МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження методів створення сервісно-орієнтованих та крос-платформених додатків за допомогою Flutter з серверною частиною на Firebase

Виконав:
студент 2 курсу групи ІПЗм-21-2
Осташко Є. В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова

Керівник доц. Ревенчук І.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ З.В. Дудар
(прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Програмної Інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121-Інженерія програмного забезпечення
(код і повна назва)
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Осташко Євгену Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів створення сервісно-орієнтованих та крос-платформених додатків за допомогою Flutter з серверною частиною на Firebase»

затверджена наказом університету від «29» березня 2023 р. № 302Ст

2. Термін подання студентом роботи до екзаменаційної комісії 19 травня 2023 р.

3. Вихідні дані до роботи клієнт-сервісний, крос-платформений додаток, фреймвор Flutter, serverless технологія на Firebase, пояснювальна записка.

4. Перелік питань, що потрібно опрацювати в роботі вступ, аналіз предметної галузі, виявлення проблем, постановка задачі, формування не- та функціональних вимог до програмного забезпечення, архітектура та проєктування програмного продукту, висновки, перелік посилань.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	29.03.2023	виконано
2	Вивчення наявних аналогів та сучасних методів вирішення задачі	01.04.2023	виконано
3	Визначення проблеми та постановка задачі дослідження	02.04.2023	виконано
4	Розробка методології дослідження та опис його етапів	04.04.2023	виконано
5	Розробка сервісу та аналіз цього процесу	25.04.2023	виконано
6	Підготовка звіту до кваліфікаційної роботи	05.05.2023	виконано
7	Підготовка презентації та доповіді	08.05.2023	виконано
8	Попередній захист	10.05.2023	виконано
9	Занесення диплому в електронний архів	16.05.2023	виконано
10	Допуск до захисту у зав. кафедри	17.05.2023	виконано

Дата видачі завдання «____28____» ___лютого____ 2023 р.

Студент _____
(підпис)

Керівник роботи _____ ___доц. Ревенчук І.А__
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи магістра містить: 86 с., 15 рис., 21 джерел.

ІНФОРМАЦІЙНА СИСТЕМА, SERVERLESS, FIREBASE, FLUTTER, DART, BLoC, ЧИСТА АРХІТЕКТУРА, STRIPE.

Об’єктами дослідження є технологія побудови крос-платформних клієнтських застосунків Flutter та сервіс Firebase.

Метою роботи є дослідження методів створення крос-платформних сервісно-орієнтованих Flutter додатків з серверною частиною на Firebase.

Методи розробки базуються на таких технологіях, як Flutter, Dart, Typescript, Firebase.

В результаті роботи було отримано методи створення сервісно-орієнтованих та крос-платформених додатків за допомогою Flutter з серверною частиною на Firebase для мінімальних затрат на налаштування серверу та створено систему, яка складається з мобільних додатків для Android / iOS, веб додатку та серверної частини на Firebase, з якою спілкуються клієнтські додатки.

INFORMATION SYSTEM, SERVERLESS, FIREBASE, FLUTTER, DART, BLoC, CLEAN ARCHITECTURE, STRIPE.

The objects of the research are the technology to build cross-platform client applications that is Flutter and the Firebase service.

The method of work is to research the option of a more efficient development of client-server applications.

Development methods are based on such technologies as Flutter, Dart, Typescript, Firebase.

As a result of the work, methods of creating service-oriented and cross-platform applications using Flutter with a server part on Firebase were obtained for minimal costs for configuring the server, and a system was created, which consists of mobile

applications for Android / iOS, a web application and a server part on Firebase , with which client applications communicate.

Умови публікації пояснювальної записки

Я, Осташко Євген Васильович
(прізвище, ім’я, по батькові)

студент групи ІПЗм-21-2 здобувач вищої освіти на другому (магістерському) рівні

кафедра програмної інженерії,
(повна назва кафедри)

заявляю: моя кваліфікаційна робота на тему

Дослідження методів створення сервісно-орієнтованих та крос-платформених додатків за допомогою Flutter з серверною частиною на Firebase,
(назва роботи)

що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

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

ЗМІСТ

ВСТУП

У наш час найпопулярнішим видом програмних систем є клієнт-серверні, тому для розробки таких систем існує безліч технологій на багатьох мовах програмування різних типів. Кожен раз на початку розробку тієї чи іншої системи розробники обирають найбільш оптимальну технологію для їх конкретних потреб, зважаючи на рівень підготовки команди, попередній досвід, функціонал, який потрібно реалізувати та наданий замовником час реалізації.

З часом розробники все більше схиляються до крос-платформної розробки через велику кількість переваг такого підходу, основні з яких – необхідність меншої кількості спеціалістів, більш швидка розробка декількох клієнтських застосунків, легша підтримка через спільну кодову базу та зазвичай набагато менша витрата часу. У якомусь сенсі веб застосунки набули найбільшої популярність частково через їх крос-платформність, так як веб сторінки можна запускати практично на будь якій операційній системі через нативний для неї веб-браузер.

На кафедрі програмної інженерії також були дослідження стосовно конкретних клієнтських технологій та їх ефективності [1].

На даний момент існує багато різних фреймворків та бібліотек для крос-платформної розробки для багатьох різних операційних систем. Прикладом таких технологій можуть бути Qt, Boost, OpenGl, .NET Core, Flutter для десктопної розробки під Windows, OS X, Linux та React Native, Xamarin, Ionic і знову Flutter, для розробки під iOS та Android [2].

Таким чином можна одразу побачити першу перевагу фреймворку Flutter порівняно зі схожими технологіями – список підтримуваних клієнтів включає усі найпопулярніші і найпоширеніші сучасні операційні системи, як мобільні, так і десктопні, включаючи веб. Тобто в теорії достатньо одного досвідченого розробника для розробки та підтримки одразу 6 клієнтських застосунків, для кожного з яких бізнес логіка буде однакова і тільки інтерфейс буде

підлаштовуватись під нативні потреби з допомогою безліч офіційних і користувальницьких бібліотек. Підстроювання і випуск додатку на нову платформу, навіть не передбачену до цього під час усього циклу розробки, з Flutter не потребує ні наймання нових спеціалістів, ні великих часових або матеріальних витрат, більшість роботи буде на дизайнері, для розробки більш нативного інтерфейсу для нової платформи. Якщо розроблений до цього інтерфейс достатньо уніфікований та підходить для нової платформи випуск на неї додатку взагалі не потребує ніяких затрат.

Серверні застосунки, до яких під'єднуються клієнтські, зазвичай потребують окремої команди спеціалістів, яка розробить підходящу архітектуру для ефективного розподілу навантаження між фізичними серверами та легкого і, важливо, захищеного доступну до потрібних даних з клієнтських застосунків. Така розробка, очевидно, займає багато часу та потребує високої кваліфікації розробників.

Але якщо подивитись на сучасний ринок для абсолютної більшості програмних систем такий підхід може бути занадто дорогим і, що важливо, зайвим для їх потреб [3]. Коли не відомо чи «вистрелить» сервіс, до якого розробляється програмна система, треба мінімізувати витрати усіх ресурсів, для мінімізації втрат у випадок невдачі. Якщо заздалегідь відомо, що сервісом буде користуватись не велика кількість людей, витрачати час і ресурси на побудову максимально ефективного унікального серверного застосунку також не має сенсу.

У таких ситуаціях дуже вигідно скористатись сервісом, який надає весь необхідний функціонал серверного застосунку без необхідності складного налаштування архітектури, прикладом якого може виступати Firebase.

Firebase містить у собі усе що може знадобитись для сучасного сервісу, з основного: NoSQL база даних, з одразу доступними веб-сокетами, сховище для файлів, просте налаштування правил доступу до даних на JavaScript-подібній мові, інструменти для аналітики, крашлітики, авторизації та багато чого іншого. Також існують бібліотеки, для усіх сучасних мов програмування для зручного доступу до сервісів Firebase, без необхідності будувати власний механізм доступу до API.

З написаного вище можна зрозуміти, що для абсолютної більшості проектів найбільше всього підходить комбінація Flutter + Firebase, яка і зекономить велику кількість ресурсів на розробці та підтримці сервісу, що відкриває двері для великої кількості стартапів, які не могли би бути реалізовані до цього через нестачу ресурсів.

Але зручність цих технологій для розробки невеличких, простих сервісів не означає, що вони не підходять для більш масштабних проектів. Зокрема популярність Flutter для розробки крос-платформних мобільних додатків вже набула великих масштабів не зважаючи на відносну молодість технології. Ця технологія підходить і для великих, багатомільйонних сервісів, вже навіть більше ніж варіант розробки додатків нативно. Це можна побачити по таким застосункам і компаніям як Google Pay, BMW, Alibaba Group, eBay, Toyota та багато іншим [4].

Широкий спектр застосування також не оминув і Firebase. Серед гігантів, які користуються Firebase сервісами можна виділити Alibaba, The New York Times, The Economist та багато інших [5].

Те що ці дві обрані для дослідження технології використовуються одними з найбільших компаніями у світі доводить їх ефективність та необхідність на сучасному ринку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Клієнт-сервісні застосунки з’явились дуже давно, практично разом із інтернетом, за цей час зі зростанням потреби в ефективних методах розробки такого роду застосунків з’являлись нові, більш сучасні мови програмування і фреймворки, намагаючись задовольнити різко зростаючу клієнтську базу і зайняти на той час вільні сфери бізнесу. Чим ближче до нашого часу тим простіше ставала розробка клієнт-серверних додатків, все більше роботи делегувалось до фреймворків і бібліотек, все менше ресурсів було потрібно для розробки комплексних сервісів. Зі зростанням потреби в сервісах, зростала кількість програмістів і з’являлось все більше інструментів для більш комфортної розробки.

Не складно зрозуміти що чим менше роботи доводиться робити програмісту тим менше спеціалістів потрібно для розробки, тим менше буде витрачено і часових і матеріальних ресурсів. Прогрес не зупинявся і дійшов до моменту, де навіть розробка на фундаментально різні операційні системи можна було виконувати за допомогою одних і тих самих інструментів, що і називається крос-платформною розробкою. Якісь технології треба компілювати під конкретні платформи, якісь, працюють одразу за допомогою, наприклад, віртуальної машини, яка окремо зроблена для кожної платформи.

У нашій ситуації аналогічних Flutter технологій, які дозволяють розробляти додатки одразу на 6 платформ не існує, інші технології підтримують тільки частину з сучасних операційних систем.

React Native – крос-платформний фреймворк для розробки під Android/iOS на мові програмування JavaScript який вийшов у 2015 році, мабуть найбільший конкурент Flutter, та найкращий варіант розробки під мобільну платформу до його з’явлення. У Flutter також можна побачити деяке натхнення з цього фреймворку, механізм будування екранів та базове управління станами дуже схожі. Також дуже зручна функція Flutter «fast refresh», яка дозволяє швидко побачити зміни у коді на

екрані без необхідності повністю перезбирати додаток також вперше з’явилась у React Native. За допомогою цієї технології розроблено Skype, Bloomberg, деякі невеличкі модулі Facebook та Instagram, та багато чого іншого.

Ionic – як і React Native мова програмування – JavaScript, випущено у 2013 році, допомагає будувати гібридні мобільні та десктопні додатки використовуючи комбінацію веб і нативних технологій. Має інтеграцію Angular, React та View фреймворків. Через більшу орієнтованість на веб технології краще всього впорюється з веб додатками, але мобільні додатки легко відрізнити від нативних. Також для розробки на цьому фреймворку існує окрема IDE – Ionic Studio. З допомогою цієї технології розроблено такі мобільні додатки, як T-Mobile, BBC (дитячі освітні додатки) та EA Games.

Xamarin – вийшов у 2011 році, використовує мову програмування C# і .Net фреймворк для розробки під Android, iOS та Windows. Досі має широку аудиторію розробників, в основному серед користувачів .NET, велика кількість компонентів дозволяють імітувати нативний вигляд додатків. За допомогою цієї технології розроблено такі додатки як UPS, Alaska Airlines, Academy Members та багато іншого[6].

Існує більше не таких популярних фреймворків, але вони не набули широкого застосування, тому вважати їх конкурентами немає сенсу.

Перелічені фреймворки усі мають якісь недоліки, які вирішуються більш сучасним Flutter – усі не підтримують весь список сучасних операційних систем, деякі використовують більш старі мови програмування, та у всіх, на відміну від Flutter, продуктивність програє нативним варіантам.

Firebase містить у собі безліч різних сервісів для аналітики, баз даних, дистрибуції додатків, реалізації пуш повідомлень та багато чого іншого, тому аналогічні сервіси зазвичай не мають такого ж широкого спектру послуг.

Parse – сервіс з відкритим вихідним кодом який багато людей використовують замість Firebase через багато схожих сервісів та додаткових переваг. Цей сервіс працює з будь яким хмарним сховищем, тоді як Firebase тільки на Google Cloud. Також зазвичай кажуть, що міграція даних на цьому сервісі легша

ніж з Firebase. Індивідуальний хостінг також безкоштовний, так як його можна завантажити і встановити на будь-який сервер, тоді як Firebase користувачі для цього потребують оновлення проекту до enterprise статусу, що значно дорожче.

Back4App – надбудова над Parse, яка надає можливість використовувати як GraphQL, так і Rest API, також її поточні запити дозволяють у реальному часі синхронізувати та зберігати дані.

Subabase – ще одна альтернатива з відкритим кодом, головна відмінність якої це використання SQL бази даних замість NoSQL. Як і у Firebase цей сервіс надає такі корисні функції як real-time дані, автентифікацію, сховища для файлів і т.п. На відміну від Firebase, який надає безкоштовно необмежену кількість проектів і більшість сервісів Subabase надає лише 3 безкоштовних проекти.

Також існують наступні сервіси, які можна вважати альтернативами індивідуальних сервісів Firebase:

Автентифікація:

- Autho – зручний для підключення одинарної автентифікації;
- Passport – автентифікація для Node.js, краще всього працює для express базованих веб документів;
- Okta – гнучкий сервіс автентифікації, який об’єднує усі ваші додатки.

