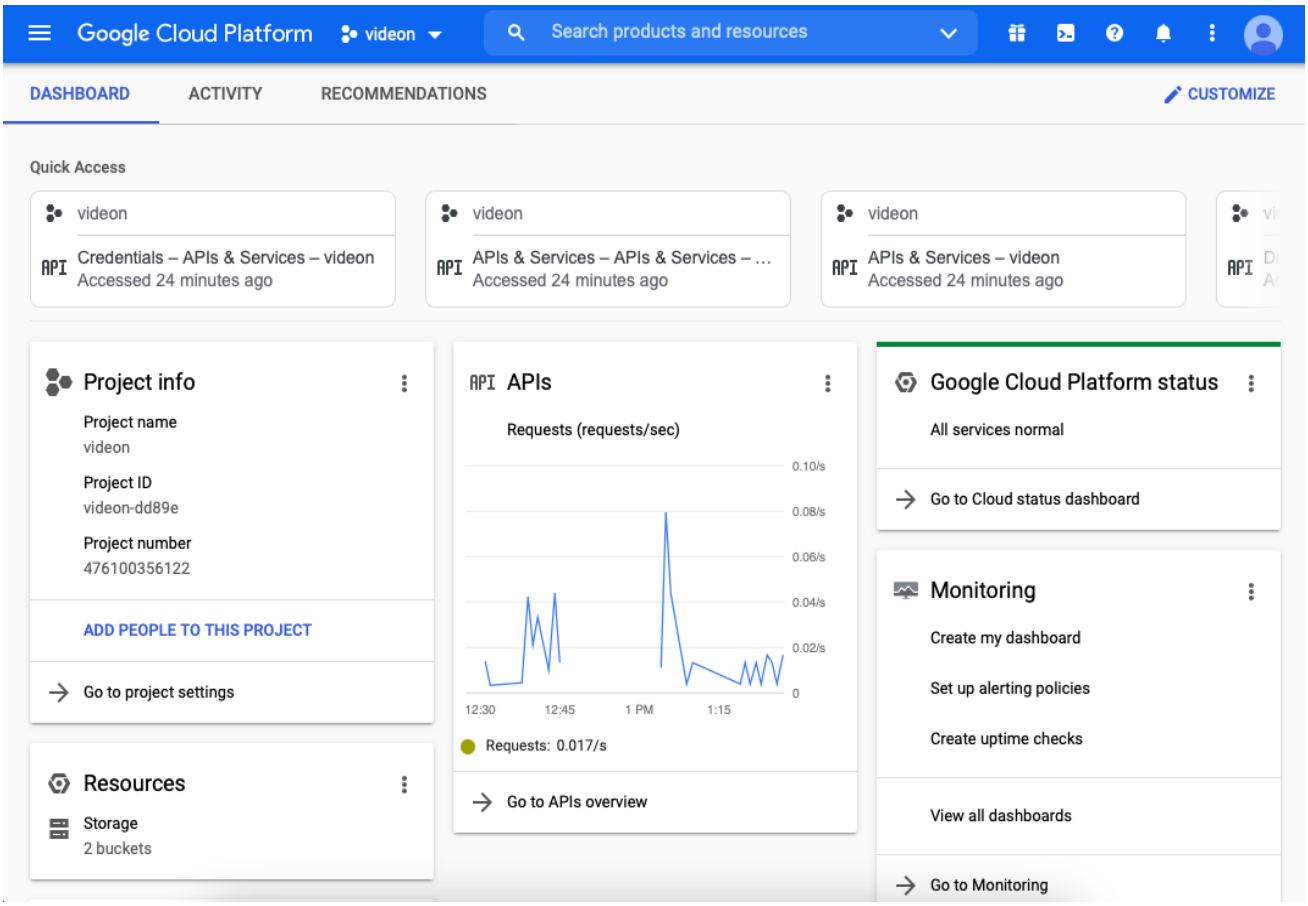

Рис. 2.2. Інтерфейс *Google Cloud Platform*

2.3. Сервіси *FlutterFire*

FlutterFire – це набір плагінів *Flutter*, які дозволяють додаткам використовувати служби *Firebase*. На сьогодні випущено сім плагінів: *Authentication*, *Cloud Firestore*, *Cloud Functions*, *Cloud Messaging*, *Cloud Storage*, *Core*, *Crashlytics* [15]. Усі вони підтримуються у мобільній, настільній та веб-розробці. На рис.2.3 зображено додавання *Firebase* до *Android* додатку.

FlutterFire забезпечує підтримку роботи зі змінами в колекціях та документах у реальному часі.

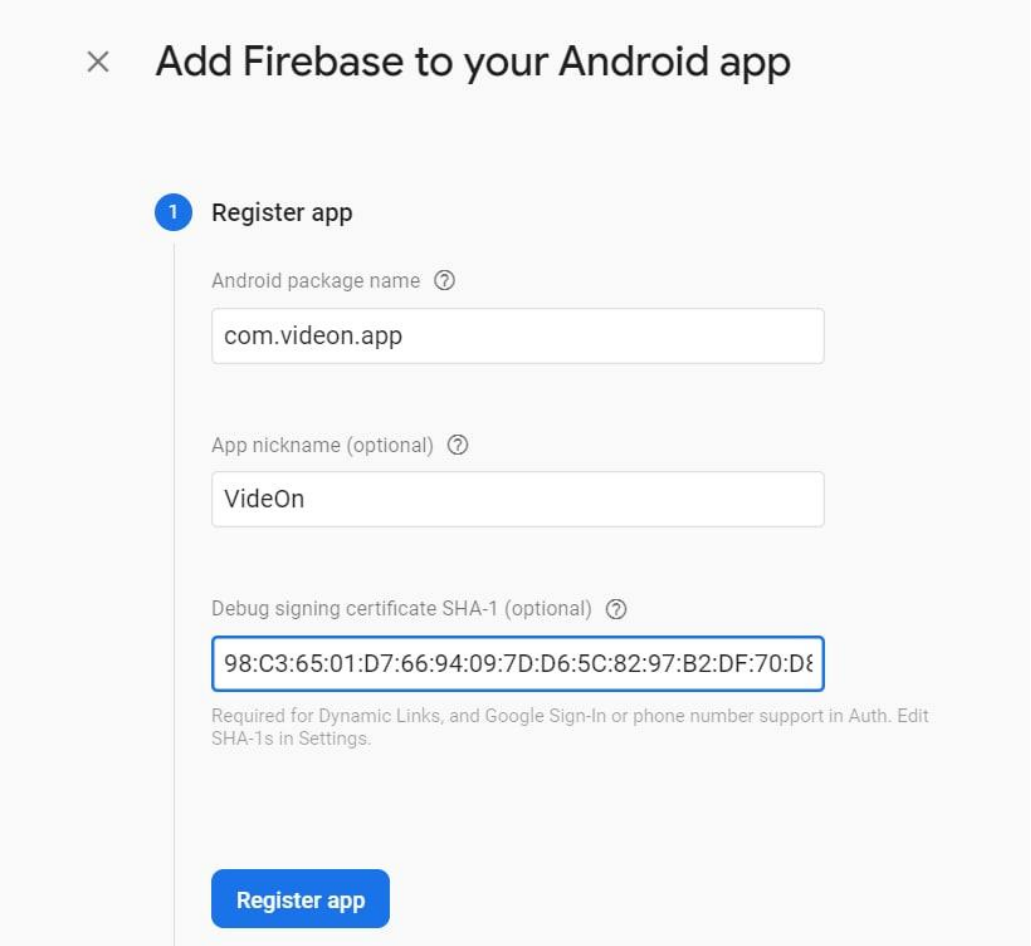

Рис. 2.3. Підключення *Firebase* у додаток

2.3.1. *Authentication*

Firebase Authentication надає серверні сервіси та прості у використанні *SDK* для аутентифікації користувачів у додатку. Аутентифікація підтримується за допомогою паролів, телефонних номерів, популярних постачальників об'єднаних ідентифікаційних даних, таких як *Google, Facebook* та *Twitter* тощо.

У додатку *VideOn* основний спосіб аутентифікації відбувається через електронну пошту та пароль, додатковими засобами є аутентифікація через *Google, AppleID, Facebook, Twitter*. На рис.2.4 показано усі доступні для *Firebase* способи аутентифікації. Ті, які підключені у додаток, позначені як *Enabled*.

