**Міністерство освіти і науки України**
**Національний технічний університет**
**«Дніпровська політехніка»**

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

**Кафедра** Програмного забезпечення комп'ютерних систем
(повна назва)

# ПОЯСНЮВАЛЬНА ЗАПИСКА
## кваліфікаційної роботи ступеня

*магістра*
(назва освітньо-кваліфікаційного рівня)

| | |
|---|---|
| **студента** | *Попової Єлизавети Сергіївни* |
| | (ПІБ) |
| **академічної групи** | *122М-21-1* |
| | (шифр) |
| **спеціальності** | *122 Комп'ютерні науки* |
| | (код і назва спеціальності) |
| **освітньої програми** | *«Комп'ютерні науки»* |
| | (назва освітньої програми) |
| **на тему:** | *Дослідження ефективності фреймворку Flutter при розробці високодинамічних мобільних додатків* |

_____*Є.С. Попова*

| **Керівники** | **Прізвище, ініціали** | **Оцінка за шкалою** | | **Підпис** |
|---|---|---|---|---|
| | | **рейтинговою** | **інституційною** | |
| розділів кваліфікаційної роботи | | | | |
| **спеціальний** | *проф. Мещеряков Л.І.* | 98 | відмінно | |
| | | | | |

| | | | | |
|---|---|---|---|---|
| **Рецензент** | | | | |

| | | | | |
|---|---|---|---|---|
| **Нормоконтролер** | *проф. Лактіонов І.С.* | | | |

**Дніпро**
**2022**

**Міністерство освіти і науки України**
**Національний технічний університет**
**«Дніпровська політехніка»**

**ЗАТВЕРДЖЕНО:**
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 20 22 Року

# ЗАВДАННЯ
**на виконання кваліфікаційної роботи**

**спеціальності** *122 Комп'ютерні науки*
*(код і назва спеціальності)*

**студенту** *122м-21-1* *Попової Єлизавети Сергіївни*
(група) (прізвище та ініціали)

**Тема кваліфікаційної роботи** *Дослідження ефективності фреймворку Flutter при розробці високодинамічних мобільних додатків*

## 1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 31.10.2022 р. № 1200 -с

## 2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

**Об'єкт досліджень** – практичне застосування фреймворку Flutter з метою підвищення рівня оптимізації розробки високодинамічних мобільних додатків на iOS та Android платформах.

**Предмет досліджень** – декларативний підхід фреймворку Flutter при розробці програмного забезпечення для мобільних пристроїв на основі єдиної бази коду.

**Мета НДР** – проведення дослідження ефективності фреймворку Flutter, як інструменту для розробки високодинамічних нативних мобільних додатків під різні операційні системи на замовлення бізнесу.

**Вихідні дані для проведення роботи** – теоретичні та експериментальні дослідження на основі декларативного підходу до побудови мобільних інтерфейсів фреймворку Flutter.

## 3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

**Новизна запропонованих рішень** полягає у запропонованому та обґрунтованому способі удосконалення та пришвидшення роботи над

високодинамічними мобільними додатками за допомогою фреймворку Flutter, який, в свою чергу, гарантує ефективність роботи системи завдяки декларативному підходу до побудови мобільних інтерфейсів.

**Практична цінність** полягає у створенні високодинамічного мобільного додатку, який дозволить оцінити переваги та ефективність проектування оптимізованих мобільних додатків за допомогою фреймворку Flutter, під різні операційні системи, на основі єдиної бази коду, та гнучкого декларативного підходу до побудови інтерфейсу.

## 4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання нечіткої моделі даних. В результаті роботи повинен бути розроблений програмний комплекс для вирішення задачі ідентифікаційної експертизи на основі нечіткої моделі даних.

## 5 ЕТАПИ ВИКОНАННЯ РОБІТ

| **Найменування етапів робіт** | **Строки виконання робіт (початок – кінець** |
|---|---|
| Аналіз теми та постановка задачі | 12.09.2022-30.09.2022 |
| Побудова нечіткої моделі представлення даних для вирішення задачі ідентифікаційної експертизи | 01.10.2022-31.10.2022 |
| Створення автоматизованої системи для вирішення задачі ідентифікаційної експертизи бензинів | 01.11.2022-01.12.2022 |

## 6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

**Економічний ефект** від реалізації результатів роботи очікується позитивним завдяки скороченню кіл-сті розробників, і відповідно, скороченню затрат на заробітну плату, для реалізації розробки додатків під різні пристрої на замовлення бізнесу.

**Соціальний ефект** від реалізації результатів роботи очікується позитивним завдяки удосконаленню підходу до побудови інтерфейсів мобільних додатків під різні платформи, що також значно дозволить зменшити загальний час на розробку.

## 7 ДОДАТКОВІ ВИМОГИ

Завдання видав _____ *Мещеряков Л.І.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Попова Є.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *05.09.2022 р.*

Термін подання кваліфікаційної роботи до ЕК *09.12.2022*

# РЕФЕРАТ

Пояснювальна записка: 115 стор., 38 рис., 4 додатки, 51 джерело.

**Об'єкт дослідження**: практичне застосування фреймворку Flutter з метою підвищення рівня оптимізації розробки високодинамічних мобільних додатків на iOS та Android платформах.

**Предмет дослідження**: декларативний підхід до побудови мобільних інтерфейсів фреймворку Flutter при розробці програмного забезпечення для мобільних пристроїв на основі єдиної бази коду.

**Мета роботи:** проведення дослідження ефективності фреймворку Flutter, як інструменту для розробки високодинамічних нативних мобільних додатків під різні операційні системи на замовлення бізнесу.

**Методи дослідження.** Для вирішення поставлених задач використані методи: принципи використання засобів крос-платформного програмування, принципи декларативного програмування, об'єктно-орієнтоване програмування.

**Новизна отриманих результатів** полягає у запропонованому та обґрунтованому способі удосконалення та пришвидшення роботи над високодинамічними мобільними додатками за допомогою фреймворку Flutter, який, в свою чергу, гарантує ефективність роботи системи завдяки декларативному підходу до побудови мобільних інтерфейсів.

**Практична цінність результатів** полягає у створенні високодинамічного мобільного додатку, який надасть можливість оцінити переваги та ефективність проектування оптимізованих мобільних додатків за допомогою фреймворку Flutter під різні операційні системи, на основі єдиної бази коду та гнучкого декларативного підходу до побудови мобільного інтерфейсу.

**Область застосування.** Розроблена інформаційна система може застосовуватися на замовлення бізнесу, а також як навчальний матеріал для дослідження студентами, для отримання ними певних навичок і знань.

**Значення роботи та висновки.** Удосконалений підхід до побудови інтерфейсу мобільних додатків дозволяє розробляти додатки зі значним скороченням як матеріальних витрат, так і тимчасових, що підтверджується розробленим програмним продуктом в даній роботі.

**Прогнози щодо розвитку досліджень.** Покращити інформаційну систему, додавши можливість авторизації користувачів, здійснення ними дзвінка, та можливісті отримання повідомлень під час роботи додатку у фоновому режимі, з метою поліпшення оптимізації мобільного додатку.

**Список ключових слів:** мобільні додатки, операційні системи, крос-платформна розробка, iOS, Android, Flutter, Dart, React, React Native.

## ABSTRACT

Explanatory note: 115 pages, 38 figures, 4 appendices, 51 sources.

**Object of research**: practical application of the Flutter framework in order to increase the level of optimization of the development of highly dynamic mobile applications on iOS and Android platforms through a single code base.

**Subject of research**: declarative approach to building mobile interfaces of the Flutter framework in the development of software for mobile devices based on a single code base.

**Purpose of Master's thesis**: to conduct a study of the effectiveness of the Flutter framework as a tool for developing highly dynamic native mobile applications for different operating systems for business.

**Research methods**. To solve the tasks used methods: principles of using cross-platform programming tools, principles of declarative programming, object-oriented programming (OOP).

**Originality of research** is the proposed and substantiated way to improve and accelerate the work on highly dynamic mobile applications using the Flutter framework, which, in turn, guarantees the efficiency of the system due to the declarative approach to building mobile interfaces.

**Practical value of the results** is to create a highly dynamic mobile application that will provide an opportunity to evaluate the benefits and efficiency of designing optimized mobile applications using the Flutter framework for different operating systems, based on a single code base and a flexible declarative approach to building a mobile interface.

**Scope of application**. The developed information system can be used by order of business, as well as as a teaching material for students to study, to obtain certain skills and knowledge.

**The value of the work and conclusions**. An improved approach to building the interface of mobile applications allows you to develop applications with a significant reduction in both material and time costs, as evidenced by the developed software product in this work.

**Research forecast and development**. To improve the information system by adding the ability to authorize users, making a call, and the ability to receive notifications while the application is running in the background, in order to improve the optimization of the mobile application.



**List of keywords**: mobile applications, operating systems, cross-platform development, iOS, Android, Flutter, Dart, React, React Native.

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система;

ПЗ – програмне забезпечення;

SDK – Software development kit;

ART – Android Runtime, середовище виконання Android додатків;

DEX – Dalvik Executable, файл з байткодом для Dalvik VM;

HTML – HyperText Markup Language;

CSS – Cascading Style Sheets;

PWA – Progressive Web App, прогресивний вебзастосунок;

AOT – Ahead-of-Time компіляція, компіляція до роботи програми;

API – Application Programming Interface;

FPS – Frames Per Second, кількість кадрів на секунду;

JIT – Just-In-Time компіляція, компіляція під час роботи програми;

JS – JavaScript;

JSX – JavaScript XML;

BLoC – Business Logic Components:

UI – User Interface;

UX – User Experience.

**ЗМІСТ**

# ВСТУП

**Актуальність теми.** На сьогоднішній день мобільні додатки беруть безпосередню участь у нашому повсякденному житті. Існує цілий ряд мобільних додатків, які ми використовуємо для полегшення нашої повсякденної діяльності: соціальні мережі, електронна пошта, онлайн покупки, транспортні додатки, додатки для обміну текстовими та відеоповідомленнями тощо.

Існують мільйони додатків, доступних для нас у різних магазинах мобільних додатків, таких як Google Play Store та App Store. Постійно зростаюче використання мобільних додатків спонукає компанії та індивідуальних розробників інвестувати свій час та гроші в розробку додатків, орієнтованих на цих користувачів. Такий високий попит змусив компанії та розробників шукати нові способи розробки мобільних додатків для охоплення великої кількості користувачів з меншими витратами на розробку. Різноманітність мобільних платформ була основною перешкодою для бізнесу та розробників для пошуку нових шляхів розробки мобільних додатків, а не слідування економічно неефективним методам розробки мобільних додатків орієнтованих під конкретну платформу. Крім того, розробка мобільних додатків потребує частих удосконалень та адаптацій, щоб відповідати швидкозмінним вимогам зручності використання.

Виробники портативних пристроїв все частіше адаптують нові дизайни, які варіюються від зміни розмірів екрану пристрою до поліпшення можливостей пристрою. Тож розробка мобільних додатків нині є однією з найпопулярніших завдань в сфері інформаційних технологій. Зростає потреба в розробці якісного і

відповідного сучасним тенденціям продукту.

Успіх, відповідно, як і якість мобільного додатку залежить від вибору платформи та технологій для його розробки. Саме вони зумовлюють життєздатність і конкурентоспроможність додатку, його функціонал, масштабованість і складність обслуговування. Також це впливає і на вартість розробки. Сьогодні існують різні методи і фреймворки, які допомагають створювати мобільні додатки. Таким чином, можливості сучасних технологій розробки дозволяють створювати мобільні додатки різної складності.

Тож вибір методу розробки мобільного додатка − це дуже важливий етап в його розробці, на який впливають кілька факторів, таких як технічна оцінка розробників, потреба в доступі до інформації на пристрої, вплив швидкості інтернету на додаток. Вибір тієї чи іншої платформи також залежить від бюджету і вимог, що пред'являються до майбутнього додатку[12].

Тобто мобільний додаток повинен адаптуватися до всіх цих нових змін, щоб зберегти своїх користувачів. Хоча існують мільйони мобільних додатків на різних ринках додатків, їх загалом можна розділити на три категорії: додатки для конкретних платформ, адаптивні веб-додатки та крос-платформні нативні додатки.

**Мета і задачі дослідження.** Метою роботи є дослідження ефективності фреймворку Flutter, як інструменту для розробки високодинамічних нативних мобільних додатків під різні операційні системи на замовлення бізнесу.

Для досягнення мети дослідження необхідно розв’язати наступні задачі:

1. Провести аналіз існуючих технологій, які дозволяють займатися одночасною розробкою під різні операційні системи. (здійснити вибір інструментальних засобів розробки мобільного додатку, дослідивши особливості розробки додатків для операційних систем Android та iOS.)
2. Проаналізувати нове рішення від Google – фреймворк Flutter як засіб для розробки мобільних додатків.
3. Здійснити порівняння та побудувати структурні моделі усіх можливих рішень для розробки програмного забезпечення для мобільних операційних систем.
4. Розробити додаток за допомогою фреймворку Flutter для визначення

ефективності програмного забезпечення для мобільних пристроїв.

5. Підвести підсумки щодо вибору найкращьої технології для розробки мобільного програмного забезпечення.

**Об’єктом досліджень** є практичне застосування фреймворку Flutter з метою підвищення рівня оптимізації розробки високодинамічних мобільних додатків на iOS та Android платформах за рахунок єдиної бази коду.

**Предметом досліджень** є декларативний підхід до побудови мобільних інтерфейсів фреймворку Flutter при розробці програмного забезпечення для мобільних пристроїв на основі єдиної бази коду.

**Методи дослідження**. Для вирішення поставлених задач використані методи: принципи використання засобів крос-платформного програмування, принципи декларативного програмування, об’єктно-орієнтоване програмування.

**Наукова новизна роботи:** полягає у запропонованому та обґрунтованому способі удосконалення та пришвидшення роботи над високодинамічними мобільними додатками за допомогою фреймворку Flutter, який в свою чергу гарантує ефективність роботи системи завдяки декларативному підходу до побудови мобільних інтерфейсів та єдиної бази коду.

**Практичне значення** отриманих в роботі результатів полягає в тому, що розроблену систему можна використовувати для створення платформоорієнтованих високодинамічних користувацьких додатків із спільною кодовою базою, яка дозволяє: зробити розроблення користувацьких додатків більш гнучким; скоротити витрати грошових та управлінських ресурсів на велику команду розробників. Через використання Flutter в якості базової платформи, за допомогою розробленої системи можна створити додаток, який буде зібрано на більше ніж п’яти платформ.

**Особистий внесок автора.** Було створено соціальну мережу, за допомогою якої можна дослідити ефективність фреймворку Flutter при розробці мобільних додатків для таких платформ, як iOS та Android. Також було здійснено дослідження фреймворку Flutter та його порівняння з іншими аналогами розробки кросплатформенних та нативних додатків для мобільних пристроїв.

**Структура та обсяг кваліфікаційної роботи.** Відповідно до мети, завдань, і предмета дослідження, кваліфікаційна робота складається з реферату, вступу, трьох основних розділів і висновків, списку використаних джерел та 4 додатків. Загальний обсяг роботи містить 115 сторінок друкованого тексту, із них основної частини - 63 сторінок з 38 рис., спеціальної – 19 сторінок, списку використаних джерел з 51 найменуванням на 4 сторінках, 4 додатках на 29 сторінках.

# РОЗДІЛ 1
# АНАЛІЗ ІНСТРУМЕНТІВ, МЕТОДІВ ТА ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

## 1.1. Аналіз поточної ситуації в розробці мобільних додатків

### 1.1.1 Актуальність мобільних пристроїв

Згідно результатів досліджень глобальної аналітичної компанії Statista – користувачі проводять більше часу в смартфонах, ніж на будь-якому іншому пристрої. Зважаючи на це, не дивно, що використання мобільних телефонів з часом лише зростає (рис. 1.1). Щодня половина користувачів смартфонами проводить за ними понад 5 годин 24 хвилини, а 26% – більше 7 годин на день. Власник смартфона перевіряє свій мобільний телефон одразу після пробудження зранку та безпосередньо перед відходом до сну [5]. У середньому користувачі перевіряють свої телефони щонайменше 96 разів на день, або раз на десять хвилин.


Рис. 1.1. Статистика проведення часу користувачів у смартфоні

Також стрімке зростання мобільних додатків пояснюється тим, що швидкісний мобільний інтернет стає доступнішим, як і недорогі смартфони. Так за даними звітів Statista станом на 2022 рік - у світі налічується приблизно 6,65 мільярда користувачів смартфонів (рис. 1.2). Це 86% світового населення. Досить величезний відсоток, враховуючи той факт, що у 2016 році кількість користувачів смартфонів у світі становила лише 3,67 мільярда, або 45% від загальної кількості населення на той час.

Крім того, очікується, що поширеність мобільних телефонів стане більш помітною. Прогнозується, що до 2026 року кількість користувачів смартфонів становитиме 7,52 мільярда. Це на 12% більше, ніж поточні 6,65 мільярда.

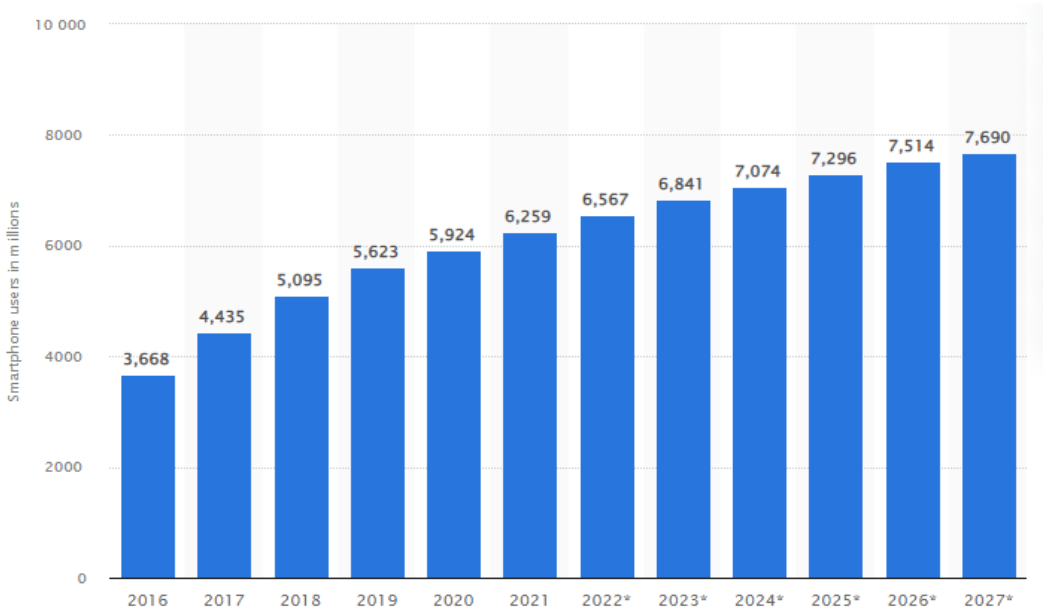


Рис. 1.2. Кількість власників смартфонів в усьому світі з 2016 по 2021 рік, з прогнозами з 2022 по 2027 рік

Наразі ж статистика порталу Statista.com надає інформацію про глобальний трафік мобільних даних з 2015 по 2022 рік. З цього виходить, що починаючи с 2019 року, відсоток генерування трафіку з мобільних пристроїв вже перевищив відсоток генерування з персональних комп’ютерів , а на даний момент - на мобільні пристрої припадає приблизно половина світового веб-трафіку: у другому кварталі 2022 року на мобільні пристрої припадало 58,99 відсотка глобального трафіку веб-сайтів, постійно коливаючись біля 50-відсоткової позначки з початку 2017 року, перш ніж остаточно перевершити її в 2020 році (рис. 1.3).

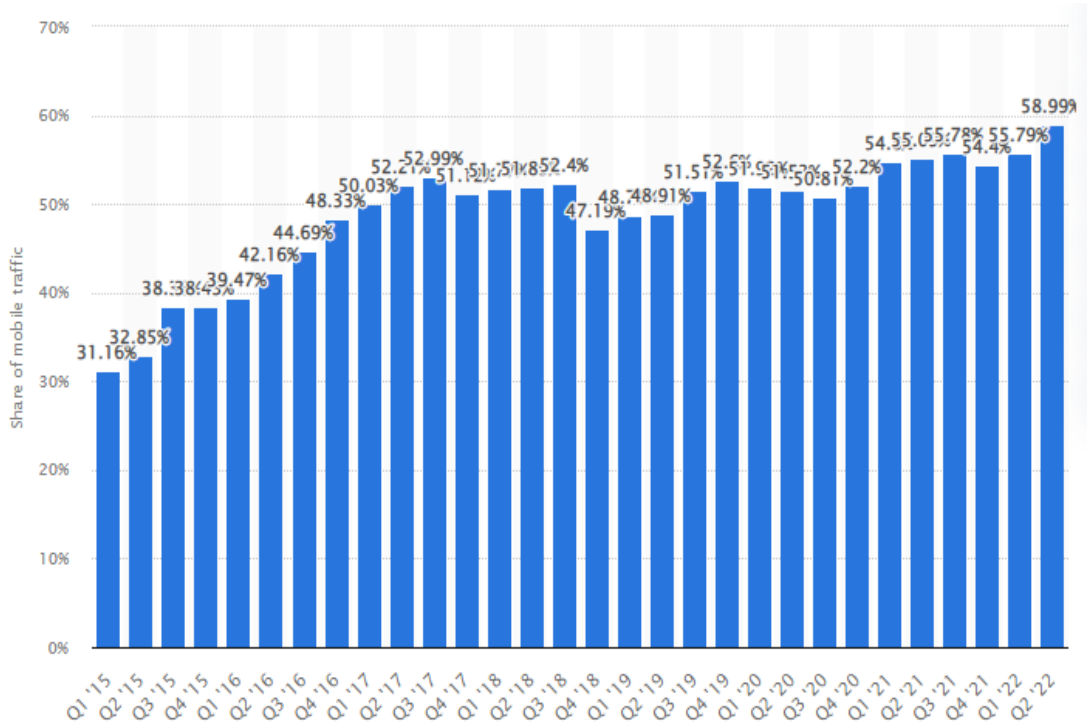


Рис. 1.3. Відсоток відвідуваності веб-сайтів мобільних пристроїв у всьому світі з 1 кварталу 2015 року до 2 кварталу 2022 року

### 1.1.2 Аналіз мобільних операційних систем

За весь час існування смартфонів з'явилася чимала кількість мобільних операційних систем, таких як Samsung Bada, Symbian, Firefox OS і Windows Mobile/Phone. Але жодна з них не змогла скласти гідну конкуренцію Android і iOS від компаній Google і Apple відповідно.

На сьогодні картина поширеності ОС в світі згідно ірландському сервісу StatCounter займається аналізом трафіку в інтернеті та має вигляд (рис. 1.5). На операційній системі Android щорічно з'являється все більше і більше нових виробників дешевих мобільних пристроїв, що і пояснює її популярність, бо чим багатша країна, тим вище відсоток поширеності в ній iOS (що пов'язано, з більш високою вартістю техніки Apple).

Так, в США і Сполученому Королівстві пристроїв на iOS належить 46% ринку, в Норвегії - 44%, в Бельгії - 38%. Тож при розробці мобільного додатку слід орієнтується на підтримку лідерів мобільних ОС – Android та iOS.


Рис. 1.5. Відсоток розповсюдженості мобільних ОС у світі

## 1.2 Аналіз поточної ситуації щодо існуючих рішень

Ринок мобільних додатків сьогодні залишається одним зі стрімко зростаючих сегментів ринку програмного забезпечення. Це зумовлено удосконаленням можливостей мобільних девайсів, що дозволяє виконувати все більше завдань різної складності, без допомоги існуючих рішень для стаціонарних комп'ютерів. Також розвиваються нові фреймворки для розробки крос-платформенних додатків, які мають за мету зменшення витрат на розробку мобільних додатків, які можуть бути використані користувачами різних операційних систем, та покриття більшого сегменту ринку.

На даний момент можна виділити наступні типи додатків для мобільних пристроїв:

- нативні додатки;
- крос-платформні додатки;
- прогресивні веб-додатки;
- гібридні додатки.

Кожен із цих підходів до розробки мобільних додатків має свій набір

переваг і недоліків. Вибираючи правильний підхід до розробки для своїх проектів, розробники враховують бажану взаємодію з користувачем, обчислювальні ресурси та власні функції, необхідні для програми, бюджет розробки, часові цілі та ресурси, доступні для підтримки програми.

### 1.2.1 Нативні технології розробки мобільних додатків

Нативний додаток – це застосунок, встановлена на мобільному пристрої, програмне забезпечення якого, спеціально запрограмоване та адаптоване дляопераційної системи та відповідних кінцевих пристроїв. Нативні додатки найчастіше розробляються для Android і Apple iOS, оскільки ці дві операційні системи найбільш широко представлені на ринку і, маючи частку ринку близько 99%, фактично складають весь ринок. Крім Android та iOS, Windows Phone, як і раніше, відіграє невелику роль, але більшість компаній нехтують розробкою нативних додатків для Windows Phone через його низьку частку наринку.

Нативні застосунки можуть взаємодіяти з обладнанням кінцевого пристрою та використовувати його. Наприклад, можна отримати доступ до камери, GPS, гіроскопа або інших датчиків на мобільному пристрої. Крім того, нативний додаток може отримати доступ до існуючого або попередньо встановленого програмного забезпечення на смартфоні або планшеті. Це означає, що можна отримати доступ до даних та оперативної пам'яті пристрою, і, наприклад, зображення зі смартфона можна використовувати або редагуватиу застосунку.