Cloud Firestore дозволяє зберігати і синхронізувати у реальному часі дані збережені у його NoSQL базі даних. Також цей сервіс збудований з ціллю витримувати навантаження найбільших сучасних додатків, має такі альтернативи:

- Mongo DB – теж NoSQL, дуже масштабуємо серед декількох платформ;
- Amazon DynamoDB – гнучкий та швидкий сервіс NoSQL бази даних;
- Oracle Database – база даних з об’єктно орієнтованим функціоналом;
- Azure CosmoDB – також гнучка NoSQL база даних розрахована на ефективне масштабування.

Cloud Storage дозволяє користувачам зберігати файли, має такі аналоги:

- Cloud Flare – надійне сховище для зберігання об’єктів;
- IBM Db2 – гнучке хмарне сховище даних.

Хостінг:

- Heroku – хмарний хостінг який полегшує побудову, запуск та підтримку додатків;
- DigitalOcean – простий, доступний хмарний хостінг.

Моніторинг додатку (аналітика, крашлітика, перформанс):

- Mixpanel – простий сервіс для мобільної аналітики з пре-збудованими таблицями;
- Woopra – аналітика спеціально налаштована для мобільних ігор;
- Amplitude – надає цифрові інструменти для поліпшення зростання бізнесу.
- Embrace – дозволяє побачити всі сесії всіх користувачів для повного розуміння наданого досвіду користування додатком;
- Bugsnag – надає можливість мониторингу помилок і інформації щодо стабільності;
- Instabug – надає такий самий функціонал як і Crashlytics, але додатково має механізм збору відгуків користувачів, що може допомогти з вирішенням тієї чи іншої проблеми.

У Firebase мається ще багато різноманітних сервісів і їх аналогів, перераховано було основні, які найближче відносяться безпосередньо до розробки клієнт-серверних застосунків.

1.2 Виявлення проблем та актуалізація рішень

Головна ціль будь якого бізнесмену це мінімізувати витрати без втрати якості на реалізацію того чи іншого бізнес процесу, у нашому випадку – розробити клієнт-серверний застосунок. Крос-платформна розробка сильно збільшує потенційну аудиторію за рахунок додаткових доступних операційних систем без збільшення витрат на розробку. Використання таких сервісів як Firebase збільшує швидкість

розробки та не потребує додаткових спеціалістів що також мінімізує витрати на розробку сервісу.

Використання комбінації Flutter + Firebase допомагає реалізувати бачення без великих витрат, що особливо корисну в умовах обмежених ресурсів, але функціонал обох технологій не обмежує невеличкими додатками, на них, як було доведено на практиці одними з найбільших компаній у світі, такими як Google, можна реалізувати комплексні, професійні та сучасні сервіси.

1.3 Постановка задачі

У цій роботі буде досліджено методи створення сервісно-орієнтованих та крос-платформених додатків за допомогою Flutter з серверною частиною на Firebase для виявлення реальної практичної ефективності такої комбінації для розробки клієнт-серверних застосунків з точок зору рівня витрати ресурсів та якості отриманих програмних систем, та розробки інструментів для полегшення процесу розробки.

Стосовно Flutter треба дослідити такі пункти:

- наявні інструменти та середи розробки;
- можливість побудувати сучасний додаток з ефективною багатошаровою архітектурою та за всіма сучасними принципами чистого коду [7] [8];
- затрати на адаптацію додатку під різні платформи;
- продуктивність додатку порівняно до аналогів та нативних додатків.

Над Firebase сервісами потрібно провести такі дослідження:

- перелік доступних сервісів та інструментів розробки;
- рівень захищеності та гнучкість налаштування правил доступу до даних;
- комфорт встановлення зв’язку з клієнтським застосунком;
- ефективність обробки масивних навантажень на систему;
- гнучкість налаштування баз даних під конкретні потреби сервісу.

У дослідженні важливо зрозуміти на скільки вказані технології ефективні для вирішення типових завдань, які можуть виникнути при розробці сучасного клієнт-серверного застосунку.

2 АНАЛІЗ ВИКОРИСТАНИХ У ДОСЛІДЖЕННІ ТЕХНОЛОГІЙ

2.1 Flutter

Під час розробки Flutter додатки працюють через віртуальну машину, що дозволяє відображати зміни у кодовій базі без необхідності повної перекомпіляції додатку. Для релізної версії Flutter додатки компілюються напряму у машинний код, незважаючи чи на Intel x64, або ARM, або взагалі на JavaScript, якщо збираємо під веб. Фреймворк знаходиться у відкритому доступі і має активну екосистему сторонніх пакетів, що додають функціонал до базового фреймворку [9][10].

2.1.1 Архітектурні шари

Flutter існує як розширювана шарова система, яка існує як серія незалежних бібліотек, кожна з яких залежить від шарів, які лежать в основі [11]. Ніякий шар не має привілейований доступ до шарів під ним і кожна частина фреймворку розроблена так, щоб її можна було замінити чи зовсім прибрати за необхідністю (рис. 2.1).

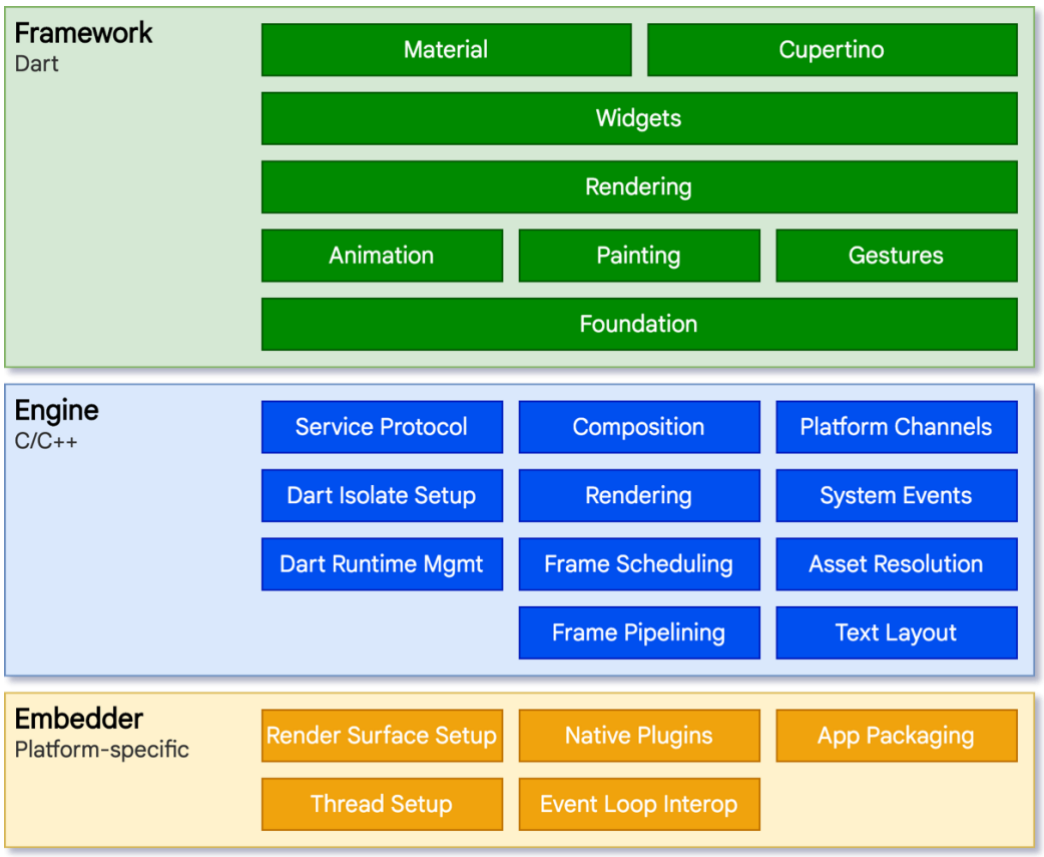

Рисунок 2.1 – Архітектура Flutter

Для операційної системи Flutter додатки запаковані так само як і нативні додатки. Окремий для кожної платформи embedder надає точку входу і координує з операційною системою такі речі, як поверхні для рендерінгу, вводу даних, і управляє циклом подій повідомлень. Embedder написаний на мові програмування, яка обрано згідно з платформою (Java і C++ для Android, Objective-C для iOS і т.п.).

В ядрі Flutter знаходиться Flutter Engine (далі двигун), який у більшості своїй написаний на C++ і підтримую примітиви, необхідні для всіх Flutter додатків. Двигун растерізує компоновані сцени кожен раз, коли потрібно відобразити новий кадр. Він надає низькорівнену реалізацію базового інтерфейсу Flutter, включаючи графіку, композицію тексту, доступ до файлової системи, архітектуру плагіну і ланцюжок інструментів для компіляції Dart. Двигун доступний для фреймоврку через відповідний пакет – dart:ui, який огортає підлеглий C++ код в Dart класи.

Більшість часу розробники взаємодіють з Fluttter через шар фреймворку, який надає сучасний, реактивний фреймворк написаний на Dart. Він включає велику

кількість різних фундаментальних та платформних бібліотек для побудови екранів, які містяться у різних шарах шару фреймворку. Якщо йти з низу вверх маємо такий розклад:

- базові фундаментальні класи і блоки для будування такі як анімації, малювання та жести;
- шар рендерінгу, який надає абстракцію для роботи з компонуванням;
- шар віджетів, який є абстракцією композиції. Кожний об’єкт рендерінгу з шару рендерінгу має відповідний віджет клас, також цей шар дозволяє визначати комбінації віджетів, які потім можна перевикористовувати;
- Material і Cupertino бібліотеки, які надають віджети, типові для Material і iOS мов дизайну.

2.1.2 Віджети

Віджети – базові блоки для будування користувальницького інтерфейсу. Кожна частина всіх екранів додатків являються віджетами. Віджети ієрархічно засновані на принципі композиції. Кожен віджет знаходиться в середині свого батьківського віджету і може отримати від нього контекст. Така структура продовжується аж до кореневого віджету (контейнер, який містить Flutter додаток).

Віджети не підлягають зміні, додатки оновлюють користувацький інтерфейс у відповідь до подій (наприклад взаємодія з користувачем) звертаючись до фреймворку щоб він замінив віджет у ієрархії іншим віджетом. Фреймворк потім порівнює нові і старі віджети і ефективно оновлює інтерфейс користувача.

Віджети зазвичай компоновані з великої кількості одноцільових віджетів. Існують віджети як для відображення чогось на екрані користувача, для і для ефективного розташування віджетів, додавання пробілів і відступів. Фреймворк надає доступ до двох основних класів віджетів – Stateful (зі станом) та Stateless (без стану). Для віджетів, які не містять ніякого стану та не мають необхідності

змінювати внутрішню структуру віджетів самим треба використовувати Stateless. Якщо структуру віджету треба змінювати в результаті якоїсь події, треба використовувати Stateful віджет. Кожен раз, коли віджет зі станом сигналізує необхідність зміни користувальницького інтерфейсу фреймворк ініціює механізм перебудови структури віджетів.

Коли структура віджетів стає все глибшою виникає проблема передачі даних між віджетами. Для цього у фреймворці існує третій вид віджетів – InheritedWidget. Такий віджет може містити будь який тип даних і легко передавати їх усім віджетам-нащадкам.

2.1.3 Рендерінг

Коли аналізують крос-платформні фреймворки не в останню чергу звертають увагу на його продуктивність. Як не дивно, але технологія рендерінгу Flutter надає йому рівень продуктивності до одно-платформному фреймворку. У прикладі мобільних крос-платформних фреймворків зазвичай створюється абстракція над бібліотеками інтерфейсу конкретних платформ. Ці нативні бібліотеки зазвичай перетворюють класи у вигляд на Canvas (полотно) об’єкті. Взаємодія коду крос-платформного фреймворку з нативним створює багато додаткових процесів, які сповільнюють додаток.

Flutter у свою чергу мінімізує такі абстракції не користуючись нативними бібліотеками рендерінгу на користь своєї структури віджетів. Dart код, який малює вигляд Flutter додатку компілюється в нативний код, який напряму використовує малювання на полотні. У Flutter для рендерінгу використовується технологія Skia, як і на Android. Також двигун фреймворку містить свою особисту копію Skia, дозволяючи розробнику оновлювати їх додатки з останніми покращеннями без необхідності оновлювати версію нативної операційної системи.

Принцип Flutter у тому, що просте є швидким [12]. Процес протікання даних від події до інструкцій рендерингу можна побачити на рисунку 2.2.