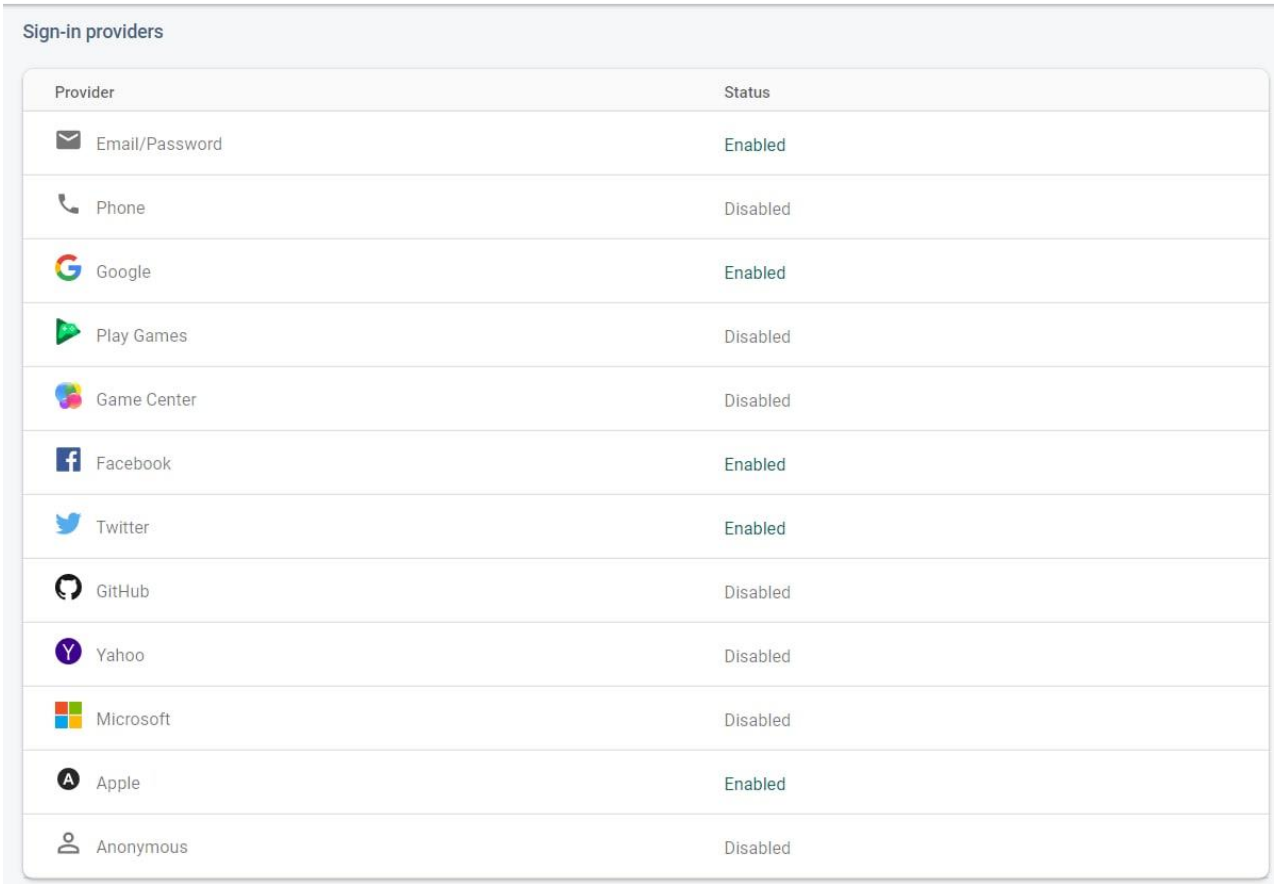

Рис. 2.4. Методи аутентифікації у *Firebase*

2.3.2. *Cloud Firestore*

Cloud Firestore – це розміщена у хмарі база даних *NoSQL*, до якої *iOS*, *Android* та веб-додатки можуть отримувати безпосередній доступ через власні *SDK*. Як і *Firebase Realtime Database*, вона підтримує синхронізацію даних між клієнтськими додатками в реальному часі і пропонує автономну підтримку для мобільних пристроїв і Інтернету, для створення адаптивних додатків, які працюють незалежно від затримки в мережі або підключення до Інтернету. *Cloud Firestore* також пропонує безшовну інтеграцію з іншими продуктами *Firebase* і *Google Cloud*, включаючи *Cloud Functions*.

Доступ до даних у *Cloud Firestore* захищається за допомогою аутентифікації *Firebase* та правил безпеки *Cloud Firestore* для *Android, iOS* та *JavaScript*.

Cloud Firestore проєкту має наступні колекції: *bug_report* (звіт про помилку), *popular* (популярні), *support* (підтримка), *users* (користувачі). На рис.2.5 наведено колекції *Cloud Firestore*.

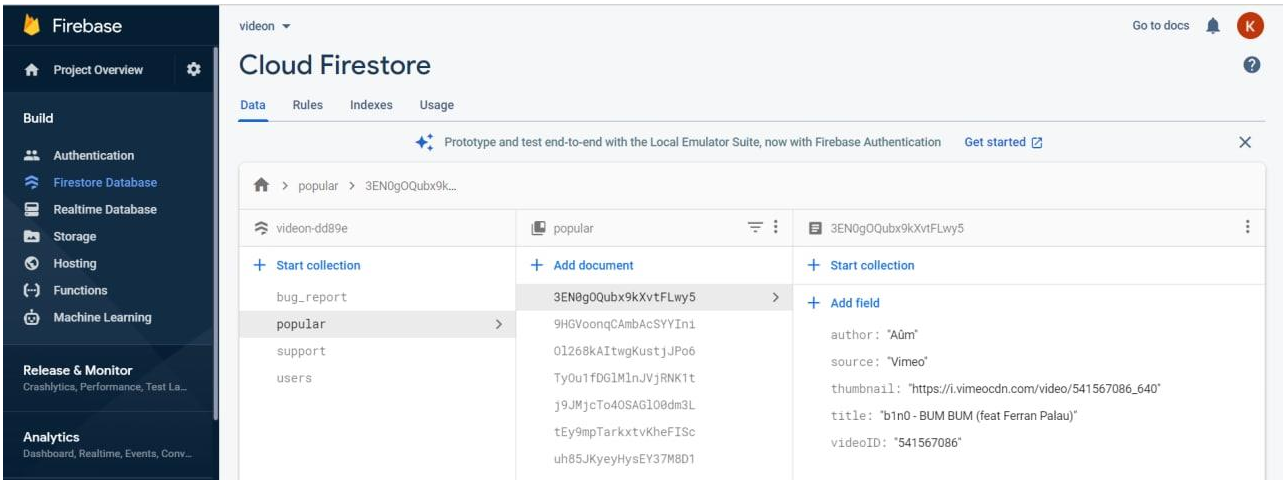

Рис. 2.5. Колекції *Cloud Firestore*

2.3.3. *Cloud Storage*

Cloud Storage для *Firebase* – це послуга зберігання об’єктів, створена для масштабу *Google*. Пакети *SDK Firebase* для хмарного сховища додають безпеку *Google* для завантаження файлів для програм *Firebase*, незалежно від якості мережі.

Cloud Storage для *Firebase* створено для розробників додатків, яким потрібно зберігати та обслуговувати вміст, створений користувачами.

2.4. Ключ інтерфейсу прикладного програмування

Ключ інтерфейсу прикладного програмування (*API key*) – це ідентифікатор, який використовується для аутентифікації користувача, розробника або програми, що викликає *API*. *API* також можна описати як спосіб спілкування різних програм. Для багатьох користувачів основна взаємодія з *API* здійснюватиметься за допомогою ключів *API*, які дозволяють іншим програмам отримувати доступ до облікового запису без введення пароля.

API – це частина сервера, яка приймає запити та надсилає відповіді. Основна ідея полягає в тому, щоб сервер додатку спілкувався безпосередньо із потрібним сервером із запитом створити подію із заданими деталями. Потім сервер додатку отримає відповідь, обробить її та відправить відповідну інформацію додатку, наприклад, повідомлення про успішну авторизацію. *API* повертає дані у форматі *JSON*.

Нерідкі випадки, коли команди розробників розбивають свої програми на кілька серверів, які спілкуються між собою за допомогою *API*. Сервери, які виконують допоміжні функції для основного сервера додатків, зазвичай називають мікросервісами.

Різні платформи можуть реалізовувати та використовувати ключі *API* по-різному.