Нативні додатки насправді є пропрієтарним програмним забезпеченням, а це означає, що користувачам або третім особам дуже складно вносити змінита коригувати додаток. Вихідні код додатків зазвичай знаходяться під замком, тому їх не можна просто скопіювати. Тим не менш, зареєстровані розробники можуть отримати уявлення про код через відповідне середовище розробки.

Також для розробки додатків важливо, щоб вони були доступні для конкретної операційної системи «див. табл. 1.1». Це може бути Android чи iOS. Додаток повинна бути точно узгодженим з обладнанням. Також важливу роль у

розробці та встановлені додатків відіграє те, що програмне забезпечення може бути встановлене на пристроях, з якими додаток може взаємодіяти або, принаймні, бути сумісним.

Таблиця 1.1

**Зведена таблиця порівняння операційних систем**

| | Android | iOS |
|---|---|---|
| Programming Language | Java, C/C++, Kotlin | Objective-C, Swift |
| IDE | Android Studio, Eclipse | Xcode |
| SDK | Android SDK/NDK | iOS SDK |

Android – це операційна система для мобільних пристроїв, розроблена Google і яка вперше з'явилася на ринку у вересні 2008 року. Остання версія програмного забезпечення – Android 12. Google надає його безкоштовно виробникам смартфонів та планшетів, які можуть налаштувати інтерфейс, щоб відрізняти свої продукти від продуктів конкурентів. У свою чергу, вони повинні дотримуватися технічної специфікації, наданої Google, і інтегрувати найважливіші сервіси північноамериканського гіганта (GMail, Maps, GoogleNow, Chrome).

Через велику кількість компаній і, отже, безліч кінцевих пристроїв, Android є найбільш широко використовуваною операційною системою для мобільних кінцевих пристроїв і займає близько 72,44% ринку мобільних ОС.

Ядро Linux є технічною основою Google Android (рис. 1.6). Нативні додатки для Android написані мовами програмування Java та Kotlin. Щоб розробитизастосунок для Android, на додаток до відповідних знань у галузі програмування потрібен комплект розробника програмного забезпечення(SDK). SDK – це середовище розробки, в якому розробляються додатки. SDKдля розробки нативних застосунків для Android називається Android Studio і також був розроблений Google.

Android Studio – це програма з відкритим кодом, яку можна безкоштовно завантажити з Інтернету. Крім Android Studio, для програмування додатків Android потрібно ще одне програмне забезпечення – Java Development Kit(JDK), який також доступний для безкоштовного завантаження. Обидві програми можна встановити на будь-який комп'ютер, і відразу розпочати розробку. Завдяки природі програмного забезпечення з відкритим вихідним кодом, кожен може розробити нативний додаток для Android.

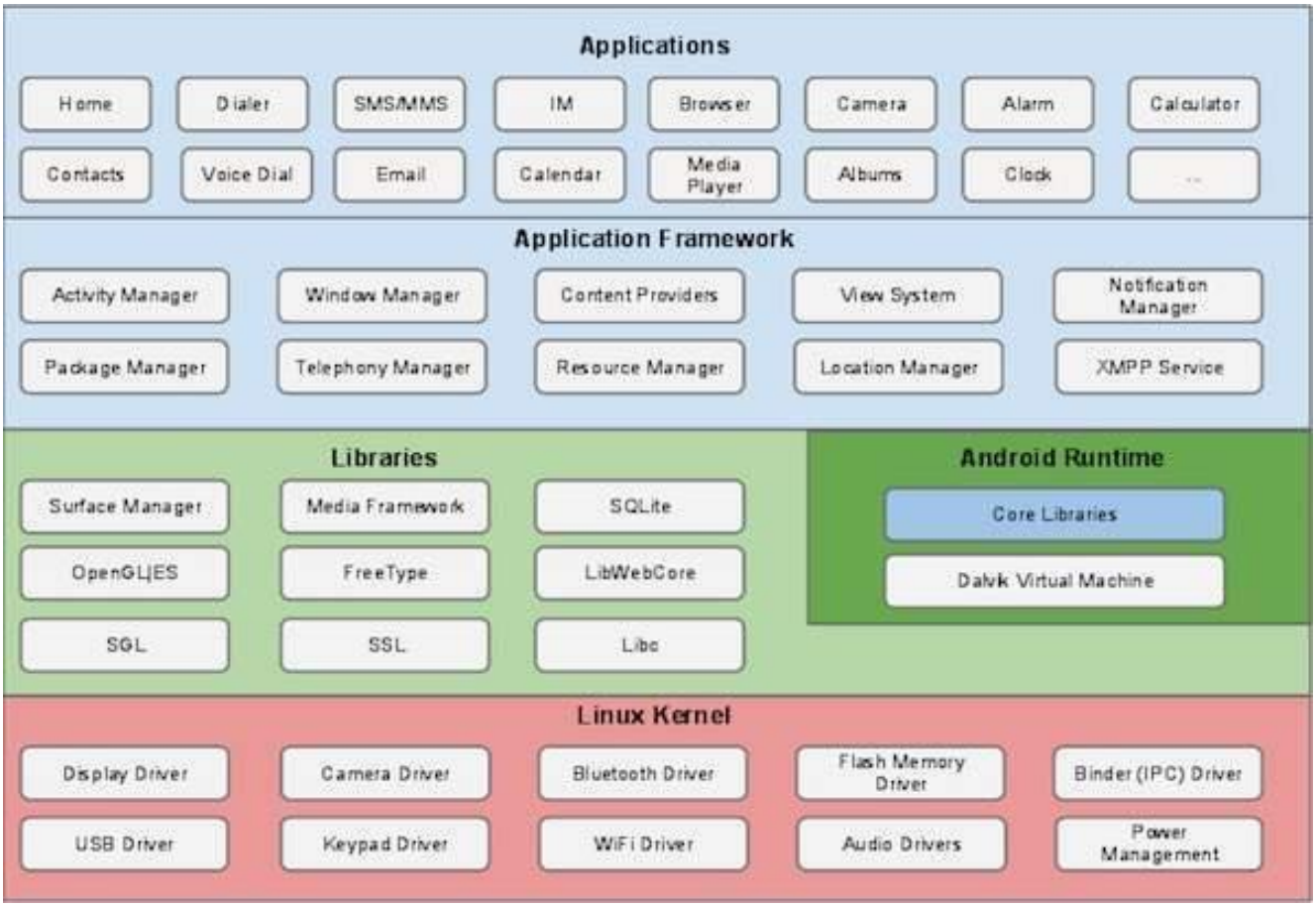


Рис. 1.6. Архітектура ОС Android.

Як і Android, iOS є операційною системою для мобільних пристроїв. iOS була розроблена Apple і вперше випущена у червні 2007 року. Остання версія програмного забезпечення – iOS 15.1. На відміну від Android, iOS використовується тільки для Apple і тільки для її мобільних телефонів (iPhone) і планшетів (iPad). Інші компанії не мають доступу до програмногозабезпечення і, отже, не мають

права використовувати його. Незважаючи наобмеження лише однією компанією, близько 26,75% операційних систем намобільних пристроях є версією iOS.

Нативні додатки для iOS можна розробляти мовою програмування Swift, яка була спеціально розроблена Apple, або Objective-C (рис. 1.7). Через різні мови програмування Android та iOS, написаний код не можна використовувати для обох нативних додатків, бо може призвести до витрат на нативну розробку. Apple також має нативне середовище розробки, SDK для додатків iOS називається «Xcode». Який, як і Android Studio, доступний безкоштовно в Інтернеті. Однак, на відміну від Android Studio, Xcode не можна встановити на кожному комп'ютері, це має бути комп'ютер Apple або комп'ютер під керуванням macOS. Тому комп'ютер Apple або емулятор macOS потрібні для розробки додатків для iOS.

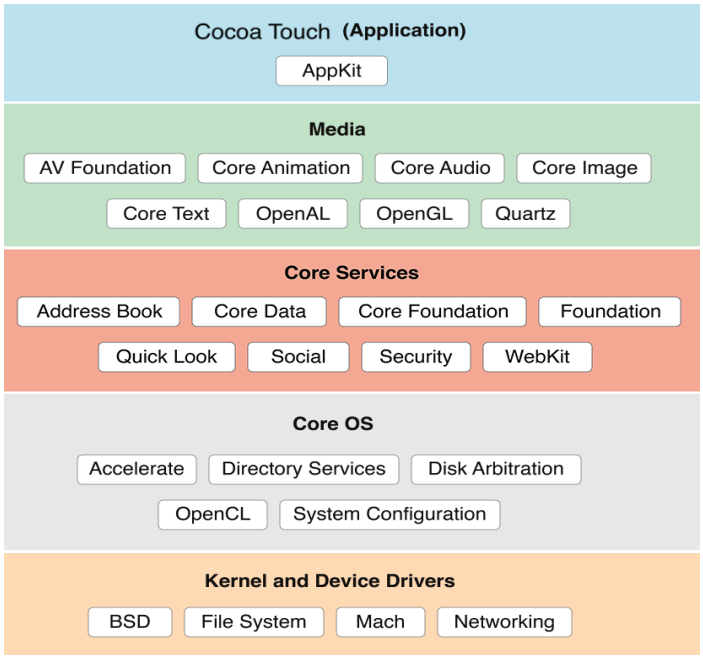


Рис. 1.7. Архітектура ОС iOS.

Мови програмування для розробки нативних додатків. Для розробки додатків потрібне середовище розробки. Робота з програмування зосереджена на операційній системі та додатку. Застосунок має вихідний код, який також можна редагувати ретроспективно. Також пропонуються інтерфейси API. Мова програмування залежить від системи яку використовують.

І Google для Android, і Apple для iOS надають кращий спосіб розробки

нативних застосунків. Відповідно до цього, нативний додаток для Android розробляється на Java, а Android Studio використовується як інтегроване середовище розробки (IDE). Крім того, також можна використовувати Kotlin як мову програмування. Інтерфейс користувача (UI) створюється декларативно за допомогою мови опису XML. Графічний дизайнер IDE надає підтримку. За такого підходу можна максимально адаптувати додаток добажаного обладнання та версії Android.

Комп'ютер Apple – обов'язкова вимога для розробки додатку для iOS. Xcode використовується як середовище розробки. При програмуванні Swift є більш простим у використанні наступником Objective-C. Дизайн інтерфейсу користувача виконується за допомогою розкадровок в графічному дизайнері Xcode.

Переваги нативних додатків. Розробка нативних застосунків дає безліч переваг таких як:

- висока швидкість;
- автономна функціональність;
- зрозумілий та інтерактивний інтерфейс;
- мінімальний обсяг помилок;
- максимальна продуктивність.

Оскільки нативні мобільні додатки не містять складного коду, такого як гібридні та кросплатформні додатки, вони виявляються порівняно швидше.

Більшість елементів застосунків відображаються швидко, оскільки вони попередньо завантажуються заздалегідь. Через високу швидкість розробки та рентабельність стартапи віддають перевагу нативним додаткам.

Автономна функціональність нативних додатків полягає у тому, що вони працюють бездоганно навіть за відсутності підключення до Інтернету. Цезабезпечує більшу зручність для користувачів, оскільки вони отримують доступ до всіх функцій застосунку в режимі польоту або в автономному середовищі. Ця функція автономної підтримки у нативних додатках важлива для користувачів застосунків, які мешкають у районах з низьким рівнем підключення до Інтернету, що знаходяться у віддалених районах або мають обмежену доступність даних.

Чудовий інтерфейс – це те, що підприємці-початківці можуть очікувати від нативних додатків. Оскільки нативні застосунки створені для конкретної ОС, вони дотримуються рекомендацій, які забезпечують покращений досвід, який повністю відповідає конкретній операційній системі в усіх відношеннях.

Більше того, оскільки нативні додатки дотримуються рекомендацій, це дозволяє користувачам взаємодіяти з додатками за допомогою жестів та дій, з якими вони вже знайомі. Оскільки нативні застосунки мають одну базу коду і не покладаються на кросплатформні інструменти, у них виникає мінімальна кількість помилок. Максимальна продуктивність нативних додатків досягається завдяки тому, що розробка оптимізована для конкретної платформи. Оскільки нативні додатки створюються для конкретної платформи, ці додатки виявляються чуйними та порівняно швидкими. Більше того, ці застосунки скомпільовані з використанням основних мов програмування та API, що робить нативні додатки ефективнішими.

Недоліки нативних додатків. Головним недоліком нативних додатків є неможливість повторного використання коду. Якщо нативні додатки повинні бути створені для iOS та Android окремо, потрібно виконувати частину кодування та розробляти додатки для обох мобільних операційних систем. І це вимагатиме багато часу, зусиль, витрат і ресурсів, на відміну від гібридних мобільних додатків із загальним серверним кодом або крос-платформного додатку з повторно використовуваною базою коду.

### 1.2.2. Крос-платформні технології розробки мобільних додатків

Щоб зробити мобільний додаток доступним для широкого загалу, він повинен працювати як на пристроях Android, так і на iPhone. Традиційні підходи до розробки мобільних додатків від Google і Apple розрізняються за багатьма аспектами, такими як мова програмування, інструменти та API інтерфейси, тому необхідно розробити два окремі нативні додатки.

Тож має сенс розробляти додаток для різних платформ. Цей процес називається крос-платформовою розробкою і зводить до мінімуму зусилля,

оскільки розробляється тільки один застосунок, що працює як на Android, такі на iOS. Кросплатформний додаток розробляється проміжною мовою програмування, яка не є частиною операційної системи пристрою. Пізніше він компілюється у відповідну операційну систему за допомогою платформи додатків, такої як Xamarin, React Native або Flutter.

Частина, більшість або весь код можна використовувати на всіх цільових платформах, включаючи iOS та Android. Кожні шість місяців народжується новий кросплатформовий фреймворкдля додатків, проте найпопулярнішими залишаються React Native, Flutter та Xamarin.

Переваги кросплатформних додатків. Кросплатформні додатки повинні бути обов'язковими для багатьохкомпаній, оскільки вони пропонують безліч переваг, таких як:

- можуть допомогти у найкоротші терміни зробити компанію відомоюцільовій групі;
- зниження витрат на разробку;
- простота обслуговування та швидкість розгортання;
- швидкий процес розробки;
- багаторазовий код;
- швидкий вивід на ринок та адаптування.

Кросплатформні додатки допомагають у найкоротші терміни зробитикомпанію відомою цільовій групі, завдяки тому, що їх можна розгорнути нарізних платформах, включаючи Інтернет. Це дозволяє зв'язуватися з користувачами платформ iOS та Android, тим самим збільшуючи охоплення.

Найкраще у кросплатформній розробці додатків – це скорочення витрат, оскільки не потрібно писати код більше одного разу. Так як коди, що багаторазово використовуються, і гнучка розробка додатків з використанням інструментів знижують витрати на розробку. Це гарний варіант для тих, хто не готовий витрачати гроші, але хоче просувати свій бренд на обох платформах.

Маючи для запуска всього один фрагмент коду, легко розробити додаток і запустити його на всіх платформах. Це також спрощує обслуговування та внесення

змін. Оновлення можна легко синхронізувати на всіх платформах та пристроях, заощаджуючи час та гроші.

Ще одна перевага, яку слід додати до списка, – швидкий процес розробки робить процес розгортання додатку швидким. Оскільки існує єдинийкод для кількох платформ, він допоможе скоротити витрати на розробку на 50–80%. Команда кросплатформних розробників може легко здати свою роботу в очікуваний термін. Завдяки кросплатформній розробці розробникам не потрібно щоразу писати новий код, тому що той самий код можна використовувати знову і знову. Це заощаджує багато часу та ресурсів, оскільки виключається повторення.

Як згадувалося вище, потрібно написати код тільки один раз, і можна запускати його будь-де і на різних платформах. Ця концепція робить процес дуже швидким, оскільки скорочує час виведення на ринок за рахунок швидкого розгортання. Через це також легко перетворити та налаштувати додаток. Таким чином, можна легко внести зміни до додатку.

Недоліки крос-платформних додатків. У ті часи, коли розроблялися кросплатформні фреймворки, вони обмежувалися створенням простих мобільних додатків та ігор. Однак згодом вони стали однією з найкращих технологій розробки мобільних додатків, оскільки стали більш адаптованими, потужними та гнучкими, ніж раніше. Тим не менш, вони стикаються з низкою проблем, зокрема:

- кросплатформні розробники знаходять підтримку кроссумісності додатків за допомогою обмежених інструментів;

- зниження продуктивності через неузгоджену взаємодію міжнативними та не нативними компонентами гаджетів;

- додаток повинен бути розроблений складним чином, в іншому випадку може виникнути поганий користувальницький досвід, оскільки збої, пов'язані з продуктивністю, дуже поширені;

- якщо бізнес-додаток керує даними компанії, це не рекомендується для крос-платформних додатків, оскільки це не дуже хороша ідея з міркувань безпеки.

### 1.2.3. Гібридні технології розробки мобільних додатків

Гібридна розробка додатків – це комбінація нативних та веб-рішень, де розробникам необхідно вбудовувати код, написаний за допомогою CSS, HTML та JavaScript, у нативний додаток за допомогою плагінів, включаючи Ionic's Capacitor, Apache Cordova тощо, які дозволяють отримати доступ до вбудованих функцій.

HTML5 відповідає за структуру та вміст гібридного додатку. У CSS3 створюються різні таблиці стилів, які визначають, як певні компоненти мають відображатися у додатку. CSS3 відповідає за створення та дизайн гібридного додатку. JavaScript використовується для розробки різних інтерфейсів (API) із апаратним забезпеченням системи. Це забезпечує доступ до GPS, камери, мікрофона та різних датчиків на смартфоні.

Розробка мовами Інтернету є великою перевагою для багатьох агентств з розробки додатків, оскільки веб-розробники вже мають великий досвід програмування з використанням HTML5, CSS3 та JavaScript.

Крім цих мов програмування, часто використовуються фреймворки, частини коду яких можна писати у вихідному коді. Це дозволяє інтегрувати в гібридний додаток специфічні для операційної системи функції, такі як: подвійне торкання або інші жести пальцями. Дані платформи не надаються Google або Apple, але були створені третіми сторонами.

Найпопулярнішими інструментами для розробки гібридних додатків є Ionic, Onsen і Sencha Touch. Для користувачів гібридний додаток виглядає як нативний застосунок. Він навіть завантажується як нативний застосунок, через магазин додатків, і встановлюється на пристрої. Після встановлення додаток повністю доступний офлайн. Коли гібридний додаток відкривається, запускається інтерфейс схожий на браузер, який працює аналогічно до веб-додатку.

Однак, на відміну від веб-застосунків, гібридні додатки використовують не веб-браузери, а швидше веб-представлення. Ці WebView використовуються для відображення веб-вмісту додатку. Використовується нативний механізм рендерингу браузера пристрою (але не браузер), який надає користувачеві зручний інтерфейс.

Переваги гібридних додатків. Розробка гібридного мобільного додатка дає безліч переваг, таких як:

- незалежна від платформи розробка;

- швидкий вихід на ринок; - простота обслуговування;

- мінімальні витрати при простоті розробки;

- окращений UI/UX.

Гібридний додаток потрібно розробляти лише один раз, і не потрібно розробляти індивідуально для кожної операційної системи (Android, iOS, Windows), як нативні додатки. Це означає, що гібридні застосунки можна встановлювати та використовувати у всіх операційних системах, а це означає, що можна охопити велику кількість потенційних користувачів.

Стартапи зазвичай прагнуть забезпечити більш швидке постачання MVP ринку. У цьому випадку платформа розробки гібридних додатків є ідеальним вибором, оскільки вона забезпечує швидку розробку та запуск додатку на ринку на порівняно ранній стадії. Оскільки гібридні застосунки засновані виключно на веб-технологіях, підтримувати гібридних додатків простіше, ніж нативні та крос-платформні застосунки зі складним кодуванням.

Оскільки життєвий цикл розробки гібридних додатків супроводжує уніфіковану розробку, це приносить полегшення стартапам з обмеженим бюджетом. Їм не потрібно окремо вкладати кошти для створення різних версій додатків для кількох платформ. Платформи розробки гібридних застосунків дозволяють розробникам розробляти одну версію і використовувати її для декількох платформ (наприклад Android і iOS).

Гібридні додатки об'єднують переваги нативних і веб-додатків, пропонуючи бездоганний і покращений інтерфейс користувача на платформах Android і iOS. Розробка гібридних додатків має легкий інтерфейс користувача гібридного додатку, який забезпечує над швидке завантаження графіки та контенту.

Недоліки. Існують певного виду недоліки, про які варто подумати при виборі гібридний додатків:

- немає офлайн підтримки;

- невідповідності ОС.

Гібридні застосунки не пропонують офлайн-підтримку, на відміну від нативних додатків. Користувачам необхідно дочекатися підключення до Інтернету, щоб отримати доступ до функцій додатку.

Оскільки в гібридних додатках розгорнутий єдиний код, існують функції, специфічні для конкретної операційної системи, які не працюють належним чином в інших системах, наприклад, деякі функції, специфічні для платформи Android, можуть не працювати на пристроях iOS.

Гібридний додаток виглядає як нативний додаток, але не має індивідуальних адаптацій для різних операційних систем. Це означає, що типовий зовнішній вигляд відповідної мобільної платформи, чи то Android, iOS або Windows, не може бути відтворений. Складні додатки не працюватимуть ідеально з цим рішенням, більша кількість функцій уповільнить його роботу.

### 1.2.4. Розробка за допомогою технології прогресивних веб-додатків

Прогресивні веб-додатки (PWA) – це золота середина між нативними додатками та мобільними веб-сайтами. Їхня розробка порівняно недорога, оскільки вони засновані на крос-платформних веб-стандартах, і так заощаджують витрати на розробку та обслуговування незалежних нативних додатків для iOS та Android.

PWA – це веб-сайти, створені за допомогою мов HTML/CSS та JavaScript, як і раніше, а не нативні додатки, які можна завантажити з App Store або Google Play. Однак їх також можна перетворити на нативні застосунки. Подібно до адаптивних веб-сайтів, прогресивні веб-додатки відображають весь контент, ідеально адаптований для відповідного кінцевого пристрою.

PWA мають перевагу перед адаптивними сайтами насамперед у тому, що їх простіше та гнучкіше використовувати для кінцевого споживача. З погляду користувача, вони самі по собі не виконують ніяких дратівливих або шпигунських фонових дій на смартфонах, планшетах чи ноутбуках.

PWA – це, по суті, звичайні веб-сайти, до яких додані щонайменше три

додаткові стандарти:

а) За промовчанням HTTPS: Підключення до Інтернету завжди повинно бути здійснено через захищене з'єднання HTTPS;

б) Маніфест веб-додатку: маніфест веб-додатку надає основну інформацію про веб-додаток, таку як назва веб-сайту, його автор, значок та короткий опис. За допомогою маніфесту веб-додатку в кінцевому підсумку гарантується, що зовнішній вигляд веб-застосунку більше не відрізнятиметься від зовнішнього вигляду нативних додатків;

в) Сервісний працівник: у кожному PWA є сервісний працівник, який зберігає дані на пристрої, наприклад, статті, які ви вже прочитали. Таким чином, можна порівняти базову функціональність додатку, навіть якщо пристрій взагалі не підключено до Інтернету.

Крім того, можуть бути додані такі типові функції:

- push-сповіщення: додаток не обов'язково повинен бути відкритим для отримання повідомлень;

- автономна підтримка (через вищезгаданий сервіс-воркер): деякі функції PWA також можна використовувати без підключення до Інтернету;

- негайне заряджання екранами-скелетами: екрани-скелети, тобто скелетні екрани, прискорюють завантаження і піклуються про враження. Вони працюють таким чином, що скелет, основна структура сторінки, відображається якнайшвидше при запуску додатку. Лише поступово рамки наповнюються змістом;

- значок ярлика у вигляді додатку, який можна додати на головний екран;

- відео телефонія та чати;

- поділитися API: контентом можна поділитися через Twitter, Whatsapp тощо через Share API;

- Bluetooth API: він дозволяє використовувати низку додатків для таких пристроїв, як Apple Watch та пристрої IOT, які підключені до мобільного телефону через Bluetooth.

Не всі функції нативних додатків можна відобразити у прогресивному веб-застосунку. Між іншим:

- читання SMS (наприклад, для пін-коду, відправленого по SMS) та списку тих, хто дзвонив;

- використання Bluetooth та GPS у фоновому режимі, тобто поки PWA не використовується активно.

Однак жоден із цих пунктів не повинен регулярно виступати як смертоносний аргумент на користь використання PWA.

Все більше і більше функцій, які раніше були зарезервовані для нативних додатків, тепер також можуть бути відображені через JavaScript, так що веб-додатки та нативні додатки все більше і більше зближуються. Так звані прогресивні веб-застосунки (PWA) майже ні в чому не поступаються класичним додаткам.

Мова розмітки HTML5 зробила якісний стрибок для Інтернету. Крім покращення семантики, однією з основних цілей була підтримка мобільного Інтернету. Наприклад, були введені спеціальні типи введення для форм, які відображають клавіатури, що збігаються на смартфонах і планшетах – для адреса електронної пошти та числових вводів. А параметри анімації CSS3 дозволяють переміщати веб-сайт з економією ресурсів, не покладаючись на JavaScript.