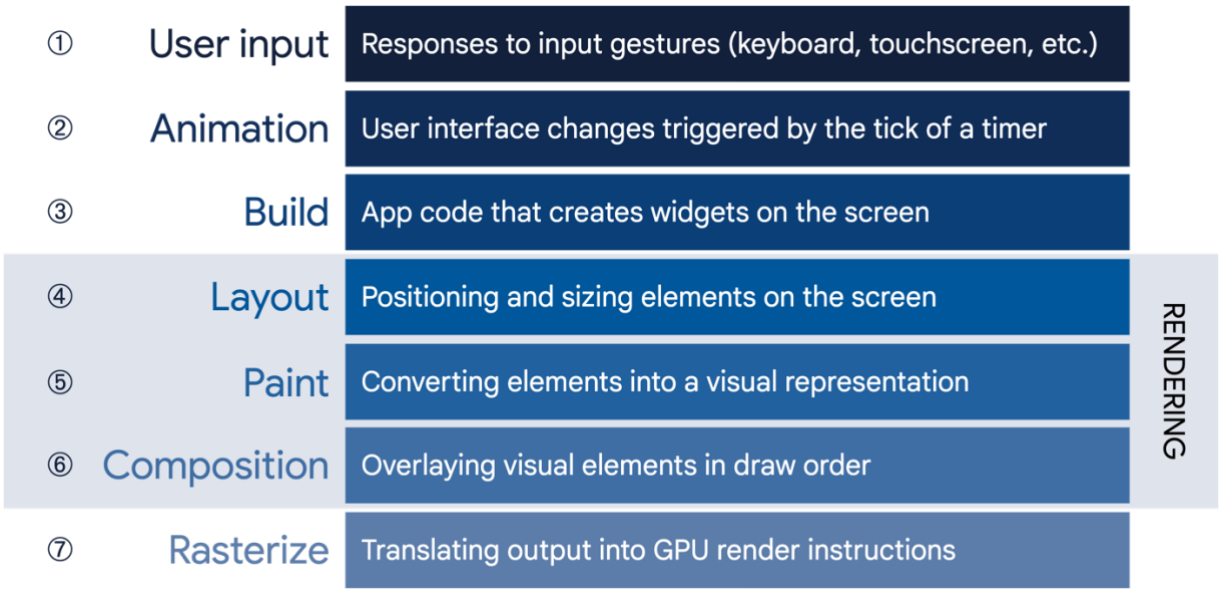

Рисунок 2.2 – Конвеєр рендерингу

Тобто процес рендеру проходить таким чином:

- коли Flutter хоче перемалювати свій фрагмент у відповідних віджетах викликається спеціальний метод «build»;
- під час фази будування Flutter перетворює структуру віджетів на відповідне дерево елементів (рис. 2.3) з одним елементом на кожний віджет. Елементи поділяються на компонентні, які просто містять у собі інші елементи, та елементи рендер об’єктів, які приймають участь у фазі малювання;

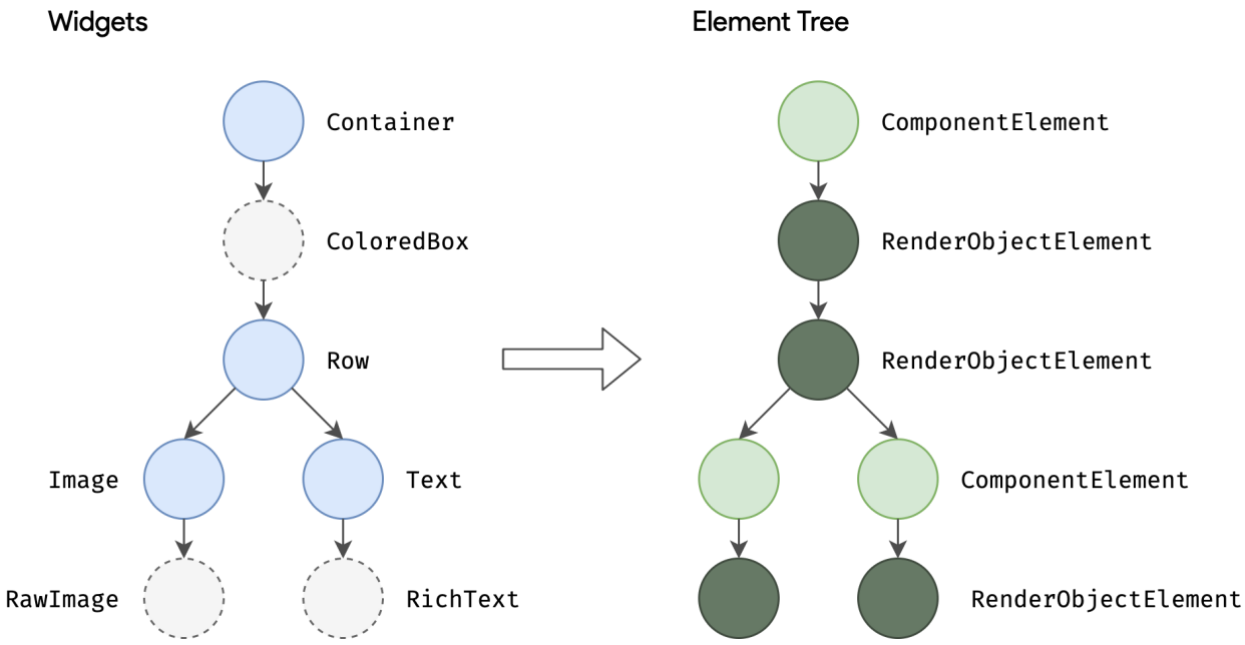

Рисунок 2.3 – Перетворення структури віджетів у дерево елементів

– під час стадії будування Flutter створює або оновлює об’єкт, який наслідує від RenderObject для кожного елементу рендер об’єкту у дереві (рис. 2.4);

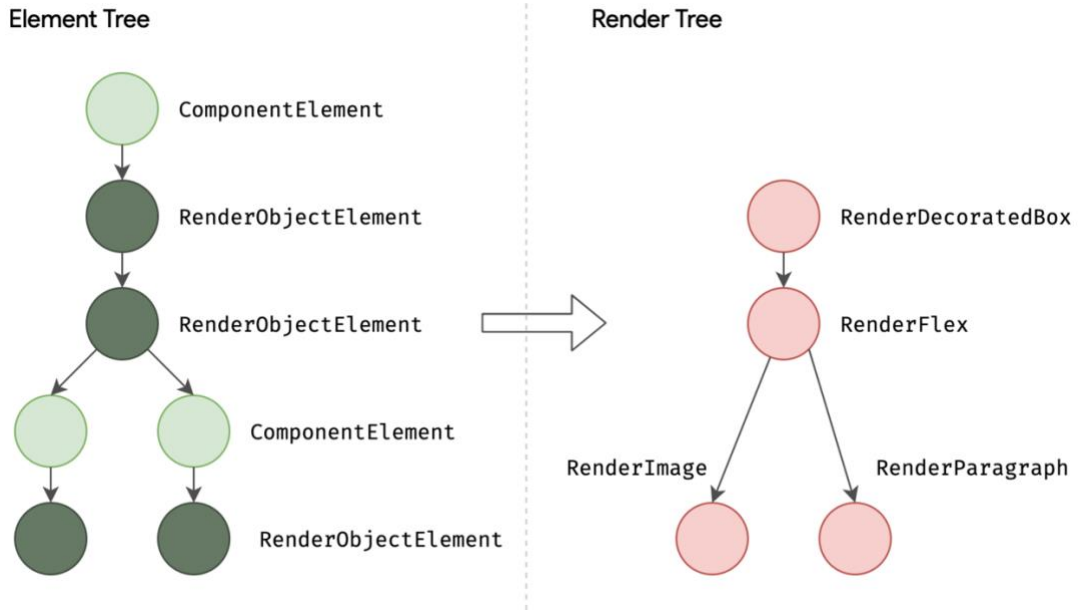

Рисунок 2.4 – Перетворення дерева елементів на дерево рендер об’єктів

– далі фреймворк проходить через дерево рендеру спочатку в глибину і передає вниз обмеження по розміру від батька до нащадка. Потім нащадки відповідають передаючи свої розміри батькам, які обов’язково повинні

бути в межах отриманих обмежень (рис. 2.5). Після цього кожен об’єкт готовий до малювання.

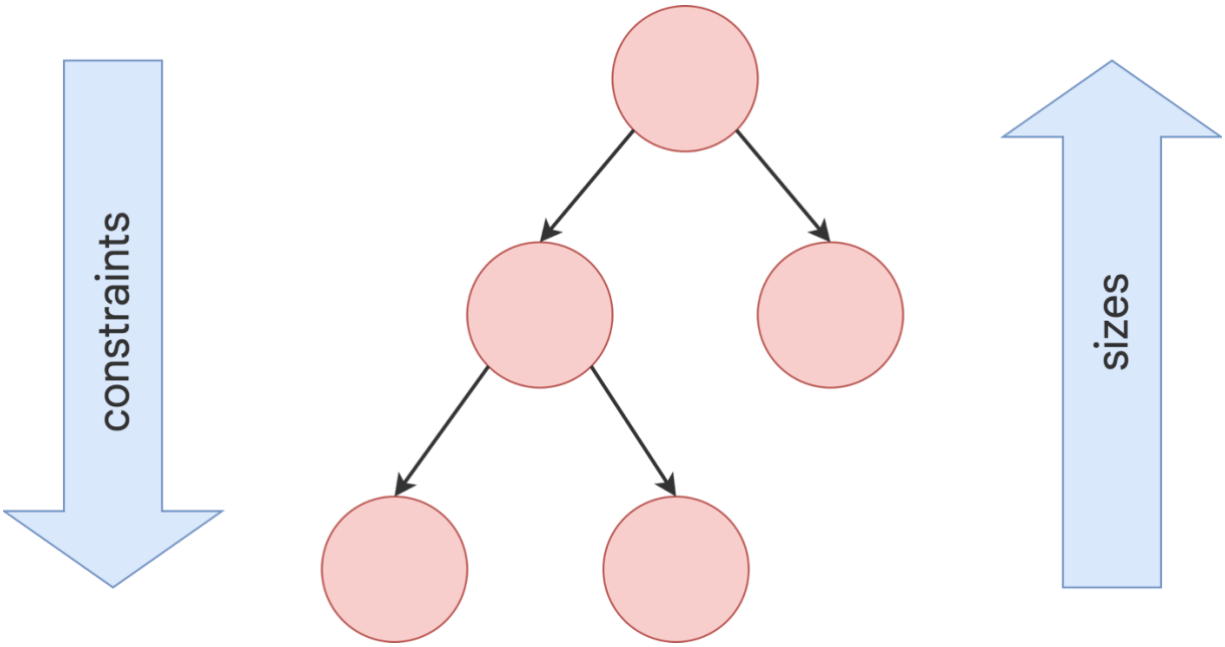

Рисунок 2.5 – Передача обмежень та розмірів в дереві рендер об’єктів

Корінь всіх рендер об’єктів це RenderView, який репрезентує повний вихід дерева рендеру.

2.1.4 Взаємодія з платформою

Раніше було описано яким чином флатер взаємодіє з платформою на низькому рівні. Тепер треба розібратись яким чином розробники можуть самі створювати плагіни, які на пряму взаємодіють з кодом платформи.

Для такої взаємодії Flutter містить спеціальний механізм, який називається platform channels, який дозволяє реалізувати спілкування між Dart кодом і спеціальним для платформи кодом як можна побачити на рисунку 2.6.

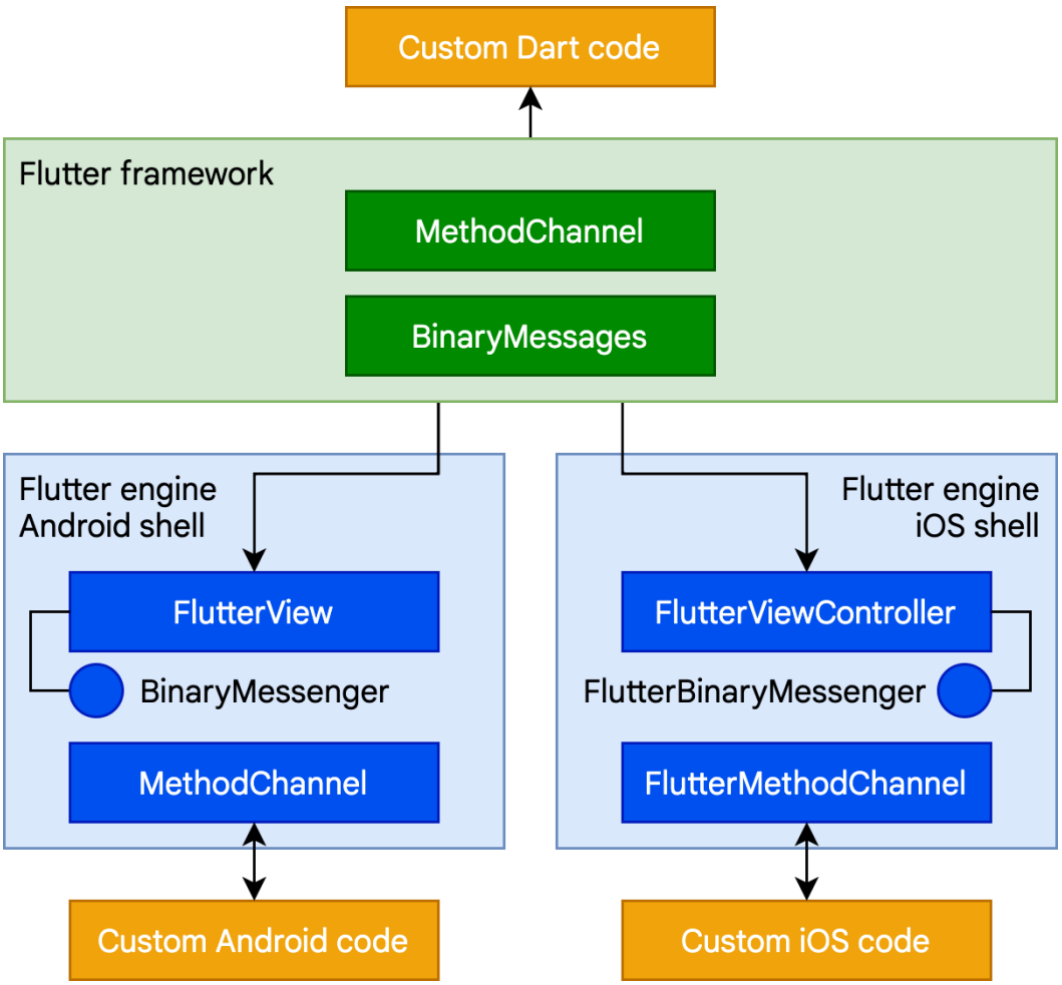

Рисунок 2.6 – Взаємодія Dart коду з кодом платформи

Дані серіалізуються із стандартного Dart типу, такого як Map, в стандартний формат і потім десеріалізуються в еквівалентний тип на мові програмування, яка використовується на платформі (наприклад HashMap в Kotlin та Dictionary в Swift).

2.1.5 Продуктивність

Перш ніж почати використовувати фреймворк Flutter з усіма його заявленими неймовірними можливостями, компанія все одно повинна дослідити всі проблеми, які може спричинити фреймворк, і найпоширенішим питанням є те, чи є компроміс у продуктивності достатньо малим, щоб виправдати використання фреймворку замість нативних рішенньь, які присутні на ринку протягом десятиліть і були

перевірені та вдосконалені з часом більше, ніж будь-яке інше рішення для платформ.