2.4.1. *YouTube API*

Google Cloud Platform надає *API* сервісів *Google*. Із запропонованих було обрано *YouTube Data API v3*. Підключення *API* ключа сервісу *YouTube* наведено на рис. 2.6.

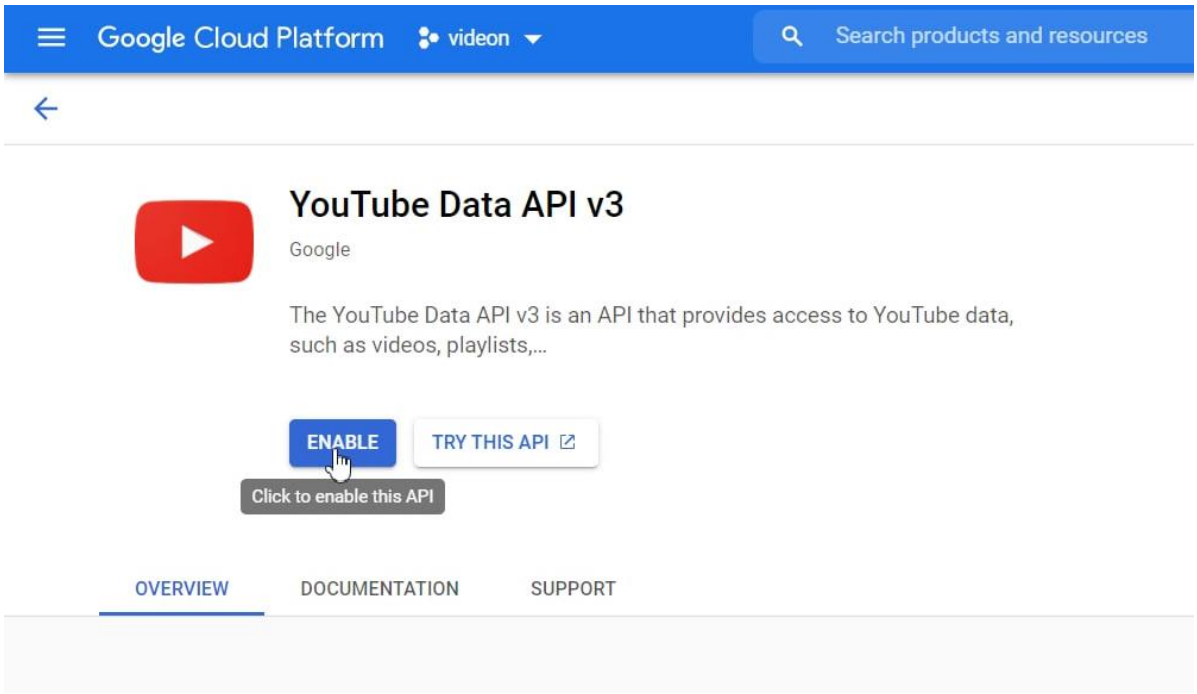

Рис. 2.6. Підключення *YouTube API*

API Data YouTube v3 – це API, який забезпечує доступ до даних *YouTube* за допомогою *REST API*. *API Data YouTube v3* дозволяє включити функціональність *YouTube* у власний додаток. Можна використовувати *API* для отримання результатів пошуку, оновлення та видалення таких ресурсів, як відео чи списки відтворення.

REST – це архітектурний стиль або філософія дизайну для взаємодії з Інтернет-ресурсами (наприклад, відео) за допомогою стандартних методів *HTTP*, включаючи *GET*, *POST* і *PATCH*.

2.4.2. *Twitch API*

API Twitch надає інструменти для розробки інтеграцій з *Twitch*. Для здійснення викликів через *API* потрібен ідентифікатор клієнта. Щоб отримати його, треба перейти на консоль розробника *Twitch* та зареєструвати свій додаток (рис. 2.7).

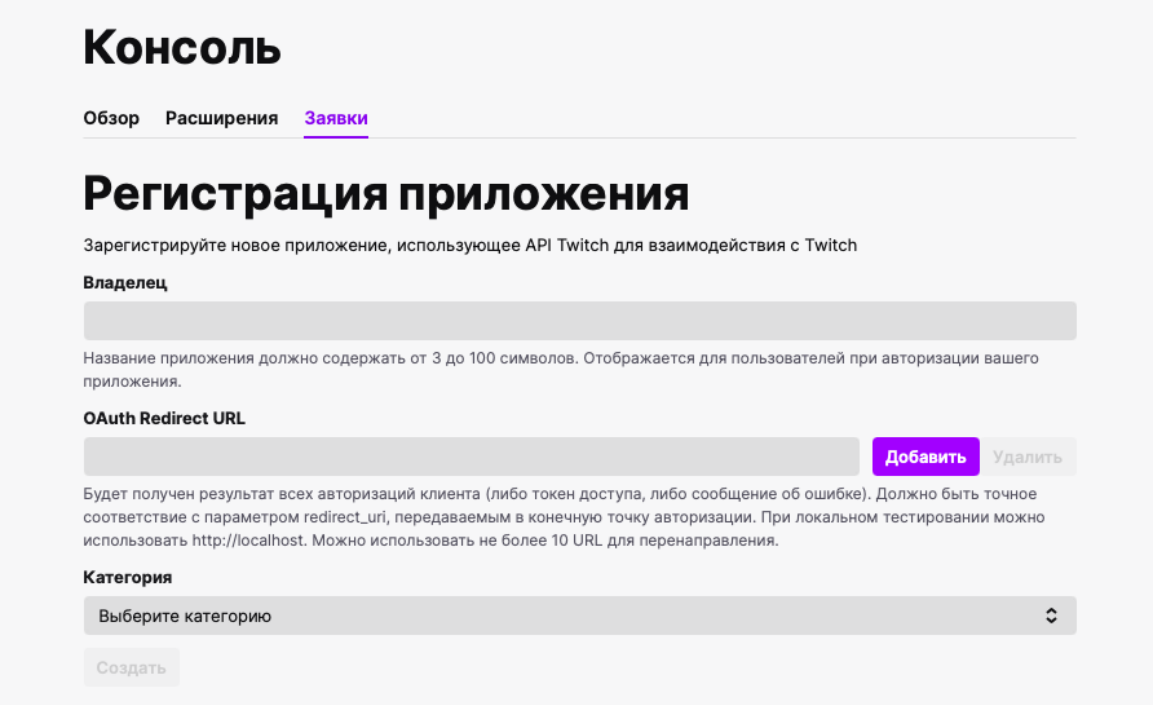

Рис. 2.7. Реєстрація додатку у *Twitch*

Після цього *Twitch* надає ідентифікатор клієнта, який є унікальним для кожного додатку і використовується для створення єдиного облікового запису для всіх користувачів *VideOn*. Це дозволяє отримувати усі можливості платформи *Twitch* без додаткової авторизації в аккаунт користувача.

2.4.4. *Vimeo API*

Для отримання *Vimeo API* потрібно зареєструвати свій додаток. Додаток у цьому сенсі може бути повнофункціональним мобільним додатком, динамічною веб-сторінкою або трирядковим сценарієм. Далі потрібно налаштувати середовище розробки для підключення та взаємодії з *Vimeo API*.

Маркер доступу дозволяє програмі робити запити *API*. Сам маркер – це рядок символів, які представляються в *API* як пароль. Але він також виконує інші функції, він ідентифікує додаток залежно від його типу та способу визначення його властивостей, та визначає діапазон допустимих дій або обсягів, які може виконувати додаток.

2.5. Висновки до розділу

У даному розділі розкрито сутність архітектури програмування *MVC,* визначено та налаштовано основні інструменти, які використовуються для подальшої розробки додатку.

У якості бази даних було обрано хмарну БД *Cloud Firestore*, яка є одним із плагінів сервісу *Firestore*. *Firestore* має велику кількість бібліотек для мобільних пристроїв. За допомогою бібліотек *Firestore* можна отримати усі переваги пакету *Firebase* – аутентифікацію, зберігання даних, та інші. Прості дані можна зберігати в документах, які подібних до *JSON*, складні ієрархічні впорядковуються у підколекції, що є зручним при великих масштабах баз даних. Використання хмарних баз даних дозволяє робити додаток гнучким, а розробку швидшою та економічно вигіднішою.

Для аутентифікації та авторизації у додатку використовується плагін *Firebase Authentication*, який має вбудований оброблювач помилок. Також було підключено альтернативні методи входу: *Google, AppleID, Facebook, Twitter*.