Крім зручності використання та дизайну, перш за все, багато мобільних функцій роблять додаток цікавим. В останні роки тут було представлено численні API-інтерфейси JavaScript, спеціально призначені для мобільних пристроїв. Це включає API геолокації, який має доступ до інформації про місцезнаходження пристрою, і API орієнтації пристрою, який використовується для доступу до орієнтації пристрою. API-інтерфейси, що забезпечують потокову передачу та захоплення аудіо та відео, в даний час все ще перебувають у розробці.

API вібрації та стану батареї навіть гарантує, що можна налаштувати вібросигнали та запросити поточний статус батареї.

Якщо ви побачите поточний статус різних API-інтерфейсів JavaScript, можна виявити, що велика кількість API-інтерфейсів спеціально розроблена для мобільних пристроїв. За допомогою Notification-API вже давно можна створювати сповіщення. Однак для цього завжди необхідно, щоб веб-сайт, з якого надходить повідомлення, був відкритий у браузері.

З новим Push API та Service Worker API більше немає потреби відкривати відповідний веб-сайт. JavaScript реєструється у браузері через API сервіс-воркера, який виконується незалежно від веб-сайту. Все це можна порівняти з фоновими службами нативних мобільних додатків.

За допомогою сервісних працівників можна також завантажувати всі файли, які потрібні для автономного використання. Залежно від того, як використовується додаток, в ідеалі також можна використовувати без підключення до Інтернету.

Для PWA є інструмент аналітики Lighthouse. Це інструмент із відкритим вихідним кодом для автоматичного аудиту веб-застосунків та PWA, створений Google і доступний через Chrome DevTools. Інструмент точно оцінює продуктивність, доступність, прогресивні покращення та рівні пошукової оптимізації, що робить його незамінним як для розробників додатків, так і для фахівців служби підтримки. Lighthouse надає оцінку (число від 0 до 100) та список невдалих/пройдених аудитів для кожного параметра. Інструмент також дає змогу запускати додаткові перевірки вручну.

Існує кілька фреймворків та інструментів JavaScript для створення прогресивних веб-застосунків, і кожен з них пропонує різні можливості.

Популярність, документація, підтримка, продуктивність та специфіка процесу розробки – все це відіграє важливу роль у прийнятті рішення про те, які фреймворки та інструменти найкраще підходять для розробки PWA.

Використання добре відповідного фреймворку має прискорити рендеринг та розробку, тоді як роздутий розмір деяких фреймворків та бібліотек може негативно позначитися на темпах розвитку проєкту.

Переваги PWA. Основні переваги PWA, які роблять їх такими продуктивними це:

- підвищення залученості та конверсії;
- вклад медіа-гігантів;
- підтримка основних платформ;
- технічні переваги;
- короткий час виходу на ринок.

У порівнянні зі звичайними веб-сайтами, PWA демонструють більш високий рівень залученості та конверсії завдяки своєму зовнішньому вигляду, поведінці та інтерфейсу користувача, схожому на додатки.

PWA використовуються такими медіа-гігантами як Twitter, Microsoft Office, Uber, Forbes, Alibaba, New York Times, Pinterest та іншими провіднимикомпаніями як єдині або додаткові застосунки для різних платформ.

Google і Microsoft впровадили PWA для Android та Windows і активно їх просувають. Google надає різні засоби та інструменти розробки, бібліотеки, фреймворки і список покращень, сумісних з Chrome, а також повністю просуває PWA в магазині Google Play. З боку Microsoft підтримка проявляється у наявності PWA в Microsoft Store, а також автоматичному пошуку, оцінці та додаванні таких високоякісних прогресивних додатків на ринок.

У порівнянні з нативними мобільними та настільними додатками, PWA важать небагато та можуть бути швидко встановлені, завантажені (особливо після початкового завантаження) та оновлені через Google Play, App Store та Microsoft Store або безпосередньо з браузерів, повністю минаючи торгові майданчики.

Крім цього, PWA є адаптивними та чуйними, а завдяки вимогі протоколу HTTPS вони безпечні з самого початку. На розробку PWA потрібно менше часу, особливо якщо розробники використовують код і контент існуючих веб-додатків і спільно використовують кодову базу на різних платформах. PWA також можна створювати за допомогою «простого» JavaScript або певних інструментів чи засобів, які можуть ще більше прискорити процес розробки.

Недоліки PWA. PWA має небагато недоліків, але вони досить суттєві:

- розряджає акумулятор;
- проблеми зі застарілими пристроями;
- PWA мають обмеження;
- сумісність із iOS.

З огляду на те, що PWA написані у вигляді складних кодів, телефонам доводиться старанніше працювати, щоб інтерпретувати код. Ось чому вони споживають більше енергії, ніж нативні додатки.

PWA існують лише кілька років, тому не дивно, що старі мобільні пристрої із застарілими веб-браузерами (чи застарілими версіями сучасних браузерів) не надто добре їх підтримують. Хоча ця проблема неминуче вирішиться сама собою в майбутньому, деякі компанії можуть поскаржитися на неї.

Якими б потужними не були PWA порівняно з традиційними веб- сайтами, вони не можуть робити все, що можуть робити мобільні додатки.

Оскільки вони написані на JavaScript, вони не такі економічні, як застосунки, написані рідними мовами, такими як Kotlin або Swift.

Їхня продуктивність також не така гарна, як продуктивність нативних додатків, що багато в чому пов'язано з тим фактом, що JavaScript є однопотоковою мовою програмування. На даний момент доступ до деяких важливих функцій пристрою, як і раніше, відсутній, включаючи Bluetooth, датчики наближення, навколишнє освітлення, розширені елементи керування камерою та інші.

Основним аргументом проти прогресивних веб-додатків є те, що Apple не зацікавлена в дозволі PWA на iPhone, незважаючи на повну підтримку macOS, і тот факт, що PWA визнані життєздатними варіантами розробки додатків для iPhone з моменту випуску iOS11 у 2017 році, вони, як і раніше, мають певні обмеження.

У iOS кожен веб-додаток заснований на Safari, але сам Safari не може підтримувати все, що можуть запропонувати PWA. Наприклад, досі немає підтримки повідомлень для PWA у фоновому режимі. Крім того, Safari дозволяє зберігати у пам'яті пристрою лише до 50 МБ. Таким чином, PWA не підходить для iOS-додатків для потокової передачі мультимедіа або будь- якого іншого додатку, що значною мірою залежить від автономного сховища.

У посібнику App Store рекомендується доставляти динамічні додатки Safari, а не публікувати їх в App Store. Оскільки ринок регулярно піддається масовим «чисткам», видаляючи несумісні застосунки, розміщення вашого PWA може бути ризикованим. iOS не дозволяє встановлювати застосунки, яких немає в офіційному магазині додатків. Ось чому переважна більшість користувачів iPhone навіть не усвідомлює, що є можливість встановлювати PWA з браузера, а ті, хто це робить, не захочуть докладати зусиль для їх встановлення.

### 1.3. Висновки до першого розділу

У першому розділі, виходячи з проведеного аналізу, можна стверджувати, що нативні додатки, які дають найкращу продуктивність, добре підходять для комплексних проектів, що працюють зі складною графікою (напр., відеоігри). Неправильний вибір типу мобільного додатку, і відповідно до цього - засобу розробки, може позначитися на кінцевому продукті. У кращому випадку можна отримати невдоволеність користувачів, а у гіршому – взагалі не буде змоги впровадити такий застосунок. Згідно цих міркувань, існує таблиця порівняння типів мобільних додатків (рис. 1.8).

| | Native apps | Cross-platform apps | Progressive web apps | Hybrid apps |
|---|---|---|---|---|
| Number of codebases | One per platform | One, but compiled for each platform | One total | One for the app, another for the container |
| Languages and frameworks | Native only | Team's choice | Web only | Web and native |
| Access to SDKs and APIs | Yes | Yes | No | Limited |
| Performance | Highest | High | Lowest | Low |
| Access to device hardware | Complete | Most | Very little | Some |
| Responsiveness to user input | Good | Good | Worst | Poor |
| Interactivity | High | High | Lowest | Low |
| Device resource use | High | High | Low | Medium |
| Requires connectivity | No | No | Yes | Yes |
| Cost to build and maintain | Highest | High | Lowest | Lower |
| Where the app is stored | Device | Device | Server | Device and server |
| Deployed through | Marketplace | Marketplace | Browser | Marketplace |
| Requires outside approval | Yes | Yes | No | Yes |

Рис. 1.8. Порівняння типів мобільних додатків

Що стосується менш вимогливих додатків, при їх розробці на кілька платформ (наприклад, iOS та Android), доцільно звернути увагу на такі засоби як Flluter чи React Native. Їх кінцевий продукт, що матиме нативний користувацький

інтерфейс, буде поводитись швидко та плавно при правильному застосуванні технологій. А користувач не помітить жодної втрати продуктивності без спеціальних тестів. Крім того, це суттєво скорочує термін розробки, а відповідно і вартість.

# РОЗДІЛ 2
# АНАЛІЗ ТА РОЗБІР РОБОТИ ФРЕЙМВОРКІВ FLUTTER ТА REACT NATIVE

## 2.1. Аналіз методів роботи фреймворку Flutter

Flutter – популярний та безкоштовний мобільний open-source фреймворк для побудови інтерфейсу користувача, і створення кросплатформних програм від компанії Google. Вийшов у травні 2017 року. Однією з його переваг є можливість написання єдиної кодової бази для платформ IOS і Android.

Також наразі вже з'явилася підтримка для веб браузерів – Flutter Web. Завдяки цьому з'явилася можливость використання одної мови програмування для розробки декількох мобільних додатків.

Основи архітектури Flutter. Flutter складається з двох основних частин: набору програмного забезпечення (SDK) та фреймворку з бібліотекою інтерфейсу користувача на основі віджетів.

- Фреймворк — це бібліотека інтерфейсу користувача на основі віджетів, яка містить різноманітні багаторазово використовувані елементи інтерфейсу користувача, такі як повзунки, кнопки, введення тексту та інші. Пізніше ці елементи можна персоналізувати відповідно до ваших потреб.
- SDK — це набір інструментів для розробки додатків і компіляції вашого коду у рідний машинний код для Android та iOS.

Flutter – це доволі молода, але багатообіцяюча платформа, яка вже привернула увагу достатньо великих компаній, які запустили свої додатки. Цікава дана платформа своєю простотою, порівняно з розробкою веб-застосунку і швидкість роботи на рівні нативних додатків. Висока продуктивність застосунків і швидкість розробки досягається за рахунок декількох технік:

1. На відмінну від багатьох відомих сьогодні мобільних платформ, Flutter не використовує Javascript взагалі. В якості мови програмування для Flutter вибрали Dart, який компілюються в бінарний код, завдяки якому досягається швидкість виконання операцій, яку можна порівняти з нативними технології.

2. Flutter не використовує нативні компоненти, тому нічого не потрібно розробляти для комунікації з ними. Замість цього інтерфейс будується як в ігрових

програмах. Фон, текст, медіа-елементи, кнопки – весь рендер відбувається в Flutter. Навіть враховуючи це, «Hello World» застосунок займає зовсім не багато місця.

3. Для побудови інтерфейсу користувача використовується декларативний підхід, що схоже на шлях ReactJs, на основі віджетів. Для ще більшого приросту до швидкості роботи інтерфейсу віджети малюються за необхідністю – тільки якщо в моделі відбулися зміни (подібно до того як відбувається у світу FrontEnd).

4. Також варто зазначити, що в фреймворк вбудований так званий Hot-reload, якого до сих пір не було в нативних платформах.

Мова програмування Dart. Dart – це достатньо нова мова програмування, яка швидко набирає популярність, особливо з виходом фреймворку Flutter. Це гнучка мова, яка підходить як і для простих програм, так і для повнофункціональних застосунків. Вона використовує об’єктно-орієнтований підхід і C-подібний синтаксис для більшої простоти і доступності.

Характеристика мови програмування Dart:

- Передбачання використання класів. На відмінну від мови програмування Javascript, яка дає можливість вибору щодо використання класів, Dart підпорядковує на використання ООП. Також тут присутня підтримка інтерфейсів, абстрактних класів, generics, mixins та статична типізація.
- В той час як більшість мов програмування мають статичну або динамічну типізацію, в Dart є можливість декларувати типи змінних так використовувати змінні без явного вказання типу.
- Підтримка компіляції коду на Dart в Javascript для виконання та розгортання в усіх сучасних браузерах.
- Підтримка Isolates для потоків. Вони не розділяють пам'ять, а спілкування здійснюється за допомогою повідомлень.

Dart має коротку криву навчання, для того щоб набути всіх необхідних знань, непотрібно витрачати дуже багато часу вивчаючи всі нюанси мови. Таку простоту мови забезпечує підтримка як і строгої, так і не строгої типізації. Це робить Dart простішим для тих, хто переходить з інших мов програмування. Синтаксис простий і його можна зрозуміти без особливих складностей. Хоча мова добре структурована

так само, як і C, їй вдається перемогти останню з точки зору простоти.

Dart підтримує як і AOT, так і JIT компіляцію. Хоча дана функція доступна не на всіх фреймворках. Вони зазвичай використовуються в різних ситуаціях, наприклад, при компіляції нативних застосунків використовується AOT. JIT може бути дуже корисним під час етапу розробки, тому що вона дозволяє майже миттєво побачити зміни в коді застосунку [11].

**Архітектурні шари**. Flutter розроблено як розширювана багатошарова система. Він існує як ряд незалежних бібліотек, кожна з яких залежить від базового рівня. Жоден рівень не має привілейованого доступу до нижнього рівня, і кожна частина рівня структури створена як необов'язкова та замінна (рис. 2.1).

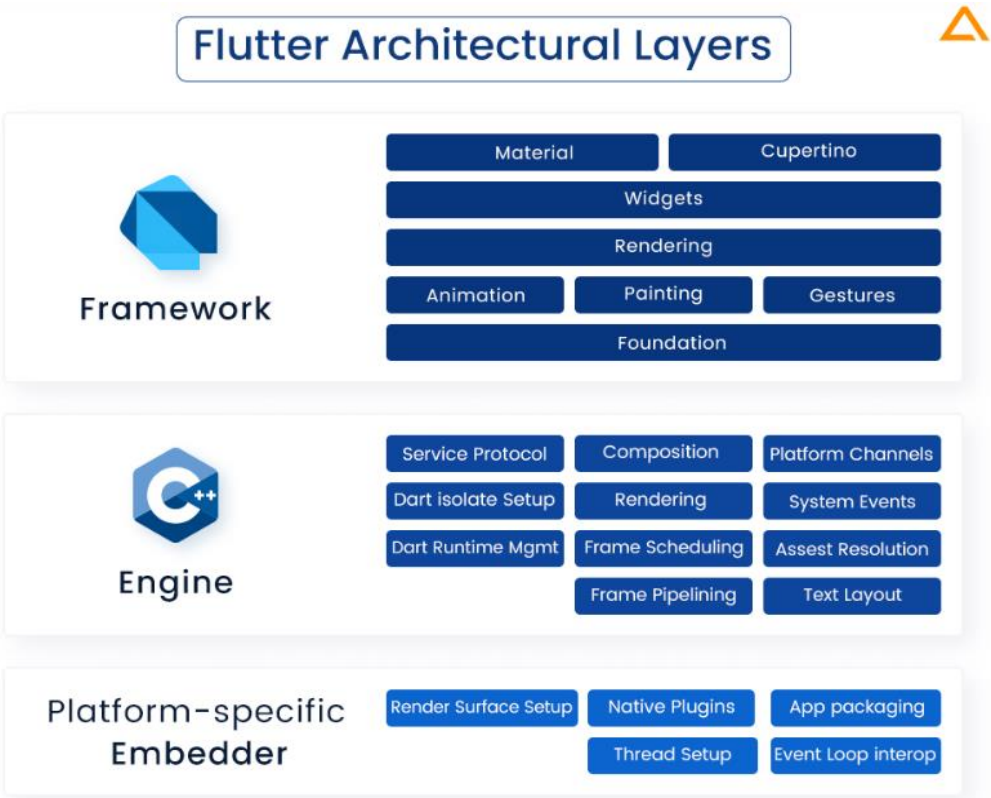


Рис. 2.1. Архітектура Flutter.

Для основної операційної системи програми Flutter упаковані так само, як і будь-яка інша нативна програма. Спеціальна платформа для вбудовування забезпечує точку входу; координує роботу з базовою операційною системою для доступу до таких служб, як поверхні відтворення, доступність і введення; і керує

циклом подій повідомлення. Засіб для вбудовування написано мовою, яка відповідає платформі: зараз Java і C++ для Android, Objective-C/Objective-C++ для iOS і macOS і C++ для Windows і Linux. Використовуючи embedder, код Flutter можна інтегрувати в існуючу програму як модуль, або код може являти собою весь вміст програми. Flutter включає кілька вбудованих пристроїв для звичайних цільових платформ, але існують і інші вбудовані пристрої.

В основі Flutter лежить двигун Flutter, який здебільшого написаний мовою C++ і підтримує примітиви, необхідні для підтримки всіх програм Flutter.

Механізм відповідає за растеризацію складених сцен щоразу, коли потрібно намалювати новий кадр. Це забезпечує низькорівневу реалізацію основного API Flutter, включаючи графіку (через Skia), макет тексту, файловий і мережевий введення/виведення, підтримку доступності, архітектуру плагінів, а також середовище виконання Dart і ланцюжок інструментів компіляції.

Двигун піддається впливу фреймворку Flutter через dart:ui, який загортає базовий код C++ у класи Dart. Ця бібліотека розкриває примітиви найнижчого рівня, такі як класи для управління введенням, графікою та підсистемами відтворення тексту. Як правило, розробники взаємодіють із Flutter через Flutter framework, який надає сучасну реактивну структуру, написану мовою Dart. Він містить багатий набір платформ, макетів і базових бібліотек, які складаються з ряду рівнів. Працюючи знизу вгору, отримуємо такий функціонал:

- Основні базові класи та служби будівельних блоків, такі як: анімація, малювання та жести, які пропонують широко використовувані абстракції над основною основою**.**
- Рівень візуалізації забезпечує абстракцію для роботи з макетом. За допомогою цього шару можливо побудувати дерево об’єктів, які можна відобразити. Також можливо керувати цими об’єктами динамічно, при цьому дерево автоматично оновлює макет, щоб відобразити зміни.
- Шар віджетів — це абстракція композиції. Кожен об’єкт візуалізації в шарі рендерингу має відповідний клас у шарі віджетів. Крім того, рівень віджетів

дозволяє визначати комбінації класів, які можна повторно використовувати. Це рівень, на якому вводиться модель реактивного програмування.

- Бібліотеки Material і Cupertino пропонують комплексні набори елементів керування, які використовують примітиви композиції рівня віджетів для реалізації мов дизайну Material або iOS.

Фреймворк Flutter відносно невеликий, є багато функцій вищого рівня, які можуть використовувати розробники, реалізовані у вигляді пакетів, включаючи плагіни платформи, як, наприклад, camera та webview.

**Веб-підтримка Flutter**. Хоча загальні архітектурні концепції застосовуються до всіх платформ, які підтримує Flutter, існують деякі унікальні характеристики веб-підтримки Flutter, які заслуговують на коментар.

Dart компілює JavaScript протягом усього часу існування мови, маючи інструментарій, оптимізований як для розробки, так і для виробництва. Багато важливих програм компілюються з Dart у JS і працюють у робочому стані сьогодні, включно з інструментами для рекламодавців для Google Ads. Оскільки фреймворк Flutter написаний на Dart, компілювати його в JavaScript було відносно просто.

Однак механізм Flutter, написаний мовою C++, призначений для взаємодії з основною операційною системою, а не з веб-браузером. Тому потрібен інший підхід. В Інтернеті Flutter забезпечує повторну реалізацію двигуна поверх стандартних API браузера. Зараз у нас є два варіанти відтворення вмісту Flutter в Інтернеті: HTML і WebGL. У режимі HTML Flutter використовує HTML, CSS, Canvas і SVG. Для візуалізації в WebGL Flutter використовує версію Skia, скомпільовану в WebAssembly під назвою CanvasKit . Хоча режим HTML пропонує найкращі характеристики відносно розміру коду, але CanvasKit забезпечує найшвидший шлях до графічного стеку браузера та пропонує дещо вищу точність графіки з нативними мобільними цілями.

Веб-версія діаграми архітектурного рівня виглядає, як показано на рис. 2.2:

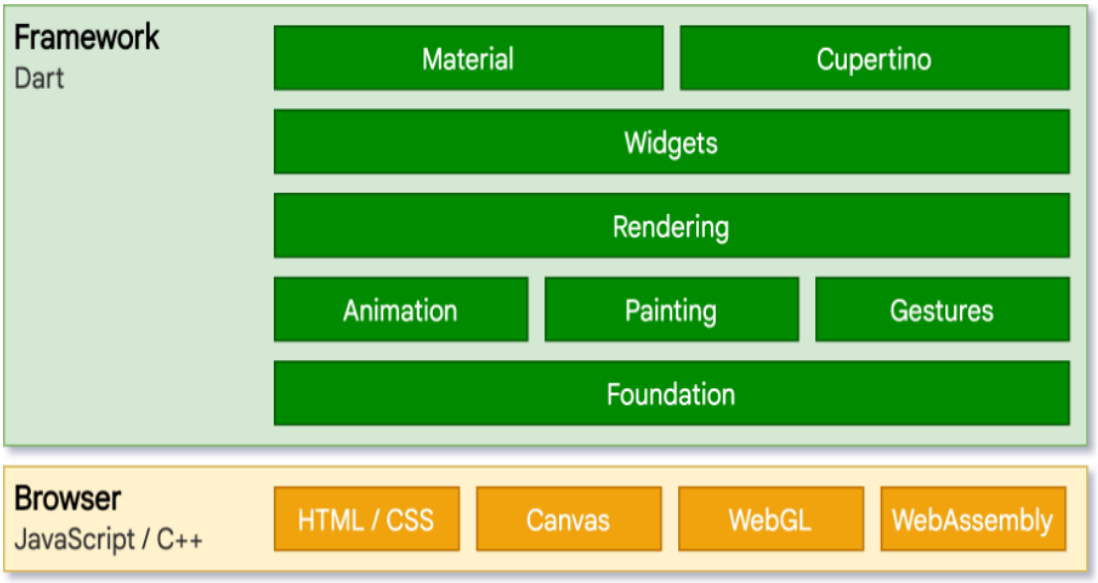


Рис. 2.2. Веб-версія діаграми архітектурного рівня

Можливо, найпомітнішою відмінністю від інших платформ, на яких працює Flutter, є те, що Flutter не потребує надавати середовище виконання Dart. Натомість фреймворк Flutter (разом із будь-яким написаним кодом) компілюється в JavaScript. Варто також зазначити, що Dart має дуже мало мовних семантичних відмінностей у всіх своїх режимах (JIT проти AOT, нативна проти веб-компіляції), і більшість розробників ніколи не напишуть рядок коду, який зіткнеться з такою різницею.

Під час розробки Flutter web використовує dartdevc, компілятор, який підтримує інкрементну компіляцію і тому дозволяє гаряче перезавантаження (хоча зараз не гаряче перезавантаження) для програм. І навпаки, коли ви готові створити робочу програму для Інтернету, dart2js використовується високооптимізований робочий компілятор JavaScript Dart, який упаковує ядро Flutter і фреймворк разом із вашою програмою в мінімізований вихідний файл, який можна розгорнути на будь-якому веб-сервері. Код можна запропонувати в одному файлі або розділити на кілька файлів за допомогою відкладеного імпорту.

**Реактивні інтерфейси користувача**. Зовні Flutter — це реактивна, псевдодекларативна структура інтерфейсу користувача, у якій розробник забезпечує відображення стану програми в стан інтерфейсу, а структура бере на себе завдання оновлення інтерфейсу під час виконання, коли стан програми

змінюється. Ця модель натхненна роботою Facebook над власним фреймворком React, який включає переосмислення багатьох традиційних принципів дизайну.

У більшості традиційних фреймворків інтерфейсу користувача початковий стан інтерфейсу користувача описується один раз, а потім окремо оновлюється кодом користувача під час виконання у відповідь на події. Одна з проблем цього підходу полягає в тому, що в міру ускладнення програми розробник повинен знати, як каскадно змінюється стан по всьому інтерфейсу користувача. Наприклад, розглянемо такий інтерфейс користувача, як показано на рис. 2.3.

Є багато місць, де можна змінити стан: поле кольору, повзунок відтінку, перемикачі. Коли користувач взаємодіє з інтерфейсом користувача, зміни повинні відображатися в усіх інших місцях. Гірше того, якщо не вжити обережності, незначна зміна однієї частини інтерфейсу користувача може спричинити хвилі ефектів для, здавалося б, не пов’язаних фрагментів коду.

Одним із рішень цього є такий підхід, як MVC, коли ви надсилаєте зміни даних у модель через контролер, а потім модель надсилає новий стан у представлення через контролер. Однак це також проблематично, оскільки створення та оновлення елементів інтерфейсу є двома окремими кроками, які можуть легко вийти з ладу.

Flutter разом з іншими реактивними фреймворками використовує альтернативний підхід до цієї проблеми, явно відокремлюючи інтерфейс користувача від його основного стану.

За допомогою API у стилі React створюється лише опис інтерфейсу користувача, а фреймворк піклується про використання цієї єдиної конфігурації для створення та/або оновлення інтерфейсу користувача відповідно.

У Flutter віджети (схожі на компоненти в React) представлені незмінними класами, які використовуються для налаштування дерева об’єктів. Ці віджети використовуються для керування окремим деревом об’єктів для макета, яке потім використовується для керування окремим деревом об’єктів для композиції. За своєю суттю Flutter — це низка механізмів для ефективного проходження

модифікованих частин дерев, перетворення дерев об’єктів у дерева об’єктів нижчого рівня та поширення змін між цими деревами.

