МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ТЕСТУВАННЯ З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ FLUTTER»**

Виконав:
студент 2 курсу, групи 8.1229
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
В. В. Пасічник
(ініціали та прізвище)
Керівник завідувач кафедри комп'ютерних наук, доцент, к.т.н. Борю С. Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри ІТЕЗ, доцент, к.т.н. Шило Г. М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя

2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук, к.т.н., доцент

_____ Борю С. Ю.
(підпис)

« ____ » _____ 2020 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Пасічнику Вадиму Володимировичу
(прізвище, ім'я та по батькові)

1. Тема роботи Розробка мобільного додатку для тестування з використанням фреймворку Flutter

керівник роботи Борю Сергій Юрійович, к.т.н., доцент
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » травня 2020 року № 576-с

2. Строк подання студентом роботи 01.12.2020

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Вивчення тематичної літератури.
2. Аналіз вимог до програмного продукту.
3. Реалізація мобільного додатку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 23.05.2020

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.09.2020	
2.	Збір вихідних даних.	31.10.2020	
3.	Обробка методичних та теоретичних	10.11.2020	
	джерел.		
4.	Розробка першого та другого розділу.	26.12.2020	
5.	Розробка третього та четвертого розділу.		
6.	Оформлення та нормоконтроль	10.12.2020	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	15.12.2020	

Студент _____ В. В. Пасічник
(підпис) (ініціали та прізвище)

Керівник роботи _____ С. Ю. Борю
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О. Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка мобільного додатку для тестування з використанням фреймворку Flutter»: 56 с., 19 рис., 1 табл., 8 джерел.

ДОДАТОК ДЛЯ ТЕСТУВАННЯ, КРОСПЛАТФОРМЕННІ МОБІЛЬНІ ДОДАТКИ, РОЗРОБКА КРОСПЛАТФОРМЕННИХ МОБІЛЬНИХ ДОДАТКІВ, FLUTTER, IONIC, REACT NATIVE, XAMARIN.

Об’єкт дослідження – фреймворк Flutter.

Мета роботи – розробка мобільного додатку для тестування з використанням фреймворку Flutter.

Методи дослідження – метод наукового узагальнення та класифікації (при дослідженні фреймворків для розробки кросплатформенних додатків).

У даній кваліфікаційній роботі розглядається проблема розробки кросплатформенних мобільних додатків. У результаті виконання роботи, було реалізовано кросплатформенний мобільний додаток, який дозволяє об’єднувати людей в групи, створювати тести та проходити їх. У зв’язку з активним розвитком мобільних телефонів та збільшенням їх розповсюдженості, дана робота є досить актуальною, оскільки в ній аналізуються основні фреймворки для кросплатформенної мобільної розробки та розглядаються проблеми реалізації кросплатформенності.

SUMMARY

Master’s Qualification Thesis «Development of a mobile application for testing using the Flutter framework»: 56 pages, 19 figures, 1 table, 8 references.

APP FOR TESTING, CROSSPLATFORM MOBILE APPS, CROSSPLATFORM MOBILE APPS DEVELOPMENT, FLUTTER, IONIC, REACT NATIVE, XAMARIN.

The object of the study is Flutter framework.

The aim of the study is mobile app developed with the Flutter framework.

The method of research is the method of scientific generalization and classification (in the study of frameworks for cross-platform apps development).

In this qualification work the problem of automatic segmentation of language is considered. As a result of the work, a software library was implemented, which allows automatic division of human speech into sounds with rather high accuracy. In connection with the active development of machine learning in the field of sounds, this work is very relevant and important, since it solves the significant current problem – automatic segmentation of human speech.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС	Операційна система.
ПВД	Прогресивний веб-додаток.
ПЗ	Програмне забезпечення.
AOT	Ahead-of-Time.
API	Application Programming Interfaces.
ARM	Advanced RISC Machine.
ART	Android Runtime.
CSS	Cascading Style Sheets.
FPS	Frames per second.
HTML	HyperText Markup Language.
IDE	Integrated development environment.
IL	Intermediate Language.
JIT	Just-in-time.
JS	JavaScript.
JSON	JavaScript Object Notation.
SDK	Software development kit.
UWP	Universal Windows Platform.

ВСТУП

Проникнення смартфонів у повсякденне життя росте в усьому світі. Очікується, що до 2024 року три з чотирьох використовуваних телефона будуть смартфонами. Згідно зі статистикою StatCounter, доля користувачів настільних приладів знизилась до 45,66%.

Одним з перших кроків на шляху до цифрового успіху розробника є рішення про мобільну операційну систему. Цей вибір був непростим десять років тому, коли, окрім Android та iOS, також Microsoft, RIM та Symbian були активно розвиваючимися та “життєздатними” варіантами.

Сьогодні вибір значно простіше, оскільки єдиними великими гравцями залишаються Android та iOS, які разом складають близько 99% від загальної частини ринку мобільних операційних систем. Згідно з різними статистичними даними, Android виграє по кількості користувачів, але немає нестачі й у прибічниках iOS, доля якого на ринку складає 25,75%. В той час, як Google Play Store може похвалитись великою кількістю додатків (2,5 млн), Apple App Store містить більш ніж 1,8 млн застосунків.

Оскільки вибір мобільної операційної системи є питанням особистих уподобань, а не продуктивності чи доступності, буде доречним створення мобільного застосунку як для Android, так і для iOS. Для цього існує три способи:

а) окремі нативні додатки для Android та iOS;

б) прогресивний веб-додаток (ПВД);

в) один кросплатформений додаток для двох систем.

Нативне рішення, як слідує з назви, передбачає розробку додатку на рідній для даної платформи мові програмування: Java або Kotlin для Android, Objective-C або Swift для iOS. Будучи глибоко орієнтованою на операційну систему, розробка нативних додатків має свої переваги та недоліки. З одного боку, таке рішення забезпечує доступ до всіх функцій операційної системи (ОС), дозволяє необмежено налаштовувати інтерфейс і попереджає будь-які

проблеми з продуктивність. З іншого боку, якщо ви хочете охопити обидва типи користувачів, вам доведеться створювати два окремих додатки, які потребують більше часу, грошей та зусиль.

Прогресивний веб-додаток – технологія у веб-розробці, яка додає сайтам можливості мобільних застосунків і трансформує сайт в застосунок. На виході отримується гібрид сайта з мобільним додатком. Однак, як будь-який інший варіант, ПВД не досконалі, оскільки вони використовують більше енергії акумулятора та не можуть отримати доступ до всіх функцій даного пристрою, наприклад, до календаря, контактів і т.д. Крім цього, втрачається можливість перехресного входу в веб-додаток за допомогою Facebook, Instagram, та інших застосунків. Незважаючи на те, що ПВД не потребує встановлення з Google Play Store або Apple App Store, останні виконують функцію особливо зручних бібліотек для користувачів.

Кросплатформеність – це здатність програмного забезпечення (ПЗ) працювати на кількох платформах.

Кросплатформена мобільна розробка дозволяє охопити обидві операційні системи, iOS та Android, одним кодом. Вона не передбачає написання коду на рідній мові програмування, однак забезпечує майже нативний досвід завдяки інтерфейсу візуалізації з використанням власних елементів керування.

На даний момент багато компаній використовують кросплатформенні рішення, хтось задумується про перехід на них в найближчому майбутньому. Це не тільки самі розробники кросплатформенних фреймворків, як, наприклад, компанія Facebook зі своїм React Native, на якому працюють додатки Facebook та Instagram, але й інші великі гравці на ринку, у яких є свої продукти, наприклад, фреймворк Flutter використовують Alibaba, Philips Hue, Hamilton, Tencent, Grab, Groupon и та інші.

Існує безліч статей, в яких докладно аналізуються всі переваги кросплатформенних додатків. Однак плюси та мінуси варто розглядати на платформі, яка має всі шанси стати в 2020 році найбільш популярною серед розробників – Flutter.

1 ОГЛЯД ОСНОВНИХ ФРЕЙМВОРКІВ

У даному розділі буде розглянуто список найпопулярніших фреймворків для кросплатформенної мобільної розробки та їх проблеми, підходи до реалізації кросплатформенності в додатках, а також, існуючі мобільні додатки для розробки тестів.

1.1 Аналіз фреймворків для кросплатформенної розробки та їх проблем

1.1.1 Xamarin

Xamarin – це кросплатформена технологія, частина платформи .NET, призначена для створення мобільних та веб-додатків. Основою ідеєю є сумісність служб, які написані на різних мовах програмування. На сьогоднішній день реалізована для платформ Windows, FreeBSD, а також в варіанті для ОС Linux (проект Mono). Поділяється на дві основні частини – це середовище виконання, свого роду віртуальна машина, а також інструменти розробника.

У якості середовищ розробки виступають VisualStudio, C++, C#, SharpDevelop. Як і Java, середовище .NET створює байт-код, який виконується віртуальною машиною. Код створюється на мові common intermediate language (CIL). Використання байт-коду дозволяє реалізувати кросплатформеність на рівні вже скомпільованого проекту. Перед запуском, байт-код перетворюється Just-in-time (JIT) компілятором в машинний код.

1.1.2 Ionic

Ionic – це кросплатформений software development kit (SDK), з повністю відкритим кодом, який використовує фреймворк Cordova і плагіни Capacitor для розробки мобільних додатків. Користувачі можуть створювати застосунки і налаштовувати їх для роботи з операційними системами Windows, iOS, Android, а також із сучасними браузерами. Ionic надає користувацькі компоненти і засоби для взаємодії з цими компонентами – наприклад, такі, як віртуальна прокрутка, вкладки, навігація, типографіка і т.д.