З огляду на те, що проєкт носить інформативну, а не комерційну направленість, було прийняте рішення використовувати безкоштовні *API* для отриманні відео із потокових сервісів таких як *YouTube, Twitch* та *Vimeo*.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО МОДУЛЮ ДЛЯ ОНЛАЙН ПЕРЕГЛЯДУ ВІДЕО НА МОБІЛЬНОМУ ПРИСТРОЇ

В даному розділі буде розглянуто створення додатку *VideOn* за допомогою використання фреймворку *Flutter* та мови програмування *Dart*. Буде детально описано створення бази даних додатку, підключення *API* та розробку основного функціоналу. Вищезазначені аспекти будуть розглянуті у відповідних підрозділах.

Перш ніж розпочати розробку додатку, необхідно встановити вимоги до проєкту, описати план розробки, створити дизайн-макет, підключити БД та провести інтеграцію з *API*.

3.1. Опис середовища програмування

При написанні додатку використовувалось інтегроване середовище розробки *IntelliJ IDEA* та віртуальний пристрій *Android Virtual Device*.

IntelliJ IDEA – це інтегроване середовище розробки від компанії *JetBrains* для мов JVM. Середовище надає інструменти для продуктивної роботи і підходить для створення комерційних, мобільних і веб-додатків. *IntelliJ IDEA* доступне у двох виданнях: *Community Edition*, яке ліцензоване *Apache2.0*, та комерційне видання, відоме як *Ultimate Edition*. Обидва вони можуть бути використані для створення комерційного програмного забезпечення. Простота використання, гнучкість та дизайн робить *IntelliJ IDEA* відмінним від аналогів.

Віртуальний пристрій *Android (AVD)* – це конфігурація, яка визначає характеристики *Android*-телефону, планшета, *Wear OS, Android TV* або автомобільної ОС, яке імітується в емуляторі *Android*.

3.2. Вимоги до додатку

Вимоги до реалізації додатку наступні:

– використання мови *Dart* у поєднанні із фреймворком *Flutter*;

– використання бази даних *Firestore* для зберігання інформації;

– висока швидкодія;

– використання архітектури *MVC*;

Розділи додатку повинні бути реалізовані наступним чином:

– *Watch Now* (дивитись зараз) , яка агрегує відео з різних джерел (основний розділ додатку). Тут представлені розділи *Recommendations* (рекомендації), *Continue to watch* (продовжити перегляд) та стрічка із популярними відео із сервісів *YouTube*, *Twitch*, *Vimeo*.

– *Browse* (перегляд) – перехід безпосередньо до сервісу.

– *Library* (бібліотека) – відображаються улюблені відео (сторінка *Favorites*) та збережені на пристрій відео (сторінка *Saved*).

– *Profile* (профіль) – налаштування профілю користувача. Реалізація можливостей встановлення фото профілю, зміна паролю, перегляд політики конфіденційності, налаштування повідомлень, перегляд інформації про додаток.

– *Feedback* (зворотній зв’язок) – технічна підтримка у вигляді чату у реальному часі та можливість надсилання повідомлення про помилку на електронну пошту додатку.

У додатку повинна бути реалізована авторизація та аутентифікація за допомогою електронної пошти та опціонально за допомогою сервісів *Google, AppleID, Facebook* та *Twitter*.

Системі вимоги наступні:

– Підтримка смартфонів *iPhone* з технологією *3G* або вище та версію ОС вище *iOS* 8.0 та *Android* версії 4.2 і вище.

Вимоги до дизайну:

– додаток повинен бути адаптивним та кросплатформеним;

– ергономічний дизайн;

3.3. Планування розробки додатку

Створення додатку включає в себе наступні етапи:

1) розробити дизайн, підібрати кольори, типографіку, іконки;

2) створити та налаштувати проєкт, перенести усі стилі;

3) зареєструвати *Cloud Firestore* та підключити її до проєкту;

4) розробити екрани реєстрації та входу, написати *backend;*

5) розробити екрани *Watch*, *Browse*, *Library*, *Profile*;

6) провести інтеграцію з *API*.

3.3.1. Розробка дизайну додатку

Зручний та зрозумілий дизайн надає додатку перевагу серед конкурентів. Так як додаток створений для перегляду відео, було прийняте рішення зробити мінімалістичний інтерфейс у кольорах, запропонованих *Apple Design Guidelines* та зробити основний акцент на відео. Дизайн та шаблон було розроблено за допомогою програми *Sketch*. Макети сторінок наведено на рис. 3.1.

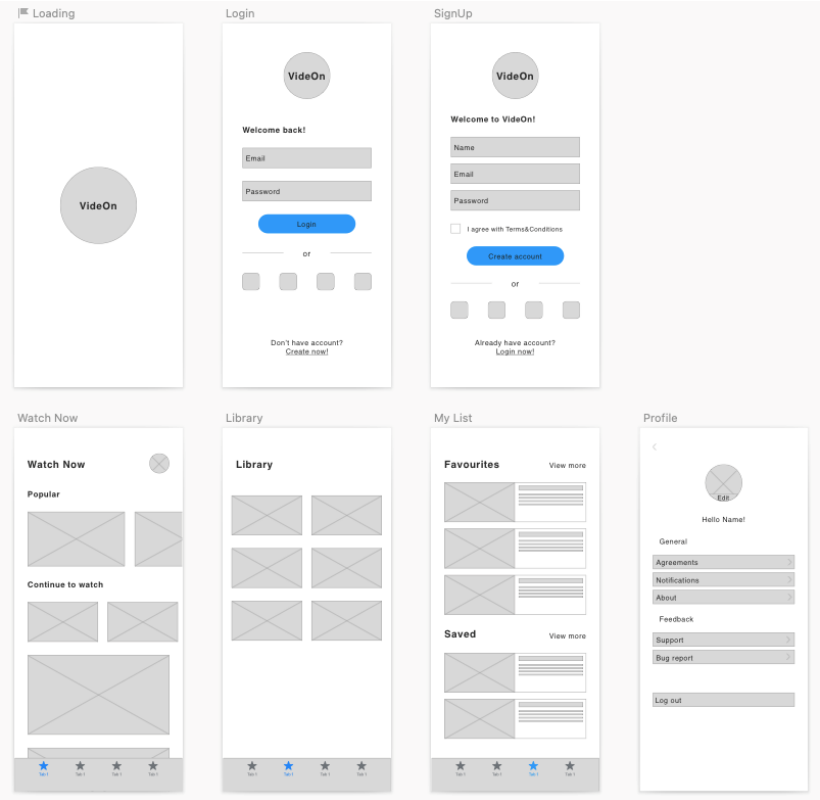

Рис. 3.1. Дизайн-макет додатку

3.3.2. Створення та налаштування проєкту, перенесення стилів

Перше, що потрібно зробити – це створити проєкт. Це можна зробити наступним чином:

– запустимо *IDE IntelliJ IDEA*;

– створимо проєкт *VideOn*;

– створимо папку *global* та папку *appstyle*, яка містить два файли *colors.dart* та *fonts.dart* ;

– у файл *colors.dart* додати наступний код:

```
Color charcoalGrey(){
  return Color(0xff303336);}
Color sapphire(){
  return Color(0xff0091ff);}
Color graphite(){
  return Color(0xff6d7278);}
Color obsidian(){
  return Color(0xff000000);}
Color calcite(){
  return Color(0xffffffff);
}
```

– у файл *fonts.dart* написати код для шрифтів додатку. Нижче наведено фрагмент коду для шрифту-заголовку *titleFont* :

```
TextStyle titleFont() {
  return TextStyle(
      letterSpacing: 2.4,
      color: Color.fromARGB(255, 0, 145, 255),
      fontSize: 44,
      fontFamily: 'Noteworthy-Lt',
    );
}
```

3.3.2. Створення та налаштування проєкту, перенесення стилів

Cloud Firestore – це найновіша *NoSQL* база даних *Firebase* для розробки мобільних додатків, яка дозволяє легко зберігати, синхронізувати та запитувати дані. *Cloud Firestore* базується на успіхах бази даних у реальному часі за допомогою нової, більш інтуїтивної моделі даних.

Для створення БД у *Cloud Firestore* необхідно створити проєкт у *Firebase* та створити нову *Firesore Database* (рис. 3.2).

