ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення автоматизованих систем

Пояснювальна записка

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: **«Розробка мобільного додатку для планування часу на платформі Flutter»**

Виконав: студент 4 курсу, групи ІСД–42

спеціальності 126 Інформаційні системи та технології

Новіков І. І.
(прізвище та ініціали)

Керівник Казначеєва А. В.
Рецензент _____

Київ – 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти - «Бакалавр»
Напрям підготовки – 126 – Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення автоматизованих систем

_____К.П. Сторчак

«____» _____2024 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

_____Новіков Ігор Іванович_____

1.Тема роботи: Розробка мобільного додатку для планування часу на платформі Flutter
Керівник роботи Казначеєва Анастасія Василівна,
затверджені наказом вищого навчального закладу від 27 лютого 2024 року №36.
2. Строк подання студентом роботи 20 травня 2024 року
3. Вхідні дані до роботи:
Інформаційно-ентропійний метод;
Класифікація мобільних додатків
Документація Flutter та Dart
Архітектура мобільних додатків MVP
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
4.1 Аналіз сучасних технологій розробки мобільних додатків
4.2 Визначення ефективності фреймворку Flutter для мобільної розробки
4.3. Огляд створеного графічного інтерфейсу мобільного додатку
4.4. Опис розробки та основних функцій програмного забезпечення додатку
5. Перелік графічного матеріалу
1.Титульний слайд
2. Об’єкт, предмет, мета дослідження
3. Огляд класифікації мобільних додатків та технологій їх створення

4. Порівняльне дослідження технологій мобільної розробки (в т. ч. Flutter)
5. Опис графічного інтерфейсу створеного мобільного додатку
6. Опис архітектури і основних функцій програмного забезпечення додатку
8. Висновки
9. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	10.03.24	Виконано
2	Дослідження сучасного стану сонячної електроенергетики	13.03.24	Виконано
3	Вивчення технологічної бази розробки пристрою	17.03.24	Виконано
4	Розробка концепції апаратної частини пристрою	25.03.24	Виконано
5	Розробка програмного забезпечення пристрою	02.04.24	Виконано
6	Написання основної частини кваліфікаційної роботи	10.04.24	Виконано
7	Написання вступу, висновків та реферату	16.04.24	Виконано
8	Розробка демонстраційних матеріалів	18.04.24	Виконано
9	Попередній захист роботи	19.04.24	Виконано
10	Подання роботи в деканат	20.05.24	Виконано

Студент _____ Новіков І. І.

Керівник роботи _____ Казначеєва А. В.

РЕФЕРАТ

Текстова частина бакалаврської роботи: 75 с., 66 рис., 27 джерел.
МОБІЛЬНИЙ ДОДАТОК, ПЛАНУВАННЯ ЧАСУ, FLUTTER, FIREBASE FIRESTORE, АРХІТЕКТУРА ДОДАТКУ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, СИНХРОНІЗАЦІЯ ДАНИХ, УПРАВЛІННЯ ЗАДАЧАМИ
Об'єкт дослідження – процес розробки мобільних додатків для ефективного планування часу.
Предмет дослідження – способи використання платформи Flutter та Firebase Firestore для створення мобільного додатку, який забезпечує зручне планування задач і управління часом.
Мета роботи – дослідження сучасних технологій розробки мобільних додатків, розробка концепції мобільного додатку для планування часу, що включає зручний користувацький інтерфейс і ефективний обмін даними.
Методи дослідження – методи програмної інженерії, методи розробки мобільних додатків з використанням Flutter, методи роботи з хмарною базою даних Firebase Firestore.
Визначено стан сучасної розробки мобільних додатків для планування часу та вплив використання кросплатформених інструментів на ефективність розробки і зручність користування додатками.
Здійснено порівняння різних платформ для розробки мобільних додатків, виявлено переваги використання Flutter і Firebase Firestore для створення додатків з реальним часом синхронізації даних.
На основі результатів виконаних досліджень розроблено мобільний додаток для планування часу з енергомодулями на основі Flutter і Firebase Firestore.
Упровадження розробленого додатку дозволяє підвищити ефективність управління часом користувачів, забезпечує зручний інтерфейс для планування задач, а також сприяє розвитку практичних навичок у студентів при роботі з сучасними технологіями розробки мобільних додатків.

ЗМІСТ

РЕФЕРАТ 4

ВСТУП 6

1 ТЕХНОЛОГІЧНІ АСПЕКТИ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ 7

1.1 Класифікація та особливості застосування мобільних додатків 7

1.2 Етапи проєктування і створення мобільних додатків 14

1.3 Принципи та інструменти побудови користувацького інтерфейсу 20

1.4 Архітектура мобільних додатків 25

2 ОСОБЛИВОСТІ ВИКОРИСТАННЯ ФРЕЙМВОРКУ FLUTTER ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ 31

2.1 Ключові технології та підходи фреймворку Flutter 31

2.2 Способи паралельної розробки користувацького інтерфейсу і програмного забезпечення 39

2.3 Аналіз ефективності використання Flutter для створення мобільного додатку 41

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПЛАНУВАННЯ ЧАСУ У ФРЕЙМВОРКУ FLUTTER 49

3.1 Порівняння існуючих рішень для тайм-менеджменту 49

3.2 Користувацький інтерфейс додатку 52

3.3 Програмне забезпечення додатку 59

ВИСНОВОК 67

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 68

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 70

ВСТУП

Вибір теми кваліфікаційної роботи був зумовлений необхідністю розробки ефективних інструментів для планування часу, зручності управління завданнями та підвищення продуктивності користувачів. Створення мобільного додатку для планування часу є актуальним завданням, оскільки сучасний світ вимагає від людей ефективного управління своїм часом. Використання кросплатформних інструментів, таких як Flutter, дозволяє значно спростити процес розробки та забезпечити високу продуктивність і зручність використання додатку.

Мобільний додаток для планування часу має на меті забезпечити користувачів інструментами для ефективного управління своїми задачами та часом. Використання платформи Flutter та Firebase Firestore забезпечує швидку розробку, високу продуктивність та реальне синхронізацію даних, що дозволяє користувачам завжди мати актуальну інформацію про свої задачі.

Об’єктом дослідження є процес розробки мобільних додатків для ефективного планування часу. Предметом дослідження є способи використання платформи Flutter та Firebase Firestore для створення мобільного додатку, який забезпечує зручне планування задач і управління часом. Метою роботи є дослідження сучасних технологій розробки мобільних додатків, розробка концепції мобільного додатку для планування часу, що включає зручний користувацький інтерфейс і ефективний обмін даними.

Відповідно до мети кваліфікаційної роботи було поставлено наступні задачі:

1. Дослідити сучасні технології розробки мобільних додатків та обрати найкращі інструменти для створення додатку для планування часу.
2. Вивчити особливості використання платформи Flutter для розробки кросплатформних додатків.
3. Ознайомитися з можливостями Firebase Firestore для забезпечення реальної синхронізації даних.
4. Розробити концепцію мобільного додатку для планування часу, що включає зручний користувацький інтерфейс і ефективний обмін даними.

Методика дослідження складається з двох основних етапів – теоретичного і практичного. Спочатку вивчається проблема, проводиться пошук її рішень, пошук шляхів виконання завдань. Проводиться дослідження і теоретичне обґрунтування матеріалів та на основі отриманих даних розробляється план виконання практичної частини. Практичний етап включає в себе безпосередньо розробку додатку – його інтерфейсу, логіки роботи та налаштування обміну даними з базою даних.

Наукова новизна дослідження полягає в тому, що дослідження і розробка додатку для планування часу з використанням Flutter та Firebase Firestore до цього часу не були достатньо досліджені, і задача цього дослідження полягає в створенні такого додатку з пропозиціями щодо подальшого вдосконалення.

Практична значущість результатів дослідження полягає в можливості використовувати розроблений додаток для підвищення ефективності управління часом користувачів, забезпечення зручного інтерфейсу для планування задач, а також сприяння розвитку практичних навичок у студентів при роботі з сучасними технологіями розробки мобільних додатків.

1 ТЕХНОЛОГІЧНІ АСПЕКТИ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

1.1 Класифікація та особливості застосування мобільних додатків

Мобільний додаток – це програмне забезпечення, розроблене для роботи на мобільних пристроях, таких як смартфони та планшети. З розвитком технологій мобільні додатки стали невід'ємною частиною повсякденного життя, забезпечуючи користувачам широкий спектр функціональних можливостей від спілкування до управління бізнесом.

Мобільні додатки можна розділити на кілька категорій за їх призначенням (рис. 1.1). До соціальних мереж та комунікацій належать такі додатки, як Facebook, WhatsApp, Instagram, які забезпечують взаємодію користувачів через обмін повідомленнями, фото та відео. Вони є платформами для соціальної взаємодії та мережевого спілкування. До категорії ігор входять додатки на зразок Candy Crush, PUBG Mobile, Among Us, що характеризуються високим рівнем інтерактивності, привабливою графікою та захоплюючим геймплеєм. Бізнес та продуктивність охоплюють додатки, такі як Microsoft Office, Trello, Slack, які допомагають користувачам управляти завданнями, документами та командною роботою. Вони спрямовані на підвищення ефективності роботи та організації бізнес-процесів. Освітні додатки, як-от Duolingo, Khan Academy, Coursera, надають користувачам доступ до курсів, навчальних матеріалів та інтерактивних завдань, сприяючи саморозвитку та навчальному процесу. До додатків для здоров’я та фітнесу належать MyFitnessPal, Calm, Headspace, що допомагають користувачам стежити за своїм здоров’ям, займатися фізичними вправами та медитацією. Магазини та покупки включають додатки на зразок Amazon, eBay, AliExpress, які забезпечують можливість здійснення покупок онлайн, включаючи оплату товарів та послуг.

Рисунок 1.1 – Різноманіття мобільних додатків [1]

Мобільні додатки також класифікуються за технологією створення (рис. 1.2., рис. 1.3). Нативні додатки розробляються для конкретної платформи (iOS або Android) з використанням відповідних мов програмування (Swift для iOS, Kotlin або Java для Android). Переваги включають високу продуктивність та доступ до всіх функцій пристрою, тоді як недоліками є необхідність окремої розробки для кожної платформи. Кросплатформні додатки використовують фреймворки типу React Native, Flutter, Xamarin для створення додатків, які працюють на різних платформах. Вони дозволяють зекономити час та кошти на розробку, хоча можуть мати компроміси в продуктивності та доступі до специфічних функцій. Веб-додатки працюють через браузер та адаптуються під мобільні пристрої (PWA - прогресивні веб-додатки). Переваги включають кросплатформність та легкість оновлення, тоді як недоліками є обмежений доступ до апаратних ресурсів пристрою та потреба в інтернет-з'єднанні.

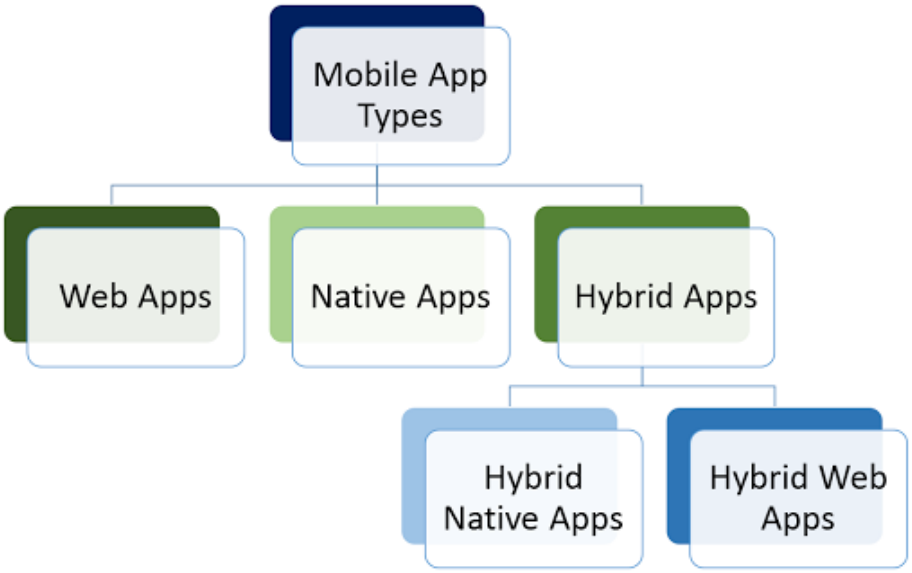


Рисунок 1.2 – Класифікація мобільних додатків за технологією створення [3]

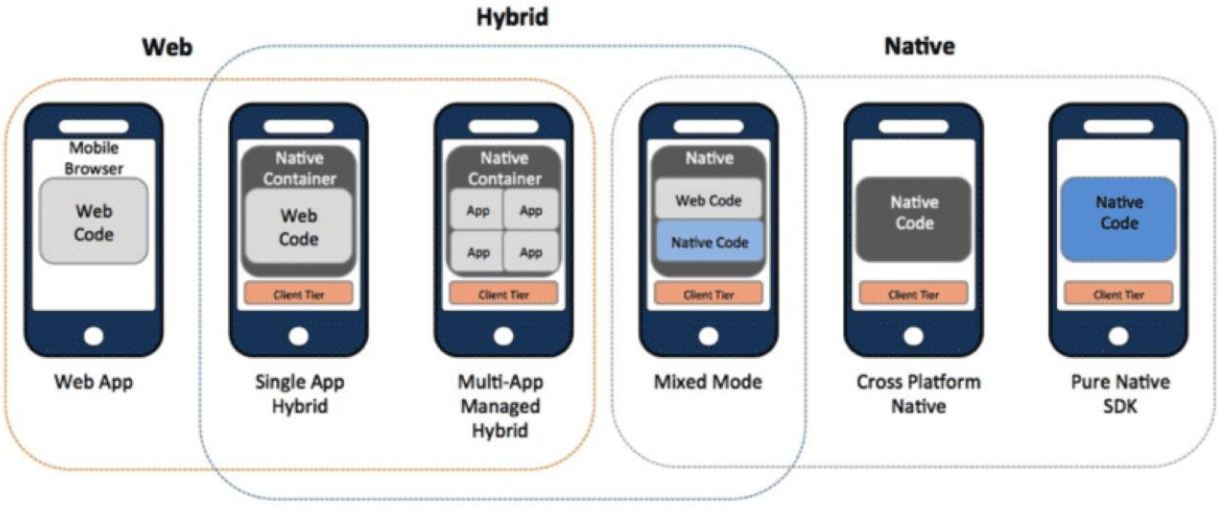


Рисунок 1.3 - Класифікація мобільних додатків за технологією створення [4]

Нативні додатки розроблені спеціально для певної платформи і можуть використовувати весь спектр можливостей пристрою – камеру, GPS-датчик, акселерометр, компас, список контактів і так далі (рис. 1.4). Вони можуть розпізнавати стандартні жести, встановлені в операційній системі за замовчуванням, або використовувати абсолютно нові жести, унікальні для конкретного рішення. Варто сказати, що багато нативних програм можуть працювати без підключення до Інтернету. [5]

Рисунок 1.4 – Ключові особливості нативних додатків [2]

Оскільки нативні додатки оптимізовані для конкретної ОС, вони органічно вписуються в будь-який смартфон, а це означає, що їх швидкість і стабільність продуктивності залишаються незмінними незалежно від пристрою.

Нативні додатки можуть мати доступ до системи сповіщень пристрою та, залежно від призначення нативної програми, можуть повністю або частково працювати без підключення до Інтернету.

Мобільний веб-додаток – це, по суті, веб-сайт, адаптований та оптимізований для будь-якого смартфона (рис. 1.5). Єдине, що потрібно використовувати, це браузер, а також підключення до Інтернету. [5]

Рисунок 1.5 – Ключові особливості мобільних веб-додатків [2]

Є кілька технологій, які можна використовувати для розробки мобільних веб-додатків. Найпоширеніші технології включають HTML/CSS, JavaScript і такі фреймворки, як Ionic, React Native, Flutter та інші.

Під час роботи з такими рішеннями користувачі виконують усі дії, які вони виконують при переході на будь-який сайт. Вони також отримують можливість «встановлювати» ці програми на свій робочий стіл, створюючи закладку веб-сторінки.

Мобільні веб-додатки є кросплатформними, тобто можуть працювати незалежно від пристрою. Їх особливість у тому, що вони не використовують програмне забезпечення для конкретного пристрою. Більше того, оскільки вони є мобільною версією сайту з розширеними функціями взаємодії, веб-додатки не займають місця на смартфоні.

Веб-додатки набули широкого розповсюдження приблизно в той час, коли HTML5 почав розвиватися, і можливість отримати доступ до більшості нативних функцій програми через звичайний браузер стала. Важко точно сказати, де пролягає чітка межа між мобільними веб-додатками та звичайними веб-сторінками, оскільки HTML5 з кожним днем стає все більш досконалим, і все більше сайтів вирішують використовувати його в повній мірі.

Вибір між нативним та веб-додатком для мобільних пристроїв є важливим рішенням, що впливає на кінцевий продукт, його продуктивність та

користувацький досвід (рис. 1.6). Нативні додатки розробляються спеціально для конкретної платформи, такої як iOS або Android, використовуючи відповідні мови програмування (Swift для iOS, Kotlin або Java для Android). Веб-додатки, з іншого боку, є веб-сайтами, оптимізованими для мобільних пристроїв, які працюють через браузер і не залежать від операційної системи пристрою.

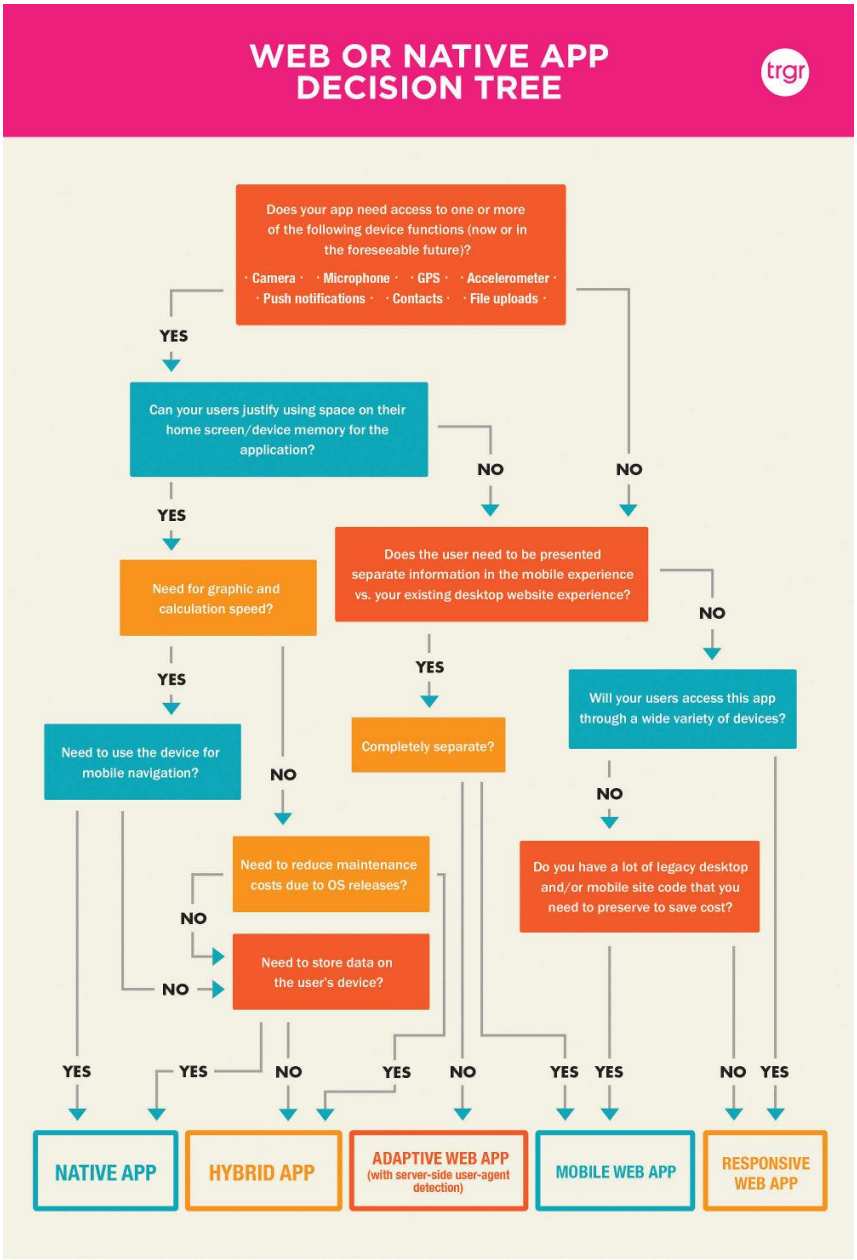


Рисунок 1.6 – Дерево рішень при виборі між нативним та веб-додатком [3]

Однією з головних переваг нативних додатків є висока продуктивність та доступ до всіх функцій пристрою, таких як камера, GPS, сповіщення та сенсори. Це дозволяє створювати більш інтерактивні та багатофункціональні додатки. Крім того, нативні додатки забезпечують кращий користувацький досвід (UX) завдяки інтуїтивно зрозумілому інтерфейсу та плавній роботі. Важливо також зазначити,

що нативні додатки можуть працювати офлайн, що є великою перевагою для користувачів, які мають обмежений доступ до Інтернету.

Однак, розробка нативних додатків має свої недоліки. Основним з них є висока вартість та тривалість розробки, оскільки додатки для різних платформ (iOS та Android) потребують окремого кодування, тестування та підтримки. Це вимагає залучення розробників з різними технічними навичками, що збільшує витрати та час на розробку.

Веб-додатки мають свої переваги, зокрема кросплатформність. Вони працюють на будь-якому пристрої з веб-браузером, що значно знижує витрати на розробку та підтримку. Веб-додатки легко оновлюються, оскільки зміни відбуваються на сервері, і користувачам не потрібно завантажувати та встановлювати оновлення. Це забезпечує швидкий розгортання нових функцій та виправлень.

Однак, веб-додатки мають обмежений доступ до функцій пристрою і не можуть повною мірою використовувати можливості апаратного забезпечення, що впливає на продуктивність і користувацький досвід. Крім того, веб-додатки потребують постійного інтернет-з’єднання для коректної роботи, що може бути незручним для користувачів з нестабільним або обмеженим доступом до Інтернету.

Вибір між нативним та веб-додатком залежить від конкретних потреб та цілей проєкту. Якщо додаток вимагає високої продуктивності, інтеграції з апаратними можливостями пристрою та забезпечення найкращого користувацького досвіду, нативний додаток буде оптимальним вибором. Однак, якщо метою є швидкий запуск на всіх платформах з обмеженим бюджетом та підтримка постійного оновлення, веб-додаток може бути більш підходящим варіантом. У багатьох випадках, для досягнення балансу між продуктивністю та доступністю, варто розглянути кросплатформні фреймворки, такі як React Native або Flutter, які поєднують переваги обох підходів.

Гібридні додатки дуже схожі на нативні, але мають свої відмінності (рис. 1.7). Такі додатки розробляються для різних ОС одночасно на одній мові програмування. Як результат, це позбавляє розробників від багатьох неполадок, пов’язаних із розробкою окремих версій програми для кожної ОС. Крім того, як і нативні програми, гібридні додатки вимагають доступу до елементів пристрою. Але, на відміну від нативних додатків, для стабільної роботи їм потрібне підключення до Інтернету. [5]

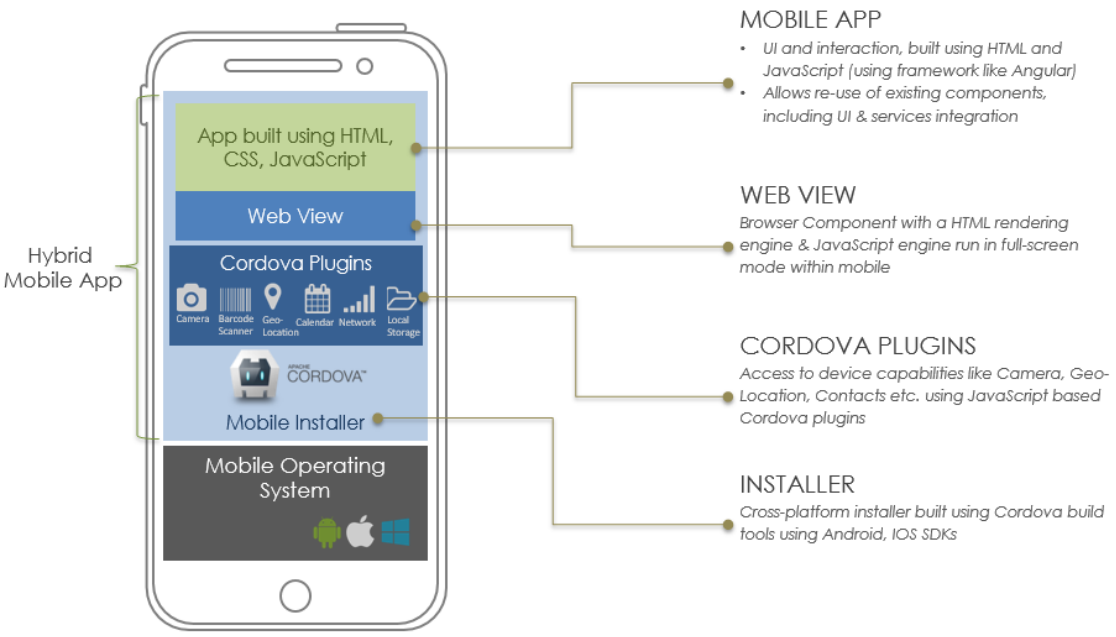


Рисунок 1.7 – Ключові особливості гібридних додатків

Гібридні додатки створюються з використанням таких веб-технологій, як HTML, CSS і JavaScript, а потім поміщаються в нативний контейнер, який дозволяє встановлювати та запускати їх на мобільних пристроях. Деякі з найпоширеніших технологій для розробки гібридних програм включають Apache Cordova, React Native, Ionic, Xamarin, Flutter та інші. [5]

Кросплатформний додаток – це програма, яка може працювати на кількох платформах, наприклад iOS, Android і Windows. Кросплатформні програми розробляються з використанням єдиної кодової бази, що може заощадити час і гроші на розробці. [5]

Розробку крос-платформних додатків можна здійснювати за допомогою таких технологій, як HTML/CSS, LavaScript і фреймворків (Ionic, React Native, Flutter, Xamarin тощо).

