МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Дослідження технологій створення кросплатформенних мобільних застосунків**

Виконав: студент 2 курсу, групи 8.1219-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Андрєєв Р.Т.
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н. В. І. Попівщий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Айті Діменшн»
В.С. Тряпічко
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

Кафедра _____ програмного забезпечення автоматизованих систем
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри_____ В.Г. Вербицький
“_01_” __вересня_____2020_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

_____Андрєєву Роману Тарасовичу_____
(прізвище, ім’я, по батькові)

1.Тема роботи ___Дослідження технологій створення кросплатформенних мобільних застосунків_____

керівник роботи _____ В.І. Попівщий, к.ф.-м.н., доцент.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _“25”_травня_2020_року № 600-с_____

2. Строк подання студентом кваліфікаційної роботи___30.11.2020____
3. Вихідні дані магістерської роботи
- комплект нормативних документів;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми створення кросплатформенних мобільних додатків;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)_____слайдів презентації

3

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.09.2020

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.09-10.09.20	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.20	виконано
3	Аналіз існуючих методів рішення	13.09-17.09.20	виконано
4	Дослідження області кросплатформенних мобільних додатків	18.09-24.09.20	виконано
5	Узгодження подальших дій з науковим керівником	25.09-26.09.20	виконано
6	Аналіз теоретичних відомостей	27.09-15.10.20	виконано
7	Проектування інтерфейсу мобільних додатків	15.10-23.10.20	виконано
8	Узгодження інтерфейсу з науковим керівником	23.10-24.10.20	виконано
9	Реалізація функціоналу авторизації користувачів	25.10-14.11.20	виконано
10	Представлення отриманих результатів науковому керівнику і узгодження плану подальшого дослідження	15.11-16.11.20	виконано
11	Реалізація функціоналу хмарного сховища даних	16.11-20.11.20	виконано
12	Проведення аналізу можливостей розроблених програмних за стосунків	21.11-23.11.20	виконано
13	Оформлення звіту	24.11-29.11.20	виконано

Студент _____ Андрєєв Р.Т.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Попівщий В.І.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 103

Рисунків: 39

Таблиць: 12

Джерел: 31

Андрєєв Р. Т. Дослідження технологій створення кросплатформенних мобільних застосунків.

Кваліфікаційна робота для здобуття ступеня вищої освіти магістра за спеціальністю 121 – Інженерія програмного забезпечення, науковий керівник В.І. Попівщий. Інженерний інститут ЗНУ.

Мета кваліфікаційної роботи полягає у дослідженні методів створення мобільних кросплатформенних застосунків, їх особливостей, а також у створенні власних мобільних застосунків за допомогою двох різних технологій, які будуть працювати у режимі онлайн та будуть взаємодіяти з одним хмарним сховищем. У нашому випадку перший мобільний застосунок буде створений за допомоги технології React Native, другий за допомоги Flutter, а серверну систему завдяки хмарному сервісу Firebase. Застосунки повинні бути з базовими функціональними можливостями.

Досліджено методи і сучасні конкуруючі технології створення мобільних кросплатформенних застосунків, їх проблематику, можливості розробки, кількісні і якісні характеристики, та використання систем. Для розробки першого продукту використовувалися мова JavaScript та фреймворк React Native, для другого – мова Dart та фреймворк Flutter, для створення загальної серверної системи – Firebase Authentication та Firebase Cloud Storage.

Ключові слова: *КРОСПЛАТФОРМЕННИЙ, JAVASCRIPT, TYPESCRIPT, JSX, REACT, REACT NATIVE, DART, FLUTTER, FIREBASE, FIREBASE AUTHENTICATION, FIREBASE CLOUD STORAGE, МОБІЛЬНИЙ ЗАСТОСУНОК, TODO APP.*

SUMMARY

Pages: 103

Figures: 39

Tables: 12

Sources: 31

Andrjejev R.T. Research of technologies for creating cross-platform mobile applications.

Qualification work for obtaining a master's degree in specialty 121 - Software Engineering, supervisor V.I. Popivchiy. ZNU Engineering Institute.

The purpose of the qualification work is to study the methods of creating mobile cross-platform applications, their features, as well as to create their own mobile applications using two different technologies that will work online and interact with a single cloud storage. In our case, the first mobile application will be created using React Native technology, the second using Flutter, and the server system thanks to the cloud service Firebase. Applications should be with basic functionality.

Methods and modern competing technologies of creation of mobile cross-platform applications, their problems, possibilities of development, quantitative and qualitative characteristics, and use of systems are investigated. JavaScript and the React Native framework were used to develop the first product, Dart and the Flutter framework were used for the second, and Firebase Authentication and Firebase Cloud Storage were used to create a shared server system.

Keywords: *CROSSPLATFORM, JAVASCRIPT, TYPESCRIPT, JSX, REACT, REACT NATIVE, DART, FLUTTER, FIREBASE, FIREBASE AUTHENTICATION, FIREBASE CLOUD STORAGE, MOBILE APPLICATION, TODO APP*.

ЗМІСТ

ВСТУП

Актуальність теми

З появою мобільного Інтернету мобільний телефон став основним джерелом ресурсів необхідних для роботи, дозвілля та освіти. Бажанням більшості людей є вихід в Інтернет та залишатися онлайн постійно. Саме тому відбувається стрімкий розвиток індустрії мобільних застосунків в останні роки. Основною перевагою мобільних додатків є те, що користувач може отримати доступ до них, перебуваючи в будь-якому місці чи ситуації. На сьогоднішній день існують безліч мобільних додатків, які суттєво полегшують повсякденне життя. Одні слугують для встановлення з'єднання з мережею, чи пошуку магазинів зі знижками, та навіть замовлення доставки їжі додому, інші допомагають прокласти оптимальний маршрут до місця призначення, чи зорієнтуватися на місцевості. Люди купують товари, бронюють та замовляють послуги з використанням різноманітних мобільних застосунків. За основу кожного з таких застосунків лягли певні утиліти, що в результаті дозволяє швидко вирішувати поставлену задачу, економити час і досягати максимально комфортного рівня життя.

Однією з невирішених проблем сучасного суспільства була та є відмінність операційних систем на різних пристроях. Ще з далеких 80-90-х років попереднього століття почалась ворожнеча між операційними системами (ОС). Розробники не встигали за темпом розвитку кожної програмної гілки. Це все не задовольняло користувачів, які отримували контент частково: наприклад, хтось отримував одне програмне забезпечення на Android, інший — зовсім інакше на iOS. Деяких застосунків взагалі не можна було знайти десь окрім спеціальної операційної системи. Крім проблеми з аналогічним програмним забезпеченням для кожної ОС для звичайних користувачів, також потерпали розробники мобільних застосунків — не існувало однієї, цілком відповідної усім критеріям, технології розробки таких додатків. Наймалися різні люди для розробки під кожну опера-

ційну систему, дизайнери обходили проблемні місця кожної гілки. Різність була майже у всьому — від назви файлів до структури зберігання даних, навіть на серверах. Такий хаос також був у мобільному програмуванні до винаходження кросплатформенних технологій, які доступні на даний час.

На даний момент існує достатньо стабільних засобів, які вже показали життєздатність на ринку, щоб створювати гідні користування застосунки. Вони дозволяють дуже сильно знизити час та складність розробки додатків, що напряму позначається на вартості усього процесу. В цій роботі для дослідження було вибрано дві з цих технологій: React Native, Flutter. Для порівняння можливостей технологій було обрано дві основні операційні системи: Android та iOS.

Мета і завдання дослідження

Метою роботи є дослідження використання можливостей різних кросплатформенних технологій на прикладі створення застосунку ToDo на операційних системах Android та iOS, та дослідження їх характеристик, таких як:

- якість;
- користь;
- оптимізованість;
- зручність.

Відповідно до мети ставляться такі завдання:

1. Аналіз мов програмування для розробки кросплатформенних мобільних застосунків.
2. Аналіз архітектурних шаблонів для розробки мобільного застосунку.
3. Аналіз технологій обміну даними.
4. Аналіз розроблених програмних продуктів.
5. Оцінка використаних та досліджених підходів.

6. Розробка алгоритму роботи мобільного додатку кожної з операційних систем.
7. Створення кросплатформенного мобільного додатку.

Об’єкт дослідження

Кросплатформенний застосунок у стилі ToDo.

Предмет дослідження

Кількісні та якісні характеристики, а також зручність та оптимізованість, застосунка кожного з фреймворків та операційних систем.

Методи дослідження

Для розв’язання представлених завдань використовуються такі методи дослідження:

1. Аналіз джерел про технології створення кросплатформенних застосунків.
2. Проведення аналогії з-поміж існуючих на ринку систем створення кросплатформенних застосунків.
3. Синтез отриманих результатів досліджень.
4. Порівняльний аналіз кросплатформенних програмних продуктів.

Наукова новизна одержаних результатів

Під час дослідження було виявлено, що більшість мобільних клієнтських систем розроблюються під множину операційних систем, але майже усі з них не мають одного джерела коду, що призводить до проблем при розробці та оновленні програмного забезпечення. Технології, які є досліджені та порівняні між собою мають кросплатформенну новизну, тобто розроблюваний додаток має одне джерело коду, яке транслюється у багато операційних систем. Це покращує оптимізованість, масштабованість та зручність, зменшує витрати на розробку. Розроблений застосунок на цих платформах призваний полегшити роботу з колекціями та нагадуваннями

користувачу. Порівняння платформ призване полегшити вибір, яку саме платформу треба обрати для розробки проекту, щоб зекономити час та підвищите працездатність команди.

Практичне значення одержаних результатів

На базі отриманих результатів можна зрозуміти, які фреймворки та мови програмування є найбільш зручними для вирішення конкретних задач розробки кросплатформенних мобільних застосунків; ознайомитися з найпоширенішими проблемами, що виникають у процесі створення такого застосунку, та методами їх вирішення.

Апробація результатів кваліфікаційної роботи магістра

Результати роботи було представлено на науково-технічних конференціях студентів, магістрантів, аспірантів, молодих вчених викладачів [2] і опубліковані в збірнику наукових праць студентів, аспірантів і молодих вчених «Молода наука -2020» [1].

Глосарій

Application Programming Interface (API) — це обчислювальний інтерфейс, який визначає взаємодію між кількома посередниками програмного забезпечення [4]. Він визначає види дзвінків або запитів, які можна зробити, спосіб їх здійснення, формати даних, які слід використовувати, домовленості, яких слід дотримуватися тощо.

Back-end — програмно-адміністративна частина застосунку.

Callback — передача виконуваного коду в якості одного з параметрів іншого коду. Callback дозволяє в функції виконувати код, який задається в аргументах під час виклику. Цей код може бути визначений в інших контекстах програмного коду і бути недоступним для прямого виклику з цієї функції. Деякі алгоритмічні завдання в якості своїх вхідних даних мають не тільки числа або об'єкти, а й дії (алгоритми), які природним чином задаються як зворотні виклики.

Cloud Storage — це модель зберігання даних у комп'ютері, в якій цифрові дані зберігаються в логічні пули, а фізичне зберігання охоплює кілька серверів (зазвичай у кількох місцях). Фізичне середовище, як правило, належить хостинговим компаніям, вони ж і керують цим середовищем. Ці постачальники хмарних систем зберігання даних відповідають за зберігання наявної інформації та доступ до неї, а також за роботу фізичного середовища. Користувачі можуть купувати у постачальників послуг хмарного сховища можливість зберігати там дані.

Create, read, update and delete (CRUD) – це чотири основні функції постійного зберігання. CRUD також іноді використовується для опису правил користувацького інтерфейсу, які полегшують перегляд, пошук та зміну інформації, часто використовуючи комп’ютерні форми та звітів.

Framework — інфраструктура програмних рішень, що полегшує розробку складних систем [3]. Спрощено дану інфраструктуру можна вважати своєрідною комплексною бібліотекою. Вживається також слово «каркас», а деякі автори використовують його в якості основного, в тому числі не базуючись взагалі на англомовному аналогу.

Front-end — це інтерфейс для взаємодії між користувачем і back-end. Front-end та back-end можуть бути розподілені між однією або кількома системами.

JSON — це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті та може бути прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією).

Алгоритм — набір інструкцій, які описують порядок дій виконавця, щоб досягти результату розв'язання задачі за скінченну кількість дій; система правил виконання дискретного процесу, яка досягає поставленої мети за скінченний час. Для візуалізації алгоритмів часто використовують блок-схеми. Для комп'ютерних програм алгоритм є списком деталізованих інструкцій, що реалізують процес обчислення, який, починаючи з початко-

вого стану, відбувається через послідовність логічних станів, яка завершується кінцевим станом.

Автентифікація (authentication) — процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора. З позицій інформаційної безпеки автентифікація є частиною процедури надання доступу для роботи в інформаційній системі, наступною після ідентифікації і передує авторизації [6].

Авторизація (authorization) — керування рівнями та засобами доступу до певного захищеного ресурсу, як у фізичному розумінні (доступ до кімнати готелю за карткою), так і в галузі цифрових технологій (наприклад, автоматизована система контролю доступу) та ресурсів системи залежно від ідентифікатора і пароля користувача або надання певних повноважень (особі, програмі) на виконання деяких дій у системі обробки даних [6].

Бібліотека в програмуванні (library) — збірка підпрограм або об’єктів, що використовуються для розробки програмного забезпечення.

Графічний інтерфейс користувача (GUI, Graphical user interface) — тип інтерфейсу, який дозволяє користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації [5].

Дедлайн — це остаточний, затверджений термін здачі проекту. Також цим терміном позначають кінцеву дату або час, коли дія мала завершитися, інакше, якщо прострочити і заступити за цю межу, то дія вже не матиме сенсу.

Ідентифікатор (identificator) — ім'я об'єкта програми (змінної, масиву, структури, функції тощо), що дозволяє звернутись до об'єкта; ознака, яка цілком визначає сутність, в наперед визначеному просторі.

Клієнт в інформатиці (client) — апаратний або програмний компонент обчислювальної системи, який надсилає запити серверу.

Клієнт-серверна архітектура — є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними [7].

Ком'юніті (community) — різні інтернет-спільноти за інтересами: тематичні веб-форуми, блоги, групи.

Кросплатформенність (cross-platform) — властивість програмного забезпечення працювати більш ніж на одній програмній або апаратній платформі; технології; що дозволяють досягти такої властивості. Кросплатформенність дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого програмного забезпечення. Кросплатформмеими можна назвати більшість сучасних високорівневих мов програмування.

Купа пам'яті (*Memory heap*) – назва структури даних, за допомогою якої реалізована динамічно розподілювана пам'ять програми. Розмір купи — розмір пам'яті, виділеної операційною системою (ОС) для зберігання купи (під купу).

Мобільний застосунок (mobile application) — програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Багато мобільних застосунків встановлені на самому пристрої або можуть бути завантажені на нього з онлайн магазинів мобільних застосунків, таких як App Store, Google Play, Windows Phone Store та інших, безкоштовно або за плату. Спочатку мобільні застосунки використовувалися для швидкої перевірки електронної пошти, але їх високий попит призвів до розширення їх призначень і в інших областях, таких як ігри для мобільних телефонів, GPS, спілкування, перегляд відео та користування Інтернетом.

Нативний додаток (native application) — це прикладна програма, яка була розроблена для використання на певній платформі або на певному пристрої [9]. Термін «нативний додаток» часто згадується в контексті мобільного розробки, оскільки мобільне програмне забезпечення традиційно пишеться так, щоб програма могла працювати на певній апаратній

платформі. Оригінальний додаток встановлюється на мобільному пристрої безпосередньо виробником або може бути завантажений зі сторонніх джерел (таких як магазини додатків Google Play або App Store).

Облік — належним чином організована система збору, нагромадження, обробки, групування, узагальнення і реєстрації (фіксації) необхідної інформації або її сукупних даних, що відображають кількісну чи якісну характеристику подій, явищ, фактів, процесів, об'єктів тощо. Є важливою складовою соціального (в т. ч. державного) регулювання суспільних відносин та управління процесами (насамперед економічними) й відповідними об'єктами. Відіграє значну роль при прийнятті правових, економічних, управлінських та інших рішень повноважними органами або компетентними особами.

Обліковий запис (account) у комп'ютерних системах — сукупність наданої інформації про користувача, засобів та прав користувача відносно багатокористувацької системи.

Операційна система (operating system) — скорочено ОС — це базовий комплекс програм, що виконує керування апаратною складовою комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організовує взаємодію з користувачем. Операційна система звичайно складається з ядра операційної системи та базового набору прикладних програм.

Сервер або серверне програмне забезпечення в інформатиці (server) — програмний компонент обчислювальної системи, що виконує сервісні (обслуговуючі) функції за запитом клієнта, надаючи йому доступ до певних ресурсів або послуг.

Шаблони проектування програмного забезпечення (*software design patterns*) — ефектні способи вирішення задач проектування програмного забезпечення [8]. Шаблон не є закінченим зразком, який можна безпосередньо транслювати в програмний код. Об'єктно-орієнтований шаблон найчастіше є зразком вирішення проблеми і відображає відношення між кла-

сами та об'єктами, без вказівки на те, як буде зрештою реалізоване це відношення [8].

Утиліта (*utility*) — сервісна програма, що допомагає керувати файлами, отримувати інформацію про комп'ютер, діагностувати й усувати проблеми, забезпечувати ефективну роботу системи. Утиліти розширюють можливості ОС [12].

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ СТВОРЕННЯ КРОСПЛАТФОРМЕННИХ ЗАСТОСУНКІВ

1.1 Загальні відомості про кросплатформенні технології

Фреймворк (англ. Framework, каркас, платформа, структура, інфраструктура) — інфраструктура програмних рішень, що полегшує розробку складних систем. Спрощено дану інфраструктуру можна вважати своєрідною комплексною бібліотекою, але при цьому вона має ряд обмежень, що задають правила створення структури проекту та написання коду [10].

Кросплатформенність – властивість програмного забезпечення працювати більш ніж на одній програмній або апаратній платформі; технології; що дозволяють досягти такої властивості [11]. Кросплатформенність дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого програмного забезпечення. Кросплатформеними можна назвати більшість сучасних високорівневих мов програмування.

Дуже часто у сфері розробки програмного забезпечення ми стикаємося з проблемою кросплатформенності застосунків. Кожен застосунок повинен підтримувати усі операційні системи і, бажано, реалізовувати один і той же програмний (та стилістичний) інтерфейс. Хоча на даний момент існує безліч технологій, які дозволяють просто створювати кросплатформні додатки, багато програмних інженерів досі використовують операційно-спрямовані платформи, думаючи, що програють в оптимізації або в часі з новими технологіями.

Ця повсюдна омана виходить не із-за будь-яких недоліків технологій, а саме з консервативності самих програмістів. На даний час вже існує достатня кількість кросплатформенних технологій, які зайняли свою нішу на ринку і дозволяють як скоротити час на розробку програми, так і зберегти 100% продуктивність цільових мов.

1.2 Приклади задач розробки кросплатформенних застосунків

Приклад задач для вирішення яких використовується кросплатформенні технології:

1. Розробка кросплатформенних мобільних застосунків.
2. Розробка кросплатформенних десктопних застосунків.
3. Розробка кросплатформенних ігрових застосунків.
4. Розробка кросплатформенного програмного забезпечення з однаковими логікою та інтерфейсом у короткий проміжок часу зі збереженням рівня оптимізації.

1.3 Основні постановки задач розробки кросплатформенних застосунків

Кросплатформенність є однією з найфундаментальніших проблем розробки програмного забезпечення. Задача кросплатформенності має величезне практичне значення.

При створенні програмного забезпечення розробники часто мають ціллю декілька операційних систем, та використовують декілька вузконаправленних команд для розробки застосунку для кожної операційної системи з мовою програмування, яка підтримує лише цю ОС. У цих випадках замовник витрачає більше коштів на підтримку розробки, яка стає складнішою із-за подальшої комунікації між командами, дизайнерами та менеджерами. Така розробка часто закінчується невдачею.

Коли програмний інженер обирає окрему технологію, яка підтримує розробку під декілька операційних систем, вже на цьому етапі людей, якщо порівняти з шаблоном окремих команд під кожну ОС, стає в рази менше. Ми виграємо час та ресурси. Але кожна технологія має якісь недоліки. Тож перед тим, як ставити задачу та використовувати недосліджену технологію, було б добре дослідити усі окремі кросплатформенні технології

та їх процес створення застосунків, щоб при розробці нового застосунку шанс невдачі суттєво зменшився.

Наведемо деякі типові постановки задач створення кросплатформенних мобільних застосунків [18].

1. **Задача вибору технології**, яка полягає в тому, щоб обрати правильну кросплатформенну технологію, яка повністю задовільнить потреби розробки застосунку.

2. **Розробка програмного забезпечення на мові технології**. Кожна кросплатформенна технологія має основну мову програмування для подальшої інтерпретації її у нативні мови програмування.

3. **Аналіз застосунку на кожній операційній системі**, який полягає в тому, щоб макет замовника був повністю відтворений на усіх операційних системах однаково.

1.4 Кросплатформенні мобільні застосунки

Окремою гілкою треба виділити кросплатформенні мобільні застосунки. Різниця між основними з них, наприклад Android, iOS та Windows Phone, дуже велика. Кожна з технологій, що використовує інтерпретацію основної мови в нативні мови цих операційних систем, потребує детального дослідження та вивчення основних принципів тих чи інших аспектів цієї технології.

Деякі кросплатформенні технології потребують додаткових нативних знань з кожної технології окремо для того, щоб робити деякі дії, які не можливі усередині ядра кросплатформенної технології. Наприклад, деякі технології створення кросплатформенних мобільних застосунків потребують додаткового приєднання нативних модулів, шрифтів та інших активів, які будуть використані усередині коду програмної мови технології.

1.5 Огляд існуючих технологій створення кросплатформенних мобільних застосунків

Відокремимо основні кросплатформенні мобільні технології, які вже мають достатню кількість застосунків на ринку – це **Xamarin, Ionic, React Native** та **Flutter**.

1.5.1 Огляд Flutter

Flutter — це програмний каркас із відкритим кодом, для створення додатків для платформ Android та iOS, а також на вебі, розроблений компанією Google [21]. Він є основним способом створення додатків для Google Fuchsia. Перша версія Flutter носила назву «Sky» і працювала тільки під Android [24]. Вона була представлена в 2015 році на саміті розробників Dart із заявленою можливістю рендеринга 120 фреймів в секунду. 4 грудня 2018 року під час Flutter Live було оголошено про випуск першої стабільної версії 1.0 [22].