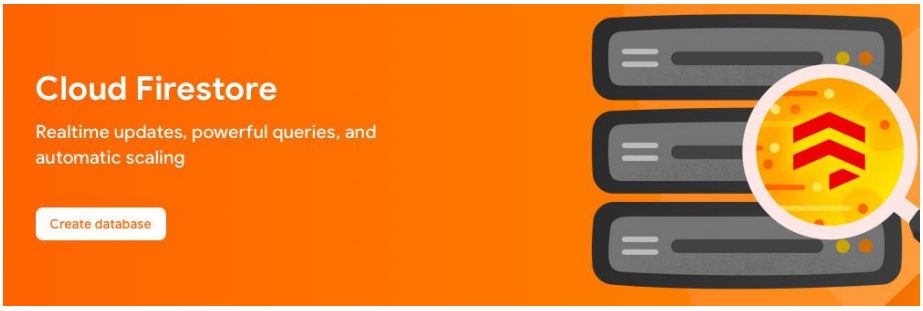

Рис. 3.2. Створення *Cloud Firestore*

Після успішного створення, необхідно створити колекцію, додати до неї документи, у документах додати необхідні поля. Заповнена база даних наведена на рис. 3.3. На цьому створення бази даних для додатку завершено.

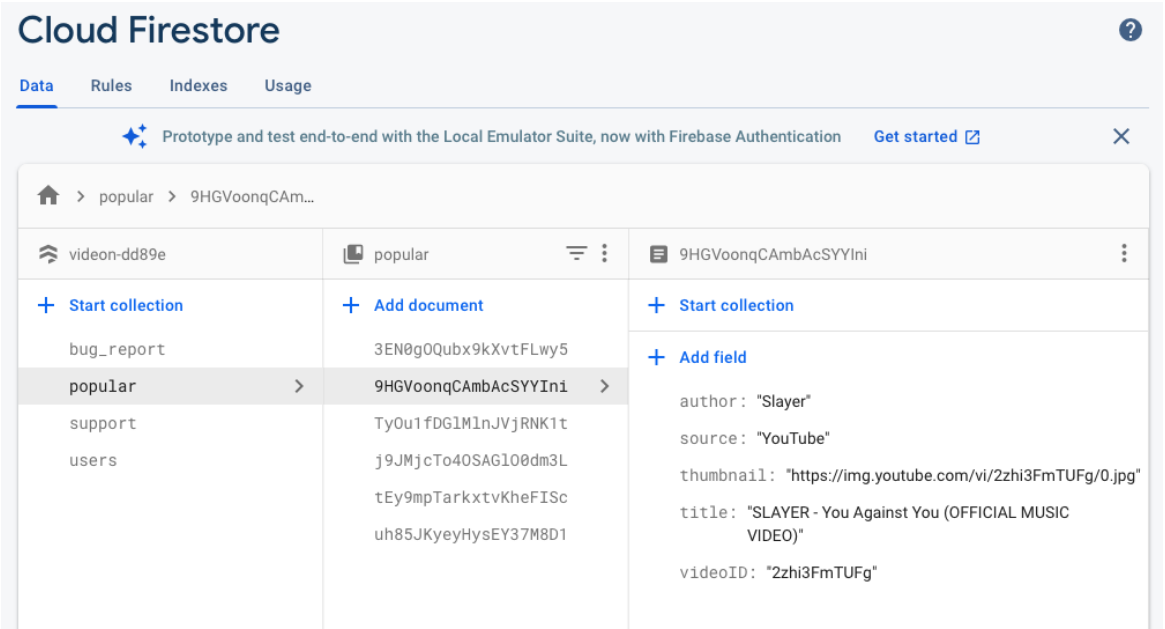

Рис. 3.3. Створена база даних

Для підключення необхідно завантажити файл та перемістити його до зазначеної директорії (рис. 3.4):

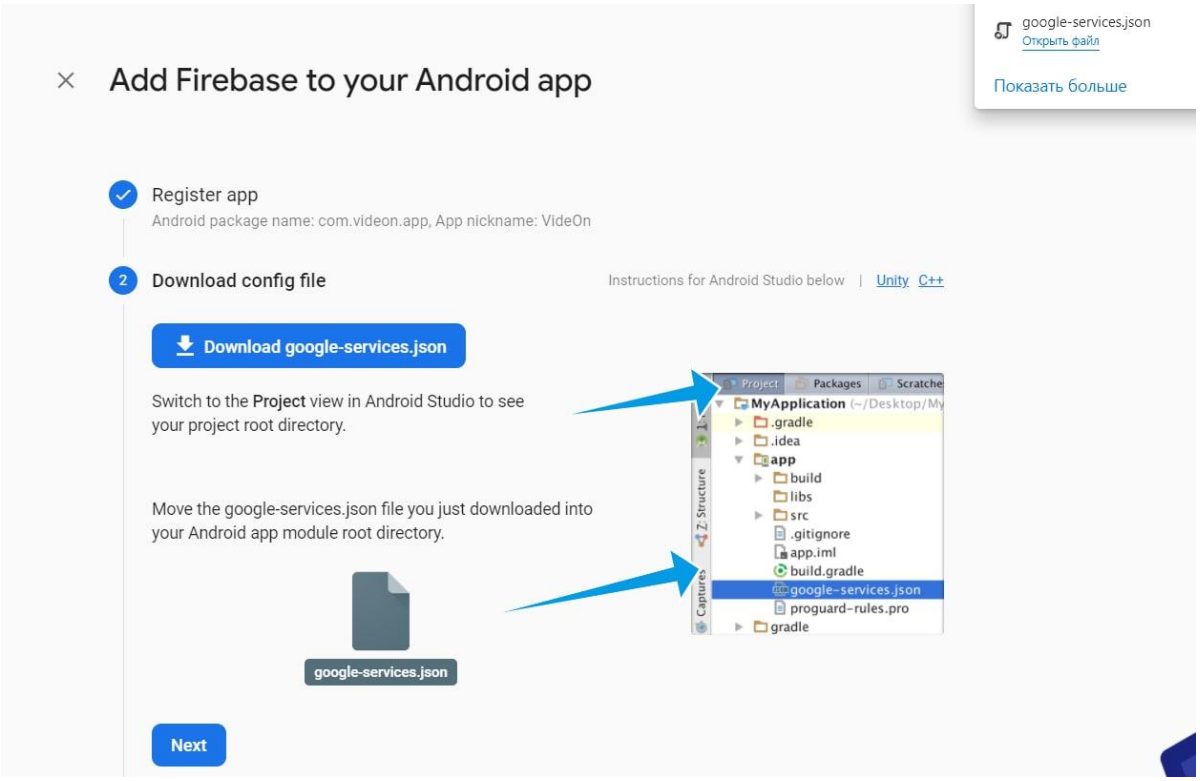

Рис. 3.4. Додавання файлу для взаємодією із *Firebase* до проєкту

У файлі *main.dart* треба замінити початкову функцію *main()* наступним кодом:

```
void main() async {
WidgetsFlutterBinding.ensureInitialized();
await Firebase.initializeApp();
runApp(MyApp());
}
```

3.3.4. Розробка екранів реєстрації та входу, *backend* функції

Спочатку необхідно додати бібліотеки проєкту. Основні бібліотеки та їх опис наведено у табл. 3.1.

Таблиця 3.1

Бібліотеки проекту

Назва бібліотеки	Опис
firebase_storage	Бібліотека для роботи із *Cloud Firestorage*
flutter_markdown	Бібліотека розмітки
flutter_webview_plugin	Бібліотека для перегляду *HTML* сторінок у додатку
material.dart	Бібліотека елементів інтерфейсу *Android*
path_provider	Бібліотека для роботи із локальними директоріями
provider	Бібліотека для надання глобального доступу
splashscreen	Бібліотека для завантаження екрану-заставки
widgets.dart	Бібліотека для підключення віджетів
youtube_api	Бібліотека для обробки даних з *YouTube API*

Тепер коли до проєкту підключено базу даних, можна розробити екрани входу та реєстрації. Для цього необхідно створити у проєкті нову папку *screens*, в середині неї папку *login_screen* та додати до неї два файли: *login_screen.dart* та *register_screen.dart*.