Фреймворк Flutter робить весь процес розробки в рази простішим, швидшим і, як наслідок, набагато дешевшим, але якщо продуктивність кінцевого продукту буде турбувати користувачів, це може зробити продукт неуспішним на ринку, роблячи всі витрачені ресурси на розробку витраченими марно, а збереження значної їх частини завдяки вибору дешевшого процесу розробки безкорисним. Цей ризик є причиною того, що відносно нові технології зазвичай впроваджуються у великих компаніях роками, і вони переважно використовуються невеликими компаніями, які можуть собі дозволити більші ризики через більшу обмеженість в ресурсах. У певному сенсі ці невеликі компанії проводять стрес-тестування цих технологій на реальних проектах, у разі успішності яких більш великі компанії замисляться про їх використання.

Flutter не є винятком, хоча його швидке зростання популярності перевершує більшість, не в останню чергу тому, що його джерелом є одна з найбільших компаній у світі – Google, яка також використовувала його для переписування деяких своїх популярних, широко використовуваних продуктів, таких як Google Pay. Ці продукти показали себе не менш ефективними ніж їхні нативні аналоги, з перевагою у набагато меншій і простішій кодовій базі.

Справжні великі проекти, написані з використанням Flutter, самі по собі доводять ефективність цієї технології, але важливо точно знати, наскільки продуктивність Flutter відрізняється від нативних альтернатив, особливо якщо проект має бути якомога швидшим, наприклад, для відеоігри важливий кожен кадр на секунду для найбільш оптимального досвіду користувача навіть на старих пристроях.

Точне порівняння Flutter і нативного Android додатку вже було зроблено та опубліковано в статті [13], вона містить відносно точне представлення відмінностей у продуктивності. Оскільки технологія, яку використовує Flutter, працює практично однаково для кожної платформи та зводиться до малювання

елементів на наданому платформою полотні, данні з статті можна вважати близькими для всіх платформ, які підтримує Flutter.

У згаданій статті дослідження завершується тим, що нативна програма працює швидше, ніж програма Flutter, у всіх аспектах, окрім рендерингу потоків, той факт, що є аспект кросплатформної структури, який працює швидше, ніж нативний аналог, вражає, але решта тестів показала більшу ефективність нативного додатку. Але це не означає, що продуктивність Flutter є проблемою, оскільки уважно вивчивши дані, зібрані автором, єдина велика різниця в продуктивності, яку можна помітити неозброєним оком, це запуск програми, де Flutter схоже виконує важку роботу, решта тестів показує незначну різницю в середньому в 50 мс, а найпоширеніші ситуації, з якими стикаються користувачі, виконуються з різницею в меншому діапазоні 10-30 мс.

Підсумовуючи, додатки Flutter працюють достатньо ефективно для абсолютної більшості випадків у сфері розробки додатків, оскільки більшість додатків, як правило, не виконують складних завдань інтерфейсу користувача.

Також важливо відзначити, що величезна популярність Flutter призвела до швидкого збільшення фінансування відповідної команди, а той факт, що це проект із відкритим кодом, дозволив великій кількості розробників самостійно вдосконалювати різні аспекти фреймворку. Багато з цих змін було офіційно включено до стабільної версії фреймворку після перегляду, тому ефективність Flutter зростає дуже швидко порівняно з більшістю інших фреймворків, що означає, що продуктивність може також покращитись у майбутньому.

2.2 Firebase

2.2.1 Доступні сервіси

У цьому розділі буде зроблено акцент на сервіси, які можуть бути корисними в типовому клієнт-серверному додатку, так як Firebase містить безліч спеціальних сервісів по типу Machine Learning.

Архітектура Firebase складається з декількох компонентів, які взаємодіють між собою щоб надати фреймворк для розробки та хостінгу веб та мобільних додатків [14].

Firebase Cloud Storage – сховище об’єктів, яке доступне на платформі Google Cloud. Коли цей сервіс додається до Firebase додатку отримується доступ до заходів безпеки Google і можливості безпечно скачувати, чи завантажувати дані.

Cloud Firestore – масштабований, гнучкий сервіс бази даних для серверної, мобільної і веб розробки. Він служить базою даних NoSQL документів. Надає «з коробки» доступ до веб сокетів, тобто можливість реактивно оновлювати дані у клієнтському додатку. Також має свою систему правил безпеки, яка пишеться на JavaScript-подібній мові програмування і контролює доступ до колекцій і документів згідно із потребами користувача.

Authentication – можна застосовувати для автентифікації користувачів через логін і пароль, пошту, номер телефону і різні соціальні мережі.

Hosting – сервіс хостінгу для веб контенту. Можна комбінувати з хмарними функціями, для розробки і хостінгу власних мікро сервісів.

App Distribution – дозволяє роздавати пре-релізні версії застосунків та налаштовувати для них список тестувальників.

Cloud Functions – надає можливість створення як HTTP ендпоінтів, так і функцій-тригерів, які реагують на прописані події, наприклад створення документу у певній колекції. Корисні коли сервіси, які потрібні для реалізації якогось функціоналу у додатку мають веб-хуки, наприклад сервіс для управління виплатами Stripe.

Remote Config – хмарне сховища даних типу ключ-значення, для управління якимись необхідними для клієнтського додатку прапорами і параметрами, наприклад мінімальну версію додатку, до якої користувач буде змушено обновити застосунок.

In-App Messaging – дозволяє створити систему для відправки пуш повідомлень на збережені, зазвичай у Cloud Firestore у документі користувача, токени пристроїв. Особливо корисно на мобільній платформі, де пуш повідомлення більш всього поширені.

Crashlytics – дозволяє відстежувати помилки у під'єднаних до сервісу додатків. Може містити код помилки, повідомлення, також відстежує вилети.

Performance – дозволяє побачити на скільки затратні ті чи інші процеси у застосунку, на яких платформах і з якими даними. Корисно використовувати для комплексного функціоналу задля тестування впливу змін на продуктивність.

Google Analytics – дозволяє створити безліч подій, які можуть відбуватись у додатку і аналізувати їх за допомогою реальних користувачів. Таким чином можна виділити пріоритетний функціонал, той що майже ніким не використовується і планувати розробку базуючись на реальних даних.

Test Lab – дозволяє проводити тестування додатків на пристроях, забезпечених безпосередньо Google. Тобто запуск автоматичних тестів делегується до цього сервісу.

Dynamic Links – дозволяє розробити так звані «глибокі» посилання, які будуть відправляти користувача до, наприклад, відповідного мобільного додатку, де вона буде оброблена, замість веб сторінки.

AdMob – дозволяє додати інтегровану Google рекламу до додатку, одразу надаючи метод монетизації сервісу.

Описані вище сервіси треба протестувати в контексті типового сервісу, та взаємодії з Flutter додатком.

2.2.2 Cloud Firestore

Оскільки Cloud Firestore відіграє роль бази даних, у якій зберігається більшість даних додатків, які використовують Firebase як хмарне сховище, важливо розуміти, як він працює та наскільки продуктивний у порівнянні з альтернативами [15][16].

2.2.2.1 Локації серверів

Коли мова йде про продуктивність серверу, перше, що спадає на думку, це його розташування. Чим ближче сервер до користувача, тим швидше буде час відповіді. Під час вибору розташування для служб Google Cloud Platform, які включають Cloud Firestore, розробнику пропонуються два варіанти: мультирегіональне розташування або розташування у одному конкретному регіоні.

Мультирегіональне розташування максимально підвищує доступність і довговічність вашої бази даних. Він зберігає репліки даних на кількох серверах у кількох регіонах, деякі регіони, які називаються свідками, беруть участь лише в процесі реплікації. Цей метод дозволяє мати узгоджені дані та час запиту в кількох регіонах, що рекомендовано для глобальних сервісів. На момент написання роботи доступні 2 мультирегіони: Європа та США.

Розташування в одному регіоні максимізує ефективність і вартість маніпуляцій з даними для конкретного регіону. Ррозташування в одному регіоні було б ефективнішим для програм, які не розроблені як глобальні та використовуються в певному регіоні, оскільки час запиту збільшується, чим далі ви перебуваєте від вибраного регіону. Доступні кілька регіонів в Америці, Європі, Азії та Австралії.

Оскільки Firebase бере з користувача плату за кількість документів, які ви читаєте, створюєте та видаляєте, очевидно, що мультирегіональне розташування приведе до більшої ціни за операції з документами, але забезпечить більш ефективний доступ до даних для багатьох регіонів світу.

2.2.2.2 Переваги і недоліки документо-орієнтованої NoSQL бази даних

Оскільки Cloud Firestore є документо-орієнтованої NoSQL базою даних, важливо розуміти принципи, що лежать в основі такого підходу до зберігання даних, оскільки не кожен сервіс буде ефективно її використовувати.

Основними характеристиками архітектури баз даних NoSQL є [17]:

- безсхемна структура;
- забезпечення ефективного та динамічного зростання представлень даних;
- горизонтальне масштабування шляхом реплікації даних колекцій і шардингу над масивними кластерами.

Основні переваги баз даних NoSQL:

- обсяг: дані в стані спокою - від терабайтів до ексабайтів наявних даних для обробки;
- швидкість: дані в русі - потокові дані, від мілісекунд до секунд для відповіді;
- мінливість: дані в багатьох формах – структуровані, неструктуровані, текстові тощо;
- не побудовано на основі таблиць і не використовує SQL для маніпулювання даними;
- може обробляти неструктуровані, безладні та непередбачувані дані;
- корисний для роботи з великими наборами розподілених даних.

Недоліки такі:

- невпорядкованість даних, важче запитувати необхідні дані;

- відсутність JOIN - відсутність зв'язків між даними змушує розробника надсилати кілька запитів на доступ до деяких даних.

Отже, підсумовуючи, якщо дані для служби потребують суворої структури для максимальної ефективності, було б рекомендовано використовувати звичайну реляційну базу даних SQL, але це не означає, що Cloud Firestore не можна використовувати для таких випадків. Навіть незважаючи на те, що структури документів не є строгими, і в одній колекції дозволено мати документи з різними структурами, структуру можна забезпечити вручну зусиллями розробників.

У Cloud Firestore на верхньому рівні знаходяться лише колекції, кожна колекція може містити лише документи, але кожен документ може містити підколекції. Структура може бути досягнута зусиллями клієнтської програми шляхом визначення строгих моделей і шляхів для зберігання даних. Це не вирішує проблему отримання кількох різних документів за допомогою одного запиту, але дає змогу передбачити вміст і розташування документів. Для більшості завдань, які можуть виникнути в типовому сервісі, цього має бути достатньо.

2.2.3 Інструменти розробки

Так як і Flutter, і Firebase належать одній компанії – Google, взаємодія між цими технологіями зроблена максимально доступною [18]. Також важливо те, що будь які онови і зміни у Firebase будуть адаптовані до контексту Flutter в приоритеті, що пришвидшує поліпшення сервісу завдяки новому функціоналу і виправленню помилок.

У Flutter на сайті, де поширюються як користувацькі, так і офіційні пакети для фреймворку (pub.dev) знаходяться всі необхідні бібліотеки для комфортної взаємодії клієнтського додатку з сервісами Firebase. Вони абстрагують роботу з API Firebase, що дозволяє набагато легше користуватись цими сервісами, не створюючи механізм роботи з Rest, як це довелось би робити зі звичайним сервером. Такі самі

пакети також можна знайти і для більшості сучасних клієнтських фреймворків, таких як React, View.js, Java Spring тощо.

3 МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ

В процесі дослідження треба проаналізувати наявні методи розробки додатків на Flutter с серверною частиною на Firebase, розробити власне архітектурне рішення та інструменти для максимально легкої розробки та підтримки запропонованого архітектурного рішення Flutter додатку. Також потрібно дослідити взаємодію Flutter та Firebase та розробити за необхідністю інструменти для більш ефективної комунікації двох технологій і відповідно більш швидкого процесу розробки.

3.1 Архітектура системи

Першочергова задача при розробці будь-якої програмної системи це розробка сучасної масштабованої архітектури, яка б в теорії була ефективна у реальних застосунках з великою аудиторією.