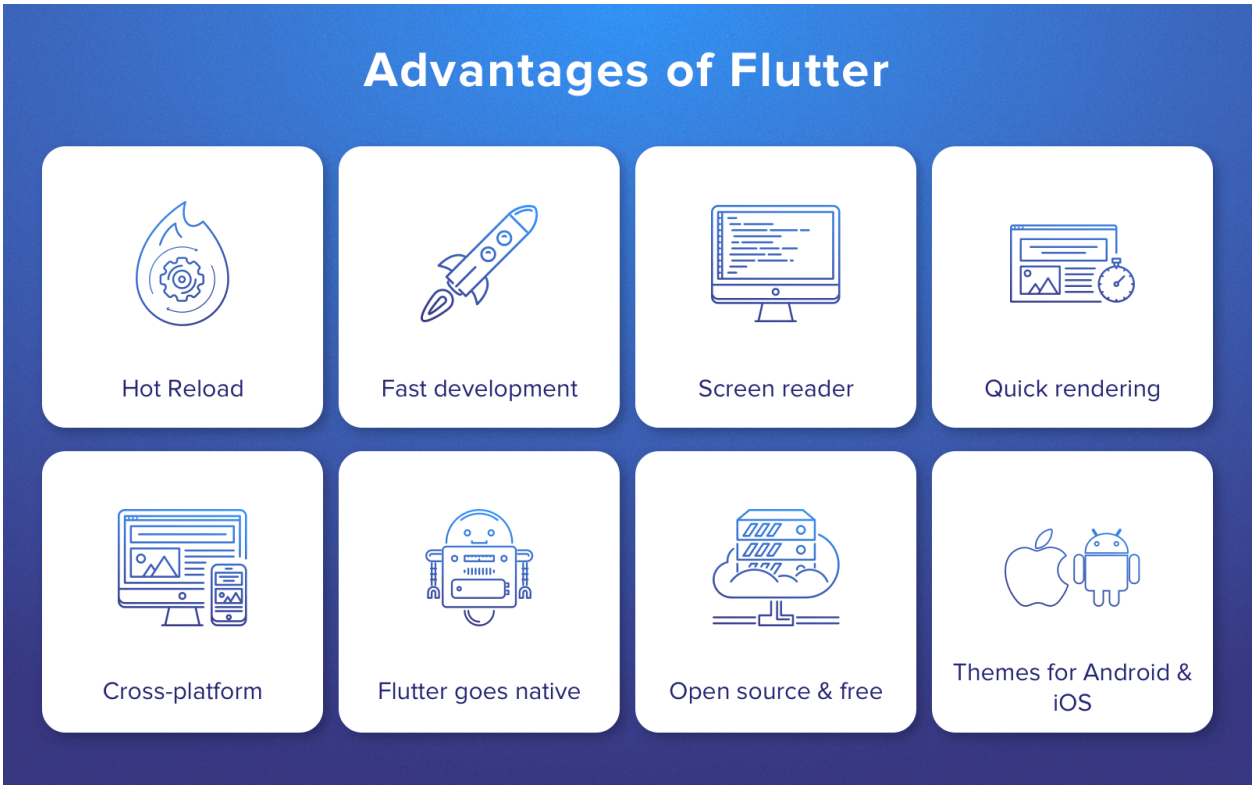

Рис. *1 Основні переваги Flutter*

Головні переваги [25, 26, 28]:

- можливість досягти 120 fps на смартфонах, де у екрана є така функція, за допомогою Dart-компонентів, які написані розробниками мови програмування Dart;
- підтримка декларативної архітектури, наприклад компоненти, які вкладені у інші компоненти, щоб досягнути більш гнучкої структури;
- дуже велика кількість інструментів розробника, що дозволяє тестувати застосунок та досліджувати помилки не витрачаючи на це багато часу;
- власний графічний движок (немає необхідності робити інтерфейс окремо для Android і iOS);
- програмний каркас із відкритим кодом;
- підтримує операційну систему Google Fuchsia;
- унікальна мова програмування – Dart;
- найновітніша платформа;
- має нескладну структуру;
- низький вхідний поріг.

Головні недоліки [25, 27]:

- кінцевий інсталяційний пакет більше, так як в нього додається віртуальна машина Dart;
- інтерфейс створюється за допомогою коду, через що грань між логікою і дизайном набагато тонше;
- бібліотек (і інформації) менше, ніж для нативної розробки;
- нестабільність.

1.5.2 Огляд React Native

React Native — це програмний каркас із відкритим кодом, для створення додатків для Android, iOS, Windows Phone, Linux, macOS або

Windows платформ [14]. Представлений компанією Facebook у 2015 році, що застосовує React архітектуру до нативних IOS, Android та UWP додатків. Принципи роботи React Native в основному такі ж, як ReactJS, за винятком того, що він не маніпулює DOM через VirtualDom. Він працює у фоновому процесі (який інтерпретує Javascript код написаний розробниками) безпосередньо на кінцевому пристрої і спілкується з нативною платформою. Очевидно, що Facebook виправив помилку, про яку Марк Цукерберг згадував в 2012 році. React Native взагалі не покладається на HTML, все написано на Javascript і залежить від нативних SDK [16].

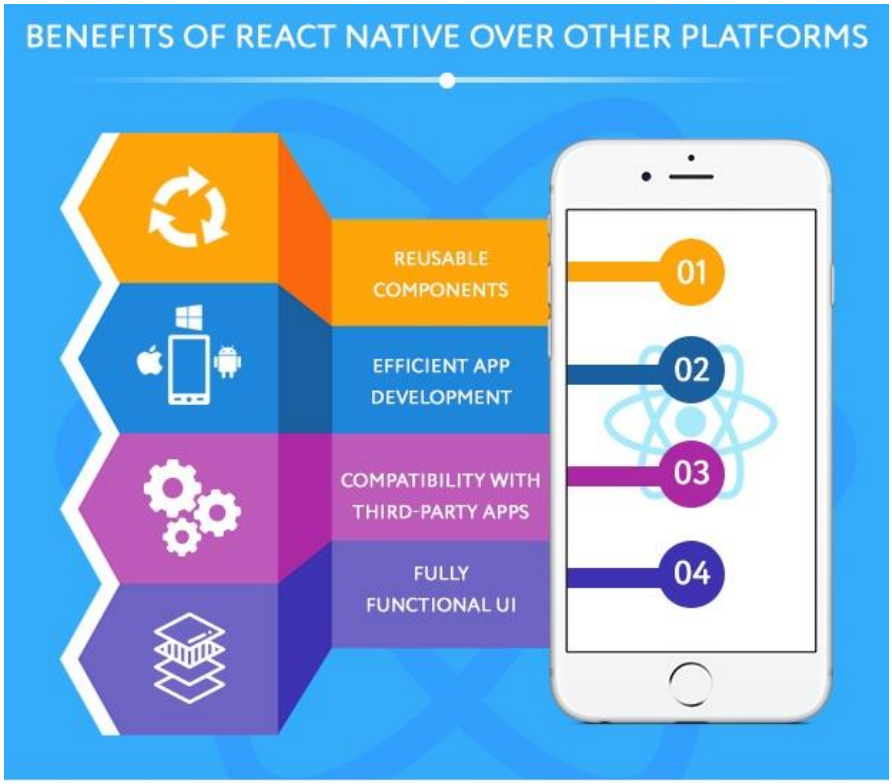

Рис. *2 Основні переваги React Native*

Головні переваги [14-18]:

- досягнення 60 кадрів у секунду за допомогою розділення JavaScript та UI потоків;
- використовує одну з найпоширеніших мов програмування – JavaScript;
- має підтримку TypeScript;

- лише симулює роботу CSS – насправді, це лише синтаксис;
- має багато бібліотек та внутрішніх кросплатформенних компонентів, які ідентично представлені у кожної окремої операційної системі;
- використовує React API;
- використовує JSX технологію замість HTML5;
- має нескладну структуру;
- середній вхідний поріг.

Головні недоліки [15, 19]:

- JavaScript UI потік не до кінця оптимізований при великій кількості даних.

1.5.3 Огляд Ionic

Ionic — це повна SDK з відкритим кодом для розробки гібридних мобільних додатків, створена Максом Лінчем, Бен Сперрі та Адамом Бредлі з Drifty Co. в 2013 році. Оригінальна версія була випущена в 2013 році та побудована на основі AngularJS та Apache Cordova. Однак остання версія була відновлена як набір веб-компонентів, що дозволяє користувачеві вибирати будь-які платформи інтерфейсу користувача, такі як Angular, React або Vue.js. Він також дозволяє використовувати компоненти Ionic без фреймворку користувальницького інтерфейсу. Ionic надає інструменти та послуги для розробки гібридних мобільних, настільних та прогресивних веб-додатків на основі сучасних технологій та практик веб-розробок, використовуючи такі веб-технології, як CSS, HTML5 та Sass. Зокрема, мобільні додатки можуть бути побудовані за допомогою цих веб-технологій, а потім розповсюджені через власні магазини додатків, які встановлюватимуться на пристрої, використовуючи Cordova або Capacitor [20].

Рис. *3 Основні компоненти Ionic*

Головні переваги [20]:

- створення базової програми з вибором шаблонів (наприклад додаток з боковим меню, додаток з табами, картами Google Maps, пустий додаток);
- збірка та запуск в емуляторі, на реальному пристрої чи в браузері;
- функція «Live Reload» в браузері і на пристрої;
- генерація іконок, екранів загрузки та інших елементів;
- найкраща кросплатформність ціною в оптимізованність (так як платформа не використовує нативний рендеринг).

Головні недоліки [20]:

- застаріла платформа.

- для правильної роботи деяких плагінів потрібна тонка настройка, наприклад глибинних посилань - модуль легко підключити, але складно змусити працювати як слід;
- незручність «збірки» навіть найпростішого застосунку;
- недостатня оптимізованість.

1.5.4 Огляд Xamarin

Xamarin – американська компанія в галузі розробки ПЗ. Займається розробкою і підтримкою Mono та інструментів для розробки застосунків мовою C# для iOS, Android, Windows, Mac. Була заснована в травні 2011 Міґелем де Ікаса і Нетом Фрідманом (Nat Friedman) після розподілу активів компанії Novell [23]. Станом на 2013 рік в компанії працювало близько 76 осіб. У лютому 2016 року куплена компанією Microsoft за неназваною ціною. У складі Microsoft робота Xamarin буде зосереджена на розвитку платформи для розробки мобільних застосувань на мові C# з використанням технологій .NET. Поєднання напрацювань Xamarin з продуктами Visual Studio, Visual Studio Team Services і Azure утворює рішення, яке охопить усі аспекти, необхідні для розробки, тестування і поширення мобільних застосувань для будь-яких категорій пристроїв, включаючи пристрої на платформах Android і iOS [23]. Xamarin стверджує, що єдиною IDE, яка дозволяє розробляти додатки для Android, iOS і Windows у Microsoft Visual Studio. Компанія Xamarin постачає додаткові модулі до Microsoft Visual Studio, що дозволяє розробникам створювати програми для Android, iOS та Windows у середовищі IDE за допомогою завершення коду та IntelliSense. Xamarin для Visual Studio також має розширення в Microsoft Visual Studio, які надають підтримку для створення, розгортання та налагодження програм на тренажері або пристрої. Наприкінці 2013 року компанія Xamarin і Microsoft оголосили про партнерство, яке включало подальшу технічну інтеграцію та програми для клієнтів, щоб вони могли створювати спільні бази розробників для всіх мобільних платформ. Крім

того, Xamarin тепер включає підтримку Microsoft Portable Class Libraries і більшість функцій C # 5.0, таких як async / await [29]. Генеральний директор і співзасновник компанії Xamarin, Nat Friedman, оголосив альянс на запуску Visual Studio 2013 у Нью-Йорку. 31 березня 2016 року Microsoft оголосила про те, що вони об'єднують все програмне забезпечення Xamarin з кожною версією Microsoft Visual Studio, включаючи Visual Studio Community, і це додало різноманітні можливості Xamarin, які були попередньо встановлені в Visual Studio, наприклад, емулятор iOS [7, 8].

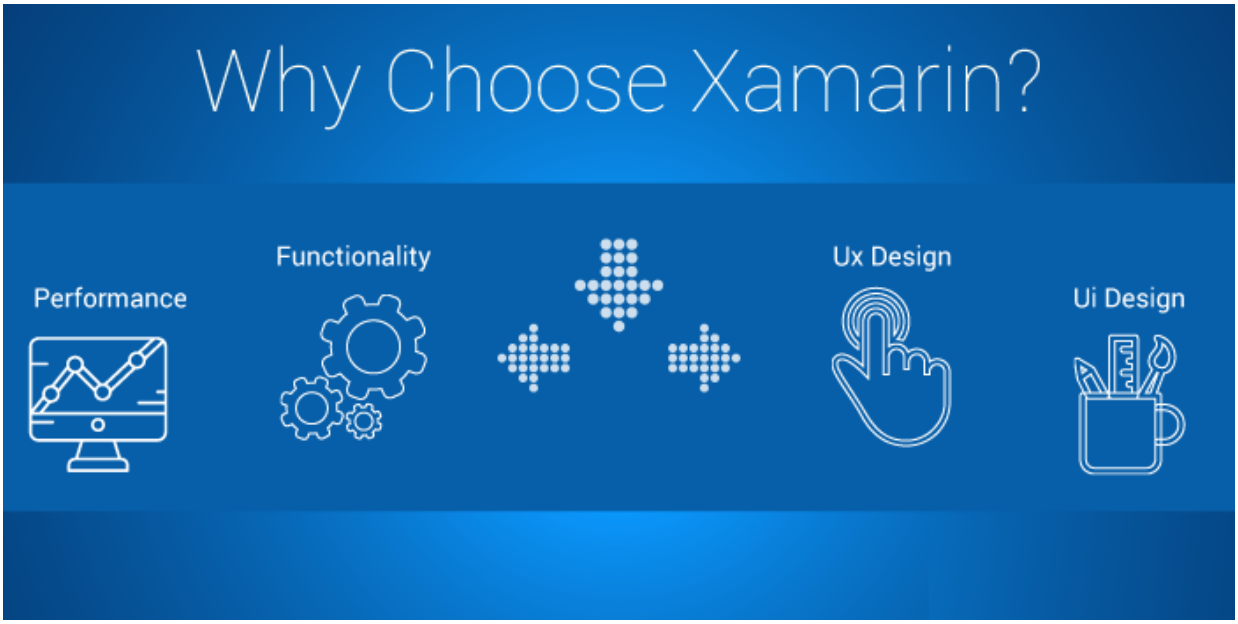

Рис. *4 Основні переваги Xamarin*

Головні переваги [23, 29]:

- ідеальна сумісність з Інтернетом Речей;
- оптимізованість до тестування;
- можливість створювати призначені для користувача інтерфейси, подібні «рідним»;
- просте освоєння.

Головні недоліки [23, 29]:

- застаріла платформа;

- затримки з оновленнями платформ;
- обмежений доступ до open-source бібліотек;
- обмеженість екосистеми;
- вимога до базових знань нативної мови програмування;
- не підходить для додатків з високопродуктивною графікою;
- значний розмір додатків;
- складнощі з інтеграцією.

1.6 Результати аналізу існуючих технологій

Згідно огляду існуючих технологій було вирішено обрати для дослідження найсучасніші з них: React Native та Flutter.

React Native демонструє стабільність на ринку вже декілька років, постійно оновлюючись та розвиваючись як платформа.

Flutter у свою чергу ще зовсім нова платформа, але вона встигла показати себе у різних напрямках створення оптимізованих кросплатформенних мобільних застосунків.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ КЛАСИФІКАЦІЇ ЯКОСТІ КРОСПЛАТФОРМЕННИХ МОБІЛЬНИХ ЗАСТОСУНКІВ

2.1 Класифікація якості на основі метрик програмного забезпечення

Для вимірювання характеристик якості використовують метрики. Метрика програмного забезпечення – це захід, що дозволяє отримати чисельне значення деякої властивості програмного забезпечення або його специфікацій. Оскільки кількісні методи добре зарекомендували себе в інших областях, багато теоретиків і практики інформатики намагалися перенести даний підхід і в розробку програмного забезпечення [13].

Метрика якості програм – система вимірювань якості програм. Вимірювання характеристик можна виконати об'єктивно і достовірно. Однак не слід виключати того, що оцінка якості ПЗ в цілому може бути пов'язана з суб'єктивною інтерпретацією одержуваних оцінок [13].

Залежно від характеристик і особливостей застосовуваних метрик їм ставляться у відповідність різні вимірювальні шкали [13]:

• номінальною шкалою відповідають метрики, що класифікують програми на типи за ознакою наявності або відсутності певної характеристики без урахування градацій;

• порядкової шкалою відповідають метрики, що дозволяють ранжувати деякі характеристики шляхом порівняння з опорними значеннями, тобто вимір по цій шкалі фактично визначає взаємне положення конкретних програм;

• інтервального шкалою відповідають метрики, які показують не тільки відносне положення програм, але і те, як далеко вони знаходяться один від одного;

• відносній шкалі відповідають метрики, що дозволяють не тільки розташувати програми певним чином і оцінити їх положення відносно

один одного, а й визначити, як далеко оцінки залежать від кордону, починаючи з якої характеристика може бути виміряна.

Всі метрики ПЗ поділяються на два класи [13]:

1. Метрики, що характеризують найбільш специфічні властивості програм, тобто метрики оцінки якості самого ПЗ.

2. Метрики оцінки технічних характеристик і чинників розробки програм, тобто метрики оцінки умов розробки програм.

В даний час у світовій практиці використовується кілька сотень метрик програм. Існуючі вимірювання якості програм можна згрупувати за шістьма напрямками [13]:

1. Вимірювання топологічної та інформаційної складності програм (виробляються за допомогою відповідних метрик і являють собою непрямі оцінки надійності).

2. Оцінки функціональної надійності програмних систем, що дозволяють прогнозувати прояв помилок у програмі (виробляються безпосередньо за допомогою моделей надійності).

3. Вимірювання продуктивності ПЗ і оцінки підвищення його ефективності шляхом виявлення помилок проектування.

4. Вимірювання рівня мовних засобів і оцінки їх застосування.

5. Вимірювання сприйняття і розуміння програмних текстів, орієнтовані на психологічні чинники, які є важливими для супроводу і модифікації програм.

6. Вимірювання продуктивності праці програмістів для прогнозування термінів розробки програм і планування робіт зі створення програмних комплексів.

2.2 Порівняння технологій на основі метрики кількості рядків коду

Кількість рядків коду (англ. Source Lines of Code — SLOC) — метрика програмного забезпечення, що використовується, щоб виміряти розмір комп'ютерної програми, рахуючи число рядків в тексті вихідного коду програми [20]. Дана метрика від початку розроблена для оцінки зусиль, докладених при розробці програмного забезпечення. Однак через те, що одна й та сама функціональність може бути розбита на декілька рядків або записана в один рядок, дана метрика стала неефективною з появою нових мов програмування, у яких в одному рядку можна записати більше однієї команди.

Розрізняють фізичні і логічні рядки коду. Логічні рядки коду — кількість команд програми. Фізичні рядки — кількість всіх рядків програми.

Для метрики SLOC існує велика кількість похідних, покликаних отримати окремі показники проекту, основними серед яких є [20]:

- число порожніх рядків;
- число рядків, що містять коментарі;
- відсоток коментарів (відношення рядків коду до рядків коментаря, похідна метрика стилістики);
- середнє число рядків для функцій (класів, файлів);
- середнє число рядків, що містять вихідний код для функцій (класів, файлів);
- середнє число рядків для модулів.

За допомогою кількості рядків коду у даному випадку кросплатформенних мобільних технологій можна зрозуміти, наскільки зручно використовувати той чи інший фреймворк програмістові. Якщо порахувати кількість написаних рядків коду у кожному роздільному файлі, який використовується, наприклад, для описання одного компонента чи екрану мобільного застосунку, то можна оцінити не тільки зручність, а ще й приблизний

час написання дизайнерського інтерфейсу у випадку, коли програміст ідеально розуміє, що треба написати – залишається тільки час написання самого коду на клавіатурі та саме розділення файлової архітектури.

2.3 Порівняння технологій за фізичними показниками

Кожний мобільний застосунок використовує ресурси пристрою, на якому запущений цей додаток. Але не тільки фізичні показники самого пристрою відіграють роль на тому, як буде себе вести застосунок. Окрім цього важливу роль грає оптимізованість збірки та сам принцип роботи фреймворку, який був використаний у розробці.

Коли платформа не є оптимізованою для роботи застосунків під кожну операційну систему, у користувачів починається багато невдоволень, наприклад, із-за того, що коли використовується додаток, дуже швидко виснажується заряд батареї.

Основні фізичні властивості, які впливають на пристрій та його роботу у застосунках:

1. Оперативна пам’ять – швидкодійна пам'ять, призначена для запису, зберігання та читання інформації у процесі її обробки. Кожен застосунок має максимальну відмітку оперативної пам’яті, яку він може використати у своїх цілях.
2. Процесор – основний компонент комп'ютера, призначений для керування всіма його пристроями та виконання арифметичних і логічних операцій над даними. Якщо застосунок одночасно буде виконувати декілька складних арифметичних задач чи отримувати та оброблювати дуже багато даних, то пристрій буде ставити застосунок у чергу, знижуючи на ці проміжки кількість кадрів у секунду до нуля.
3. Графічний адаптер – електронний пристрій, частина комп'ютера, призначена для генерації та обробки зображень з подальшим їхнім виведенням на екран периферійного пристрою. При застосун-

ках, які працюють з графікою та використовують графічний рендеринг відео чи зображень, значна частка навантаження буде приходитись на графічний адаптер.

Якщо ці фізичні частини навантажені некоректними діями застосунків та самою операційною системою, виникає недостача ресурсів пристрою, що безпосередньо впливає на частоту оновлення кадрів того чи іншого додатка – значно знижується частота оновлення кадрів на одну секунду, що у свою чергу викликає невдоволення користувача.

Щоб запобігти втечі ресурсів у кросплатформенних застосунках, треба ретельно досліджувати усі внутрішні та зовнішні модулі, умови використання та зокрема платформу та мову програмування, на яких розроблюється проектна система [11-14, 18, 24].

Саме тому дуже важливим критерієм оцінки є критерій фізичних показників кросплатформенного додатка на кожної з операційних систем для яких він є розроблюваним.

На новітніх кросплатформених технологіях на даний час дуже складним є питання заміру фізичних показників життєвого циклу застосунку. Але декілька з характеристик можна порівняти, якщо застосувати правильні бібліотеки та модулі, які позволяють програмістові заміряти ті чи інші фізичні показники.

2.3.1 Внутрішні інструменти розробника React Native

Для React Native платформи існує так званий «Монітор Продуктивності» (англ. «Performance Monitor»). Він надає програмістові багато інформації про даний стан застосунку [18, 19]:

- кількість зайнятої оперативної пам’яті (RAM-показник);
- кількість кадрів оновлення екрану на одну секунду (англ. «Frames Per Second»);
- кількість активних «Views» на даний час;

- графічне та числове відображення UI та JavaScript потоків.