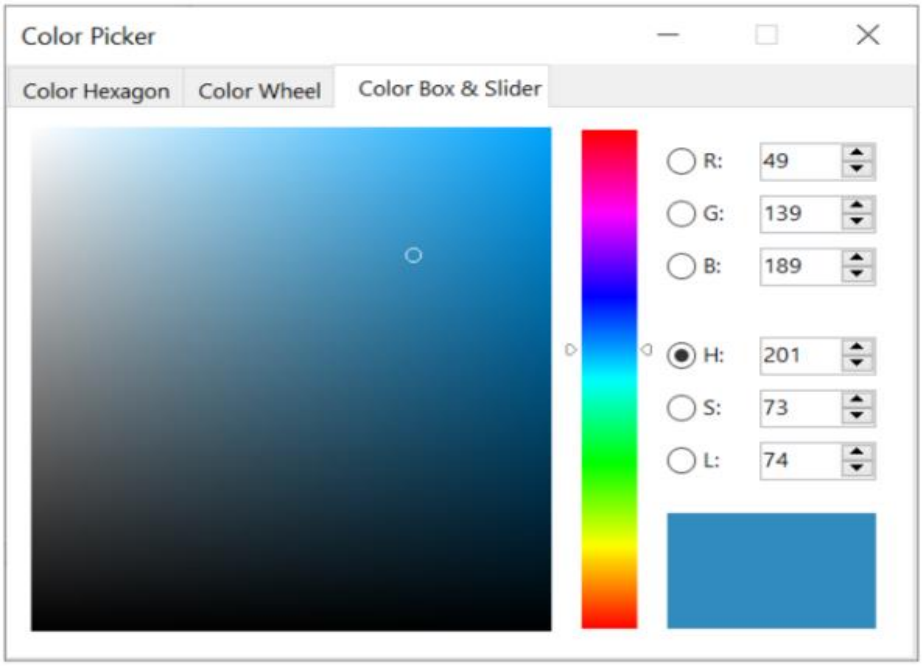


Рис. 2.3. Інтерфейс користувача Flutter

Віджет оголошує свій інтерфейс користувача шляхом заміни build() методу, який є функцією, яка перетворює стан на UI:

```
UI = f(state)
```

За своєю структурою метод  build() швидко виконується та не має побічних ефектів, дозволяючи фреймворку викликати його щоразу, коли це необхідно.

Цей підхід покладається на певні характеристики середовища виконання мови (зокрема, швидке створення екземпляра об’єкта та видалення). На щастя, Dart особливо добре підходить для цього завдання.

Віджети. Як згадувалося, Flutter наголошує на віджетах як на одиниці композиції. Віджети є будівельними блоками інтерфейсу користувача програми Flutter, і кожен віджет є незмінною декларацією частини інтерфейсу користувача.

Віджети утворюють ієрархію на основі композиції. Кожен віджет вкладається в батьківський елемент і може отримувати контекст від батьківського елемента. Ця структура ведеться аж до кореневого віджета (контейнера, у якому

розміщено програму Flutter, зазвичай MaterialAppабо CupertinoApp), як показує цей тривіальний приклад:

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
void main() => runApp(const MyApp());
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('My Home Page'),
        ),
        body: Center(
          child: Builder(
            builder: (context) {
              return Column(
                children: [
                  const Text('Hello World'),
                  const SizedBox(height: 20),
                  ElevatedButton(
                    onPressed: () {
                      print('Click!');
                    },
                    child: const Text('A button'),
                  ),
                ],
              );
            },
          ),
        ),
      ),
    );
```

```
  }
}
```

У попередньому коді всі екземпляри класів є віджетами.

Програми оновлюють свій інтерфейс користувача у відповідь на події (такі як взаємодія користувача), повідомляючи фреймворку замінити віджет в ієрархії іншим віджетом. Потім структура порівнює нові та старі віджети та ефективно оновлює інтерфейс користувача.

Flutter має власну реалізацію кожного елемента керування інтерфейсу користувача, а не відкладається на ті, що надаються системою: наприклад, існує чиста реалізація Dart як для елемента керування iOS Switch , так і для еквівалента Android. Цей підхід дає кілька переваг:

- Забезпечує необмежену можливість розширення. Розробник, якому потрібен варіант елемента керування Switch, може створити його будь-яким довільним способом, не обмежуючись точками розширення, наданими ОС.
- Уникає значного обмеження продуктивності, дозволяючи Flutter комбінувати всю сцену одночасно, без переходів між кодом Flutter і платформи.
- Відокремлює поведінку програми від будь-яких залежностей операційної системи. Програма виглядає та працює однаково на всіх версіях ОС, навіть якщо ОС змінила реалізацію своїх елементів керування.

**Композиція.** Віджети, як правило, складаються з багатьох інших невеликих одноцільових віджетів, які об’єднуються для створення потужних ефектів.

Там, де це можливо, кількість концепцій дизайну зведена до мінімуму, але загальний словниковий запас має бути великим. Наприклад, у шарі віджетів Flutter використовує ту саму основну концепцію (a Widget), щоб представити малюнок на екрані, макет (розташування та розмір), інтерактивність користувача, керування станом, оформлення тем, анімацію та навігацію.

У шарі анімації пара понять Animations і Tweens охоплює більшу частину простору дизайну. У шарі візуалізації RenderObjects використовуються для опису макета, малювання, перевірки попадання та доступності. У кожному з цих випадків

відповідний словниковий запас стає великим: існують сотні віджетів і об’єктів візуалізації, а також десятки типів анімації тощо.

Ієрархія класів навмисно поверхнева та широка, щоб максимізувати можливу кількість комбінацій, зосереджуючись на невеликих компонованих віджетах, кожен з яких добре виконує одну функцію. Основні функції є абстрактними, і навіть базові функції, такі як доповнення та вирівнювання, реалізовані як окремі компоненти, а не вбудовані в ядро. (Це також відрізняється від більш традиційних API, де такі функції, як відступи, вбудовані в загальне ядро кожного компонента макета.) Тож, наприклад, щоб відцентрувати віджет, замість коригування умовної Alignвластивості, ви обертаєте його у Center віджет.

Є віджети для заповнення, вирівнювання, рядків, стовпців і сіток. Ці віджети макета не мають власного візуального представлення. Натомість їхня єдина мета — контролювати деякі аспекти макета іншого віджета. Flutter також містить утиліти-віджети, які використовують переваги цього композиційного підходу.

Наприклад, Container - начастіше використовуваний віджет, складається з кількох віджетів, що відповідають за макет, малювання, позиціонування та розмір. Зокрема, Container складається з віджетів - LimitedBox, ConstrainedBox, Align, DecoratedBoxі Transform, які можна побачити, прочитавши його вихідний код.

Визначальною характеристикою Flutter є те, що можна детально ознайомитися з джерелом будь-якого віджета та перевірити його. Отже, замість того, щоб створювати підкласи Container для створення налаштованого ефекту, можна скомпонувати його та інші віджети новими способами, або просто створити новий віджет, використовуючи приклад Container - як натхнення.

**Створення віджетів**. Як згадувалося раніше, визначення візуального представлення віджета, та перевизначення методу build() - для повернення нового дерева елементів. Це дерево представляє частину інтерфейсу користувача віджета в більш конкретних термінах. Наприклад, віджет панелі інструментів може мати функцію побудови, яка повертає горизонтальний макет деякого тексту та різних кнопок. За потреби фреймворк рекурсивно запитує кожен віджет про створення,

доки дерево не буде повністю описано конкретними рендерабельними об’єктами. Потім фреймворк зшиває візуалізовані об’єкти в дерево візуалізованих об’єктів.

Функція створення віджета не повинна мати побічних ефектів. Кожного разу, коли функцію викликають, віджет повертає нове дерево віджетів, незалежно від того, що віджет повертав раніше. Фреймворк виконує важку роботу, щоб визначити, які методи збирання потрібно викликати на основі дерева об’єктів візуалізації. У кожному відрендереному кадрі Flutter може відтворити лише ті частини інтерфейсу користувача, де стан змінився, викликавши build() метод цього віджета. Тому важливо, щоб методи побудови поверталися швидко, а важка обчислювальна робота повинна виконуватися деяким асинхронним способом, а потім зберігатися як частина стану, який буде використаний методом побудови.

Незважаючи на відносно наївний підхід, це автоматизоване порівняння є досить ефективним, оскільки дозволяє створювати високопродуктивні інтерактивні програми. Крім того, дизайн функції збірки спрощує ваш код, зосереджуючись на оголошенні того, з чого складається віджет, а не на складнощах оновлення інтерфейсу користувача з одного стану в інший.

**Стан віджета**. Фреймворк представляє два основних класи віджетів: віджети із збереженням *стану* та віджети *без стану.*

Багато віджетів не мають змінного стану: вони не мають жодної із властивостей, які змінюються з часом (наприклад, значка чи мітки). Ці віджети підкласу StatelessWidget.

Однак, якщо унікальні характеристики віджета потребують змін на основі взаємодії користувача чи інших факторів, цей віджет має *статус*.

Наприклад, якщо віджет має лічильник, який збільшується щоразу, коли користувач натискає кнопку, тоді значення лічильника є станом цього віджета. Коли це значення змінюється, віджет потрібно перебудувати, щоб оновити його частину інтерфейсу користувача. Ці віджети підкласи StatefulWidget, і (оскільки сам віджет є незмінним) вони зберігають змінний стан в окремому класі, який підкласи State. StatefulWidgets не мають методу побудови; натомість їхній користувальницький інтерфейс будується через їхній Stateоб’єкт.

Кожного разу, коли ви змінюєте Stateоб’єкт (наприклад, шляхом збільшення лічильника), ви повинні викликати setState() , щоб повідомити фреймворку про оновлення інтерфейсу користувача, State знову викликавши метод збірки.

Наявність окремих об’єктів стану та віджетів дозволяє іншим віджетам обробляти віджети без стану, не турбуючись про втрату стану.

Замість того, щоб утримувати дочірній елемент та зберегти його стан, батько може створити новий екземпляр дочірнього елемента в будь-який час, не втрачаючи постійного стану дочірнього. Фреймворк виконує всю роботу з пошуку та повторного використання існуючих об’єктів стану, коли це необхідно.

**Архітектура BLoC**. Шаблон Flutter BLoC (компонент бізнес-логіки) — це архітектурний шаблон, заснований на окремих компонентах (компонентах BLoC). Компоненти BLoC містять лише бізнес-логіку, яку можна легко використовувати між різними програмами Dart. Спочатку шаблон BLoC був задуманий для повторного використання того самого коду незалежно від платформи: веб-програма, мобільний додаток, бек-енд. Отже, так, цей шаблон було розроблено з метою полегшити навантаження на розробників під час розробки програм для іншої платформи з ідеєю повторного використання коду (рис. 2.4).

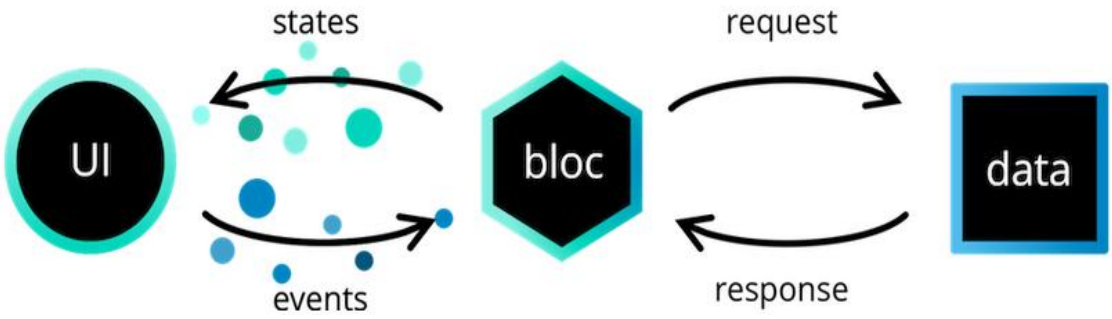


Рис. 2.4. Архітектура BLoC

**Потоки**. У загальному розумінні stream - це безперервний потік або послідовність будь-чого. Але технічно кажучи, Stream — це не що інше, як безперервний потік даних. Візьмемо для прикладу конвеєрні стрічки**.** Конвеєрна стрічка має два кінці, з одного кінця товар проходить (вхід), потім він обробляється і після обробки виходить з іншого кінця стрічки (вихід).

Ключові терміни:

- **Stream,** конвеєрна стрічка називається струмком;
- **StreamController,** це те, що контролює потік;
- **StreamTransformer,** це те, що обробляє вхідні дані;
- **StreamBuilder,** це метод, який приймає потік як вхідні дані та надає нам конструктор, який перебудовується щоразу, коли є нове значення потоку;
- **sink,** властивість, яка приймає вхід;
- **stream,** властивість, яка дає вихід із потоку.

Маючи трохи уявлення про потоки, та повертаючись до Flutter, однією з головних перешкод, з якими можна стикнутися під час кодування у Flutter, полягає в роботі з інтерфейсом користувача та всім іншим, оскільки у Flutter немає жодної проміжної мови проектування для інтерфейсу користувача, як в Android є XML. Замість цього написаний у Flutter код зберігається в одному місці. Ось тут і вступає в гру BLoC, який допомагає ефективно усунути всі перешкоди.

- BLoC не тільки вирішує проблему спільного використання коду, але й використовує функціональні можливості Streams для керування та поширення змін стану у Flutter.
- BLoC допомагає нам відокремити бізнес-логіку від інтерфейсу користувача.

Іншими словами, компоненти інтерфейсу повинні турбуватися лише про інтерфейс, а не про логіку. Таким чином, незважаючи на те, що Flutter не має мови-посередника, з використанням BLoC, можна розділити обидві речі, роблячи незалежними одна від одної. І так як BLoC використовує *Streams,* то можна зробити відповідні висновки щодо його використання, як показано на рис. 2.5:

- Вхідні дані приймаються за допомогою властивості sink;
- Вихід надається за допомогою властивості stream;
- Віджети надсилають події до BLoC через приймачі;
- BLoC сповіщає віджети через потоки;
- Тепер, коли бізнес-логіка та компоненти інтерфейсу користувача розділені, є можливість будь-коли змінити бізнес-логіку програми без жодного

впливу на інтерфейс користувача;

- Так само можливо змінити інтерфейс користувача без будь-якого впливу на бізнес-логіку.

Після того як було встановлено що із себе представляє даний архітектурний підхід, логічним виявляється питання навіщо для state management використовувати саме BLoC, коли в екосистемі платформи Flutter вже існують інші шляхи. Для визначення відповіді на це питання потрібно більш детально дослідити інші підходи і переконатися що BLoC є найефективнішим способом.

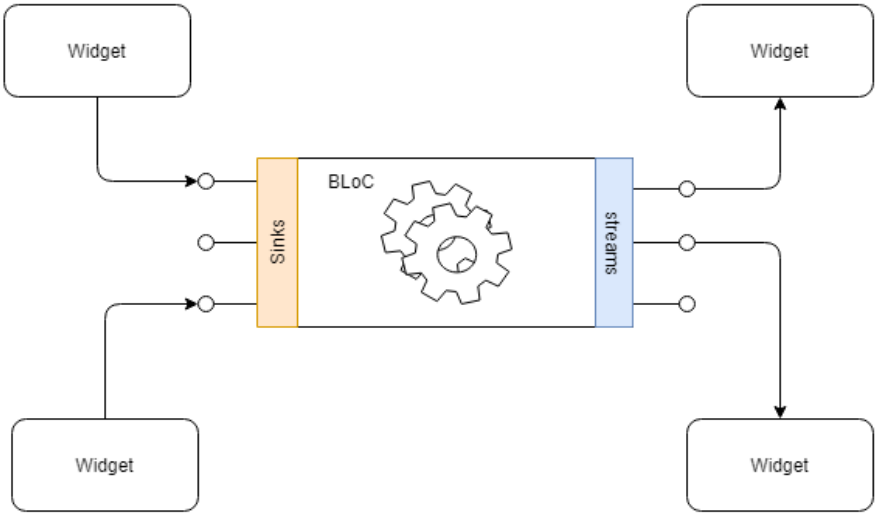


Рис. 2.5. Структура BLoC

Першим способом контролю стану застосунку є метод setState(). Ця функція повідомляє фреймворк про зміну внутрішньої моделі об’єкту. Кожний раз, коли потрібно поміняти об’єкт стану, зміни потрібно робити в функцію, яка передається в setState() як аргумент. Виклик setState() повідомляє фреймворк, що зміни у внутрішній моделі об’єкту можуть викликати мутації користувацького інтерфейсу в цьому піддереві. В свою чергу фреймворк запановує повторний build інтерфейсу застосунку. В даному підході існують дві основні проблеми. По перше, навіть проста зміна стану може викликати build всього дерева віджетів. По друге, це не вирішує проблему розділення коду для бізнес логіки і UI.

Другим способом є використання класу InheritedWidget. Наведений клас –

спеціальний віджет, який визначає контекст в корені потрібного піддерева, з можливістю доступу до даного контексту в буде-якому дочірному віджеті. Основні його функції - це поширення змін вниз по дереву і оновлення віджетів при кожному build. Проблемами даного способу є те, що стан InheritedWidget є final, що означає неможливість його зміни після створення. Також не вистачає функціональних можливостей для звільнення ресурсів і уникнення витоку даних.

Третім підходом є використання Scoped Model. Насправді це сторонній пакет, який підтримує Brain Egan. Він побудований на основі InheritedWidget, пропонуючи трохи кращий спосіб для доступу, запису і зміни стану застосунку. Це дозволяє нам передачу даних від батьківських до дочірніх віджетів, а також оновлювати всі віджети, які використовують модель коли вона змінюється. Для реєстрації змін стану в Scoped Model викликається метод notifyListeners(). На жаль, недоліком з ростом моделі буде важко орієнтуватися де саме потрібно викликати той самий notifyListeners().

З наведеного вище можна зробити висновок, що альтернативи мають деякі серйозні недоліки, в той час коли BLoC вирішує наведені проблеми.

Тепер постає питання коли потрібно використовувати даний підхід для максимальної ефективності. Для цього потрібно розібрати поняття локального і глобального стану. Для пояснення можна навести такий приклад: Форма входу в систему, де користувач має ввести ім’я і пароль і комунікація з API для валідації даних користувача перед спрацювання аунтефікації. В цьому випадку форму і її валідацію можна представити у вигляді локального стану, так як ця частина логіки відноситься лише до одного компоненту програми і більше не зустрічається. В той самий час, від комунікації застосунку з API залежить робота усього застосунку, тому це відноситься до глобального стану. Під час виконання роботи було визначено, що BLoC найкраще використовувати для локального стану. У випадку з глобальним станом варто подивитися в сторону інших підходів, наприклад, Redux.

## 2.2. Аналіз методів роботи фреймворку React Native

React Native — це кросплатформенний фреймворк, вперше представлений у 2015 році Facebook, який обіцяв стати універсальним рішенням для запуску нативних програм, які працюватимуть на кількох платформах і матимуть єдину кодову базу в основі. Оригінальна архітектура задовольняла потреби розробників, але вона також мала ряд недоліків і вразливостей, які часто викликали проблеми в процесі розробки. Завдяки сильній підтримці спільноти та зростанню популярності React серед веб-розробників, React Native вдалося набрати обертів і стати одним із найпопулярніших інструментів для створення економічно ефективних, масштабованих і привабливих програм.

Щоб зміцнити позиції фреймворку на ринку, у 2018 році Facebook оголосив про оновлену архітектуру React Native, яка, як очікувалося, зробила технологію більш надійною та вирішила довгострокові проблеми, на які скаржилися розробники програмного забезпечення всі ці роки.

**React порівняно з React Native.** Простіше кажучи, React Native не є "новішою" версією React, хоча React Native використовує її.

React (також відомий як ReactJS) - це бібліотека , яка використовується для створення інтерфейсу веб-сайту. Аналогічно React Native, він також був розроблений командою інженерів Facebook.

Тим часом React Native, що працює на базі React, дає змогу розробникам використовувати набір компонентів призначеного для користувача інтерфейсу для швидкої компіляції та запуску додатків для iOS і Android.

Як React, так і React Native використовують суміш JavaScript і спеціальної мови розмітки JSX. Однак синтаксис, який використовується для відображення елементів у компонентах JSX, відрізняється між React і React Native. Крім того, React використовує деякі HTML і CSS, тоді як React Native дає змогу використовувати власні елементи мобільного користувацького інтерфейсу.

**Стара архітектура React Native. Компоненти React Native.**

React Native - це платформо-діагностичне рішення, що означає, що воно не

має жодних зв'язків з конкретною платформою і може бути однаково добре використано для більш ніж однієї операційної системи. Розробники React Native пишуть єдину кодову базу, використовуючи код JavaScript, а потім транскрибують дерево React, щоб його можна було інтерпретувати нативною інфраструктурою.

Тобто завдання React Native полягало в тому, щоб забезпечити перетворення дерева React-компонентів у щось таке, що виявиться працездатним на різних мобільних платформах. Це означає наступне:

- Коректне формування інтерфейсу.
- Забезпечення доступу до нативних можливостей різних платформ.

Ще у 2013 році React Native об'єднав веб-технологію React з власним стеком нативних платформ, який часто був розрізненим і повільно ітеративним. Тоді у 2015 році з'явилася ідея створити два потоки, об'єднати їх за допомогою моста (Bridge) і зробити так, щоб ці дві сторони спілкувалися у стандартному форматі (рис. 2.6).

Поточна архітектура React Native базується на 3 основних стовпах:

1. Потік JS. Це - потік, у якому здійснюється читання і компіляція JavaScript-коду. Тут же виконується основна частина логіки програми. Бандлер Metro комбінує весь JS-код в єдиний файл, відповідає за обробку JSX- і TS-конструкцій. Потім отриманий код відправляють рушію JavaScriptCore, засобами якого цей код може бути запущений.

2. Потік Native. Тут відбувається виконання нативного коду. Цей потік взаємодіє з JavaScript-потоком тоді, коли потрібно оновити інтерфейс або звернутися до нативних функцій. Цей потік можна розділити на дві частини. Перша, Native UI, відповідає за використання нативних засобів формування інтерфейсу. Друга, Native Modules, надає доступ до особливих можливостей платформи, на якій працює додаток. Вона готова до роботи після запуску програми. Тобто, наприклад, якщо React Native використовує Bluetooth-модуль, він завжди буде активним, навіть у тому випадку, якщо він, насправді, не використовується.

3. Потік Shadow. У ньому виконується перерахунок макета програми. Тут використовується рушій Yoga, який є власною розробкою Facebook. У цьому потоці виконуються обчислення, пов'язані з формуванням інтерфейсу програми.

Результати цих обчислень відправляють потоку, відповідальному за виведення інтерфейсу.

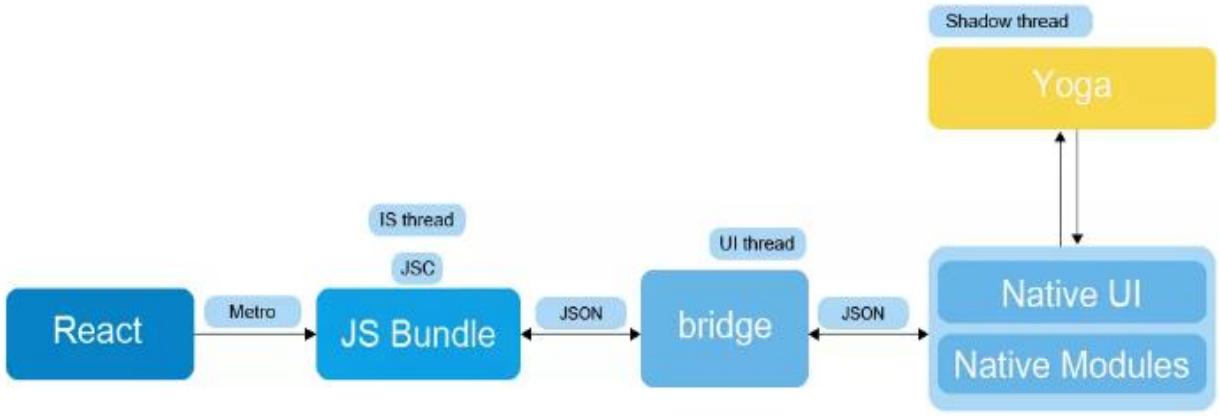


Рис. 2.6. Стара архітектура React Native

Для організації взаємодії між потоками JS і Native використовується модуль Bridge (міст). Цей модуль написаний на C++, в його основі лежить асинхронна черга. Коли міст отримує дані від однієї зі сторін, він серіалізує ці дані, перетворюючи в рядковий вигляд, і передає їх через чергу. Коли дані прибувають у пункт призначення, вони десеріалізуються.

Це означає, що всі потоки покладаються на асинхронні JSON-повідомлення, що передаються через міст. Кожна зі сторін відправляє ці повідомлення, очікуючи (але не маючи гарантії) того, що колись у майбутньому на ці повідомлення буде отримано відповідь. За такої схеми обміну даними існує ризик перевантаження каналу зв'язку.

Ось приклад, який зазвичай наводять, щоб проілюструвати те, як подібна схема обміну даними здатна спричинити проблеми з продуктивністю програми.

Припустимо, що користувач додатка виконує прокрутку величезного списку. Коли в нативному середовищі відбувається подія onScroll, інформація асинхронно передається в JavaScript-оточення. Але нативні механізми не чекають доти, доки JavaScript-частина застосунку зробить свою справу і їм про це відзвітує. Через це виникає затримка, що виражається в появі в списку, перед виведенням його вмісту, порожнього простору.

Аналогічно, перед тим, як результати перерахунку макета дійдуть до екрана, система має виконати кілька сеансів обміну даними. Так, перш ніж дані макета потраплять у нативне середовище, їх потрібно обробити в Yoga. А це, зрозуміло, означає, що такі дані теж доводиться передавати через міст.

Як видно, надсилання JSON-даних з використанням асинхронних механізмів, а також серіалізація та десеріалізація цих даних, спричиняють проблеми з продуктивністю. Цей механізм недосконалий. Але як ще можна налагодити взаємодію між нативними механізмами і JavaScript? Відповісти на це питання можна за допомогою технології JSI.

**Нова архітектура React Native.** Під час перепроєктування архітектури React Native виконується поступова відмова від функцій моста, на зміну яким приходить новий механізм, званий JavaScript Interface (JSI).

Застосування JSI відкриває можливості для деяких чудових поліпшень. Перше таке поліпшення полягає в тому, що JS-бандл більше не покладається на JSC. Інакше кажучи, рушій JSC тепер легко можна замінити на щось інше, що, цілком можливо, вирізняється вищою продуктивністю. Наприклад - на движок V8.

Друге поліпшення - це те, що лежить в основі нової архітектури React Native. Воно полягає в тому, що, завдяки використанню JSI, у JavaScript можна зберігати посилання на C++-об'єкти (Host Objects) і викликати методи цих об'єктів. У результаті виявляється, що JavaScript-частина додатка і нативні механізми знатимуть один про одного набагато більше, ніж раніше.

Іншими словами, JSI дає змогу забезпечити повну взаємодію між усіма потоками. Завдяки використанню концепції спільного володіння (shared ownership), JavaScript-код може запускати нативні методи безпосередньо з JS-потоку. При цьому немає необхідності в тому, щоб використовувати JSON для передачі даних. Це дає змогу позбутися проблем, характерних для використання мосту, пов'язаних із можливим переповненням черги та з асинхронним передаванням даних (рис. 2.7).

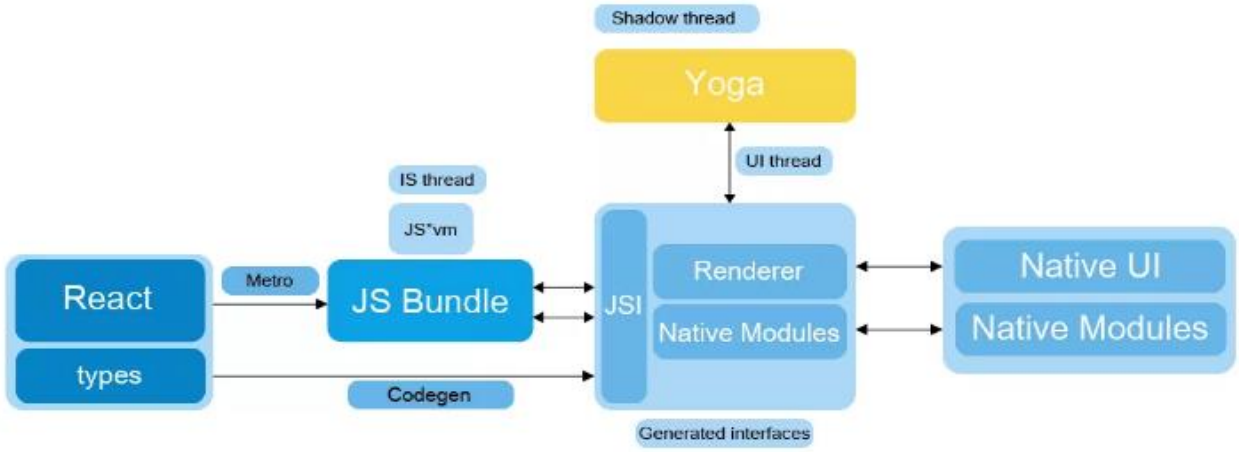


Рис. 2.7. Нова архітектура React Native

На додачу до того, що вищеописана схема значно покращує взаємодію між різними потоками, нова архітектура, крім того, дає змогу здійснювати безпосереднє керування нативними модулями. Це означає, що нативні модулі можна використовувати тоді, коли вони потрібні, а не ініціалізувати їх усі під час запуску програми. Це призводить до серйозного збільшення швидкості запуску додатків.

Цей новий механізм може відкрити дорогу до розробки нових проєктів, здатних на те, що було недоступно старим RN-додаткам. Річ у тім, що тепер у нашому розпорядженні опинилася міць C++. А це означає, що на базі React Native тепер можна буде створювати набагато більше різновидів додатків, ніж раніше.

**Як працює React Native?** Ідея написання мобільних додатків на JavaScript здається трохи дивною. Як можна використовувати React у мобільному середовищі? Щоб зрозуміти технічну основу React Native, спочатку нам потрібно буде згадати одну з функцій React, віртуальний DOM.

У React віртуальний DOM діє як прошарок між описом розробника того, як все має виглядати, та роботою, виконаною для реального відтворення вашої програми на сторінці. Щоб відобразити інтерактивні інтерфейси користувача в браузері, розробники повинні відредагувати DOM браузера або об’єктну модель документа. Це дорогий крок, і надмірні записи в DOM мають значний вплив на продуктивність. Замість того, щоб безпосередньо відтворювати зміни на сторінці, React обчислює необхідні зміни за допомогою версії DOM у пам’яті та повторно відображає мінімальну необхідну кількість (рис. 2.7).

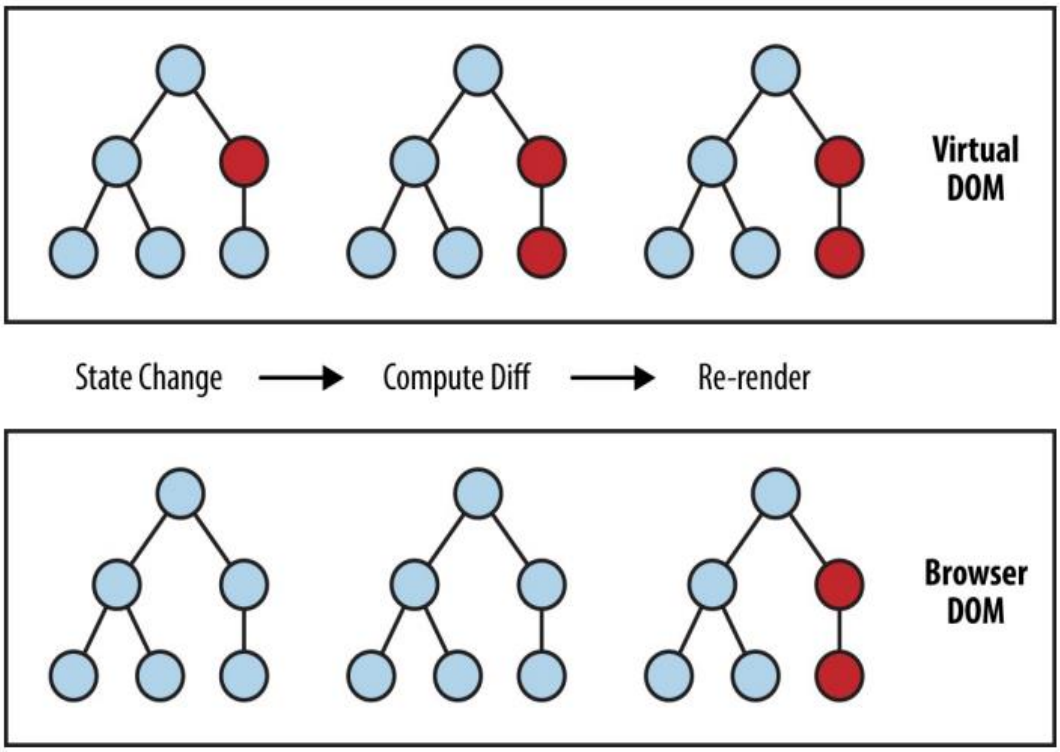


Рис. 2.7. Виконання обчислень у віртуальному DOM обмежує рендеринг у DOM браузера

У контексті React в Інтернеті більшість розробників сприймають Virtual DOM переважно як оптимізацію продуктивності. Віртуальний DOM, безумовно, має переваги в продуктивності, але його реальний потенціал полягає в потужності його абстракції. Розміщення чистого шару абстракції між кодом розробника та фактичним відтворенням відкриває багато цікавих можливостей. Що, якби React міг рендерити в ціль, відмінну від DOM браузера? Зрештою, React вже «розуміє», як має виглядати ваш додаток.

Дійсно, ось як працює React Native, як показано на рис. 2.8. Замість відтворення в DOM браузера React Native викликає API Objective-C для відтворення в компонентах iOS або API Java для відтворення в компонентах Android. Це відрізняє React Native від інших кросплатформних варіантів розробки додатків, які часто призводять до відтворення веб-подання.

Все це можливо завдяки «мосту», який надає React інтерфейс до рідних елементів інтерфейсу хост-платформи. Компоненти React повертають розмітку зі

своєї функції рендерингу, яка описує, як вони мають виглядати. З React для Інтернету це перекладається безпосередньо на DOM браузера.

Для React Native ця розмітка перекладається відповідно до хост-платформи, тому <View> може стати спеціальним UIView для iOS.

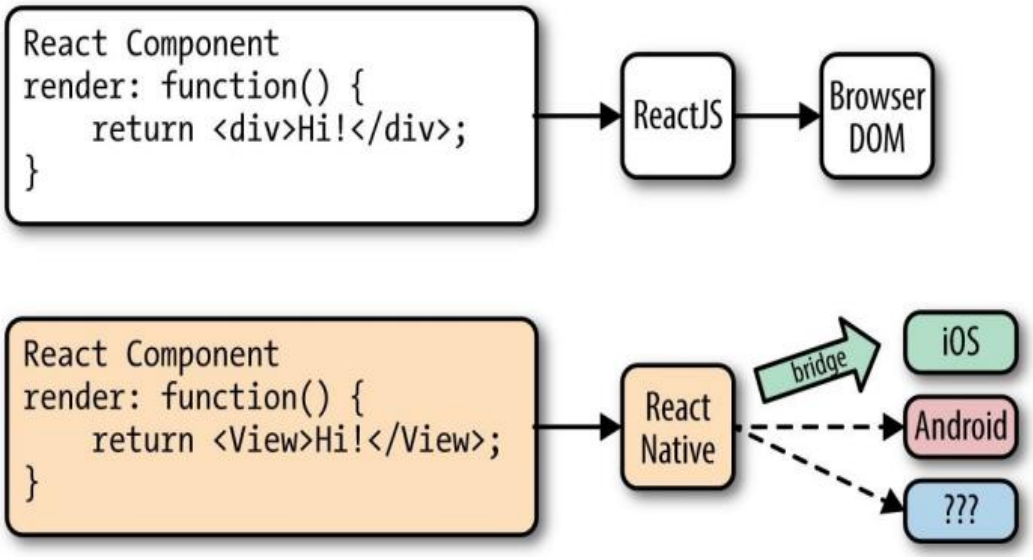


Рис. 2.8. React можна рендерити для різніх цілей

На даний момент React Native підтримує iOS та Android. Через рівень абстракції, наданий Virtual DOM, React Native також може націлюватися на інші платформи — на когось просто потрібно написати міст.

**Життєвий цикл візуалізації.** Якщо ви звикли працювати в React, життєвий цикл React повинен бути вам знайомий.

Коли React запускається в браузері, життєвий цикл візуалізації починається з монтування вашого React компоненти (рис. 2.9).

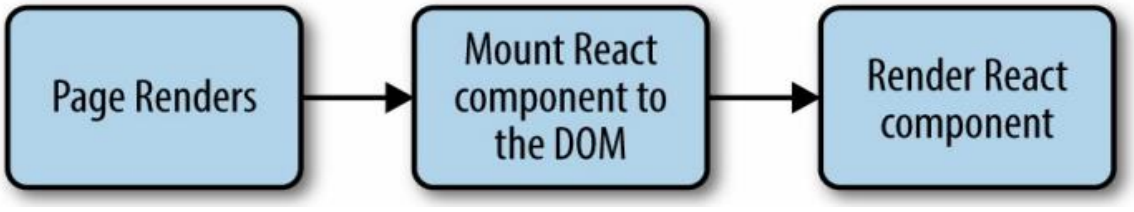


Рис. 2.9. Монтаж компонентів у React

Після цього React обробляє рендеринг і повторне рендеринг вашого компонента за потреби (рис. 2.10).

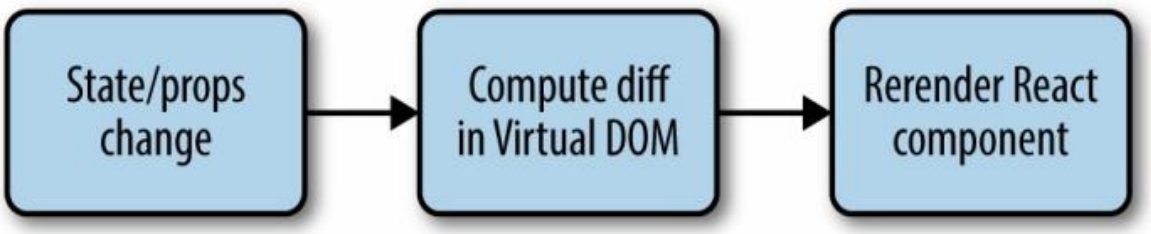


Рис. 2.10. Відтворення компонентів у React

Для етапу візуалізації розробник повертає HTML-розмітку з методу рендерингу компонента React, який React потім рендерить безпосередньо на сторінці, якщо це необхідно.

Для React Native життєвий цикл той самий, але процес візуалізації дещо відрізняється, оскільки React Native залежить від мосту. Раніше було коротко розглянуто міст (рис. 2.6). Міст перекладає виклики JS і викликає базові API та елементи інтерфейсу хост-платформи (тобто в Objective-C або Java, відповідно). Оскільки React Native не працює в основному потоці інтерфейсу користувача, він може виконувати ці асинхронні виклики, не впливаючи на роботу користувача.

**Створення компонентів у React Native.** Весь код React створений на компонентах. Компоненти React Native здебільшого такі ж, як і компоненти React, але з деякими важливими відмінностями щодо рендерингу та стилізації.

**Робота з Views.** Під час написання в React для Інтернету ви відтворюєте звичайні елементи HTML ( , <p>, , <a> тощо). За допомогою React Native усі ці елементи замінено специфічними для платформи компонентами React (рис. 2.11). Найпростішим є кросплатформний <View>, простий і гнучкий елемент інтерфейсу користувача, який можна розглядати як аналог . На iOS наприклад, компонент <View> рендериться у UIView, тоді як на Android він рендериться у View.

| React | React Native |
|---|---|
|  | <View> |
|  | <Text> |
| <li>, <ul> | <ListView> |
| <img> | <Image> |

рис. 2.11. Різниця поміж компонентами React та React Native

Інші компоненти залежать від платформи. Наприклад, компонент <DatePickerIOS> (передбачувано) відображає стандартний засіб вибору дати iOS. Ось уривок із прикладу програми UIExplorer, який демонструє засіб вибору дати iOS. Використання просте, як і слід було очікувати:

```
<DatePickerIOS
  date={this.state.date}
  mode="date"
  timeZoneOffsetInMinutes={this.state.timeZoneOffsetInHours * 60}
/>
```

Рис. 2.12. Фрагмент коду

Це відображається у стандартному засобі вибору дати iOS (рис. 2.13).


Рис. 2.13. DatePickerIOS, як випливає з назви, є специфічним для iOS

Оскільки всі наші елементи інтерфейсу тепер є компонентами React, а не базовим HTML таких елементів, як , вам потрібно буде явно імпортувати кожен компонент, який ви хочете використовувати. Наприклад, нам потрібно було імпортувати компонент <DatePickerIOS> таким чином:

```
var React = require('react-native');
var {
  DatePickerIOS
} = React;
```

Рис. 2.14. Фрагмент коду

Програма UIExplorer, яка входить до стандартних прикладів React Native, дозволяє переглядати всі підтримувані елементи інтерфейсу користувача.

Оскільки ці компоненти відрізняються від платформи до платформи, те, як ви структуруєте свої React компоненти стають ще більш важливими під час роботи в React Native. У React для Інтернету ми часто маємо суміш компонентів React: одні керують логікою та їхніми дочірніми компонентами, тоді як інші компоненти відтворюють необроблену розмітку. Якщо ви хочете повторно використовувати код під час роботи в React Native, збереження розділення між цими типами компонентів стає критичним. Компонент React, який рендерить елемент <DatePickerIOS>, очевидно, не можна повторно використовувати для Android. Однак компонент, який інкапсулює асоційований

логіку можна використовувати повторно. Потім компонент перегляду можна замінити на основі вашої платформи. Ви також можете призначити версії компонентів для певної платформи, якщо хочете, щоб мати, наприклад, файл picker.ios.js і picker.android.js. Ми розглянемо це «Компоненти з версіями для певної платформи».

**Використання JSX.** У React Native, як і в React, JSX використовується для написання представлень, поєднуючи розмітку та JavaScript, які компілюються в один файл. JSX зустрів сильну реакцію, коли React вперше дебютував. Бо ідея нового поділу файлів, а також поєднання розмітки, логіки керування та навіть стилів в одному місці, збити розробників з пантелику.

JSX надає пріоритет поділу проблем, а не поділу технологій. У React Native це ще суворіше дотримується. У світі без браузера є ще більш доцільним уніфікація стилів, розмітки та поведінки в одному файлі для кожного компонента. Відповідно, файли .js у React Native насправді є файлами JSX.

Компонент JSX, приблизно має вигляд, як компонент pureJavaScript React для Інтернету (рис. 2.15):

```
var HelloMessage = React.createClass({
  displayName: "HelloMessage",

  render: function render() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name
    );
  }
});

React.render(React.createElement(HelloMessage, { name: "Bonnie" }), mountNode);
```

Рис. 2.15. Фрагмент коду

За допомогою JSX з’явилася можливість написання коду більш стисло. Замість виклику React.createElement і передачі списку атрибутів HTML використовується XML-подібна розмітка (рис. 2.16):

```
var HelloMessage = React.createClass({
  render: function() {
    // Instead of calling createElement, we return markup
    return <div>Hello {this.props.name}</div>;
  }
});

// We no longer need a createElement call here
React.render(<HelloMessage name="Bonnie" />, mountNode);
```

Рис. 2.16. Фрагмент коду

Обидва фрагменти відображатимуть наступний HTML на сторінці (рис. 2.17):

```
<div>Hello Bonnie</div>
```

Рис. 2.17. Фрагмент коду

**Стилізація рідних компонентів.** В Інтернеті стилізування компонентів React відбувається за допомогою CSS, як і будь-якого іншого елементу HTML. Так як CSS є необхідною частиною Інтернету. Зазвичай React не впливає на написання CSS. Він полегшує використання вбудованих стилів та динамічно створює назви класів на основі атрибутів і стану, але в інших випадках - React здебільшого відіграє роль агностику щодо того, як стилі обробляються в Інтернеті.

Не веб-платформи мають широкий спектр підходів до макета та стилю. Під час роботи з React Native, на щастя, використовується один стандартизований підхід до стилю. Частина зв'язку між React і хост-платформою включає реалізацію сильно скороченої підмножини CSS. Ця вузька реалізація CSS покладається насамперед на flexbox для макета та фокусується на простоті, а не на реалізації повного спектру правил CSS.

На відміну від Інтернету, де підтримка CSS різниться в різних браузерах, React Native може забезпечити узгоджену підтримку правил стилю. Подібно до різних елементів інтерфейсу користувача, є можливість побачити багато прикладів підтримки стилів у програмі UIExplorer, яка є одним із прикладів, які поставляються з React Native.

React Native також наполягає на використанні вбудованих стилів, які існують як об'єкти JavaScript. Команда React раніше виступала за цей підхід в React для веб-додатків. Раніше вбудовані стилі виглядали подібним чином (рис. 2.18):

```
// Define a style…
var style = {
  backgroundColor: 'white',
  fontSize: '16px'
};