Для максимально ефективної реалізації принципу розділення залежностей треба зменшити людський фактор при проектуванні і підтримці системи. Це можна зробити за рахунок статичних перевірок оточення розробки. Для цього потрібно фізично відділити як окремий, пов’язаний між собою функціонал, так і різні шари в його середині, що змусить явно указувати доступ до тих чи інших даних у відповідному файлі залежностей, що спростить розуміння того, як функціонал пов’язан між собою. Отже основна архітектура Flutter додатку повинна буди наступною: програма являє собою набір незалежних модулів, які фізично розділені на окремі пакети. Модуль містить 3 пакети для кожного шару відповідно до принципів чистої архітектури (шари представлення, домену і даних) і код, який склеює ці окремі шари у єдину сутність. Хоч функціонал і відділений перехід між екранами різних модулів не може не існувати, тому варто також виділити усю

необхідну для навігації інформацію у окремий пакет у шарі презентації, щоб для навігації не доводилось створювати залежність на цілий шар презентації. Діаграму архітектури окремого модулю можна побачити на рисунку 3.1

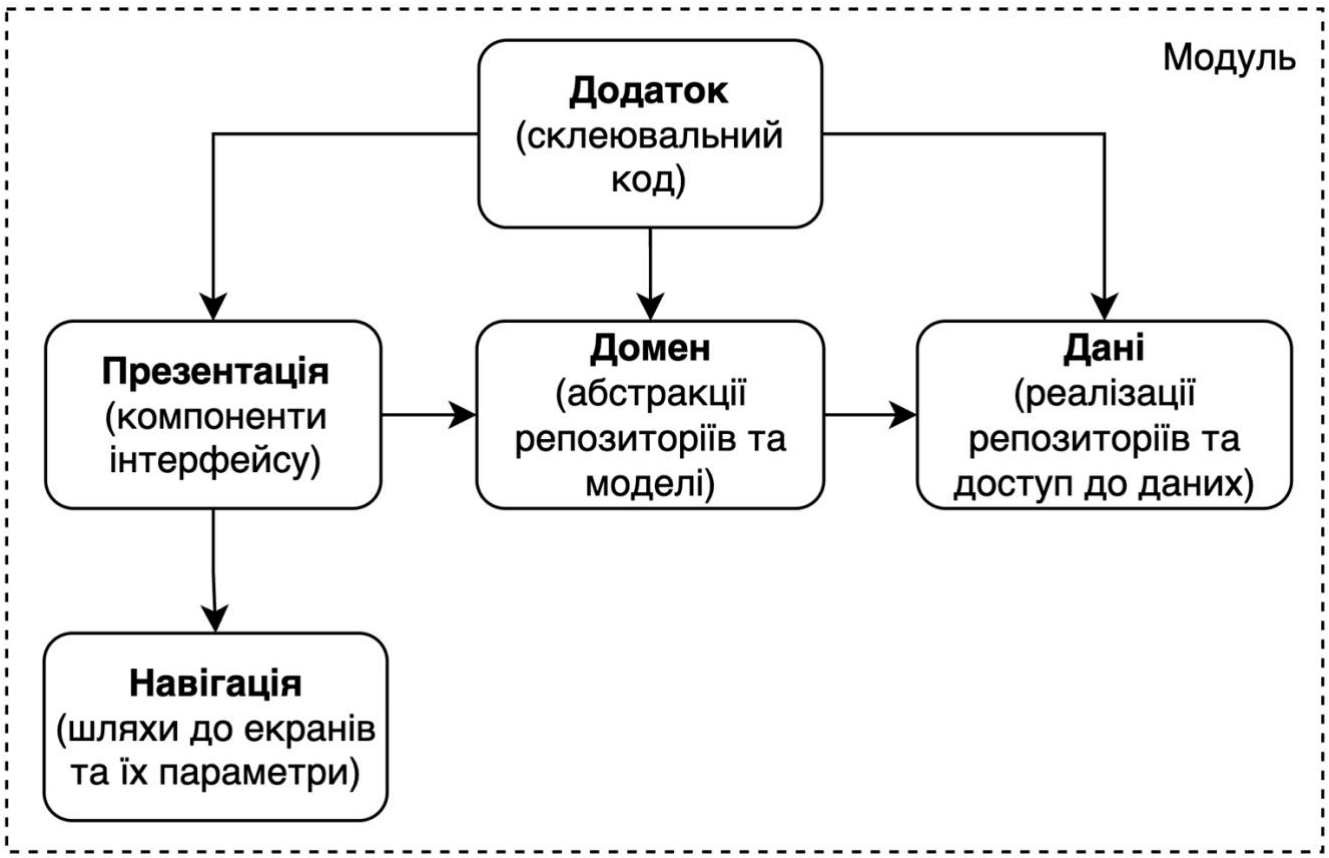

Рисунок 3.1 – Архітектура окремого модулю Flutter додатку

Склеювальний код представляє собою клас, який поєднує шари модулю. Це конфігурація модулю, яка дозволяє в залежності від стеку екранів впроваджувати, або позбуватись залежностей (якщо у стеці є екрани зі списку модулю залежності потрібно впровадити, та якщо нема – позбутись, якщо вони були впроваджені раніше). Також модулі можуть містити залежності на експорт, що дозволить мати модулі без екранів. На рисунку 3.2 можна побачити діаграму класу модулю. Конфігурація не зберігатиме ніяких даних у своїх екземплярах так що її можна копіювати скільки завгодно.

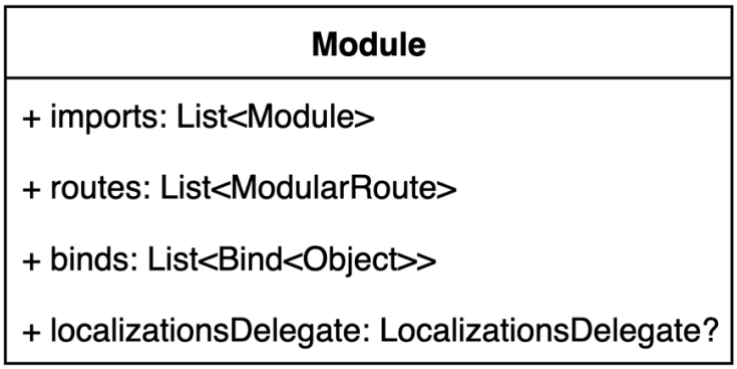

Рисунок 3.2 – Діаграма класу модулю

Модулі мають ієрархічну структуру, із модулем усього додатку у корні, що дозволяє визначати області залежностей, впроваджуючи і позбуваючись їх таким чином більш ефективно. Шляхи до модулів і відповідно до екранів, визначених там, задаються у строковій формі веб-подібного формату та визначаються у листі routes у спеціальному класі – ModuleRoute, який на відміну від класу для екранів приймає шлях та модуль в якості параметрів.

В результаті у типічному модулі отримуємо ієрархічну систему із 5 пакетів і зважаючи на те, що додаток складається з багатьох модулів отримуємо багато пакетів, які треба створювати при розробці. Такий підхід полегшує підтримку додатку, однак не важко здогадатись, що розробка таким чином дуже ускладняється. Створення новий пакетів, дублювання схожих файлів і т.п. буде займати занадто багато часу, що подовжить розробку, що у свою чергу відверне більшість розробників від подібної структури не зважаючи на її переваги. На рисунку 3.3 можна побачити сегмент структури папок додатка, який виходить у результаті впровадження розробленої архітектури (кожна папка являється окремим Dart пакетом).

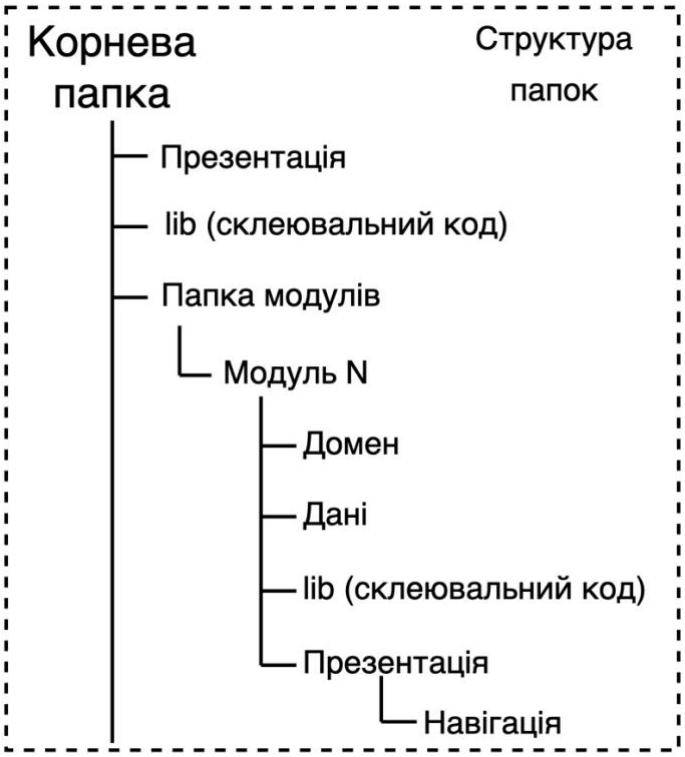

Рисунок 3.3 – Сегмент структури папок

Найкраще рішення такої проблеми це автоматизація створення цих пакетів. Потрібно написати скрипти і шаблони для найпоширеніших ситуацій, а саме:

- створення пакету модулю на основі визначеного шаблону з усіма під пакетами одразу наповненими базовими файлами;
- створення окремих пакетів-шарів для конкретного модулю;
- виклик команд підтягування залежностей та генерації коду у всіх модулях та у конкретному.

Такі скрипти значно полегшають і прискорять процес розробки нового функціоналу, що залишить всі переваги такого підходу, і ліквідує недоліки.

Ще одна проблема, яку викликає такий підхід, це однорідність версій залежностей. Так як додаток розділено на багато пакетів залежності для кожного пакету потрібно вказувати вручну, що може призвести до не однорідності версій залежностей у випадку їх дублювання у декількох окремих пакетах, що в свою чергу може призвести до несподіваних конфліктів та заблокувати компіляцію додатку. Для вирішення цієї проблеми можна скористатись автоматичними версіями залежностей мови програмування Dart. Це працює таким чином, що якщо

не вказувати версію залежності при її додаванні вона автоматично отримає версію, яка не конфліктує з іншими залежностями, якщо така існує. Щоб використати цю поведінку для однорідних версій треба створити окремий пакет, який буде складатись виключно із файлу із всіма залежностями додатку і їх версіями. Далі, якщо потрібна якась залежність, потрібно додати цей пакет зі всіма залежностями та потрібну залежність без версії. Таким чином нова залежність підтягне версію із файлу зі всіма залежностями, щоб уникнути конфліктів.

Ще одна менш важлива проблема це щира кількість файлів залежностей, яких у розробленому підході може бути сотні. У процесі розробки постійно додаються та видаляються потрібні залежності, що додає людського фактору до контролю набору залежностей для кожного пакету, що призводить до пакетів з імпортованими залежностями, які у ньому не використовуються. Процес чистки таких файлів залежностей також потрібно автоматизувати за допомогою скрипту, який, бажано, буде виконуватись автоматично при відправці нового коду у систему контролю версій.

Нарешті, най незначна проблема, яка стосується імпортів у файлах з кодом, це проблема їх формату. Бажано у файлах, у розробленому підході, мати виключно відносні імпорти до файлів, з того самого пакету, для очевидного розуміння взаємодій між файлами. Для цього також треба розробити скрипт, який буде виправляти неправильні імпорти, помічати ті, які неможливо виправити автоматично, та сортувати імпорти.

Для навігації потрібно також створити свою систему, так як це обов’язкова операція будь-якого додатку. У рамках багатомодульного додатку, розділеного фізично на пакети потрібно створити можливість узагальнити різні конфігурації навігація аби їх можна було використати однаково не зважаючи на знаходження у різних пакетах. Найкращим способом буде описати конфігурації шляхів навігації через запечатані класи, які будуть розширювати базовий інтерфейс, який буде містити гетери імені та параметрів екрану.

Для компонування всього функціоналу між собою, контролю ініціалізації та видалення залежностей потрібно створити систему, яка буде знову ж таки

автоматично цим займатись, значно полегшуючи роботу розробника. Найближче до оптимального рішення серед розроблених спільнотою з урахуванням спланованої архітектури є пакет flutter_modular, оскільки він дозволяє нам скомпонувати дані модулю у один зручний клас, кожен з яких має власні залежності, які вводяться та видаляються залежно від стану навігації, тому, якщо немає екранів, які належать до певного модуля, залежності цього модулю видаляються з пам’яті автоматично. Так само якщо ми переходимо на екран, який належить до певного модулю залежності цього модулю автоматично ініціалізуються. Однак, рішення має ряд недоліків, які потрібно виправити для більш ефективного процесу розробки, вирішення яких описано нижче.

По-перше, вкладена навігація, яка дозволяє нам відкривати екрани всередині інших екранів, які ми використовуємо для головного екрана з нижньою панеллю навігації, не має кешування. Це означає, що кожна вкладка завантажується кожного разу, коли користувач переходить до неї, через те що вкладка вважається екраном, і попередня вкладка підміняється на нову, що є поганою практикою, оскільки надсилається багато непотрібних запитів. Це може бути вирішено відмовленням від використання таб як екранів, що являється компромісом, який нам не підходить як мінімум через те, що контроль залежностей відбувається на основі навігації. Це можна вирішити наступним чином: для вкладених екранів існує спеціальний віджет, який розміщується на екрані та перехоплює навігацію за шляхом, який починається з шляху екрану, який містить у собі такий віджет. Модуляр містить такий віджет з відповідними змінами для підтримки маніпуляцій із залежностями. Це треба змінити таким чином, щоб для кожної вкладки був свій окремий віджет, який містить у собі вкладений екран, і всі ці віджети повинні знаходитись на екрані один за одним, переходячи на поверхню за індексом, щоб фреймворк автоматично не збував екрани, якщо їх немає у дереві віджетів.

По-друге, не має можливості передати до стеку декілька екранів одночасно. Це потрібно коли справа доходить до відкриття декількох екранів одночасно після переходу за динамічним посиланням чи push-сповіщенню, оскільки, коли ми ведемо користувача до відповідного екрана, задній стек екранів має бути

заповнений правильними елементами. Для вирішення цієї проблеми потрібно додати нову функції до класу навігації модуляра, яка буде надавати таку можливість. Так як базовий пакет модуляру розроблений виключно для одиночної навігації потрібно переписати як алгоритм отримання стеку екранів на основі переданих даних, так і спосіб відображання екранів.

По-третє локалізація ніяк не використовується модулем та делегати локалізації, які знаходяться окремо у кожному модулі, доведеться вручну експортувати із модулю та додавати у список делегатів додатку. Цю проблему можна вирішити додаванням поля у модуль, яке повертає делегат локалізації, та рекурсивного алгоритму збору цих делегатів по дереву модулів під час ініціалізації кореневого модулю.

В результаті отримуємо архітектурне рішення та інструменти, для його більш швидкого і зручного впровадження. Архітектура може бути так само ефективно використана у проектах з серверною частиною не на Firebase, так як зміни потрібно буде впровадити виключно у окремий пакет, який відповідає за прямий доступ до даних.

3.2 Firebase інтеграція

Для кожного сервісу Firebase компанія Google розробила та підтримує окремі пакети, які позбавляють необхідності взаємодіяти с сервісами на пряму використовуючи REST API. У процесі аналізу логіки взаємодії Flutter та Firebase, та окремих Firebase сервісів було виявлені такі аспекти, які можна покращити та/або оптимізувати:

- структуризація firestore документів для передбачуваності формату даних, отриманих зі сховища;
- робота з пагінацією документів firestore сховища при використанні веб сокетів;

- правила безпеки для хмарних сховищ мають набір функцій, які використовуються практично у будь-якому сервісі;
- завантаження файлів у Firebase Storage;
- комунікація із хмарними функціями;
- конфігурація аналітичних подій.