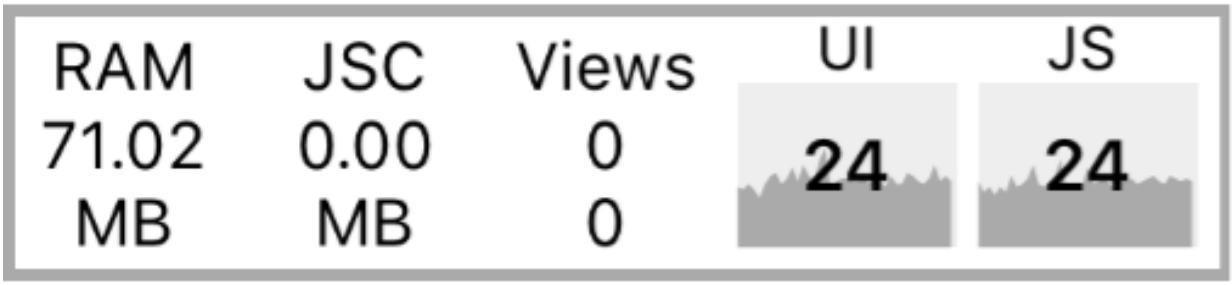

Рис. *5 Монітор продуктивності на React Native-застосунку*

2.3.2 Внутрішні інструменти розробника Flutter

Для мобільних застосунків мови Dart на платформі Flutter є набагато більше внутрішніх інструментів розробника [28, 30, 31]:

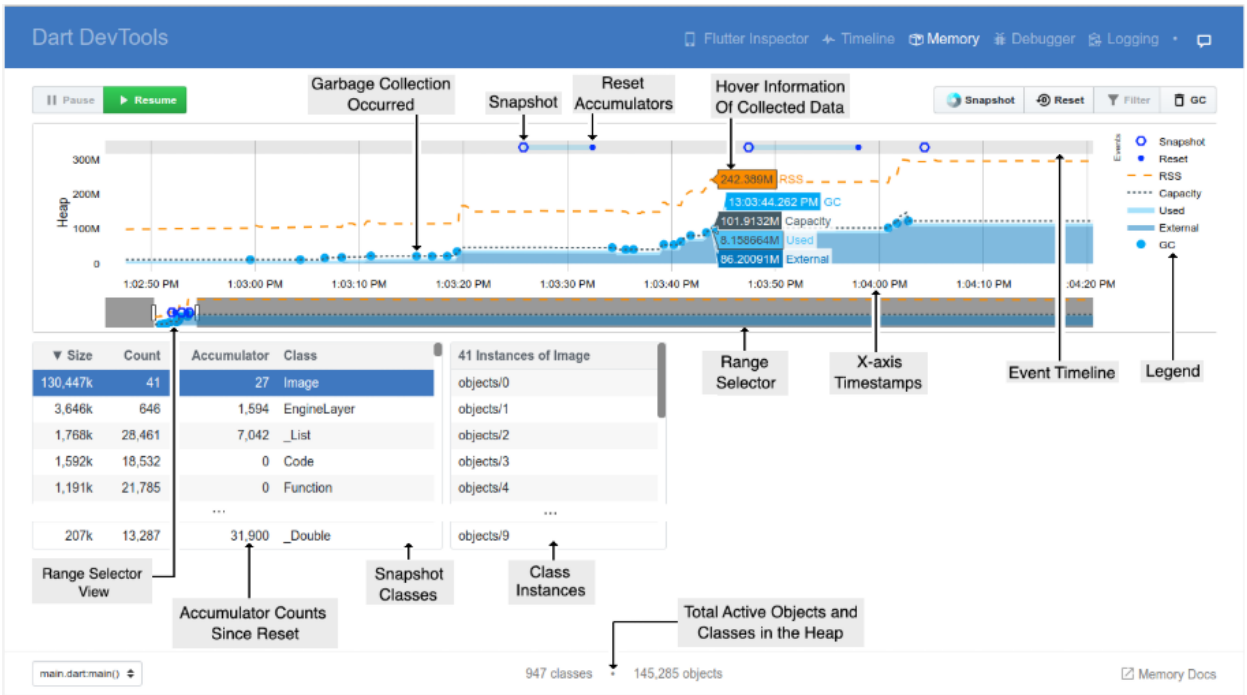

Рис. *6 Інструменти розробника Dart*

- легенда – усі зібрані дані про стан пам’яті; можна обирати деякі елементи для відображення тільки тої порції даних, які потребує розробник;

- селектор діапазону – всі зібрані дані пам'яті (англ. «timeseries»); найбільш лівий або перший час / дані (інформація про пам'ять) у селекторі - це коли запускається програма; найбільш правильний або останній час / дані - це інформація про постійну пам'ять, яка отримується (в прямому ефірі) до припинення роботи програми;
- перегляд селектора діапазону – детальний перегляд даних, зібраних для цього часового діапазону (не сіра зона);
- мітка часу осі X – час зібраної інформації пам’яті (ємність, використана, зовнішня, RSS (розмір постійного набору) та «Garbage Collector» (збирання сміття));
- наведена інформація о зібраних даних – у певний час (вісь x) детально зібрані дані пам'яті;
- мітка «Garbage Collector occurred» – інформація щодо зібрання сміття;
- хронологія подій – коли відбулася дія користувача (наприклад, натиснута кнопка "Знімок" або "Скинути");
- знімок – відобразить таблицю поточних активних об'єктів пам'яті;
- скидання «акумулятора» – скидання кожного поля колонки акумулятора до нуля;
- «всього активних об'єктів і класів у купі» – всього класів, виділених у купі, та загальних об'єктів (екземплярів) у купі;
- екземпляри класу – клацання класу в таблиці класу «Snapshot» відображає кількість активних примірників цього класу.

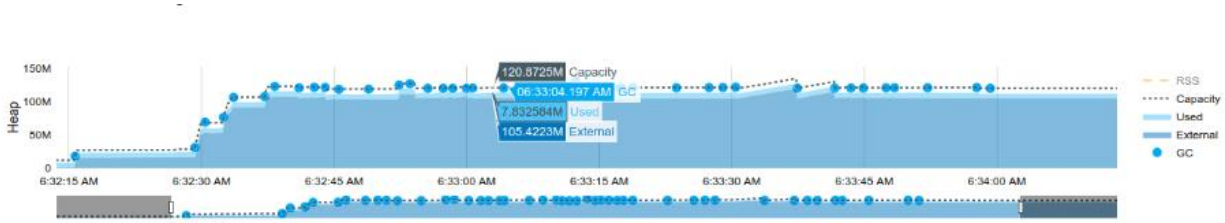

Рис. *7 Графічне відображення пам’яті у Dart DevTools*

Зокрема, Dart DevTools надають ще й можливість слідкування за процесором смартфона:

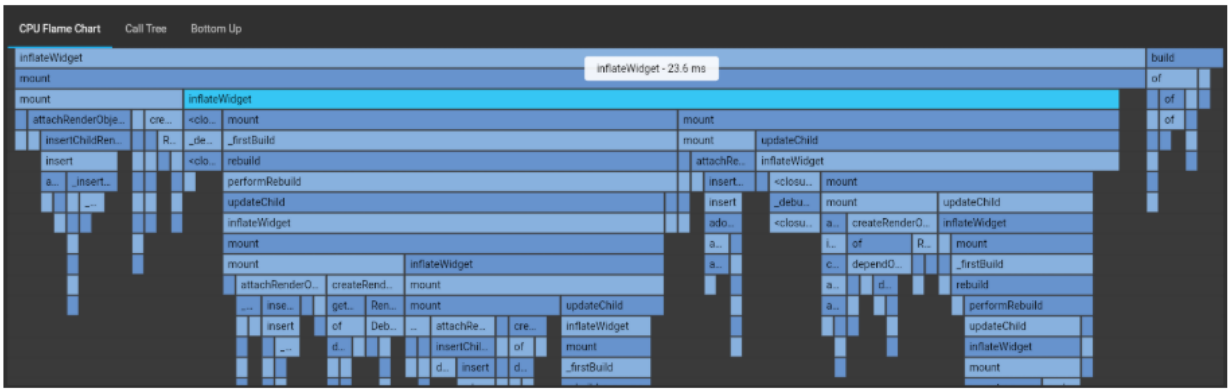

Рис. *8 Dart DevTools: CPU Flame Chart*

На цій вкладці профайлера відображаються зразки процесора за записану тривалість. Цю діаграму слід розглядати як слід стека зверху вниз, де найбільший кадр стека називає той, що знаходиться під ним. Ширина кожного кадру стека представляє кількість часу, який він витратив на процесор. Кадри стека, що займають багато процесорного часу, можуть стати хорошим місцем для пошуку можливих поліпшень продуктивності.

РОЗДІЛ 3 ПРОЕКТ ПРОГРАМНОЇ ЧАСТИНИ КРОСПЛАТФОРМЕНОГО МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Архітектура системи

Клієнтська частина додатка складається з двох платформ (Flutter та React Native), кожна з яких повинна реалізовувати однаковий графічний інтерфейс для Android та iOS операційних систем.

Самостійно додаток повинен реалізовувати окрему навігацію, для спрощення сприйняття та кращого інтерфейсу для користувача, та зв’язок з хмарним сховищем даним:

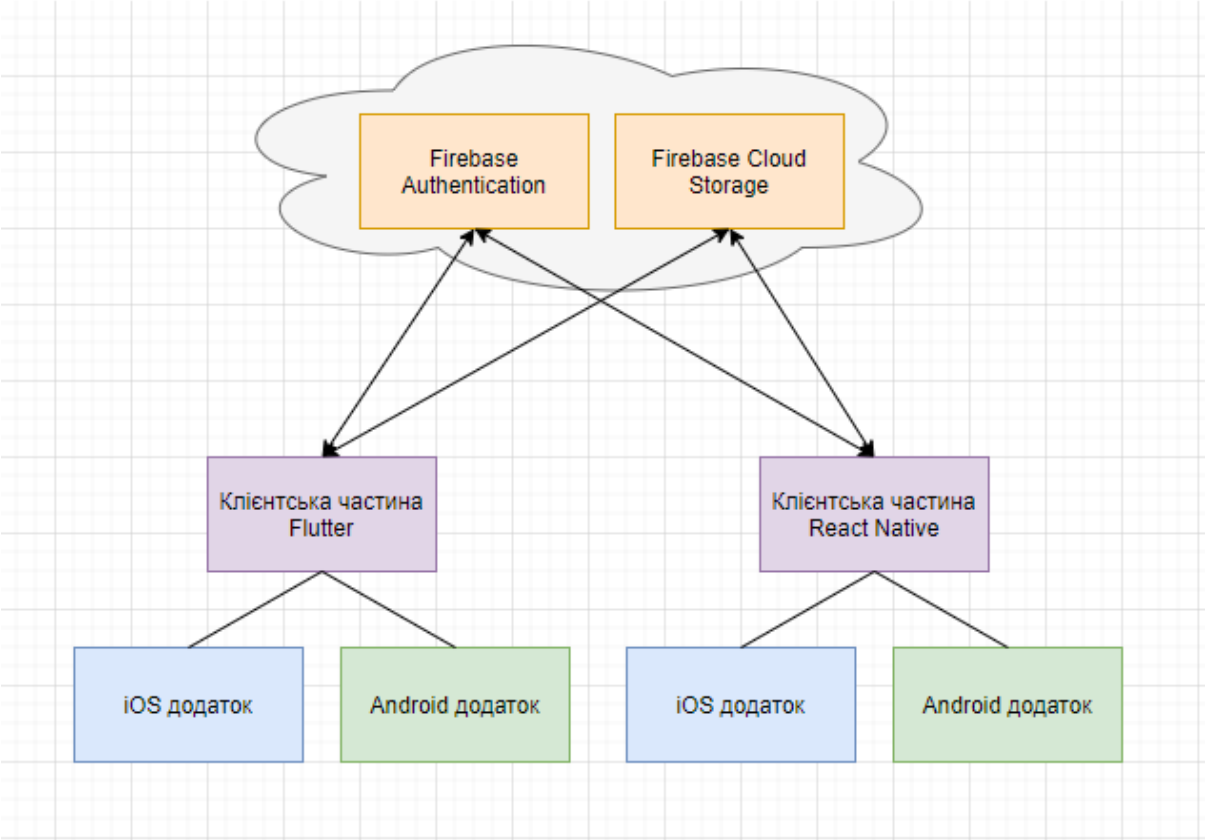

Рис. *9 Архітектура системи*

Застосунок на кросплатформенних технологіях може бути написаний в будь-якому стилі з використанням підключених модулей для навіга-

ції, графічного відображення, анімації тощо. Завдяки кросплатформеним властивостям, модулі можуть бути використані в будь-якому середовищі.

3.1.1 Архітектура Flutter застосунку

Структура будь-якого застосунку Flutter виглядає наступним чином: всередині основної директорії застосунку є поділення на декілька нативних директорій та основну директорію, де починає свій старт програма. У цій директорії є основний файл мови Dart, який запускається з початку застосунку та знаходиться у директорії **lib** під іменем **main.dart**. Будь-які інші компоненти системи повинні імпортуватися, як об'єктно-орієнтовані класи, які всередині мови Dart називаються віджетами. Зберігання інших dart-файлів також знаходяться поряд **main.dart** файлом, для зручності поміщених у вкладені директорії, такі як **screens/**, **components/**, **utils/**, **constants/** і т. д.

Кожен розділений файл втілює якусь частину програми вцілому. Взаємодія між файлами налаштовується крізь імпорт джерел, які потрібні.

```
1   import 'dart:async';
2
3   import 'package:deity_flexion_app/components/default_progress_indicator.dart';
4   import 'package:deity_flexion_app/components/line.dart';
5   import 'package:deity_flexion_app/components/notes_list_view.dart';
6   import 'package:deity_flexion_app/constants.dart';
7   import 'package:deity_flexion_app/data/note.dart';
8   import 'package:deity_flexion_app/screens/notes_add.dart';
9   import 'package:deity_flexion_app/screens/notes_edit.dart';
10  import 'package:deity_flexion_app/services/firebase_auth.dart';
11  import 'package:deity_flexion_app/services/firebase_storage.dart';
12  import 'package:firebase_auth/firebase_auth.dart';
13  import 'package:flutter/material.dart';
14  import 'package:modal_progress_hud/modal_progress_hud.dart';
15
```

Рис. *10 Приклад імпорту інших джерел у мові Dart*

Окрім основного файла та файлів, створенних корисувачем, є ще файли конфігурації. Саме там Flutter бере інформацію щодо ім'я застосунку, його релізної версії, під'єднаних модулів, шрифтів та

зображень, які приєднав програміст. Усі ресурси зберігаються у директорії **assets/**.

Натівні директорії (з назвою **android/** та **ios/**), хоча й зовсім рідко, використовуються для підключення нативних бібліотек (якщо Flutter не може цього зробити за вас) та інших нативних опцій, які дуже часто потрібні при розробці дуже спеціфічного програмного забезпечення. Для розробки звичайного користувацького інтерфейсу не потрібно працювати напряму з нативними мовами програмування.

3.1.2 Архітектура React Native застосунку

Структура React Native може декілька відрізнятися, але основних способів реалізації структури проекта два:

1. З обгортанням у Expo-модуль, робота з нативними директоріями через спеціальну обгортку.
2. Без Expo-обгортки, робота напряму с нативними директоріями.

Expo-додаток

Обгортка Expo була створена у ранніх релізах фреймворка. Основною ціллю було спростити або зовсім прибрати роботу з нативною частиною. Структура React Native застосунку у такому випадку виглядає наступним чином:

- одна директорія **.expo/**, у якій зібрана уся робота з нативною частиною, усі модулі інтеграції до фреймворка (робота з навігацією, аудіо, анімацією та іншим);
- одна директорія **source/**, всередині якої зібрані усі файли основні виконавчі файли JavaScript (або TypeScript);
- файли конфігурації різних доповнень (модулів та розширень) та гіт конфігурація до React Native застосунку, які знаходяться у корні проекту;

- основний виконавчий файл, **index**, з якого починається виконання програми.

Для запуску Expo-застосунок потребує встановлений емулятор мобільного пристрою потрібної платформи та починає запускатися за допомогою **expo-cli** команд.

Expo передбачає чітко обмежений незмінний набір модулів, які можна використовувати як бібліотеки, для спрощення роботи та зменшення кількості коду. Місце знаходження модулів: директорія **node_modules/**, яка з'являється та кешується після першого встановлення застосунку. Будь-яка нативна конфігурація є непередбачованою, доступу до редагування та написання нативних модулів немає.

На відміну від React Native додатку без Expo-обгортки, для запуску на реальному мобільному пристрої Expo запускає сервер, під'єднаний к поточному Wi-Fi роутеру, використовуючи локальну IP адресу, та відображає QR-код, скануючи який, програміст потрапляє до Expo-android чи Expo-ios застосунків та переходить до виконання коду. Обов'язковою умовою є те, що пристрій під'єднаний саме до тієї ж локальної мережі, що й джерело запуску. Якщо на пристрої не встановлений Expo-додаток, то користувач переходить до магазину застосунків, на окрему сторінку Expo-застосунка для його подальшого встановлення.

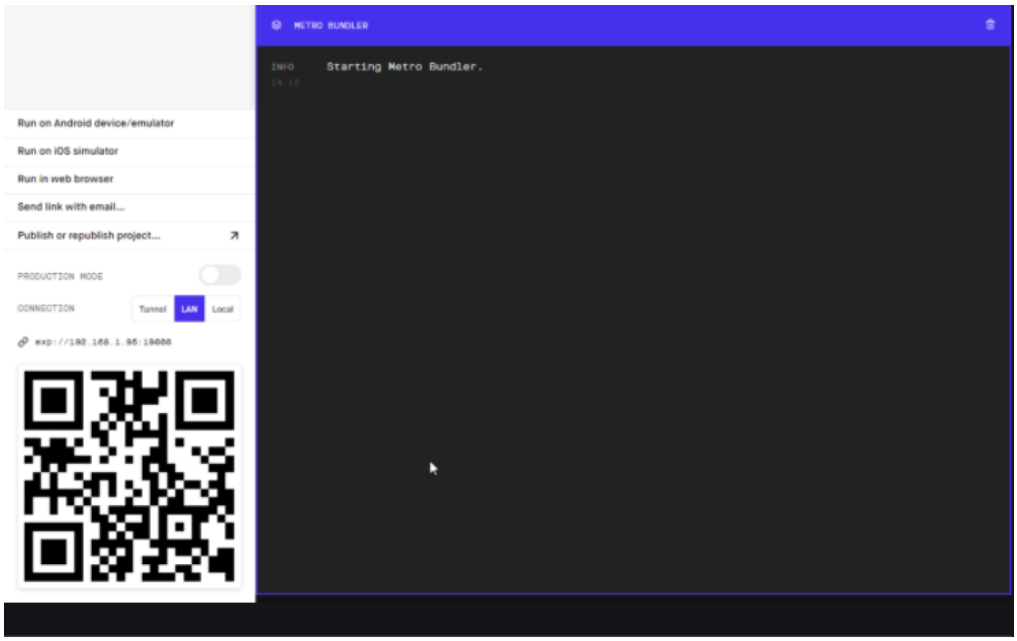

Рис. *11 Приклад запуску Expo-сервера зі згенерованим QR-кодом*

Native-додаток

Структура React Native застосунку у цьому випадку виглядає наступним чином:

- дві нативні директорії **android/** та **ios/**, для Android та iOS операційних систем відповідно, які відображають кінцевий стан інтерпретації головного коду (після оновлення основної директорії – нативні теж оновлюються головним сервером, якщо він запущений);
- одна директорія **source/**, всередині якої зібрані усі файли основні виконавчі файли JavaScript (або TypeScript);
- файли конфігурації різних доповнень (модулів та розширень) та гіт конфігурація до React Native застосунку, які знаходяться у корні проекту;
- основний виконавчий файл, **index**, з якого починається виконання програми.

Звичайний запуск на емуляторі мобільного пристрою відбувається за допомогою **react-native-cli** та пакет-менеджерів, по типу **NPM** та **Yarn**, які

запускають локальний сервер та реалізують обробку головного JavaScript-джерела у нативні мови програмування.

Дуже зручно оброблювати кожну із нативних директорій у відокремленій для цієї мети IDE: **AndroidStudio** та **xCode**. Там є можливість запустити пристрій-емулятор разом із сервером одним натиском миші. Для цього треба запустити проекти не з головної директорії, а з нативних, та натиснути кнопку з підписом/іконкою «Run».

На відміну від додатку з Expo-обгорткою, запуск виконавчого коду на реальному пристрої можливий лише за умови під'єднання цього пристрою до комп'ютера, де встановлений проект. За допомоги спеціальних команд потрібно запустити головний сервер та вказати підключенний пристрій у параметрах команди.

Будь-яка нативна конфігурація підтримується, є також можливість написання додаткових модулів на нативних мовах програмування.

На відміну від додатку з Expo-обгорткою, не містить модулів за замовчуванням, потребує детального встановлення та налаштування кожного з обраних для розробки NPM модулів.

3.1.3 Реалізація архітектури клієнтської частини

Було обрано реалізовувати стандартну конфігурацію для Flutter-застосунку та нативну конфігурацію для React Native (з більш великими можливостями та доступом до нативних директорій напряму).

Для розробки був обраний простий та зрозумілий для користувача інтерфейс ToDo-застосунка, з можливістю реєстрації, логіну користувачів та збереженням даним у хмарному сервісі. До отримання та редагування даних користувач може потрапити з головної сторінки, зайшовши у свій профіль та натиснувши на одну з двох категорій: «Завдання» та «Записки». У кожній з категорій користувач може переглядати, додавати, редагувати та видаляти дані.

3.2 Засоби реалізації

Під час розробки програмної системи, через низку переваг, було вирішено використовувати наступні засоби:

1. Мова програмування JavaScript.
2. Мова програмування Dart.
3. Мова програмування TypeScript.
4. Середовище розробки Visual Studio Code.
5. Середовище розробки Android Studio.
6. Середовище розробки Xcode.
7. Хмарний сервіс Firebase.
8. Локальне сховище даних Redux.
9. Бібліотека Redux-Saga.
10. Бібліотека для зберігання кешу Persist.
11. Менеджери пакетів Node Package Manager та Yarn.
12. Система контролю версій Git.
13. Середовище для відображення стану Git SourceTree.

3.2.1 Мова програмування JavaScript

JavaScript (вимовляється ДжаваСкрипт) — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript [15]. Найчастіше використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки [6].

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією [8]. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація,

автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

JavaScript, наразі, є однією з найпопулярніших мов програмування в інтернеті. Але спочатку багато професійних програмістів скептично ставилися до мови, цільова аудиторія якої складалася з програмістів-любителів [12]. Поява AJAX змінила ситуацію та повернула увагу професійної спільноти до мови, а подальші модифікації мови за стандартами ES2015 та ES2017 внесли багато корисних можливостей, яких не вистачало для ефективного програмування. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза браузером.

JavaScript має C-подібний синтаксис, але в порівнянні з мовою C має такі корінні відмінності [11, 17, 20]:

- об'єкти, з можливістю інтроспекції і динамічної зміни типу через механізм прототипів;
- функції як об'єкти першого класу;
- обробка вийнятків;
- автоматичне приведення типів;
- автоматичне прибирання сміття;
- анонимні функції.