Також Ionic пропонує інтерфейс командної стоки й сервіси для рішення інших задач, наприклад, розвернення коду та автоматичної збірки. Включає також і власне інтегроване середовище розробки (Integrated development environment – IDE) – Ionic Studio. Розробник може підключати додаткові модулі фреймворка Cordova, включати push-повідомлення, створювати значки додатків.

1.1.3 React Native

React Native – це платформа для створення мобільних додатків, створена компанією Facebook, яка має відкритий код. Дозволяє розробляти додатки для iOS, Android, Universal Windows Platform (UWP) та Web. React Native не використовує Cascading Style Sheets (CSS) і HyperText Markup Language (HTML) і дозволяє створювати код на мовах Swift та Objective-C для iOS, а також на Java для Android. React Native дозволяє створювати кросплатформенні додатки, компоненти платформи взаємодіють із власними Application Programming Interfaces (API) за допомогою декларативної парадигми інтерфейса React та JavaScript (JS).

Компоненти React обертають існуючий власний код та взаємодіє з власними API-інтерфейсами. Це дозволяє створювати власні додатки для

цілих нових груп розробників і також дозволяє існуючим командам працювати набагато швидше. У своїй більшості принципи роботи React Native ідентичні React, однак, на відміну від останнього, він працює у фоновому режимі на кінцевому пристрої, інтерпретуючи код, написаний на JavaScript.

1.1.4 Flutter

Flutter – це досить молода платформа, яка приваблює розробників своєю простотою. Швидкість її роботи та висока продуктивність досягається за рахунок використання декількох технік.

Перш за все, Flutter не використовує JavaScript, його творці надали перевагу мові програмування Dart, з якої код легко компілюється в двійковий. Завдяки цьому швидкість виконання операцій цілком можна порівнювати з такою у мов Swift, Kotlin та Java. Також, платформа не використовує нативні компоненти, відмальовуючи інтерфейс в графічному движку за необхідності – лише у випадку, якщо в нього внесені зміни.

В ОС Linux, iOS, Android та Windows Flutter працює за допомогою віртуальної машини Dart з JIT-компілятором. Одна з головних переваг цього SDK – реалізація функції “гарячого перезавантаження”, завдяки чому, зміни коду можуть бути застосовані в уже запущеному додатку, не потребуючи його перезавантаження. Віджети Flutter оснащені вбудованими елементами, які спрощують розробку продукту – віртуальна прокрутка, навігація, типографіка та іконки. Код, написаний на Flutter, компілюється за допомогою нативного компілятору Dart.

1.1.5 Порівняння фреймворків

Наведемо основні характеристики та дані, щодо розглянутих фреймворків, у вигляді таблиці (таблиця 1).

Таблиця 1 – Порівняльна таблиця фреймворків для розробки кросплатформенних мобільних додатків

	Flutter	**React Native**	**Ionic**	**Xamarin**
Мова	Dart	HTML, CSS, JavaScript + React	JavaScript, HTML, CSS + Angular, React, Vue	C# + .NET
Перший реліз	2017	2015	2013	2011
Розробник	Google	Facebook	Drifty Co.	Microsoft
Платформи	Android, iOS, Google, Fuchsia, Web, Desktop	Android, iOS, UWP	Android, iOS, Web	Android, iOS, UWP
Відкритий ресурс	так	так	так + платні пакети	так + платні пакети
Інструменти фронтенда	Built-in widgets	Native + Declarative UI components	HTML, CSS + widgets	Xamarin. iOS/Android or Xamarin.Forms
Продуктивність	Дуже висока	Висока, близька до нативної	Середня через веб-технології	Висока, близька до нативної

1.1.6 Недоліки архітектури

У кросплатформенної архітектури є типові проблеми. Частково вони обумовлені тим, що шар абстракції намагається підтримувати одразу кілька платформ. Це породжує складності: на кожній окремо взятій платформі додаток відображається та поводиться по різному. Іноді це не проблема, але бувають випадки, коли унікальний дизайн і поводження на обох платформах є тим, на що робиться основна ставка при розробці продукту.

У приклад можна поставити додаток Reflectly. Спочатку він був написаний на React Native, уперше з’явився на iOS, а пізніше була спроба запуску на платформу Android. Запуск застосунку провалився (рисунок 1.1).

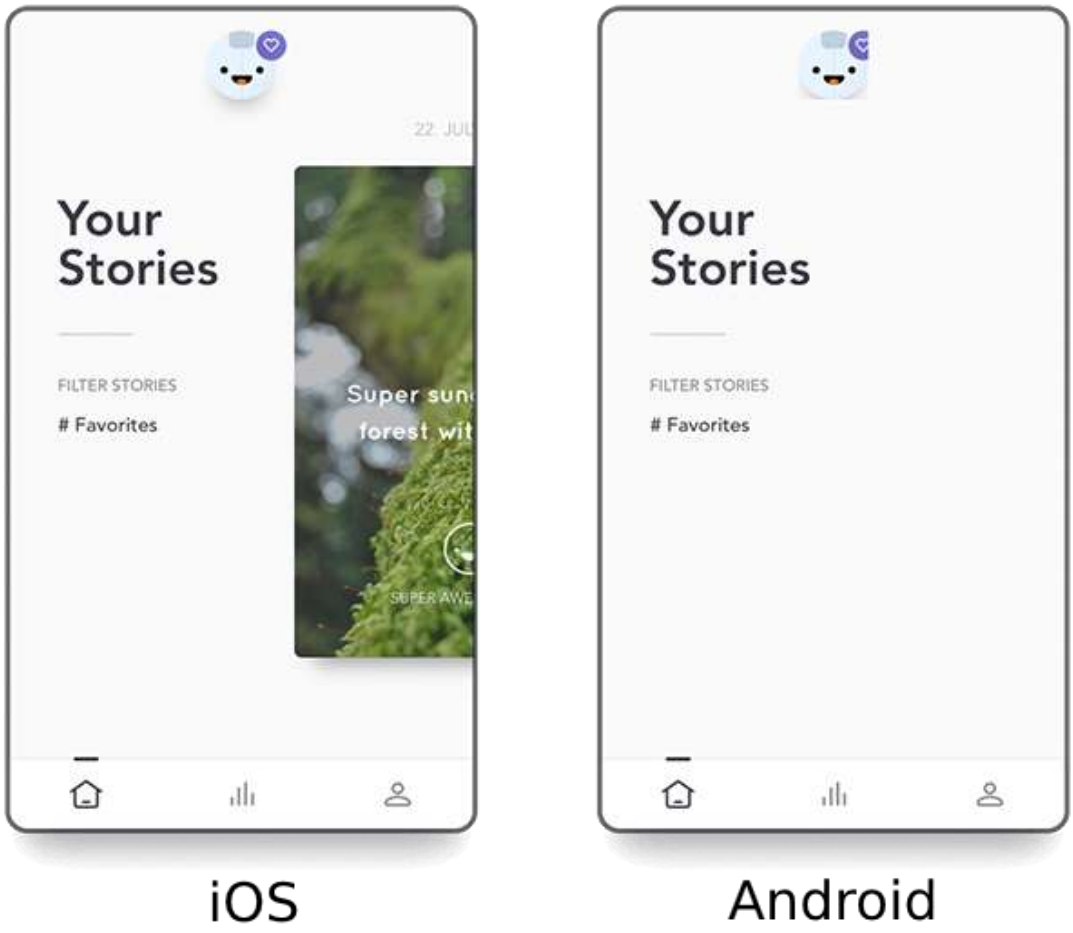

Рисунок 1.1 – Реалізація додатку Reflectly з використанням React Native на iOS (зліва) та Android (справа)

Через значні проблеми з функціоналом, додаток довелося повністю переписати на Flutter (рисунок 1.2).