// ...and then apply it.
var tv = (
  <Text style={style}>
    A styled Text
  </Text>);
```

Рис. 2.18. Фрагмент коду

Також слід додати, що React Native надає можливіть роботи з деякими

утилітами для створення та розширення об’єктів стилю, які роблять роботу з вбудованими стилями більш керованим процесом.

**API платформи хосту**. Можливо, найбільша різниця між React для Інтернету та React Native полягає в API хост-платформах. В Інтернеті актуальною проблемою є фрагментарне та непослідовне прийняття стандартів - все ж більшість браузерів підтримують спільне ядро спільних функцій. Однак у React Native специфічні для платформи API відіграють набагато більшу роль у створенні чудового, природного відчуття взаємодії з користувачем.

Також є багато інших варіантів, які слід розглянути. Мобільні API включають усе: від зберігання даних до служб визначення місцезнаходження та доступу до обладнання, наприклад камери. Оскільки React Native розширюється на інші платформи, можлива поява інших видів API. Наприклад, як би виглядав інтерфейс між React Native і гарнітурою віртуальної реальності.

За замовчуванням React Native для iOS і Android включає підтримку багатьох часто використовуваних функцій, а React Native може підтримувати будь-який асинхронний рідний API. React Native робить зручним і простим використання API хост-платформи, тому є можливість вільно експериментувати. Обов’язково треба подумати про те, що буде здаватися «правильним» для конкретної цільової платформи, і створювати з урахуванням природних взаємодій.

Нажаль React Native Bridge не відкриє всю функціональність хост-платформи. Якщо потрібна непідтримувана функція, можливо самостійно додати її до React Native. Крім того, велика ймовірність, що хтось уже зробив це, тому обов’язково зв’яжіться зі спільнотою, щоб дізнатися, чи буде підтримка чи ні.

Також варто зазначити, що використання API хост-платформи має наслідки для повторного використання коду. Компоненти React, які потребують специфічної платформи, також будуть залежати від платформи. Ізоляція та інкапсуляція цих компонентів надасть програмі додаткову гнучкість. Також слід пам’ятати, що таких речей, як DOM, насправді не існує у React Native.

## 2.3. Висновки до другого розділу

У другому розділі в ході проведеного аналізу, були розглянуті такі аспекти, як: архітектура, принцип роботи та внутрішня структура фреймворків React Native та Flutter, які дозволяють розробляти високодинамічні мультиплатформенні мобільні додатки одночасно.

Окрім цього, був проаналізований та порівняний процес відображення графічного інтерфейсу. React Native, який для цього використовує нативні компоненти, та Flutter, який має власні набори віджетів, що надають дизайну інтерфейсу користувача ширшу кастомізацію. Тобто для створення нативного інтерфейсу можна використати як бібліотеку віджетів Cupertino, так і створити щось повністю індивідуальне. Таким чином, Flutter дозволяє обрати - використати будь-який віджет із бібліотеки, або розробити власне рішення.

Також були розглянуті методи управління станом додатку, та способи їх реалізації. До цього відносяться: стандартне керування станом між віджетами і компонентами, та керування станом, що підпорядковує підхід BLoC і Redux для великого потоку даних між компонентами мобільного додатку.

# РОЗДІЛ 3
# ПОРІВНЯННЯ ТА ДОСЛІДЖЕННЯ ЗАСОБІВ FLUTTER ТА REACT NATIVE

## 3.1. Аналіз, порівняння та дослідження фреймворків Flutter та React Native між собою.

Flutter проти React Native є дуже дискусійною темою, коли мова йде про вибір правильного фреймворку в ландшафті розробки мобільних додатків. Кожен із зазначених фреймворків має свої плюси та мінуси, як переваги, так і недоліки. Перш ніж приступити до глибокої дискусії Flutter проти React Native, слід знати важливість кросплатформної розробки мобільних додатків.

Оскільки спостерігається різке зростання використання мобільних телефонів і сильна залежність від додатків, необхідно мати точний час для створення єдиного додатку, який найкраще підходить для всіх платформ. Розробка має здійснюватися таким чином, щоб його можна було легко оновити. Такий підхід краще обслуговується, заощаджує кошти, а також час. Ось деякі з потенційних причин, за якими виникає важливість кросплатформної розробки мобільних додатків. Давайте глибше заглибимося в основну тему «Flutter Vs React Native».

1) Популярність: якщо розмірковівати над цією темою з точки зору «популярності Flutter проти React Native», тоді спадає на думку кілька речей, наприклад, яка з них найпопулярніша. Частка ринку Flutter у 2021 році становила 42%, React native – 38%, а інші фреймворки, разом узяті, можуть становити решту, включаючи Cordova, Ionic, Xamarin та інші. Можна зробити висновок, що з точки зору популярності та частки ринку нативи Flutter і React належали до фреймворків, які набули популярності у 2021 році, при цьому Flutter мав на 4% більшу частку ринку, ніж натив React (рис. 3.1).

2) Продуктивність: якщо проаналізувати нативну продуктивність Flutter проти React на мобільних пристроях, то Flutter швидший, ніж нативний React. Це

пояснюється тим, що Flutter передбачає компіляцію власних бібліотек ARM і x86 без будь-яких додаткових рівнів, що робить його швидким. React Native використовує кодування JavaScript, яке включає перехід від коду до нативного середовища мобільного пристрою, оскільки процес перемикання як третього колеса споживає більше ресурсів і часу.

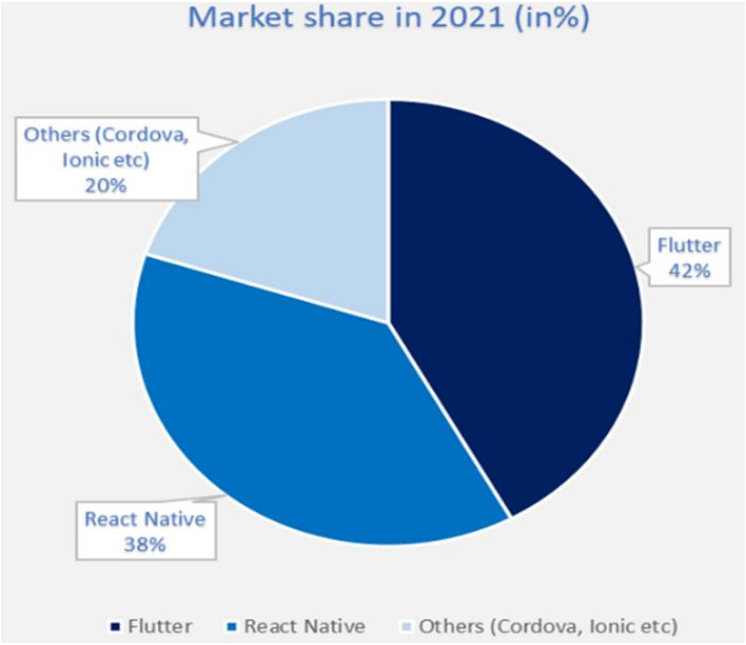


Рис. 3.1. Частка ринку Flutter та React Native у 2021 році

3) Узгодженість: на основі узгодженості - мова Flutter вважається узгодженою на всіх платформах (як на iOS, так і Android). Розробникам просто потрібно додати матеріальний інтерфейс користувача або Cupertino. Однак у випадку з React Native - розробникам потрібно спочатку запустити код на пристроях iOS і Android, щоб перевірити узгодженість.

4) Елементи та функції: Flutter сприяє швидшій розробці, оскільки підтримує дизайн інтерфейсу користувача на основі віджетів і функції гарячого перезавантаження, щоб точно виявляти зміни. Тим самим допомагаючи розробникам розробити MVP додатків за короткий проміжок часу. Навіть нативний React використовує функцію гарячого перезавантаження та використовує нативні компоненти, які одразу доступні для використання. Отже, це сприяє простішій

розробці та чудовій взаємодії з користувачем. Обидві рамки є помітними на основі цього аспекту.

5) Надійність: React Native підтримує величезну кількість бібліотек і має спільноту досвідчених професіоналів. Нативні елементи React також включають використання нативних графічних компонентів, а також з точки зору порівняння коду React Native випереджає Flutter, оскільки багато людей вважають, що мова програмування JavaScript більш популярна. Тому на даний момент React Native надійніший, ніж Flutter. Але з часом це може змінитися.

6) Економічна ефективність: з точки зору економічної ефективності це сприяє швидкому розвитку та скорочує час і витрати.

Остаточне рішення щодо того, який фреймворк має бути на картках організації, не залежатиме виключно від вищезгаданих факторів. Вибираючи між Flutter vs React Native, компаніям слід розглянути доцільність, життєздатність і провести аналіз ринку разом із аналізом внутрішніх компетенцій компанії чи Startup. З найвідоміших технологій, які обговорювалися, обидві мають як переваги, так і недоліки. Це залежить від кількох факторів, таких як швидкість, час розробки, час виходу на ринок, обсяг проекту, вимоги клієнта, попит і пропозиція технічного таланту, стадія компанії, чи є вона стартапом чи зрілою компанією, конкуренція на ринку, вартість розробки, відповідний розподіл ресурсів, а також планування ресурсів. React Native (запущений Facebook у 2015 році), будучи старішим, більше прийнятий зрілими компаніями та іншими компаніями Behemoth. З іншого боку, Flutter (запущений компанією Google у 2017 році) набув популярності, а також почав використовуватися компаніями-початківцями, оскільки ресурси є економічно ефективними, а якісні продукти розробляються за коротший час виходу на ринок. Знову ж таки, це залежить від здійсненності та розсуду компанії, наприклад, час виходу на ринок має вирішальне значення для кількох фінтех-стартапів, тому багато з них створили свій мобільний додаток за допомогою Flutter і отримали чудові результати. Деякі інші зрілі стартапи використовували React Native, оскільки їхня мета полягала в тому, щоб показати приклад брендингу взаємодії з користувачем. Тому, вибираючи відповідну структуру, необхідно

проаналізувати сильні сторони, слабкі сторони, можливості та загрози та вибрати структуру, яка відповідає цілям компанії.

## 3.1.Розробка мобільного додатку для визначення ефективності фреймворку Flutter

В рамках дослідження цієї кваліфікаційної магістрерської роботи, була розроблена мобільна соц-мережа для визначення ефективності Flutter, як фреймворку для розробки високодинамічних мобільних застосунків.

Чат-додатки - це засіб, за допомогою якого сьогодні спілкується світ. Кожен користувач смартфону залучений до щоденного спілкування за допомогою наявних чат-додатків. Серед переліку широко використовуваних і популярних чат-додатків, одним з них є чат-додаток WhatsApp. Додаток досить простий і зручний у використанні з сучасним інтуїтивно зрозумілим інтерфейсом. Майже будь-хто може використовувати його без будь-якої попередньої взаємодії з цим додатком, оскільки загальний робочий процес та дизайн інтерфейсу легко зрозуміти. Таким чином, такі додатки зробили світ меншим. Зважаючи на високий попит на функції обміну повідомленнями в кожному додатку, існує також високий попит на інтуїтивно зрозумілі та потужні екрани інтерфейсу з найсучаснішими функціями.

Елементи, що використовуються в інтерфейсі, впливають на загальну зручність використання програми. Це впливає на користувацький досвід та взаємодію з додатком. Отже, під час розробки треба забезпечити чистий та інтуїтивно зрозумілий інтерфейс для користувачів.

Дивлячись на те, наскільки простий користувальницький інтерфейс додатку для обміну повідомленнями WhatsApp, було створено інтерфейс, що нагадує інтерфейс оригінального додатку WhatsApp, використовуючи фреймворк Flutter для розробки даного мобільного додатку.

Основна мета - продемонструвати програмування додатків Flutter шляхом створення екрану списку розмов WhatsApp. Завдяки можливості створювати ідеальні піксельні дизайни інтерфейсу, є можливість побачити, як Flutter робить

реалізацію інтерфейсу легкою та красивою.

Розроблений додаток повинен виконувати такі функції:

- Обмін повідомленнями: є можливість обміну повідомленнями з вибраним контактом.
- Контакти: є можливість створювати групи с обраними контактами.
- Статус: є можливість створити власну історію, поділитися нею із друзями.
- Інформація: є можливість переглянути свою інформацію або відредагувати її.
- Рядок пошуку: є можливість пошуку контакту за допомогою пропозицій, щоб швидко його знайти.
- Камера: є можливість відкрити свою камеру і зробити гарну фотографію.

Переходячи до розробки, передусім було створено новий проект Flutter. Для цього заздалегідь потрібно було переконатися, що Flutter SDK та інші вимоги, пов'язані з розробкою Flutter-додатків, встановлені належним чином. При створенні проекту, в потрібному локальному каталозі було виконано наступну команду:

```
flutter create meApp
```

Після того, як проект майбутного мобільно додатку було створено, у каталозі проекту в терміналі була виконана наступна команда для запуска проекту в доступному емуляторі, або на реальному пристрої:

```
flutter run
```

Після успішної збірки на головному екрані комп’тера з'явився готовий до роботи емулятор із чистою сторінкою.

Для дослідження процесу створення функціональної частини додатку, як приклад, розглянемо процес створення головоного екрану «Chats», та його структуру. І на основі отриманої інформації будуть створені екрани «Camera»,

«Status» і «Calls».

**Екран «Chats».** Даний розділ матиме два основні розділи:

- розділ навігації;
- розділ списку чатів;

Створюємо файли екрана в окремій папці. Знаходимо папку ./lib, в якій створюємо папку під назвою ./screens, де зберігатиметься файл екрану програми. Після цього у папці ./lib/screens створюємо файл під назвою chats.dart. Тепер додаємо простий клас віджета Stateful у файл conversion.dart:

```
import 'package:flutter/material.dart';
class Chats extends StatefulWidget {
  @override
  _ ChatsState createState() => _ChatsState();
}
class _ ChatsState extends State< Chats > {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        elevation: 0.0,
        title: Text("MeApp"),
        backgroundColor: Color(0xFF128C7E),
      ),
      body: Container()
    );
  }
}
```

**Створення розділу «Список чатів».** Тепер додаємо інтерфейс списку чатів безпосередньо під розділом навігації у нижньому контейнері екрана chats.dart. Тут створюємо окремий віджет для елементів списку бесід, який також називається бесідою. У каталозі ./lib/components потрібно створити файл під назвою chatsList.dart. Загальна реалізація кодування цього віджета представлена у

фрагменті коду нижче:

```
import 'package:flutter/material.dart';
Widget chat (
   String url, String name, String message, String time, bool messageSeen) {
 return InkWell(
  onTap: () {},
  child: Padding(
   padding: const EdgeInsets.symmetric(vertical: 12.0),
   child: Row(
    crossAxisAlignment: CrossAxisAlignment.start,
    mainAxisAlignment: MainAxisAlignment.start,
    children: [
     CircleAvatar(
      backgroundImage: NetworkImage(url),
      radius: 25.0,
     ),
     SizedBox(
      width: 8.0,
     ),
     Expanded(
      child: Column(
       mainAxisAlignment: MainAxisAlignment.start,
       crossAxisAlignment: CrossAxisAlignment.start,
       children: [
        Row(
         children: [
          Expanded(
           child: Text(
            name,
            style: TextStyle(
             fontSize: 16.0,
             fontWeight: FontWeight.w700,
            ),
           ),
```

```
                ),
                Text(time),
              ],
            ),
            SizedBox(
              height: 5.0,
            ),
            Row(
              children: [
                Expanded(child: Text(message)),
                if (messageSeen)
                  Icon(
                    Icons.check_circle,
                    size: 16.0,
                    color: Colors.green,
                  ),
                if (!messageSeen)
                  Icon(
                    Icons.check_circle_outline,
                    color: Colors.grey,
                    size: 16.0,
                  ),
              ],
            ),
          ],
        ),
      ),
    ),
  );
}
```

Віджет приймає п'ять властивостей. URL-адреса зображення, ім’я, повідомлення, час і логічне значення messageSeen. Усі ці атрибути передаються з

батьківського віджета. Тут InkWell було повернуто як батьківський компонент, щоб зробити кожен елемент у списку розмов доступним для кліку. Потім маємо віджет «Row» із його дочірнім віджетом, який показує зображення, ім’я користувача, останнє повідомлення, час повідомлення та піктограму з умовним рендерингом зі змінною messageSeen, яка показує, чи бачить повідомлення користувач чи ні. На виході маємо подібну структуру:

```
class Home extends StatelessWidget {
  final _tabs = <Widget>[
    Tab(icon: Icon(Icons.camera_alt)), Tab(text: 'CHATS'),
    Tab(text: 'STATUS'), Tab(text: 'CALLS'),
  ];

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
      length: _tabs.length,
      initialIndex: 1,
      child: Scaffold(
        // top app bar
        appBar: AppBar(
          title: Text('WhatsApp'),
          actions: <Widget>[
            IconButton(icon: Icon(Icons.search), onPressed: () {}),
            IconButton(icon: Icon(Icons.more_vert), onPressed: () {}),
          ],
          bottom: TabBar(tabs: _tabs),
        ),

        // body (tab views)
        body: TabBarView(
          children: <Widget>[
            Text('camera'),
            ChatList(),
```

```
        Text('status'),
        Text('calls'),
      ],
```

Дивлячись на структуру коду можемо зробити висновок, що Flutter має широкий набір «віджетів» програми, які часто використовуються (ListView, TabView, TopAppBar, тощо), та якими є можливість користування просто з коробки. І як результат маємо розроблений екран (рис. 3.2):

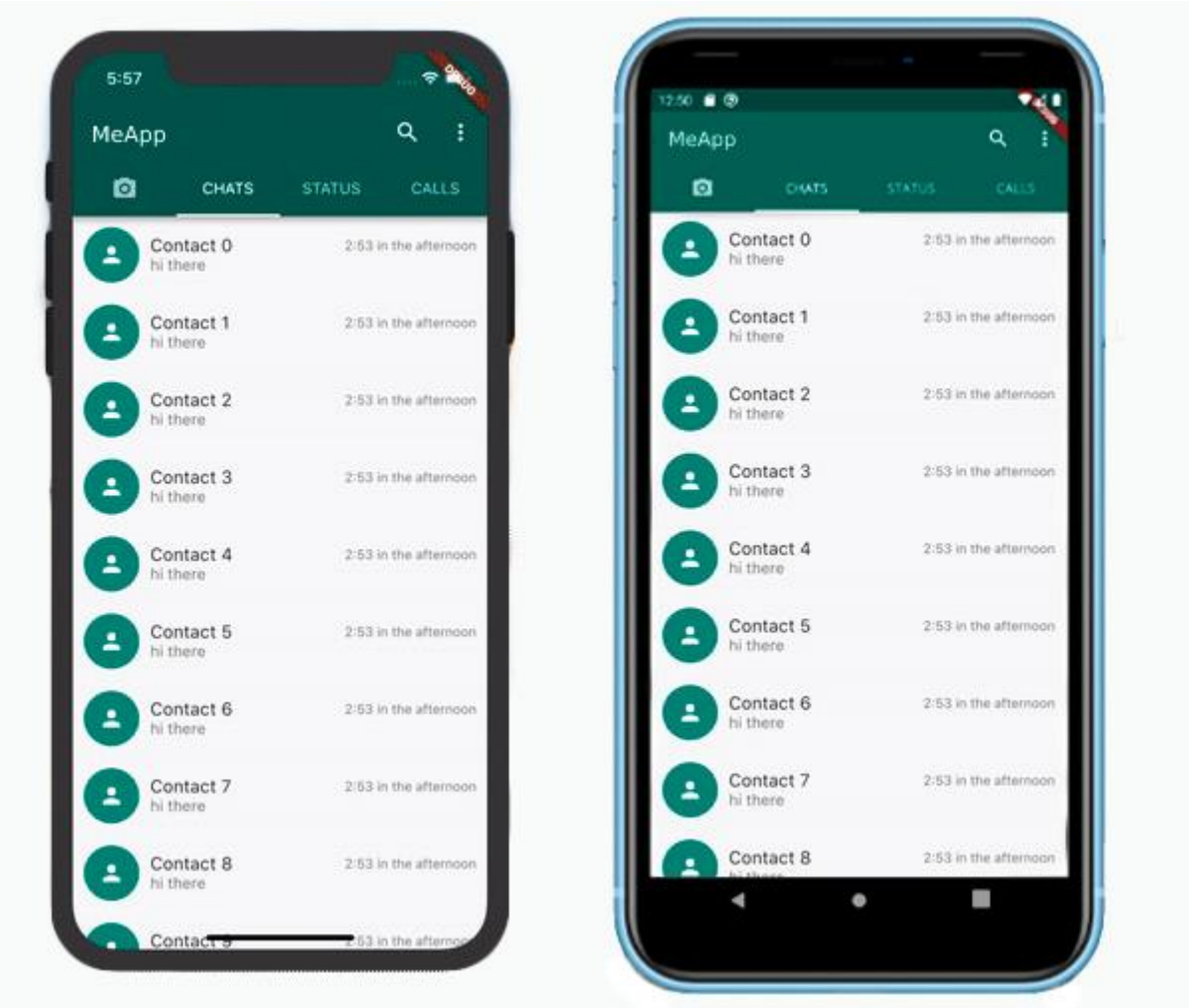


Рис. 3.2. Вигляд розробленого екрану «Chats» на платформах iOS та Android

Далі переходимо до створення ChatRoom( ).

Тут розміщуємо верхню панель, тіло повідомлень, а також нижню частину з введенням тексту, і кнопку надіслати. І так використовуючи комбінацію рядків і

стовпців, обгортаємо деякі дочірні елементи за допомогою Expanded ( flex: x, ), щоб контролювати скільки місця може займати кожний child (рис. 3.3).

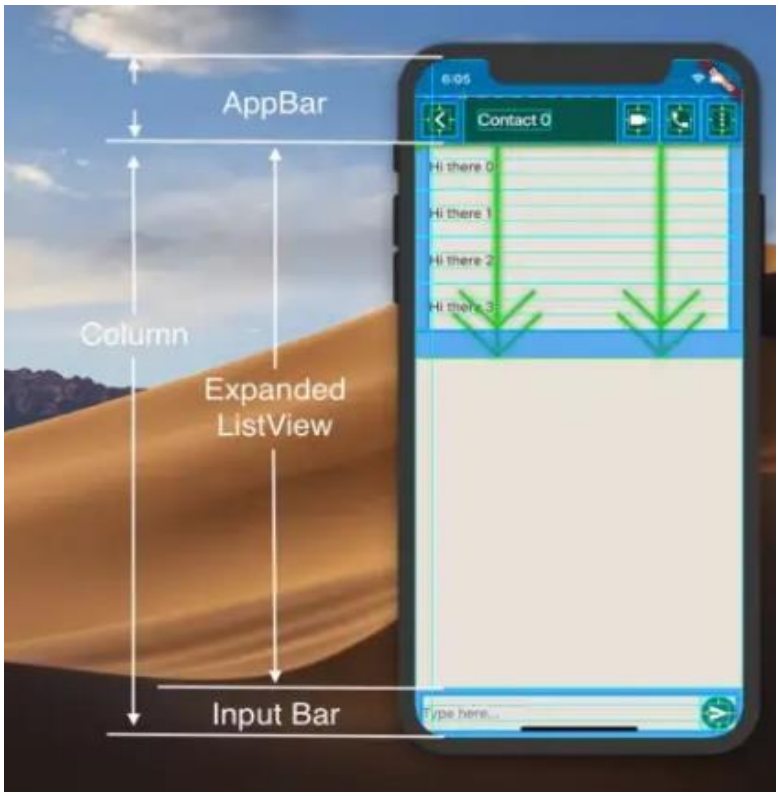


Рис. 3.3. Розміщення ChatRoom за допомогою Flutter's Column, Expanded тощо.

**Підключення ChatRoom() до ChatList().** Коли користувач торкається одного з ListTiles у вкладці CHATS ListView, то він переходить до перегляду чату, натискаючи перегляд ChatRoom поверх «стека навігації». Завдяки виклику методу Navigator.push у властивості onTap ListTile, маємо змогу це здійснити. Також є працююча кнопка «Назад» (←), яка буде автоматично додана як головний значок у верхній панелі додатків для повернення на головний екран (рис. 3.4).

Повертаючись до дизайну сторінки обміну повідомлень маємо верхню панель, віджет повідомлень і округле поле введення тексту. Налаштовуємо компоненти сторінки по черзі, починаючи з AppBar.

**Налаштування віджета — AppBar.** З коробки налаштування Material Design AppBar маємо лише один провідний значок. Також є присутнєю кнопка «Назад» (←), із провідним значком CircleAvatar (рис. 3.5).

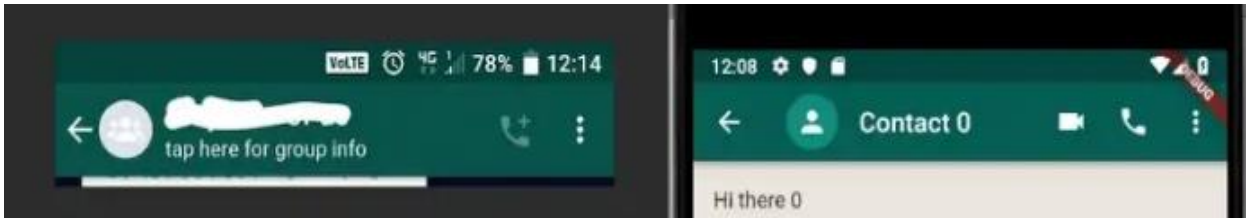


Рис. 3.4. Порівнюємо навігацію існуючого WhatsApp (ліворуч) із нашим (праворуч)


Рис. 3.5. Налагоджуємо відступи в AppBar

Додамо створений код:

```
return Scaffold(
    backgroundColor: Color(0xFFECE5DD),
    appBar: AppBar(
      title: SizedBox(
        width: double.infinity,
        child: Stack(
          overflow: Overflow.visible,
          children: <Widget>[
            Positioned(
              left: leftOffset,
              top: defaultIconButtonPadding,
              child: CircleAvatar(
                radius: avatarRadius,
```

```
      child: chatItem.avatar,
    ),
  ),
  Positioned(
    left: leftOffset + avatarRadius * 2 + 8.0,
    top: defaultIconButtonPadding +
        avatarRadius / 2 -
        titleLineHeight,
    child: Text(chatItem.name),
  ),
],
```

Кілька моментів, на які слід звернути увагу щодо наведеного вище фрагмента:

- SizedBox із нескінченною шириною імітує рядок із максимальною шириною;
- «leftOffset» є від’ємним значенням, через що дочірні елементи «переповнюються» лівим заповненням батьківського елемента;
- Віджет стека розміщує своїх дочірніх елементів за замовчуванням у верхньому лівому куті, один на одному.

Пам’ятаючи про дочірнє розміщення елементів за замовчуванням, розмістимо елементи child, обчисливши верхнє/праве/нижнє/ліве значення віджету. З точки зору CSS, це нестатичний позиціонований батьківський елемент з абсолютним позиціонуванням дочірніх елементів. І ось результат нашого коригування (рис. 3.6):

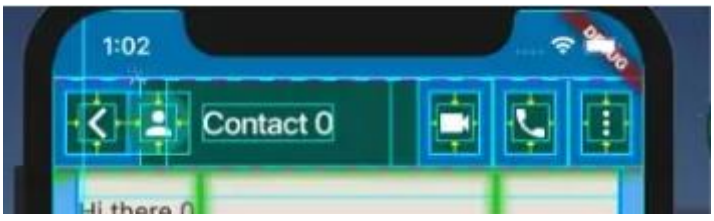


Рис. 3.6. Віджет стека дозволяє переповнювати Positioned-children

**Налаштування віджета — панель повідомлень.** На панелі повідомлень, як показано на рис. 3.7, маємо заокруглені (майже напівкруглі) кінці з однією піктограмою на початку, та двома піктограмами наприкінці текстового поля.

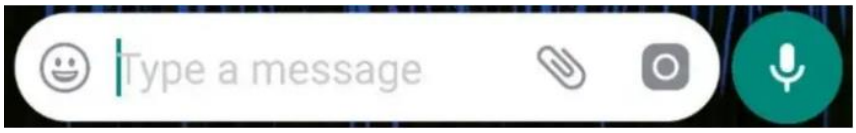


Рис. 3.7. Панель повідомлень

Розмістимо 3 іконки біля поля введення тексту — одну на початку і дві на кінці. Крім того, створимо це таким чином, щоб кінці стали заокругленими. Для заокруглення кутів віджета скористаємося ClipRRect Flutter, задавши йому бажаний радіус. Щоб зробити кінці панелі повідомлень більш закругленими, пропишемо більший радіус, щоб арки кутів зливалися. Отримаємо таку загальну реалізацію панелі повідомлень с використанням контейнеру ClipRRect:

```
final roundedContainer = ClipRRect(
  borderRadius: BorderRadius.circular(20.0),
  child: Container(
    color: Colors.white,
    child: Row(
      children: <Widget>[
        SizedBox(width: 8.0),
        Icon(Icons.insert_emoticon,
            size: 30.0, color: Theme.of(context).hintColor),
        SizedBox(width: 8.0),
        Expanded(
          child: TextField(
            decoration: InputDecoration(
              hintText: 'Type a message',
              border: InputBorder.none,
            ),
          ),
```

```
        ),
        Icon(Icons.attach_file,
            size: 30.0, color: Theme.of(context).hintColor),
        SizedBox(width: 8.0),
        Icon(Icons.camera_alt,
            size: 30.0, color: Theme.of(context).hintColor),
        SizedBox(width: 8.0),
      ],
    ),
  ),
);
```

Розмістимо цей округлений контейнер і кнопку надсилання всередині рядка, таким чином завершили InputBar (рис. 3.8):