JavaScript є основною мовою для розробки кросплатформених додатків на платформі React Native.

3.2.2 Мова програмування Dart

Dart — мова програмування, яку розробляє компанія Google, позиціонуючи як мову структурованого програмування для Веб [24]. Розробники вважали, що в довгостроковій перспективі Dart може стати прогресивною заміною JavaScript [25], котрий потерпає від наявних в даний час проблем

з розширюваністю, продуктивністю і підтримкою розробки складних застосунків [24]. Мова має схожий на Java синтаксис, не вимагає явного визначення типів і її можна використовувати для створення серверних та клієнтських застосунків.

У березні 2015 компанія Google представила оновлену стратегію просування Dart, у котрій вирішено не прив'язувати Dart до браузеру і відмовитися від ідеї інтеграції віртуальної машини Dart у Chrome. Розробку буде зосереджено на застосуванні Dart як проміжної мови, скомпільованої в JavaScript. Розвиток Dart як окремої мови, альтернативної JavaScript і безпосередньо підтримуваної у браузерах, визнано недоцільним. Замість цього Dart рухатиметься у бік якіснішої інтеграції з JavaScript і генерації оптимального JavaScript-коду. При цьому розробку віртуальної машини Dart VM буде продовжено, але вона позиціонуватиметься в основному для створення серверних і мобільних застосунків [21].

Мова має схожий на Java синтаксис, не вимагає явного визначення типів і може використовуватися для створення серверних і клієнтських застосунків. Для запуску всередині браузера код мовою Dart може бути перетворений в JavaScript-подання або запущений безпосередньо під управлінням спеціального JavaScript-інтерпретатора Dartboard [20].

Мова підходить як для розробки одним програмістом невеликих скриптів без жорсткої структури, так і для створення високо масштабованих великих модульних проектів, підтримуваних великим колективом з потребою більш явної типізації для того, щоб уникнути плутанини і помилок. При цьому явне задання типів не обов'язкове, наприклад, можна почати розробку без вказання типів, а надалі при необхідності додати їх (наприклад, спочатку написати "var x", а потім замінити на "num x"). У кожному скрипті використовується власний простір імен, для використання зовнішніх об'єктів, функцій або змінних слід їх явно імпортувати за допомогою конструкції "import". Всі змінні, початково, діють тільки в межах поточного скрипту і не експортуються глобально.

Dart є основною мовою програмування для розробки кросплатформених додатків на платформі Flutter.

3.2.3 Мова програмування TypeScript

TypeScript — мова програмування, представлена Microsoft восени 2012; позиціонується як засіб розробки веб-застосунків, що розширює можливості JavaScript [13]. Розробником мови TypeScript є Андерс Гейлсберг (англ. Anders Hejlsberg), який створив раніше C#, Turbo Pascal і Delphi. TypeScript є зворотньо сумісним з JavaScript. Фактично, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверною платформою Node.js.

Переваги над JavaScript

1. Можливість явного визначення типів (статична типізація) [13].
2. Підтримка використання повноцінних класів (як в традиційних об'єктно-орієнтованих мовах) [13].
3. Підтримка підключення модулів [13].

За задумом ці нововведення мають підвищити швидкість розробки, читабельність, рефакторинг і повторне використання коду, здійснювати пошук помилок на етапі розробки та компіляції, а також швидкодію програм.

Планується, що в силу повної зворотної сумісності адаптація наявних застосунків на нову мову програмування може відбуватися поетапно, шляхом поступового визначення типів. Підтримка динамічної типізації зберігається — компілятор TypeScript успішно обробить і не модифікований код на JavaScript.

Основний принцип мови — будь-який код на JavaScript сумісний з TypeScript, тобто в програмах на TypeScript можна використовувати стандартні JavaScript-бібліотеки і раніше створені напрацювання. Більш того, можна залишити наявні JavaScript-проекти в незмінному вигляді, а дані

про типізації розмістити у вигляді анотацій, які можна помістити в окремі файли, які не заважатимуть розробці і прямому використанню проекту (наприклад, подібний підхід зручний при розробці JavaScript-бібліотек).

На момент релізу представлені файли для сприйняття розширеного синтаксису TypeScript для Vim і Emacs, а також плагін для Microsoft Visual Studio.

Одночасно з виходом специфікації розробники підготували файли з деклараціями статичних типів для деяких популярних JavaScript-бібліотек, серед яких jQuery.

3.2.4 Середовище розробки Visual Studio Code

Visual Studio Code — засіб для створення, редагування та зневадження (налаштування) сучасних веб-застосунків і програм для хмарних систем [16]. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X [16].

Компанія Microsoft представила Visual Studio Code у квітні 2015 на конференції Build 2015. Це середовище розробки стало першим кросплатформовим продуктом у лінійці Visual Studio.

За основу для Visual Studio Code використовуються напрацювання вільного проекту Atom, що розвивається компанією GitHub [14]. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовують браузерний рушій Chromium і Node.js. Примітно, що про використання напрацювань вільного проекту Atom на сайті Visual Studio Code і в прес-релізі і в офіційному блозі не згадується.

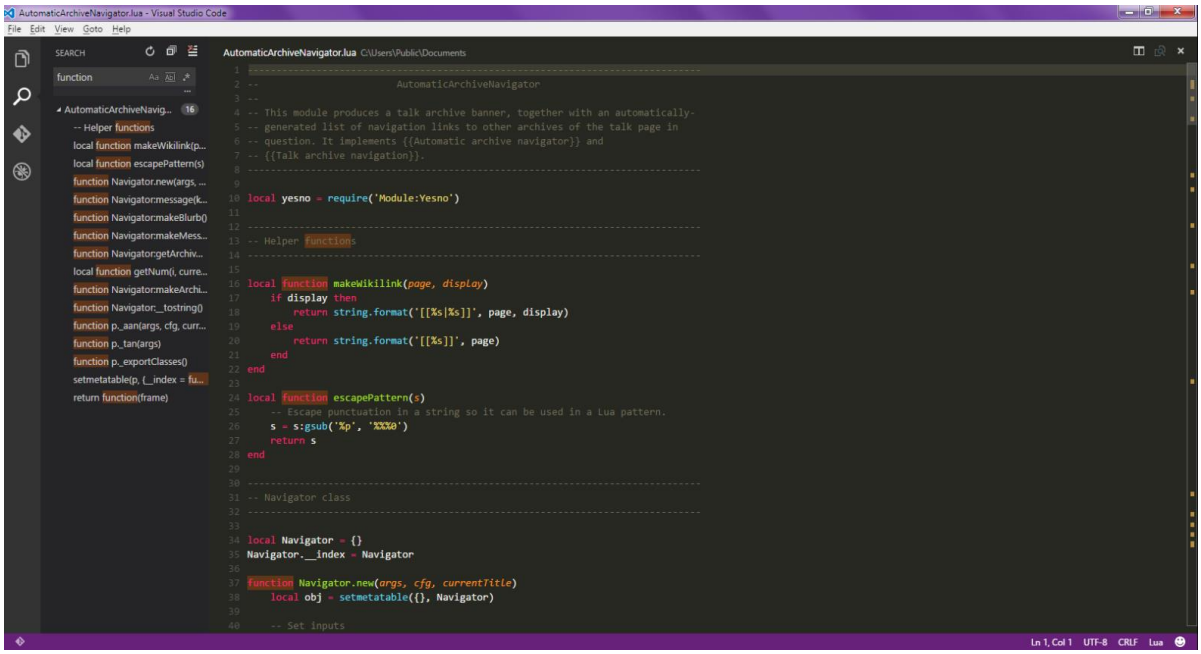

Рис. *12 Середовище розробки Visual Studio Code*

Редактор містить вбудований зневаджувач, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як легковагове рішення, що дозволяє обійтися без повного інтегрованого середовища розробки. Серед підтримуваних мов і технологій: JavaScript, C++, C#, TypeScript, jade, PHP, Python, XML, Batch, F#, DockerFile, Coffee Script, Java, HandleBars, R, Objective-C, PowerShell, Luna, Visual Basic, Markdown, JSON, HTML, CSS, LESS і SASS, Haxe.

3.2.5 Середовище розробки Android Studio

Android Studio — інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня 2013 року на конференції Google I/O [28].

Android Studio прийшло на зміну плагіну ADT для платформи Eclipse [29]. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, що розвивається компанією JetBrains.

Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0.

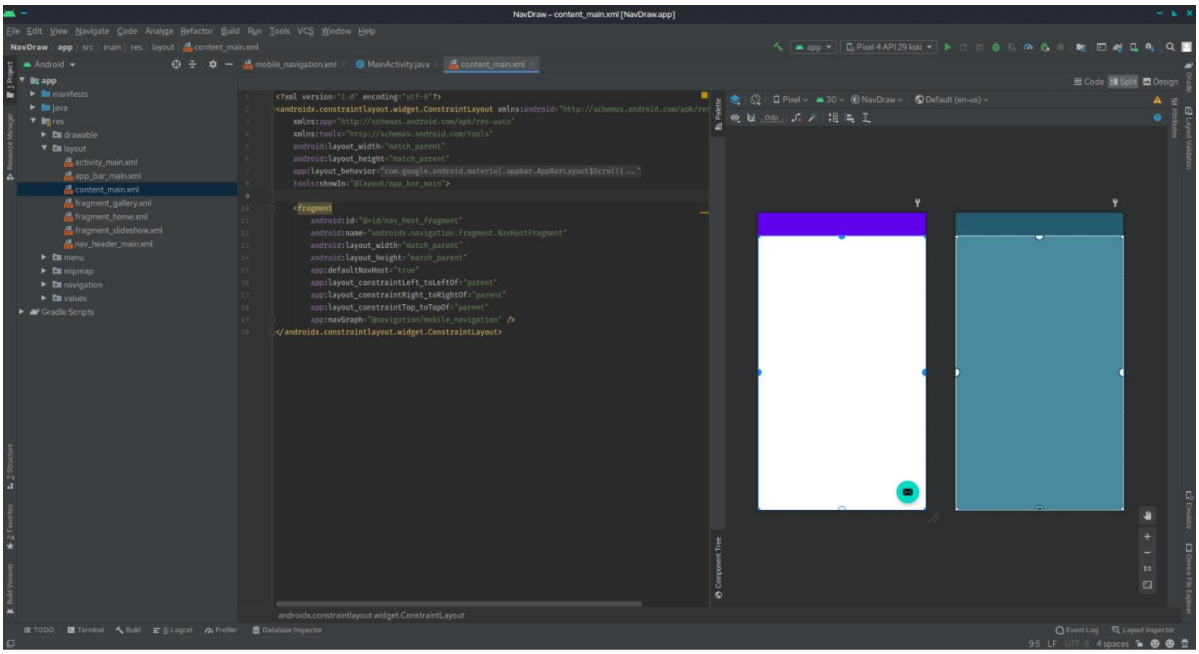

Рис. *13 Середовище розробки Android Studio*

Бінарні складання підготовлені для Linux (для тестування використаний Ubuntu), Mac OS X і Windows. Середовище надає засоби для розробки застосунків не тільки для смартфонів і планшетів, але і для носимих пристроїв на базі Android Wear, телевізорів (Android TV), окулярів Google Glass і автомобільних інформаційно-розважальних систем (Android Auto). Для застосунків, спочатку розроблених з використанням Eclipse і ADT Plugin, підготовлений інструмент для автоматичного імпорту існуючого проекту в Android Studio [31].

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android. У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано

кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції.

Для прискорення розробки застосунків представлена колекція типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану). Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів. У середовище вбудовані функції завантаження типових прикладів коду з GitHub.

До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання. У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard. Вбудовані засоби генерації цифрових підписів. Надано інтерфейс для управління перекладами на інші мови.

3.2.6 Середовище розробки Xcode

Xcode — це інтегроване середовище розробки (IDE) для macOS, що містить набір засобів розробки програмного забезпечення, розроблених Apple для розробки програмного забезпечення для macOS, iOS, iPadOS, watchOS та tvOS [17]. Вперше він вийшов у 2003 році; останній стабільний випуск — версія 12.1, випущена 20 жовтня 2020 року, і доступна через Mac App Store безкоштовно для користувачів macOS Catalina. Зареєстровані розробники можуть завантажувати попередні версії та

попередні версії пакету через веб-сайт Apple Developer. Xcode включає інструменти командного рядка (Command Line Tools), які дозволяють розробляти стиль UNIX за допомогою програми Terminal у macOS. Їх також можна завантажити та встановити без основної IDE [17].

Xcode підтримує вихідний код для мов програмування C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit (Rez) та Swift, з різноманітними моделями програмування, включаючи, але не обмежуючись, Cocoa, Carbon та Java. Треті сторони додали підтримку GNU Pascal, Free Pascal, Ada, C#, Go, Perl та D [19].

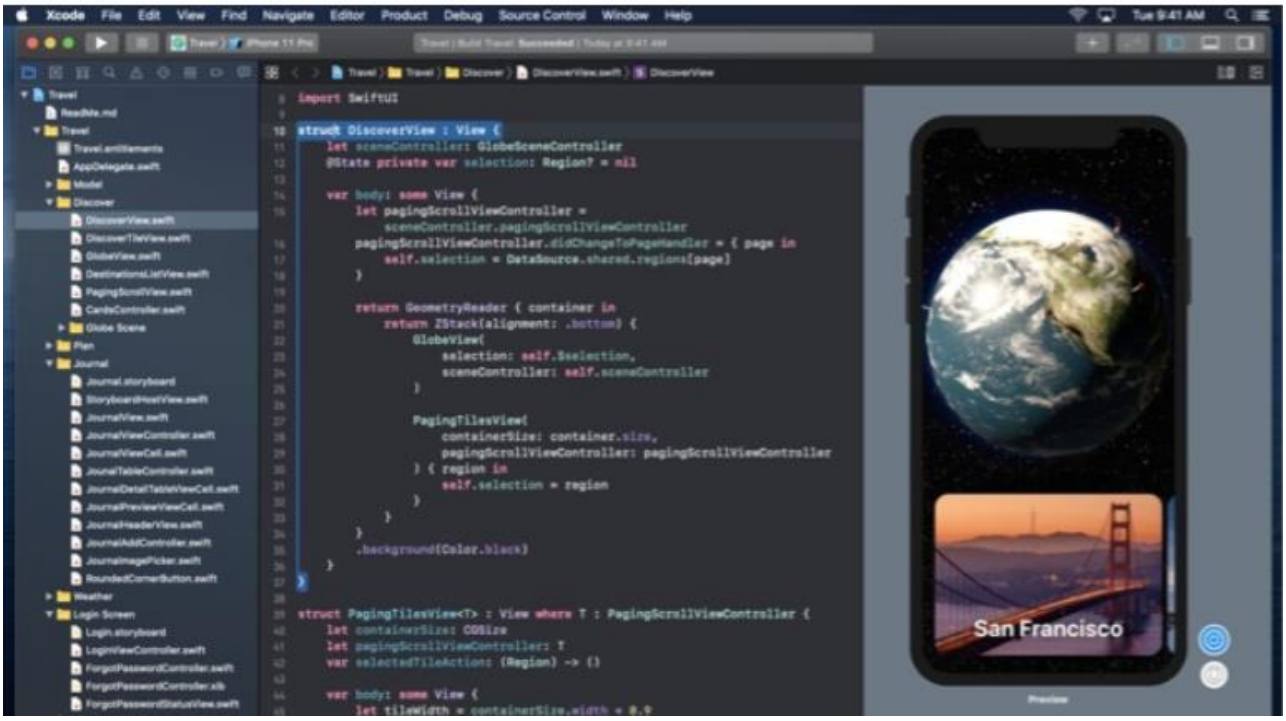

Рис. *14 Середовище розробки Xcode*

Xcode може створювати багатооб’ємні двійкові файли, що містять код для декількох архітектур із виконуваним форматом Mach-O. Вони називаються універсальними двійковими файлами, що дозволяють програмному забезпеченню працювати як на платформах PowerPC, так і на базі Intel (x86), і вони можуть включати як 32-розрядний, так і 64-розрядний код для обох архітектур. Використовуючи iOS SDK, Xcode

можна також використовувати для компіляції та налагодження програм для iOS, які працюють на процесорах архітектури ARM [11].

Xcode також інтегрує вбудовану підтримку управління вихідним кодом за допомогою системи управління версіями Git та протоколу, що дозволяє користувачеві створювати та клонувати сховища Git (які можуть розміщуватися на сайтах, що розміщують сховища вихідних кодів, таких як GitHub, Bitbucket та Perforce, або самостійно розміщується за допомогою програмного забезпечення з відкритим кодом, такого як GitLab), а також для фіксації, проштовхування та витягування змін, автоматизуючи завдання, які традиційно виконуються за допомогою Git із командного рядка.

3.2.7 Хмарний сервіс Firebase

Firebase — це платформа, розроблена Google для створення мобільних та веб-додатків. Спочатку це була незалежна компанія, заснована в 2011 році. У 2014 році Google придбав платформу, і тепер вона є їхньою флагманською пропозицією для розробки додатків [15].

На платформі Firebase 18 продуктів, розділених на три групи: «Develop», «Quality» та «Grow». Щоб почати користуватись Firebase продуктами, потрібно створити обліковий запис Firebase та додати ключову інформацію щодо застосунку. Після цього треба загрузити файли, які містять інформацію щодо зв'язку з Firebase платформою, налаштувати їх всередині Front-end застосунку, та використати запропоновану документацією бібліотеку, яка допоможе використовувати необхідні для клієнтської сторони серверні функції.

Firebase Authentication

Один із продуктів Firebase платформи, який вміщає у собі сервіс серверної авторизації для клієнтської частини (мобільні, дескопні та веб-додатки).

За допомоги Firebase Authentication клієнтська сторона кросплатформенного мобільного додатка не потребує серверного додатка для обробки токенів доступу або зберігання інформації о користувачах.

Щоб налаштувати Firebase Authentication потрібно додати firebase/auth підрозділ бібліотеки Firebase до клієнтського додатку, обробити цільову платформу у нативних конфігураціях (зокрема Android та iOS операційні системи).

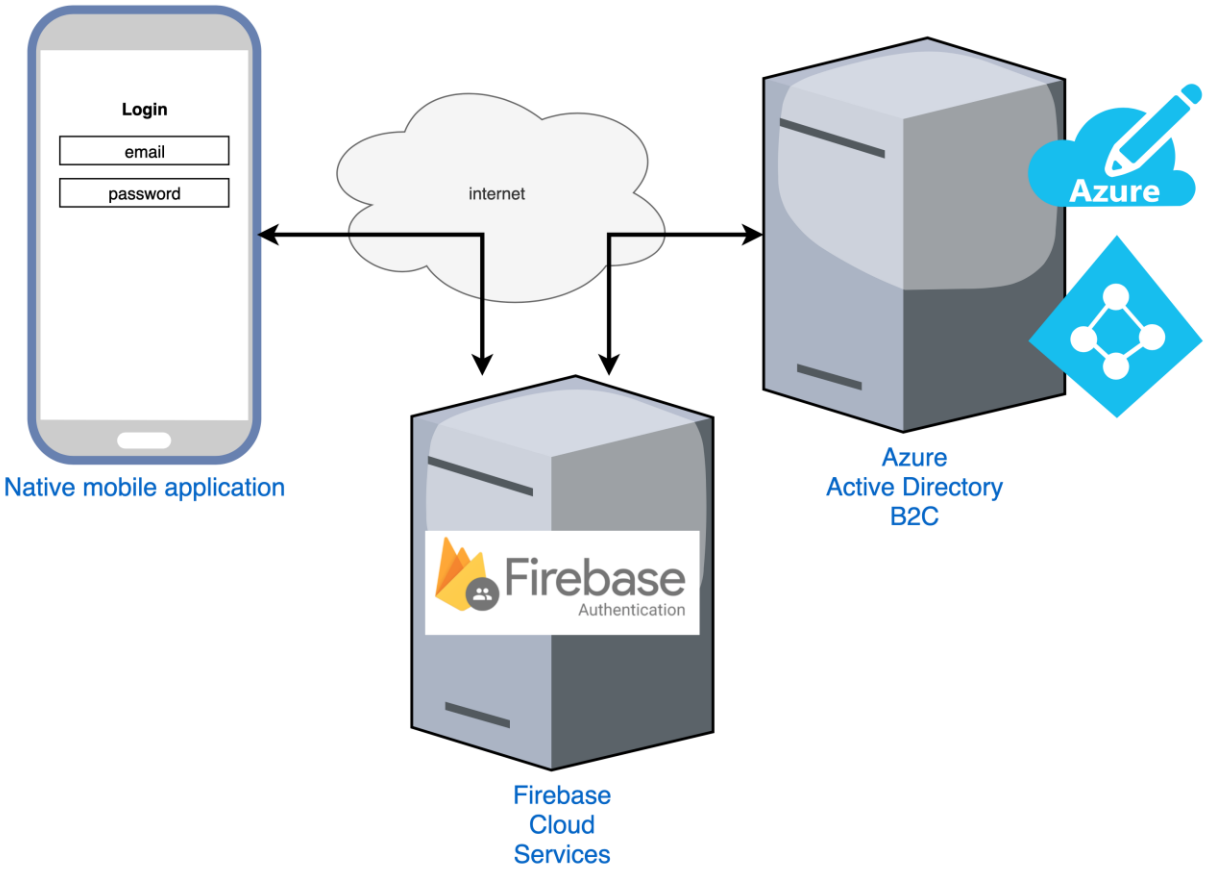

Рис. *15 Схема роботи Firebase Authentication*

Firebase Cloud Storage

Один із продуктів Firebase платформи, який вміщає у собі сервіс серверного сховища даних та доступу до них.

За допомоги Firebase Cloud Storage клієнтська сторона кросплатформеного мобільного додатка не потребує зберігання даних та доступ до них через HTTP-запроси до серверного застосунку (див. рис. 16).

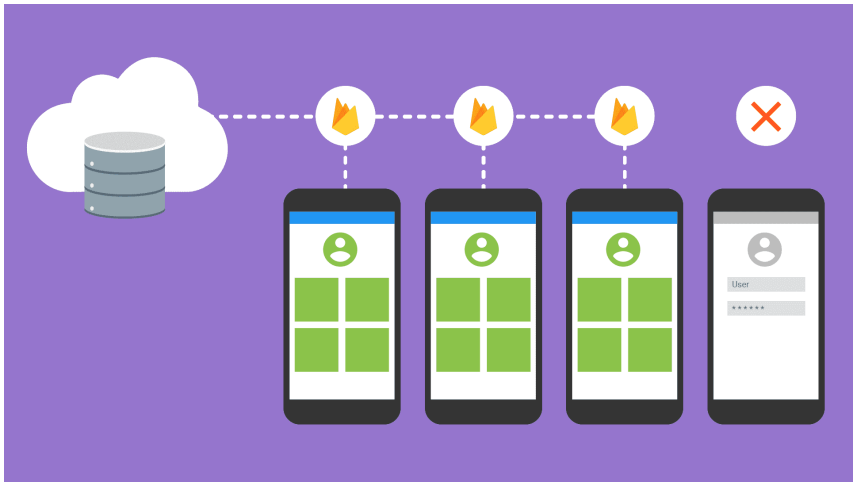

Рис. *16 Зберігання даних до Firebase Cloud Storage*

Щоб налаштувати Firebase Cloud Storage потрібно додати firebase/storage підрозділ бібліотеки Firebase до клієнтського додатку, обробити цільову платформу у нативних конфігураціях (зокрема Android та iOS операційні системи).