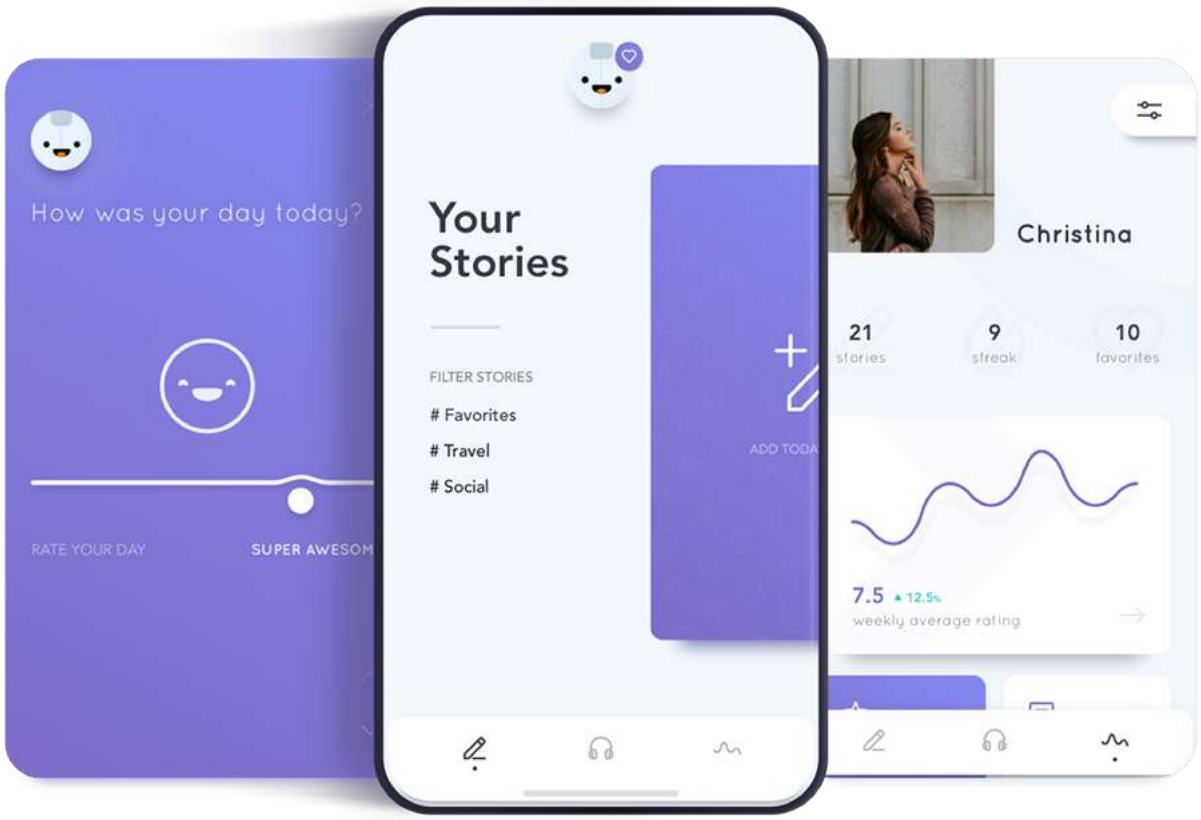

Рисунок 1.2 – Реалізація додатку Reflectly з використанням Flutter

У даному випадку, додаток реалізує унікальний функціонал, який не схожий на стандартний набір форм та списків, дизайн також є незвичайним та дуже важливим.

У кросплатформенних додатків продуктивність нижча, ніж у нативних. Це пов’язано з тим, що всі дії з платформою – рендерінг, виклик нативних методів – спочатку серіалізуються в текстовий рядок у додатку, передаються на платформу, уже там виконується десеріалізація і тільки після цього отримується бажаний результат. Саме через такий набір дій, продуктивність додатків зменшується.

Інструменти для розробки кросплатформенних додатків наразі не так розповсюджені, як класичний frontend чи нативні додатки. Цим зумовлені

проблеми з плагінами, так, нерідко плагіни випускаються в Open Source, і практично відразу закінчується їх підтримка, оскільки існуюча версія вирішує проблему розробника. Бувають гірші ситуації, коли закінчується не тільки підтримка, але і реагування на pull requests – запити з кодом для виправлення помилок або створення нового функціоналу від інших розробників.

Існує міф про те, що будь якого web-розробника можна швидко перекваліфікувати в розробника мобільних додатків за допомогою «магічного кросплатформеного фреймворка X». Начебто, все виглядає схожим у вебі та кросплатформі – набір описання стилів зовнішнього вигляду та програмний функціонал. Але не враховується той факт, що до наявних знань треба ще й швидко додати розуміння щонайменше двох платформ – Android та iOS, з якими розробник ще не стикався. До всього цього треба додати й специфічні тонкості розробки, які наявні у будь якій платформі.

1.1.7 Висновок

У даному підрозділі було розглянуто самі популярні фреймворки для кросплатформенної розробки мобільних додатків. По наведеним даним, можна зробити висновок, що інструменти, які працюють за допомогою web-технологій, показують гірші результати в продуктивності. Також, можна помітити, що з роками технології покращуються і показують здатність працювати майже на одному рівні з нативними додатками, як, наприклад, фреймворк Flutter.

1.2 Аналіз підходів до кросплатформенності

У даному підрозділі буде розглянуто які існують методи реалізації кросплатформенності у різних фреймворків.

1.2.1 Xamarin

Роботу Xamarin графічно можна представити наступним зображенням (рисунок 1.3).

Рисунок 1.3 – Графічне представлення роботи фреймворку Xamarin

Xamarin працює поверх фреймворка Mono, який представляє вільнорозповсюджувану реалізацію .NET Framework. Він може працювати на різних платформах – Linux, MacOS та інші.

На рівні кожної окремої платформи Xamarin покладається на ряд субплатформ. Зокрема:

– Xamarin.Android – бібліотеки для створення додатків на ОС Android;
– Xamarin.iOS – бібліотеки для створення додатків на iOS.

Ці субплатформи мають велике значення, оскільки через них додатки можуть скеровувати запити до прикладних інтерфейсів на пристроях під управлінням Android та iOS.

З допомогою Xamarin.Android код C# з використанням Xamarin компілюється в Intermediate Language (IL), який потім при запуску додатку компілюється в нативну збірку. Xamarin-додатки запускаються в середовищі виконання Mono. Напряму код не може звертатися до API Android. Для цього необхідно звернутися до функціональності простору імен Android.* та Java.*, які надаються віртуальною машиною Android Runtime (ART). Спеціальний прошарок Managed Callable Wrappers дозволяє транслювати виклики коду в нативних викликах і звертатися до функціональності простору імен Android.* та Java.*.

І навпаки, коли ART звертається до застосунку з кодом Xamarin, то всі виклики проходять через обгортку Android Callable Wrappers.

Бібліотека Xamarin.iOS на відміну від Xamarin.Android, яка використовує JIT-компиляцию, застосовує Ahead-of-Time (AOT) компіляцію коду C# в нативний Advanced RISC Machine (ARM) код. Xamarin використовує проміжний шар Selectors для трансляції викликів коду Objective-C в код C# і шар Registrars для оберненої трансляції коду C# в Objective-C. У результаті, ці шари в цілому представляють проміжний шар, який позначений як «bindings» (див. рис. 1.3) і який дозволяє взаємодіяти коду Objective-C з кодом C#.

Можливість створення кросплатформенних додатків представлена технологією Xamarin.Forms, яка працює, у деякому сенсі, на рівень вище, ніж Xamarin.Android та Xamarin.iOS. Тобто, з допомогою Xamarin.Forms ми один раз можемо визначити візуальний інтерфейс, один раз до нього прив’язати логіку C#, і все це буде працювати на Android, iOS и Windows. Потім Xamarin.Forms за допомогою рендерерів (renderer) – спеціальних об’єктів для зв’язку контролів на XAML/C# з нативними контролами – транслюють візуальні компоненти Xamarin.Forms в графічний інтерфейс, специфічний для кожної платформи.

1.2.2 Ionic

Роботу Ionic графічно можна представити наступним зображенням (рисунок 1.4).

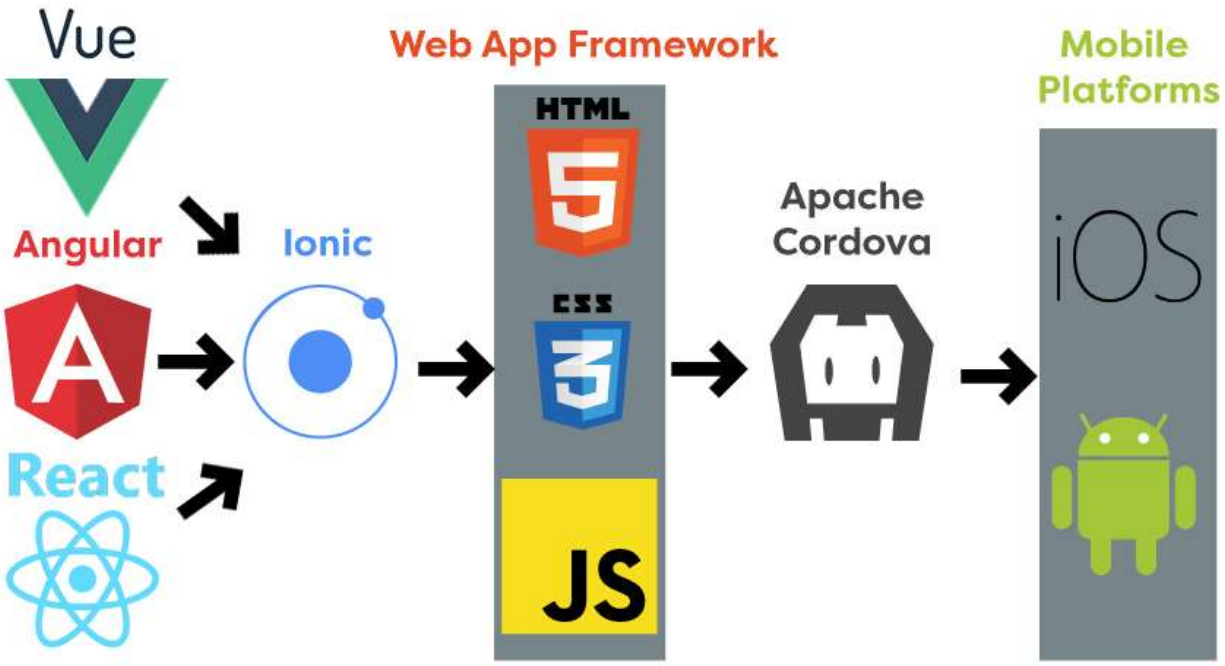

Рисунок 1.4 – Графічне представлення роботи фреймворку Ionic

Фреймворк Ionic працює з web-технологіями – HTML, CSS, JS, а також з web-фреймворками, такими як Angular, React та Vue.

Ionic працює всередині особистого контейнера, який використовує фреймворк Cordova або, в останніх версіях, Capacitor. Вони дають змогу користуватися будь-якими API та власними функціями пристрою. Інтерфейс застосунку Ionic працює в WebView, який фактично являє собою браузер, непомітний для користувача.