Міжплатформні програми розробляються з використанням єдиної кодової бази, тоді як гібридні програми розробляються з використанням двох кодових баз, по одній для кожної платформи. Крім того, міжплатформні програми можуть не мати доступу до всіх функцій платформи, тоді як гібридні програми зазвичай можуть отримати доступ до всіх функцій платформи. І гібридні та рідні програми можуть бути більш продуктивними, ніж кросплатформні додатки.

Особливості застосування мобільних додатків включають кілька важливих аспектів. Інтерфейс користувача (UI) та досвід користувача (UX) є ключовими для успіху додатку. Додатки повинні бути інтуїтивно зрозумілими та зручними у використанні, адаптованими до різних розмірів екрану та орієнтацій. Безпека є критично важливою, особливо в додатках для фінансів та здоров’я, тому використовуються шифрування та двофакторна аутентифікація. Продуктивність додатків має включати швидкий час завантаження та стабільну роботу навіть при обмежених ресурсах пристрою, а також оптимізацію споживання батареї.

1.2 Етапи проєктування і створення мобільних додатків

Щодня кількість додатків, випущених на ринок, зростає. За результатами останніх досліджень зараз у світі більше мобільних пристроїв, ніж людей. На думку SensorTower до 2026 року мобільні додатки принесуть 233 мільярди доларів доходу (рис. 1.8).

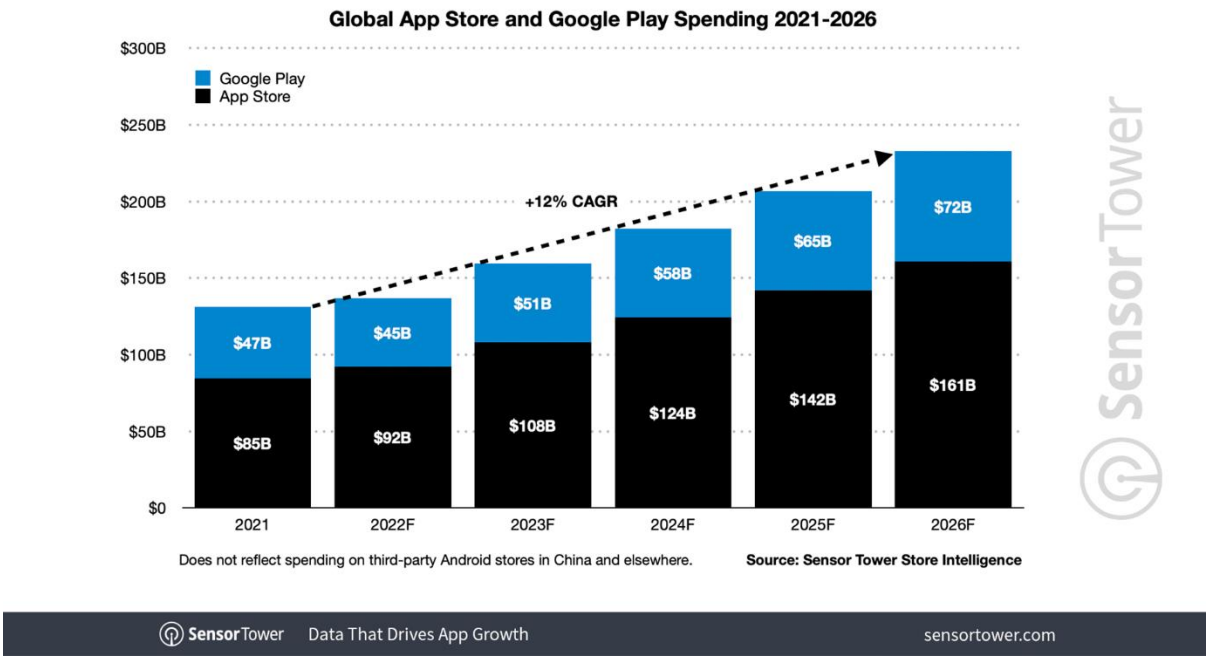


Рисунок 1.8 – Прогноз зростання ринку мобільних додатків до 2026 року [6]

Життєвий цикл розробки додатків (ADLC) — це структурована послідовність, яка керує плануванням, створенням, тестуванням, розгортанням і обслуговуванням додатків. ADLC складається з кількох ключових етапів, кожен з яких важливий для успішної розробки високоякісного програмного забезпечення (рис. 1.9).

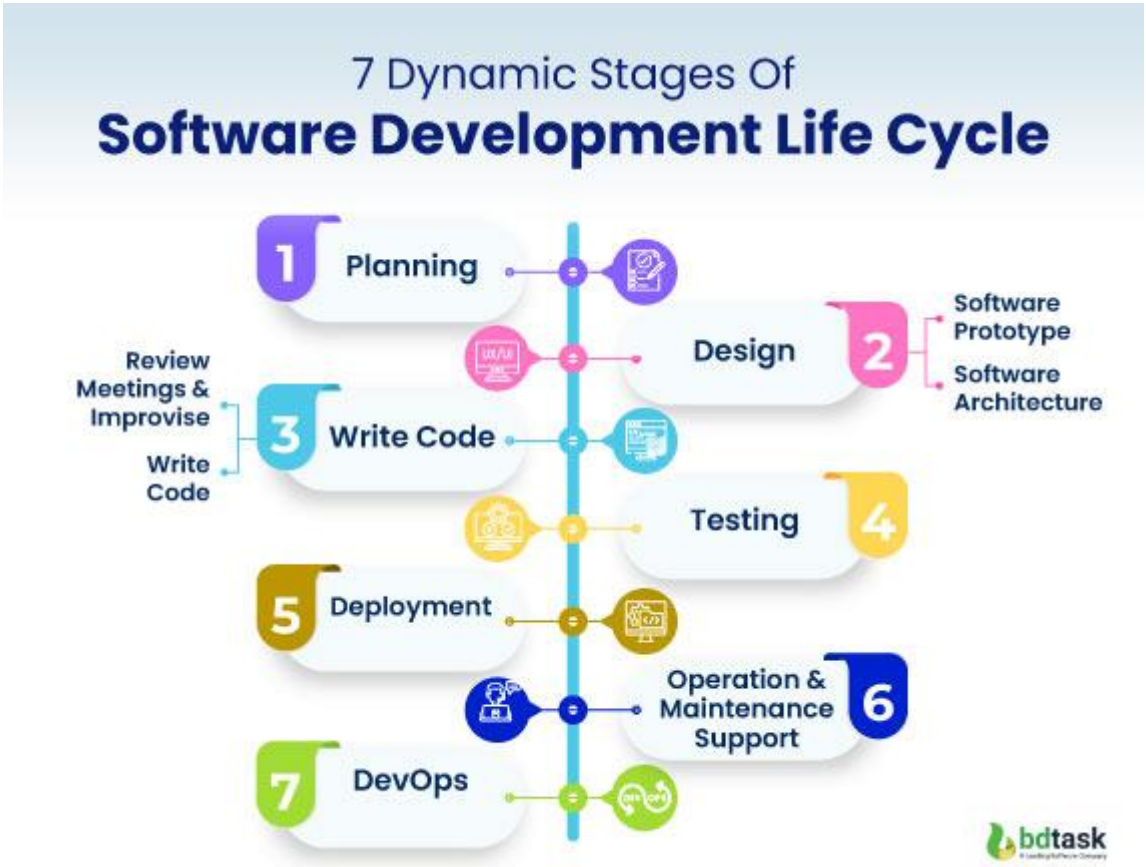


Рисунок 1.9 – Життєвий цикл розробки програми [8]

Управління розробкою додатків вимагає структурованого підходу для забезпечення успіху. 7 етапів життєвого циклу програми служать дорожньою картою, яка направляє команди розробників через тонкощі втілення ідеї в життя. Нехтування цими етапами може призвести до неузгодженості цілей, марної витрати ресурсів, програмного забезпечення з помилками та незадоволених користувачів. [7]

Життєвий цикл розробки додатків (Application Development Life Cycle, ADLC) є структурованим підходом, що охоплює всі етапи створення програмного забезпечення.

У контексті розробки мобільних додатків, ADLC складається з семи ключових кроків: планування, аналіз вимог, проектування, розробка, тестування, розгортання та підтримка. Кожен з цих кроків є критично важливим для забезпечення успішної розробки якісного мобільного додатку.

Перший крок — планування — включає визначення цілей проєкту, оцінку його масштабів та ресурсів, а також розробку графіку робіт. На цьому етапі важливо ідентифікувати цільову аудиторію та основні функціональні можливості майбутнього додатку. Крім того, планування охоплює оцінку бюджету та визначення ключових етапів реалізації проєкту.

Другий крок — аналіз вимог — передбачає збір та документування вимог до додатку. Це включає як функціональні вимоги (наприклад, які саме функції повинен виконувати додаток), так і нефункціональні вимоги (наприклад, вимоги до безпеки, продуктивності та сумісності). Важливим аспектом цього етапу є тісна

співпраця з замовником та кінцевими користувачами для точного розуміння їхніх потреб та очікувань.

Третій крок — проектування — включає розробку архітектури додатку, проектування інтерфейсу користувача (UI) та досвіду користувача (UX). Архітектурне проектування визначає структуру додатку, включаючи серверну частину (бекенд), клієнтську частину (фронтенд) та бази даних. Проектування UI/UX фокусується на створенні інтуїтивно зрозумілого та зручного інтерфейсу, що забезпечує приємний досвід користувача.

Четвертий крок — розробка — є процесом безпосереднього написання коду згідно з технічними вимогами та проектними рішеннями. Цей етап може бути розбитий на два паралельні процеси: фронтенд-розробка (створення клієнтської частини додатку) та бекенд-розробка (створення серверної частини додатку). Команда розробників працює над реалізацією функціоналу, інтеграцією з базами даних та зовнішніми сервісами, а також над забезпеченням безпеки додатку.

П'ятий крок — тестування — включає перевірку функціональності, продуктивності та безпеки додатку. Функціональне тестування забезпечує, що всі частини додатку працюють згідно з вимогами. Тестування продуктивності перевіряє, як додаток працює під навантаженням, а тестування безпеки виявляє потенційні вразливості. На цьому етапі також проводиться тестування на різних пристроях та операційних системах для забезпечення сумісності.

Шостий крок — розгортання — включає публікацію додатку в магазинах додатків, таких як App Store та Google Play. Це також може включати попереднє тестування на обмеженій аудиторії (бета-тестування), щоб виявити і виправити можливі проблеми перед масовим запуском. Процес розгортання також включає підготовку маркетингових матеріалів, таких як опис додатку, скріншоти та іконки.

Сьомий і останній крок — підтримка — включає моніторинг продуктивності додатку, збір зворотного зв'язку від користувачів та регулярне оновлення додатку. Це забезпечує виправлення виявлених помилок, покращення функціональності та додавання нових можливостей у відповідь на зміни ринку та потреби користувачів. Підтримка також включає технічну підтримку користувачів та забезпечення безперервної роботи додатку.

Кожен з цих кроків ADLC є важливим для забезпечення успішної розробки мобільного додатку. Підходячи до кожного етапу з ретельним плануванням та увагою до деталей, команда розробників може створити якісний, надійний та конкурентоспроможний продукт, що задовольнить потреби користувачів та замовника.

Існує кілька моделей життєвого циклу розробки додатків, кожна з яких має власний підхід до управління циклом розробки програмного забезпечення. Ось чотири поширені моделі ADLC із прикладами (рис. 1.10).

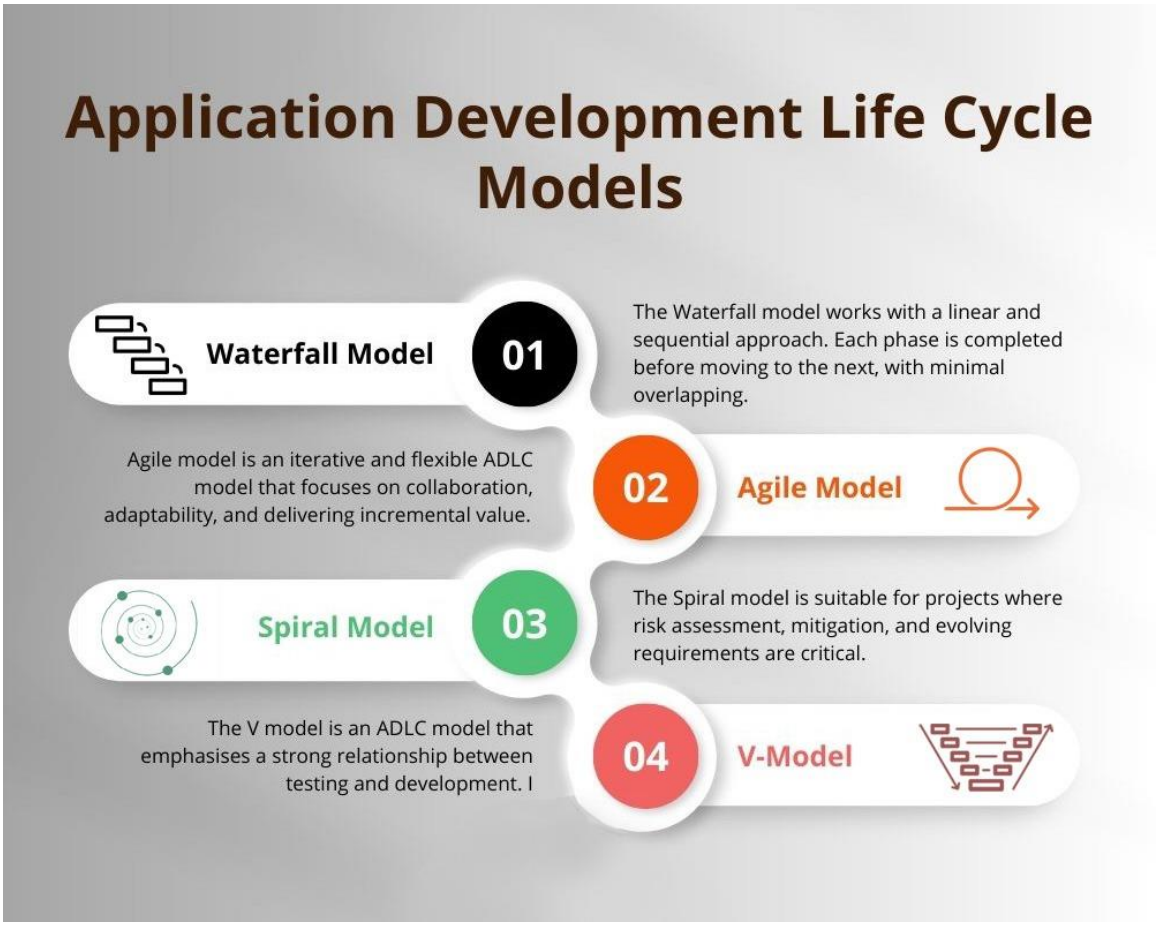


Рисунок 1.10 – Моделі життєвого циклу розробки мобільних додатків [9]

Модель Waterfall працює з лінійним і послідовним підходом. Кожен етап завершується перед переходом до наступного з мінімальним перекриттям. Він включає окремі етапи, такі як збір вимог, проектування, розробка, тестування та розгортання. Коли фаза закінчена, вона рідко повертається до попередніх фаз (рис. 1.11). Наприклад, якщо розробляється невеликий мобільний додаток із чітко визначеними вимогами та фіксованим бюджетом, модель Waterfall може підійти.

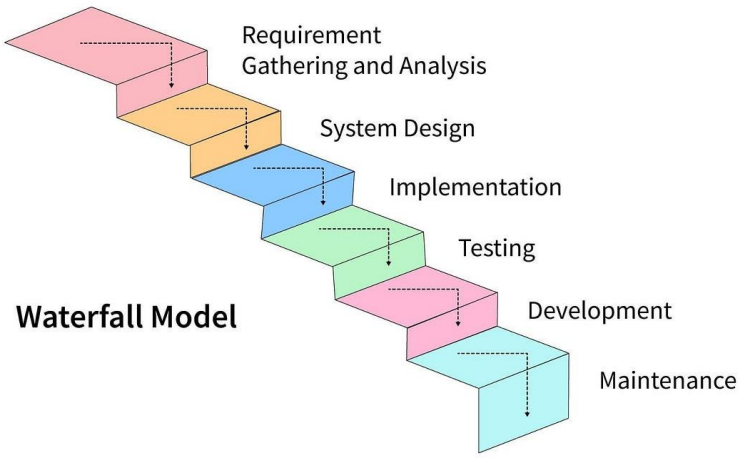


Рисунок 1.11 - Модель Waterfall

Модель Agile — це ітераційна та гнучка модель ADLC, яка зосереджена на співпраці, адаптивності та забезпеченні додаткової цінності. Вона передбачає розбиття всього процесу розробки програмного забезпечення на менші ітерації та спринти, де кожен спринт забезпечує функціональну частину програми. Agile сприяє постійному зворотному зв'язку, постійному вдосконаленню та здатності реагувати на мінливі вимоги (рис. 1.12). Наприклад, якщо розробляють складну веб-програму, яка потребує частих оновлень і включає відгуки користувачів, гнучкі методології, такі як Scrum або Kanban, можуть бути корисними.

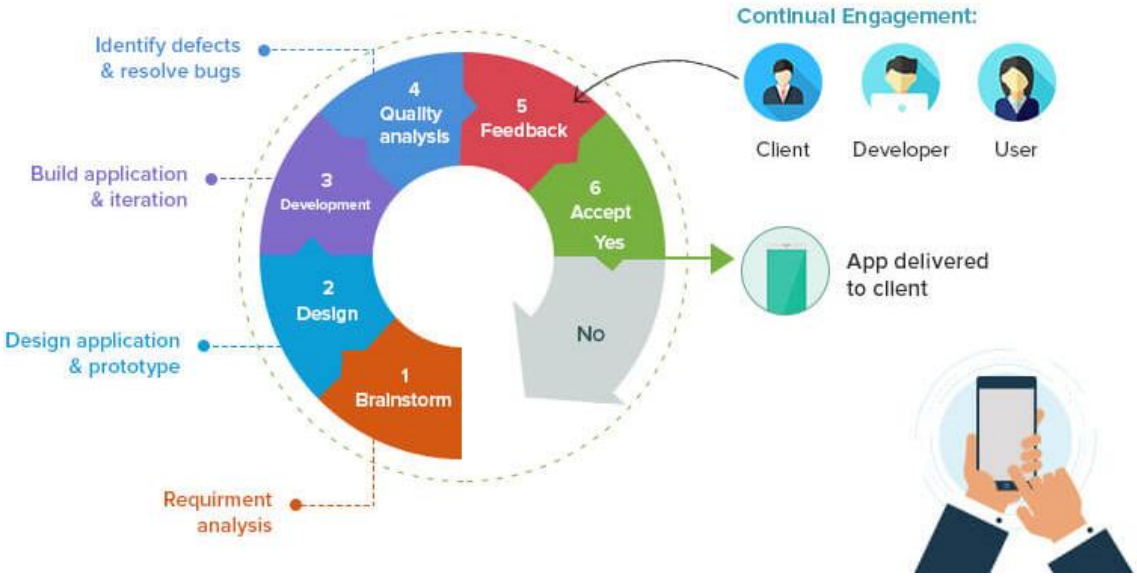


Рисунок 1.12 - Модель Agile

Спіральна модель підходить для проектів, де критично важливі оцінка ризиків, пом'якшення та зміни вимог. Наприклад, розробка програмного забезпечення для великої організації зі складними бізнес-процесами та змінними нормативними вимогами. Спіральна модель поєднує в собі елементи водоспаду та ітераційного підходів. Вона наголошує на управлінні ризиками та включає ітерації планування, аналізу ризиків, розробки та тестування (рис. 1.13). Це дозволяє створювати прототипи на ранній стадії та дає змогу зацікавленим сторонам надавати відгуки та визначати потенційні ризики на кожній ітерації.

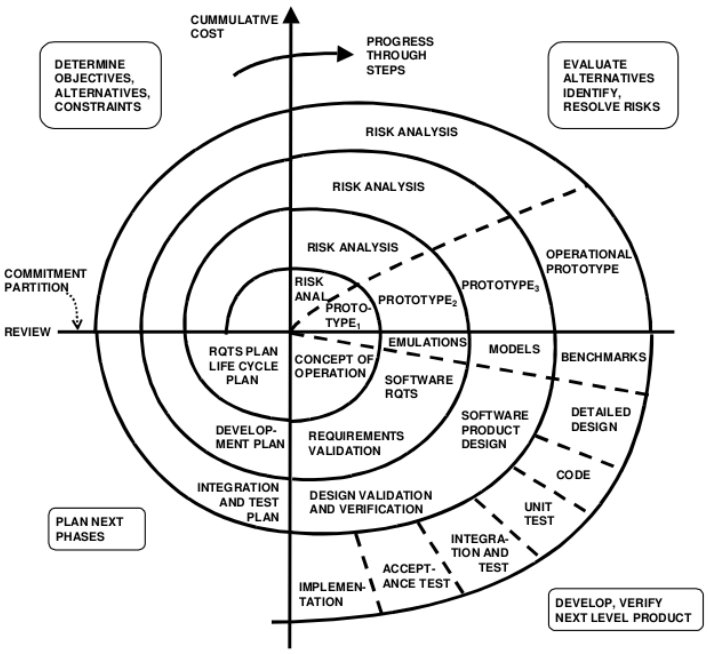


Рисунок 1.13 – Спіральна модель

Модель V — це модель ADLC, яка підкреслює тісний зв'язок між тестуванням і розробкою. Вона дотримується послідовного та структурованого підходу, коли кожна фаза розробки має відповідну фазу тестування. V-подібна форма представляє паралельну природу тестування та розробки, причому діяльність з тестування відображає відповідну діяльність з розробки (рис. 1.14). Ця модель забезпечує тісну інтеграцію тестування протягом усього процесу розробки, що забезпечує вищу якість і зниження ризиків проблем під час розгортання.

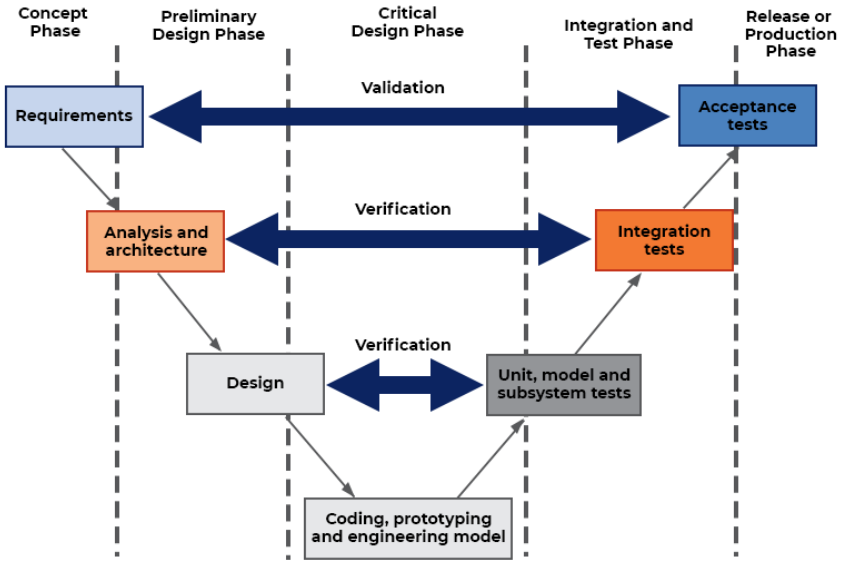


Рисунок 1.14 – Модель V

Структурований підхід до розробки мобільних додатків значно підвищує ефективність цього процесу завдяки чітко визначеним етапам і методичному

виконанню завдань. Такий підхід дозволяє зменшити ризики, пов'язані з непередбаченими проблемами, оскільки кожен етап має свої специфічні цілі та результати, які необхідно досягти перед переходом до наступного. Це сприяє кращій комунікації та співпраці між членами команди, забезпечуючи прозорість та координацію дій. Крім того, структурований підхід дозволяє більш точно планувати ресурси та час, що оптимізує витрати і підвищує загальну якість продукту завдяки регулярному тестуванню та інтеграції зворотного зв'язку від користувачів.

1.3 Принципи та інструменти побудови користувацького інтерфейсу

Побудова користувацького інтерфейсу (UI) мобільних додатків є критично важливим аспектом, який безпосередньо впливає на користувацький досвід (UX) та успіх продукту. Основні принципи та інструменти, що використовуються для створення UI, спрямовані на забезпечення інтуїтивності, естетичної привабливості та функціональності додатків. На рис. 1.15 показано приклад цільного користувацького інтерфейсу мобільного додатку, який в подальшому використовується для розробки програмного забезпечення.