У файлі *login_screen.dart* створимо *StatefulWidget* та визначимо метод *build*. Для того, щоб додаток на різних пристроях виглядав однаково, визначимо коефіцієнти стиснення, які описані у наступному коді:

```
double _height = MediaQuery.of(context).size.height;
double _width = MediaQuery.of(context).size.width;
double coefH = _height / 896;
double coefW = _width / 414;
```

Метод *return* повертає основний віджет, у нашому випадку це *Scaffold*, який містить параметр *resizeToAvoidBottomInset*. Цей параметр відповідає за зсув контенту при відкритті клавіатури:

resizeToAvoidBottomInset: false

Віджет вертикальної компоновки *Column* містить усі необхідні контейнери. Нижче описаний для полів введення електронної пошти та паролю:

```
Container(
        padding: EdgeInsets.symmetric(horizontal: 48*coefH),
        width: double.maxFinite,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text("Welcome back!",style: headlineFont(),),
            SizedBox(height: 48*coefH),
            CustomInput(hint:"Email", controller: _email, obscure: false,
            height: 44*coefH, width: double.maxFinite, textInputType: TextInputType.emailAddress,),
            SizedBox(height: 24*coefH),
            CustomInput(hint:"Password", controller: _password, obscure: true,
             height: 44*coefH, width: double.maxFinite, textInputType: TextInputType.visiblePassword,),],),),
```

Після створення усіх необхідних контейнерів, необхідно створити клас *LoginWithEmail*, який містить у собі функцію *Login()* та *notifyAboutUser()*:

```
class LoginWithEmail{
  Future<void> login(String email, String password, BuildContext context) async{
    try {
      UserCredential userCredential = await FirebaseAuth.instance.signInWithEmailAndPassword(
email: email,
password: password);
```

if(userCredential.user != null){

print(userCredential.user.uid);

notifyAboutUser();

return Navigator.push(context, MaterialPageRoute(builder: (context)=> WatchNowScreen()));}} on FirebaseAuthException catch (e) {

if (e.code == 'user-not-found') {

print('No user found for that email.');

showError("No user found for that email.", context);

} else if (e.code == 'wrong-password') {

print('Wrong password provided for that user.');

showError("Wrong password provided for that user.", context); }}}

Stream<ConcreteUser> notifyAboutUser(){

return FirebaseAuth.instance.authStateChanges()

.map((User user) => user !=null ? ConcreteUser.fromFirebase(user) :null);}}

Метод *return* повертає основний віджет, у нашому випадку це *Scaffold*, який містить параметр *resizeToAvoidBottomInset*. Цей параметр відповідає за зсув контенту при відкритті клавіатури.

Функція *Login()* містить у собі блок try catch.

У тілі *try* викликається запит *signInWithEmailAndPassword* до *Firebase Auth*. У разі успішного отримання даних про користувача, викликається функція *notifyAboutUser()* та відбувається перехід на екран *WatchNowScreen*.

У тілі *catch* відслідковуються коди помилок. Наприклад, *e.code == 'user-not-found'* та виводиться відповідна помилка функцією *showError*.

Екран *RegisterScreen* працює таким самим чином, однак оброблюються інші помилки та викликається метод *createUserWithEmailAndPassword*:

Future<void> createUserInDB(String id, String name, String email, String password, BuildContext context) async{

CollectionReference _users = FirebaseFirestore.instance.collection('users');

return _users

doc(id)

.set({

'name' : name,

'email' : email,

'password' : password,

'profilePicture' : "null",})

.then((value) {

print("user added");

return Navigator.push(context, MaterialPageRoute(builder: (context)=> WatchNowScreen()));})

.catchError((error) => print("Failed to add user: $error")); }

На рис. 3.5 наведено інтерфейс екрану входу.

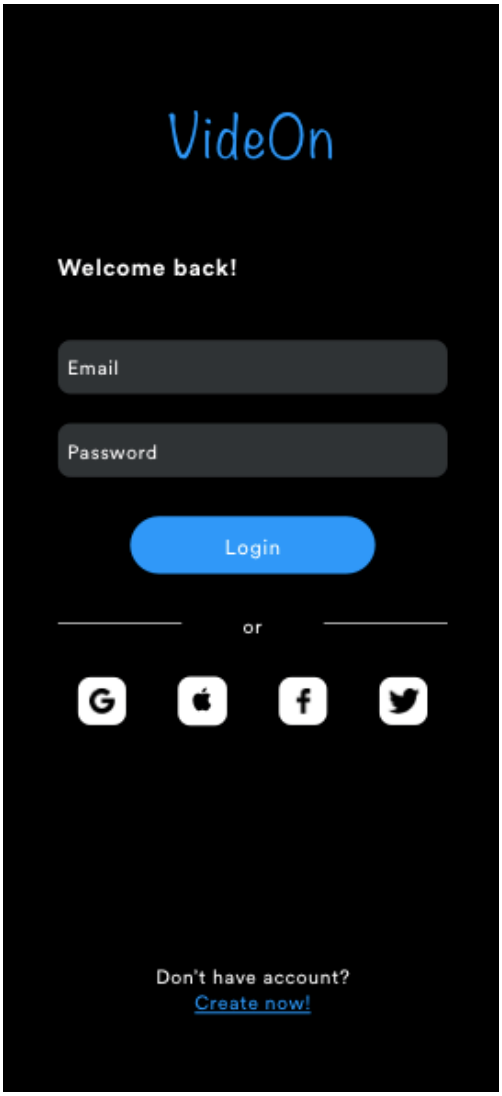

Рис. 3.5. Екран входу у додаток

3.3.5. Розробка екранів *Watch*, *Browse*, *Library* та *Profile*

Екрани *Watch*, *Browse* та *Library* мають однаковий шаблон: назва екрану у верхньому лівому куті та панель вкладок знизу. Основний вміст сторінки заповнюється відповідно до потреби. Панель вкладок реалізується як *bottomNavigationBar* у віджеті *Scaffold*. Наприклад, для відображення джерел відео, використовується віджет *GridView.count*, код якого наведено нижче:

```
GridView.count(
        childAspectRatio: 1.5,
        padding: EdgeInsets.only(left: 21),
        shrinkWrap: true,
        crossAxisCount: 2,
        children: [
          GestureDetector(
            onTap: ()=> Navigator.push(context, MaterialPageRoute(builder: (context)=> YouTubeScreen()
    )
    ),
            child: SourceCard(source: 'assets/images/YT_Logo.png', color: Colors.white,)),
          GestureDetector(
            onTap: ()=> Navigator.push(context, MaterialPageRoute(builder: (context)=> TwitchScreen()
    )
    ),
            child: SourceCard(source: 'assets/images/TW_Logo.png', color: Color.fromARGB(255, 121, 41, 235)
    ))
    )
    ]
    )
```

На рис. 3.6 наведено головний екран додатку *Watch Now*

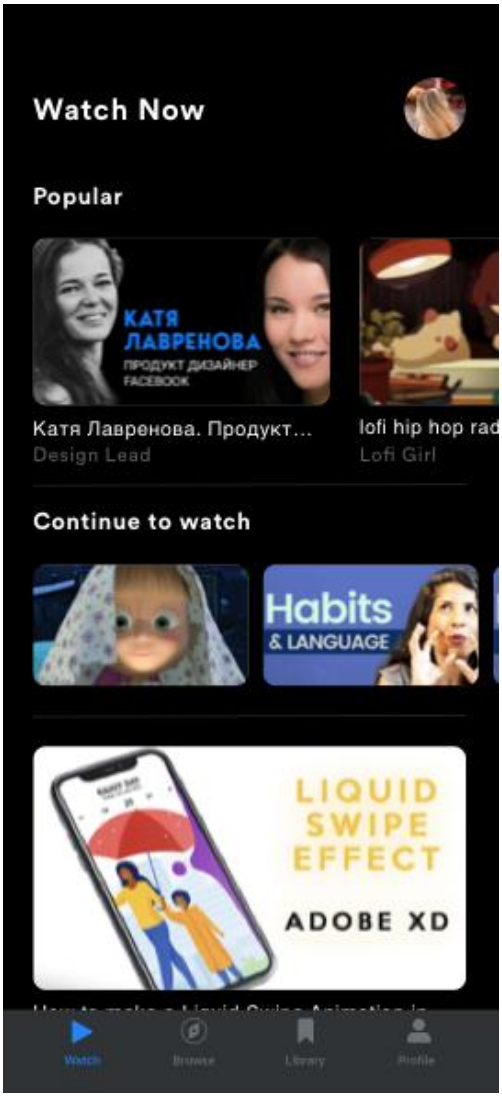

Рис. 3.6. Екран *Watch Now*

Екран *Profile* надає змогу користувачу змінити фото профіля, змінити пароль, переглянути політику конфіденційності додатку, звернутися до технічної підтримки у чаті або через електронну пошту. Весь функціонал профіля реалізується наступним чином: створюється контейнер, у тіло якого передається назва екрану, на який буде відбуватись перехід; описується навігація на екран, що відповідає за конкретну функцію.