3.2.8 Локальне сховище даних Redux

Redux — це бібліотека JavaScript з відкритим кодом для управління станом програми [14, 20, 26]. Найчастіше використовується з такими бібліотеками, як React або Angular, для побудови користувальницьких інтерфейсів . Подібно до архітектури Facebook Flux, її створили Дан Абрамов та Ендрю Кларк.

Бібліотека використовує просте, обмежене API, яку призначене для передбачуваного контейнера для стану програми. Він працює подібно до функції зменшення, концепції функціонального програмування.

Архітектура Redux

Спосіб роботи (структура сбереження даних) Redux виглядає наступним чином (див. рис. 17):

1. Представлення може викликати будь-який із заявлених раніше Actions.

2. Actions генерують об’єкт зі своїм типом та даними, які передані із View, та потім викликають Dispatcher.
3. Reducers оброблюють будь-які надіслані їм Actions, частіше усього за допомогою конструкції Switch-Case. Всередині Reducers реалізовані як чисті функції, тобто при однакових вхідних даних очікується однаковий результат виконання коду.
4. Reducers зберігають результат у Store. При цьому, Store є повністю незмінний (Immutable), тому збереження відбувається процесом створення нових даних (перезапис старих з добавленням нових).
5. Обновлюється State того чи іншого Reducer’а, та повідомляються усі підписники до цих даних (View, яка підписалася на оновлення).

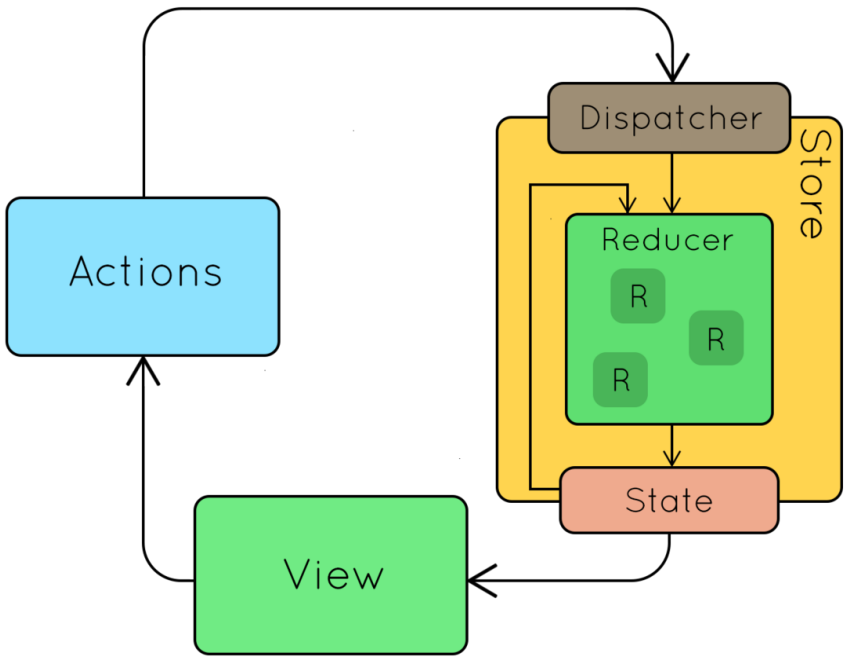

Рис. *17 Зберігання даних до Firebase Cloud Storage*

3.2.9 Бібліотека Redux-Saga

Redux-saga — це бібліотека, яка має на меті зробити побічні ефекти додатків (тобто асинхронні речі, такі як отримання даних та нечисті речі,

такі як доступ до кешу браузера) простішими в управлінні, більш ефективними у виконанні, легкими для тестування та кращими при обробці помилок [14, 16].

Психічна модель полягає в тому, що сага реалізовується як окрема нитка у вашому додатку, яка відповідає виключно за побічні ефекти. Redux-saga є проміжним програмним забезпеченням redux, що означає, що цей потік можна запускати, призупиняти та скасовувати з основної програми за допомогою звичайних дій відновлення, має доступ до повного стану програми відновлення, а також може відправляти дії відновлення [16].

Архітектура Redux-Saga

Саги працюють всередині Redux-контейнера, як доповнення (Middlewares): вони запускаються поміж Actions та Reducers, затримуючи поточну дію до тих пір, поки не буде виповнена сервера робота (робота з ефектами на стороні). Після отримання даних від сервера, чи будь-яких інших асинхроних дій, саги дописують інформацію у об'єект дії та передають його до секції Reducers (див. рис. 18).

Redux-Saga також реалізовують всередині біблиотеки методи звертання до Redux-контейнера та взаємодію з ним.

Middlewares

Окрім саг, можуть бути інші проміжні дії, такі як логери платформи, збереження даних у кеш Persist та інші.

Конструкція генераторів

Саги потребують особливого написання виконавчих функцій, які звертатимуться до серверних API. Для цього у мовах програмування зазвичай існують функції-генератори, які використовують ключове слово «yield» для затримання основого потоку виконання до тих пір, поки звер-

тання до серверу отримає відповідь або не закінчиться отведений час на виконання (timeout response).

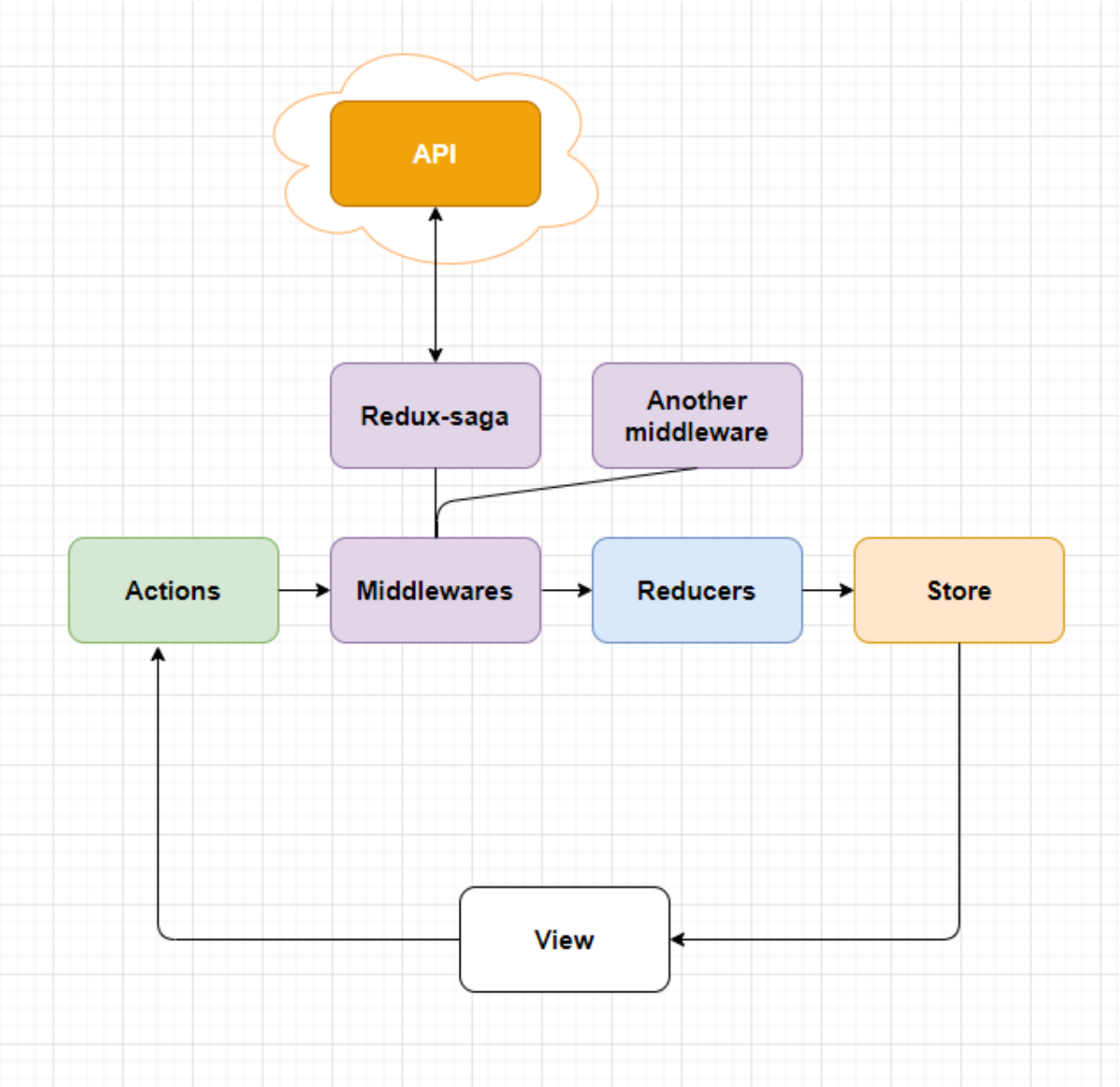

Рис. *18 Конструкція Redux-Saga у звичайному React додатку*

3.2.10 Бібліотека для зберігання кешу Persist

Однією із альтернатив зберігання кешу для Redux є бібліотека Persist [19]. Цей модуль дозволяє автоматично зберігати та загружати дані до Redux-контейнера, при старті та завершенні користування додатком. Persist виконує це, прокидуючи відповідні дії у частину Reducers, які доповнює автоматично згенерованою частиною (див. рис. 19).

Для налаштування цієї бібліотеки, треба встановити модуль до додатка та при створенні головного контейнера-сховища Redux обгорнути його до persistStore функції. Окрім цього треба обгорнути весь додаток до обгортки PersistGate. Усі функції та обгортки, що потребує бібліотека, знаходиться непосередньо всередині неї та може бути застосовано через import у початку файла.

За потреби збереження не усього стану застосунку чи деяких інших додаткових опцій маніпулювання бібліотекою, можна використовувати налаштування, як параметри до функції-створення кешованого контейнера.

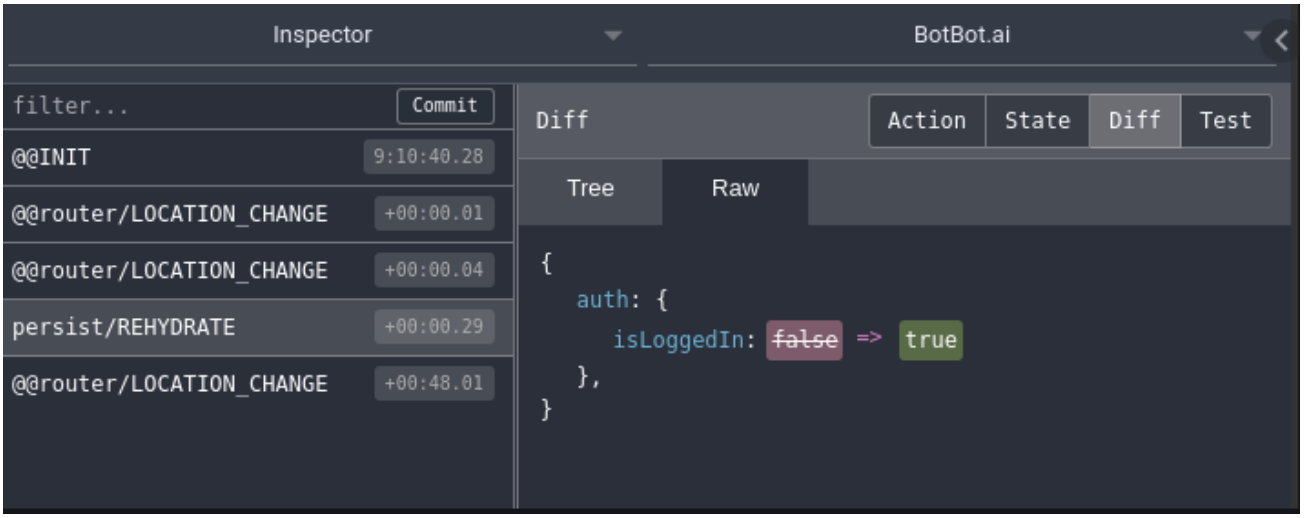

Рис. *19 Приклад кешування застосунку бібліотекою Persist*

3.2.11 Менеджери пакетів Node Package Manager та Yarn

Обов’язковою умовою деяких платформ, що створюють кросплатформенні мобільні додатки, є тісне співпрацювання з менеджерами пакетів. Зазвичай ці менеджери відповідають за поширення окремих модулів від інших авторів, які вказані у спеціальному файлі (наприклад package.json) та використовуються для збереження місця на комп’ютері та в хмарному сховищі (GitHub, BitBucket, GitLab та інші), що є дуже корисним за признаками розширюваності та модульності застосунків.

Додатково може вказуватися інформація щодо версії модулів, їх ім’я, адреси та інші. Після встановлення модулів генерується інший файл (зазвичай називається package-lock.json), який вказує версії, встановлені локально. Ця інформація може бути корисною та також відправлятися на гіт сховище.

Node Package Manager

Самий відомий менеджер пакетів з дуже великою кількістю підтримуваних модулів. Не має змоги встановлювати модулі паралельно. Після генерації package-json.lock файлу не оновлює його, навіть після повторного опрацювання модулів з package.json. При редагуванні модулів застосунку до консолі виводиться дуже багато сміття. Має набір команд, за допомоги яких користувач може якісно працювати з версіями модулей.

Yarn

Yarn – пакетний менеджер на основі NPM. Надає усі можливості, що й NPM, але має низку переваг, серед яких:

1. Швидше встановлення чи редагування модулів за допомоги паралельних процесів.
2. Замість package-json.lock файлу, генерує yarn-lock.json із іншою структурою.
3. Оновлює yarn-lock.json після кожної обробки модулів.
4. Має більш структуровані консольні команди.
5. Має більш оптимізований консольний інтерфейс при використанні команд.

3.2.12 Система контролю версій Git

Git — це розподілена система контролю версій для відстеження змін у будь-якому наборі файлів, початковою ціллю якої була координація роботи між програмістами, які співпрацюють над вихідним кодом під час ро-

зробки програмного забезпечення [18]. Її цілі включають швидкість, цілісність даних та підтримку розподілених нелінійних робочих процесів.

За допомоги системи контролю версій Git можна організувати не тільки командну роботу, але й зручно працювати над проектами власноруч. Система гілок надає надзручний інтерфейс для розподілення праці та менеджменту часу програміста (див. рис. 20). Також слід відокремити те, що не потрібно зберігати купу архівів на комп'ютері, як це слід було робити раніше. Система контролю версій допомагає зручно переміщатися від однієї версії проекту до іншої.

Git має стандартний процес зберігання усіх завантажених до нього файлів: необхідно створити репозиторій (зазвичай робиться один на проект), у якому створити першу гілку та завантажити коміт для ініціалізації контролю версій. За допомоги комітів виконується нумерація файлів, які представлені у формі зліпків. Кожен зліпок має бінарну структуру та підпис версії, до якої цей зліпок був зроблений. Зліпки вміщають тільки необхідну інформацію, так що місця вони займають не так багато, як збереження цифрової копії.

Система контролю версій надає зручний процес отримання будь-якої збереженої версії у короткий проміжок часу. Але частина файлів не повинні бути завантажені на гіт. Це стосується, наприклад, файлів білдів, кешу та модулів проекта. Тому існує .gitignore файл, до якого записується уся інформація щодо типів файлів, які потребують завантаження кожного разу, а які не потребують зовсім.

Система має ряд команд, які надають змогу працювати з версіями для кожного проекту окремо.

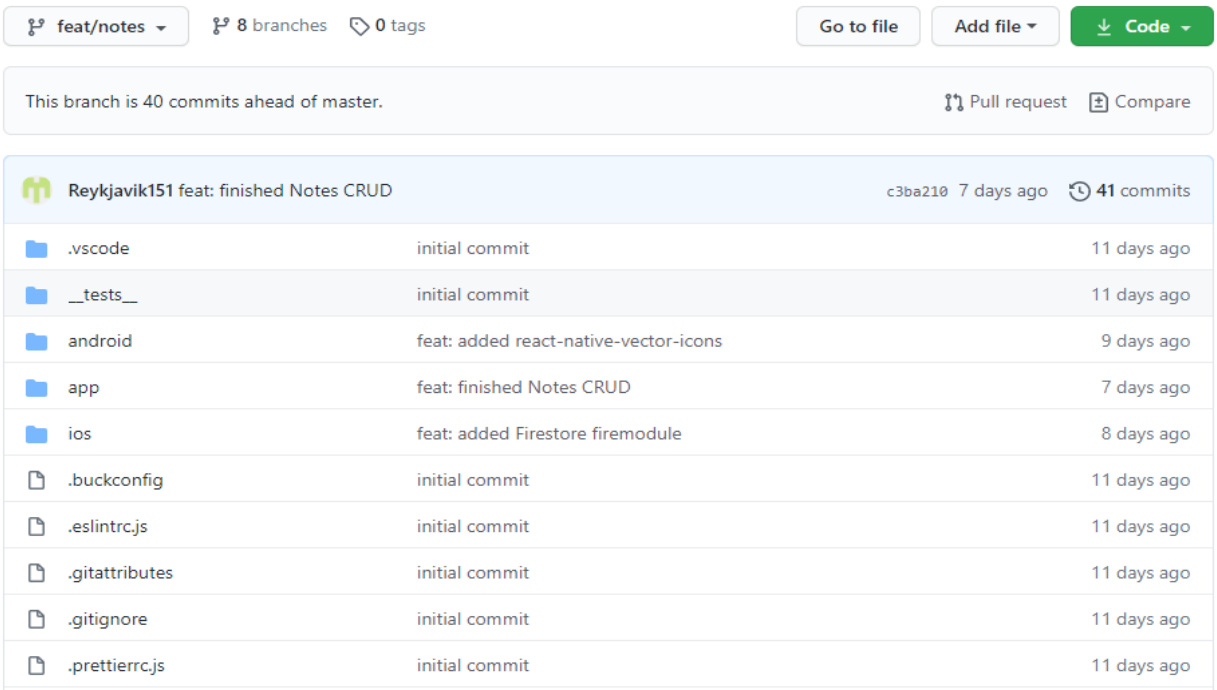

Рис. *20 Система контролю версій Git на прикладі React Native застосунку*

3.2.13 Середовище для відображення стану Git SourceTree

SourceTree — програмне забезпечення, що візуалізує стан системи контролю версій та надає більш зручний інтерфейс для виконання Git-команд (див. рис. 21) [7].

Надає можливості програмістові [7] :

- переглядання усіх файлів проекту та їх різності між версіями;
- обробка комітів, гілок;
- створення pull-запитів;
- під’єднання декількох репозиторіїв одночасно.

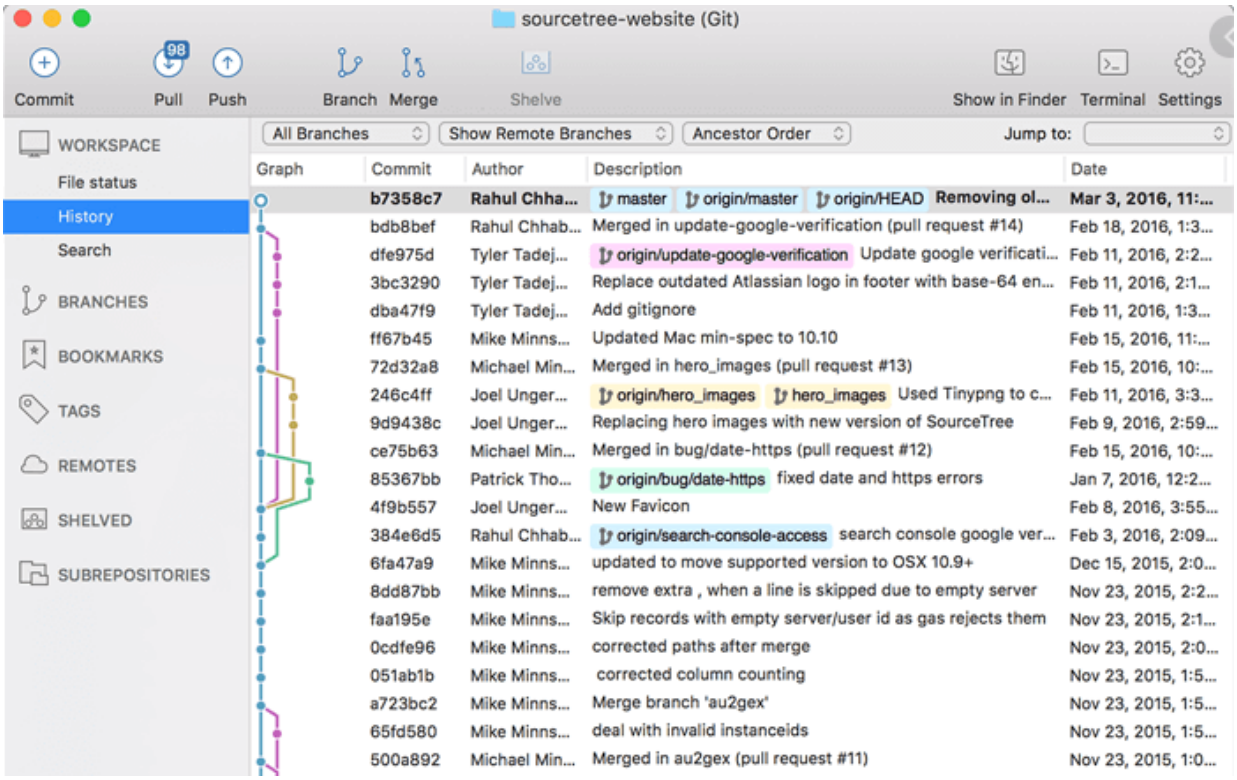

Рис. *21 Інтерфейс середовища відображення стану Git SourceTree*

3.3 Модулі і алгоритми

Під час розробки ToDo-застосунку було розроблено наступні модулі:

1. Модуль автентифікації Firebase користувача.
2. Модуль хмарного сховища даних Firebase Cloud Storage.
3. Модуль інтерфейсу.

3.3.1 Модуль автентифікації Firebase користувача

Нижче на рисунку 22 представлена архітектура модулю автентифікації Firebase, на якій можна простежити етапи проходження всього алгоритму.