```
final inputBar = Padding(
  padding: EdgeInsets.all(8.0),
  child: Row(
    children: <Widget>[
      Expanded(
        child: roundedContainer,
      ),
      SizedBox(
        width: 5.0,
      ),
      GestureDetector(
        onTap: () {},
        child: CircleAvatar(
          child: Icon(Icons.send),
        ),
      ),
    ],
  ),
);
```


Рис. 3.8. Налагодження друку рамок панелі введення

**Налаштування віджета — Потік повідомлень.** Тепер прийшов час зайнятись найважливішою частиною сторінки – контейнером з повідомленнями.

Замість прямокутника або прямокутника з маленьким трикутником, прикріпленим до верхнього правого кута як «дзьоб», віджет повідомлень у WhatsApp є «заокругленим прямокутником із заокругленим дзьобом».

Переглянувши документацію Flutter, і не знайшовши жодної готової реалізації для цього, було вирішено створити свою власну. Знадобилося вирізати прямокутник, використавши утиліту ClipPath. Це дозволило вирізати прямокутник, вказавши межу розрізу за допомогою координат (рівнянь).

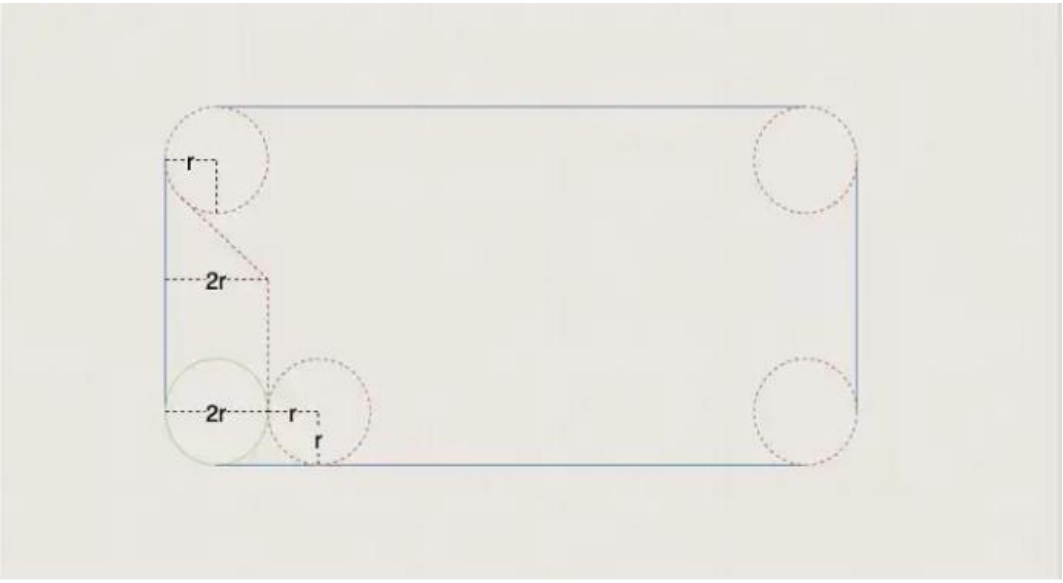


Рис. 3.9. Макет для вирізання нитки повідомлення із прямокутника

Використовуючи наведений вище план, таким чином був створений власний

потік повідомлень. Він спрямовує точку для переміщення з верхнього лівого кута, обертаючи проти годинникової стрілки, аж до верхнього лівого кута. Таким чином було отримано кінцевий ланцюжок повідомлень (рис. 3.10):


Рис. 3.10. Отриманий ланцюжок повідомлень

Решта роботи:

- встановлюємо максимальну ширину для кожної нитки;
- упаковуємо текст;
- створюємо «гнучкий початок/кінець» для вхідних/вихідних повідомлень;
- додаємо різні кольори для вхідних/вихідних повідомлень.

Використовуючи BoxContraint для максимальної ширини, встановлюємо softwrap true для Text, заздалегідь налаштувавши mainAxisAlignment у Row, і встановивши властивість кольору Container як основну.

Таким самим чином був розроблений головний екран “Chat”, і сторінка для обміну повідомленнями. Використаємо це як приклад, і створимо подібним чином екрани «Camera», «Status» та «Calls», та на виході отримаємо такий результат, як показано на рис. 3.11 та 3.12:

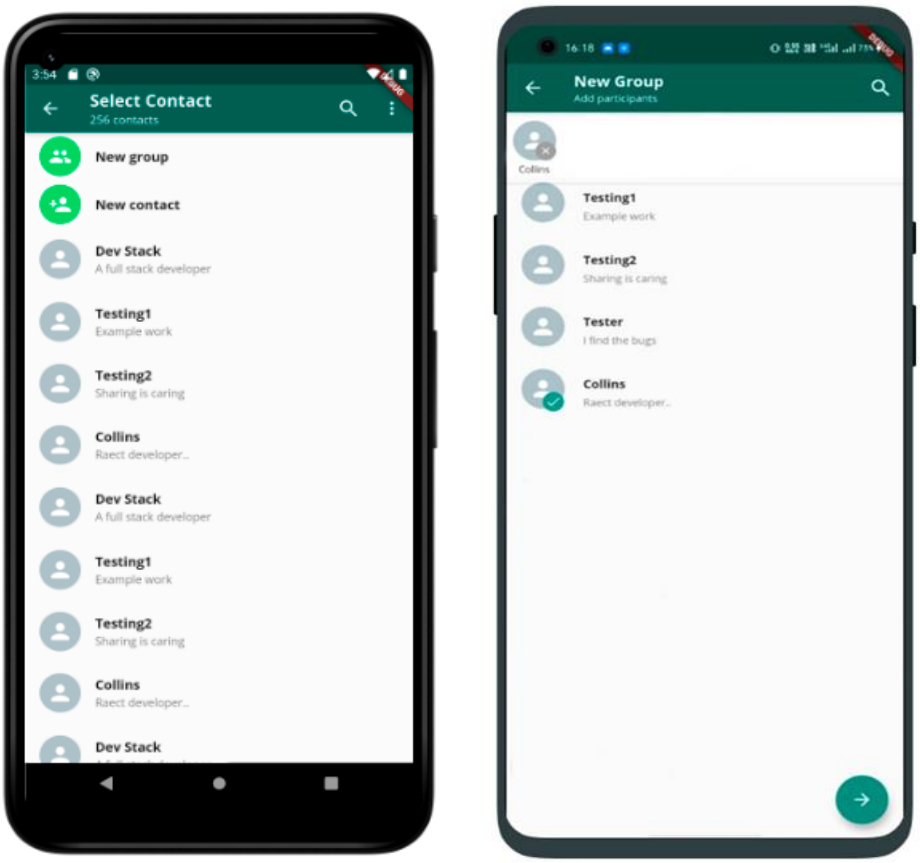


Рис. 3.11. Скріншоти розробленого екрану “Calls”

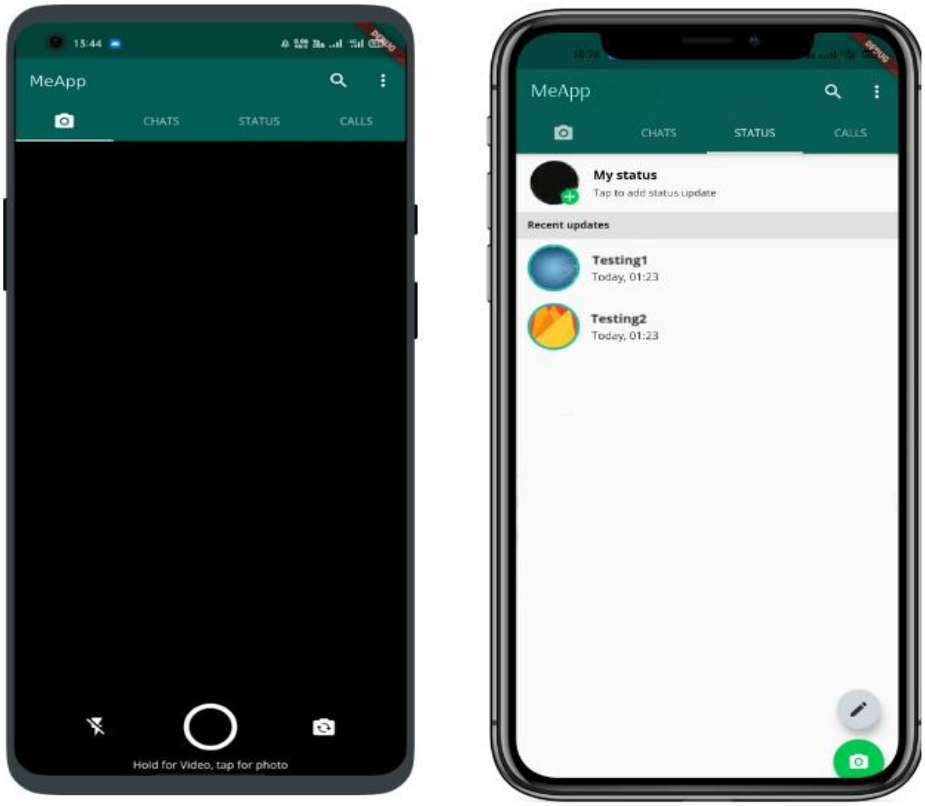


Рис. 3.11. Скріншоти розроблених екранів “Calls” та “Camera”

## 3.1. Висновки до третього розділу

В цьому розділі теоретично і на практиці були досліджені фреймворки для розробки мобільних додатків Flutter та React Native, а також було проведено порівняння між ними. Була взята до уваги різниця в підходах, процесі розробки та підтримці існуючих рішень. Крім того, була досліджена різниця в популярності, продуктивності, узгодженості, надійсноті та економічній ефективності.

За проведеним аналізом можна зробити висновки, що Flutter є сучаснішим за фреймворк React Native, і може похвалитися вищою продуктивністю, можливістю налаштування та кращою документацією. Flutter здатний забезпечити розробку більш масштабованих, швидких та високопродуктивних додатків, сумісних із численними платформами.

Крім цього, у процесі проведення дослідження було створено мобільний додаток, а точніше соціальну мережу, що реалізує всі функції сучасного високодинамічного мобільного додатку.

Дана соц-мережа має декілька екранів, за допомогою яких користувачі мають змогу обмінюватись повідомленнями з обраним контактом у реальному часі, завдяки розгорнутому серверу та створеній базі даних.

Створений додаток можна використовувати у будь-якому проєкті на Flutter іншими розробниками для дослідження ефективності взаємодії мови Dart з такими платформами, як iOS та Android. Використані методи та підходи можуть застосовуватися при розробці програмного забезпечення під мобільні пристрої на замовлення бізнесу, щоб розробка високодинамічних мобільних додатків проходила швидшими та ефективнішими методами.

## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Офіційна документація Flutter [Електронний ресурс]. – Режим доступу : https://flutter.dev/.
2. Офіційна документація React Native [Електронний ресурс]. – Режим доступу : https://reactnative.dev/.
3. Vladimir Novick. React Native – Building Mobile Apps with JavaScript. – Chapter 4 – 2017 – P. 114.
4. Rob Whitaker. Developing Inclusive Mobile Apps: Building AccessibleApps for iOS and Android. United States of America:Apress, 2020.359 с.
5. Windmill E. Flutter in Action Л.: Manning Publications, 2019. – 368 с.
6. Abshak P., Abshishek N. React Native for Mobile Development (2nd ed.) Л.: Apress, 2017. – 648 с.
7. Frank Zammetti. Practical Flutter: Improve your Mobile Development with Google’s Latest Open-Source SDK, 2020. 364 с.
8. Wargo J.Apache Cordova API Cookbook.Л.:Gardner Books,2017.–328 с.
9. Best Language For Mobile App Development. [Електронний ресурс]. – Режим доступу : https://www.webiotic.com/best-language-for-mobile-app-development-what-youneed-to-know/.
10. Shaun Lewis, Mike Dunn. Native Mobile Development: A Cross-Reference for iOS and Android. – Chapter 1 – 2019 – P. 1-3.
11. «Android. Збірник рецептів » від Яна Ф. Дарвіна. По суті книга – покрокова інструкція з прикладами для вирішення завдань по роботі з web-службами для досвідчених знавців Java і інтерфейсу Java SE API.
12. Eisenman, B. Learning React Native [Text] / Bonnie Eisenman. ¾ 2nd edition. ¾ Sebastopol: O’Reilly Media, 2017. ¾ 242 p.
13. Garbade, M. Native vs. cross-platform app development: pros and cons [Електронний ресурс] / Dr. Michael J. Garbade. ¾ Режим доступу: https://codeburst.io/native-vs-cross-platform-app-development-prosand-cons-49f397bb38ac.
14. Brian Kayfitz. Flutter Cookbook: Over 100 proven techniques andsolutions for app development with Flutter 2.2 and Dart, 2020. 376 с.
15. Flutter, Flutter for iOS developers [Електронний ресурс] / Flutter. ¾ Режим доступу: https://flutter.dev/docs/get-started/flutter-for/ios-devs.
16. Napoli, M. Beginning Flutter: A Hands On Guide to App Development [Text] / Varco L. Napoli. ¾ 1st edition. ¾ Birmingham: Wrox, 2019. ¾ 528 p.
17. Google Inc., Building web apps in WebView [Електронний ресурс] / Google Inc. ¾ Режим доступу:

https://developer.android.com/guide/webapps/webview.

18. Facebook, Communication between native and React Native [Електронний ресурс] / Facebook. ¾. Режим доступу: https://facebook.github.io/react-native/docs/communication-ios.

19. Google Inc., About Android App Bundles [Електронний ресурс] / Google Inc. ¾ Режим доступу: https://developer.android.com/guide/ap p-bundle.

20. Alghamdy, R. Z. (2019). The impact of mobile language learning (WhatsApp) on EFL context: Outcomes and perceptions. International Journal of English Linguistics, 9(2), 128-135. Режим доступу: https://doi.org/10.5539/ijel.v9n2p128.

21. Mike Katz, Kevin David Moore, Vincent Ngo. Flutter Apprentice (First Edition): Learn to Build Cross-Platform Apps, 2019. 154 с.

22. Alessandro Biessek. Flutter for Beginners: An introductory guide tobuilding cross-platform mobile applications with Flutter and Dart 2, 2020. 431 с.

23. Edward Thornton. Flutter For Beginners: A Genius Guide to Flutter AppDevelopment, 2020. 221 с.

24. Simone Alessandria. Flutter Projects: A practical, project-based guide tobuilding real-world cross-platform mobile applications and games, 2019. 364 с.

25. Rap Payne. Beginning App Development with Flutter: Create Cross-Platform Mobile Apps, 2018. 324 с.

26. Sheppard Dennis. Beginning Progressive Web App Development. United States of America: Apress, 2017. 286 с.

27. Dieter Meiller. Modern App Development with Dart and Flutter 2: AComprehensive Introduction to Flutter (de Gruyter Stem), 2020. 863 с.

28. Panhale Mahesh. Beginning Hybrid Mobile Application Development. United States of America: Apress, 2015. 246 с.

29. Філімончук Т.В., Чепелєв Є.О. Модель мобільного застосунку з використанням фреймворку Flutter. Проблеми інформатизації: Тезидоповідей дев'ятої міжнародної науково-технічної конференції. Том 2, секція 4. Черкаси – Харків – Баку – Бельсько-Бяла. 2021. С. 99.

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

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

32. Documentation | Android Developers [Електронний ресурс]: Developers Documentation for app developers. - Режим доступу:

https://developer.android.com/docs.

33. iOS Technology Overview. – Електрон. дан. – Режим доступу: https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/ipho neos techoverview/Introduction/Introduction.html.

34. Instruments User Guide. – Електрон. дан. – Режим доступу: https://developer.apple.com/library/mac/documentation/developertools/conceptual/instrumentsuserguide/Introduction/Introduction.html.

35. Mobile Applications for Training Plan Using Android Devices: A Systematic Review and a Taxonomy Proposal – [Електронний ресурс]. – Режим доступу:https://www.researchgate.net/publication/342641112_Mobile_Applications_for_Training_Plan_Using_Android_Devices_A_Systematic_Review_and_a_Taxonomy_ Proposal.

36. Firebase documentation [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs.

37. iOS Technology Overview. – Електрон. дан. – Режим доступу: https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/ipho neos techoverview/Introduction/Introduction.html.

38. Mark Clow. Learn Google Flutter Fast: 65 Example Apps, 2019. 134 c.

39. R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1. – The Internet Society, 1999. – 114 c.

40. App Store Resource Center. – Електрон. дан. – Режим доступу: https://developer.apple.com/appstore/index.html.

41. Roy Derks. React Projects: Build 12 real-world applications. – Chapter 12 – 2019 – P. 416.

42. Google Maps SDK for iOS. – Електрон. дан. – Режим доступу: https://developers.google.com/maps/documentation/ios/?hl=ru.

43. Mobile operating systems' market share worldwide from January 2012 to October 2020. – Електрон. дан. – Режим доступу: https://www.statista.com/statistics/272698/global-market-share-held-by-mobileoperatingsystems-since-2009/.

44. Mike Katz, Kevin David Moore, Vincent Ngo. Flutter Apprentice (First Edition): Learn to Build Cross-Platform Apps, 2019. 154 c.

45. Akshat Paul, Abhishek Nalwaya. React Native for Mobile Development. – Chapter 1 – 2019 – P. 37.

46. Alex Forrester. How to Build Android Apps with Kotlin: A hands-onguide to developing, testing, and publishing your first apps with Android. UnitedKingdom: Packt Publishing, 2021. 794 p.

47. Fu Cheng. Flutter Recipes: Mobile Development Solutions for iOS

andAndroid, 2018, 164 с.

48. Jonathan Sande, Matt Galloway. Dart Apprentice (First Edition): Beginning Programming with Dart, 2018. 364 с.

49. Порівняння фреймворків для кросплатформної мобільної розробки: React Native, Flutter, Ionic, Xamarin та PhoneGap [Електронний ресурс] – Режим доступу: https://tproger.ru/translations/crossplatform-frameworks-for-mobile-development/.

50. Hidenori Matsubayashi. Graphical User Interface Using Flutter in Embedded Systems. – Embedded Linux Conference Europe. – 2020.

51. Sean Liao. Migrating to Swift from Android. –Chapter 1 –2014 –P. 3-5.

# ЛІСТИНГ ПРОГРАМИ

Код файлу CameraScreen.dart:

```
import 'dart:math';
import 'package:camera/camera.dart';
import 'package:chatapp/Screens/CameraView.dart';
import 'package:chatapp/Screens/VideoView.dart';
import 'package:flutter/material.dart';
import 'package:path/path.dart';
import 'package:path_provider/path_provider.dart';

List<CameraDescription> cameras;

class CameraScreen extends StatefulWidget {
  CameraScreen({Key key}) : super(key: key);

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

class _CameraScreenState extends State<CameraScreen> {
  CameraController _cameraController;
  Future<void> cameraValue;
  bool isRecoring = false;
  bool flash = false;
  bool iscamerafront = true;
  double transform = 0;

  @override
  void initState() {
    super.initState();
    _cameraController = CameraController(cameras[0], ResolutionPreset.high);
    cameraValue = _cameraController.initialize();
  }

  @override
  void dispose() {
    super.dispose();
    _cameraController.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          FutureBuilder(
              future: cameraValue,
              builder: (context, snapshot) {
                if (snapshot.connectionState == ConnectionState.done) {
```

```
        return Container(
            width: MediaQuery.of(context).size.width,
            height: MediaQuery.of(context).size.height,
            child: CameraPreview(_cameraController));
        } else {
          return Center(
            child: CircularProgressIndicator(),
          );
        }
      }),
    Positioned(
      bottom: 0.0,
      child: Container(
        color: Colors.black,
        padding: EdgeInsets.only(top: 5, bottom: 5),
        width: MediaQuery.of(context).size.width,
        child: Column(
          children: [
            Row(
              mainAxisSize: MainAxisSize.max,
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                IconButton(
                  icon: Icon(
                    flash ? Icons.flash_on : Icons.flash_off,
                    color: Colors.white,
                    size: 28,
                  ),
                  onPressed: () {
                    setState(() {
                      flash = !flash;
                    });
                    flash
                        ? _cameraController
                            .setFlashMode(FlashMode.torch)
                        : _cameraController.setFlashMode(FlashMode.off);
                  }),
                GestureDetector(
                  onLongPress: () async {
                    await _cameraController.startVideoRecording();
                    setState(() {
                      isRecoring = true;
                    });
                  },
                  onLongPressUp: () async {
                    XFile videopath =
                        await _cameraController.stopVideoRecording();
                    setState(() {
                      isRecoring = false;
                    });
                    Navigator.push(
                        context,
```

```
            MaterialPageRoute(
              builder: (builder) => VideoViewPage(
                   path: videopath.path,
                 )));
      },
      onTap: () {
       if (!isRecoring) takePhoto(context);
      },
      child: isRecoring
          ? Icon(
            Icons.radio_button_on,
            color: Colors.red,
            size: 80,
           )
          : Icon(
            Icons.panorama_fish_eye,
            color: Colors.white,
            size: 70,
           ),
    ),
    IconButton(
      icon: Transform.rotate(
       angle: transform,
       child: Icon(
        Icons.flip_camera_ios,
        color: Colors.white,
        size: 28,
       ),
      ),
      onPressed: () async {
       setState(() {
        iscamerafront = !iscamerafront;
        transform = transform + pi;
       });
       int cameraPos = iscamerafront ? 0 : 1;
       _cameraController = CameraController(
          cameras[cameraPos], ResolutionPreset.high);
       cameraValue = _cameraController.initialize();
      }),
   ],
  ),
  SizedBox(
   height: 4,
  ),
  Text(
   "Hold for Video, tap for photo",
   style: TextStyle(
    color: Colors.white,
   ),
   textAlign: TextAlign.center,
  )
 ],
```

```
            ),
          ),
        ),
      ],
    ),
  );
 }
 void takePhoto(BuildContext context) async {
  XFile file = await _cameraController.takePicture();
  Navigator.push(
      context,
      MaterialPageRoute(
          builder: (builder) => CameraViewPage(
              path: file.path,
            )));
 }
}
```

Код файлу CameraView.dart:

```
import 'dart:io';
import 'package:flutter/material.dart';

class CameraViewPage extends StatelessWidget {
 const CameraViewPage({Key key, this.path}) : super(key: key);
 final String path;

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.black,
    appBar: AppBar(
      backgroundColor: Colors.black,
      actions: [
        IconButton(
            icon: Icon(
              Icons.crop_rotate,
              size: 27,
            ),
            onPressed: () {}),
        IconButton(
            icon: Icon(
              Icons.emoji_emotions_outlined,
              size: 27,
            ),
            onPressed: () {}),
        IconButton(
            icon: Icon(
              Icons.title,
              size: 27,
            ),
```

```
        onPressed: () {}),
    IconButton(
        icon: Icon(
          Icons.edit,
          size: 27,
        ),
        onPressed: () {}),
  ],
),
body: Container(
  width: MediaQuery.of(context).size.width,
  height: MediaQuery.of(context).size.height,
  child: Stack(
    children: [
      Container(
        width: MediaQuery.of(context).size.width,
        height: MediaQuery.of(context).size.height - 150,
        child: Image.file(
          File(path),
          fit: BoxFit.cover,
        ),
      ),
      Positioned(
        bottom: 0,
        child: Container(
          color: Colors.black38,
          width: MediaQuery.of(context).size.width,
          padding: EdgeInsets.symmetric(vertical: 5, horizontal: 8),
          child: TextFormField(
            style: TextStyle(
              color: Colors.white,
              fontSize: 17,
            ),
            maxLines: 6,
            minLines: 1,
            decoration: InputDecoration(
                border: InputBorder.none,
                hintText: "Add Caption....",
                prefixIcon: Icon(
                  Icons.add_photo_alternate,
                  color: Colors.white,
                  size: 27,
                ),
                hintStyle: TextStyle(
                  color: Colors.white,
                  fontSize: 17,
                ),
                suffixIcon: CircleAvatar(
                  radius: 27,
                  backgroundColor: Colors.tealAccent[700],
                  child: Icon(
                    Icons.check,
```

```
                    color: Colors.white,
                    size: 27,
                  ),
                )),
            ),
          ),
        ),
      ],
    ),
  ),
 );
 }
}
```

Код файлу CreateGroup.dart:

```
import 'package:chatapp/CustomUI/AvtarCard.dart';
import 'package:chatapp/CustomUI/ButtonCard.dart';
import 'package:chatapp/CustomUI/ContactCard.dart';
import 'package:chatapp/Model/ChatModel.dart';
import 'package:flutter/material.dart';

class CreateGroup extends StatefulWidget {
 CreateGroup({Key key}) : super(key: key);

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

class _CreateGroupState extends State<CreateGroup> {
 List<ChatModel> contacts = [
  ChatModel(name: "Dev Stack", status: "A full stack developer"),
  ChatModel(name: "Balram", status: "Flutter Developer....."),
  ChatModel(name: "Saket", status: "Web developer..."),
  ChatModel(name: "Bhanu Dev", status: "App developer...."),
  ChatModel(name: "Collins", status: "Raect developer.."),
  ChatModel(name: "Kishor", status: "Full Stack Web"),
  ChatModel(name: "Testing1", status: "Example work"),
  ChatModel(name: "Testing2", status: "Sharing is caring"),
  ChatModel(name: "Divyanshu", status: "....."),
  ChatModel(name: "Helper", status: "Love you Mom Dad"),
  ChatModel(name: "Tester", status: "I find the bugs"),
 ];
 List<ChatModel> groupmember = [];
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
     title: Column(
      mainAxisAlignment: MainAxisAlignment.start,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
```

```
      Text(
        "New Group",
        style: TextStyle(
          fontSize: 19,
          fontWeight: FontWeight.bold,
        ),
      ),
      Text(
        "Add participants",
        style: TextStyle(
          fontSize: 13,
        ),
      )
    ],
  ),
  actions: [
    IconButton(
        icon: Icon(
          Icons.search,
          size: 26,
        ),
        onPressed: () {}),
  ],
),
floatingActionButton: FloatingActionButton(
    backgroundColor: Color(0xFF128C7E),
    onPressed: () {},
    child: Icon(Icons.arrow_forward)),
body: Stack(
  children: [
    ListView.builder(
        itemCount: contacts.length + 1,
        itemBuilder: (context, index) {
          if (index == 0) {
            return Container(
              height: groupmember.length > 0 ? 90 : 10,
            );
          }
          return InkWell(
            onTap: () {
              setState(() {
                if (contacts[index - 1].select == true) {
                  groupmember.remove(contacts[index - 1]);
                  contacts[index - 1].select = false;
                } else {
                  groupmember.add(contacts[index - 1]);
                  contacts[index - 1].select = true;
                }
              });
            },
            child: ContactCard(
              contact: contacts[index - 1],
```