Реалізація роботи екрану політики конфіденційності описана наступним кодом:

```
GestureDetector(
        onTap: ()=> Navigator.push(context, MaterialPageRoute(builder: (context)=> PrivacyPolicyScreen())),
        child: Container(
         padding: EdgeInsets.symmetric(horizontal: 8),height: 44*coefH,
         decoration: BoxDecoration(borderRadius: BorderRadius.circular(10), color: charcoalGrey()),
        child: Row(
          mainAxisAlignment:MainAxisAlignment.spaceBetween,
          children: [
           Text("Privacy Policy",style: bodyFont(),),
           Image.asset('assets/images/next.png')
          ],),),
    ),
```

3.3.6. Інтеграція *API*

У середовищі розробки *IntelliJ IDEA* було створено проєкт додатку та проведено базові налаштування, після чого створено файл *keys.dart*, який містить дані для авторизації додатку у відео сервісах, що використовуються.

У файлі, який відповідає за стрічку відео на сторінці *Watch Now* описуємо дві змінні:

```
List<Video> _videos = [];
Future<List<Video>> _dataModel;
```

Зміна *_videos* відповідає за список об’єктів типу *Video*, змінна *_dataModel* – змінна, яка очікує результат у майбутньому в форматі *List<Video>*.

Функція *getYTVideos()* працює із таким самим типом даних, як *_dataModel* та являється асинхронною, що є необхідною умовою *REST API* запитів. *http.get()* – функція, яка працює із *googleapis.com* та повертає *JSON* дані.

```
Future<List<Video>> getYTVideos() async {
```

var data = await http.get("https://youtube.googleapis.com/youtube/v3/videos?chart=mostPopular&key=$ApiKey&part=snippet");

Далі відбувається декодування *JSON* даних за допомогою функції *json.decode()*.

var jsonData = json.decode(data.body);

Далі використовується цикл для обходу всіх об’єктів в середині блоку *items*.

for (var u in jsonData['items']) {

Наступними рядками створюємо об’єкт типу *Item* та передаємо в його конструктор змінну *u*.

Item item = Item.fromJson(u);

Після цього у список відео додається об’єкт типу *Video* з джерела *YouTube*.

_videos.add(Video.fromYT(item));

}

print(_videos.length);

return _videos;

}

Аналогічними чином відбувається взаємодія із *api.twitch.tv* та *api.vimeo.com*.

Функція *initState()* – вбудований механізм фреймворку *Flutter*, який автоматично спрацьовує при відкритті екрану. Викликає заповнення списку *_dataModel* з різних джерел. Нижче наведено код функції *initState()*:

```
@override
 void initState() {
  _dataModel = getYTVideos();
  _dataModel = getTWVideos();
  _dataModel = getVMVideos();
  super.initState();
 }
```

3.4. Висновки до розділу

Даний розділ присвячено програмній реалізації проєкту. Окремо виділені питання створення бази даних додатку, підключення *API* та розробку основного функціоналу.

У даному розділі також були описані розроблені екрани додатку, наведені програмні коди їх реалізації.

Усі елементи вікон додатку були розроблені на мові програмування *Dart*, для підключення даних *API* використовувались відповідні бібліотеки.

За допомогою векторного редактору *Sketch* було розроблено дизайн-макет та повноцінний дизайн додатку. У якості основних кольорів обрано кольорову схему, яка запропонована *Apple Design Guidelines*.

У середовищі розробки *IntelliJ IDEA* було створено проект та перенесено створені стилі. Це дозволяє швидко та зручно використовувати їх у проекті.

База даних проекту була створена за допомогою плагіну *Cloud Firestore*. *Cloud Firestore* – це *NoSQL* база даних *Firebase* для розробки мобільних додатків, яка дозволяє легко зберігати, синхронізувати та запитувати дані. Підключення БД до проекту відбувається за допомогою додавання *JSON* файлу у відповідну директорію проекту.

Екрани додатку *Watch, Browse* та *Library* мають єдиний шаблон. Це дозволяє користувачу легше орієнтуватися у додатку. Усі функціональні елементи виділено блакитним кольором, інші – сірим. Екрани реалізовані віджетом *Scaffold*, який дозволяє розміщувати *uppBar, body* та *bodyNavigationBar*.

Екран *Profile* надає інформацію про аккаунт користувача, а також надає можливість базового налаштування профілю та технічну підтримку у чаті.

Інтеграція *API* дозволила отримати відео із трьох різних джерел – *YouTube, Twitch, Vimeo*. Це досягається шляхом отримання *API keys* кожного сервісу та конвертації *JSON* даних у потрібний формат.

Після роботи над розробкою додатку зроблено висновок, що фреймворк Flutter було обрано дуже вдало, оскільки додаток має інтерфейс та анімацію схожу на нативні додатки *Android* та *iOS*, а розробка була швидкою і зручною.

ВИСНОВКИ

Дипломний проект присвячений темі “Програмний модуль для онлайн перегляду потокового відео на мобільному пристрої”. Актуальність даної тематики обумовлена щорічним ростом кількості користувачів інтернетом, що в свою чергу приводить до зростання кількості користувачів смартфонів та інтернет-ресурсів. Найчастіше смартфони використовують в розважальних цілях, зокрема для перегляду відеоконтенту онлайн.

У дипломному проєкті було створено програмний модуль, який дозволяє переглядати потокове відео нлайн на мобільному пристрої. Для вирішення даної задачі було :

1) проаналізовано існуючі технології для розробки кросплатформених мобільних додатків;

2) описано технології налаштування та використання плагінів сервісу Firebase;

3) на основі *Cloud Firestore* розроблено базу даних додатку для збереження даних про користувача;

4) реалізовано програмний модуль мовою програмування *Dart*, яка дозволяє створювати кросплатформені додатки.

Після огляду існуючих рішень виділено три основні способи розробки кросплатформених додатків: нативна розробка, кросплатформена і гібридне рішення, що комбінує обидва підходи. Аналіз показує, що нативні додатки перевершують за працездатністю кросплатформені і гібридні типи розробок, проте кросплатформені додатки набувають все більшої популярності, оскільки вони дозволяють реалізувати додаток, який буде працювати на різних платформах і тому є економічно вигіднішими.

Було проведено аналіз мов програмування та фреймворків, як засобів створення мобільних додатків. Після їх детального огляду визначено, що більшість сучасних фреймворків активно використовується для розробки як настільних, так і мобільних додатків із різним функціоналом і рівнем складності.

У якості фреймворку для розробки додатку *VideOn* було обрано нову технологію *Flutter*, яку презентовано компанією *Google*. Даний фреймворк дозволяє створити зручні у користуванні мобільні додатки, які за своєю взаємодією із користувачем схожі на нативні додатки.

Підбиваючи підсумки, можна підкреслити наступні переваги даної програми:

– безкоштовність;

– відсутність реклами під час перегляду відео;

– відкритий код;

– агрегація відео із різних сервісів;

– малий розмір;

– єдиний аккаунт для усіх сервісів.

Під час виконання дипломного проекту були вирішені наступні поставлені завдання:

– розробка дизайну;

– реєстрація та підключення до проекту бази даних *Cloud Firestore*;

– розробка екранів реєстрації та входу;

– реалізація технічної підтримки ;

– інтеграція з *API* для відтворення потокового відео.

Результати дипломної роботи рекомендується використовувати для тестування потокових відеосервісів, та в навчальному процесі для відтворення навчальних відеоматеріалів з декількох джерел, шляхом агрегації їх з різних сервісів в один додаток.

Дана робота потребує подальшої доробки, а саме: додавання нових відеосервісів, повноцінна реалізація версії для *iOS*, інтеграція з вже існуючими акаунтами для персональних рекомендацій.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. правила оформлення. – 88 с.

2. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

3. *Global digital population as of January* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.statista.com/statistics/617136/digital-population-worldwide/*

4. *YouTube is Responsible for 37% of All Mobile Internet Traffic January* [Електронний ресурс] – Режим доступу до ресурсу: *https://www.statista.com/chart/17321/global-downstream-mobile-traffic-by-app/*

5. *Mobile Operating System Market Share Worldwide* [Електронний ресурс] – Режим доступу до ресурсу: *https://gs.statcounter.com/os-market-share/mobile/worldwide*

6. Рейтинг мов програмування 2021: частка *Python* зменшується, а *TypeScript* обійшов *C++* [Електронний ресурс] – Режим доступу до ресурсу: *https://dou.ua/lenta/articles/language-rating-jan-2021/*

7. *Cross-platform vs Native Mobile App Development: Choosing the Right Development Tools for Your Project* [Електронний ресурс] – Режим доступу до ресурсу: *https://medium.com/all-technology-feeds/cross-platform-vs-native-mobile-app-development-choosing-the-right-dev-tools-for-your-app-project-47d0abafee81*

8. *Excerpt From: Apple Inc. "The Swift Programming Language (Swift 5.4)". Apple Books. https://books.apple.com/ua/book/the-swift-programming-language-swift-5-4/id881256329,* 2014. – 500 с.

9. *Dart overview* [Електронний ресурс] – Режим доступу до ресурсу: *https://dart.dev/overview*

10. Шилдт Г. Java. Полное руководство, 10-е изд. : Пер. с англ. – СПб.: ООО "Диалектика", 2020. – 1488с. : ил. – Парал. тит. англ.

11. *Programming Languages Ranking: Top 10 for 2021* [Електронний ресурс] Режим доступу до ресурсу: *https://www.cleveroad.com/blog/programming-languages-ranking*

12. P*ros and Cons of Java | Advantages and Disadvantages of Java* [Електронний ресурс] – Режим доступу до ресурсу: *https://data-flair.training/blogs/pros-and-cons-of-java/*

13. Шарп Д. Microsoft Visual C#. Подробное руководство / Дж. Шарп. – СПб: Питер, 2017. – 848 с. – (8-е издание).

14. *PYPL PopularitY of Programming Language* [Електронний ресурс] – Режим доступу до ресурсу: *https://pypl.github.io/PYPL.html*

15. *Flutter documentations* [Електронний ресурс] – Режим доступу до ресурсу: *https://flutter.dev/docs*

16. Сравнение фреймворков для кроссплатформенной мобильной разработки: *React Native*, *Flutter*, *Ionic*, *Xamarin* и *PhoneGap* [Електронний ресурс] – Режим доступу до ресурсу: *https://tproger.ru/translations/cross-platform-frameworks-for-mobile-development/*

17. *Dart overview* [Електронний ресурс] – Режим доступу до ресурсу: *https://dart.dev/overview*

18. *How to upload data to Firebase Firestore Cloud Database* [Електронний ресурс] – Режим доступу до ресурсу: *https://medium.com/@devesu/how-to-upload-data-to-firebase-firestore-cloud-database-63543d7b34c5*

ДОДАТОК А

Лістинг коду модель даних відео

video.dart

```
using System;
using System.Data;
using System.Data.SqlClient;
using Microsoft.SqlServer.Server;
public partial class Triggers
{
   // Enter existing table or view for the target and uncomment the attribute line
   [Microsoft.SqlServer.Server.SqlTrigger (Name="trigger_ins_SQL1000", Target="SQL_1000", Event="FOR INSERT")]
   public static void Trigger_ins()
   {
      SqlTriggerContext triggContext = SqlContext.TriggerContext;
      SqlParameter vl = new SqlParameter("@value", System.Data.SqlDbType.NChar);
      if (triggContext.TriggerAction == TriggerAction.Insert)
      {

         using (SqlConnection conn = new SqlConnection("context connection=true"))
         {
            conn.Open();
            SqlCommand sqlComm = new SqlCommand();
            SqlPipe sqlP = SqlContext.Pipe;
            sqlComm.Connection = conn;
            sqlComm.CommandText = "SELECT value from INSERTED";
            vl.Value = sqlComm.ExecuteScalar().ToString();
            SqlContext.Pipe.Send(vl.Value.ToString());
            string s = ((string)vl.Value).ToString();
            char c = Convert.ToChar(32);
```

```
            s=s.TrimEnd(c);
            if (s.Equals("123"))
            {
               sqlComm.CommandText = "INSERT t3(val) VALUES('123')";
               sqlP.Send(sqlComm.CommandText);
               sqlP.ExecuteAndSend(sqlComm);

            }
         }
      }

   }
}
```

ДОДАТОК Б

Лістинг коду модуля для отримання відео з різних джерел та об’єднання їх в один список

feed.dart

```
Future<List<Video>> _dataModel;
List<Video> _videos = [];


 Future<List<Video>> getYTVideos() async {
  var data = await http.get(

"https://youtube.googleapis.com/youtube/v3/videos?chart=mostPopular&key=$ApiKey&part=snippet");
  var jsonData = json.decode(data.body);
  print(jsonData);
  for (var u in jsonData['items']) {
   Item item = Item.fromJson(u);
   _videos.add(Video.fromYT(item));
            }
            print(_videos.length);
            return _videos;
           }


           Future<List<Video>> getTWVideos() async {
            Map<String, String> headers = {
              'Accept': 'application/vnd.twitchtv.v5+json',
              'Client-ID': clientId,
            };
var data = await http.get(
     "https://api.twitch.tv/kraken/streams/?limit=5", headers: headers);
  var jsonData = json.decode(data.body);
```

```
    for (var u in jsonData['streams']) {
      Stream stream = Stream.fromJson(u);
      _videos.add(Video.fromTW(stream));
    }
    print(_videos.length);
    return _videos;
  }

  Future<List<Video>> getVMVideos() async {
    Map<String, String> headers = {
      'Authorization': 'bearer $access_token',
    };
    var data = await http.get(
        "https://api.vimeo.com/categories/travel/videos?per_page=5", headers:
headers);
    var jsonData = json.decode(data.body);

    for(var u in jsonData['data']) {
     Datum datum = Datum.fromJson(u);
      _videos.add(Video.fromVM(datum));
    }
    print(_videos.length);
    return _videos;
  }
  @override
  void initState() {
    _dataModel = getYTVideos();
    _dataModel = getTWVideos();
    _dataModel = getVMVideos();
    super.initState();
  }
```

ДОДАТОК В

Лістинг коду головного екрану програми *Watch Now*

watch_now_screen.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:vide_on/global/app_style/colors.dart';
import 'package:vide_on/global/app_style/fonts.dart';
import 'package:vide_on/global/custom_widgets/profile_pic_container.dart';
import 'package:vide_on/screens/landing_screen/browse_screen/browse_screen.dart';
import 'package:vide_on/screens/landing_screen/library_screen/library_screen.dart';
import 'package:vide_on/screens/landing_screen/profile_screen/profile_screen.dart';
import
'package:vide_on/screens/landing_screen/watch_now_screen/continue_watch.dart';
import 'package:vide_on/screens/landing_screen/watch_now_screen/feed.dart';
import
'package:vide_on/screens/landing_screen/watch_now_screen/recommendations.dart';

class WatchNowScreen extends StatefulWidget {
  @override
  _WatchNowState createState() => _WatchNowState();
}

class _WatchNowState extends State<WatchNowScreen> {
  @override
  Widget build(BuildContext context) {
    double _height = MediaQuery.of(context).size.height;
    double _width = MediaQuery.of(context).size.width;
    double coefH = _height / 896;
```

```
double coefW = _width / 414;

return Scaffold(
  body: ListView(
    controller: ScrollController(),
    children: [
      SizedBox(height: 40),
      Padding(
        padding: EdgeInsets.symmetric(horizontal: 32),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Text(
              "Watch Now",
              style: title_1Font(),
            ),
            ProfilePictureContainer(),
          ],
        ),
      ),
      Padding(
          padding: EdgeInsets.only(left: 32, top: 32, bottom: 24),
          child: Text(
            "Recommendations",
            style: headlineFont(),
          )),
      RecommendationsContainer(height: 190 * coefH),
      Padding(
          padding:
              EdgeInsets.symmetric(horizontal: 32 * coefW, vertical: 16),
```

```
            child: Container(
              height: 1,
              color: charcoalGrey(),
            )),
          Padding(
            padding: EdgeInsets.only(left: 32,
          bottom: 24),
            child: Text(
              "Continue to watch",
              style: headlineFont(),
            )),
          ContinueWatchContainer(height: 98 *
          coefH),
          Padding(
            padding:
                EdgeInsets.symmetric(horizontal:
          32 * coefW, vertical: 24),
            child: Container(
              height: 1,
              color: charcoalGrey(),
            )),
          FeedContainer(),
        ],
      ),

    );
  }
}
```