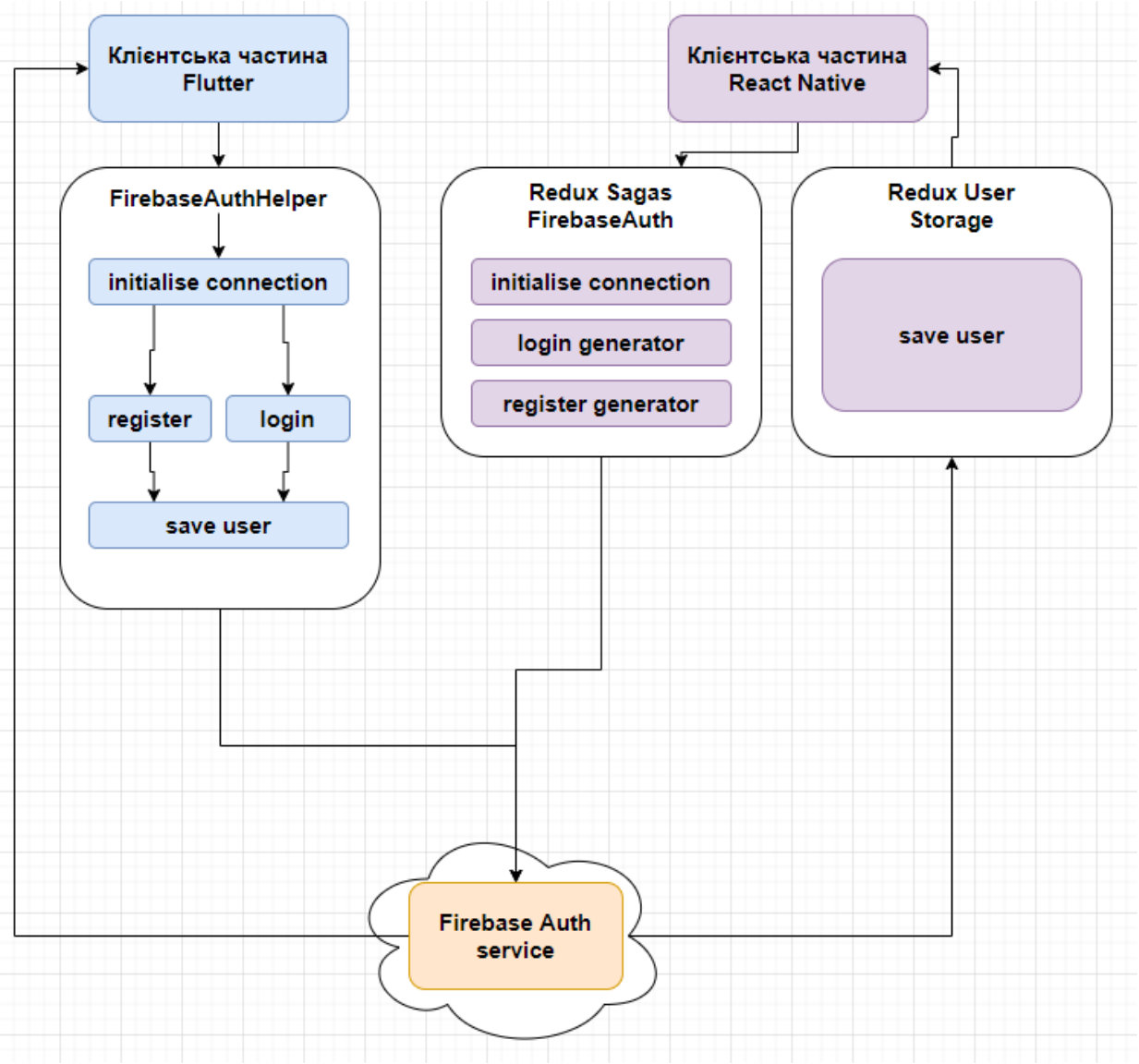

Рис. *22 Модуль автентифікації користувача через Firebase Auth*

Компонент Flutter FirebaseAuthHelper

Клієнтська частина на платформі Flutter реалізовує об'єктно-орієнтовану модель: клас FirebaseAuthHelper. Це представлення має три етапи:

1. Етап створення з'єднання. Ініціалізація з'єднання з Firebase та створення окремого Firebase екземпляра, який використовує нативні конфігурації Android чи iOS директорії, в залежності аід виконавчої операційної системи.
2. Етап проходження автентифікації. На цьому етапі логіка методів розділяється на реєстрацію та логін, в залежності від обраної оп-

ції користувачем. Обидві відрізняються лише основним виконавчим процесом Firebase бібліотеки.

3. Етап збереження інформації користувача. Отримана інформація від Firebase бібліотеки буде оброблена та збережена у результаті позитивної відповіді. У разі негативної, буде збережена помилка.

В лістингу 1 наведена головна реалізація автентифікації класу FirebaseHelper на мові програмування Dart.

Лістинг *1 Клас автентифікації FirebaseAuthHelper*

```
class FirebaseAuthHelper {
    static Future<UserCredential> register({String email,
String password}) async {
        FirebaseAuth instance = FirebaseAuth.instance;
        UserCredential newUser = await instance
.createUserWithEmailAndPassword(
            email: email,
            password: password,
        );
        return newUser;
      }

    static Future<UserCredential> login({String email,
String password}) async {
        FirebaseAuth instance = FirebaseAuth.instance;
        UserCredential newUser = await in-
stance.signInWithEmailAndPassword(
            email: email,
            password: password,
        );
        return newUser;
      }

      static User getCurrentUser() {
        return FirebaseAuth.instance.currentUser;
      }
}
```

Компонент React Native Redux Sagas FirebaseAuth

Клієнтська частина на платформі React Native реалізує структуру з'єднання з сервісом через Redux та Redux-Saga бібліотеки. Це представлення має наступні етапи:

1. Компонент застосунку у потрібний момент прокидує дію, яка перехоплюється однією з саг, підписаною на оновлення дій.
2. Перша сага очікує дію з типом LoginAction та реалізує авторизацію вже існуючого користувача.
3. Друга сага очікує дію з типом RegisterAction та реалізує авторизацію нового користувача.
4. Кожна з саг після вдалої авторизації викликає збереження користувацької інформації, отриманої з виклику сервіса FirebaseAuth, до Redux сховища.
5. Викликається StaticNavigator, який оброблює навігаційні дії у додатку та перенаправляє користувача до головної сторінки.
6. Redux оброблює останній тип прокидуваємої дії для збереження інформації та оновлення стану застосунку.
7. Обробка помилок присутня на кожному етапі за допомоги конструкції try-catch та зберігається як помилка для подальшого відображення користувачу.

В лістингу 2 наведена головна реалізація обробки дій авторизації генераторами всередині саг.

Лістинг *2 Реалізація обробки дій автентифікації Redux Sagas FirebaseAuth*

```
export function* login({ email, password }: LoginAction)
{
     try {
          const userCredentialsResponse:
FirebaseAuthTypes.UserCredential = yield call(
           FirebaseHelper.login,
           email,
           password,
         );
```

```
        yield put<SaveUserCredentialsAction>(
userActionCreators.saveUserCredentials(userCredentialsRes
ponse));
        yield call(StaticNavigator.resetTo, 'HomeStack');
      } catch (err) {
        yield put<SetErrorAction>(userActionCreators
.setError(err));
      }
    }

export function* register({ email, password }:
RegisterAction) {
      try {
        const userCredentialsResponse:
FirebaseAuthTypes.UserCredential = yield call(
          FirebaseHelper.register,
          email,
          password,
        );

        yield put<SaveUserCredentialsAction>(
userActionCreators.saveUserCredentials(
userCredentialsResponse));
        yield call(StaticNavigator.resetTo, 'HomeStack');
      } catch (err) {
        yield put<SetErrorAction>(userActionCreators.
setError(err));
      }
}
```

Компонент FirebaseAuth Service

Для використання сервісу автентифікації було налаштовано платформу Firebase наступним чином:

1. Зареєстровано обліковий запис користувача Firebase.
2. Додано два застосунку під кожну з операційних систем.
3. Включено обробку автентифікації сервером у вкладці налаштувань.

4. Включено зберігання інформації користувачів окремо від основної бази даних.

На рисунках 23 та 24 наведений налаштований хмарний сервіс зберігання аккаунтів користувачів.

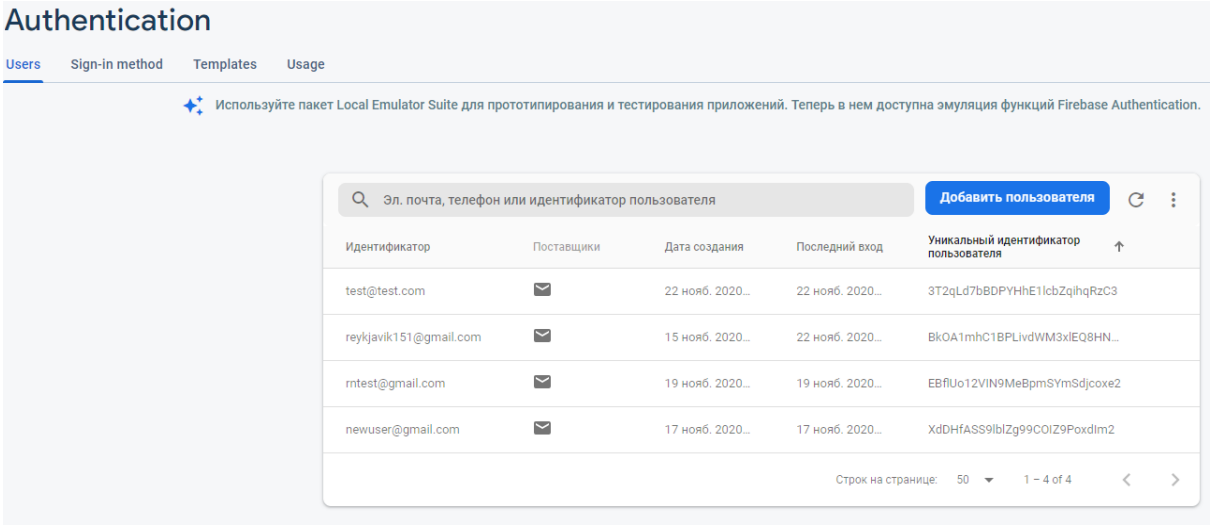

Рис. *23 Збережені користувачі платформою Firebase*

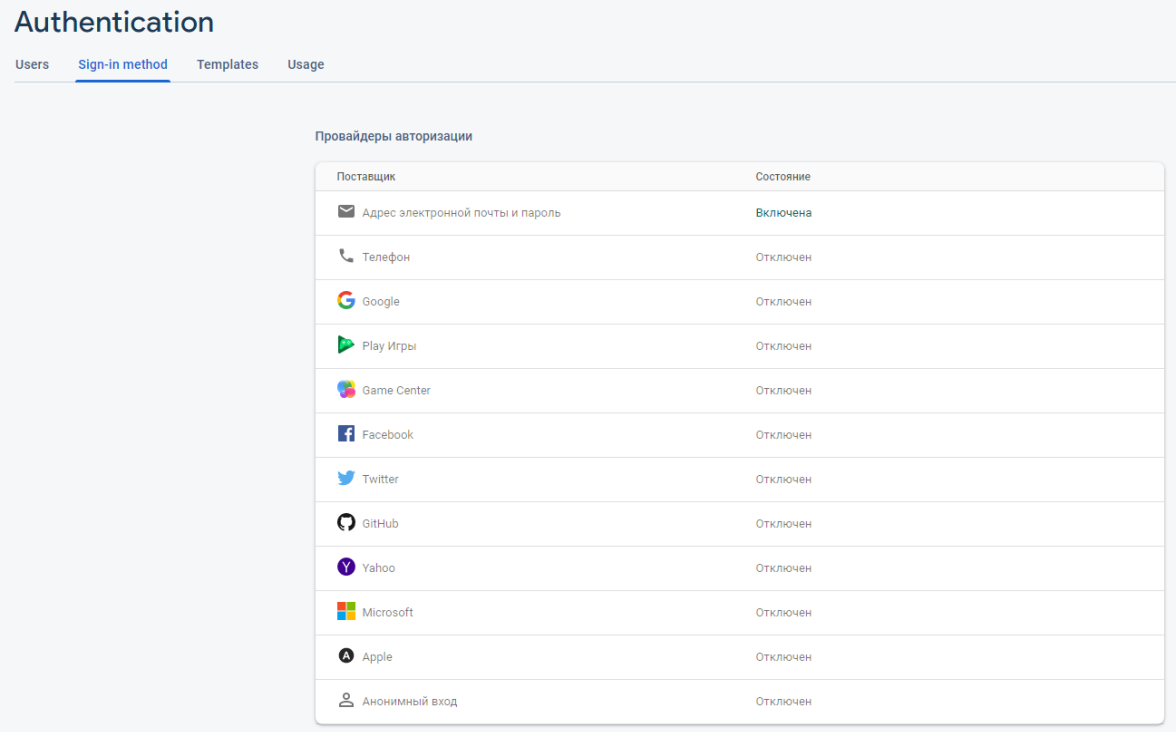

Рис. *24 Налаштовані методи авторизації Firebase*

Компонент React Native Redux User Storage

Після виконання отримання інформації щодо облікового запису користувача, ця інформація надходить до Redux сховища та потрапляє в User Reducer.

В середині знижувач-функція оновлює сховище та генерує новий стан додатка, після чого Redux-контейнер оновлює усіх підписників-компонентів.

В лістингу 3 наведена реалізація збереження інформації користувача до сховища.

Лістинг *3 Сбереження інформації наданої при автентифікації*

```
export interface UserState {
     isLoading: boolean;
     userCredentials: FirebaseAuthTypes.UserCredential |
null;
     error: string | null;
}

const INITIAL_STATE: UserState = {
     isLoading: false,
     userCredentials: null,
     error: null,
};

type Handler<A> = (state: UserState, action: A) =>
UserState;

const login: Handler<LoginAction> = (state) => ({
     ...state,
     isLoading: true,
});

const register: Handler<RegisterAction> = (state) => ({
     ...state,
     isLoading: true,
});

const saveUserCredentials:
Handler<SaveUserCredentialsAction> = (state, {
userCredentials }) => ({
```

```
    ...state,
    userCredentials,
    isLoading: false,
    error: null,
});

const setError: Handler<SetErrorAction> = (state, { error
}) => ({
    ...state,
    isLoading: false,
    error,
});

export const userReducer = createReducer<UserState,
UserAction>(INITIAL_STATE, {
    [userActionTypes.LOGIN]: login,
    [userActionTypes.REGISTER]: register,
    [userActionTypes.SAVE_USER_CREDENTIALS]:
saveUserCredentials,
    [userActionTypes.SET_ERROR]: setError,
});
```

3.3.2 Модуль хмарного сховища даних Firebase Cloud Storage

Нижче на рисунку 25 представлена архітектура модулю хмарного сховища даних Firebase Cloud Storage, на якій можна простежити етапи проходження всього алгоритму.

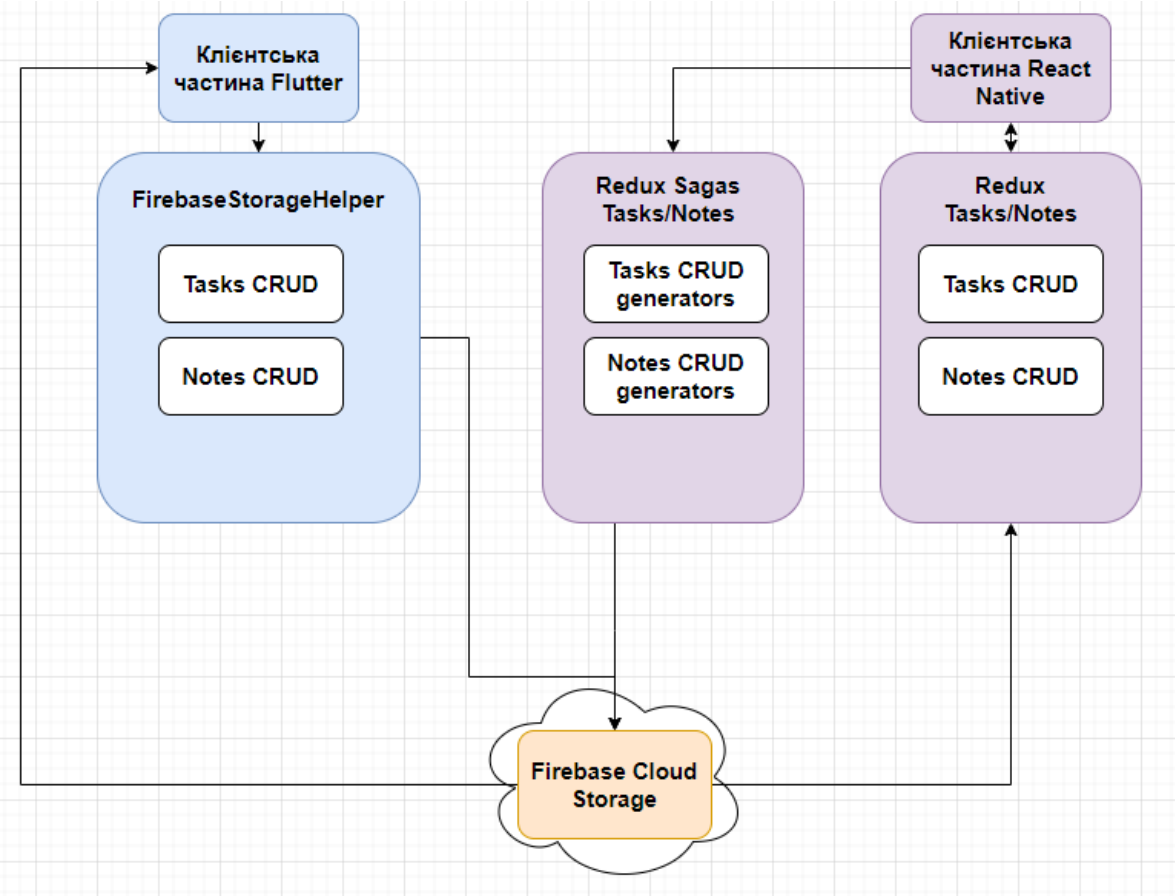

Рис. 25 Архітектура модуля хмарного сховища та з'єднання з ним

Компонент Flutter FirebaseStorageHelper

Клієнтська частина на платформі Flutter реалізовує об'єктно-орієнтовану модель: клас FirebaseStorageHelper, який складається з декількох частин:

1. Створення, редагування, отримання та видалення колекції Tasks.
2. Створення, редагування, отримання та видалення колекції Notes.

В лістингу 4 наведена реалізація роботи з Firebase колекціями Tasks та Notes.

Лістинг *4 Клас FirebaseStorageHelper для роботи з Firebase колекціями*

```
class FirebaseStorageHelper {
static Future<List<Task>> getTasksByUid(String uid) async
{
    CollectionReference tasksRef =
FirebaseFirestore.instance.collection('tasks');
```

```
    List<Task> resultTasks = [];

    QuerySnapshot snapshots = await tasksRef
        .where('ownerUid', isEqualTo: uid)
        .orderBy('createdAt')
        .get();

    snapshots.docs.forEach((element) {
        Map<String, dynamic> elementData =
    element.data();
        resultTasks.add(
            new Task(
            id: element.id,
            status: StatusHelper.
getStatusByString(elementData['status']),
            ownerId: elementData['ownerUid'],
            title: elementData['title'],
            description: elementData['description'],
            createdAt: elementData['createdAt'],
            ),
          );
        });

        return resultTasks;
    }

static Future<List<Note>> getNotesById(String uid) async
{
    CollectionReference notesRef =
FirebaseFirestore.instance.collection('notes');

    List<Note> resultNotes = [];

    QuerySnapshot snapshots = await notesRef
        .where('ownerUid', isEqualTo: uid)
        .orderBy('createdAt')
        .get();

    snapshots.docs.forEach((element) {
      Map<String, dynamic> elementData = element.data();
      resultNotes.add(
        new Note(
          id: element.id,
```

```
            ownerId: elementData['ownerUid'],
            title: elementData['title'],
            body: elementData['body'],
            createdAt: elementData['createdAt'],
          ),
        );
      });

      return resultNotes;
}

static Future<Task> addTask({Task task}) async {
      CollectionReference tasksRef =
FirebaseFirestore.instance.collection('tasks');

      DocumentReference doc = await tasksRef.add({
        'status':
StatusHelper.getStringByStatus(task.status),
        'ownerUid': task.ownerId,
        'title': task.title,
        'description': task.description,
        'createdAt': task.createdAt,
      });

      DocumentSnapshot snapshot = await doc.get();
      task.id = snapshot.id;

      return task;
}

static Future<Note> addNote({Note note}) async {
      CollectionReference notesRef =
FirebaseFirestore.instance.collection('notes');

      DocumentReference doc = await notesRef.add({
        'ownerUid': note.ownerId,
        'title': note.title,
        'body': note.body,
        'createdAt': note.createdAt,
      });

      DocumentSnapshot snapshot = await doc.get();
      note.id = snapshot.id;
```

```
    return note;
}

static Future<Task> updateTask({Task task}) async {
    DocumentReference taskRef =
FirebaseFirestore.instance.collection('tasks').doc(task.i
d);

    await taskRef.update({
      'status':
StatusHelper.getStringByStatus(task.status),
      'ownerUid': task.ownerId,
      'title': task.title,
      'description': task.description,
});

    return task;
}

static Future<Note> updateNote({Note note}) async {
    DocumentReference noteRef =
FirebaseFirestore.instance.collection('notes')
.doc(note.id);

    await noteRef.update({
      'ownerUid': note.ownerId,
      'title': note.title,
      'body': note.body,
    });

    return note;
}

static Future<bool> deleteTask({String taskId}) async {
    await FirebaseFirestore.instance.
collection('tasks').doc(taskId).delete();

    return true;
}

static Future<bool> deleteNote({String noteId}) async {
```

```
    await FirebaseFirestore.instance.
collection('notes').doc(noteId).delete();

    return true;
}
}
```

Особливості реалізації**:**

1. Колекції сортуються за часом додання на етапі отримання даних зі сховища.
2. Елементи колекції отримують ідентифікатори документа Firebase для зручності їхньої обробки.
3. Видалення елементу — при успішному видаленні повертає значення логічного типу для розуміння процесу виконання іншими елементами системи.

Компонент React Native Redux Sagas Tasks/Notes

Для клієнтської частини на платформі React Native було розділено логіку роботи з хмарним сховищем Firebase Cloud Storage на дві частини: саги та redux-контейнер. У розділі саг виконуються асинхронні процеси зв’язку з Cloud Storage через React Native Firebase бібліотеку.

В лістингу 5 показані основні частини саги Tasks, як приклад виконаного CRUD.