Як відомо дані NoSQL бази даних (БД) не структуровані, але через легкість і швидкість роботи із Firebase у деяких випадках не має сенсу розробляти та використовувати повноцінні реляційні бази даних, тож навіть якщо у базі даних планується зберігати суворо структуровані дані може бути обрана БД без таких обмежень. У такому разі структурованість даних буде досягатись штучно, за рахунок розробки структури з боку клієнту за допомогою статичних типів даних та шляхів збереження документів. Таким чином робота із firestore становиться передбачуваною із збереженням переваг NoSQL баз даних. Наприклад для зміни структури того чи іншого документу не треба проводити міграцію, так як такий тип баз даних дозволяє збереження не стандартизованих наборів даних, але треба тримати у голові необхідність зворотної сумісності, якщо такі зміни торкаються даних, які вже використовуються реальними користувачами.

Коли ми запитуємо обмежену сторінку даних за допомогою веб-сокета, завантаження наступної сторінки може здійснюватися одним із наступних способів:

- створення нового підключення до наступної сторінки даних і збереження її окремо. Такий підхід ускладнює обробку даних через чисельну кількість відкритих з’єднань, а також таких підхід може неправильно обробляти додавання та видалення елементів;
- створення окремої колекції із змінами та підписка не неї. Таким чином ми будемо локально робити зміни у даних у відповідь на зміни, однак це призведе до додаткової колекції і купи документів в ній. Такий варіант підійде виключно для не великих обсягів даних, та погано масштабується;
- видалення попереднього підключення та створення нового з обмеженням обсягу даних, що збільшується на одну сторінку. Такий підхід працює

ідеально і не має некоректної поведінки, однак він повторно запитує раніше завантажені сторінки даних знову, а будь-які операції з документами коштують грошей, так що якщо пагінація якихось даних дуже часта операція на фоні багатьох користувачів це може вилитись у великі втрати.

Отже, з точки зору економічної ефективності було б краще запитувати дані статично та відображати їх зміни лише за допомогою окремих механізмів оновлення. Це погіршить досвід користувача у деяких ситуаціях, коли такі зміни важливі, але якщо дані рідко чи взагалі ніколи не змінюються це ідеальний варіант. Рішення про те, чи слід завантажувати дані за допомогою веб-сокетів чи запиту HTTP, має прийматися окремо для кожної ситуації.

Для полегшення роботи з пагінацією варто створити компонент, який буде контролювати усі необхідні параметри та підвантаження даних.

Багато операцій у правилах безпеки firestore повторюються між сервісами, наприклад перевірка статусу автентифікації користувача. Якщо розробити набір базових функцій для правил безпеки процес їх початкового налаштування значно пришвидшиться.

Процес завантаження файлу у хмарне сховище складається з кроків створення посилання на файл, його завантаження та отримання публічного посилання на цей файл. Цей процес можна трішки оптимізувати огорнувши цей процес у єдиний метод.

Комунікація із хмарними функціями відбувається через простий клас, який огортає простий REST запит і звертається до конкретної хмарної функції за її іменем. Для більш зручної і безпечної взаємодії із хмарними функціями треба огорнути виклик функції у свій клас, який буде автоматично конвертувати помилки у створенні у додатку та приймати на вхід більш обмежені запечатані класи, що прибере вірогідність набору помилкового імені функції, або її параметрів.

Для конфігурації аналітичних подій можна скористатись зручним механізмом мови програмування Dart – розширення класів. Цей механізм дозволяє додавати методи до вже існуючих класів не вносячи зміни у самі класи, дозволяючи

таким чином уникнути необхідності створення класів, які б розширювали функціонал. У рамках нашої архітектури таких класів було б багато бо для різних модулів потрібні різні аналітичні події. З цим механізмом ми можемо описати розширення базового класу аналітики з відповідними до модуля подіями які будуть доступні виключно в рамках цього модулю.

4 АНАЛІЗ РЕЗУЛЬТАТІВ

У результаті експерименту було розроблено тестову програму Flutter на основі розробленої архітектури із службами Firebase, яка покриває потреби в базі даних, сховищі файлів, автентифікації тощо. Також було розроблено набір інструментів і скриптів для більш ефективної розробки у рамках запропонованої архітектури та більш ефективної взаємодії із сервісами Firebase. Розробка тестової системи показує реальні недоліки та проблеми, які видно виключно під час розробки повноцінного програмного продукту, дозволяючи легше виявити та покращити моменти, які складно уявити теоретично. Нижче наведено досвід процесу розробки.

4.1 Архітектура Flutter додатку

Архітектуру додатку у вигляді відокремлених пакетів на прикладі одного з модулів можна побачити на рисунку 4.1. Виділені папки являються самодостатніми пакетами, відокремленими від решти коду фізично.

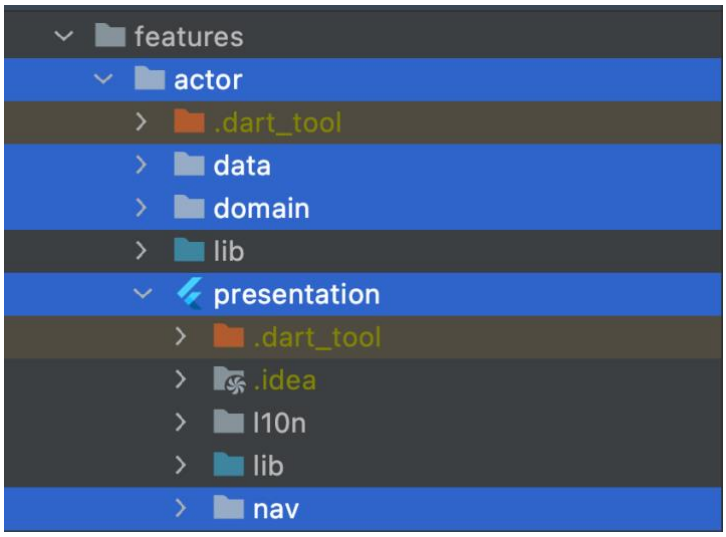

Рисунок 4.1 – Приклад фізичної структури пакету функціоналу

Для отримання даних з інших пакетів необхідно явно додати залежність у відповідний файл із залежностями як і було заплановано. Кожний пакет функціоналу містить один клас модулю модифікованого за нашими потребами пакету flutter_modular, який компонує різні шари функціоналу у один клас.

Для полегшення створення пакетів модулів було створено скрипти, які також заповнюють ці пакети за шаблонами автоматично. Самі скрипти були написані на мові програмування Dart, що полегшує їх зміну розробником при необхідності. Також алгоритм передбачає надання шаблонів зовні, що робить алгоритм більш гнучким, дозволяючи його використання для інших архітектур.

Для полегшення процесу оновлення залежностей та генерації коду були також написані скрипти. Конкретно скрипт генерації коду дозволяє нам не відправляти зайві згенеровані класи у систему контролю версій, розраховуючи на локальну генерацію потрібних класів при початковому налаштуванню проекту.

Версія Dart на момент написання статті не має так званих запечатаних класів, як, наприклад, мова програмування Kotlin. Але, це може бути вирішено за допомогою пакета генерації коду під назвою freezed. Він автоматично генерує шаблонний код для моделей, такий як код для серіалізації/десеріалізації та порівняння, і дозволяє додавати до класу поведінку, подібну до запечатаних класів. Це широко використовувалося для об’єктів передачі даних та навігаційної інформації, щоб ми могли передавати необхідні параметри на екран через такий клас. Також використовувалися інші пакети генерації коду, наприклад, для реалізації локалізації нами були прописані лише JSON-файли, а потрібний код генерувався відповідною командою. Для генерації класів для картинок і іконок теж було використано генерацію коду.

Для додавання конфігурацій збірок у Flutter додаток треба зробити налаштування окремо для кожної платформи. На щастя для цих конфігурацій знову ж таки мається пакет flutter_flavorizr, який робить усі потрібні налаштування автоматично. Для веб додатку у Flutter немає нативного механізму для додавання конфігурацій збірок, але є можливість передати конфігурацію через Dart код, що

дозволяє нам так само як і з мобільними додатками досягти автоматичного підтягування потрібних для роботи з Firebase ключів.

Для підтримки чистоти коду було написано 2 пре-коміту – скрипти, які виконуються перед кожним збереженням коду у системі контролю версій. Перший відповідає за сортування імпортів та переробку абсолютних імпортів на відносні. Він також помічає TODO коментарем імпорти на пакети, в середині цих самих пакетів, які треба переробити на відносні. Другий відповідає за чистку файлів з залежностями від тих, що не використовуються у відповідному пакеті. Ці два скрипти сильно полегшують контроль якості коду виключаючи людський фактор.

4.2 Отримані клієнтські додатки

Фреймворк Flutter із коробки містить більшість, якщо не всі віджети для послідовної розробки як для Android, так і для iOS у пакетах material і cupertino відповідно. Багато анімацій і взаємодій, наприклад прокручування, є нативними за замовчуванням і виконуються по-різному залежно від платформи, на якій запущено програму. Тому дуже легко зробити так, щоб програма виглядала природно на різних мобільних платформах. Однак іноді поведінку потрібно впроваджувати вручну. Ми можемо використовувати різні віджети та анімацію залежно від платформи, тому, щоб досягти максимального нативного відчуття, потрібне певне розуміння правил дизайну платформ, тому випадки, які не обробляються автоматично, можна реалізувати вручну (часто легко за допомогою уже визначених віджетів з відповідного платформі стандартного пакету).

Для веб-додатку потрібно внести деякі зміни в код. Як для мобільних платформ, так і для вебу є відповідні пакети, використання яких на непідтримуваній платформі призведе до помилки, тому, якщо їхнє використання є обов’язковим, потрібно скористатись умовними імпортами, та перевірити поточну платформу у місці використання відповідного коду. Окрім цього, запустити

програму у веб форматі дуже легко, однак на момент написання статті веб Flutter не є ідеальним для будь-якого використання. Веб Flutter схожий на мобільний додаток запущений у браузері, навіть виділення тексту є досить новою функцією, тому існує обмежена кількість випадків ефективного використання:

- прогресивні веб-програми;
- одно сторінкові сайти;
- існуючі мобільні Flutter додатки.

Flutter не підходить для статичних веб-сайтів із насиченим текстом потоковим вмістом. Отримані додатки можна побачити на рисунку 4.2.

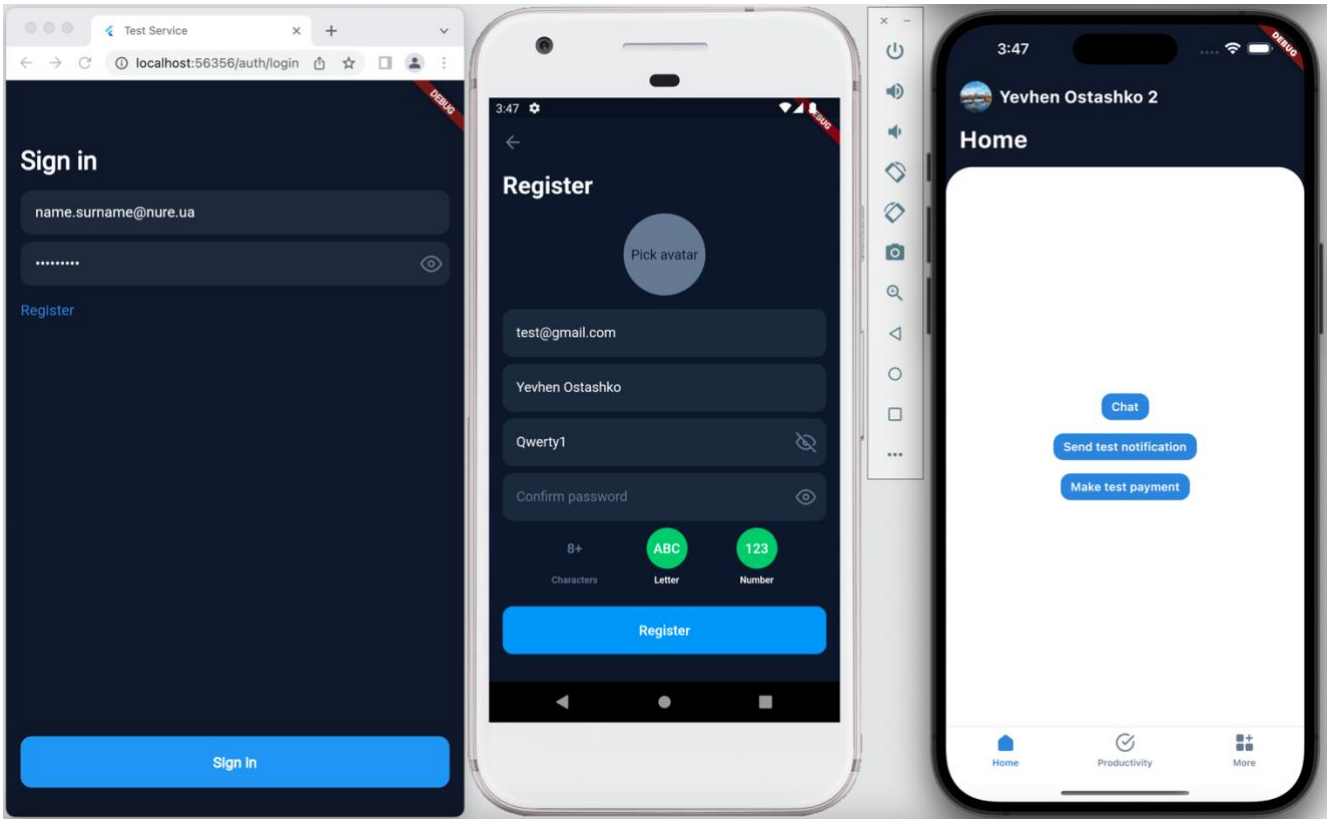

Рисунок 4.2 – Розроблені додатки

Щодо десктопних додатків, можливість їх створювати за допомогою Flutter є досить новою на момент написання роботи, тому багато функцій залишаються відсутніми. Для наших потреб підтримка Firebase ще не реалізована для десктопу, і лише підтримка MacOS зараз знаходиться в бета-версії. Через це у цій роботі десктопні додатки не тестувались

4.3 Тестування клієнтських додатків

Як і будь який інший програмний продукт Flutter додатки треба тестувати як і за допомогою людської сили, так і автоматично. Фреймворк має як усі типи автоматичного тестування, так і декілька специфічних для технології:

- unit тести – тести елементів логіки (функція, клас тощо.) на основі набору різних вхідних даних;
- віджет тести – тести конкретних віджетів, існує для перевірки роботи єдиного віджету на предмет вигляду і поведінки;
- інтеграційні тести – тестування усього додатку, або його більшої частини щоб перевірити правильність роботи системи з точки зору досвіду користувача;
- BLoC тести – різновид unit тестів заточений на роботу компоненту обраної архітектури шару презентації.