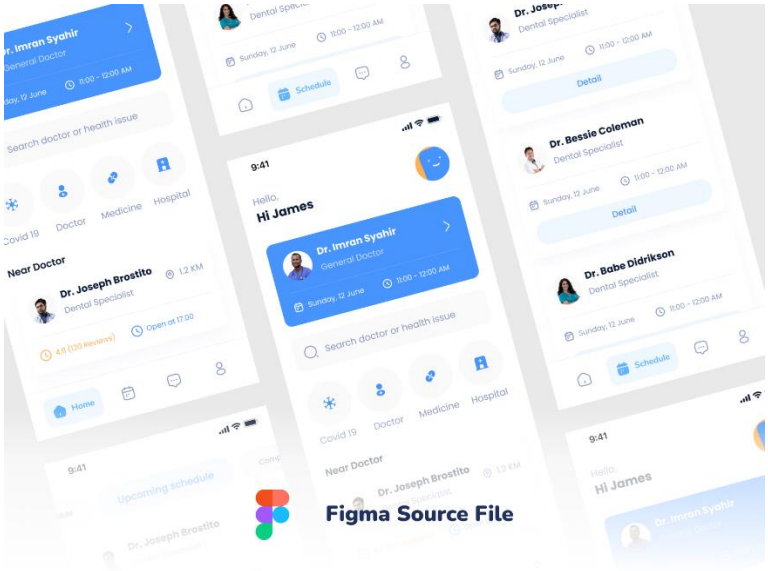


Рисунок 1.15 – Приклад повноцінного користувацького інтерфейсу

Одним з ключових принципів побудови UI є простота. Прості та зрозумілі інтерфейси допомагають користувачам швидко орієнтуватися та виконувати необхідні дії без зайвих зусиль. Це досягається шляхом мінімізації кількості елементів на екрані, використання зрозумілих іконок та лаконічного тексту. Принцип простоти також включає логічну організацію інформації, що дозволяє користувачам легко знаходити необхідні функції.

Консистентність є ще одним важливим принципом, який забезпечує однаковий вигляд та поведінку елементів інтерфейсу на всіх екранах додатку (рис. 1.16). Це сприяє створенню передбачуваного середовища для користувачів,

зменшуючи час на навчання та підвищуючи загальну зручність використання. Консистентність досягається завдяки дотриманню стандартів дизайну та використанню уніфікованих стилів для всіх компонентів додатку.

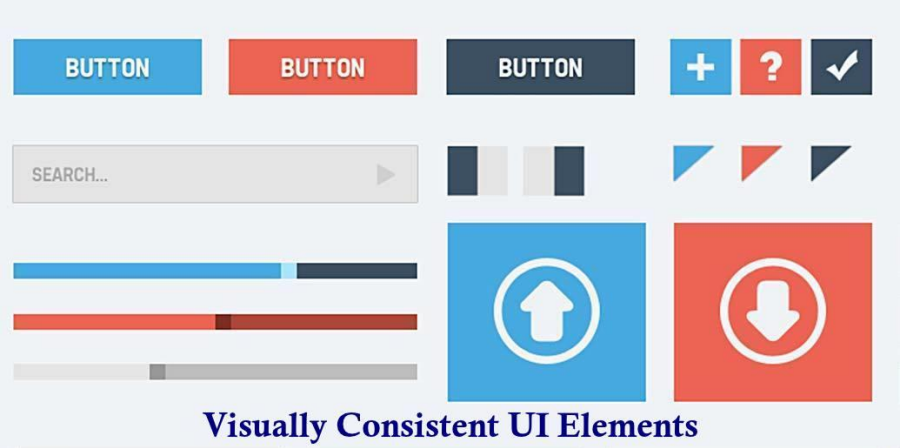


Рисунок 1.16 – Консистентність елементів користувацького інтерфейсу [10]

Адаптивність є необхідною для забезпечення коректного відображення додатку на різних пристроях з різними розмірами екрану (рис. 1.16). Інтерфейс повинен автоматично підлаштовуватися під розміри та роздільну здатність екрану, зберігаючи при цьому зручність та функціональність. Використання адаптивного дизайну та медіа-запитів дозволяє досягти цієї мети, забезпечуючи оптимальний вигляд та роботу додатку на будь-якому пристрої.

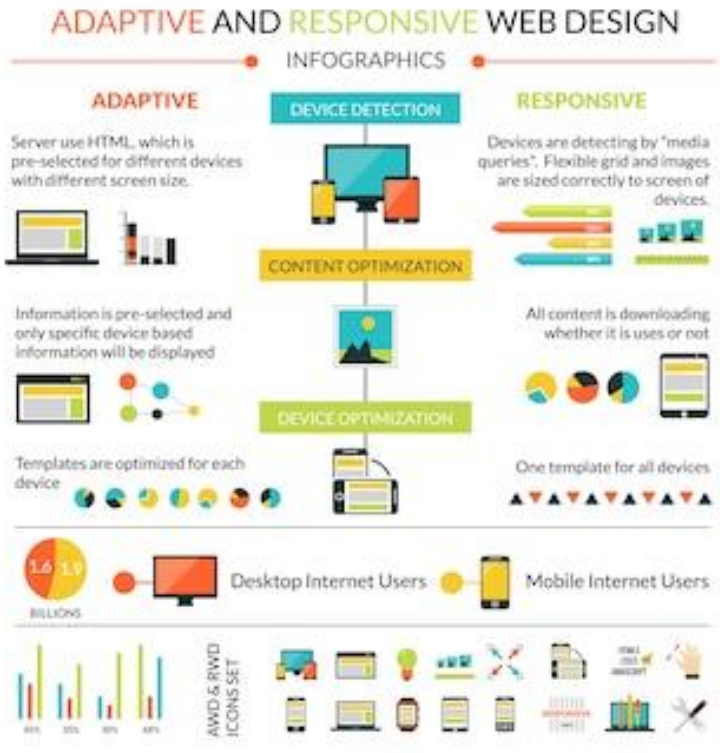


Рисунок 1.16 – Адаптивність у користувацькому інтерфейсі

Інструменти для побудови користувацького інтерфейсу мобільних додатків є різноманітними та пропонують розробникам широкі можливості. Sketch та Figma є популярними інструментами для дизайну інтерфейсів, що дозволяють створювати прототипи та макети з високою точністю. Вони підтримують спільну роботу, що полегшує співпрацю між дизайнерами та розробниками.

Для розробки нативних додатків використовуються iOS Human Interface Guidelines та Material Design Guidelines від Google, що пропонують рекомендації та шаблони для створення інтерфейсів, які відповідають вимогам відповідних платформ. Ці керівництва допомагають розробникам забезпечити консистентність та відповідність стандартам платформи, що значно підвищує якість та зручність додатків.

React Native та Flutter є потужними інструментами для кросплатформної розробки, що дозволяють створювати високоякісні інтерфейси для обох основних платформ (iOS та Android) з використанням єдиного коду. React Native, розроблений Facebook, використовує JavaScript та React для побудови компонентів інтерфейсу, тоді як Flutter, розроблений Google, використовує мову програмування Dart та власний рендеринг движок.

Adobe XD є ще одним важливим інструментом, що дозволяє дизайнерам створювати інтерактивні прототипи та макети. Він підтримує інтеграцію з іншими продуктами Adobe, що робить його зручним для використання в існуючих робочих процесах. Adobe XD також пропонує функціонал для спільної роботи, що полегшує комунікацію між членами команди та забезпечує узгодженість дизайну.

InVision є інструментом для створення інтерактивних прототипів та проведення користувацьких тестувань. Він дозволяє дизайнерам перетворювати статичні макети в інтерактивні прототипи, що допомагає отримати зворотний зв'язок від користувачів на ранніх етапах розробки. Це дозволяє виявити та виправити потенційні проблеми, перш ніж перейти до стадії розробки програмного забезпечення.

Прототипування користувацького інтерфейсу є ключовим етапом у процесі розробки мобільних додатків, що дозволяє створити інтерактивну модель додатку перед початком його безпосереднього кодування. Цей процес включає створення візуальних макетів, які демонструють основні елементи та функціональність інтерфейсу, забезпечуючи зрозуміле уявлення про кінцевий продукт. Інструменти для прототипування, такі як Sketch, Figma та Adobe XD, дозволяють дизайнерам швидко і ефективно створювати прототипи, які можна тестувати і обговорювати з командою та зацікавленими сторонами.

Прототипування допомагає виявити потенційні проблеми на ранніх стадіях розробки, що дозволяє заощадити час і ресурси, необхідні для виправлення помилок у пізніших етапах. Це також забезпечує можливість отримати зворотний зв'язок від користувачів, що є критично важливим для створення зручного та інтуїтивного інтерфейсу. Інтерактивні прототипи дозволяють користувачам безпосередньо взаємодіяти з дизайном, оцінюючи його функціональність і зручність.

На рис. 1.17 показано переваги та особливості інструментів розробки користувацького інтерфейсу.

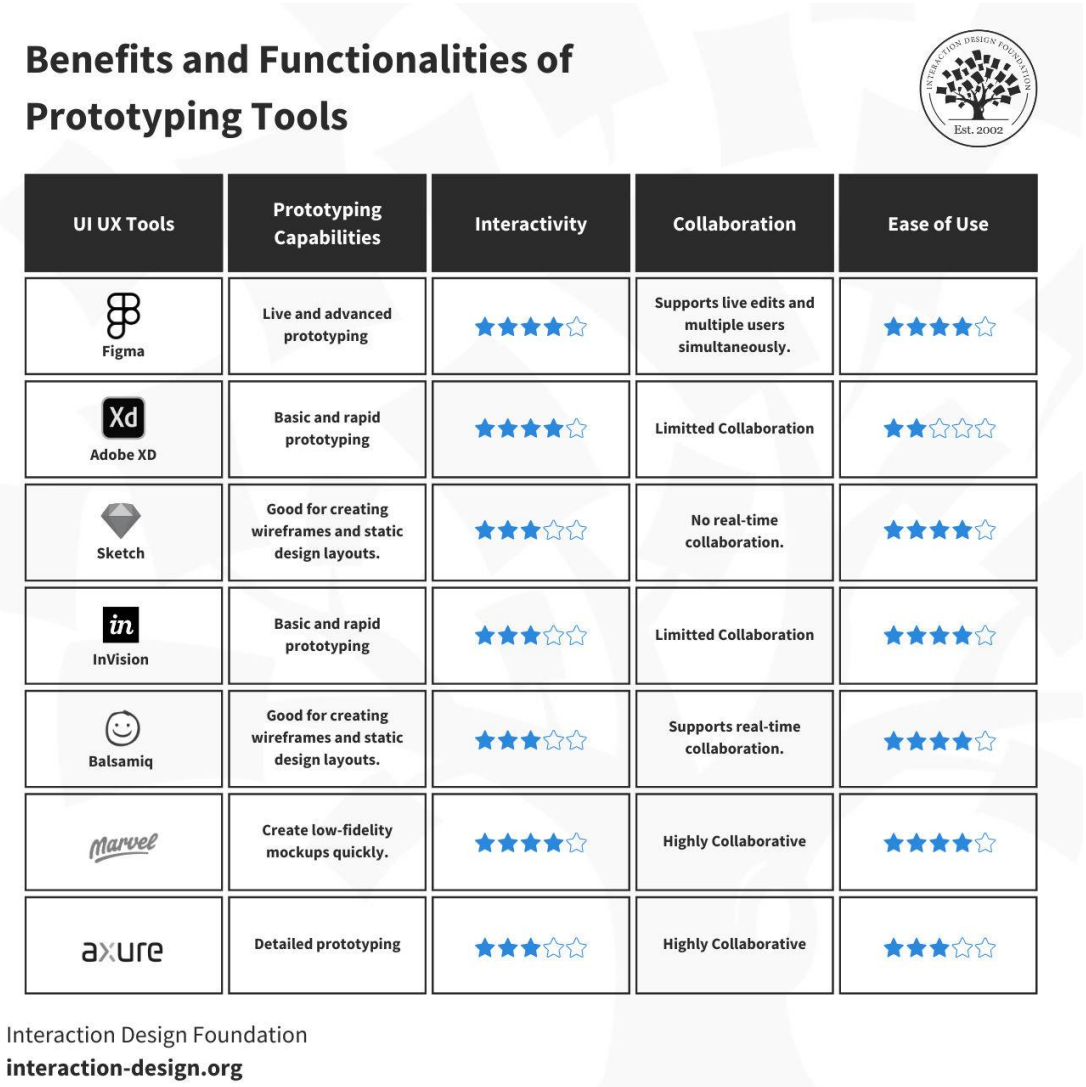
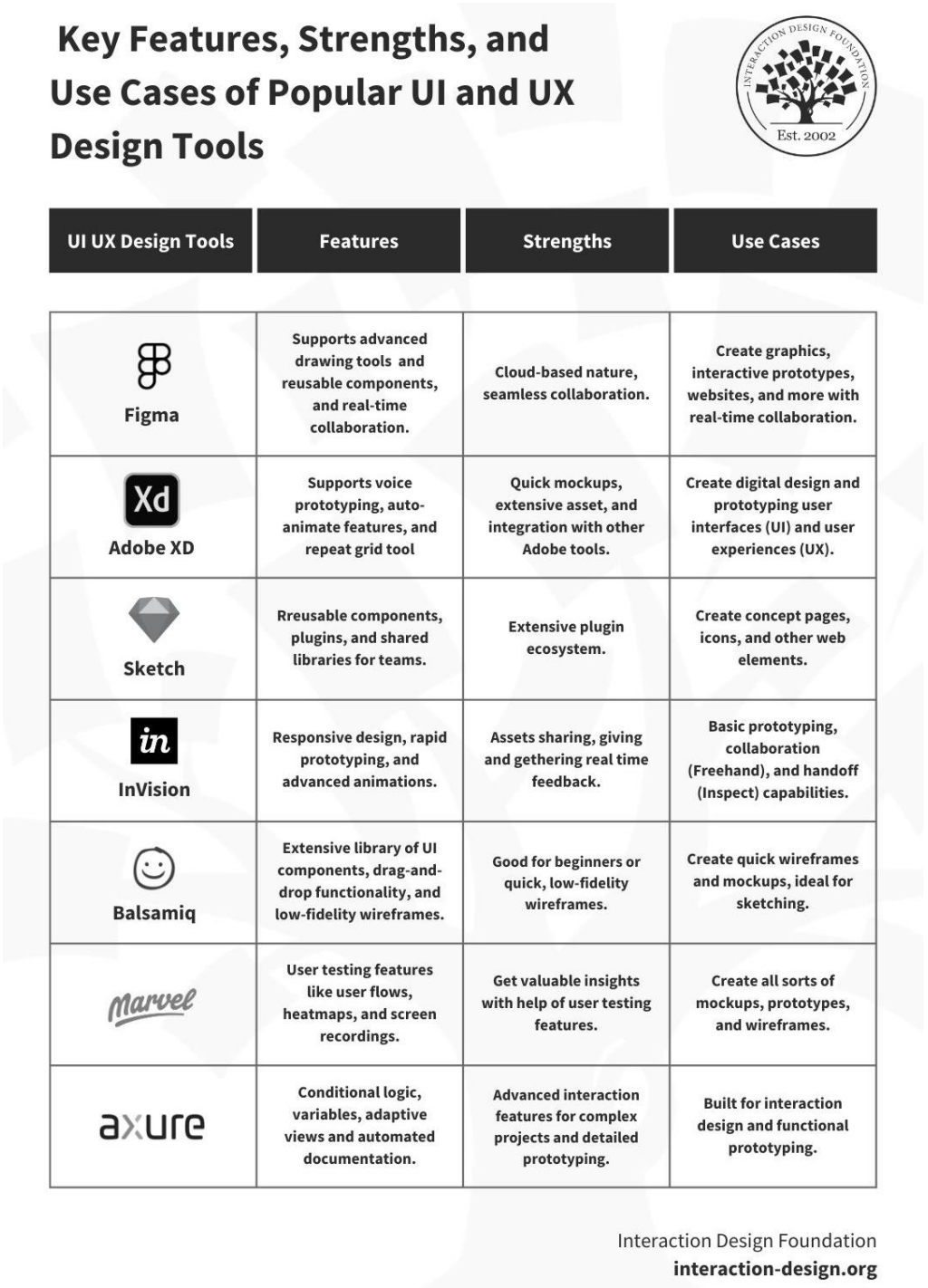


Рисунок 1.17 – Інструменти розробки користувацького інтерфейсу

Прототипи можуть бути як низької точності (Low-fidelity), що представляють загальну структуру і потоки додатку, так і високої точності (High-fidelity), що детально відображають кінцевий вигляд та поведінку інтерфейсу. Таким чином, прототипування є важливим інструментом, що допомагає командам розробників і дизайнерів створювати ефективні, зручні та привабливі мобільні додатки.

Прототипування також сприяє кращій комунікації між розробниками, дизайнерами та іншими зацікавленими сторонами, забезпечуючи візуальне представлення ідей та концепцій. Це полегшує процес узгодження вимог та очікувань, зменшуючи ризик непорозумінь та невідповідностей у кінцевому продукті. Крім того, прототипи можуть бути використані для проведення користувацьких тестувань, що дозволяє збирати цінний зворотний зв'язок безпосередньо від кінцевих користувачів ще до початку розробки. Це дає можливість вносити необхідні зміни та покращення на ранніх етапах, що значно підвищує якість та конкурентоспроможність додатку.

На рис. 1.18 показано порівняння різних інструментів створення користувацького інтерфейсу та узагальнено їх ключові особливості.



Key Features, Strengths, and Use Cases of Popular UI and UX Design Tools

UI UX Design Tools	Features	Strengths	Use Cases
Figma	Supports advanced drawing tools and reusable components, and real-time collaboration.	Cloud-based nature, seamless collaboration.	Create graphics, interactive prototypes, websites, and more with real-time collaboration.
Adobe XD	Supports voice prototyping, auto-animate features, and repeat grid tool	Quick mockups, extensive asset, and integration with other Adobe tools.	Create digital design and prototyping user interfaces (UI) and user experiences (UX).
Sketch	Rreusable components, plugins, and shared libraries for teams.	Extensive plugin ecosystem.	Create concept pages, icons, and other web elements.
InVision	Responsive design, rapid prototyping, and advanced animations.	Assets sharing, giving and gethering real time feedback.	Basic prototyping, collaboration (Freehand), and handoff (Inspect) capabilities.
Balsamiq	Extensive library of UI components, drag-and-drop functionality, and low-fidelity wireframes.	Good for beginners or quick, low-fidelity wireframes.	Create quick wireframes and mockups, ideal for sketching.
Marvel	User testing features like user flows, heatmaps, and screen recordings.	Get valuable insights with help of user testing features.	Create all sorts of mockups, prototypes, and wireframes.
axure	Conditional logic, variables, adaptive views and automated documentation.	Advanced interaction features for complex projects and detailed prototyping.	Built for interaction design and functional prototyping.

Interaction Design Foundation
interaction-design.org

Рис. 1.18 – Характерні особливості інструментів розробки UI/UX [11]

Узагальнюючи, побудова користувацького інтерфейсу мобільних додатків вимагає дотримання основних принципів простоти, консистентності та адаптивності. Використання сучасних інструментів для дизайну та розробки інтерфейсів, таких як Sketch, Figma, React Native, Flutter, Adobe XD та InVision, дозволяє створювати високоякісні, зручні та естетично привабливі додатки. Ці інструменти надають розробникам та дизайнерам можливості для ефективної співпраці, швидкого прототипування та тестування, що є ключовими факторами для успішної реалізації проєктів.

1.4 Архітектура мобільних додатків

Архітектура мобільних додатків є важливою складовою їх успішної розробки та функціонування, оскільки вона визначає структуру, компоненти та взаємодію між ними (рис. 1.19). Основна мета архітектури мобільного додатку полягає в забезпеченні масштабованості, продуктивності, безпеки та зручності в обслуговуванні.

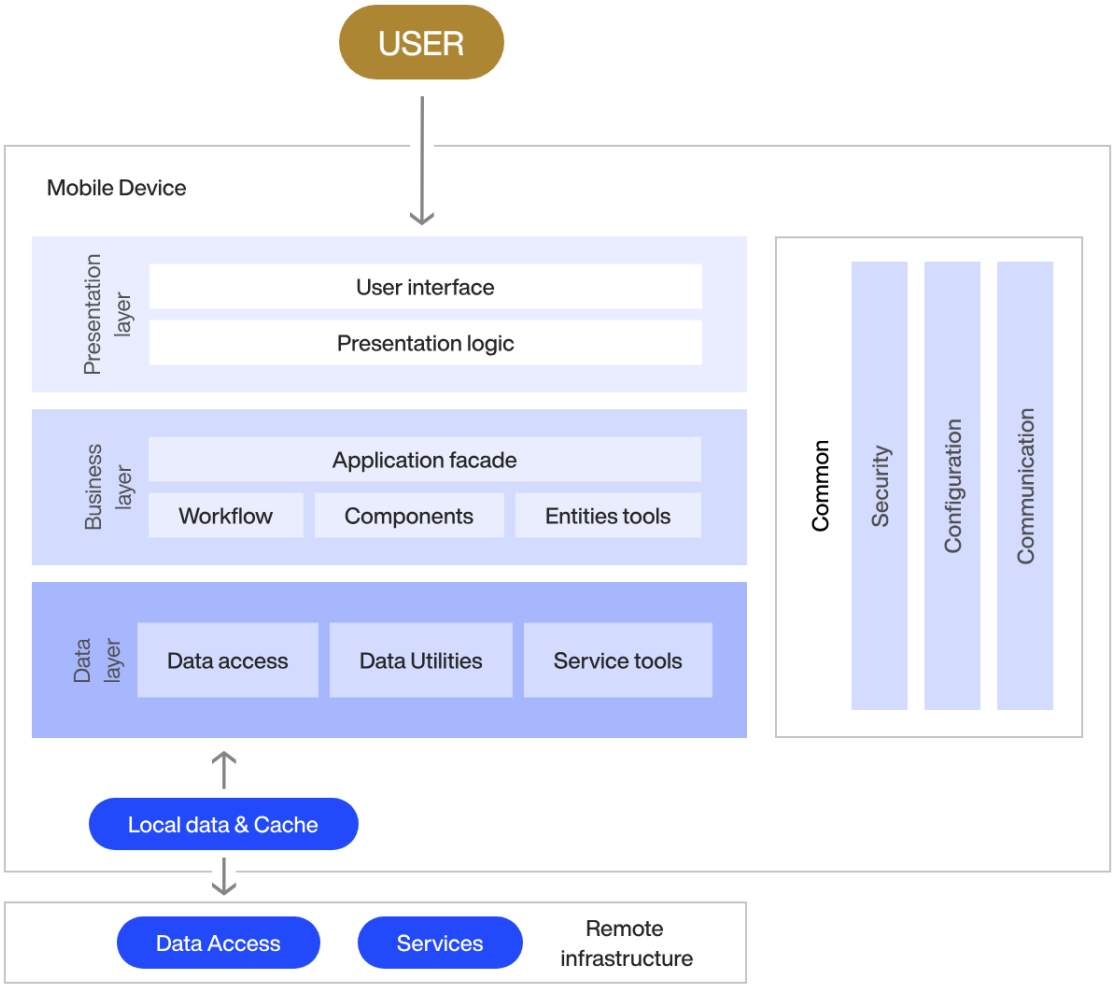


Рисунок 1.19 – Приклад архітектури мобільного додатку [12]

Однією з головних переваг архітектури програми є модульність. Це означає, що програма розділена на компоненти, які можна розробляти, оновлювати та тестувати незалежно (рис. 1.20).

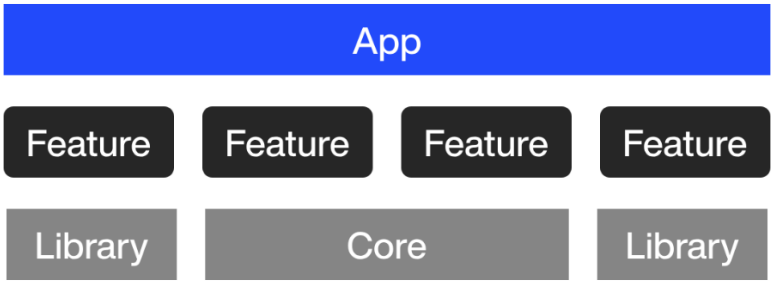


Рисунок 1.20 – Модульна архітектура програми

Завдяки модульності хороша архітектура також дозволяє багаторазове використання. Це дозволяє використовувати один код у кількох проектах, заощаджуючи величезний час і зусилля на розробку.

Архітектура мобільного додатку також має вирішальне значення для підвищення безпеки. Це дозволяє розділяти конфіденційні дані та код, а потім створювати протоколи безпеки навколо них.

Наприклад, можна встановити шифрування SSL між клієнтською машиною та логікою коду на веб-сервері, як показано на рис. 1. 21. [12]

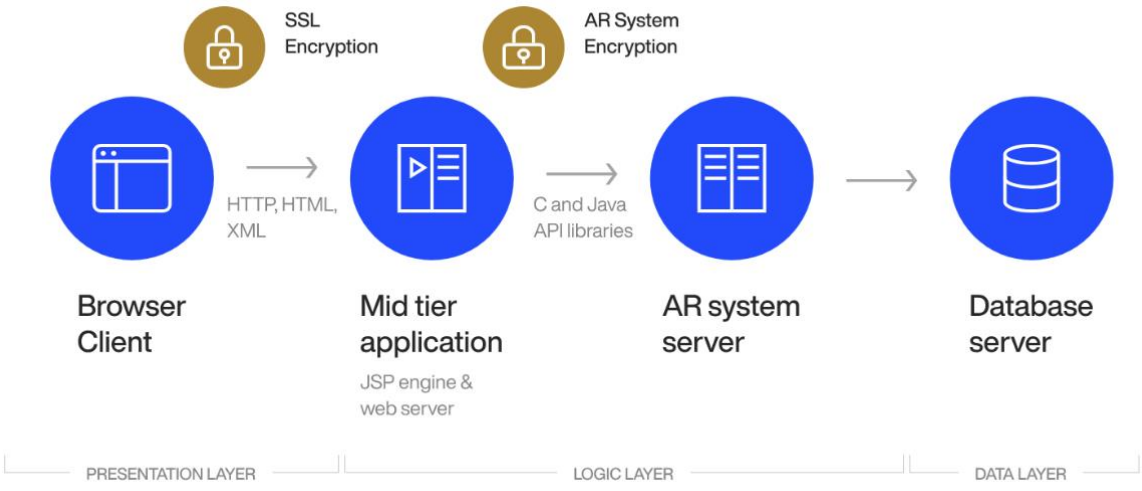


Рисунок 1.21 – Безпека даних за допомогою шифрування SSL [12]

Більшість типів архітектури програми складаються з трьох рівнів: рівень даних (data layer), рівень бізнес-логіки (business layer), та рівень презентації (presentation layer) (рис. 1.22).