WebView – це спрощений браузер, який можна вбудувати в нативний додаток, для відображення web-контенту. Його робота описується наступним зображенням (рисунок 1.5).

Якщо представити звичайний браузер у вигляді двох складових частин, перша – інтерфейс (пошуковий рядок, кнопки навігації, та інше), друга – движок, який перетворює HTML розмітку в набір пікселів, які ми бачимо та з якими взаємодіємо (рисунок 1.6).

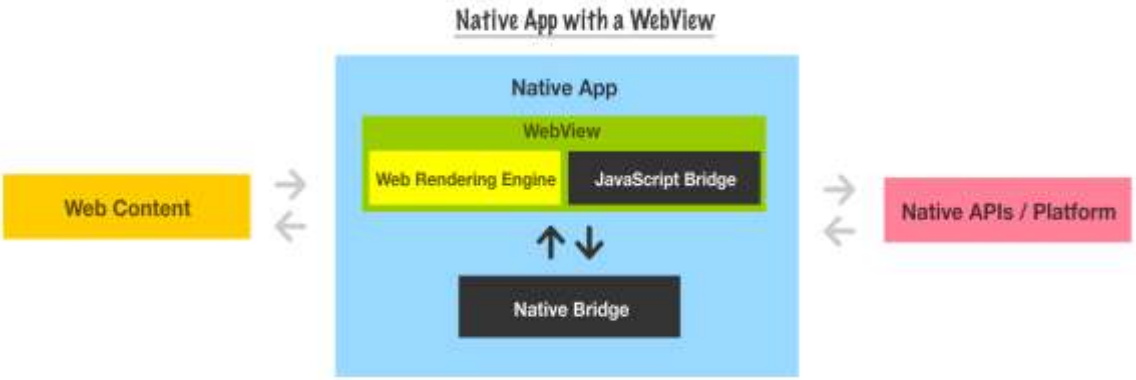

Рисунок 1.5 – Графічне представлення роботи технології WebView

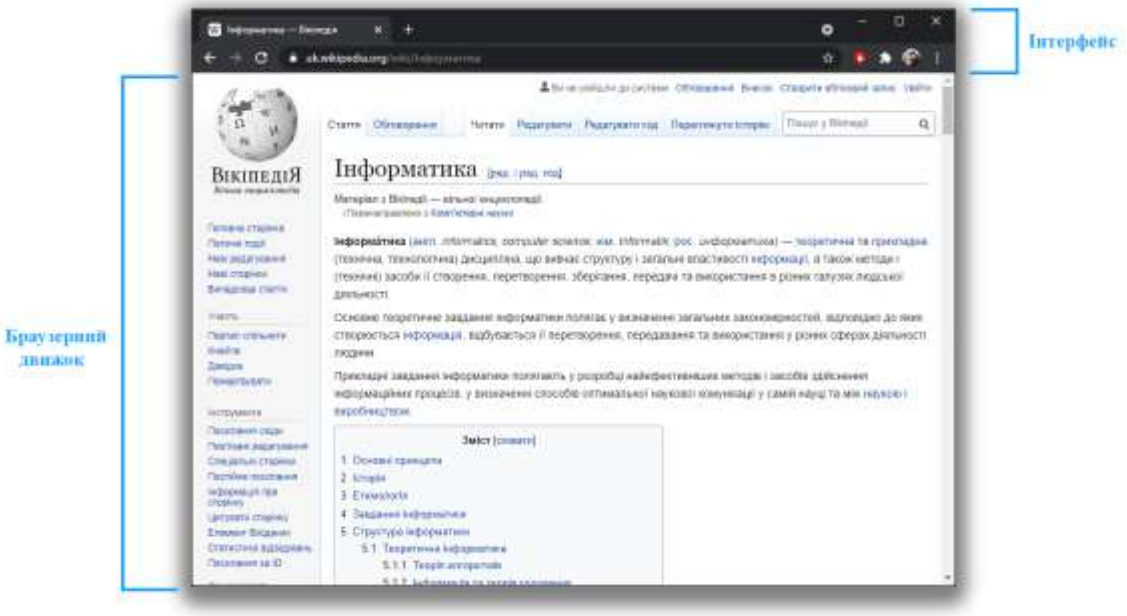

Рисунок 1.6 – Структура вікна браузера

WebView являє собою саме другу частину – браузерний движок, який не має інтерфейсу. Його можна вбудувати в нативний додаток і програмно вказати який web-контент відображати (рисунок 1.7).

При такій схемі роботи, контент, який відображається всередині WebView не має обов'язково бути локальним, тобто знаходитись на тому ж пристрої, що й додаток.

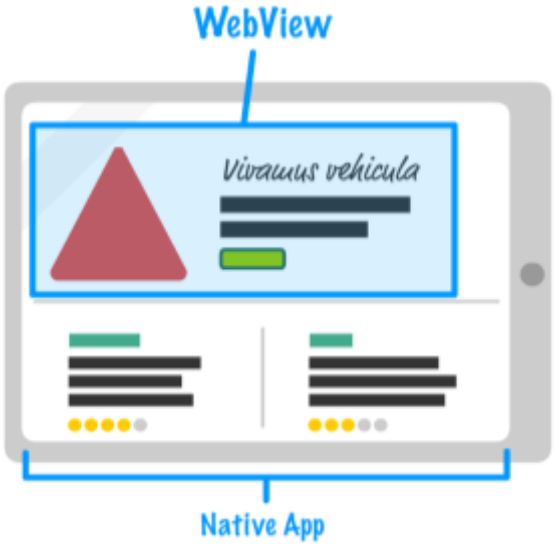

Рисунок 1.7 – Схематичне представлення WebView контенту в мобільному додатку

1.2.3 React Native

Роботу React Native графічно можна представити наступним зображенням (рисунок 1.8).

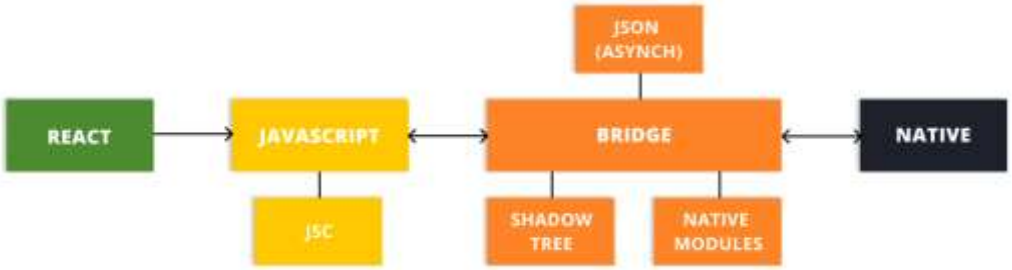

Рисунок 1.8 – Графічне представлення роботи фреймворку React Native

React Native дозволяє розробляти додатки, які складаються з JS-коду та нативного, створюючи міст між застосунком і платформою. Коли JavaScript працює разом з нативним кодом, система мосту React Native використовую бібліотеку React і переносить ієрархію компонентів у відображення на пристрої.

Під час будь яких оновлень, наприклад, коли користувач натискає кнопку, React Native перекладає це в подію (event), яку може опрацювати JavaScript. І тоді, за допомогою ретрансляції повідомлень між конкретною нативною платформою та кодовою базою JS, міст React Native перекладає нативні події в такі, які React-компонент може розібрати та опрацювати.

Така архітектура має певні проблеми. З одного боку, стандартні компоненти можуть не покривати обидві проблеми або можуть виглядати дуже по різному. З іншого боку, архітектура з використанням мосту дозволяє використовувати всі існуючі нативні представлення (views) з SDK платформи та JS бібліотек.

Будучи однопотоковим, додаток, написаний на React Native, легко зрозуміти, оскільки, все, що працює з JavaScript у мережі Інтернет, буде працювати так само і з React Native. Але при розробці архітектури додатків, важливо враховувати специфіку JS, щоб уникнути проблем з продуктивністю.

Таким чином, якщо архітектура майбутнього додатку передбачаю багата подій (events) і даних, то React Native може бути не найкращим варіантом, оскільки структура мосту може викликати затримки.

1.2.4 Flutter

Роботу Flutter графічно можна представити наступним зображенням (рисунок 1.9).

Рівень фреймворку – все, з чим працює розробник в момент написання додатку, і всі службові класи, які дозволяють взаємодіяти написаному ним коду з рівнем движка. Все, що відноситься до даного рівня, написане на мові Dart.

Рівень движка – більш низький рівень, він складається з класів та бібліотек, які дозволяють працювати рівню фреймворка. Також, тут знаходяться віртуальна машина Dart, графічний движок Skia та інші.