Для кожного з перелічених видів тестування існує відповідний пакет з усіма необхідними для тестування інструментами.

4.4 Налаштування Firebase

Проект Firebase легко створюється на відповідному сайті. Важливо знати, що за замовчуванням проект обмежений, і вам потрібно приєднати до нього банківську картку, щоб обмеження сервісів, наприклад Cloud Functions, були розблоковані. Плата не стягується до досягнення вказаного порогу використання, який різний для різних сервісів. Оскільки у нас тестова програма без великою кількості користувачів, нам не потрібно хвилюватися про списання грошей.

Далі йде підключення Firebase до наших Flutter клієнтів. На момент написання статті Firebase підтримує лише Android, iOS і веб-платформи із MacOS

платформою у бета-тесті. Для кожної платформи потрібна окрема конфігурація, але використання в додатку Flutter відбуватиметься з тим самим кодом, тобто плагіни сервісів Firebase вже роблять усі необхідні перевірки платформи.

Нарешті, щоб ми могли підключатися до баз даних і служб автентифікації, нам потрібно їх налаштувати. Для баз даних і сховища потрібно вибрати розташування сервера, обрано багато-регіональне розташування в Західній Європі. Для автентифікації нам потрібно вибрати провайдерів, обрано стандартний варіант - електронна пошта/пароль.

4.5 Аутентифікація

Для кожного сервісу Firebase створено відповідний офіційний пакет Flutter. Для автентифікації існує пакет firebase_auth. Цей пакет обробляє весь процес автентифікації, кешування користувача та забезпечує потік змін стану автентифікації. Він також підтримує скидання пароля, коли користувачеві надсилається електронний лист, налаштувати який можна у консолі Firebase.

Використання цього пакету просте, для автентифікації електронної пошти/паролю необхідно надіслати лише простий метод із електронною адресою та паролем, якщо користувач не існує, він створюється автоматично. Щоб зберегти інформацію користувача, потрібно використовувати Cloud Firestore або будь яку іншу базу даних, документ користувача варто створювати із тим самим ідентифікатор, що нам повертається при автентифікації.

4.6 Хмарні сховища

Для Cloud Firestore використовується пакет cloud_firestore. Щоб отримати

доступ до документа або колекції, шлях можна ввести як рядок або створити за шаблоном, подібним до конструктора (викликаємо метод з фрагментом шляху, який повертає цей самий клас, на якому знову можна викликати цей метод). Cloud Firestore підтримує веб-сокети із коробки, тож доступ до будь-якого документа чи колекції документів може реагувати на зміни, відображаючи будь-які зміни в базі даних у програмі без необхідності у запитах на оновлення даних. Це робить процес розробки швидшим і ефективнішим, оскільки будь-які зміни в документі відображаються автоматично, без необхідності обробляти такі випадки вручну. Для тестування роботи з Cloud Firestore було розроблено чат, який можна побачити на рисунку 4.3.

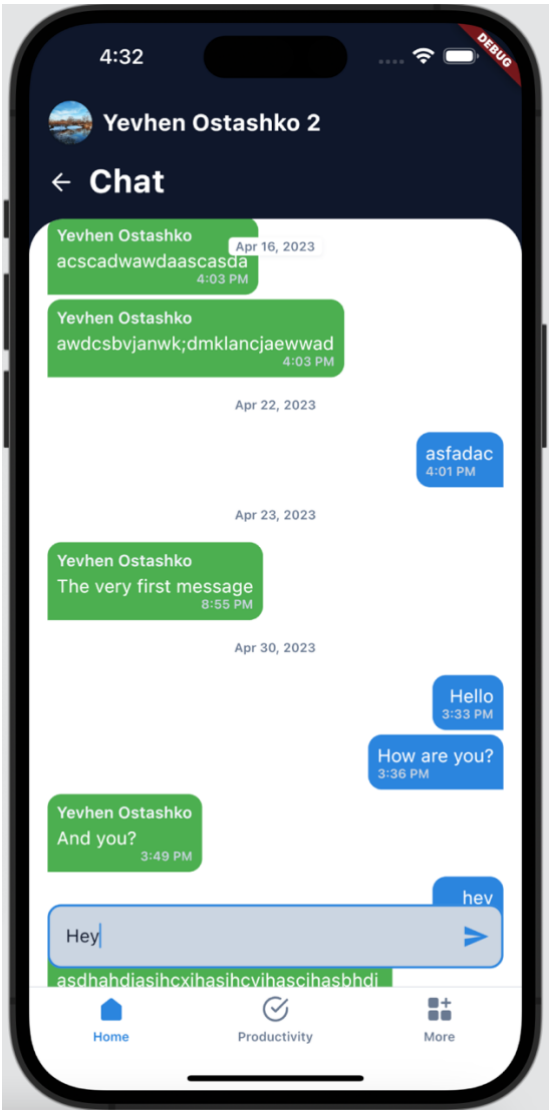

Рисунок 4.3 – Екран чату

Для полегшення роботи з пагінацією було створено клас, який сам контролює необхідну кількість даних, яку треба завантажити та надає простий інтерфейс взаємодії із собою. Цей клас можна побачити у додатку Г.

Для ефективної модифікації даних маються транзакції – операції, які модифікують дані почергово, не дозволяючи одночасний доступ до одних і тих самих даних. Також для простих операцій, таких як додавання елементів до масиву у документі мається клас FieldValue. Він дозволяє додавати елементи до масиву документу, видаляти їх, видаляти цілі поля та також додавати значення до числового поля (чи віднімати якщо додавати від’ємне значення). Також ці операції мають особливості транзакцій.

Правила безпеки для операцій з базою даних Cloud Firestore пишуться мовою, схожою на JavaScript. Щоб додати правило, потрібно додати шлях до документа, до якого потрібно додати правило (наприклад, «/users/{userId}»), і визначити умови для операцій читання та запису. Правила використовують ідентифікатор авторизованого користувача, тому вам потрібно використовувати службу автентифікації Firebase, щоб мати змогу встановити гнучкий захист для даних кожного користувача (наприклад дозволити змінення документа користувача виключно самому користувачу). Приклад написаних правил для документів користувачів та їх підколекції можна побачити на рисунку 4.4. Важливо зауважити що у випадку помилки у коді правил (наприклад при перевірці буде здійснено доступ до не існуючого поля у документі) доступ буде заборонено.

```
/// Checks if the user document with provided id exists
function userDocExists(userId) {
  return exists(/databases/$(database)/documents/users/$(userId));
}

/// Only the owner of the user document can write it
function canWriteUser(userId) {
  return authenticated() && userIdEquals(userId);
}

/// Checks if the user authenticated
function authenticated() {
  return request.auth != null && request.auth.uid != null
    && request.auth.uid != "";
}

/// Checks if current user id equals to the provided
function userIdEquals(userId) {
  return request.auth.uid == userId;
}

/// ---User

match /users/{userId} {
  /// User can write only its own document
  allow update, delete: if userDocExists(userId) && canWriteUser(userId);
  allow create: if canWriteUser(userId);
  allow read: if authenticated();

  /// ------User notification

  match /notifications/{notificationId} {
    /// User can read and write only its own notifications
    allow read, write: if authenticated() && userIdEquals(userId);
  }
}
```

Рисунок 4.4 – Приклад правил безпеки для колекції користувачів

Cloud Storage працює подібно до Cloud Firestore, тільки доступ до файлів здійснюється через загальнодоступну URL-адресу, яка отримується за допомогою спеціального методу відповідного пакета – firebase_storage. URL-адреса надасть повний, не обмежений доступ до файлу, до якого вона веде. Правила, визначені так само, як і для хмарного сховища, впливають лише на метод, який повертає загальнодоступну URL-адресу для файлу, тому, наприклад, якщо вам потрібно отримати кілька зображень, щоб показати користувачеві, вам потрібно буде викликати метод отримання URL кілька разів для кожного файлу для застосування правил. Це неефективний і тривалий процес, тому збереження URL-адреси файлу

у відповідному документі та генерування цієї URL-адреси під час завантаження файлу було б ефективнішим підходом із можливістю масштабування. Якщо доступ до файлу потрібно захистити, краще зашифрувати URL-адресу, ніж використовувати перший метод.

База даних реального часу (RTDB) є попередником Cloud Firestore. Це ефективне рішення з низькою затримкою, яке потребує синхронізації станів між клієнтами в режимі реального часу [19]. Cloud Firestore краще масштабується та має багатші та швидші запити, тому краще використовувати його замість RTDB у якості основної бази даних. Єдиним ефективним використанням RTDB, реалізованого в тестовій програмі, є контроль присутності користувача. У співпраці з хмарними функціями ми можемо відстежувати, коли користувач онлайн, оскільки на відміну від Cloud Firestore RTDB має опцію onDisconnect. Коли з'єднання користувача з Інтернетом зникає, хмарна функція змінює стан користувача у відповідному документі, дозволяючи нам знати присутність користувача. Це можна використовувати, наприклад, щоб дізнатися, чи потрібно нам надіслати користувачеві push-сповіщення, наприклад, про нове повідомлення в чаті. Якщо користувач уже присутній у чаті, сповіщення не має надсилатися.

4.7 Хмарні функції

Хмарні функції — це зручний спосіб додавання побічних ефектів до операцій за допомогою тригерів і створення кінцевих точок HTTP, якщо це необхідно. Наприклад, ми можемо надіслати push-повідомлення про створення документа без необхідності виконувати цю операцію вручну від клієнта та витрачати трафік і час користувача. Доступні тригери для створення, оновлення та видалення документів. Крім того, є тригер, який викликається для всіх 3 цих подій - тригер запису. Приклад простого тригеру на операцію створення документа нотифікації можна побачити на рисунку 4.5.

```
exports.onCreateNotification = functions.firestore
    .document(`${Collections.users}/{userId}/${UsersCollections.notifications}/{notificationId}`)
    .onCreate(async (snapshot, _) => {
        return sendPushNotification(snapshot.data() as NotificationModel, snapshot.id);
    });
```

Рисунок 4.5 – Функція-тригер на створення документу нотифікації

Що стосується хмарних HTTP функцій, то в нашому випадку було використано декілька:

– для ініціації переведення грошей між страйп аккаунтами;
– вебхук для Stripe операцій.

Щоб знати, чи платіж або будь-яка інша подія Stripe завершилася успішно чи з помилкою, у відповідній консолі потрібно визначити вебхук, який, по суті, є кінцевою точкою HTTP. Цей вебхук буде викликано, коли відбуватимуться події, визначені в консолі Stripe для конкретного хуку (тобто можна створювати декілька хуків для різних операцій, або один для всіх). У нашому випадку документ користувача містить інформацію про поточний процес платежу та зміни статусу на основі даних, отриманих у вебхуку.

Ще один використаний вид хмарної функції – pub sub. Ця функція виконується періодично в залежності від встановленого часу. У нашому випадку така функція може бути корисна для створення бекапів хмарної бази даних. Код цієї функції можна побачити на рисунку 4.6. Ця функція може буди пере використана у більшості проектів, які використовують Firebase як базу даних, єдина зміна яку потрібно внести це завантажити ключі доступу з GCP (Google Cloud Platform) для власного проекту і використати їх для автентифікації процесу створення бекапу.

```
exports.backup = functions.pubsub
    // At 12:00 AM, on day 1 of the month
    .schedule("0 0 1 * *")
    .onRun(async (_) => {
        appLogger.log(`Start of the project(${projectId}) firestore backup`);

        const timestamp = Timestamp.now();
        const timestampString = timestamp.toDate().toISOString();

        const firestoreClient = await getFirestoreClient();

        const promises: Promise<any>[] = [
            backupsCollection.doc(timestampString).set(
                {
                    name: timestampString,
                    createdAt: timestamp,
                }
            ),
            firestoreClient.projects.databases.exportDocuments({
                name: `projects/${projectId}/databases/(default)`,
                requestBody: {
                    outputUriPrefix: `gs://${getBackupStorageBucketPath(timestampString)}`
                }
            }),
        ];

        return Promise.all(promises);
    }
);
```

Рисунок 4.6 – Код періодичної хмарної функції для створення бекапу

Важливо знати, що хмарні функції мають необмежений доступ до всіх даних Firebase, тобто правила безпеки не застосовуються до операцій у них. Також функції обов’язково повинні закінчуватись повертанням значення, відсутність цього може призвести до роботи функції на її максимальний доступний час.

4.8 Тестування Firebase сервісів

Для тестування Firebase сервісів надається система система емуляторів, яка запускається на локальній машині та імітує поведінку реального сервісу. Емулятори доступні для таких сервісів: автентифікація, хмарне сховище, firestore, база даних реального часу, хмарні функції та хостинг.

Для роботи локально із переліченими сервісами у класах відповідних пакетів є змінна, яка містить шлях до локального емулятору, при передачі якої застосунок автоматично буде працювати локально, не впливаючи на реальний сервіс. Також для більш зручного тестування можливо зберегти набір даних для емулятору, наприклад документи для firestore, та запускати його з ними.