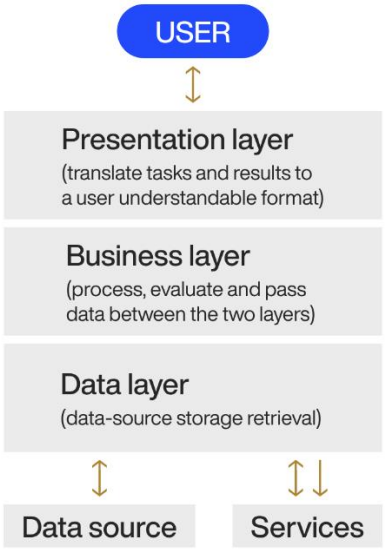


Рисунок 1.22 – Фундаментальні рівні в архітектурі мобільних додатків [12]

Рівень даних відповідає за доступ до даних та їх управління. Це включає зберігання, отримання, оновлення та видалення інформації з різних джерел даних, таких як бази даних, веб-служби або зовнішні API.

Рівень бізнес-логіки визначає логіку додатку та обробку даних (рис.1.23). Він відповідає за виконання різних операцій над даними, обробку бізнес-правил та взаємодію з рівнем даних для отримання та збереження інформації.

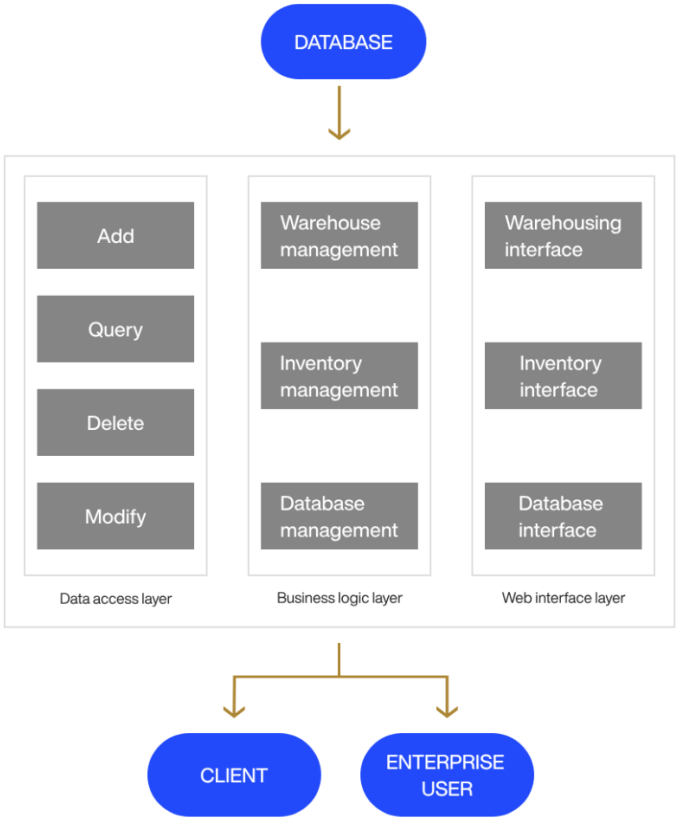


Рисунок 1.23 – Рівень бізнес-логіки у архітектурі мобільних додатків [12]

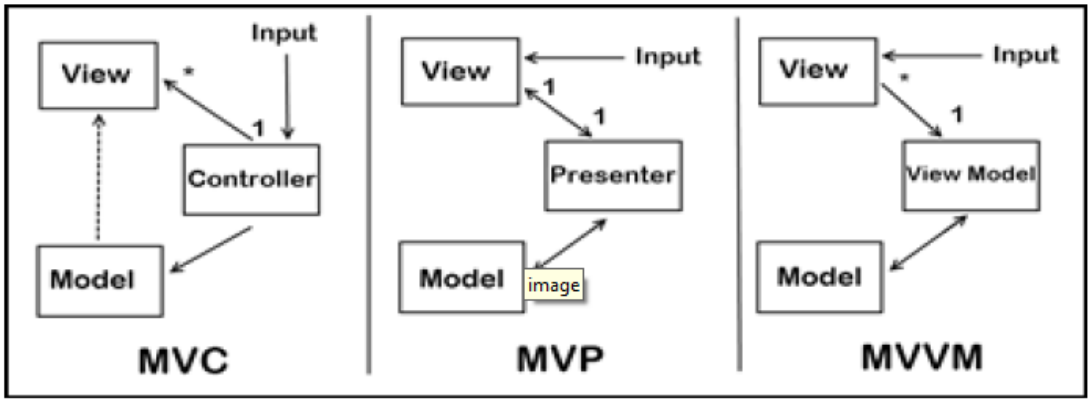
Рівень презентації відображає дані користувачам та дозволяє їм взаємодіяти з додатком. Це включає створення інтерфейсу користувача, відображення даних у зручному для сприйняття вигляді та обробку користувацьких взаємодій.

Ці три рівні взаємодіють між собою для забезпечення функціональності та ефективності додатку. Наприклад, презентаційний рівень може взаємодіяти з бізнес-логікою для отримання даних та їх відображення на екрані, а бізнес-логіка в свою чергу взаємодіє з рівнем даних для отримання або збереження інформації.

Цей підхід дозволяє розділити функціональність додатку на окремі компоненти, що полегшує його розробку, тестування та обслуговування. Використання цих трьох рівнів архітектури допомагає створити добре структуроване та модульне програмне забезпечення, яке легко масштабувати та підтримувати у майбутньому.

Існує декілька популярних архітектурних підходів, серед яких найбільш поширені: MVC (Model-View-Controller), MVP (Model-View-Presenter) та MVVM (Model-View-ViewModel) (рис. 1.24).

Difference between MVP, MVVM, and MVC.



MVC	MVP	MVVM
The Input is directed at the controller	The Input begins with View	The input begins with model
Many-to-one relation between Controller and View	One-to-one mapping between view and its associated presenter	One-to-may mapping between various and one ViewModel
View does not have any knowledge about the controller	The View hold reference to the presenter; Presenter also reacting to the events being triggered from the View,so presenter does aware of the View	The View holds reference to the ViewModel;View Model has no information about the View.Here different technology can share viewModel (i.e) WCF and Silverlight can share same ViewModel.
Controller does pass the Model to the View,so View has knowledge about the controller	View is not Model aware	View is not Model aware

Рисунок 1.24 – Порівняння популярних архітектурних підходів до розробки мобільних додатків [13]

Архітектура MVC розділяє додаток на три основні компоненти: модель, яка відповідає за логіку даних; уявлення (view), що займається відображенням

інформації; і контролер, який обробляє взаємодію користувача та координує модель і уявлення (рис. 1.25). Цей підхід сприяє розподілу відповідальності, що полегшує тестування та обслуговування додатку.

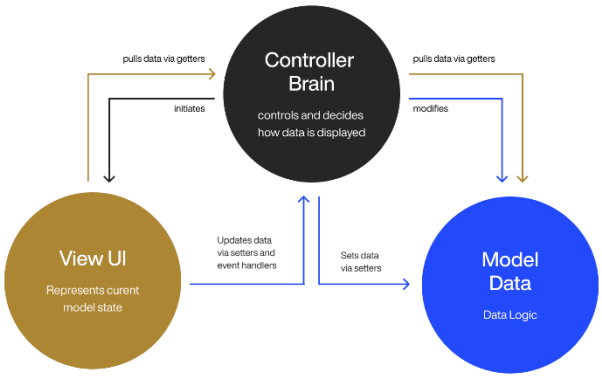


Рисунок 1.25 – Архітектура Model-View-Controller [12]

Архітектура MVP є розвитком MVC і додає проміжний шар — презентер, який відповідає за логіку презентації (рис. 1.26). У цьому підході уявлення пасивне та лише відображає дані, отримані від презентера, що значно спрощує тестування логіки додатку та забезпечує більш чітке розділення відповідальності між компонентами.

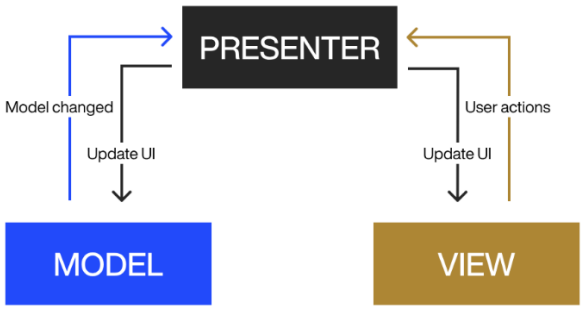


Рисунок 1.26 - Архітектура Model-View-Presenter [12]

Архітектура MVVM також спрямована на чітке розділення відповідальності, але використовує ViewModel для обробки логіки презентації та підтримки стану уявлення (рис. 1.27). Це дозволяє легко зв'язувати уявлення з моделлю за допомогою двостороннього зв'язування даних, що є особливо корисним у сучасних фреймворках, таких як Xamarin.Forms або Flutter.

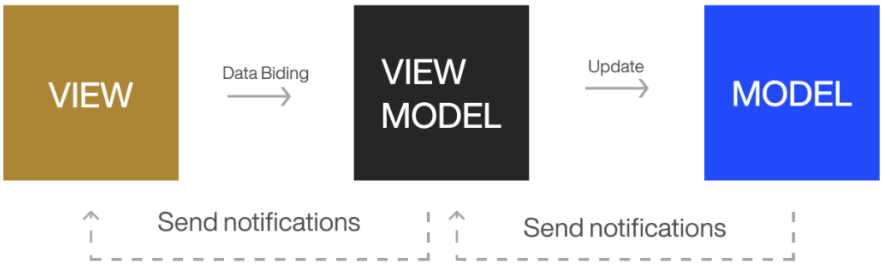


Рисунок 1.27 - Архітектура Model-View-ViewModel [12]

Однією з важливих складових архітектури мобільних додатків є управління станом. Інструменти, такі як Redux або MobX, допомагають централізовано керувати станом додатку, забезпечуючи передбачуваність і спрощуючи відладку та тестування.

Іншим важливим аспектом є вибір підходу до взаємодії з сервером. Архітектури, що базуються на RESTful API або GraphQL, забезпечують ефективну та гнучку комунікацію між клієнтом та сервером. Використання кешування та обробки офлайн-даних, таких як Room для Android або Core Data для iOS, дозволяє підвищити продуктивність і забезпечити безперебійну роботу додатку навіть за відсутності стабільного інтернет-з’єднання.

Безпека мобільних додатків є ще одним критично важливим аспектом архітектури. Використання методів шифрування, безпечного зберігання даних та автентифікації користувачів допомагає захистити додаток від потенційних загроз. Важливими є також заходи для запобігання витокам даних та захисту від атак типу «людина посередині» (MITM).

Архітектура мобільних додатків відіграє ключову роль у їх успішній розробці та експлуатації. Вибір відповідної архітектури залежить від специфіки проєкту, вимог замовника та технологічних обмежень. Правильний підхід до архітектури забезпечує гнучкість, масштабованість та високу якість кінцевого продукту, що є критично важливим для задоволення потреб сучасних користувачів.

2 ОСОБЛИВОСТІ ВИКОРИСТАННЯ ФРЕЙМВОРКУ FLUTTER ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

2.1 Ключові технології та підходи фреймворку Flutter

Flutter — це фреймворк із відкритим вихідним кодом від Google для створення красивих, нативно скомпільованих мультиплатформенних додатків із єдиної кодової бази. [14]

Згідно останніх даних у Google Trends фреймворк Flutter користується найбільшою популярністю серед схожих інструментів (рис. 2.1).

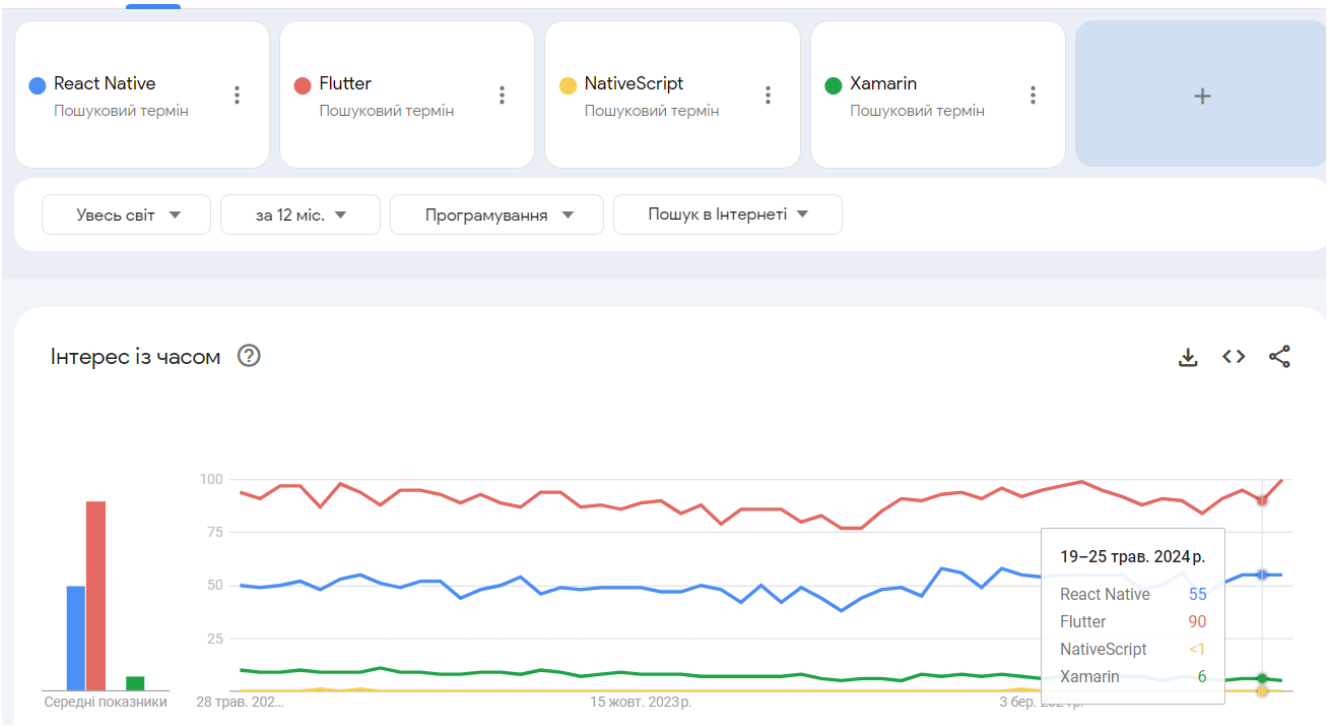


Рисунок 2.1 – Порівняння інтересу до інструментів Flutter, React Native, Xamarin та NativeScript [17]

На відміну від інших популярних рішень, Flutter не є фреймворком або бібліотекою. Це повний SDK – набір для розробки програмного забезпечення.

Flutter — це міжплатформний набір інструментів інтерфейсу користувача, розроблений для повторного використання коду в операційних системах, таких як iOS і Android, а також дозволяє додаткам безпосередньо взаємодіяти з основними службами платформи. Мета полягає в тому, щоб дозволити розробникам створювати високопродуктивні додатки, які природно виглядають на різних платформах, враховуючи відмінності там, де вони існують, і водночас ділиться якомога більшою кількістю коду.

Бібліотека — це, по суті, повторно використовуваний фрагмент коду, який ви розміщуєте у своїй програмі для виконання певної функції.

Фреймворк — це структура, яка надає скелетну архітектуру для створення програмного забезпечення.

SDK має набагато ширший обсяг, оскільки це набір інструментів, у який включено бібліотеки, документацію, API, інколи фреймворки – усе, що потрібно для розробки програмного забезпечення.

Інші технології, такі як Xamarin, React Native, Ionic або NativeScript, також використовуються для розробки програм, які працюють на кількох платформах. [16]

Flutter розроблено як розширювану 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. 2.2).

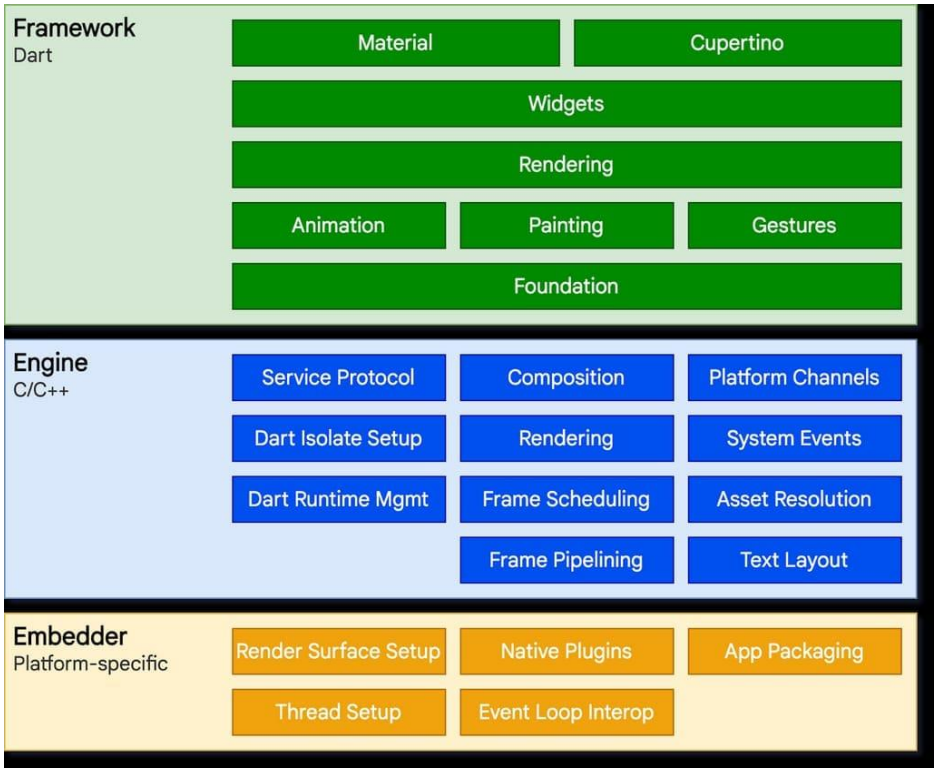


Рисунок 2.2 - Архітектура SDK Flutter [15]

Три основні архітектурні шари Flutter:

1. Embedder, який використовує специфічну для платформи мову та дозволяє програмі працювати на будь-якій ОС;
2. Engine, написаний на C/C++, який забезпечує низькорівневу реалізацію основних API Flutter. Це включає графіку (через графічну бібліотеку Skia 2D), макет тексту, файловий і мережевий ввід-вивід, підтримку спеціальних можливостей, архітектуру плагінів, а також середовище виконання Dart і ланцюжок інструментів компіляції; і
3. Framework на основі мови програмування Dart. Його реалізація необов'язкова, але вона надає багатий набір бібліотек, які можна розділити на рівні: базові базові класи, рівень візуалізації, рівень віджетів і бібліотеки Material/Cupertino.

Для основної операційної системи програми Flutter упаковані так само, як і будь-яка інша рідна програма. Спеціальна платформа для вбудовування забезпечує точку входу; координується з базовою операційною системою для доступу до таких послуг, як поверхні відтворення, доступність і введення; і керує циклом подій повідомлення. Вбудовувач написано мовою, яка підходить для платформи: наразі 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 містить низку вбудовувачів для поширених цільових платформ, але існують і інші вбудовувачі.

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. Він забезпечує низькорівневу реалізацію основного API Flutter, включаючи графіку (через Impeller на iOS і вихід на Android, а також Skia на інших платформах), макет тексту, файловий і мережевий ввід-вивід, підтримку доступності, архітектуру плагінів і середовище виконання Dart і скомпілювати інструментарій.

Двигун піддається впливу фреймворку Flutter через dart:ui, який обертає 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 платформи, макета та базових бібліотек, які складаються з серії шарів. Працюючи знизу вгору, розробник має:

- 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 ви можете побудувати дерево об’єктів, які можна відобразити. Ви можете керувати цими об’єктами динамічно, при цьому дерево автоматично оновлює макет, щоб відобразити ваші 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; багато функцій вищого рівня, які можуть використовувати розробники, реалізовано у вигляді пакетів, включаючи плагіни платформи, як-от camera та webview, а також функції, не залежні від платформи, як-от символи, http та анімації, які базуються на основних бібліотеках

Dart і Flutter. Деякі з цих пакетів походять із ширшої екосистеми, охоплюючи такі послуги, як платежі в програмі, автентифікація Apple і анімація. [15]

Технологія, яка лежить в основі Flutter, це Dart. Це оптимізована для клієнта об'єктно-орієнтована мова програмування, розроблена Google. Dart здатний компілюватися у нативний код для мобільних пристроїв і комп'ютерів, а також у JavaScript. Завдяки цій прямій компіляції не потрібен додатковий міст для зв'язку з платформою, як, наприклад, ReactNative. Це значно покращує час запуску та загальну продуктивність програми.

Іншою важливою частиною Flutter є його віджети. У Flutter SDK вони служать будівельними блоками, які можуть охоплювати майже всі аспекти розробки. Flutter пропонує не тільки широкий вибір готових віджетів, але й дозволяє налаштовувати їх або створювати власні.

Flutter також пропонує набір інструментів для автоматизованого тестування, зокрема, для трьох типів тестів: модульного тесту, тесту віджетів та інтеграційного тесту. Крім того, Flutter підтримує модель Continuous Delivery через fastlane, безкоштовну платформу, яка поєднує Flutter з такими популярними інструментами CI, як Travis, Jenkins або Cirrus (перегляньте посібник із безперервної доставки з Flutter).

Відладка у Flutter виконується за допомогою Flutter DevTools (також називається Dart DevTools). Вони використовуються для перевірки макета, аналізу продуктивності, відладки програм тощо.

Багато компаній обирають Flutter для розробки фірмових мобільних додатків, здатних забезпечувати чудову взаємодію з клієнтами. Серед відомих прихильників цього інструменту Toyota, BMW, eBay, Alibaba Group, Groupon і Etsy, щоб назвати лише деякі.

SDK також ідеально вписується в екосистему стартапів, оскільки він є відкритим кодом, багатофункціональним і економічно ефективним. Деякі стартапи, які масштабуються за допомогою Flutter:

- Nubank, бразильський необанк і на сьогодні найбільший цифровий банк у Латинській Америці;
- Invoice Ninja, платформа для виставлення рахунків і платежів для малого бізнесу;
- Reflectly, програма для психічного здоров'я на основі ШІ.

Також окрім кросплатформних мобільних додатків, які є основною метою Flutter, SDK знайшов застосування в багатьох інших сферах. Швидкий цикл розробки та портативність роблять технологію ідеальною для створення мінімально життєздатних продуктів (MVP) і створення прототипів, адже це можливість швидко протестувати свою бізнес-ідею на різних платформах.

Що стосується веб-розробки, Flutter може бути ідеальним рішенням для прогресивних веб-програм (PWA) і односторінкових програм (SPA). Це також дозволяє масштабувати існуючий мобільний проект до Інтернету та комп'ютера. [16]

Окремою перевагою, яку слід розглянути, є зручний інтерфейс Flutter із можливістю паралельного перегляду дизайну мобільного додатку та процесу розробки програмного забезпечення (рис. 2.3).

Flutter пропонує широкий спектр вбудованих інтерфейсних елементів, які можна легко налаштувати та кастомізувати. За допомогою гнучкої системи віджетів, розробники можуть створювати складні інтерфейси та анімації з високою швидкістю та ефективністю. Вбудована підтримка гарячого перезавантаження дозволяє швидко побачити зміни у реальному часі під час розробки. Flutter також надає можливості для реактивного програмування та станової управління за допомогою бібліотеки Flutter Riverpod або пакету станової управління Flutter Bloc.

Інтерфейс Flutter дозволяє використовувати різні інструменти для розробки, такі як Android Studio, IntelliJ IDEA або Visual Studio Code, забезпечуючи комфортні умови для роботи з різними платформами.

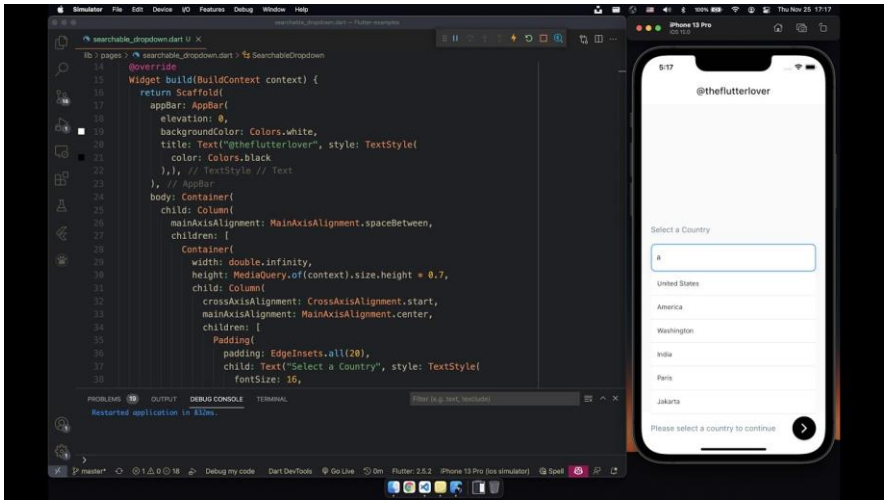


Рисунок 2.3 – Інтерфейс Flutter

Під час розробки програми Flutter запускаються у віртуальній машині, яка пропонує гаряче перезавантаження змін без потреби повної перекомпіляції. Для випуску додатки Flutter компілюються безпосередньо в машинний код, інструкції Intel x64 чи ARM, або в JavaScript, якщо націлено на Інтернет. Фреймворк має відкритий вихідний код із дозвільною ліцензією BSD і має процвітаючу екосистему пакетів сторонніх розробників, які доповнюють основні функції бібліотеки.