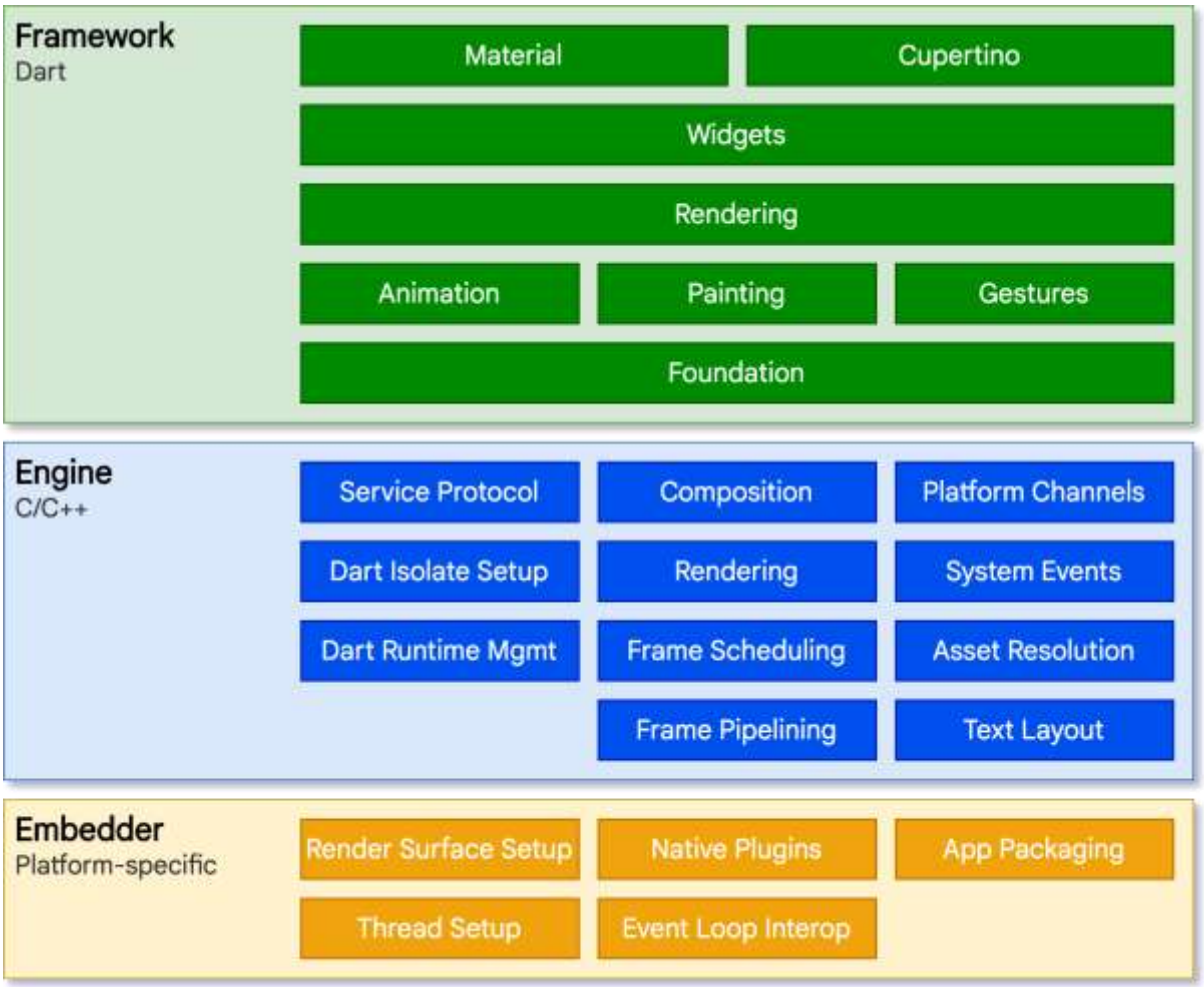

Рисунок 1.9 – Графічне представлення роботи фреймворку Flutter

Рівень платформи – специфічні механізми, які відносяться до конкретної платформу запуску.

Ключовою особливістю архітектури системи є те, що всі віджети, а також компоненти, відповідні за відмальовку віджетів, є частиною додатку, а не платформи. Саме відсутність необхідності в перемиканні контексту і використання мостів дає приріст продуктивності, яка, сприяє досягненню високих показників швидкості відмальовки – frames per second (FPS), до 60 FPS при малюванні інтерфейсу.

Важливою перевагою Flutter є те, що всі бібліотеки, доступні в нативних додатках SDK та також API платформи, можуть бути використані в додатках Flutter (рисунок 1.10).

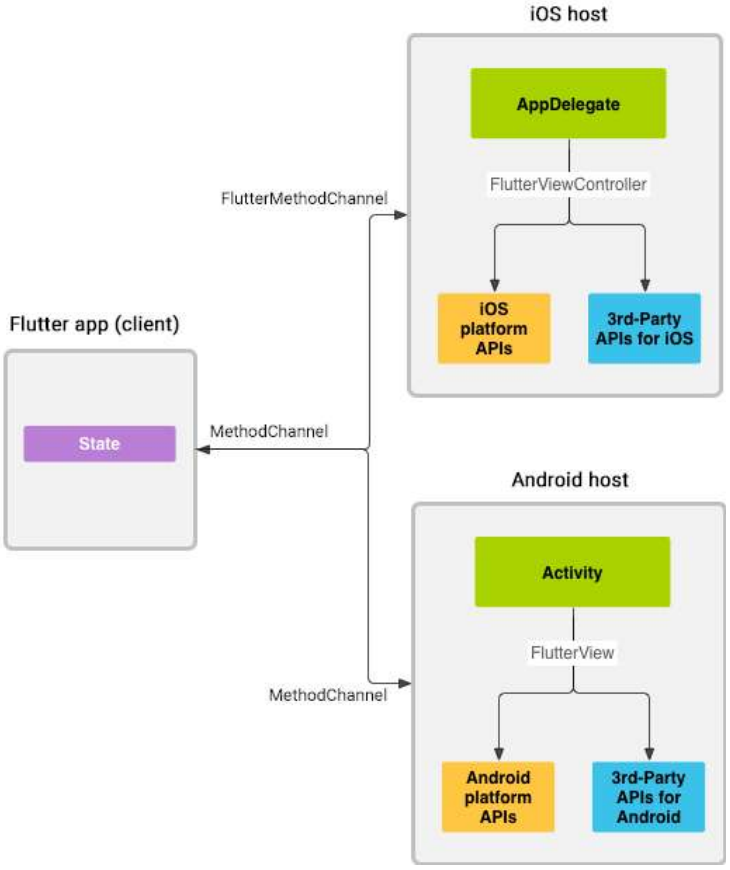

Рисунок 1.10 – Спрощене схематичне представлення взаємодії додатка, написанного з використанням Flutter, зі стороннімі API

1.2.5 Висновок

У даному підрозділі було розглянуто існуючі методи реалізації кросплатформенності у різних фреймворків. Для цього є різні варіанти – відображення web-ресурсу за допомогою своєрідного браузера в додатку, “спілкування” нативного коду з кросплатформенною частиною та компіляція безпосередньо у машинний код для виконання на пристрої.

Було вивчено роботу зазначених вище підходів до реалізації кросплатформенності. Стало зрозуміло, що робота додатків, які використовують web-технології для роботи, показують значно нижчі показники продуктивності, ніж додатки, які працюють з допомогою мосту.

Проте, і ці застосунки поступаються в швидкості роботи додаткам, які компілюються у машинний код.

1.3 Аналіз мобільних додатків для тестування

У даному розділі будуть розглянуті безкоштовні мобільні додатки, які реалізують функціонал створення тестів та їх проходження.

1.3.1 TestMaker

Додаток TestMaker від розробника Yamada Keita доступний на платформах iOS та Android, він дозволяє без обов’язкової реєстрації створювати тести з різними видами питань:

- відповідь з введенням слова або слів;
- відповідь з введенням кількох слів;
- запитання з вибором одного варіанта;
- запитання з вибором кількох варіантів, у якого можна встановити врахування порядку.

До кожного виду питань можна додати коментар та картинку. Для тесту можна вказати колір та назву. Також, тести можна згрупувати за категоріями.

Для розповсюдження створеного тесту, можна використати гіперпосилання, але для його отримання треба бути зареєстрованим. Також, можна передавати імпортований файл типу csv з тестом будь яким зручним способом.

Після реєстрації функціонал додатку майже не змінюється – стає доступною сторінка з редагуванням імені та списком тестів, а також, згадане раніше розповсюдження тесту.

Є розділ «Online tests», в якому знаходяться тести інших користувачів, однак, не зрозуміло, чи всі створені користувачами тести потрапляють в цей розділ. При створенні та редагуванні тесту немає можливості вказати, чи хоче автор розмістити його в цей розділ.

Додаток є багатомовним і встановлює стандартну мову пристрою, якщо така відсутня в додатку, встановлюється англійська. Українська мова відсутня, переклад на російську виконано на дуже низькому рівні.

Також, безкоштовна версія додатку містить рекламу, яка постійно знаходиться у нижній частині вікна додатку.

1.3.2 Testity

Додаток Testity від розробника Serhii Yaniuk доступний лише для ОС Android та має обов'язкову реєстрацію за допомогою власних даних (email-адреса, пароль та ім’я), облікового запису Google або Facebook. При створенні тесту користувач може вказати категорію та мову із наданого списку, додати назву та опис тесту, а також вказати, чи буде доступним цей тест онлайн для інших користувачів додатку.

Тести можуть мати не більше 8-ми варіантів відповіді, а питання можуть бути тільки одного виду – з кількома правильними відповідями. Звісно, можна обрати лише одну правильну відповідь, але користувач, при проходженні тесту, все одно зможе обирати декілька варіантів.

Перед кожним проходженням тесту потрібно вказати своє ім’я, хоча, додатку воно вже відоме, оскільки було вказане при реєстрації або отримане з облікового запису соціальних мереж (залежно від способу реєстрації).

При проходженні тесту немає можливості відмінити обраний варіант відповіді, тобто, якщо випадково натиснути інший варіант, то він буде доданий до вашої відповіді.