4.9 Remote config, In-App Messaging і AdMob

Remote config — це просто хмарне сховище пар ключ-значення, яке дозволяє змінювати поведінку програми без необхідності випускати оновлення. Наприклад, таким чином можна змінювати тему програми залежно від пори року. Треба зазначити що для цього сервісу немає можливості реагувати адаптивно, тому реагування на зміни у віддаленому конфігу локально можуть виходити виключно при запуску додатку.

Push-повідомлення реалізовано за допомогою пакетів firebase_in_app_messaging і flutter_local_notifications таким чином: спочатку у користувача запитується дозвіл на відправку повідомлень, у разі дозволу токен пристрою користувача отримується та зберігається в його документі під час авторизації. Коли відбувається подія, для якої потрібно надіслати push-сповіщення, ми створюємо документ у колекції сповіщень у Cloud Firestore з усією необхідною інформацією для сповіщення, наприклад ідентифікатор відправника та одержувача та тип сповіщення. Тригер Cloud Functions під час створення документа сповіщень використовує API In-App Messaging, щоб надіслати push до пристрою одержувача за допомогою токенів його девайсів [20]. Також для роботи з пуш повідомленнями на клієнтському боці було створено парсер повідомлень та логіка їх відображення коли додаток відкрито. Цей процес у більшості не змінний та може бути як і логіка створення бекапів скопійовано в будь-який проект.

Для AdMob використовується пакет google_mobile_ads. Доступні такі види

реклами:

- банер – прямокутник над або під екраном;
- інтерстиціальна – повноекранні оголошення, які потрібно закрити користувачеві;
- нативний – гнучкий тип, який дозволяє розміщувати рекламу, де хоче розробник, тому реклама ефективно інтегрується в інтерфейс програми;
- винагороджуюча – реклама, яка винагороджує користувачів за перегляд коротких відео та взаємодію з відтворюваною рекламою та опитуваннями.

Реклама — це чудовий спосіб монетизації вашого додатка, якщо вона не заважає роботі користувача. Також треба зазначити що на момент написання роботи підтримка AddMob для веб додатків відсутня і єдиний спосіб інтегрувати Google рекламу у Flutter Web цествоерення окремої html веб сторінки та інтеграція її у застосунок.

4.10 Аналітика

Google Analytics, Crashlytics і Performance легко реалізувати за допомогою відповідних пакетів Flutter. Єдине, що їм потрібно, це налаштування проекту Firebase, яке вже було зроблено раніше.

Google Analytics дозволяє нам надсилати будь-які події з простими даними у форматі ключ-значення. Наприклад, коли користувач заходить на екран, ми можемо надіслати відповідну подію з конкретними даними екрана. Коли буде зібрано достатню кількість даних, можна проаналізувати використання програми та встановити пріоритети розробки на основі досвіду користувачів [21]. Для зручного аналізу у відповідному розділу Firebase консолі можна побачити зібрані дані у структурованому вигляді, у тому числі графіки. Розроблена система опису подій у розширеннях класу аналітики показало себе як ефективне рішення.

Сервіс Crashlytics — це Google Analytics для помилок і збоїв. Він відстежує як визначені розробником помилки, які надсилаються вручну через API, так і збої програми. Тоді помилки можна побачити на відповідній вкладці в консолі Firebase і відреагувати відповідно. У процесі тестування цього сервісу було виявлено, що інколи не вистачає інформації про помилку або збій, що ускладнює їх виправлення, тому, можливо, краще скористатися окремим сервісом, який у тому числі б дозволив користувацькі відгуки і баг репорти.

Preformance працює просто: відправляємо запит на початку операції та інший на її завершенні. Корисно для складних операцій, щоб перевірити їх продуктивність на різних пристроях користувачів.

4.11 App distribution і веб хостінг

App distribution дозволяє зручно розповсюджувати мобільні додатки для iOS та Android без необхідності налаштовувати відповідні консолі. Ви можете запросити тестувальників через їхні електронні листи, щоб вони могли отримати доступ до програми. Переваги використання Firebase App Distribution порівняно з консолями платформ:

- керування попередніми випусками iOS і Android з одного місця;
- ранні випуски можна швидко доставити тестувальникам без встановлення SDK;
- у поєднанні з Crashlytics ми можемо отримати уявлення про стабільність тестових дистрибутивів.

Розмістити веб-програму можна на спеціальному домені, який зазвичай купують, або на домені, наданому Firebase. Firebase також дозволяє створювати бета-канали, що дозволяє розміщувати тестові програми на тимчасовому домені. Для використання хостингу потрібна проста консольна програма для налаштування

проекту, після чого можна легко розгорнути свою програму за допомогою простої команди.

4.12 Загальні характеристики

Процес розробки був дуже швидким і ефективним, система з віджетами дозволяє реалізувати будь-яку задумку без великих зусиль, реалізація анімацій теж дуже швидка завдяки вбудованим у фреймворк механізмам. Тобто якщо потрібно реалізувати нестандартні інтерфейси і анімації Flutter дозволяє це зробити у рекордні строки. З іншого боку така легкість може спричинити проблеми якщо розробник не достатньо досвідчений та до кінця не розуміє принципів чистого коду та архітектури. Із зростаючою популярністю все більше низько кваліфікованих розробників реалізують проекти за допомогою Flutter, створюючи і розповсюджуючи погані тенденції, які підхоплюють інші не кваліфіковані розробники. Ця проблема існує у практично будь-якій технології, але, наприклад, розробка під нативний Android навіть у самому простому своєму вигляді працює за принципами чистої архітектури, чого не можна сказати про Flutter. Це проблема не самого фреймворку, а поточного рівня кваліфікації на ринку.

Dart – одно поточна мова програмування, що означає що усі процеси, що відбуваються у додатку відбуваються у одному потоці. З першого погляду це великий недолік, який серйозно впливає на продуктивність, однак під час розробки було виявлено що паралельне виконання процесів використовується часто, та навіть якщо вони потрібні, у Dart є механізм створення окремих потоків, тільки він обмежений, потоки, які у Dart називаються ізолятами, не можуть працювати над одними і тими ж даними. Зазвичай асинхронної роботи у одному потоці вистачає для всіх операцій, створення ізолятів потрібно виключно якщо є якісь важкі розрахунки у синхронному коді, які можуть змусити програму зависнути до кінця роботи алгоритму. Такі ситуації трапляються дуже рідко, у типічних сервісах таке

навряд чи станеться. Dart дуже сучасна мова програмування, яка через популярність Flutter за останні роки була сильно оновлена, наприклад були додані механізми null безпеки.

Також було виявлено, що мобільні додатки під Android та iOS повністю відповідають всім вимогам відповідних платформ у вигляді та безпеці додатків. Flutter фреймворк містить усі необхідні інструменти для розробки мобільних додатків, які неможливо відрізнити від нативних аналогів із перевагою у швидкості розробки, простоті реалізації нестандартних завдань та продуктивності. Однак того ж не можна сказати про Web платформу, на якій фреймворк на момент написання роботи не відповідає стандартам, бракує багато необхідних інструментів та може ефективно використовуватись у обмеженій кількості сценаріїв.

4.13 Подальші дослідження

Світ інформаційних технологій розвивається дуже швидко і відносно часто з’являються нові технології, призвані виправити базові недоліки аналогів, та покращуються вже існуючі. Flutter відносно молода технологія, яка не дивлячись на це дуже швидко набрала популярність та вже використовується у багатьох реальних проектах. Завдяки цьому та тому, що вона розроблюється одною з най великих компаній у світі як з боку фінансів, так і з боку інноваційності у світі інформаційних технологій, швидкість розвитку фреймворку, та мови, на якій він написаний не може не вражати, що у тому числі додає варіанти сфер для подальшого дослідження технології.

Фреймворк продовжую покращуватись, робота на доступних платформах оптимізуватись, та навіть додаються нові платформи. Базова робота фреймворку заточена під розробку одно сторінкових додатків, що дуже ефективно працює на мобільних на десктопних платформах, але того ж самого не можна сказати про Web платформу.

Flutter Web фундаментально по результатам дослідження не підходить для розробки повноцінних веб додатків, фреймворку дужа складно транслюватись на цю платформу і виходить розробити тільки імітацію. Всі елементи фреймворку малюються на канвасі, що робить не можливим використання багатьох html компонентів. Але з розвитком фреймворку імітація може наблизитись достатньо, щоб відрізнити було достатньо складно.

Із розвитком фреймворку десктопні платформи (Windows, Linux, MacOS) скоріш за все будуть працювати так само ефективно як і мобільні через схожість їх базових принципів, у разі чого Flutter без може стати найпопулярнішою технологію для розробки клієнтських додатків, витиснувши аналоги з ринку для нових сервісів.

Також варто зазначити розвиток пристроїв, що працюють на доступних фреймворку операційних системах, з’являються нові види телефонів, з екранами, які складаються, декількома екранами одночасно тощо. З’являються нові пристрої, такі як розумні годинники, розумні будинки і багато чого іншого. Адаптація фреймворку під такі зміни і ефективність таких адоптацій може також бути проаналізовано та, у разі позитивних результатів, Flutter може отримати ще більшу розповсюдженість.

Так само можна сказати і про SAAS (Software as a service) серверів. Потреба у таких сервісах буде завжди, через полегшення процесу розробки, таких сервісів більше ніж один і кожен з них має свою переваги та недоліки, які можна порівняти та отримати алгоритм, за яким можна обрати максимально ефективний варіант для конкретних потреб.

ВИСНОВКИ

В процесі виконання завдання до кваліфікаційної роботи магістра були вирішені наступні завдання:

– ознайомився з різними технологіями крос-платформної розробки клієнтських застосунків;
– детально розглянув фреймворк Flutter на мові програмування Dart, його архітектуру та надані інструменти;
– ознайомився з різними сервісами, які надають готові інструменти для легкого налаштування серверного застосунку;
– детально розглянув доступні у Firebase сервіси;
– розглянув інструменти для зв’язку між Flutter застосунком і Firebase сервісами;
– розроблене архітектурне рішення для Flutter додатку та набір інструментів для його ефективної реалізації;
– проаналізовані та покращені деякі комунікації між Flutter та Firebase;
– побудовано тестовий сервіс на основі розробленого архітектурного рішення із використанням описаних сервісів Firebase і проаналізовано процес розробки і отриманих результатів;
– припущені майбутні напрямки дослідження використаних у роботі технологій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кіріченко І., Проніна Д. Comparison of Redux and React Hooks Methods in Terms of Performance. CEUR Workshop Proceedings. 2022. №3171. С. 791–800.

2. Vaibhav Patil. Flutter-Modern and Easy Technology to Build Applications. ResearchGate. 2023. №11. С. 458–459.

3. Sus B., Tmienova N., Revenchuk I., Vialkova V. Development of virtual laboratory works for technical and computer sciences. Communications in Computer and Information Science. 2019. 1078 CCIS. С. 383–394.

4. Flutter apps in production URL: https://flutter.dev/showcase (дата звернення: 01.03.2023).

5. Top 10 companies using Firebase URL: https://blog.back4app.com/companies-using-firebase (дата звернення: 01.03.2023).

6. Best Cross-platform development frameworks URL: https://kotlinlang.org/docs/cross-platform-frameworks.html#xamarin (дата звернення: 01.03.2023).

7. Мартін Р. Чиста архітектура, 2019. – 368 с.

8. Мартін Р. Чистий код. Створення і рефакторинг за допомогою Agile. 2019. 448 с.

9. Артем Великий. FLUTTER: A FULL INTRODUCTION TO THE FRAMEWORK. Axon. 2020. С. 2-3.

10. Flutter architectural overview URL: https://docs.flutter.dev/resources/architectural-overview (дата звернення: 03.03.2023).

11. Damian Białkowski, Felipe Diniz Dallilo. FLUTTER UM FRAMEWORK PARA DESENVOLVIMENTO MOBILE ResearchGate. 2022. С. 3-5.

12. Slavimir Stošović, Dušan Stefanović, Milan Bogdanović, Nikola Vukotić. THE USE OF THE FLUTTER FRAMEWORK IN THE DEVELOPMENT PROCESS OF HYBRID MOBILE APPLICATIONS. ResearchGate. 2022. С. 5-6.

13. D. Białkowski, J. Smołka. Evaluation of Flutter framework time efficiency in context of user interface tasks. ResearchGate. 2022. С. 4–5.

14. Anil Trimbakrao Gaikwad. FIREBASE - OVERVIEW AND USAGE. ResearchGate. 2022. С. 2-4.

15. Omar Almootassem, Syed Hamza Husain, Denesh Parthipan. A Cloud-based Service for Real-Time Performance Evaluation of NoSQL Databases. Arxiv. 2017. С. 3-4.

16. Azad, Avi Chaudhary, Jatin Chauhan. ANDROID APPLICATION USING FLUTTER AND FIREBASE WITH LBRS TO FIND PEOPLE OF THE SAME INTEREST AND COMMUNICATION PLATFORM. IRJMETS. 2022. №4. С. 4773-4775.

17. W. Khan, K. Teerath, C. Zhang та ін. SQL and NoSQL Databases Software architectures performance analysis and assessments - A Systematic Literature review. Arxiv. 2022. С. 3.

18. Jashandeep Singh, Swapnil Srivastva, Dipanshu Raj та ін. FLUTTER AND FIREBASE MAKING CROSS-PLATFORM APPLICATION DEVELOPMENT HASSLE-FREE. IRJMETS. 2022. №4. С. 3-5.

19. Shayan Bagchi. Firebase-A Cloud Hosted NoSQL Database. ResearchGate. – 2022. С. 8-9.

20. Bhavin M. Mehta, Nishay Madhani, Radhika Patwardhan. Firebase: A Platform for your Web and Mobile Applications. IJARSE. 2017. №4. С. 46-47.

21. Julian Harty, Haonan Zhang, Lili Wei та ін. Logging Practices with Mobile Analytics: An Empirical Study on Firebase. Arxiv. 2021. С. 1-3.