На рис. 2.4 наведено огляд частин, які складають звичайну програму Flutter, створену flutter create. Він показує, де Flutter Engine знаходиться в цьому стеку, підкреслює межі API та ідентифікує репозиторії, де знаходяться окремі частини. Легенда нижче пояснює деякі терміни, які зазвичай використовуються для опису частин програми Flutter.

Dart App - це додаток, який складає віджети в потрібний користувацький інтерфейс та реалізує бізнес-логіку. Власником Dart App є розробник додатка.

Framework - це високорівневий інтерфейс, що надає API для побудови високоякісних додатків, такий як віджети, виявлення натискань, розпізнавання жестів, доступність, введення тексту. Він компонує дерево віджетів додатка в сцену.

Engine відповідає за растеризацію скомпонованих сцен і забезпечує низькорівневу реалізацію основних API Flutter, таких як графіка, розміщення

тексту, виконання Dart. Він взаємодіє з Framework через API dart:ui і інтегрується з конкретною платформою за допомогою API Embedder.

Embedder координує з операційною системою для доступу до сервісів, таких як поверхні рендерингу, доступність та введення. Він управляє циклом подій та надає платформозалежний API для інтеграції Embedder в додатки.

Runner об'єднує частини, що надаються платформозалежним API Embedder, в пакет додатка, який можна запустити на цільовій платформі. Частиною шаблону додатка, створеного за допомогою flutter create, володіє розробник додатка.

Ці компоненти співпрацюють для забезпечення розробки та роботи мобільних додатків на Flutter. Вони розділяють відповідальність та надають інструменти для побудови зручного та ефективного програмного забезпечення для кінцевих користувачів. [15]

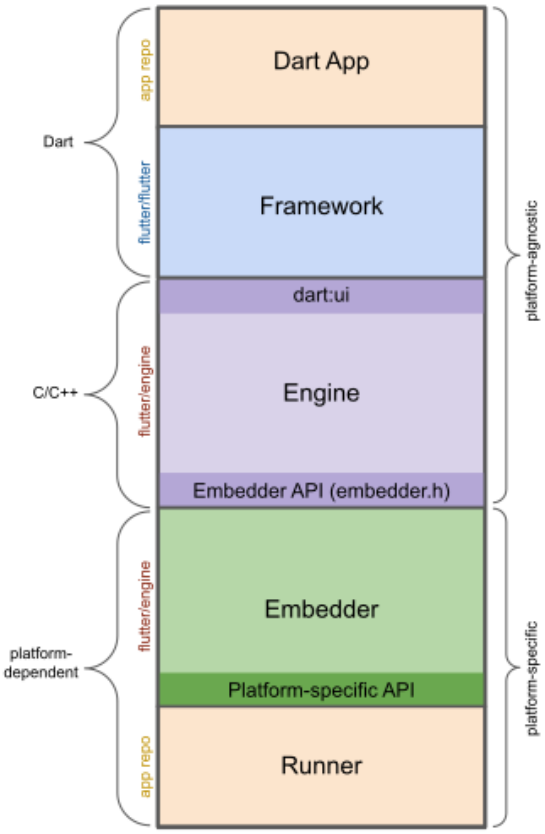


Рисунок 2.4 – Анатомія програми Flutter [15]

Принцип, що переважно застосовується у рендеринговому потоці Flutter, полягає в тому, що просте — це швидко. У Flutter існує простий потік обробки даних, що надходять до системи (рис. 2.5).

При розробці програмного забезпечення на Flutter великий акцент робиться на оптимізації швидкодії. Це досягається завдяки прямолінійному потоку обробки даних від користувача до відображення на екрані.

Початковим кроком у цьому потоці є отримання користувацького вводу, такого як натискання на екран або введення тексту. Flutter ефективно обробляє цей ввід та передає його далі для подальшої обробки.

Після отримання введення дані переходять до рівня логіки додатку, де вони обробляються відповідно до бізнес-логіки програми. Це може включати валідацію даних, розрахунки або інші операції, які потрібно виконати.

Після обробки на рівні логіки додатку дані передаються до рівня відображення, де вони рендеряться на екрані з використанням графічного процесора (GPU). Flutter використовує оптимізований підхід до взаємодії з GPU, що дозволяє досягти високої продуктивності та забезпечити швидке відображення на екрані.

Такий простий та ефективний потік обробки даних дозволяє Flutter забезпечити швидке та плавне взаємодію з користувачем, що є ключовим аспектом успішної розробки мобільних додатків.

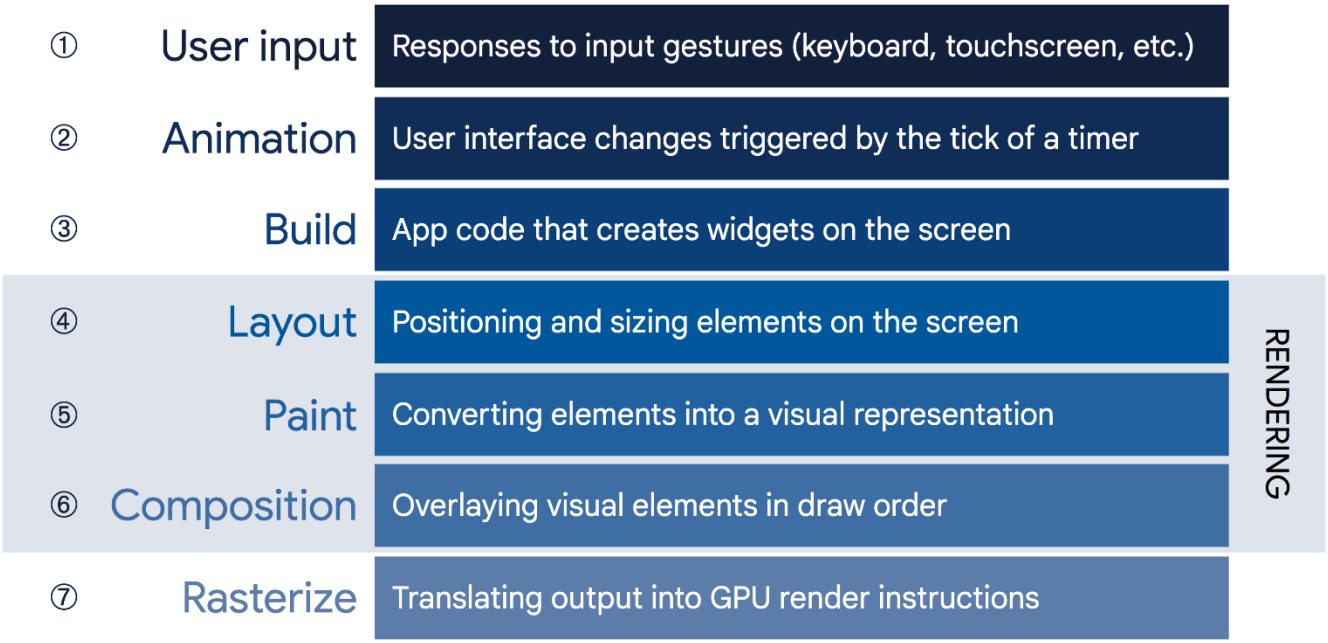


Рисунок 2.5 – Механізм передачі даних у Flutter [15]

На етапі створення Flutter перетворює віджети, виражені в коді, у відповідне дерево елементів, з одним елементом для кожного віджета (рис. 2.6). Кожен елемент представляє конкретний екземпляр віджета в заданому місці ієрархії дерева. Існує два основних типи елементів:

- ComponentElement, хост для інших елементів.
- RenderObjectElement, елемент, який бере участь у фазах макета або малювання.

На елемент для будь-якого віджета можна посилатися через його BuildContext, який є маркером розташування віджета в дереві. Це contexty виклику функції, наприклад Theme.of(context), і надається методу build()як параметр.

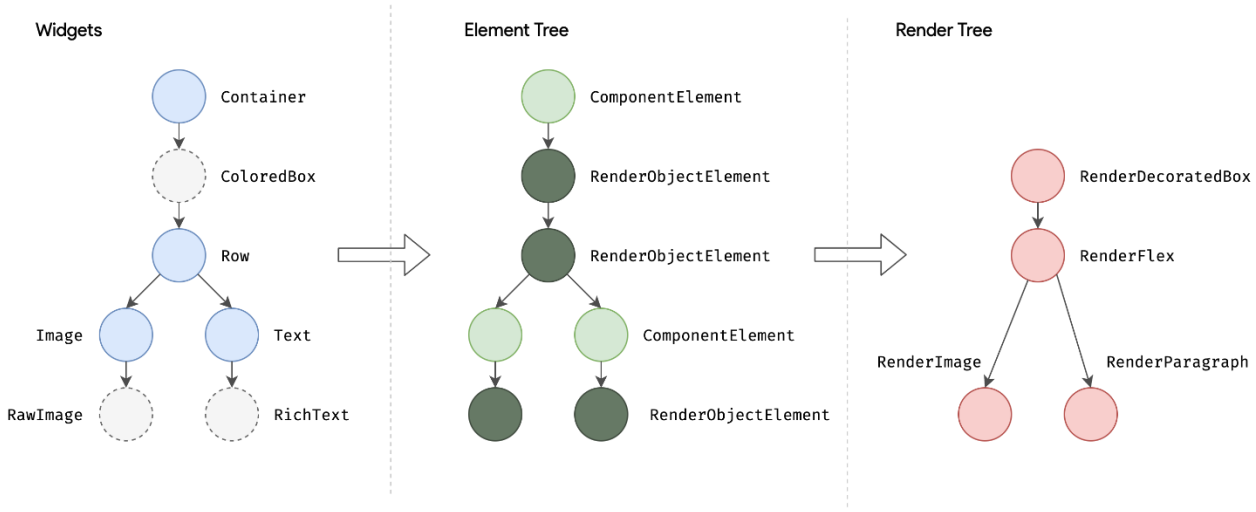


Рисунок 2.6 – Перетворення віджетів на дерево елементів з подальшою конвертацією у рендерну структуру [15]

Оскільки віджети є незмінними, включно з батьківським/дочірнім зв’язком між вузлами, будь-яка зміна дерева віджетів (наприклад, зміна Text('A')на Text('B')в попередньому прикладі) призводить до повернення нового набору об’єктів віджетів. Але це не означає, що базове представлення має бути перебудовано. Дерево елементів є постійним від кадру до кадру, тому відіграє вирішальну роль у продуктивності, дозволяючи Flutter діяти так, ніби ієрархія віджетів є повністю одноразовою, одночасно кешуючи її базове представлення. Переглядаючи лише змінені віджети, Flutter може перебудувати лише ті частини дерева елементів, які потребують переконфігурації. [15]

Для мобільних та настільних додатків Flutter дозволяє викликати власний код через канал платформи, що є механізмом зв'язку між вашим кодом на Dart та платформозалежним кодом хост-додатка (рис. 2.7). Шляхом створення загального каналу (інкапсулюючи назву та кодек), можна надсилати та отримувати повідомлення між Dart та компонентом платформи, написаним на мові, такій як Kotlin або Swift. Дані серіалізуються з типу Dart, такого як Map, у стандартний формат, а потім десеріалізуються в еквівалентне представлення в Kotlin (наприклад, HashMap) або Swift (наприклад, Dictionary).

Цей механізм комунікації дозволяє зручно обмінюватися даними між різними компонентами додатка, реалізованими на різних мовах програмування, та забезпечує єдність взаємодії між ними. Використання каналу платформи у Flutter дозволяє розширювати функціональність додатка, використовуючи мови програмування, що найкраще відповідають конкретним потребам та особливостям цільової платформи.

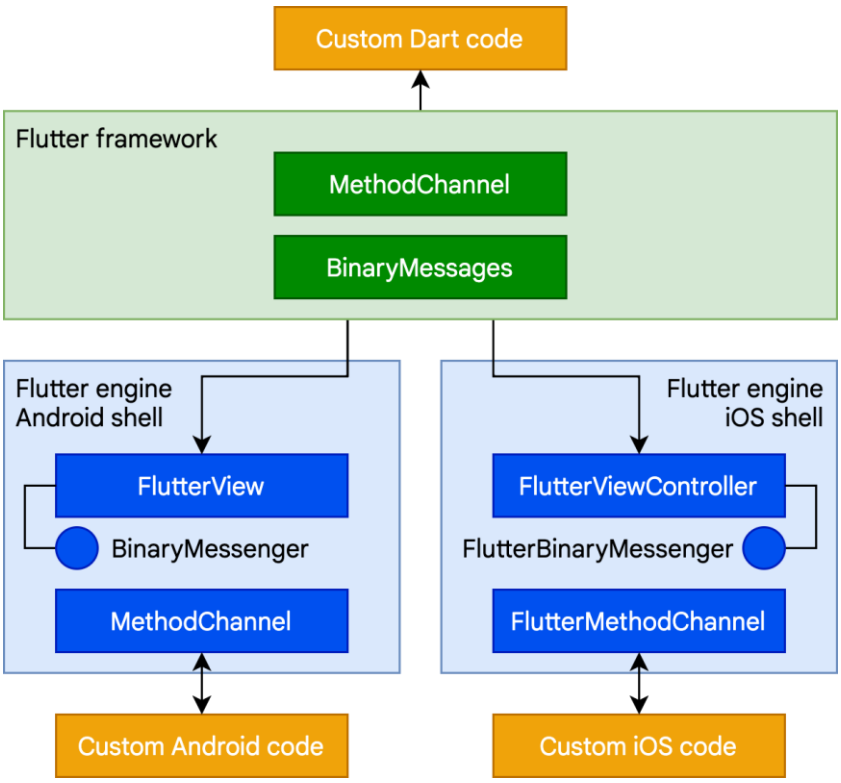


Рисунок 2.7 – Канали платформи Flutter [15]

Flutter - зручний інструмент для розробки мобільних додатків з кількох причин: він дозволяє створювати додатки для різних платформ, надає можливість швидкої перевірки змін завдяки гарячому перезавантаженню, має потужні інструменти для реалізації інтерфейсу користувача, легко інтегрується з платформеною функціональністю через платформенні канали та користується підтримкою великої та активної спільноти розробників.

2.2 Способи паралельної розробки користувацького інтерфейсу і програмного забезпечення

Паралельна розробка користувацького інтерфейсу (UI) і програмного забезпечення (ПЗ) є важливою стратегією для ефективного процесу розробки програмного забезпечення. У контексті Flutter, цей підхід набуває особливого значення, оскільки цей фреймворк дозволяє розробникам створювати якісні мобільні додатки для різних платформ. Для паралельної розробки UI та ПЗ у Flutter існують кілька підходів.

По-перше, розробники можуть використовувати підхід "складання" (composition), де робота над UI та ПЗ відбувається паралельно, і взаємодія між ними здійснюється через визначені контракти. Це дозволяє різним командам або розробникам працювати над окремими частинами додатка без взаємних блокувань.

Другий підхід - використання спеціальних інструментів для розробки інтерфейсу, таких як Flutter Inspector або DevTools, які дозволяють розробникам маніпулювати інтерфейсом в режимі реального часу, навіть під час роботи над логікою програми. Це спрощує відладку та взаємодію між командами розробників.

Третій підхід - використання інструментів контролю версій, таких як Git, для керування кодом інтерфейсу та програмною логікою. Це дозволяє розробникам працювати над різними гілками одночасно, а потім об'єднати зміни в єдину кодову базу.

Однак важливо пам'ятати про координацію між командами розробників, щоб уникнути конфліктів та забезпечити сумісність між UI та ПЗ. Крім того, важливо забезпечити регулярну зворотну зв'язок та спільний кодовий огляд для забезпечення якості та стабільності додатку.

Паралельна розробка UI та ПЗ у Flutter є можливою завдяки різноманітним підходам та інструментам, але вимагає уваги до координації та спільної роботи команди розробників.

Flutter Flow - це інструмент, який надає можливість візуальної розробки інтерфейсу користувача для додатків, що базуються на Flutter (рис. 2.8). Він дозволяє розробникам створювати і редагувати UI за допомогою перетягування та відпускання віджетів, налаштовувати їх властивості та взаємодіяти з ними безпосередньо на екрані. Flutter Flow спрощує процес розробки інтерфейсу та дозволяє швидко створювати прототипи та макети додатків без необхідності написання коду. Даний інструмент допомагає розробникам швидше відобразити свої ідеї та експериментувати з дизайном, що дозволяє зосередитися на кращому варіанті взаємодії з користувачем.

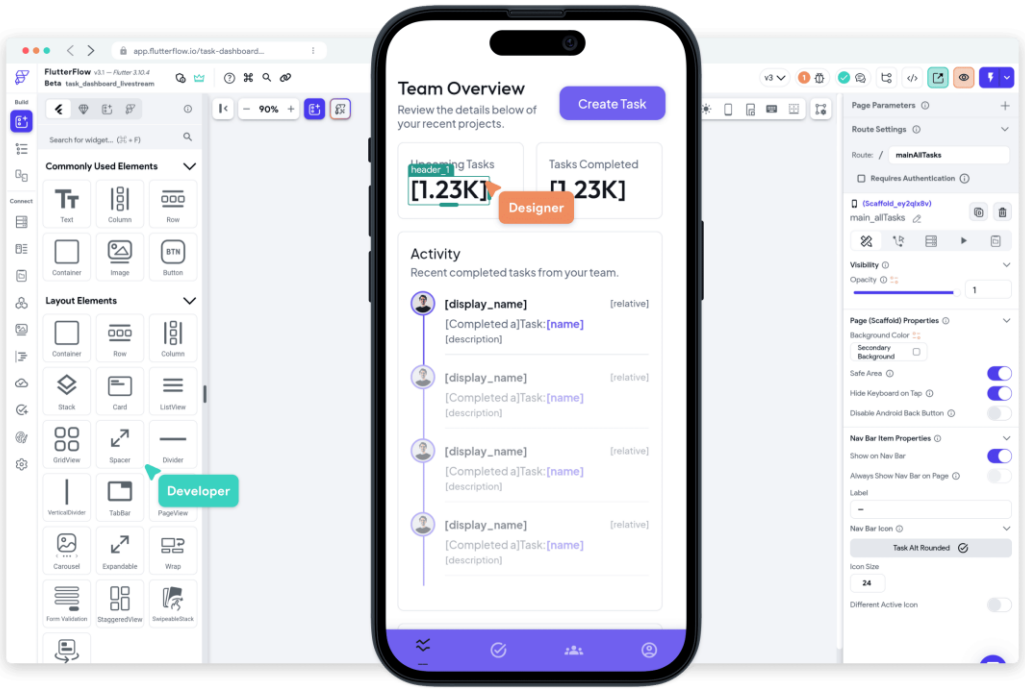


Рисунок 2.8 – Інструмент Flutter Flow [20]

FlutterFlow — це візуальне середовище розробки для створення рідних мобільних і веб-додатків. FlutterFlow допомагає швидше створювати додатки без шкоди для якості чи функцій. [19]

Переведення дизайну у Flutter за допомогою інструменту Flutter Flow є ключовим етапом у процесі розробки мобільних додатків, що базуються на даному

фреймворку. Цей процес складається з кількох послідовних етапів, кожен з яких відіграє важливу роль у створенні функціонального та естетично збалансованого додатку.

Перший етап цього процесу полягає у запуску інструменту Flutter Flow, який забезпечує можливість візуальної розробки інтерфейсу користувача. Відкривши веб-сайт Flutter Flow та вибравши тип проєкту, розробник отримує доступ до інтерфейсу, де може почати роботу.

Другий етап передбачає додавання віджетів на макет екрану додатку. Використовуючи інструменти Flutter Flow, розробник може вибирати необхідні віджети та розташовувати їх на екрані відповідно до дизайну.

Третій етап цього процесу - налаштування властивостей віджетів. Розробник може редагувати різноманітні параметри віджетів, такі як розмір, колір, шрифт тощо, для досягнення бажаного естетичного вигляду.

Четвертий етап передбачає створення взаємодії між віджетами. Використовуючи функціонал Flutter Flow, розробник може додавати переходи, анімації та інші ефекти, що покращують користувацький досвід.

П'ятий етап - це перегляд результатів роботи. Розробник може переглянути попередній вигляд додатку, щоб оцінити, як він виглядає на різних пристроях та в різних режимах.

Фінальний шостий етап полягає у експорті згенерованого коду Flutter для подальшого використання у проєкті. Цей процес надає розробникам зручність та ефективність у створенні мобільних додатків, дозволяючи швидше та зручніше перевести концепційний дизайн у функціональний продукт.

FlutterFlow є ефективним інструментом для паралельної розробки користувацького інтерфейсу та програмного забезпечення мобільних додатків через його можливості візуальної розробки, швидкого створення макетів і віджетів, а також можливості безпосередньої взаємодії з елементами інтерфейсу без потреби в написанні коду. Це дозволяє розробникам ефективно використовувати час і ресурси, прискорюючи процес розробки та поліпшуючи якість додатку.

2.3 Аналіз ефективності використання Flutter для створення мобільного додатку

Flutter обирають тисячі розробників, команд та компаній у всьому світі через його переваги універсальності, швидкості розробки та високої продуктивності. Завдяки гнучкості фреймворка, розробники можуть створювати якісні та масштабовані додатки для різних платформ, використовуючи єдиний кодову базу. Крім того, активне співтовариство розробників, постійне оновлення функціоналу та підтримка від Google роблять Flutter привабливим вибором для реалізації ідей та проектів будь-якої складності.

Згідно останньої статистики Built With станом на травень 2024 року відомо про 56 685 веб-сайтів, які використовують Flutter, і ще 34 735 сайтів, які раніше використовували Flutter. Світова статистика використання Flutter показана на рис. 2.9. [21]

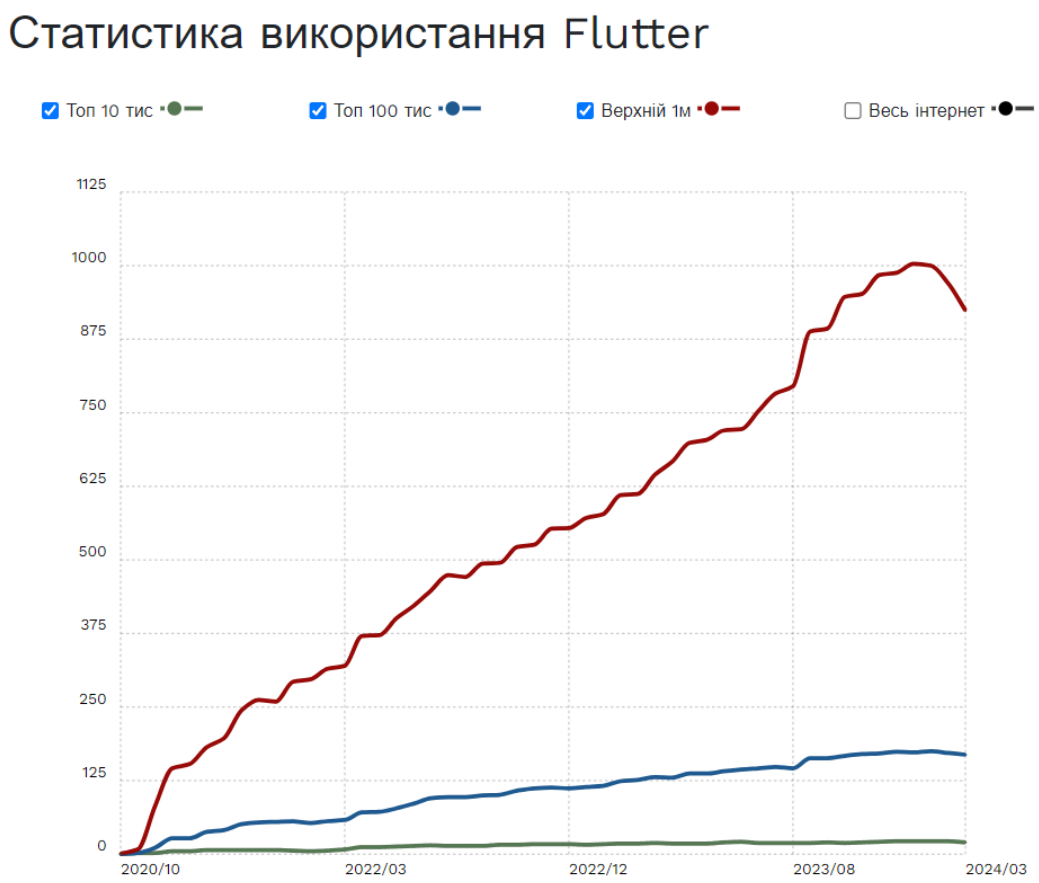


Рисунок 2.9 – Статистика використання Flutter розробниками всього світу [21]

Flutter надає розробникам можливість створювати крос-платформові додатки з високою продуктивністю та однаковим дизайном для різних пристроїв. Його гаряча перезавантаження та багатофункціональність роблять його привабливим інструментом для швидкої та ефективної розробки мобільних додатків.