Після проходження тесту користувач може бачити скільки часу він витратив, та який відсоток відповідей був правильним. Правильні та неправильні відповіді бачити неможна.

Відсутня можливість розповсюджувати тести лише для окремих користувачів, або груп користувачів. Натомість, у додатку є функція пошуку серед усіх онлайн тестів за категорією, мовою та ім’ям автора. Також, наявна функція запрошення інших користувачів за допомогою email- або sms-повідомлення, але, при ознайомленні з застосунком, ця функція постійно видавала помилку, тож перевірити її не вдалось.

Додаток є одномовним, тому весь інтерфейс на російській мові.

Загалом, додаток Testity є значно простішим за функціоналом та більш зрозумілим, ніж розглянутий раніше TestMaker. Перевагою, також, є те, що додаток не містить жодного рекламного оголошення.

1.3.3 Quizzer

Додаток Quizzer від розробника Sergio Yanes представлений лише для ОС Android, з функціоналом, доступним без реєстрації.

Додаток має лише 2 розділи: створення тестів, та проходження тесту. У розділі створення відображується список розроблених тестів з можливістю їх редагування, розповсюдження та видалення, також доступне створення нових тестів.

При створенні тесту, вказується лише його назва, після чого стає доступним додавання питань. Додаток має декілька типів питань:

- питання з однією правильною відповіддю;
- питання з кількома правильними відповідями;
- відкрите питання, критерієм правильності може бути наявність в тексті якогось слова чи фрази або сувора відповідність написаного тексту;
- питання з варіантом відповіді «вірно» або «невірно».

До кожного питання можна додати текстове пояснення та посилання на онлайн ресурс.

Якщо користувач забажає розповсюдити тест, то для цього буде згенеровано унікальний код і створено url-посилання для завантаження. Перед розповсюдженням, можна вказати, чи зможуть інші користувачі редагувати тест та продивлятися питання з відповідями.

У розділі проходження тесту, можна обрати один або декілька тестів у якості ресурсу питань, вказати кількість питань, час на одне питання та тест загалом, максимально допустиму кількість помилок та декілька інших налаштувань.

При проходженні тесту, користувач може бачити таймер на поточне питання та на тест загалом, кількість доступних можливостей помилитися, а після кожного питання може побачити додане автором пояснення.

Одразу після завершення тесту є можливість продивитися які відповіді були правильними, а які ні. Однак, після переходу до основних розділів, більше побачити свої результати не буде можливості.

Додаток є багатомовним та відображує інтерфейс на мові системи, якщо такої немає, то застосовується англійський переклад. Українська мова в додатку відсутня.

Безкоштовна версія додатку містить рекламу, яка постійно знаходиться у нижній частині вікна додатку, а також з’являється при переходах між різними вікнами, наприклад, після завершення тесту або виходу з вікна налаштувань.

З допомогою розглянутого додатку Quizzer, можна створювати скоріше вікторини, які мають у собі ігрову складову, аніж тести. Окрім назви та демонстративного тесту з назвою «Демо-вікторина», на це також вказує можливість визначити кількість «життів» (допустиму кількість помилок) на проходження тесту. Незважаючи на це, додаток дозволяє створювати досить функціональні та інформативні тести.

1.3.4 Висновок

У даному підрозділі було розглянуто три безкоштовних додатки для мобільних систем Android та iOS, які реалізують функціонал створення тестів та їх проходження. Стало зрозуміло, що окремі розробники створюють досить функціональні, але, все ж, не зовсім зручні додатки для тестування. У одних відсутній зрозумілий переклад, що ускладнює роботу користувача. У інших обмежений функціонал, що обмежує можливості користувача. Окрім того, більшість безкоштовних додатків містить багато рекламних повідомлень, які займають простір вікна додатку та заважають користувачам.

Також, розглянуті додатки мають деякі проблеми з розповсюдженням тестів. Якщо необхідно створити тест з метою контролю знань інших користувачів, то жоден з додатків не дозволяє зробити це зі свого облікового запису. Однак, деякі застосунки не дають жодної можливості розповсюдження, окрім як відкритий доступ для всіх користувачів додатку, що може викликати труднощі.

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Опис задачі, що вирішується

Ціллю даної роботи є розробка мобільного додатку для тестування з використанням фреймворку Flutter.

Для досягнення поставлених цілей, в роботі вирішуються наступні задачі:

а) огляд основних фреймворків:
   1) аналіз фреймворків для кросплатформенної розробки та їх проблем;
   2) аналіз підходів до кросплатформенності;
   3) аналіз мобільних додатків для тестування;

б) постановка задачі;

в) розробка вимог до розроблюваного мобільного додатку;

г) вибір інструментарію для реалізації;

д) процес розробки:
   1) розробка інтерфейсу та логіки додатку;
   2) розробка серверного API;

е) тестування додатку.

2.2 Розробка вимог до розроблюваного мобільного додатку

Для розроблюваного додатку, висуваються наступні вимоги:

а) додаток повинен реалізовувати реєстрацію користувачів за допомогою персональних даних або існуючих облікових записів Google та Facebook;

б) необхідно розробити форми для створення тестів з питаннями різних видів:
   1) введення одного слова чи слів;
   2) відкрита відповідь – можливість написати об’ємний, як для тесту, текст. Правильність може перевірятися наявністю в тексті заданих слів чи повною відповідністю тексту;
   3) вибір одного з варіантів;
   4) вибір декількох варіантів;
   5) встановлення відповідності;
   6) встановлення послідовності;

в) необхідна система створення окремих «кімнат» – груп, у які можна додавати користувачів та тести для проходження;

г) для доступу у кімнату необхідно розробити систему розповсюдження запрошень. Це потрібно для того, щоб творець кімнати, назвемо його адміністратором, міг відправити посилання для входу в кімнату іншим людям. При цьому, запрошені користувачі можуть лише проходити тести і не мають доступу до редагування тестів;

д) необхідна можливість додавати в кімнати тести та встановлювати час на їх проходження;

е) для адміністратора необхідна можливість дивитися результати тестувань та відповіді користувачів;

ж) додаток повинен відправляти користувачам нагадування про тести, які наближаються, у вигляді push- та email-повідомлень. При цьому, у користувача повинна бути можливість вимкнути повідомлення всього додатку або окремих кімнат;

з) необхідна окрема сторінка, на якій будуть розміщуватися найближчі за часом проведення тести;

и) користувач повинен мати можливість видалити свій обліковий запис із додатку, при цьому, також, мають бути видалені всі створені ним тести та кімнати.

3 АНАЛІЗ ОСОБЛИВОСТЕЙ ДОДАТКУ

У даному розділі будуть описані основні дані щодо реалізації додатку: обраний інструментарій для розробки, інтерфейсу, логіки та API, а також тестування додатку.

3.1 Вибір інструментарію для реалізації

Для розробки додатку, було обрано фреймворк Flutter, який показує високу продуктивність, є досить легким у вивченні та має швидко зростаючу популярність.

Flutter – молода, але багатообіцяюча платформа, яка вже привернула увагу крупних компаній. Цікава вона своєю простотою, яку можна порівняти з розробкою web-додатків, і швидкістю роботи, близькою до нативних застосунків. Високу продуктивність і швидкість розробки досягається за рахунок кількох технік:

– на відміну від багатьох відомих на сьогоднішній день мобільних платформ, Flutter не використовує JavaScript. В якості мови програмування використовується Dart – мова, яка компілюється в бінарний код. За рахунок цього досягається швидкість операцій, порівнювана з мовами Objective-C, Swift, Java та Kotlin;

– Flutter не використовує нативні компоненти, через це не довелося створювати ніяких містків для комунікації з ними. Замість цього, подібно ігровим движкам, він відмальовує весь інтерфейс самостійно. Кнопки, текст, медіа-елементи, фон – все це малюється всередині графічного движка в самому фреймворку;

– для побудови інтерфейсу в Flutter використовується декларативний підхід, натхненний web-фреймворками, в основі яких лежать

віджети (у web їх називають компонентами). Для ще більшого приросту в швидкості роботи інтерфейсу, віджети перемальовуються за необхідності – тільки, коли в них щось змінилось (подібно тому, як це робить Virtual DOM у web);

– в додаток до всього, у фреймворк вбудовано технологію hot-reload – перебудова змінених частих коду прямо в працюючому додатку без необхідності перебудови всієї програми та повторного встановлення на пристрій. Ця технологія також розповсюджена в web, проте, все ще відсутня в нативних платформах.

Окрім цього, на офіційному сайті Flutter розміщені інструкції, які допомагають швидше освоїти фреймворк розробникам з інших платформ та мов, таких як: Android, iOS, React Native, web та Xamarin.Forms. Така інструкція дозволяє ознайомитися з основами розробки на прикладі уже відомих концепцій та структур.

Dart – мова програмування високого рівня, яка використовує інтерпретатор для роботи. Розробляється компанією Google і вперше була представлена в 2011 році. Програмування на Dart дуже нагадує роботу з мовами C (Сі) та JavaScript.

3.2 Процес розробки

3.2.1 Розробка інтерфейсу та логіки додатку

Для розробки інтерфейсу додатків у Flutter використовуються віджети (widgets) – це готові класи, які приймають на вхід параметри, для описання вигляду елементів і в результаті створюють їх відображення на пристрої. Розробка інтерфейсів за допомогою віджетів була створена з натхненням від web-фреймворку React. При зміні внутрішнього стану віджету виконується

перемальовка на екрані пристрою, при цьому, перемальовується лише та частина, яка змінилася, а не весь віджет.