```
              ),
            );
          }),
      groupmember.length > 0
          ? Align(
              child: Column(
                children: [
                  Container(
                    height: 75,
                    color: Colors.white,
                    child: ListView.builder(
                        scrollDirection: Axis.horizontal,
                        itemCount: contacts.length,
                        itemBuilder: (context, index) {
                          if (contacts[index].select == true)
                            return InkWell(
                              onTap: () {
                                setState(() {
                                  groupmember.remove(contacts[index]);
                                  contacts[index].select = false;
                                });
                              },
                              child: AvatarCard(
                                chatModel: contacts[index],
                              ),
                            );
                          return Container();
                        }),
                  ),
                  Divider(
                    thickness: 1,
                  ),
                ],
              ),
              alignment: Alignment.topCenter,
            )
          : Container(),
      ],
    ));
  }
}
```

Код файлу Homescreen.dart:

```
import 'package:chatapp/Model/ChatModel.dart';
import 'package:chatapp/Pages/CameraPage.dart';
import 'package:chatapp/Pages/ChatPage.dart';
import 'package:flutter/material.dart';

class Homescreen extends StatefulWidget {
  Homescreen({Key key, this.chatmodels, this.sourchat}) : super(key: key);
  final List<ChatModel> chatmodels;
```

```
  final ChatModel sourchat;

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

class _HomescreenState extends State<Homescreen>
    with SingleTickerProviderStateMixin {
  TabController _controller;
  @override
  void initState() {
    super.initState();
    _controller = TabController(length: 4, vsync: this, initialIndex: 1);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Whatsapp Clone"),
        actions: [
          IconButton(icon: Icon(Icons.search), onPressed: () {}),
          PopupMenuButton<String>(
            onSelected: (value) {
              print(value);
            },
            itemBuilder: (BuildContext contesxt) {
              return [
                PopupMenuItem(
                  child: Text("New group"),
                  value: "New group",
                ),
                PopupMenuItem(
                  child: Text("New broadcast"),
                  value: "New broadcast",
                ),
                PopupMenuItem(
                  child: Text("Whatsapp Web"),
                  value: "Whatsapp Web",
                ),
                PopupMenuItem(
                  child: Text("Starred messages"),
                  value: "Starred messages",
                ),
                PopupMenuItem(
                  child: Text("Settings"),
                  value: "Settings",
                ),
              ];
            },
          )
        ],
```

```
      bottom: TabBar(
        controller: _controller,
        indicatorColor: Colors.white,
        tabs: [
          Tab(
            icon: Icon(Icons.camera_alt),
          ),
          Tab(
            text: "CHATS",
          ),
          Tab(
            text: "STATUS",
          ),
          Tab(
            text: "CALLS",
          )
        ],
      ),
    ),
    body: TabBarView(
      controller: _controller,
      children: [
        CameraPage(),
        ChatPage(
          chatmodels: widget.chatmodels,
          sourchat: widget.sourchat,
        ),
        Text("STATUS"),
        Text("Calls"),
      ],
    ),
  );
 }
}
```

Код файлу IndividualPage.dart:

```
// import 'package:camera/camera.dart';
// import 'package:chatapp/CustomUI/CameraUI.dart';
import 'package:chatapp/CustomUI/OwnMessgaeCrad.dart';
import 'package:chatapp/CustomUI/ReplyCard.dart';
import 'package:chatapp/Model/ChatModel.dart';
import 'package:chatapp/Model/MessageModel.dart';
import 'package:emoji_picker/emoji_picker.dart';
import 'package:flutter/material.dart';
import 'package:flutter_svg/svg.dart';
import 'package:socket_io_client/socket_io_client.dart' as IO;

class IndividualPage extends StatefulWidget {
  IndividualPage({Key key, this.chatModel, this.sourchat}) : super(key: key);
  final ChatModel chatModel;
  final ChatModel sourchat;
```

```
  @override
  _IndividualPageState createState() => _IndividualPageState();
}

class _IndividualPageState extends State<IndividualPage> {
  bool show = false;
  FocusNode focusNode = FocusNode();
  bool sendButton = false;
  List<MessageModel> messages = [];
  TextEditingController _controller = TextEditingController();
  ScrollController _scrollController = ScrollController();
  IO.Socket socket;
  @override
  void initState() {
    super.initState();
    // connect();

    focusNode.addListener(() {
      if (focusNode.hasFocus) {
        setState(() {
          show = false;
        });
      }
    });
    connect();
  }

  void connect() {
    // MessageModel messageModel = MessageModel(sourceId:
widget.sourceChat.id.toString(),targetId: );
    socket = IO.io("http://192.168.0.106:5000", <String, dynamic>{
      "transports": ["websocket"],
      "autoConnect": false,
    });
    socket.connect();
    socket.emit("signin", widget.sourchat.id);
    socket.onConnect((data) {
      print("Connected");
      socket.on("message", (msg) {
        print(msg);
        setMessage("destination", msg["message"]);
        _scrollController.animateTo(_scrollController.position.maxScrollExtent,
            duration: Duration(milliseconds: 300), curve: Curves.easeOut);
      });
    });
    print(socket.connected);
  }

  void sendMessage(String message, int sourceId, int targetId) {
    setMessage("source", message);
    socket.emit("message",
```

```
    {"message": message, "sourceId": sourceId, "targetId": targetId});
}

void setMessage(String type, String message) {
  MessageModel messageModel = MessageModel(
      type: type,
      message: message,
      time: DateTime.now().toString().substring(10, 16));
  print(messages);

  setState(() {
    messages.add(messageModel);
  });
}

@override
Widget build(BuildContext context) {
  return Stack(
    children: [
      Image.asset(
        "assets/whatsapp_Back.png",
        height: MediaQuery.of(context).size.height,
        width: MediaQuery.of(context).size.width,
        fit: BoxFit.cover,
      ),
      Scaffold(
        backgroundColor: Colors.transparent,
        appBar: PreferredSize(
          preferredSize: Size.fromHeight(60),
          child: AppBar(
            leadingWidth: 70,
            titleSpacing: 0,
            leading: InkWell(
              onTap: () {
                Navigator.pop(context);
              },
              child: Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Icon(
                    Icons.arrow_back,
                    size: 24,
                  ),
                  CircleAvatar(
                    child: SvgPicture.asset(
                      widget.chatModel.isGroup
                          ? "assets/groups.svg"
                          : "assets/person.svg",
                      color: Colors.white,
                      height: 36,
                      width: 36,
                    ),
```

```
        radius: 20,
        backgroundColor: Colors.blueGrey,
      ),
    ],
  ),
),
title: InkWell(
  onTap: () {},
  child: Container(
    margin: EdgeInsets.all(6),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.start,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          widget.chatModel.name,
          style: TextStyle(
            fontSize: 18.5,
            fontWeight: FontWeight.bold,
          ),
        ),
        Text(
          "last seen today at 12:05",
          style: TextStyle(
            fontSize: 13,
          ),
        )
      ],
    ),
  ),
),
actions: [
  IconButton(icon: Icon(Icons.videocam), onPressed: () {}),
  IconButton(icon: Icon(Icons.call), onPressed: () {}),
  PopupMenuButton<String>(
    padding: EdgeInsets.all(0),
    onSelected: (value) {
      print(value);
    },
    itemBuilder: (BuildContext contesxt) {
      return [
        PopupMenuItem(
          child: Text("View Contact"),
          value: "View Contact",
        ),
        PopupMenuItem(
          child: Text("Media, links, and docs"),
          value: "Media, links, and docs",
        ),
        PopupMenuItem(
          child: Text("Whatsapp Web"),
          value: "Whatsapp Web",
```

```
            ),
            PopupMenuItem(
              child: Text("Search"),
              value: "Search",
            ),
            PopupMenuItem(
              child: Text("Mute Notification"),
              value: "Mute Notification",
            ),
            PopupMenuItem(
              child: Text("Wallpaper"),
              value: "Wallpaper",
            ),
          ];
        },
      ),
    ],
  ),
),
body: Container(
  height: MediaQuery.of(context).size.height,
  width: MediaQuery.of(context).size.width,
  child: WillPopScope(
    child: Column(
      children: [
        Expanded(
          // height: MediaQuery.of(context).size.height - 150,
          child: ListView.builder(
            shrinkWrap: true,
            controller: _scrollController,
            itemCount: messages.length + 1,
            itemBuilder: (context, index) {
              if (index == messages.length) {
                return Container(
                  height: 70,
                );
              }
              if (messages[index].type == "source") {
                return OwnMessageCard(
                  message: messages[index].message,
                  time: messages[index].time,
                );
              } else {
                return ReplyCard(
                  message: messages[index].message,
                  time: messages[index].time,
                );
              }
            },
          ),
        ),
        Align(
```

```
alignment: Alignment.bottomCenter,
child: Container(
  height: 70,
  child: Column(
    mainAxisAlignment: MainAxisAlignment.end,
    children: [
      Row(
        children: [
          Container(
            width: MediaQuery.of(context).size.width - 60,
            child: Card(
              margin: EdgeInsets.only(
                  left: 2, right: 2, bottom: 8),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(25),
              ),
              child: TextFormField(
                controller: _controller,
                focusNode: focusNode,
                textAlignVertical: TextAlignVertical.center,
                keyboardType: TextInputType.multiline,
                maxLines: 5,
                minLines: 1,
                onChanged: (value) {
                  if (value.length > 0) {
                    setState(() {
                      sendButton = true;
                    });
                  } else {
                    setState(() {
                      sendButton = false;
                    });
                  }
                },
                decoration: InputDecoration(
                  border: InputBorder.none,
                  hintText: "Type a message",
                  hintStyle: TextStyle(color: Colors.grey),
                  prefixIcon: IconButton(
                    icon: Icon(
                      show
                          ? Icons.keyboard
                          : Icons.emoji_emotions_outlined,
                    ),
                    onPressed: () {
                      if (!show) {
                        focusNode.unfocus();
                        focusNode.canRequestFocus = false;
                      }
                      setState(() {
                        show = !show;
                      });
```

```
          },
        ),
        suffixIcon: Row(
          mainAxisSize: MainAxisSize.min,
          children: [
            IconButton(
              icon: Icon(Icons.attach_file),
              onPressed: () {
                showModalBottomSheet(
                    backgroundColor:
                        Colors.transparent,
                    context: context,
                    builder: (builder) =>
                        bottomSheet());
              },
            ),
            IconButton(
              icon: Icon(Icons.camera_alt),
              onPressed: () {
                // Navigator.push(
                //     context,
                //     MaterialPageRoute(
                //         builder: (builder) =>
                //             CameraApp()));
              },
            ),
          ],
        ),
        contentPadding: EdgeInsets.all(5),
      ),
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(
    bottom: 8,
    right: 2,
    left: 2,
  ),
  child: CircleAvatar(
    radius: 25,
    backgroundColor: Color(0xFF128C7E),
    child: IconButton(
      icon: Icon(
        sendButton ? Icons.send : Icons.mic,
        color: Colors.white,
      ),
      onPressed: () {
        if (sendButton) {
          _scrollController.animateTo(
              _scrollController
                  .position.maxScrollExtent,
```

```
                                duration:
                                    Duration(milliseconds: 300),
                                curve: Curves.easeOut);
                            sendMessage(
                                _controller.text,
                                widget.sourchat.id,
                                widget.chatModel.id);
                            _controller.clear();
                            setState(() {
                              sendButton = false;
                            });
                          }
                        },
                      ),
                    ),
                  ),
                ],
              ),
              show ? emojiSelect() : Container(),
            ],
          ),
        ),
      ),
      onWillPop: () {
        if (show) {
          setState(() {
            show = false;
          });
        } else {
          Navigator.pop(context);
        }
        return Future.value(false);
      },
    ),
  ),
 ],
 );
}

Widget bottomSheet() {
 return Container(
  height: 278,
  width: MediaQuery.of(context).size.width,
  child: Card(
   margin: const EdgeInsets.all(18.0),
   shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(15)),
   child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 10, vertical: 20),
    child: Column(
```

```
          children: [
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                iconCreation(
                    Icons.insert_drive_file, Colors.indigo, "Document"),
                SizedBox(
                  width: 40,
                ),
                iconCreation(Icons.camera_alt, Colors.pink, "Camera"),
                SizedBox(
                  width: 40,
                ),
                iconCreation(Icons.insert_photo, Colors.purple, "Gallery"),
              ],
            ),
            SizedBox(
              height: 30,
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                iconCreation(Icons.headset, Colors.orange, "Audio"),
                SizedBox(
                  width: 40,
                ),
                iconCreation(Icons.location_pin, Colors.teal, "Location"),
                SizedBox(
                  width: 40,
                ),
                iconCreation(Icons.person, Colors.blue, "Contact"),
              ],
            ),
          ],
        ),
      ),
    ),
  );
}

Widget iconCreation(IconData icons, Color color, String text) {
  return InkWell(
    onTap: () {},
    child: Column(
      children: [
        CircleAvatar(
          radius: 30,
          backgroundColor: color,
          child: Icon(
            icons,
            // semanticLabel: "Help",
            size: 29,
```

```
            color: Colors.white,
          ),
        ),
        SizedBox(
          height: 5,
        ),
        Text(
          text,
          style: TextStyle(
            fontSize: 12,
            // fontWeight: FontWeight.w100,
          ),
        )
      ],
    ),
  );
 }

 Widget emojiSelect() {
  return EmojiPicker(
      rows: 4,
      columns: 7,
      onEmojiSelected: (emoji, category) {
        print(emoji);
        setState(() {
          _controller.text = _controller.text + emoji.emoji;
        });
      });
 }
}
```

Код файлу SelectContact.dart:

```
import 'package:chatapp/CustomUI/ButtonCard.dart';
import 'package:chatapp/CustomUI/ContactCard.dart';
import 'package:chatapp/Model/ChatModel.dart';
import 'package:chatapp/Screens/CreateGroup.dart';
import 'package:flutter/material.dart';

class SelectContact extends StatefulWidget {
 SelectContact({Key key}) : super(key: key);

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

class _SelectContactState extends State<SelectContact> {
 @override
 Widget build(BuildContext context) {
  List<ChatModel> contacts = [
    ChatModel(name: "Dev Stack", status: "A full stack developer"),
    ChatModel(name: "Elizabeth", status: "Flutter Developer....."),
```

```
  ChatModel(name: "Pavel", status: "Web developer..."),
  ChatModel(name: "Mobile Dev", status: "App developer...."),
  ChatModel(name: "Collins", status: "Raect developer.."),
  ChatModel(name: "Kishor", status: "Full Stack Web"),
  ChatModel(name: "Testing1", status: "Example work"),
  ChatModel(name: "Testing2", status: "Sharing is caring"),
  ChatModel(name: "Divyanshu", status: "....."),
  ChatModel(name: "Helper", status: "Love you Mom Dad"),
  ChatModel(name: "Tester", status: "I find the bugs"),
];

return Scaffold(
    appBar: AppBar(
      title: Column(
        mainAxisAlignment: MainAxisAlignment.start,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            "Select Contact",
            style: TextStyle(
              fontSize: 19,
              fontWeight: FontWeight.bold,
            ),
          ),
          Text(
            "256 contacts",
            style: TextStyle(
              fontSize: 13,
            ),
          )
        ],
      ),
      actions: [
        IconButton(
            icon: Icon(
              Icons.search,
              size: 26,
            ),
            onPressed: () {}),
        PopupMenuButton<String>(
          padding: EdgeInsets.all(0),
          onSelected: (value) {
            print(value);
          },
          itemBuilder: (BuildContext contesxt) {
            return [
              PopupMenuItem(
                child: Text("Invite a friend"),
                value: "Invite a friend",
              ),
              PopupMenuItem(
                child: Text("Contacts"),
```

```
                value: "Contacts",
              ),
              PopupMenuItem(
                child: Text("Refresh"),
                value: "Refresh",
              ),
              PopupMenuItem(
                child: Text("Help"),
                value: "Help",
              ),
            ];
          },
        ),
      ],
    ),
    body: ListView.builder(
      itemCount: contacts.length + 2,
      itemBuilder: (context, index) {
        if (index == 0) {
          return InkWell(
            onTap: () {
              Navigator.push(context,
                  MaterialPageRoute(builder: (builder) => CreateGroup()));
            },
            child: ButtonCard(
              icon: Icons.group,
              name: "New group",
            ),
          );
        } else if (index == 1) {
          return ButtonCard(
            icon: Icons.person_add,
            name: "New contact",
          );
        }
        return ContactCard(
          contact: contacts[index - 2],
        );
      }));
  }
}
```

Код файлу VideoView.dart:

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:video_player/video_player.dart';

class VideoViewPage extends StatefulWidget {
  const VideoViewPage({Key key, this.path}) : super(key: key);
  final String path;
```

```
  @override
  _VideoViewPageState createState() => _VideoViewPageState();
}

class _VideoViewPageState extends State<VideoViewPage> {
  VideoPlayerController _controller;

  @override
  void initState() {
    super.initState();
    _controller = VideoPlayerController.file(File(widget.path))
      ..initialize().then((_) {
        // Ensure the first frame is shown after the video is initialized, even before the play button has been pressed.
        setState(() {});
      });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.black,
      appBar: AppBar(
        backgroundColor: Colors.black,
        actions: [
          IconButton(
            icon: Icon(
              Icons.crop_rotate,
              size: 27,
            ),
            onPressed: () {}),
          IconButton(
            icon: Icon(
              Icons.emoji_emotions_outlined,
              size: 27,
            ),
            onPressed: () {}),
          IconButton(
            icon: Icon(
              Icons.title,
              size: 27,
            ),
            onPressed: () {}),
          IconButton(
            icon: Icon(
              Icons.edit,
              size: 27,
            ),
            onPressed: () {}),
        ],
      ),
      body: Container(
```

```
width: MediaQuery.of(context).size.width,
height: MediaQuery.of(context).size.height,
child: Stack(
 children: [
  Container(
   width: MediaQuery.of(context).size.width,
   height: MediaQuery.of(context).size.height - 150,
   child: _controller.value.initialized
       ? AspectRatio(
          aspectRatio: _controller.value.aspectRatio,
          child: VideoPlayer(_controller),
         )
       : Container(),
  ),
  Positioned(
   bottom: 0,
   child: Container(
    color: Colors.black38,
    width: MediaQuery.of(context).size.width,
    padding: EdgeInsets.symmetric(vertical: 5, horizontal: 8),
    child: TextFormField(
     style: TextStyle(
      color: Colors.white,
      fontSize: 17,
     ),
     maxLines: 6,
     minLines: 1,
     decoration: InputDecoration(
        border: InputBorder.none,
        hintText: "Add Caption....",
        prefixIcon: Icon(
         Icons.add_photo_alternate,
         color: Colors.white,
         size: 27,
        ),
        hintStyle: TextStyle(
         color: Colors.white,
         fontSize: 17,
        ),
        suffixIcon: CircleAvatar(
         radius: 27,
         backgroundColor: Colors.tealAccent[700],
         child: Icon(
          Icons.check,
          color: Colors.white,
          size: 27,
         ),
        )),
    ),
   ),
  ),
  Align(
```

```
                alignment: Alignment.center,
                child: InkWell(
                  onTap: () {
                    setState(() {
                      _controller.value.isPlaying
                          ? _controller.pause()
                          : _controller.play();
                    });
                  },
                  child: CircleAvatar(
                    radius: 33,
                    backgroundColor: Colors.black38,
                    child: Icon(
                      _controller.value.isPlaying
                          ? Icons.pause
                          : Icons.play_arrow,
                      color: Colors.white,
                      size: 50,
                    ),
                  ),
                ),
              ),
            ],
          ),
        ),
      );
    }
}
```

# ВІДГУК КЕРІВНИКА

## НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ «ДНІПРОВСЬКА ПОЛІТЕХНІКА»

**Факультет інформаційних технологій**
**Кафедра програмного забезпечення комп'ютерних систем**

## ВІДГУК

Наукового керівника Мещерякова Леоніда Івановича, д.т.н., проф. каф. ПЗКС
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

**на магістерську роботу**

студента Попової Єлизавети Сергіївни
(прізвище, ім'я, по батькові)

курсу II групи 122м-21-1
спеціальності 122 Комп'ютерні науки
освітньої програми Комп'ютерні науки
на тему Дослідження ефективності фреймворку Flutter при розробці високодинамічних мобільних додатків

Актуальність теми Представлена магістерська кваліфікаційна робота присвячена дослідженню вирішення проблеми оптимізації розробки мобільних додатків під платформи Android та iOS за допомогою фреймворку Flutter. Даний фреймворк все частіше використовується для створення крос-платформних мобільних додатків. На сьогоднішній момент мобільна розробка є однією з передових галузей розвитку інформаційних технологій, тому з огляду на це, магістерська робота характеризується актуальністю та своєчасністю.

Мета досліджень Полягає в оптимізації розробки мобільних додатків на таких платформах, як: Android та iOS, за рахунок декларативного підходу до побудови мобільних інтерфейсів фреймворку Flutter, на замовлення бізнесу.

Коротка характеристика розділів роботи Перший розділ роботи присвячений огляду проблеми, що вирішується в ході виконння роботи, а також представлені можливих шляхів вирішення цієї проблеми. Другий розділ містить аналітичний огляд на архітектуру, композицію та передачу даних фреймворків React Native та

Flutter. В третьому розділі був розроблений мобільний додаток для визначення ефективності фремворку Flutter при розробці високодинамічних мобільних додатків.

Практичне значення роботи полягає в тому, що розроблену систему можна використовувати для створення платформоорієнтованих високодинамічних користувацьких додатків із спільною кодовою базою, яка дозволяє: зробити розроблення користувацьких додатків більш гнучким; скоротити витрати грошових та управлінських ресурсів на велику команду розробників. Через використання Flutter в якості базової платформи, за допомогою розробленої системи можна створити додаток, який буде зібрано більше ніж на п'яти платформах.

Зауваження та недоліки

Висновки та оцінка Магістром було проведено аналіз та порівняння можливих методів розв’язання поставленої задачі та обрано оптимальний варіант. Під час виконання магістерської кваліфікаційної роботи студентка Попова Є.С. проявила себе грамотним, кваліфікованим спеціалістом здатним приймати самостійно складні технічні рішення. Вважаю, що магістерська кваліфікаційна робота заслуговує оцінку «відмінно», а Попова Є.С. – присвоєння кваліфікації «магістра» з комп’ютерних наук.

Науковий керівник Мещеряков Л.І., док. техн. наук, проф., проф. каф. ПЗКС
(прізвище, ім’я, по батькові, посада, місце роботи)

«___»_____20__ р. _____
(підпис)

**РЕЦЕНЗІЯ**
**на кваліфікаційну роботу**

студента Попової Єлизавети Сергіївни
(прізвище, ім'я, по батькові)

курсу II групи 122м-21-1
кафедри програмного забезпечення комп'ютерних систем
спеціальності 122 Комп'ютерні науки
Тема роботи Дослідження ефективності фреймворку Flutter при розробці високодинамічних мобільних додатків

Стисла характеристика розділів роботи Перший розділ роботи присвячений огляду проблеми, що вирішується в ході виконання роботи, а також представленні можливих шляхів вирішення цієї проблеми. Других розділ містить аналітичний огляд на архітектуру, композицію та передачу даних фреймворків React Native та Flutter. У третьому розділі був розроблений мобільний додаток задля визначення ефективності фремворку Flutter при розробці високодинамічних мобільних додатків.

Пропозиції, внесені студентом, рівень їх наукового обґрунтування В даній кваліфікаційній роботі студентом надано декілька пропозицій щодо вирішення поставлених задач. Кожна з пропозицій була обґрунтована та підкріплена науковими даними.

Практичне значення роботи полягає в тому, що розроблену систему можна використовувати для створення платформоорієнтованих високодинамічних користувацьких додатків із спільною кодовою базою, яка дозволяє: зробити розроблення користувацьких додатків більш гнучким; скоротити витрати грошових та управлінських ресурсів на велику команду розробників. Через використання Flutter в якості базової платформи, за допомогою розробленої системи можна створити додаток, який буде зібрано більше ніж на п'яти платформах.

Якість оформлення роботи Магістерська кваліфікаційна робота, яку подано на рецензію, виконана у повному обсязі у встановлений термін. Робота є добре структурованою та достатньо проілюстрованою. Викладена основна суть проблеми, що вирішується в ході виконання роботи, і шляхів її вирішення.

Недоліки в роботі відсутність авторизації користувачів мобільного додатку.

Проте вказаний недолік не впливає на позитивне враження від роботи.

Загальний висновок Магістерська кваліфікаційна робота виконана у
(підготовленість студента до самостійної роботи як спеціаліста)
відповідності з завданням із дотриманням всіх вимог.

Оцінка магістерської роботи Робота заслуговує оцінки «відмінно», а студент Попова Є.С. – присвоєння кваліфікації «магістра» з комп'ютерних наук.

Рецензент _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

«___»_____20__ р. _____
(підпис)

## ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

| **Ім’я файла** | **Опис** |
|---|---|
| Пояснювальні документи | |
| Диплом_Попова.doc | Пояснювальна записка роботи. Документ Word. |
| Диплом_Попова.pdf | Пояснювальна записка роботи в форматі PDF |
| Програма | |
| Program.rar | Архів. Містить коди програми і откомпільовану програму |
| Презентація | |
| Презентація_Попова.ppt | Презентація роботи |