Для визначення ефективності Flutter в контексті мобільної розробки було проведено порівняльний аналіз Flutter та React Native. Згідно статистики Google Trends станом на травень 2024 року Flutter стабільно викликає більше інтересу розробників, аніж React Native, проте обидва ці інструменти є затребуваними на ринку мобільних технологій. На рис. 2.10 представлена статистика популярності двох фреймворків. [23]

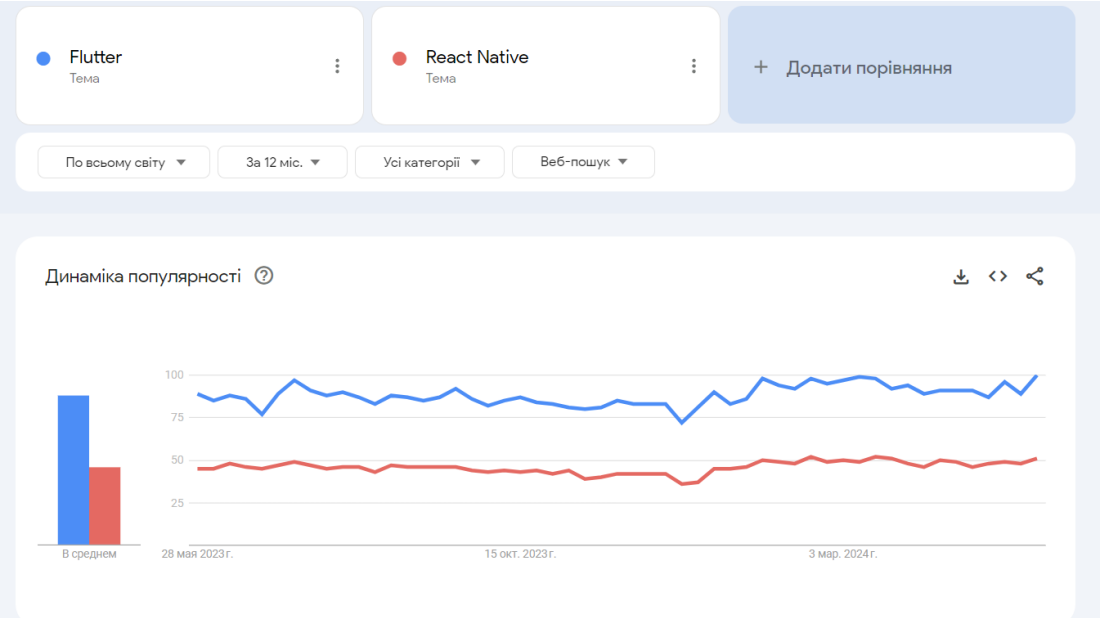


Рисунок 2.10 – Порівняння популярності Flutter та React Native [23]

Існують різні види продуктивності:

- Взаємодія з API телефону (доступ до фотографій, файлової системи, отримання GPS-місцезнаходження тощо).
- Швидкість візуалізації (плавність анімації, кількість кадрів за секунду під час зміни інтерфейсу або деякі ефекти інтерфейсу, які відбуваються в часі).
- Бізнес-логіка (швидкість математичних обчислень і маніпуляцій з пам'яттю. Цей тип продуктивності найбільш важливий для програм зі складною бізнес-логікою). [23]

Для визначення доцільності використання Flutter у контексті мобільної розробки було проведене порівняння Flutter, React Native та ще двох інструментів за вищевказаними трьома показниками продуктивності. Аналіз включає результати тестів продуктивності, які показують математичні обчислення числа Pi, реалізовані в нативному та міжплатформному підходах.

Дослідження інтенсивного використання ЦП за алгоритмом Гаусса-Лежандра для iOS виявило важливі відмінності у продуктивності різних мов програмування. (рис. 2.11) Відзначається, що Objective-C визнана найкращою мовою для розробки iOS, з урахуванням її високої швидкодії та ефективності. За результатами тестування встановлено, що Swift у 1,7 рази повільніший за Objective-C, що може впливати на продуктивність розробки та роботу додатків. Однак виявлено, що Flutter швидший за Swift на 15%, що може відкрити нові перспективи для розробки мобільних додатків. З іншого боку, React Native проявив себе у цьому тесті значно менш ефективно, будучи у 20 разів повільнішим за Objective-C, що може викликати обмеження у швидкості та продуктивності розроблених додатків.

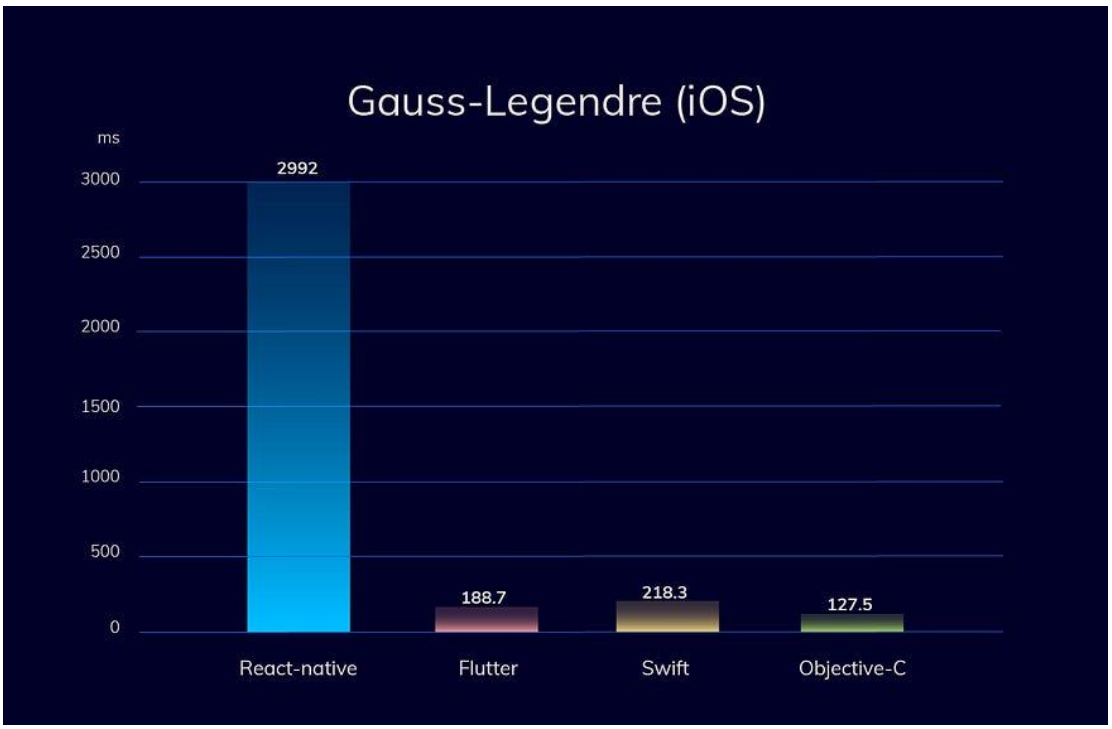


Рисунок 2.11 – Тест із інтенсивним використанням пам’яті (алгоритм Гауса–Лежандра) для iOS [23]

Дослідження ефективності мов програмування для розробки iOS-додатків при інтенсивному використанні ЦП на прикладі алгоритму Борвейна виявило значні відмінності у швидкодії. Objective-C залишається найкращим варіантом для розробки iOS-додатків завдяки своїй високій продуктивності. (рис. 2.12) За результатами тестування, Swift виявився у 1,9 рази повільнішим, ніж Objective-C, що свідчить про певні обмеження у продуктивності цієї мови. Ще більш повільним виявився Flutter, який у 5 разів поступається Swift, що може обмежувати його використання для завдань з високим навантаженням на ЦП. Найменш ефективним у цьому тестуванні виявився React Native, який більш ніж у 15 разів повільніший за Swift. Ці результати підкреслюють важливість вибору мови програмування з урахуванням вимог до продуктивності додатків. Вибір оптимальної технології для розробки є критичним для забезпечення ефективної роботи додатків, особливо у випадках інтенсивного використання обчислювальних ресурсів. Таким чином, для додатків з високим навантаженням на процесор Objective-C залишається незамінним інструментом, тоді як використання Flutter та React Native слід ретельно обмірковувати з урахуванням їх обмеженої продуктивності.

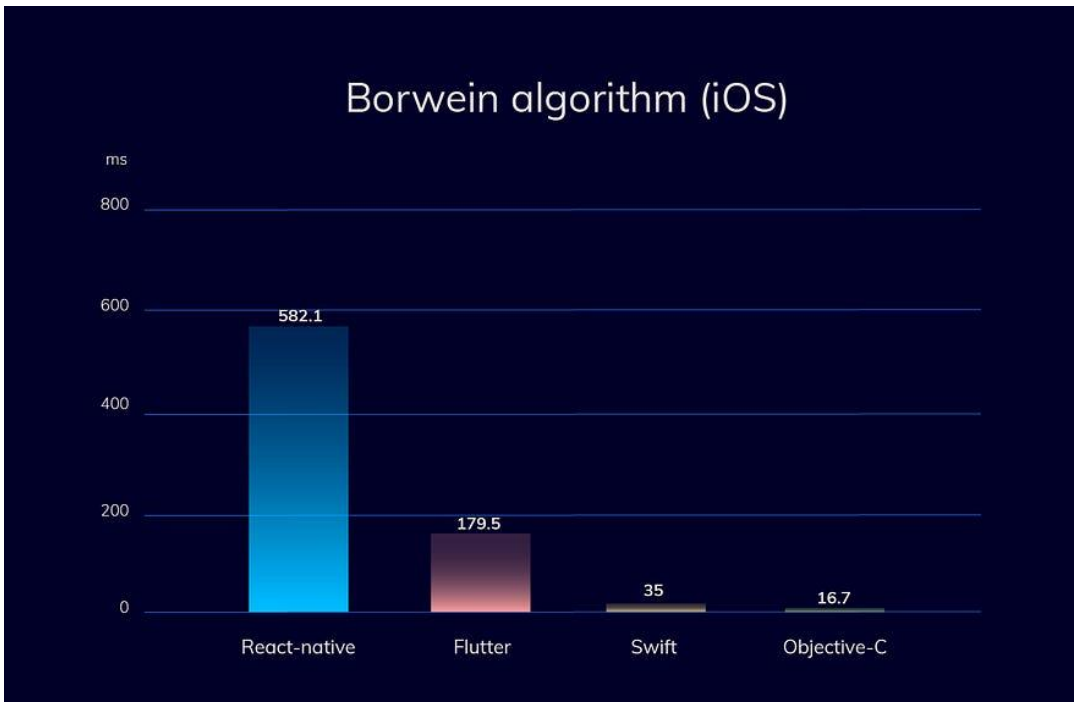


Рисунок 2.12 - Тест із інтенсивним використанням ЦП (алгоритм Борвейна) для iOS [23]

Дослідження ефективності мов програмування для розробки Android-додатків при інтенсивному використанні процесора, зокрема на прикладі алгоритму Гаусса-Лежандра, виявило суттєві відмінності у продуктивності. Java і Kotlin продемонстрували схожі показники продуктивності, що підтверджує їх статус найкращих варіантів для розробки Android-додатків (рис. 2.13). Обидві мови забезпечують високу швидкодію та ефективність, що є критично важливим для додатків з великим навантаженням на процесор. З іншого боку, Flutter виявився приблизно на 20% повільнішим, ніж нативні рішення, що може створити певні обмеження у використанні цього фреймворка для завдань, що потребують високої продуктивності. Особливо примітним є те, що React Native показав значно гірші результати, будучи приблизно в 15 разів повільнішим за нативні рішення. Це суттєве відставання в продуктивності може негативно впливати на роботу додатків, розроблених з використанням React Native, особливо в умовах інтенсивного використання процесора. Таким чином, для розробки Android-додатків, де важлива висока продуктивність, найкращим вибором залишаються Java та Kotlin. Водночас Flutter може бути використаний для менш ресурсомістких додатків, але слід враховувати його певні обмеження у швидкодії. Використання React Native вимагає ретельного обмірковування через значні втрати у продуктивності.

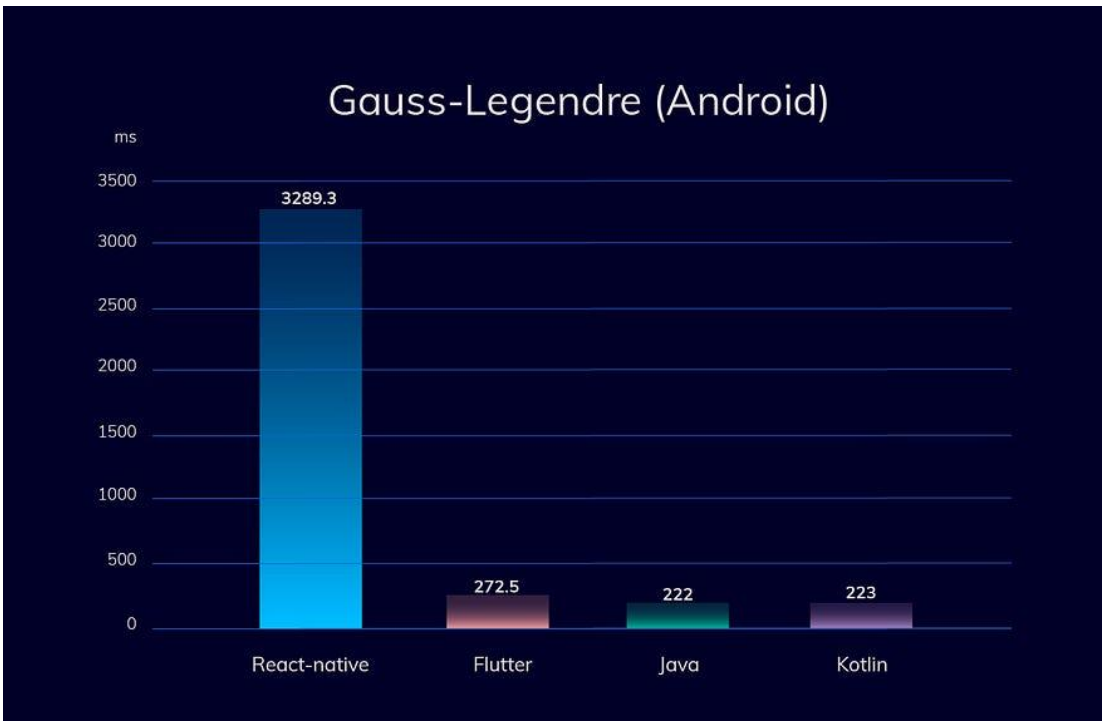


Рисунок 2.13 – Тест із інтенсивним використанням пам’яті (алгоритм Гаусса–Лежандра) для Android [23]

Дослідження ефективності мов програмування для розробки Android-додатків при інтенсивному використанні ЦП за алгоритмом Борвейна виявило суттєві відмінності у продуктивності (рис. 2.14). Java і Kotlin продемонстрували схожі показники продуктивності, підтверджуючи свій статус як найкращих варіантів для розробки Android-додатків. Ці мови забезпечують високу швидкодію та ефективність, що є критично важливим для додатків, які вимагають інтенсивного використання процесора. Дослідження показало, що нативні рішення, реалізовані за допомогою Java і Kotlin, у два рази швидші за Flutter, що вказує на певні обмеження використання Flutter для завдань з високими вимогами до продуктивності. Ще гірші результати продемонстрував React Native, який виявився приблизно в шість разів повільнішим за нативні рішення. Це значне відставання у продуктивності може негативно впливати на користувацький досвід і обмежувати можливості розробки складних та ресурсомістких додатків. Таким чином, для розробки Android-додатків, де важлива висока продуктивність, найкращим вибором залишаються Java та Kotlin. Використання Flutter може бути виправданим у випадках, коли пріоритетом є швидкий розвиток та крос-платформна підтримка, але слід враховувати його обмежену швидкодію. Використання React Native потребує особливої уваги через значні втрати у продуктивності, що робить його менш привабливим для додатків з інтенсивним використанням процесора.

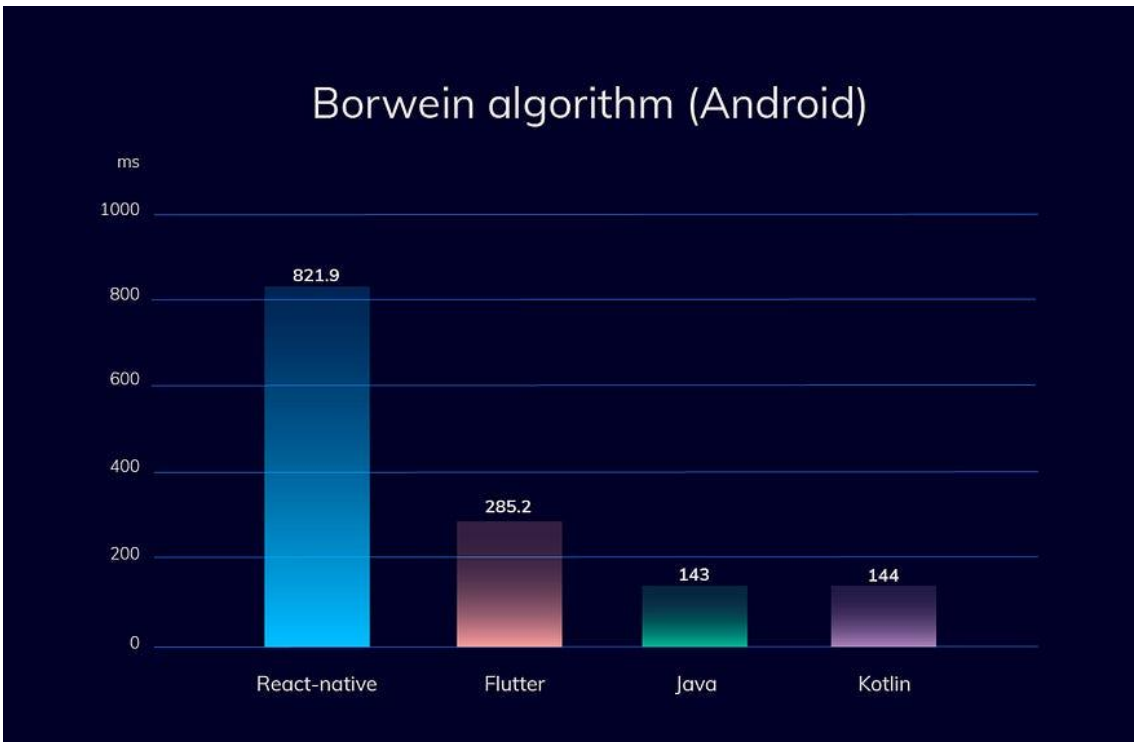


Рисунок 2.14 - Тест із інтенсивним використанням ЦП (алгоритм Борвейна) для Android [23]

У цьому дослідженні ефективності мобільних додатків було проведено тести на реальних фізичних пристроях: iPhone 6s з iOS 13.2.3 та Xiaomi Redmi Note 5 під управлінням Android 9.0. Метою було визначити продуктивність різних мов програмування та фреймворків при інтенсивному використанні процесора, використовуючи алгоритми обчислення чисел Пі Гаусса-Лежандра і Борвейна. Тести вимірювали продуктивність випускних збірок, оскільки збірки для налагодження можуть бути значно повільнішими. Кожен тест було проведено кілька разів, після чого розраховано середній результат для забезпечення точності. Алгоритми обчислювали число Пі 100 разів з точністю до 10 мільйонів цифр, що дозволило оцінити максимальне навантаження на процесор і пам’ять. Алгоритм Гаусса-Лежандра вимагав більше пам’яті, тоді як алгоритм Борвейна більше навантажував процесор. Результати показали, що нативні рішення на Java і Kotlin забезпечують найкращу продуктивність на Android, тоді як Swift та Objective-C були найефективнішими на iOS. Flutter показав продуктивність на 20% нижчу за нативні рішення на Android, але все ще кращу за React Native, який був у 15 разів повільнішим за нативні рішення. Ці дані підкреслюють важливість вибору відповідних інструментів розробки залежно від вимог до продуктивності і специфіки завдань.

На основі проведених досліджень можна зробити кілька ключових висновків щодо продуктивності кросплатформних програм. По-перше, не всі кросплатформні додатки працюють повільно, як це зазвичай вважається. Зокрема, програми, створені за допомогою Flutter, демонструють високу продуктивність, яка перевершує навіть продуктивність програм, написаних на Swift. Це свідчить про те, що Flutter є ефективним інструментом для розробки додатків, особливо тих, що

вимагають високої швидкодії. На iOS найкращими варіантами для створення надшвидких додатків залишаються Objective-C та Flutter, які забезпечують максимальну ефективність роботи додатків. Для додатків з високим навантаженням на обчислення Flutter також виявився оптимальним вибором для розробки як на Android, так і на iOS. Це пояснюється здатністю Flutter обробляти великі обсяги даних з високою швидкістю, що робить його конкурентоспроможним рішенням у порівнянні з нативними підходами. Крім того, можливість створення кросплатформних додатків на Flutter дозволяє розробникам скоротити час і витрати на розробку, зберігаючи при цьому високу продуктивність. Висока продуктивність Flutter у поєднанні з його гнучкістю та зручністю використання робить його привабливим вибором для сучасних розробників мобільних додатків. [23]

Результати порівняння Flutter з React Native показано на рис. 2.15.

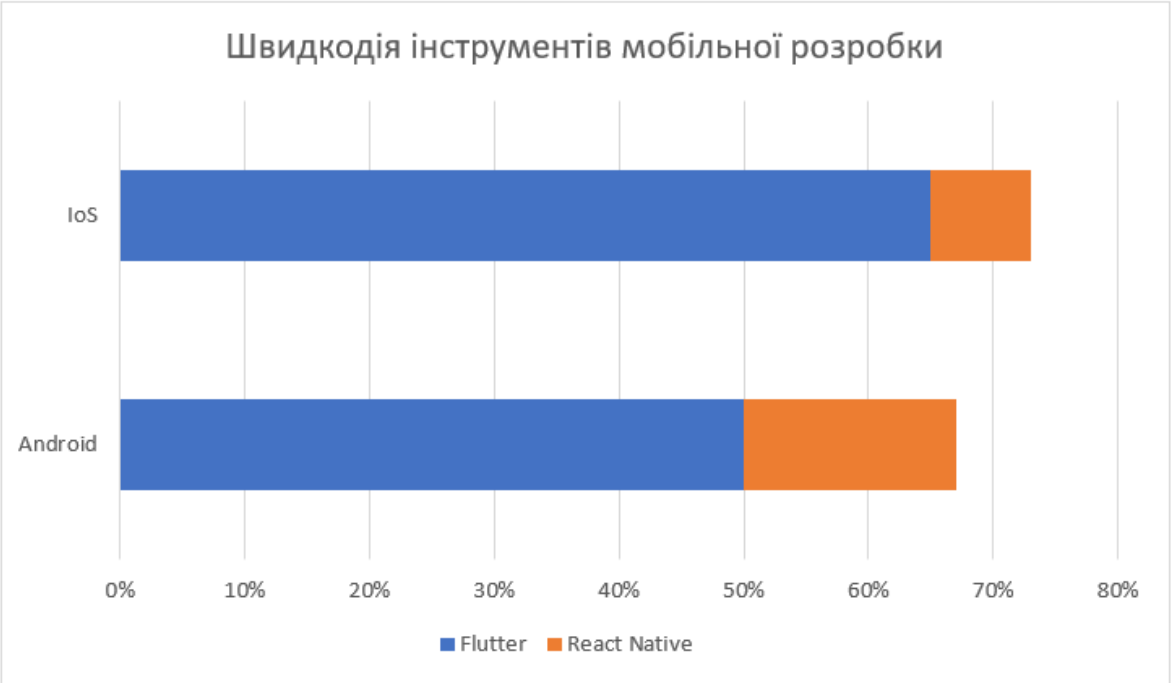


Рисунок 2.15 – Ефективність Flutter порівняно з іншими інструментами

Аналіз ефективності показав, що використання Flutter може стати розумним рішенням для багатьох проектів, що потребують високої продуктивності та кросплатформної підтримки.

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПЛАНУВАННЯ ЧАСУ У ФРЕЙМВОРКУ FLUTTER

3.1 Порівняння існуючих рішень для тайм-менеджменту

Тайм-менеджмент є критично важливим аспектом сучасного життя, і мобільні додатки, що допомагають керувати часом, набувають все більшої популярності. Серед них виділяються три провідні рішення: Todoist, Trello та Forest. Кожен з цих додатків має унікальні функціональні можливості, що забезпечують ефективне управління часом та завданнями.

Todoist – це потужний і водночас інтуїтивно зрозумілий інструмент для управління завданнями, який допомагає користувачам організовувати свої справи за допомогою списків та проектів (рис. 3.1). Додаток підтримує мультиплатформенний доступ, що дозволяє користувачам синхронізувати свої завдання на різних пристроях. Todoist пропонує такі функції, як нагадування, пріоритезація завдань, підзадачі та можливість делегування завдань іншим користувачам, що робить його ідеальним для особистого та командного використання. Завдяки своїй гнучкості та інтеграціям з іншими сервісами, такими як Google Calendar та Dropbox, Todoist стає незамінним інструментом для продуктивної роботи.