Найпростіша програма, яка виводить текст «Hello world» у фреймворку Flutter буде виглядати наступним чином:

```
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'First App',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

При написанні програми виникає необхідність створювати свої класи – віджети, які будуть реалізовувати більш складні частини інтерфейсу ніж простий текст. Кожен віджет повинен наслідувати один із класів – StatelessWidget або StatefulWidget, в залежності від того, чи буде віджет мати власний стан (state).

Кожен віджет повинен реалізовувати функцію build, яка має лише один вхідний параметр типу BuildingContext – він визначає положення віджета у дереві віджетів.

Власний стан віджета – це, у спрощеному розумінні, набір його внутрішніх змінних. Простим прикладом такого віджету зі станом, який змінюється після взаємодії користувача з додатком, може виступати звичайний лічильник, код якого буде мати наступний вигляд:

```
class Counter extends StatefulWidget {
  @override
  CounterState createState() => CounterState();
}
class CounterState extends State<Counter> {
  int counter = 0;
  void increment() {
    setState(() { counter++; });
  }
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        ElevatedButton(
          onPressed: increment,
          child: Text('Increment'),
        ),
        Text('Count: $counter'),
      ],
    );
  }
}
```

Результуючий додаток матиме наступний вид (рисунок 3.1).

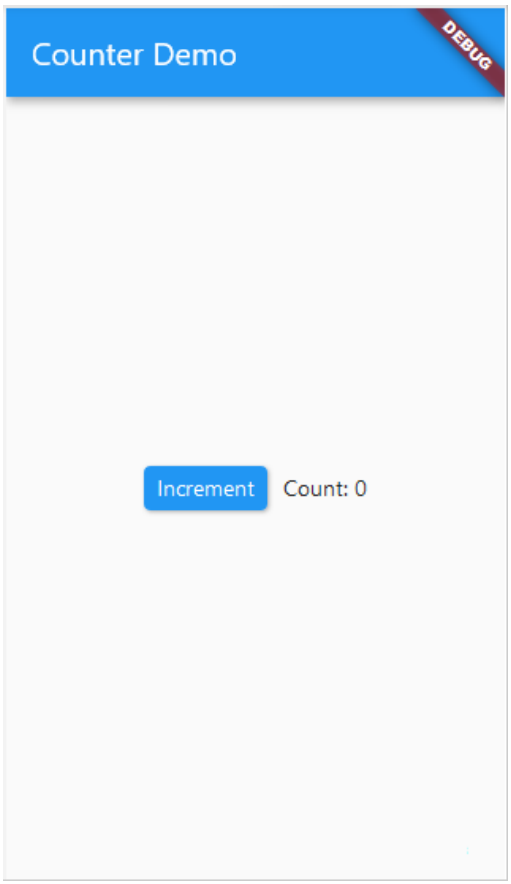

Рисунок 3.11 – Приклад зовнішнього вигляду додатку "Лічильник"

Це найпростіші приклади програм, написаних з використанням фреймворку Flutter.

Вхідною точкою нашого майбутнього додатку буде файл main.dart, який містить наступний код:

```
import 'package:flutter/material.dart';
import 'package:TesTseT/theme/style.dart';
import 'package:TesTseT/routes.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'TesTseT',
```

```
      theme: appTheme(),
      initialRoute: '/',
      routes: routes,
    );
  }
}
```

У цьому коді виконується імпорт файлу material.dart з пакету flutter, у якому реалізуються віджети для побудови додатків, такі як: Text, Container, Column, Row та інші. Також, виконується імпорт двох файлів: TesTseT/theme/style.dart та TesTseT/routes.dart – це файли реалізовані в додатку, вони описують основні частини теми додатку та доступні шляхи (routes). Для ініціалізації додатку, окрім теми та шляхів, також указуються параметри title – назва додатку, та initialRoute – шлях, який буде відкриватися при увімкненні додатку.

Для ознайомлення, наведемо приклад коду із файлу шляхів, routes.dart:

```
import 'package:flutter/widgets.dart';
import 'package:TesTseT/screens/home.dart';
import 'package:TesTseT/screens/login.dart';
import 'package:TesTseT/screens/signup.dart';
final Map<String, WidgetBuilder> routes = <String, WidgetBuilder>{
  "/": (BuildContext context) => HomePage(),
  "/login": (BuildContext context) => LoginPage(),
  "/sign-up": (BuildContext context) => SignupPage(),
};
```

У цьому файлі ми імпортуємо існуючі віджети, у нашому випадку це: HomePage, LoginPage та SignupPage. Для кожного віджета описується шлях, при переході на який його буде показано користувачеві. Такий алгоритм роботи дуже схожий на реалізації single page applications у сучасних web-фреймворках. Для переходу між шляхами, використовується команда Navigator.pushNamed(context, routeName).

Розглянемо ще один приклад віджета, який реалізує сторінку з формою для створення нової кімнати. Для створення форм, у даній роботі використовується пакет flutter_form_bloc/flutter_form_bloc.dart, який реалізує деякі з часто використовуваних типів полів, та функціонал для зручного керування станом форми.

На початку файлу необхідно виконати підключення всіх необхідних пакетів та віджетів:

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:flutter_form_bloc/flutter_form_bloc.dart';
import 'package:http/http.dart' as http;
import 'package:TesTseT/theme/style.dart';
```

Серед них є такі, що зустрічаються вперше у цій роботі:

- dart:convert – реалізує кодери та декодери для роботи з різними форматами даних, наприклад, JavaScript Object Notation (JSON);
- http/http.dart – реалізує інтерфейс для роботи з запитами до сервера, саме за допомогою цього пакету відбувається «спілкування» між додатком та API.

Далі опишемо клас, який буде основним на цій сторінці та буде наслідувати один із класів пакету для створення форм:

```
class RoomsFormBloc extends FormBloc<String, String> {
  final title = TextFieldBloc();
  final tests = MultiSelectFieldBloc<String, dynamic>(items: []);
  Map<String, int> testsMap = new Map<String, int>();
  RoomsFormBloc() : super(isLoading: true) {
    addFieldBlocs(fieldBlocs: [title, tests]);
  }
  …
}
```

На самому початку класу описуються змінні – поля класу, які є екземплярами класів двох видів полів форми – текстове поле та поле для вибору кількох варіантів (являє собою набір полів типу checkbox). Крім цих змінних є ще одна, яка є екземпляром класу Map, який є стандартним бібліотечним класом мови Dart. Також, у цій частині коду записано конструктор класу.

Далі опишемо функцію, яка буде відправляти запит на отримання списку тестів, створених поточним користувачем, та буде додавати ці тести у форму для вибору:

```
@override
void onLoading() async {
 try {
  final http.Response response = await http.get(
   'http://192.168.0.105/api/v1/user-tests/1',
   headers: <String, String>{
    'Content-Type': 'application/json; charset=UTF-8',
   },
  );
  if (response.statusCode == 200) {
   final Map<String, dynamic> responseJson =
      json.decode(response.body) as Map<String, dynamic>;
   List<String> newTestsList = [];
   for (Map<String, dynamic> data in responseJson['tests']) {
    testsMap[data['title']] = data['id'];
    newTestsList.add(data['title']);
   }
   tests.updateItems(newTestsList);
   emitLoaded();
  } else {
   throw Exception('Failed to load tests');
```

```
      }
    } catch (e) {
      emitLoadFailed();
      if (e is TypeError) {
        lastError = e.toString();
      } else {
        lastError = e.message;
      }
    }
  }
}
```

У наведеному коді також додано обробку можливих помилок, це зроблено з використанням структури try-catch. Це є важливою частиною коду, оскільки, при виникненні будь-яких помилок зі сторони API, додаток повинен їх обробити та повідомити про виникнення проблем користувачеві.

Схожим чином реалізується відправлення запиту на збереження даних у іншому методі класу – onSubmitting:

```
final http.Response response =
  await http.post('http://192.168.0.105/api/v1/rooms',
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode({
      'user': userId,
      'title': name.value,
      'tests': testsId,
    }));
```

Для цієї операції ми використовуємо тип запиту POST та вказуємо дані для збереження у тіло запиту – body. Також, використовується функція jsonEncode із пакету dart:convert, який перетворює даних у JSON формат.

У методі build віджета описується вся структура сторінки з формою. В результаті ми отримуємо сторінку наступного вигляду(рисунок 3.2).

Рисунок 3.2 – Приклад екрану додатку з формою

Таким чином, було розглянуто основні принципи побудови інтерфейсу у мобільному додатку, а також наведено приклади взаємодії додатку з API сервером.

3.2.2 Розробка серверного API

Розробка серверного API не є основною ціллю даної роботи, а виступає лише складовою частиною, яка дозволяє реалізувати запланований додаток. Саме з цієї причини, у даному розділі знаходиться лише опис обраної

технології з прикладами коду та словесний опис функціоналу API додатку, який містить прикладів коду.

Для реалізації серверної частини додатку використовується популярний сучасний фреймворк – Laravel. Це фреймворк, використовую концепцію Model-View-Controller (MVC). Він надає можливість швидко та зручно створювати web-сайти на мові PHP. Laravel володіє великим набором функцій, плагінів та шаблонів, які дозволяють втілювати в життя проекти різної складності. Цей фреймворк має критий код і розповсюджується безкоштовно для всіх розробників.

Окрім створення сайтів, Laravel, також, використовують для створення API, яке використовуватиметься іншими ресурсами чи додатками.