Лістинг *5 Саги зв’язку до Firebase Cloud Storage колекцій*

```
export function* getTasks() {
    try {
        const userUid = yield select((state) =>
    state.user.userCredentials.user.uid);

        const tasks: ITask[] = yield call(
            FirebaseHelper.getCollectionWithDocId,
            userUid,
            FIREBASE_TASKS_COLLECTION_NAME,
        );
```

```
            yield
        put<GetTasksSuccessAction>(tasksActionCreators.getTas
        ksSuccess(tasks));
            } catch (err) {
            yield
        put<GetTasksFailureAction>(tasksActionCreators.getTas
        ksFailure(err));
            }
        }

export function* addTask({ task }: AddTaskAction) {
        try {
            const taskWithId = yield
FirebaseHelper.addDoc<ITask>(FIREBASE_TASKS_COLLECTION_NA
ME, task);

            yield put<AddTaskSuccessAction>
        (tasksActionCreators.addTaskSuccess(taskWithId));
          } catch (err) {
            yield put<AddTaskFailureAction>
        (tasksActionCreators.addTaskFailure(err));
          }
        }

export function* updateTask({ task }: UpdateTaskAction) {
        try {
            yield FirebaseHelper.updateDoc<ITask>
        (FIREBASE_TASKS_COLLECTION_NAME, task);

            yield put<UpdateTaskSuccessAction>
        (tasksActionCreators.updateTaskSuccess(task));
        } catch (err) {
            yield put<UpdateTaskFailureAction>
        (tasksActionCreators.updateTaskFailure(err));
        }
}

export function* deleteTask({ taskId }: DeleteTaskAction)
{
        try {
            const isDeleted: boolean = yield
            FirebaseHelper.deleteDoc(
                FIREBASE_TASKS_COLLECTION_NAME,
```

```
            taskId,
        );

        if (isDeleted) {
            yield put<DeleteTaskSuccessAction>
    (tasksActionCreators.deleteTaskSuccess(taskId));
        } else {
            throw new Error("Task Delete: TaskID can't
be found");
        }
    } catch (err) {
        yield put<DeleteTaskFailureAction>
    (tasksActionCreators.deleteTaskFailure(err));
    }
}
```

Особливості реалізації повністю відповідають вимогам однакової міжплатформенної логіки: реалізована сага виконує усі ті ж самі пункти особливостей реалізації, вказані у попередньому Flutter компоненті.

Компонент Firebase Cloud Storage

Для використання сервісу хмарного сховища було налаштовано платформу Firebase наступним чином:

1. Зареєстровано обліковий запис користувача Firebase.
2. Додано два застосунку під кожну з операційних систем.
3. Створена база даних Cloud Storage.
4. Дозволений запис та читання усіх користувачів застосунка.
5. Створені составні індекси для обробки колекцій Tasks та Notes (сортування даних по полю ownerUid та createdAt).

Нижче на рисунках 26, 27 та 28 представлене налаштування сервісу Firebase Cloud Storage.

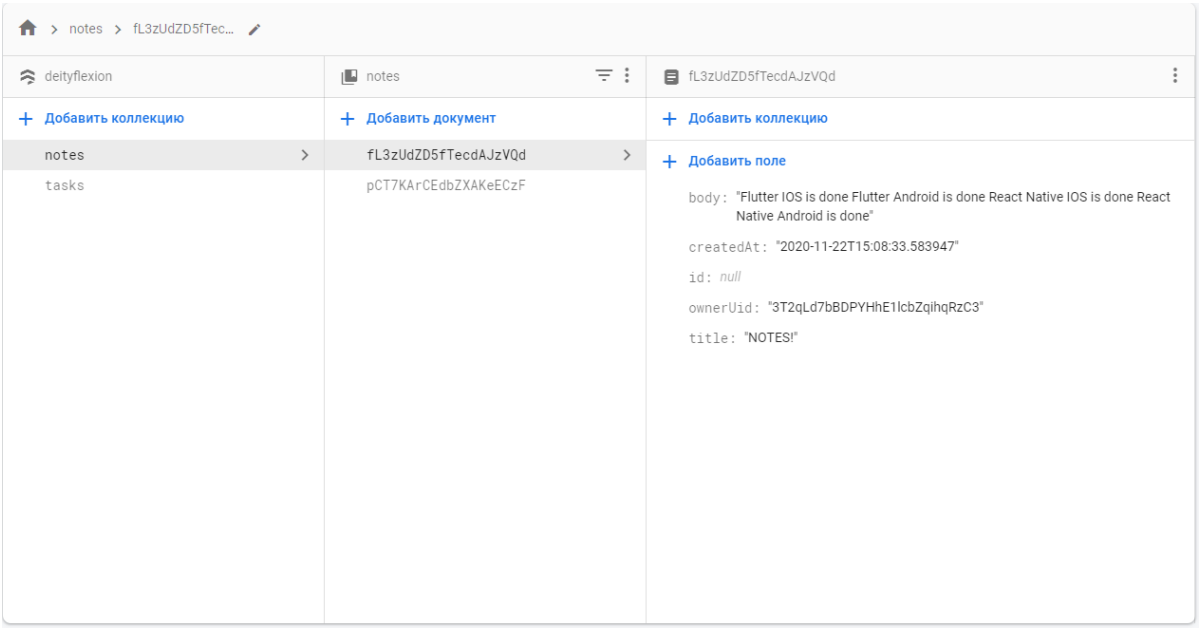

Рис. *26 Налаштування колекцій Firebase Cloud Storage*

Рис. *27 Налаштування доступу до колекцій*

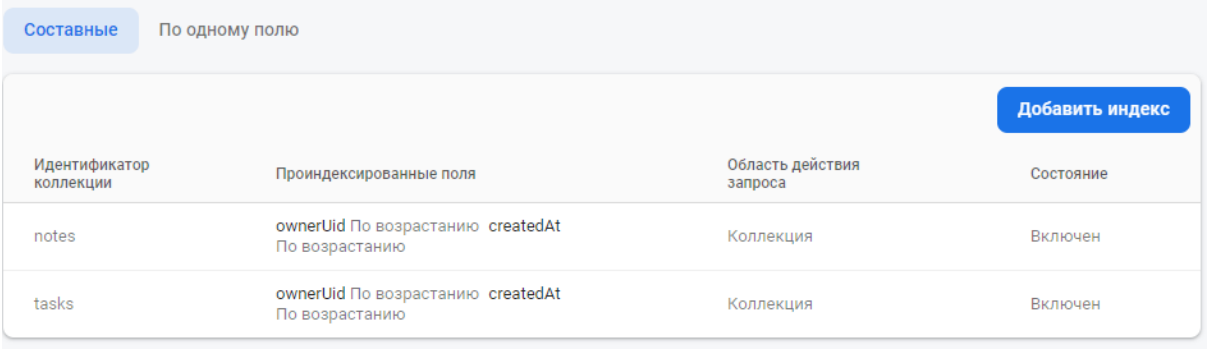

Рис. *28 Налаштування составних індексів колекцій*

3.3.3 Модуль інтерфейсу

Інтерфейс розроблений за допомогою патерну MVC (див. рис. 29).

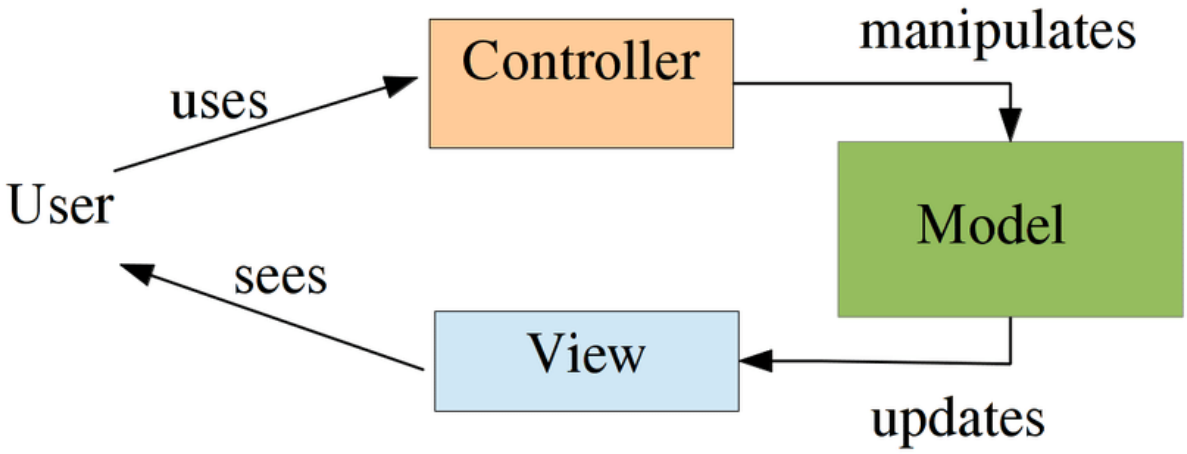

Рис. *29 Паттерн проектування Model-View-Controller*

Модуль інтерфейсу React Native

Для кожної сторінки та компонента існує своя реалізація JSX розмітки, яка css якої виноситься в окремий файл з суфіксом **.styles.ts*. Окрема також винесена типізація *props* кожного компоненту чи екрану. В *index.ts* файлі розміщується основна реалізація логіки та використання *styles* та *props*.

Створені дві директорії окремо, для компонентів це директорія *components*, а для сторінок — *screens* відповідно. Також створена директорія під застосування бібліотеки навігації та розміщення там файлів, які відповідають за вкладеність та оброку стеків навігації.

Щоб запобігти дубліката коду, деякі використані сторінки приймають параметри навігацію, щоб рендерити та оброблювати логіку схожу за інтерфейсом (наприклад, TasksAdd реалізовує усередині не тільки додавання, але й редагування, якщо переданий відповідний параметр навігації).

Директорії *services* та *utils* використовуються лише для деяких функціональностей, які реалізуються у деяких місцях програми.

На рисунку 30 можна побачити структуру файлів проекту, яка відповідає за інтерфейс.

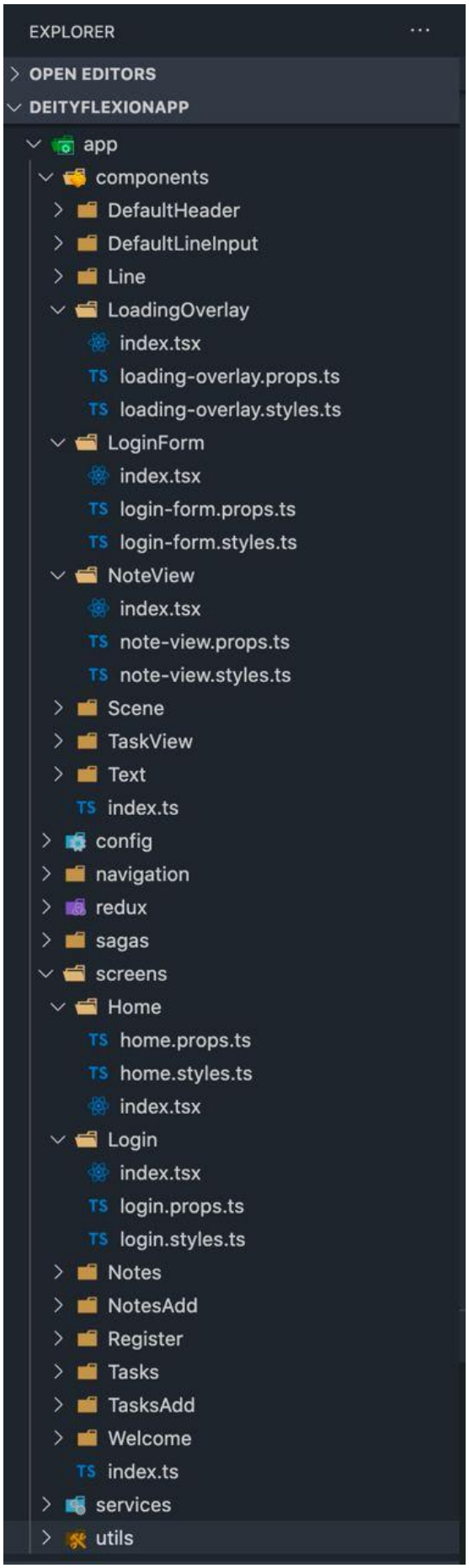

Рис. 30 Структура файлів інтерфейсу React Native застосунку

В лістингу 6 наведена головна реалізація однієї сторінки інтерфейсу React Native застосунку.

Лістинг *6 Реалізація React Native інтерфейсу HomeScreen*

```
import React, { FunctionComponent } from 'react';
import { SafeAreaView, View, TouchableOpacity } from
'react-native';
import MaterialIcon from 'react-native-vector-
icons/MaterialIcons';

import { Line } from '../../components';
import { LINE_HEIGHT } from
'../../components/Line/line.presets';
import { COLORS } from '../../utils/colors';
import { COMMON_STYLES } from '../../utils/commonStyles';
import { HomeScreenProps } from './home.props';
import { styles } from './home.styles';

export const HomeScreen:
FunctionComponent<HomeScreenProps> = ({ navigation }) =>
{
       return (
         <SafeAreaView style={styles.container}>
           <View style={[COMMON_STYLES.flexContainer,
COMMON_STYLES.centeredContainer]}>
             <TouchableOpacity
               style={[styles.button,
COMMON_STYLES.centeredContainer]}
               onPress={() =>
navigation.navigate('Tasks')}
             >
               <MaterialIcon name="check" size={115}
color={COLORS.PRIMARY_BLUE} />
             </TouchableOpacity>
           </View>
           <Line height={LINE_HEIGHT.SMALL} style={{
lineContainer: COMMON_STYLES.mb12 }} />
           <Line height={LINE_HEIGHT.LARGE} style={{
lineContainer: COMMON_STYLES.mb12 }} />
           <Line height={LINE_HEIGHT.LARGE} style={{
lineContainer: COMMON_STYLES.mb12 }} />
           <Line height={LINE_HEIGHT.SMALL} />
```

```
          <View style={[COMMON_STYLES.flexContainer,
COMMON_STYLES.centeredContainer]}>
            <TouchableOpacity
              onPress={() =>
navigation.navigate('Notes')}
              style={[styles.button,
COMMON_STYLES.centeredContainer,
COMMON_STYLES.rotation270]}
            >
              <MaterialIcon name="note" size={115}
color={COLORS.PRIMARY_BLUE} />
            </TouchableOpacity>
          </View>
        </SafeAreaView>
      );
};
```

Особливості реалізації:

1. Винесені кольори, стилі, константи, опції та властивості до окремих директорій.
2. Функціональний компонент без наслідування React.Component.
3. Типізація за допомоги TypeScript можливостей.
4. Об’єднання стилів сторінки з загальними стилями застосунку.

Модуль інтерфейсу Flutter

Для кожної віджету-сторінки та віджету-компонента існує своя Dart-реалізація інтерфейсу. Взаємодія з колекціями винесена окремо від інтерфейсної логіки, у директорію *data*. Віджети-сторінки та віджети-компоненти розділені на директорії *components* та *screens*.

Директорія *services* слугує для збереження логіки роботи з Firebase модулями.

Файл *main.dart* реалізовує сервіс навігації по застосунку. Файл *constants.dart* вміщує до себе усі константи, які використовуються як у інтерфейсі, так і у логіці.

На рисунку 31 можна побачити структуру файлів проекту, яка відповідає за інтерфейс.

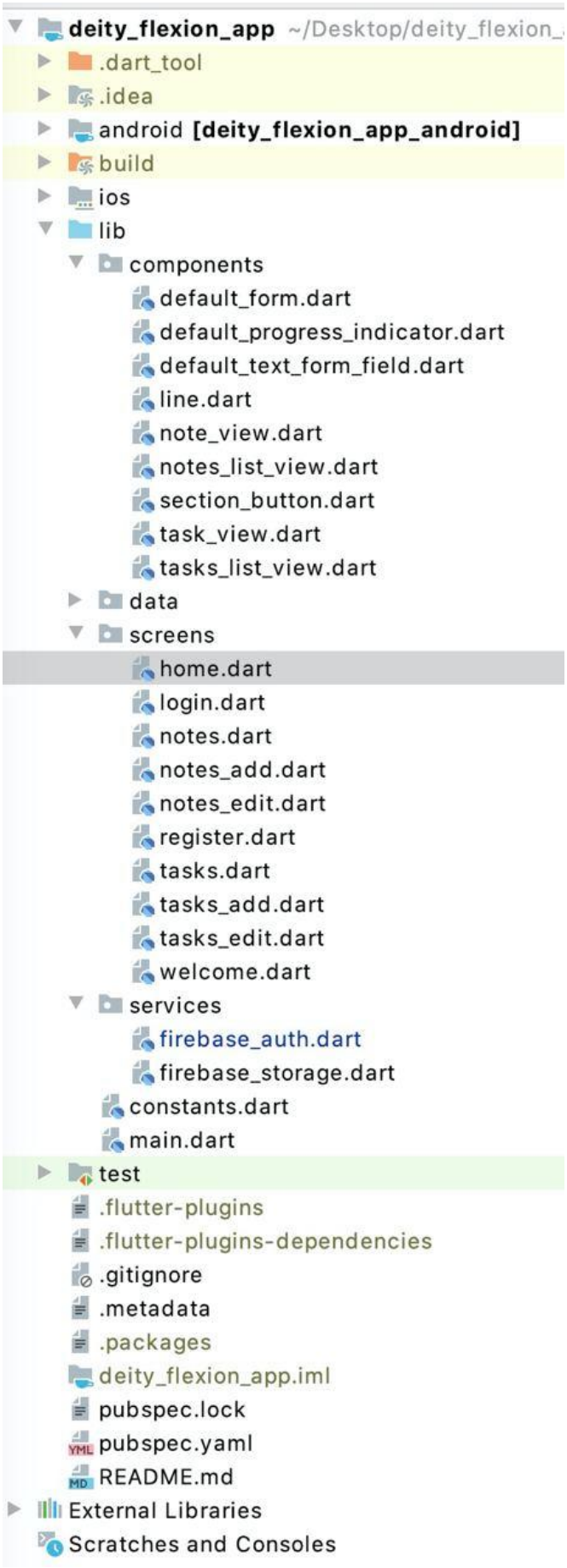

Рис. *31 Структура файлів інтерфейсу Flutter застосунку*

В лістингу 7 наведена головна реалізація однієї сторінки інтерфейсу Flutter застосунку.

Лістинг *7 Реалізація Flutter інтерфейсу HomeScreen*

```
class HomePage extends StatefulWidget {
     static String id = 'home_screen';

     @override
     _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
     @override
     Widget build(BuildContext context) {
       return Scaffold(
         backgroundColor: Colors.lightBlueAccent[200],
         body: SafeArea(
           child: Column(
             mainAxisAlignment: MainAxisAlignment.center,
             crossAxisAlignment:
CrossAxisAlignment.stretch,
             children: [
               Expanded(
                 child: Center(
                   child: Hero(
                     tag: TASKS_HERO_TAG,
                     child: SectionButton(
                       icon: Icons.check,
                       onPress: () {
                         Navigator.pushNamed(context,
TasksPage.id);
                       },
                     ),
                   ),
                 ),
               ),
               Column(
                 mainAxisAlignment:
MainAxisAlignment.center,
                 children: [
                   Hero(
                     tag: FIRST_LINE_HERO_TAG,
                     child: Line(
```

```
                        height: 12.0,
                        margin: EdgeInsets.only(bottom:
12.0),
                      ),
                    ),
                    Hero(
                      tag: SECOND_LINE_HERO_TAG,
                      child: Line(
                        height: 32.0,
                        margin: EdgeInsets.only(bottom:
12.0),
                      ),
                    ),
                    Hero(
                      tag: THIRD_LINE_HERO_TAG,
                      child: Line(
                        height: 32.0,
                        margin: EdgeInsets.only(bottom:
12.0),
                      ),
                    ),
                    Hero(
                      tag: FOURTH_LINE_HERO_TAG,
                      child: Line(
                        height: 12.0,
                      ),
                    ),
                  ],
                ),
                Expanded(
                  child: RotatedBox(
                    quarterTurns: 3,
                    child: Center(
                      child: Hero(
                        tag: NOTES_HERO_TAG,
                        child: SectionButton(
                          icon: Icons.note,
                          onPress: () {
                            Navigator.pushNamed(context,
NotesPage.id);
                          },
                        ),
                      ),
```

```
                    ),
                  ),
                ),
              ],
            ),
          ),
        );
      }
}
```

Особливості реалізації:

1. Героїчна анімація між екранів.
2. Константи, робота з даними, сервіси та навігація винесені окремо від інтерфейсної логіки.
3. Компонент с устаткуванням стану.

3.4 Проект інтерфейсу

Мобільний застосунок надає можливість створення користувацького облікового запису та колекцій, якими можна маніпулювати всередині кожного запису (див. рис. 32-34).

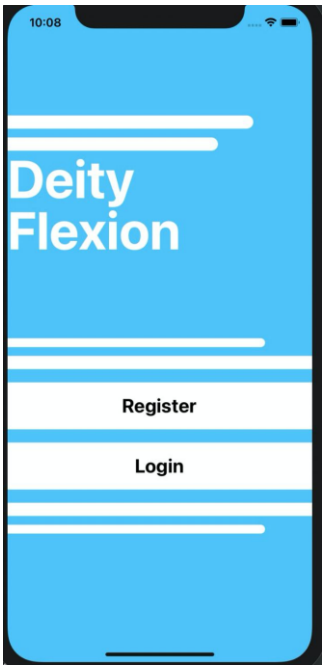

Рис. *32 Початкова сторінка для обирання шляху авторизації*

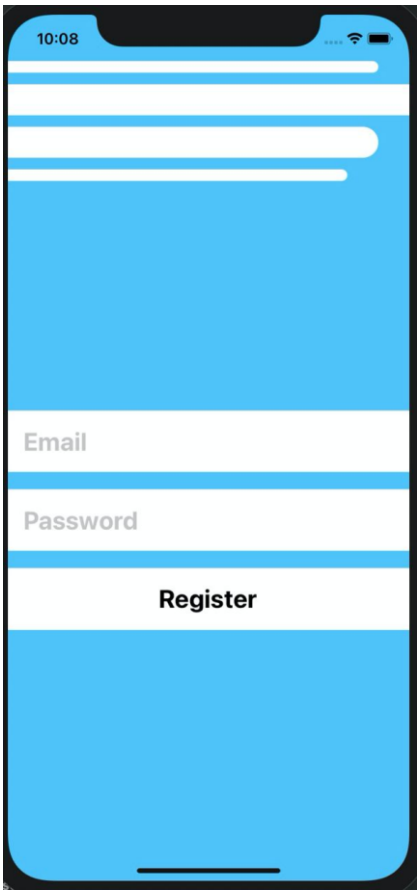

Рис. *33 Сторінка реєстрації*

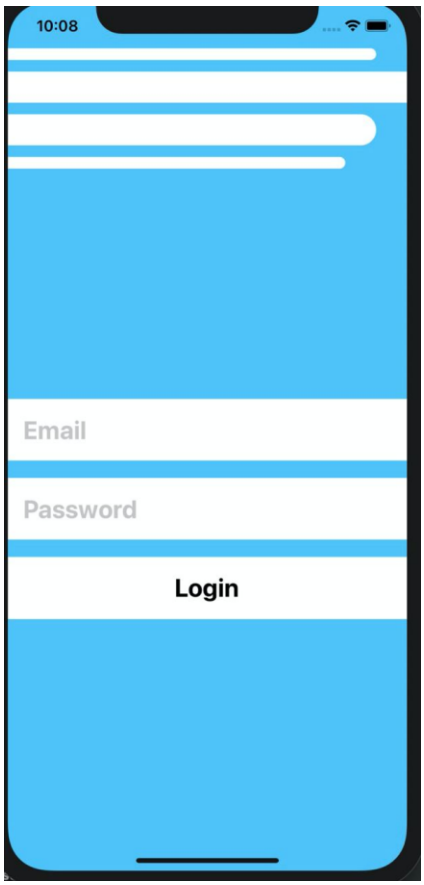

Рис. *34 Сторінка логіну*

Дії, які доступні для користувача на рисунках 32, 33 та 34, розділені на два підрозділа: реєстрація та логін.