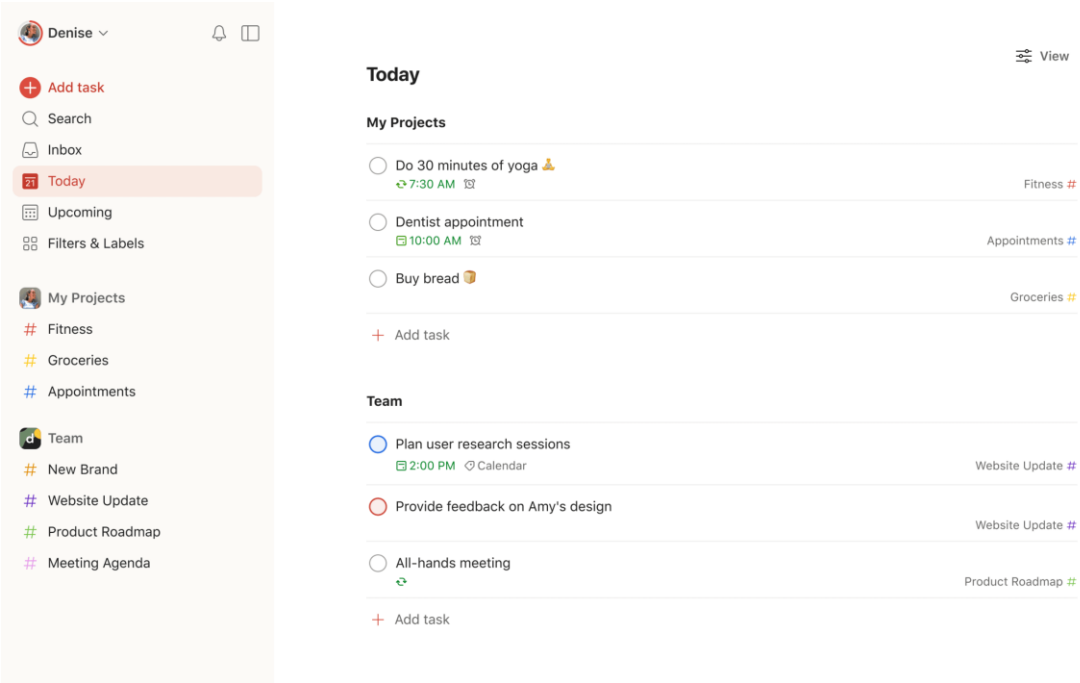


Рисунок 3.1 – Інтерфейс Todoist [24]

Trello – це візуальний інструмент для управління проектами, який використовує концепцію дошок, списків та карток для організації завдань (рис. 3.2). Кожен проект представлений у вигляді дошки, яка містить списки, а ті, в свою чергу, включають картки із завданнями. Такий підхід дозволяє легко бачити загальну картину проекту та відслідковувати прогрес виконання завдань. Trello

підтримує спільну роботу, дозволяючи кільком користувачам одночасно працювати над проектом, коментувати завдання та прикріплювати файли. Додаток також пропонує інтеграції з численними сервісами, такими як Slack, Google Drive та Jira, що робить його універсальним інструментом для управління проектами у різних сферах діяльності.

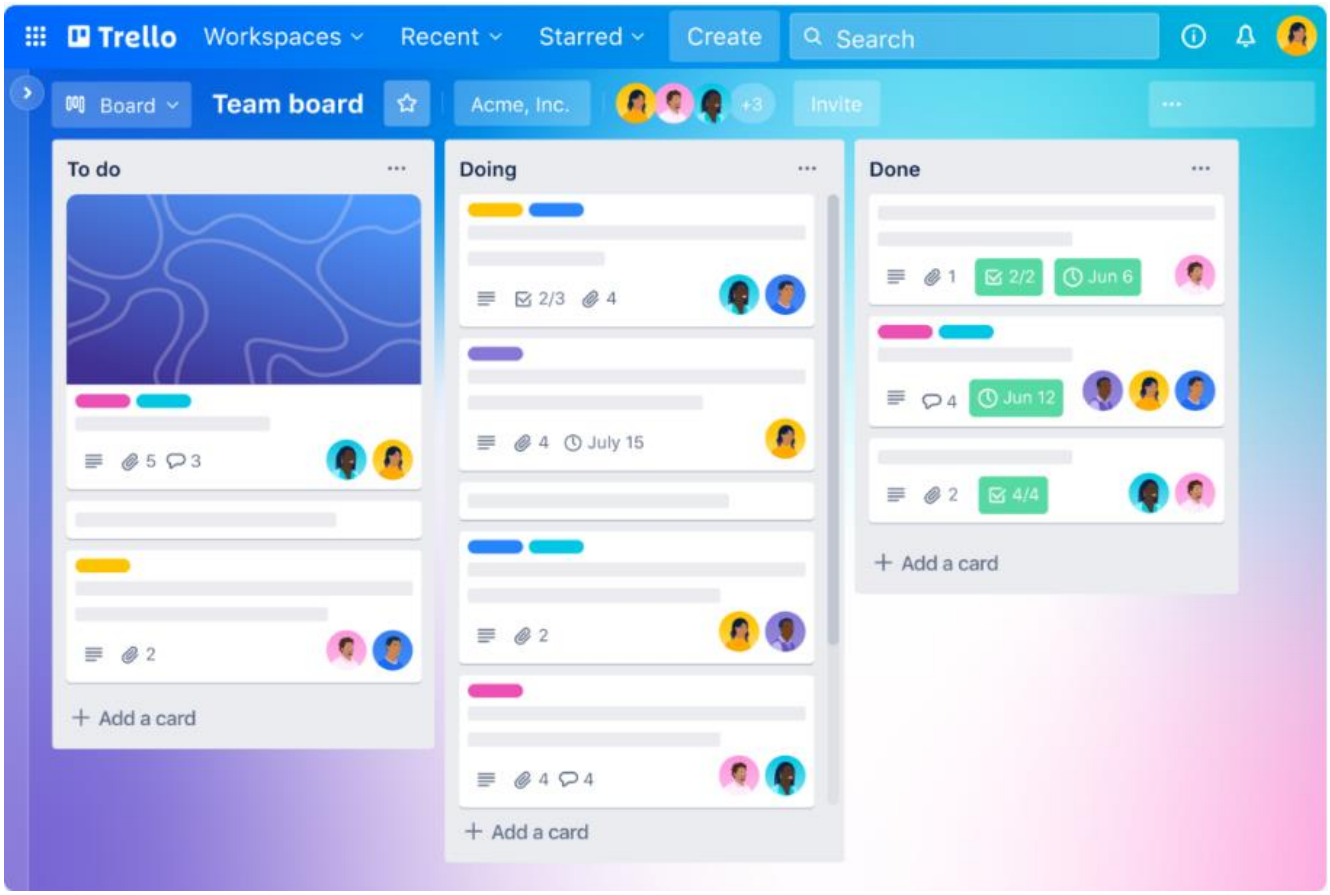


Рисунок 3.2 – Інтерфейс Trello [25]

Forest – це унікальний додаток для тайм-менеджменту, який допомагає користувачам зосередитися на своїх завданнях, використовуючи ігрові елементи (рис. 3.3). Коли користувач починає сесію роботи, він саджає віртуальне дерево, яке росте протягом заданого часу, якщо користувач не перериває роботу. Якщо ж користувач залишає додаток, дерево гине. Такий підхід стимулює користувачів залишатися зосередженими на завданнях і не відволікатися на телефон. Forest також дозволяє відстежувати прогрес, створювати цілі та аналізувати час, витрачений на різні активності. Крім того, за допомогою Forest користувачі можуть робити внесок у реальне озеленення планети, оскільки розробники додатку співпрацюють з організаціями, що займаються посадкою дерев.

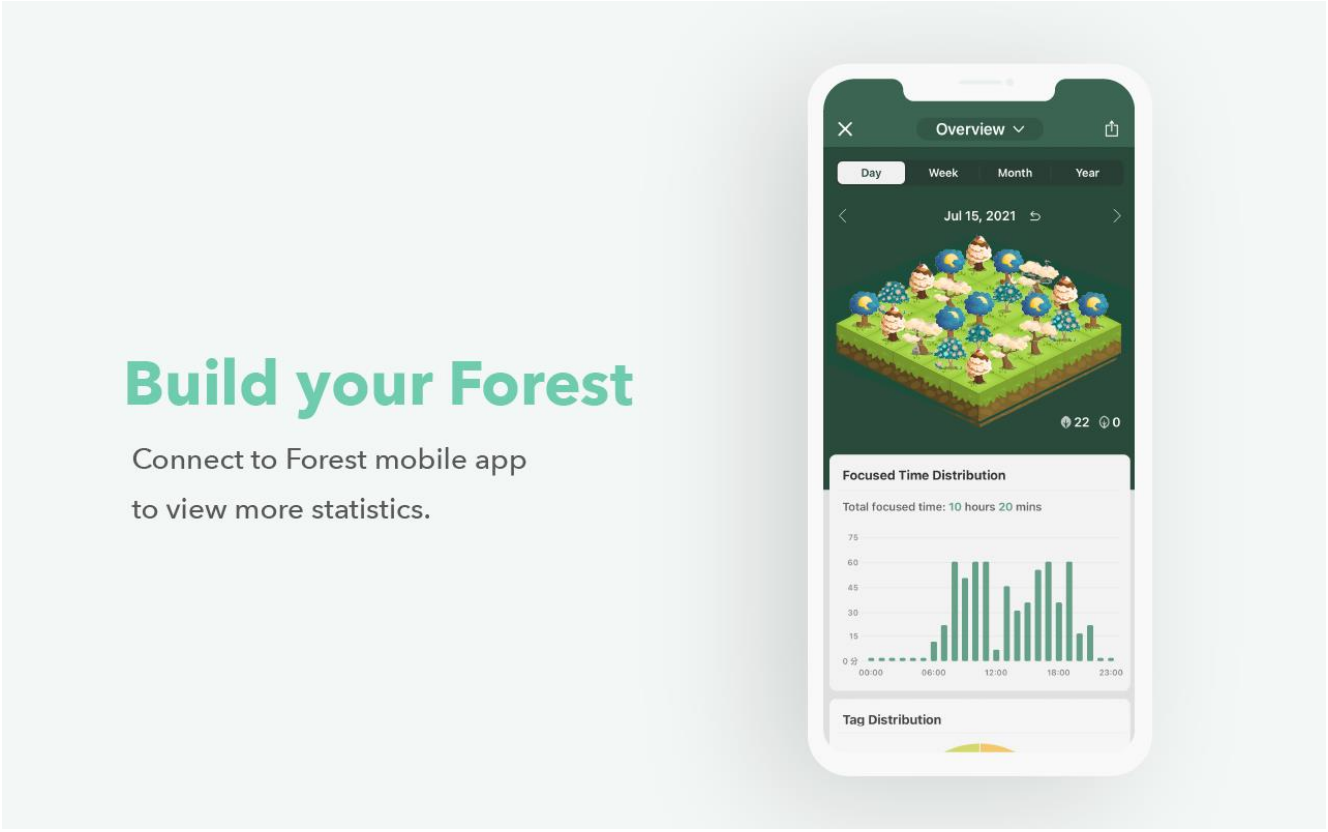


Рисунок 3.3 – Інтерфейс Forest [26]

Таким чином, кожен з цих додатків – Todoist, Trello та Forest – пропонує свої унікальні підходи до управління часом та завданнями, що дозволяє користувачам вибирати найкраще рішення відповідно до своїх потреб та стилю роботи. Незалежно від того, чи потрібно організувати список справ, управляти командними проектами чи зосередитися на завданнях, ці додатки забезпечують ефективні інструменти для підвищення продуктивності.

Зважаючи на аналіз трьох популярних додатків для тайм-менеджменту, можна виділити наступні базові вимоги до додатку для зручного та ефективного планування часу:

- Інтуїтивно зрозумілий інтерфейс.
- Гнучке управління завданнями.
- Мультиплатформенність та синхронізація даних між пристроями.
- Спільна робота та делегування завдань.
- Нагадування та повідомлення для дотримання дедлайнів.
- Інтеграція з іншими популярними сервісами.
- Мотивуючі елементи для стимулювання користувачів.
- Аналітика та звітність для оцінки продуктивності.

Врахування цих вимог при розробці додатку для планування часу допоможе створити інструмент, який буде зручним, ефективним та відповідатиме потребам широкого кола користувачів.

3.2 Користувацький інтерфейс додатку

В рамках виконання дипломної роботи було розроблено прототип мобільного додатку для забезпечення користувачам зручного та ефективного планування часу та організації робочих завдань. Додаток включає набір функціональних екранів, які дозволяють користувачам здійснювати різноманітні операції.

Користувацький інтерфейс відрізняється простотою, інтуїтивною зрозумілістю та зручністю. Прототип включає у себе такі екрани:

1. Вхід до акаунту або реєстрація (акаунти необхідні користувачам для збереження даних та легкого перемикання між девайсами)
2. Панель керування або Dashboard - екран, на якому знаходяться всі ключові функції та показано поточні задачі
3. Календар – розклад, який заповнив для себе користувач
4. Деталі дня – перелік поточних задач на день
5. Деталі задачі – додаткова інформація по поставлених задачах (наприклад, план доповіді або посилання на онлайн-зустріч)
6. Статистика та аналітика користувача – дані за певний період часу по продуктивності користувача, показники виконання задач та продуктивності.

Перший екран, «Вхід до акаунту», забезпечує користувачам можливість авторизуватися в системі, використовуючи свої облікові дані (рис. 3.1).

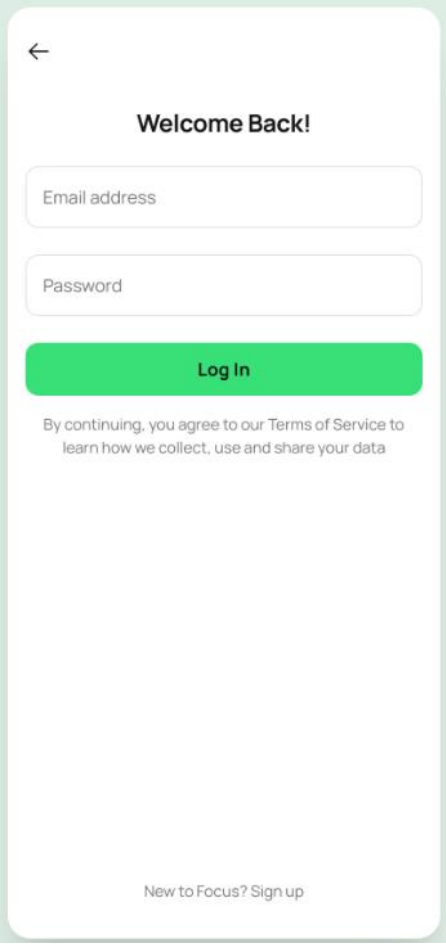


Рисунок 3.1 – Екран входу до акаунту у мобільному додатку

До основних екранів додатку входить «Dashboard», який надає користувачам огляд їхніх активних завдань, подій та інших важливих подій на даний момент (рис. 3.2).

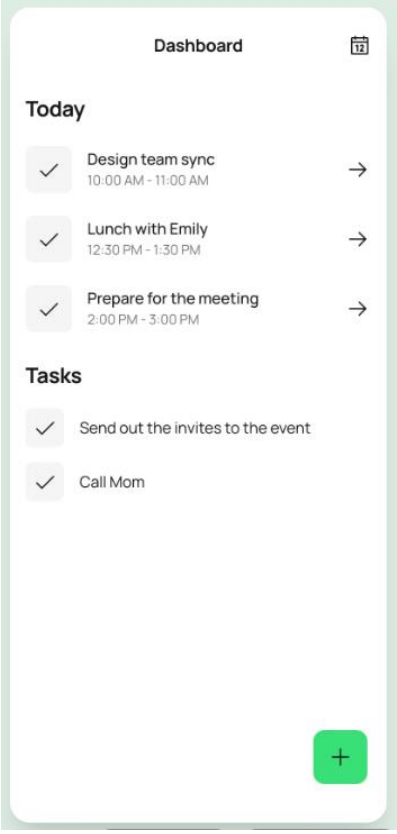


Рисунок 3.2 – Екран головної панелі мобільного додатку

На цьому екрані користувачі можуть швидко отримати огляд своєї активності, включаючи незавершені завдання, нагадування про майбутні події та інші важливі аспекти їхнього розкладу. Дашборд може відображати загальну кількість завдань, відкладених подій, поточні пріоритети або важливі дедлайни.

Цей екран є першим, що бачить користувач, коли заходить у додаток. Завдяки його наповненості такий підхід до першого враження від додатку допомагає забезпечити зручність користування та надає можливість користувачам швидко розпочати роботу.

«Календар» дозволяє користувачам переглядати свої заплановані події та завдання за допомогою календарного інтерфейсу та візуально оцінювати свій розклад (рис. 3.3).

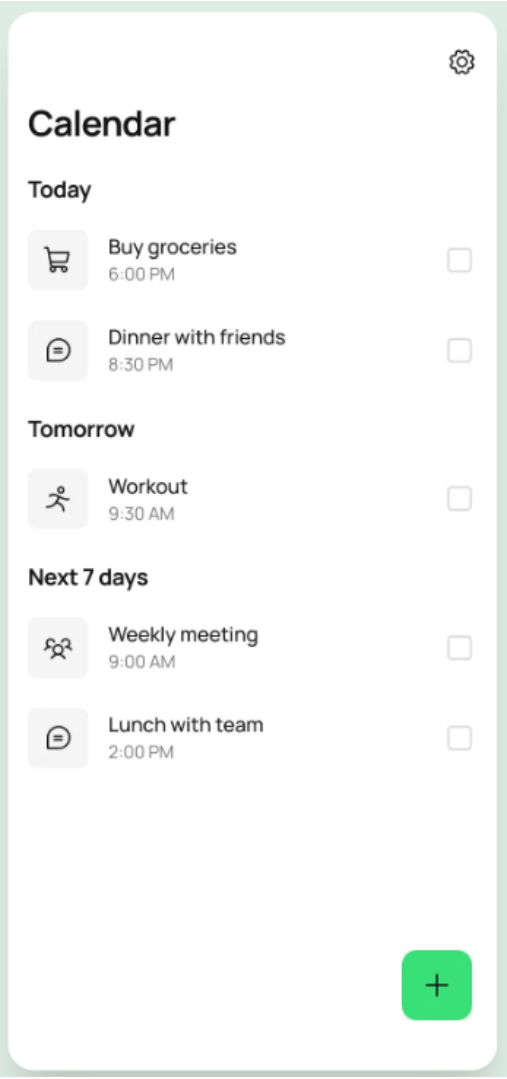


Рисунок 3.3 – Екран «Календар» мобільного додатку

У перспективі до календаря мобільного додатку можна додати функціональність фільтрації, яка дозволить користувачам швидко переглянути події за певний період часу або за категоріями. Також важливо покращити можливості інтерактивності, дозволяючи користувачам легко створювати нові події, редагувати існуючі та швидко переміщатися між різними датами. Додавання функцій нагадування та підтримки різних часових зон також може покращити користувацький досвід. Надання можливості інтеграції з іншими календарними сервісами або додатками також може розширити функціональність екрану "календар". Загалом, постійне вдосконалення цього екрану дозволить забезпечити користувачам більш ефективне управління своїм часом та подіями.

«Деталі дня» (рис. 3.4) та «Деталі задачі» (рис. 3.5) надають користувачам можливість отримати докладну інформацію про певний день або завдання, включаючи опис, дедлайни, сповіщення та іншу додаткову інформацію.

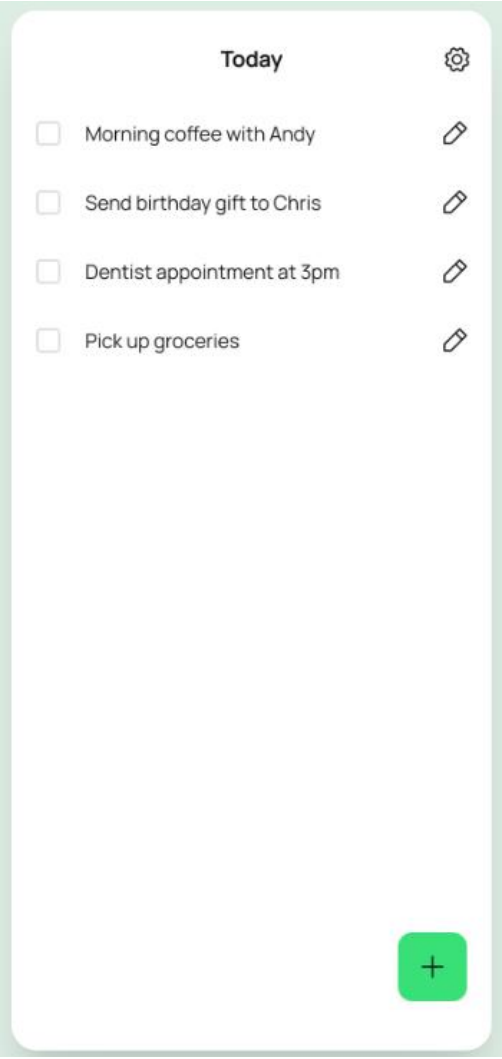


Рисунок 3.4 – Екран «Деталі дня» мобільного додатку

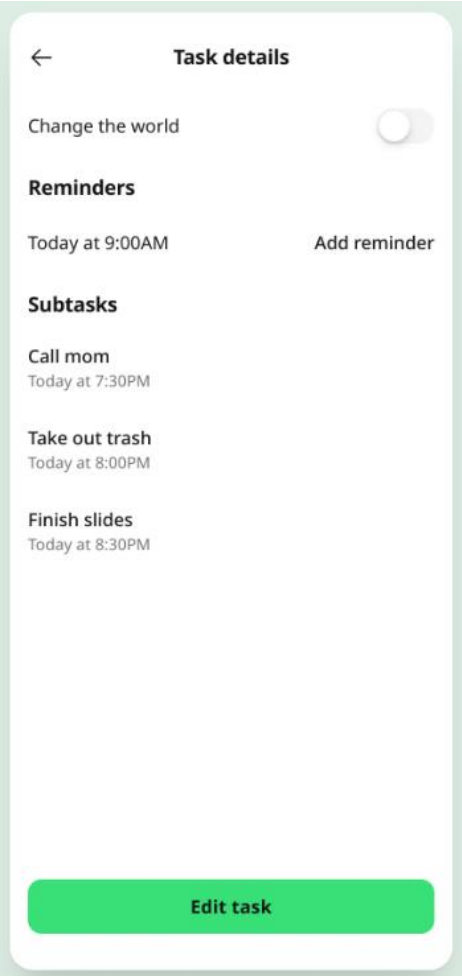


Рисунок 3.5 – Екран «Деталі задачі» мобільного додатку

«Статистика і аналітика по користувачу» надає користувачам звітні дані та аналітику щодо їхньої продуктивності, включаючи статистику виконаних завдань, час витрачений на різні види діяльності та інші метрики (рис. 3.6).

Цей екран дозволяє користувачам отримати уявлення про їхні звички, привички та тренди в управлінні часом, що може допомогти вдосконалити їхній робочий процес та забезпечити більш ефективне використання часу. Крім того, аналітична інформація, надана на цьому екрані, може слугувати джерелом мотивації для користувачів, допомагаючи їм ставити нові цілі та планувати свою діяльність більш обдумано. Наприклад, аналіз часу, витраченого на різні види діяльності, може допомогти користувачам визначити пріоритети та управляти своїми завданнями більш ефективно. Таким чином, екран статистики і аналітики користувача відіграє важливу роль у покращенні продуктивності та ефективності роботи, надаючи користувачам об'єктивну інформацію про їхню діяльність та допомагаючи їм досягти їхніх цілей.

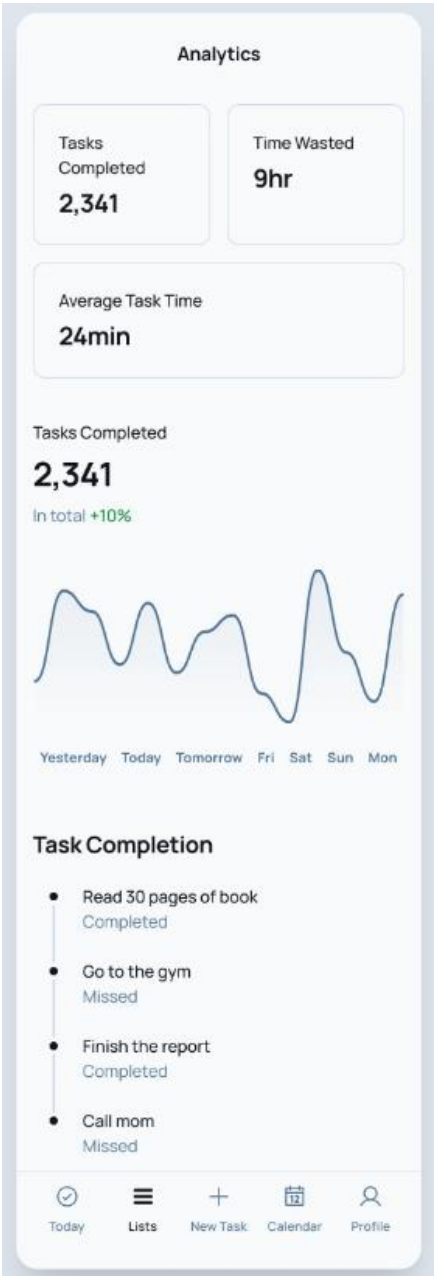


Рисунок 3.6 – Екран статистики і аналітики у мобільному додатку

Екран «Нова задача» дозволяє користувачам створювати нові завдання, заповнюючи різноманітну інформацію, таку як назва завдання, опис, термін виконання, пріоритет і т. д (рис. 3.7). Цей екран надає можливість швидко додавати нові завдання до розкладу користувача без зайвих зусиль.

Рисунок 3.7 – Екран «Нова задача» у мобільному додатку

Екран «Налаштування» дозволяє користувачам налаштовувати різні параметри додатку, такі як мова інтерфейсу, нагадування та повідомлення, категорії завдань, кольорова схема та інші налаштування, що підвищують зручність використання додатку та його придатність до індивідуальних потреб користувачів (рис. 3.8).

У перспективі екран налаштувань можна доповнити новими функціями та поліпшити існуючі, забезпечуючи більшу зручність та гнучкість для користувачів. Покращення можуть включати додавання додаткових параметрів, вдосконалення інтерфейсу користувача, інтеграцію з іншими додатками, поліпшення безпеки та конфіденційності, а також підтримку різних типів пристроїв та роздільної здатності екрану. Такі зміни сприятимуть більш зручному та корисному використанню додатку користувачами.

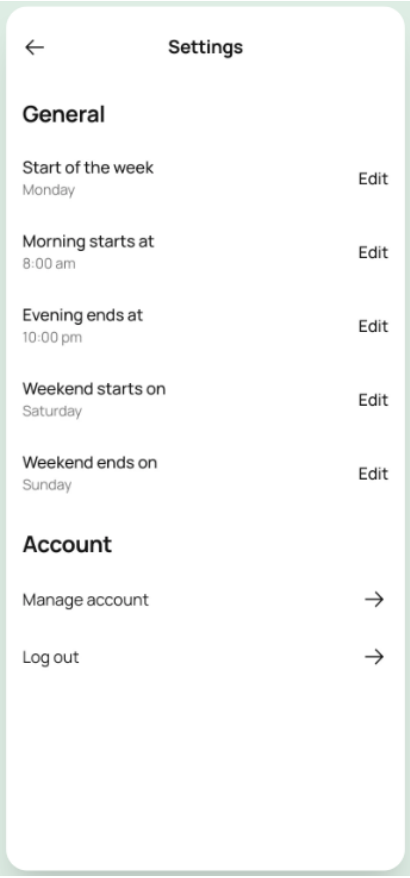


Рисунок 3.8 – Екран налаштувань мобільного додатку

Кожен екран додатку ретельно розроблений з урахуванням принципів дизайну та найкращих практик розробки програмного забезпечення, щоб забезпечити користувачам зручний та інтуїтивно зрозумілий інтерфейс.

3.3 Програмне забезпечення додатку

Архітектура програмного коду мобільного додатку базується на принципах чистої архітектури та використанні патерну MVVM (Model-View-ViewModel), що забезпечує високу модульність, розширюваність та тестованість додатку. Основні компоненти архітектури:

- Model (Модель): Відповідає за представлення даних та бізнес-логіку. Реалізує роботу з базою даних, API-запити та обробку даних.
- View (Представлення): Представлене віджетами (Widgets) у Flutter, які відображають інформацію користувачу та обробляють його дії.
- ViewModel (Проміжний шар): Відповідає за обробку бізнес-логіки та підготовку даних для відображення. Використовує паттерн спостерігача (Observer pattern) для оновлення інтерфейсу користувача.

У процесі розробки мобільного додатку використовуються різні інструменти, які допомагають розробникам ефективно створювати і налагоджувати програмне забезпечення. Основні компоненти інструментарію включають: Flutter SDK, Dart, Android Studio або IntelliJ IDEA з плагіном Flutter та Firebase.

Flutter SDK надає розробникам все необхідне для розробки мобільних додатків з використанням Flutter framework. Він включає в себе бібліотеки, документацію та інші корисні ресурси, які допомагають розробникам у кожному етапі процесу розробки.

Мова програмування Dart - це мова програмування, яка використовується для написання коду додатку. Вона має простий і зрозумілий синтаксис, що сприяє швидкому розвитку та підтримці програмного забезпечення.

Android Studio або IntelliJ IDEA з плагіном Flutter надають зручний і потужний набір інструментів для розробки та налагодження Flutter додатків. Вони мають широкий набір функцій, таких як автоматичне завершення коду, візуальні редактори та можливості налагодження, які спрощують процес розробки.

Firebase - це мобільна платформа розробки від Google, яка надає різноманітні інструменти для створення мобільних та веб-додатків. Сервіси Firebase включають в себе аналітику, зберігання даних в реальному часі, автентифікацію користувачів та інші функції, які допомагають розробникам створювати потужні та ефективні додатки.

Використання цих інструментів дозволяє створити високоякісний та ефективний мобільний додаток з використанням Flutter, забезпечуючи швидкий та надійний процес розробки.

UML-діаграма користувацького шляху у мобільному додатку показана на рис. 3.9.

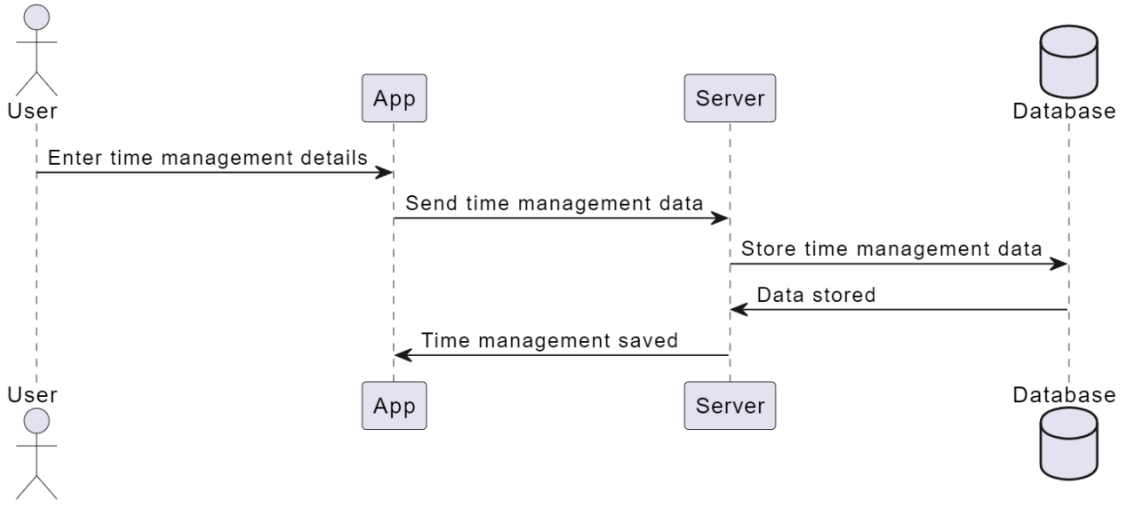


Рисунок 3.9 – Користувацький шлях у мобільному додатку для тайм-менеджменту

UML-діаграма ілюструє взаємодію користувача (User) з мобільним додатком (App), сервером (Server) та базою даних (Database) під час введення даних у систему для управління часом.

Спочатку користувач (User) вводить дані для управління своїм часом у мобільному додатку (App). Після цього, додаток (App) надсилає введені дані на сервер (Server) для подальшого збереження. Сервер (S) передає дані до бази даних (Database), де вони зберігаються. Після успішного збереження даних, сервер (Server) повідомляє мобільний додаток (App) про успішне збереження.

Така візуалізація ілюструє кроки, які виконуються в процесі введення та збереження даних у системі для управління часом, та показує послідовність взаємодій між різними компонентами системи.

Рис. 3.10 відображає алгоритм роботи програми мобільного додатку для управління задачами і часом. В основі лежать функції додавання, редагування або видалення задач у календарі, дані ж зберігаються у базі даних, до якої програма звертається із запитами. Передбачена автоматична синхронізація з сервером, тому додаток повинен мати постійний доступ у інтернет для збереження цілісності даних.

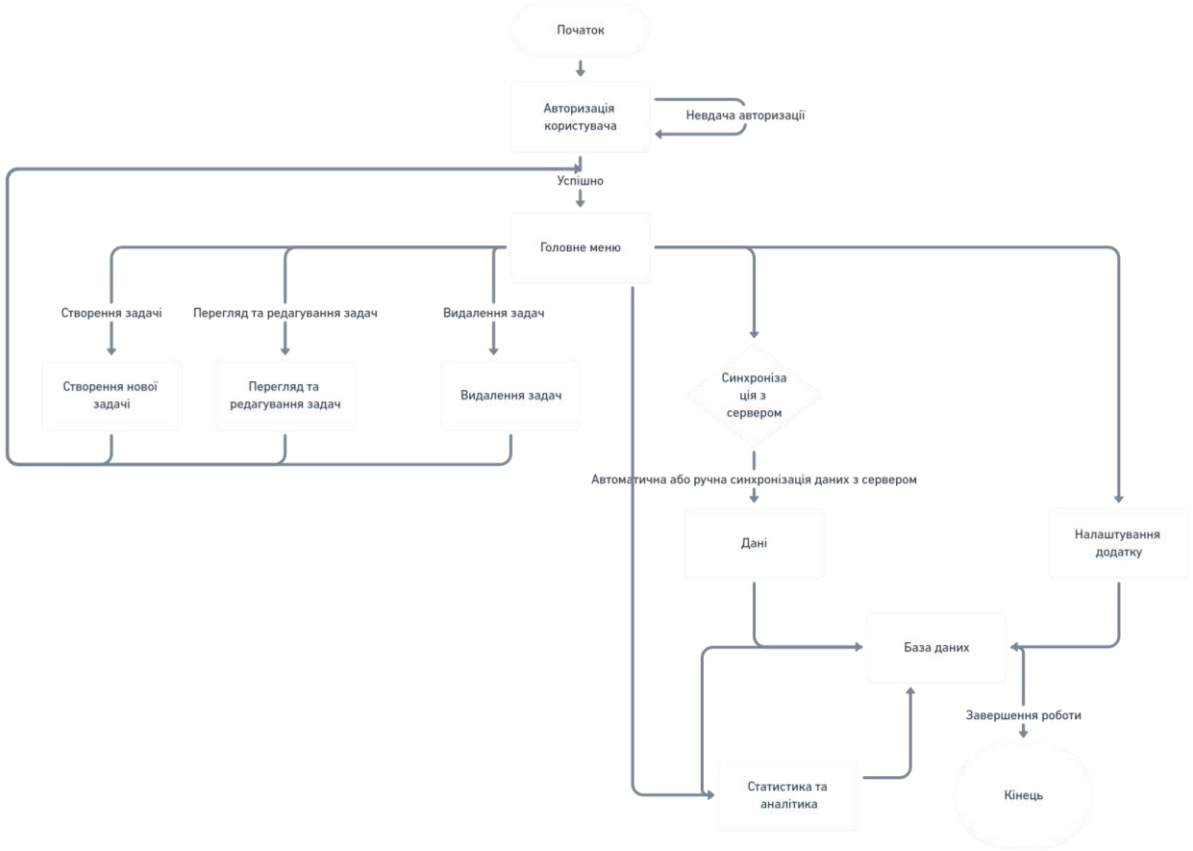


Рисунок 3.10 – Алгоритм роботи програмного забезпечення додатку

Серед основних функцій прорамного забезпечення створені такі:

- Авторизація користувача
- Відображення головного меню
- Створення нової задачі
- Перегляд та редагування задач
- Видалення задач
- Синхронізація з сервером
- Генерація статистики та аналітики
- Налаштування додатку
- Завершення роботи

Функція authenticateUser() (рис. 3.11) здійснює перевірку введених користувачем логіна та паролю. Вона підключається до бази даних, отримує дані

користувача та порівнює їх з введеними даними. Якщо дані збігаються, функція повертає true, що означає успішну авторизацію, інакше повертає false.

```
Future<bool> authenticateUser(String username, String password) async {
  // Підключення до бази даних та перевірка даних користувача
  var user = await database.getUser(username);
  if (user != null && user.password == password) {
    // Успішна авторизація
    return true;
  }
  // Невдала авторизація
  return false;
}
```

Рисунок 3.11 – Функція authenticateUser() для авторизації користувача

Функція displayMainMenu() (рис. 3.12) виводить на екран головне меню додатку з доступними функціями. Вона отримує вибір користувача, використовуючи функцію getUserInput, та виконує відповідну дію на основі вибору.

```
void displayMainMenu() {
  // Відображення головного меню з доступними функціями
  print("1. Create New Task");
  print("2. View Task List");
  print("3. Sync with Server");
  print("4. View Statistics");
  print("5. Configure Settings");
  print("6. Exit");
  // Обробка вибору користувача
  int choice = getUserInput();
  switch (choice) {
    case 1:
      createNewTask();
      break;
    case 2:
      displayTaskList();
      break;
    case 3:
      syncWithServer();
      break;
    case 4:
      generateStatistics();
      break;
    case 5:
      configureSettings();
      break;
    case 6:
      exitApplication();
      break;
    default:
      print("Invalid choice. Please try again.");
  }
}
```

Рисунок 3.12 - Функція displayMainMenu для виводу головного меню

Функція createNewTask() (рис. 3.13) дозволяє користувачеві створити нову задачу, запитуючи в нього необхідні дані, такі як назва, опис та дедлайн задачі. Після отримання цих даних, вона зберігає нову задачу в базі даних за допомогою методу database.saveTask.

```
Future<void> createNewTask() async {
  // Отримання деталей нової задачі від користувача
  String taskName = getTaskNameFromUser();
  String taskDescription = getTaskDescriptionFromUser();
  DateTime taskDeadline = getTaskDeadlineFromUser();

  // Створення нової задачі
  Task newTask = Task(name: taskName, description: taskDescription, deadline: taskDeadline);
  // Збереження нової задачі у базі даних
  await database.saveTask(newTask);
  print("Task created successfully!");
}
```

Рисунок 3.13 – Функція createNewTask() для створення нової задачі

Функції displayTaskList() та editTask() (рис. 3.14) забезпечують можливість перегляду списку задач та редагування окремих задач. Перша функція отримує всі задачі з бази даних і відображає їх на екрані, а друга дозволяє редагувати деталі вибраної задачі та зберігає зміни в базі даних.

```
Future<void> displayTaskList() async {
  // Отримання списку задач з бази даних
  List<Task> tasks = await database.getTasks();
  for (Task task in tasks) {
    print(task);
  }
}

Future<void> editTask(int taskId, String newName, String newDescription, DateTime newDeadline) async
{
  // Отримання задачі з бази даних
  Task task = await database.getTask(taskId);
  // Редагування деталей задачі
  task.name = newName;
  task.description = newDescription;
  task.deadline = newDeadline;
  // Збереження змін у базі даних
  await database.updateTask(task);
  print("Task updated successfully!");
}
```

Рисунок 3.14 - Функції displayTaskList() та editTask() для відображення і редагування існуючих задач

Функція deleteTask() (рис. 3.16) дозволяє користувачеві видалити вибрану задачу з бази даних.

```
Future<void> deleteTask(int taskId) async {
  // Видалення задачі з бази даних
  await database.deleteTask(taskId);
  print("Task deleted successfully!");
}
```

Рисунок 3.16 - Функція deleteTask() для видалення задачі

Функція syncWithServer() (рис. 3.17) відповідає за синхронізацію даних додатку з сервером. Вона отримує задачі, які потребують синхронізації, з бази даних та відправляє їх на сервер.

```
Future<void> syncWithServer() async {
  // Отримання даних для синхронізації
  List<Task> tasks = await database.getTasksToSync();
  // Відправка даних на сервер
  await server.syncTasks(tasks);
  print("Data synchronized successfully!");
}
```

Рисунок 3.17 - Функція syncWithServer() для синхронізації з сервером

Функція generateStatistics() (рис. 3.18) обчислює та відображає статистичні дані, такі як загальна кількість задач та кількість виконаних задач, використовуючи дані з бази даних.

```
Future<void> generateStatistics() async {
  // Отримання даних з бази даних
  List<Task> tasks = await database.getTasks();
  // Обчислення статистики
  int totalTasks = tasks.length;
  int completedTasks = tasks.where((task) => task.isCompleted).length;
  // Відображення статистики
  print("Total tasks: $totalTasks");
  print("Completed tasks: $completedTasks");
}
```

Рисунок 3.18 - Функція generateStatistics() для показу аналітики

Функція configureSettings() (рис. 3.19) дозволяє користувачеві налаштовувати різні параметри додатку, такі як мова та нагадування. Вона зберігає ці налаштування для подальшого використання.

```
void configureSettings() {
  // Отримання нових налаштувань від користувача
  String language = getUserLanguagePreference();
  bool enableNotifications = getUserNotificationPreference();
  // Застосування нових налаштувань
  settings.language = language;
  settings.enableNotifications = enableNotifications;
  print("Settings updated successfully!");
}
```

Рисунок 3.19 - Функція configureSettings() для запуску і функціонування меню налаштувань додатку

Функція exitApplication() (рис. 3.20) завершує роботу програми, виводячи відповідне повідомлення та закриваючи додаток.

```
void exitApplication() {
  // Завершення роботи програми
  print("Exiting application...");
  exit(0);
}
```

Рисунок 3.20 - Функція exitApplication() для завершення роботи

Ці програмні функції забезпечують основну функціональність додатку для планування часу, дозволяючи користувачам ефективно управляти своїми задачами, синхронізувати дані з сервером та аналізувати свою продуктивність.

У мобільному додатку для планування часу використовуються Firebase Firestore для зберігання даних і обміну ними між базою даних та додатком. Firebase Firestore є хмарним рішенням для зберігання даних, що дозволяє забезпечити високу надійність і масштабованість, необхідну для сучасних мобільних додатків.

Основний алгоритм роботи з Firebase Firestore включає кілька ключових етапів:

1. Підключення до бази даних
2. Виконання запитів
3. Обробка результатів
4. Закриття з'єднання

Підключення до Firebase здійснюється за допомогою бібліотеки cloud_firestore, яка є стандартом для роботи з Firebase у Flutter. Після встановлення з'єднання додаток може виконувати різноманітні операції, такі як вставка, оновлення, видалення та вибірка даних.

Для вставки нових задач у базу даних використовується функція addTask() (рис. 3.21), яка приймає об'єкт задачі і вставляє його у відповідну колекцію. Ця функція генерує запит на додавання нового документу до колекції tasks, що містить дані про задачу, такі як назва, опис та дедлайн. Після виконання запиту Firestore автоматично генерує унікальний ідентифікатор для нової задачі, який може бути використаний для подальшого звернення до цієї задачі.

```
Future<void> addTask(Task task) async {
  await FirebaseFirestore.instance.collection('tasks').add(task.toMap());
}
```

Рисунок 3.21 – Функція addTask()

Для вибірки задач з бази даних використовується функція getTasks(), яка здійснює запит на отримання всіх документів з колекції tasks (рис. 3.22). Ця функція отримує всі задачі та повертає список об'єктів задач. Отримані дані перетворюються у список об'єктів за допомогою методу fromMap().

```
Future<List<Task>> getTasks() async {
  QuerySnapshot querySnapshot = await
FirebaseFirestore.instance.collection('tasks').get();
  return querySnapshot.docs.map((doc) => Task.fromMap(doc.data())).toList();
}
```

Рисунок 3.22 – Функція getTasks()

Для оновлення існуючих задач використовується функція updateTask(), яка генерує запит на оновлення документу у колекції tasks (рис. 3.23). Ця функція приймає об'єкт задачі, що містить оновлені дані, і оновлює відповідний запис у Firestore.

```
Future<void> updateTask(Task task) async {
  await
FirebaseFirestore.instance.collection('tasks').doc(task.id).update(task.toMap());
}
```

Рисунок 3.23 - Функція updateTask()

Для видалення задачі з бази даних використовується функція deleteTask(), яка генерує запит на видалення документу з колекції tasks (рис. 3.24). Ця функція приймає ідентифікатор задачі та видаляє відповідний запис з Firestore.

```
Future<void> deleteTask(String id) async {
  await FirebaseFirestore.instance.collection('tasks').doc(id).delete();
}
```

Рисунок 3.24 – Функція deleteTask()

Синхронізація даних з Firebase Firestore здійснюється в режимі реального часу, що дозволяє автоматично оновлювати дані на всіх пристроях користувачів. Ця функціональність забезпечується за допомогою потоків даних (streams), які дозволяють додатку отримувати оновлення даних без необхідності повторного запиту до бази даних.

Технічні деталі роботи з Firebase включають налаштування структури бази даних, що визначає колекції та документи. Колекція tasks містить документи, які представляють задачі, з полями id, name, description, deadline та іншими необхідними атрибутами. Firebase Firestore забезпечує масштабованість та високий рівень безпеки за рахунок використання правил безпеки, які визначають доступ користувачів до даних.

Таким чином, Firebase Firestore забезпечує надійне та ефективне зберігання даних для нашого мобільного додатку, дозволяючи зберігати задачі, управляти ними та синхронізувати дані в режимі реального часу, забезпечуючи актуальність та доступність інформації для користувачів.

ВИСНОВОК

Вибір теми кваліфікаційної роботи був зумовлений зростанням популярності мобільних додатків для тайм-менеджменту та необхідністю створення ефективного інструменту для планування часу. Сучасні мобільні додатки стали незамінними помічниками для мільйонів користувачів, допомагаючи їм організовувати робочий і особистий час. Зважаючи на це, розробка мобільного додатку для тайм-менеджменту з використанням фреймворку Flutter є актуальною темою дослідження. Додаток, розроблений на базі Flutter, забезпечує кросплатформенність, високу продуктивність та інтерактивний інтерфейс користувача.

Мета даної роботи – дослідження технологічних аспектів розробки мобільних додатків, аналіз можливостей та ефективності використання Flutter для створення інтуїтивно зрозумілого і функціонального додатку для тайм-менеджменту. Під час дослідження було визначено, що мобільні додатки для управління часом мають такі ключові вимоги: зручний користувацький інтерфейс, висока продуктивність, інтеграція з календарями та іншими сервісами, можливість налаштування під потреби користувача, а також наявність аналітичних інструментів для відстеження прогресу.

Процес розробки мобільного додатку складався з кількох етапів: визначення функціональних вимог, проектування архітектури додатку, розробка інтерфейсу користувача та програмної частини. Особливу увагу було приділено реалізації екрану «dashboard», який є першим, що бачить користувач при вході до додатку. Цей екран надає основну інформацію про заплановані задачі, зустрічі та події, дозволяючи швидко оцінити стан справ на поточний день.

Також було розроблено екрани для створення нових задач, налаштування, календаря, детального перегляду дня та задач, а також статистики і аналітики користувача. Екран календаря забезпечує візуальне представлення всіх подій та задач, дозволяючи користувачам легко планувати свій час. Статистика та аналітика допомагають користувачам відстежувати прогрес та ефективність виконання задач, що є важливим для покращення тайм-менеджменту.

Для реалізації проекту було використано Firebase як основну базу даних, що забезпечує надійне зберігання та швидкий доступ до даних користувачів. Взаємодія між мобільним додатком та базою даних здійснюється за допомогою API, що дозволяє оперативно синхронізувати дані між різними пристроями користувача.

На основі проведених досліджень та реалізації проекту, можна зробити висновок, що використання фреймворку Flutter для розробки мобільних додатків для тайм-менеджменту є доцільним та ефективним рішенням. Запропонований додаток має всі необхідні функції для зручного та ефективного планування часу, що робить його конкурентоспроможним на ринку мобільних додатків.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Top 5 Mobile Apps That Impact the Mobile App Development Company in 2019 - Weekly Roundup. *Krify - Web and Mobile App Design & Development Company in India & UK*. URL: https://krify.co/top-mobile-apps-influence-app-development-company-in-2019/

2. What Are the Different Types of Mobile Apps? And How Do You Choose? -. CleverTap - All-in One Customer Engagement Platform. URL: https://clevertap.com/blog/types-of-mobile-apps/.

3. Types of Mobile Apps. Devopedia. URL: https://devopedia.org/types-of-mobile-apps.

4. Different Types of Mobile Apps to Develop for any Businesses # Weekly Roundup. Krify - Web and Mobile App Design & Development Company in India & UK. URL: https://krify.co/types-of-mobile-apps/.

5. Main Categories and Types of Mobile Apps [2023 Update]. Bamboo Agile | Custom Software Development Company. URL: https://bambooagile.eu/insights/main-categories-and-types-of-mobile-apps.

6. 5-Year Market Forecast: App Spending Will Reach $233 Billion by 2026. Sensor Tower - Market-Leading Digital Intelligence. URL: https://sensortower.com/blog/sensor-tower-app-market-forecast-2026.

7. A Complete Guide to the Application Development Life Cycle. Inoxoft |. URL: https://inoxoft.com/blog/stages-of-app-development/.

8. Sumaiya Simran. Software Development Life Cycle- Learn How To Build It. Bdtask | Best Software Development Company. URL: https://www.bdtask.com/blog/software-development-life-cycle.

9. Application Development Life Cycle: From Concept To Creation. The Nth Bit Labs. URL: https://thenthbit.com/application-development-life-cycle-from-concept-to-creation/.

10. The importance of Visual Consistency in UI Design. UX Passion. URL: https://www.uxpassion.com/blog/the-importance-of-visual-consistency-in-ui-design/.

11. Soegaard M. The Top UX and UI Design Tools for 2024: A Comprehensive Guide. The Interaction Design Foundation. URL: https://www.interaction-design.org/literature/article/ux-design-tools-definitive-guide.

12. The mobile app architecture guide. DECODE. URL: https://decode.agency/article/mobile-app-architecture/.

13. Clarification :MVC,MVP,MVVM. Stack Overflow. URL: https://stackoverflow.com/questions/4751633/clarification-mvc-mvp-mvvm.

14. Flutter - Build apps for any screen. Flutter - Build apps for any screen. URL: https://flutter.dev/.

15. Flutter documentation. Docs | Flutter. URL: https://docs.flutter.dev/.

16. Pros and Cons of Flutter App Development. AltexSoft. URL: https://www.altexsoft.com/blog/pros-and-cons-of-flutter-app-development/.

17. React Native, Flutter, NativeScript, Xamarin. Google Trends. URL: https://trends.google.com/trends/explore?cat=31&q=React%20Native,Flutter,NativeScript,Xamarin.

18. What is FlutterFlow? Guide, Features, Benefits. Zealous System. URL: https://www.zealousys.com/blog/flutterflow/.
19. FlutterFlow Docs. URL: https://docs.flutterflow.io/.
20. FlutterFlow - Build beautiful, modern apps incredibly fast. URL: https://flutterflow.io/.
21. Flutter Usage Statistics. Built With. URL: https://trends.builtwith.com/framework/Flutter.
22. Flutter, React Native. Google Trends. URL: https://trends.google.com.ua/trends/explore?q=/g/11f03_rzbg,/g/11h03gfxy9&hl=ru.
23. InVerita. Flutter vs Native vs React-Native: Examining performance. Medium. URL: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980.
24. Todoist | A To-Do List to Organize Your Work & Life. Todoist. URL: https://todoist.com/.
25. Manage Your Team’s Projects From Anywhere | Trello. Manage Your Team’s Projects From Anywhere | Trello. URL: https://trello.com/.
26. Forest - Stay focused, be present. Forest. URL: https://www.forestapp.cc/.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