При реєстрації та при логіні користувач заповнює імейл та пароль, натискає головну кнопку Register та пересувається до головного екрану з колекціями. Різниці для інтерфейсу між цими двома діями майже немає, але в логіці доцільно розділені Firebase Authentication вхід до облікового запису та реєстрація облікогово запису Firebase Authentication.

Слід відокремити особливості початкових сторінок: уся навігація крізь них адаптована під усі цільові платформи, та зроблена за стандартами iOS застосунку. Також оброблена можливість повертатися назад через свайп вліво, якщо користувач обрав не ту опцію, яку потрібно. Окрем цих можливостей, при запитах до сервера рендериться лоадер, який відображає стан застосунку щодо хмарних запитів.

На головному екрані колекцій (див. рис. 35) користувач може обрати до маніпулювання або Tasks (див. рис. 36), або Notes (див. рис. 37).

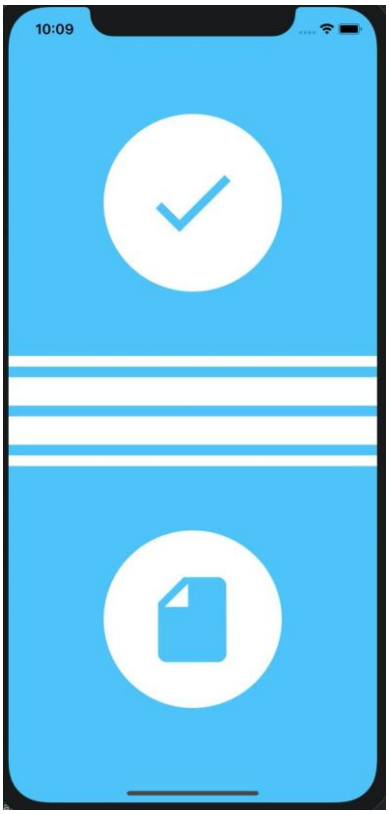

Рис. *35 Головна сторінка користувача*

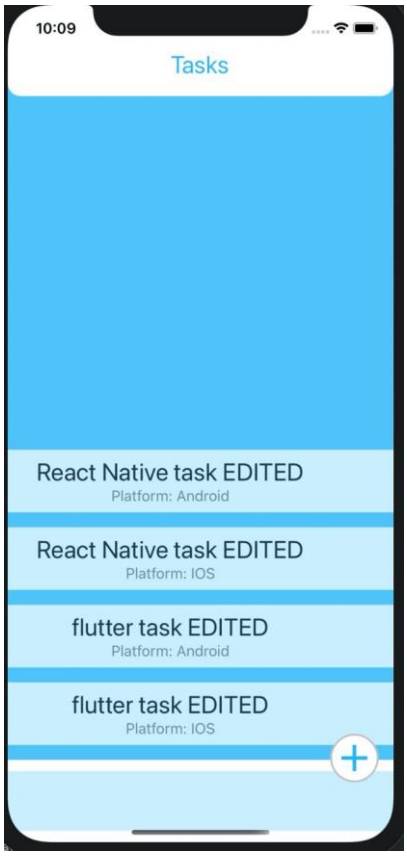

Рис. *36 Сторінка маніпулювання колекцією завдань*

Рис. *37 Сторінка маніпулювання колекцією записок*

Доступні для користувача дії маніпулювання з завданнями (див. рис. 38):

- створення завдання за допомогою заголовка та короткого опису, які не є обов’язковими при створенні;
- редагування завдання, зміна заголовку чи короткого опису;
- видалення завдання;
- переглядання (отримання з серверу для поточного користувача та їх відображення) завдань;
- зміна статусу окремого завдання.

Рис. *38 Окрема сторінка створення та редагування завдання*

Доступні для користувача дії маніпулювання з записками (див. рис. 39):

- створення записки за допомогою заголовку та багатострокового тіла, які не є обов’язковими при створенні;
- редагування записки, зміна заголовку чи багатострокового тіла;
- видалення записки;
- переглядання (отримання записок з серверу для поточного користувача та їх відображення) записок.

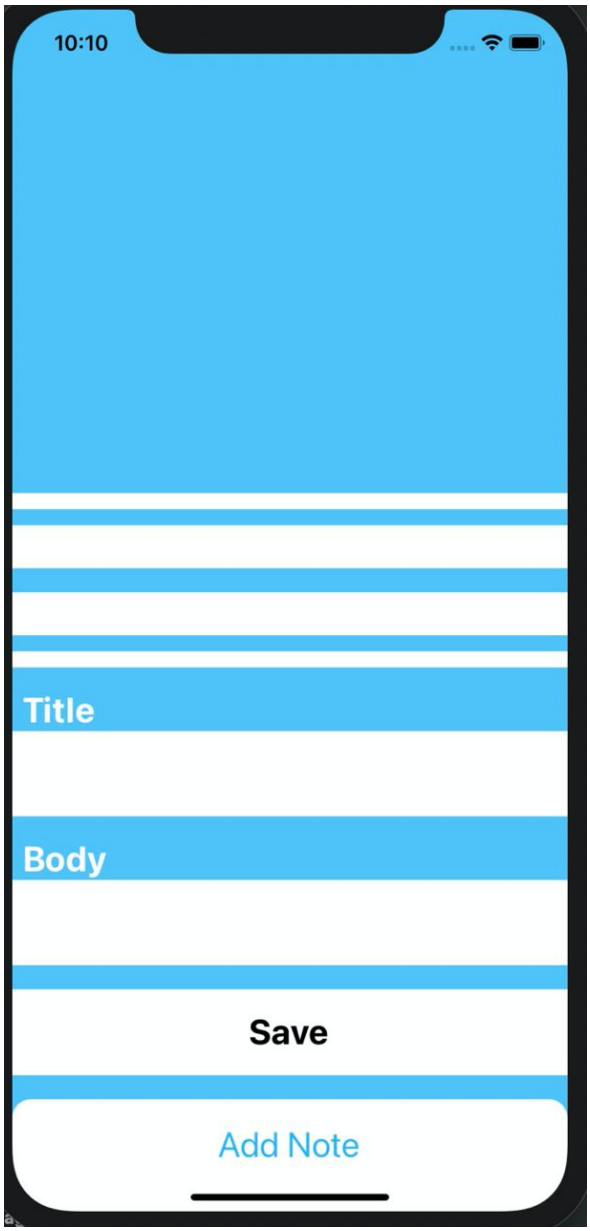

Рис. *39 Окрема сторінка створення та редагування записок*

3.5 Вимоги до апаратного забезпечення

Основною задачею клієнтського застосунку робота з колекціями завдань та записок відносно цього користувача за допомогою хмарного сховища у мережі. Обробка даних відбувається на віддаленому сервері. Тому були визначені наступні вимоги до клієнтського застосунку.

Мінімальні вимоги до апаратного забезпечення клієнту:

- Одноядерний процесор з базовою тактовою частотою 2.0 GHz.
- Оперативна пам’ять ємністю не менше 1 Гб.
- Інтернет з’єднання 5 мбіт.

Рекомендовані вимоги до апаратного забезпечення клієнту:

- Двох-ядерний процесор з базовою тактовою частотою 2.2 GHz.
- Оперативна пам’ять ємністю не менше 2 Гб.
- Інтернет з’єднання 20 мбіт.

Оскільки серверний застосунок контролюється непосередньо хмарною платформою Firebase, вимоги до забезпечення серверу відсутні.

3.6 Опис функціональних можливостей

Оскільки основною метою кваліфікаційної роботи є дослідження стану існуючих технологій для створення кросплатформенних мобільних застосунків та методів вирішення проблеми кросплатформенності, а не розробка кінцевого програмного продукту, то функціональні можливості системи орієнтовані на зручність проведення досліджень, а не для використання у якості готової програмної системи для роботи з користувацькими колекціями. Було реалізовано тестовий застосунок для створення, перегляду, редагування та видалення колекцій завдань та записок, цільовою

платформою котрого були сучасні мобільні операційні системи Android та iOS. Було досліджено зручність та якість платформ для створення кросплатформених мобільних застосунків взаємодії з хмарним сховищем Firebase. Також були досліджені якісні та кількісні характеристики кожного застосунка на кожній платформі та операційній системі поміж собою (порівняння застосунків від платформ поміж однаковими операційними системами).

РОЗДІЛ 4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ КРОСПЛАТФОРМЕНИХ МОБІЛЬНИХ ЗАСТОСУНКІВ

4.1 Дослідження сучасних технологій створення кросплатформених мобільних застосунків

Існує велика кількість технологій створення кросплатформених мобільних додатків, але не всі є зручними до використання та написання застосунків. Деякі з них не до кінця реалізовують інтерфейс платформи та виникають проблеми стосовно нативних сторін, які треба вирішувати за допомоги нативного коду. В рамках дослідження було обрано дві найсучасніші платформи: React Native та Flutter. Обрані системи відокремлюються від інших своєю зручністю, простотою рішень та поширеним ком'юніті.

Кожна з платформ має свої особливості щодо мови реалізації та її структури. Flutter використовує мову власної розробки Dart. React Native використовує дуже поширену мову програмування JavaScript, яка була доповнена у мові програмування TypeScript, яка виправила недоліки функціонального програмування.

Dart більш об'єктно-орієнтована мова, що вважається більш зручною до розробки застосунків, але у свою чергу JavaScript має доповнення у вигляді мови програмування TypeScript, яка виправляє недоліки функціонального програмування JavaScript. Написання коду досить зручне для обох платформ, вибір залишається на смак програміста.

На таблиці 1 досліджені мови реалізації платформ та їх структура.

Таблиця *1*

Мова реалізації платформи та їх структура

Платформа	**Мова**	**Структура**
React Native	JavaScript/TypeScript	Модульна
Flutter	Dart	Модульна

З точки зору зручності використання розглядалися такі показники як якість документації, підтримка різних програмних і апаратних середовищ виконання, підтримка нативних мов програмування, характеристики архітектури (наявність модулів, використання нативних директорій). Результати представлені в таблицях 2-5.

Таблиця *2*

Наявність документації

Платформа	**Документація**
React Native	Детальна онлайн документація
Flutter	Детальна онлайн документація; висока документованість модулів

Таблиця *3*

Підтримка різних операційних систем

Платформа	**Підтримка ОС**
React Native	Android, iOS, AndroidTV, MacOS, tvOS, Web, Windows
Flutter	Linux, MacOS, Windows, Android, iOS, Google Fuchsia, Web

Таблиця *4*

Характеристики архітектури платформ

Платформа	**Архітектура**
React Native	NPM модулі, має доступ до редагування нативних директорій окремо
Flutter	Flutter модулі, має доступ до редагування нативних директорій окремо

Таблиця *5*

Підтримувані нативні мови програмування

Платформа	**Мови**
React Native	Java, Kotlin, Objective-C, Swift
Flutter	Java, Kotlin, Objective-C, Swift, C++

Проаналізувавши отримані вище результати, можна скласти характеристику кожної з розглянутих платформ і виробити рекомендації щодо їх застосування.

React Native

Дана технологія здатна підтримувати досить великі проекти за допомоги найсучасніших мов програмування та модульності архітектури, створюючи дуже гнучкі застосунки з доступом до нативних налаштувань. Технологія реалізовує широкий спектр операційних систем, тому розробка на різні ОС буде коштувати замовникам не так багато, як нативні платформи для кожної окремо. Документованість платформи залишає лише позитивні відгуки зі сторони професійних розробників та груп людей, які обрали технологію як основну для розробки програмного проекту. Після введення повної підтримки TypeScript, платформа посилила підтримку об’єктно-орієнтованих функціональностей, тому перехід з інших патернових мов програмування буде досить швидкий та простий. Технологія реалізує доступ до нативних налаштувань проекту та написання власних нативних модулів до кожної операційної системи. Це є дуже корисним для величезних проектів, які потребують точного виконання технічного завдання. Швидкість підгрузки JavaScript/TypeScript коду in-fly до нативних платформ

Flutter

Технологія надає найсучасніший підхід до налаштувань модулів, нативних конфігурацій та структури проекту. Мова програмування Dart надає особливі можливості написання об’єктно-орінєтованого коду, який транслюється та виконується у нативних операційних системах. Платформа надає широкий простір щодо документації розробникам, яка має позитивний відгук ком’юніті. Flutter використовує власну модульну систему, яка теж має обсяжну документацію. Швидкість опрацювання релоадінгу

застосунку досить швидка, не затримує розробку навіть найвеличезніших проектів.

4.2 Порівняння оптимізації та зручності написання застосунків

При дослідженні були відокремлені такі характеристики та показники для порівняння: кількість рядків коду, кількість класів на застосунок, фізичні показники, такі як загруженість процесора, об'єм зайнятої оперативної пам'яті, частота оновлення екрану, швидкість виконання серверних запитів при однакових локальних даних.

На таблицях 6-12 можна побачити порівняння характеристик та показників для усіх додатків кожної платформи на цільових операційних системах. Усі характеристики були виміряні за допомоги реалізованого застосунку та інструментів розробника платформи.

Таблиця *6*

Рядків коду (LOC) на однаковий компонент

Платформа	**Кількість рядків коду LOC**
React Native	86
Flutter	110

Таблиця *7*

Рядків коду (LOC) на однакову сторінку

Платформа	**Кількість рядків коду LOC**
React Native	284
Flutter	352

Таблиця *8*

Класів на застосунок

Платформа	**Кількість класів**
React Native	3
Flutter	11

Таблиця *9*

Зайнятість процесора застосунком у середньому (CPU)

Платформа	**Зайнятість процесора iOS, %**	**Зайнятість процесора Android, %**
React Native	~4	~9
Flutter	~4	~10

Таблиця *10*

Кількість виділеної пам'яті застосунком (RAM)

Платформа	**Кількість виділеної пам'яті iOS, Мб**	**Кількість виділеної пам'яті Android, Мб**
React Native	143	120
Flutter	165	133

Таблиця *11*

Частота оновлення екрану (FPS)

Платформа	**Кількість кадрів iOS**	**Кількість кадрів Android**
React Native	~58	~55
Flutter	~59	~54

Таблиця *12*

Час виконання запитів до хмарного сховища

Платформа	**Час виконання запиту iOS, мс**	**Час виконання запиту Android, мс**
React Native	~250	~275
Flutter	~300	~275

Проаналізувавши отримані вище результати, можна скласти характеристики зручності та оптимізованості кожної з платформ та виробити рекомендації щодо їх застосування.

React Native

Експериментальним шляхом досліджено, що платформа не вимагає дуже багато коду, щоб реалізовувати логіку з інтерфейсом. TypeScript вніс достатньо об’єктно-орієнтованих функціональностей, але ця платформа перш усього функціональна, тому функціональне програмування переважає над об’єктно-орієнтованим. Кількість класів, які були розроблені – 3.

За фізичними показниками платформа є дуже швидкою та займає менше оперативної пам’яті, ніж Flutter, на обох мобільних операційних системах. Частота оновлення екрану у деяких місцях анімації (переходів між екранами, запуску лоадерів тощо) на iOS падає до 55, усі інші взаємодії мають 59-60 кадрів. На Android ОС частота оновлення у середньому 55 кадрів, що на 1 більше, ніж у Flutter Android застосунку. Виконання серверних запитів приблизно одна й та сама, але на iOS системі React Native показує себе краще.

Flutter

Експериментальним шляхом досліджено, що платформа вимагає декілька більше зусиль для написання однакового інтерфейсу з логікою, ніж на React Native платформі. Але тут є об’єктно-орієнтоване розділення, яке компенсує більш великі файли класів. Кількість класів, які були розроблені – 11.

За фізичними показниками платформа є дуже швидкою, але займає декілька більше оперативної пам’яті, ніж React Native, на обох мобільних операційних системах. Частота оновлення екрану є більш стабільною та майже не покидає показник у 60 кадрів на iOS, що є дуже вагомою характеристикою. На Android ОС частота оновлення у середньому 54 кадра, що на 1 менше, ніж у React Native Android застосунку. Виконання серверних запитів приблизно одна й та сама, але на iOS системі React Native показує більш швидкий результат.

Результати досліджень

Обидві платформи є життєздатними до створення кросплатформених мобільних застосунків. Flutter та React Native у багатьох показниках сходяться до одного значення, але є й різниця. Flutter треба обирати розробникам, які більш схильні до об’єктно-орієнтованого підходу. React Native можна обирати, коли є схильність до функціонального програмування. За фізичними показниками обидві платформи показали відмінний результат. Flutter займає більше ресурсів для збереження стабільності застосунку. Взаємодія з віддаленим сервером досить швидка, але React Native перемагає за середніми показниками.

ВИСНОВКИ

1. Встановлено актуальність розробки кросплатформених мобільних застосунків. Шляхом аналізу технологій для створення кросплатформених мобільних додатків, було сформовано набір з платформ, що відповідають найсучаснішим нормам, для дослідження, проведено їх порівняння та аналіз, а також було визначено стек технологій та інструментів розробки, визначено їх відносну продуктивність, ступінь інтеграції та зручність використання.

2. Досліджено сучасні методи створення кросплатформених мобільних додатків.

3. Досліджена можливість використання систем хмарних сховищ у поєднанні з платформами створення кросплатформених мобільних додатків: управління колекціями та ведення користувацьких облікових записів.

4. Основним інструментом для реалізації кросплатформених мобільних застосунків обрано мови програмування JavaScript, TypeScript та Dart, які надають можливість використовувати додаток на будь-якій платформі, транслюючи вихідний код до нативних операційних систем.

5. Розроблено програмний застосунок для демонстрації можливостей технологій створення застосунків під основні мобільні операційні системи: Android та iOS.

6. Налаштовано серверний сервіс автентифікації та хмарного сховища даних.

7. Досліджені існуючі технології створення кросплатформених мобільних застосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андрєєв Роман, студент магістратури ФЕЕІТ ІІ ЗНУ. Наук. кер.: к.ф.-м.н., доц. Попівщий В.І. «ВИКОРИСТАННЯ СУЧАСНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ КРОСПЛАТФОРМНИХ МОБІЛЬНИХ ЗАСТОСУНКІВ». Збірник наукових праць студентів, аспірантів і молодих вчених «Молода наука-2020» : у 5 т. Запорізький національний університет. Запоріжжя: ЗНУ, 2020. Т.5. С. 71-73.

2. Андрєєв Р. Т., магістрант, Попівщий В.І., доцент, к.ф.-м.н. – науковий керівник. Сучасні технології створення кросплатформних мобільних додатків. МАТЕРІАЛИ XXV науково-технічної конференції студентів, магістрантів, аспірантів, молодих вчених та викладачів. Запоріжжя : ЗНУ, 2020. С. 155.

3. A. Boduch, React and React Native: A complete hands-on guide to modern web and mobile development with React.js, 3rd Edition. 2020. 526 с.

4. Adam D. Scott, JavaScript Everywhere: Building Cross-Platform Applications with GraphQL, React, React Native, and Electron. 2020. 344 с.

5. D. Ward, React Native Cookbook: Step-by-step recipes for solving common React Native development problems, 2nd Edition. 2019. 618 с.

6. R. Derks, React.js Projects: Build 12 real-world applications from scratch using React, React Native, and React 360. 2020. 432 с.

7. A. Paul, React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android Applications, 2nd Edition. 2019. 237 с.

8. N. Dabit, React Native in Action. 2019. 320 с.

9. F. Zammetti, Practical React Native: Build Two Full Projects and One Full Game using React Native. 2018. 334 с.

10. M. Grzesiukiewicz, Hands-On Design Patterns with React Native: Proven techniques and patterns for efficient native mobile development with JavaScript. 2018. 302 с.

11. A. Boduch, React and React Native, 2nd Edition. 2018. 540 c.

12. D. Abbot, Fullstack React Native: The Complete Guide to React Native. 2017. 552 c.

13. E. Rodriguez Martinez, React: Cross-Platform Application Development with React Native. 2018. 182 c.

14. E. Bush, JavaScript Applications with Node.js, React, React Native and MongoDB. 2018. 392 c.

15. B. Eisenman, Learning React Native: Building Native Mobile Apps with JavaScript, 2nd Edition. 2017. 244 c.

16. J. Lebensold, React Native Cookbook: Bringing the Web to Native Platforms. 2018. 176 c.

17. A. Boduch, React and React Native. 2017. 604 c.

18. E. R. Martinez, React Native Blueprints. 2017. 346 c.

19. V. Novick, React Native – Building Mobile Apps with JavaScript. 2017. 434 c.

20. C. Villa, S. Bershadskiy, React Native Cookbook. 2017. 437 c.

21. B. Burd, Flutter For Dummies. 2020. 384 c.

22. A. Singh, Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter: Build scalable real-world projects to implement end-to-end neural networks on Android and iOS. 2020. 380 c.

23. S. Alessandria, Flutter Projects: A practical, project-based guide to building real-world cross-platform mobile applications and games. 2020. 490 c.

24. C. Zaccagnino, Programming Flutter: Native, Cross-Platform Apps the Easy Way. 2020. 370 c.

25. A. Biessek, Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2. 2019. 512 c.

26. E. Windmill, Flutter in Action. 2020. 310 c.

27. R. Payne, Beginning App Development with Flutter: Create Cross-Platform Mobile Apps. 2019. 309 c.

28. F. Cheng, Flutter Recipes: Mobile Development Solutions for iOS and Android. 2019. 522 c.

29. M. L. Napoli, Beginning Flutter: A Hands On Guide to App Development. 2019. 528 c.

30. F. Zammetti, Practical Flutter: Improve your Mobile Development with Google’s Latest Open-Source SDK. 2019. 396 c.

31. P. Mainkar, Google Flutter Mobile Development Quick Start Guide: Get up and running with iOS and Android mobile app development. 2019. 152 c.

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Андрєєв Роман Тарасович, студент _2_ курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти sp115-01@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Дослідження технологій створення кросплатформених мобільних застосунків**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2020 Підпис _____Андрєєв Роман Тарасович (студент)

Дата 30.11.2020 Підпис _____ Попівщий Василь Іванович (науковий керівник)