У Laravel, для реалізації шляхів, використовується архітектура Representational State Transfer (REST) API. Усі доступні для запитів на сервер шляхи описуються в окремих файлах, коли розробляється саме API, а не сайт, то всі шляхи, частіше за все, описуються у файлі з назвою api.php. Опис найпростішого шляху виглядає наступним чином:

```
Route::get('foo', function () {
    return 'Hello World';
});
```

Функціонально, цей шлях не має сенсу, тому, частіше в описі шляху використовують також відсилки до контролерів та їх методів. Розглянемо більш складний приклад:

```
Route::group(
['middleware' => 'auth', 'prefix' => 'v1/' ],
function ($router) {
    Route::put('user/{user}/settings', 'UserController@changeUserSettings');
    Route::get('user/{user}/ rooms ', 'UserController@getRooms');
    Route::get('user/{user}/tests', 'UserController@getTests');
    Route::delete('user/{user}', 'UserController@delete');
});
```

У наведеному вище прикладі опису шляхів у Laravel, наявні 4 різних шляхи, які об’єднані в одну групу. Для групи задано декілька параметрів: middleware – проміжний обробник запиту, та prefix – доповнення до основного шляху, який буде використовуватися для всіх внутрішніх шляхів. Так, у даному прикладі, використовується middleware auth, який відповідає за перевірку, чи наявний у запиті токен і чи коректний він.

У групі використовується декілька типів запитів, це PUT, GET та DELETE кожен з них визначає метод HTTP запиту, це вказує серверу на те, яку дію ми хочемо виконати з ресурсом. При цьому, для одного шляху може бути визначено декілька методів та різні методи контролерів для їх обробки.

Після визначення методу запиту, першим параметром є рядок, який визначає символьне представлення шляху. Наприклад, шлях визначений як «user/{user}/settings» матиме наступний вигляд у запиті: «http://localhost/api/v1/user/123/settings». Частину «localhost» буде замінено на адресу конкретного ресурсу, частина «123» вказує на поле id у таблиці користувачів бази даних нашого API. Таким чином, сервер розуміє, що прийшов запит на зміну даних користувача з id 123.

Для роботи з базою даних у фреймворку Laravel використовується реалізація технології Object-Relational Mapping (ORM) під назвою Eloquent. Це дозволяє робити запити до бази даних з використання вже існуючих методів класів, а не написання повних запитів власноруч. Наприклад, щоб отримати запис з бази даних про кімнату з id 10, достатньо написати «Room::find(10)», де «Room» - клас, наслідник «Illuminate\Database\Eloquent\Model». Якщо функціонал вимагає написання конкретного запиту вбазу даних, то це також можливо методами класу «Illuminate\Support\Facades\DB». Наприклад:

```
$users = DB::table('users')
            ->select(DB::raw('count(*) as user_count, status'))
            ->where('status', '<>', 1)
            ->groupBy('status')
```

```
->get();
```

Окрім описаного, фреймворк Laravel володіє значною кількістю інших корисних можливостей, таких, як:

– створення міграцій та сідерів (seeder), які виконують зміну структури бази даних та заповнення таблиць вказаними даними відповідно;

– створення cron завдань, які виконують заданий код з деяким інтервалом;

– опис відносин між моделями Eloquent, наприклад, можна описати відношення між моделями Post та Comment, як один до багатьох, оскільки кожен пост (запис у блозі, наприклад) може мати багато коментарів, тоді код «Post::find(1)->comments» поверне одразу всі коментарі, написані до цього посту, такий функціонал є дуже зручним у роботі;

– а також черги, валідація запитів, слухачі подій та багато інших.

4 ТЕСТУВАННЯ ДОДАТКУ

4.1 Процес тестування

Тестування додатку проводилося безпосереднім його використанням користувачем. При цьому, було перевірено роботу усіх функцій, запланованих для першої версії додатку:

- реєстрація;
- вхід;
- створення тестів та кімнат;
- додавання тестів у кімнати;
- запрошення користувачів у кімнати;
- проходження тестів.

Перший екран застосунку складається з логотипу додатку, та двох кнопок, які перенаправляють користувача на екран входу або реєстрації. Сторінка має наступний вигляд (рисунок 4.1).

Рисунок 4.1 – Вигляд основного екрану додатку

Новому користувачу необхідно зареєструвати, тож спершу він натискає другу кнопку – «Sign Up», після чого виконується відкриття екрану з формою реєстрації (рисунок 4.2).

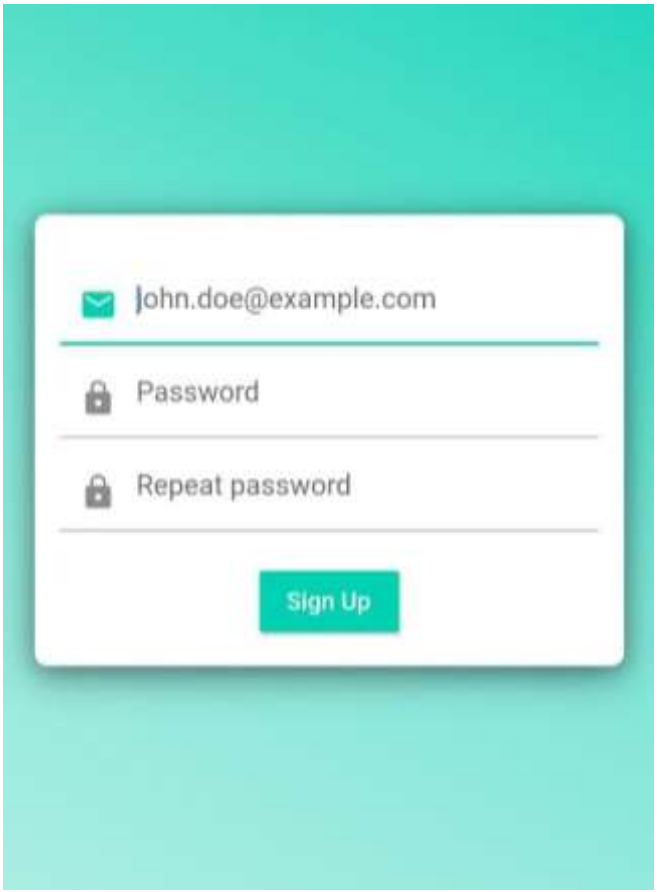

Рисунок 4.2 – Вигляд екрану реєстрації

При реєстрації виконується перевірка введених даних за заданим правилами, до яких входять:

– коректність email-адреси;
– мінімальна довжина паролю;
– рівнозначність полів «пароль» та «підтвердження паролю»;
– унікальність email-адреси у базі даних.

Після виникнення однієї із зазначених помилок, вона відображається біля відповідного поля (рисунок 4.3).

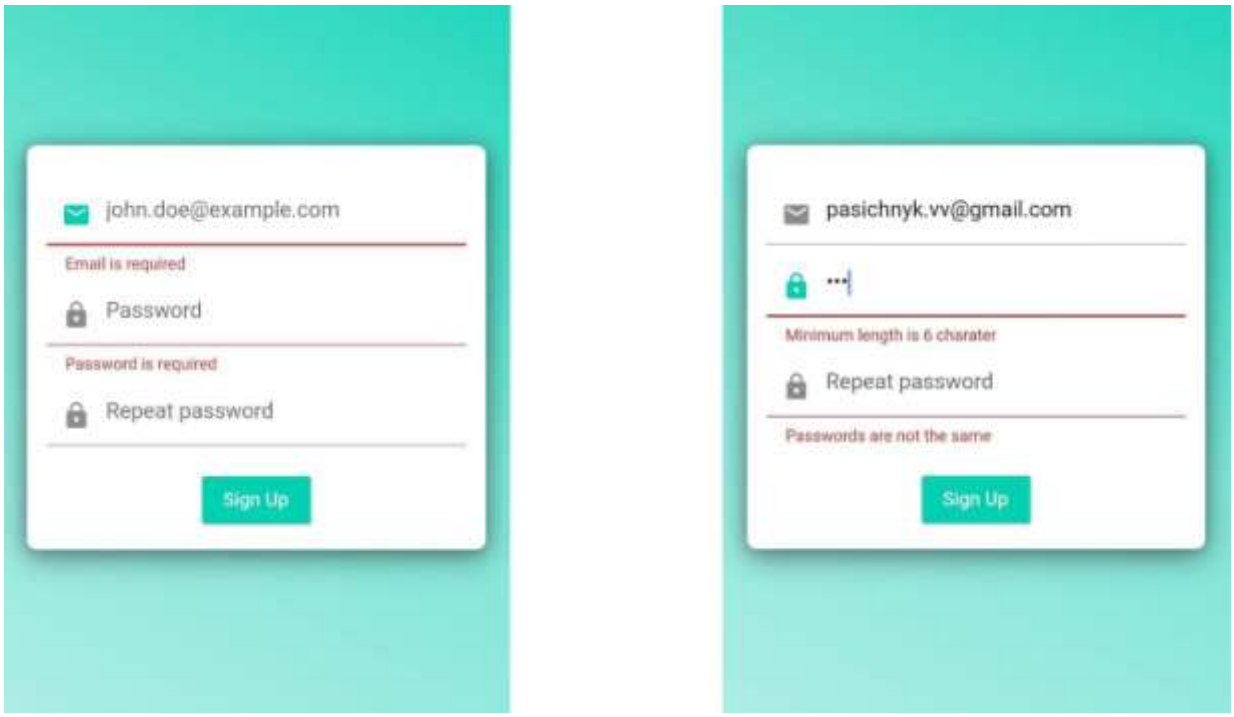

Рисунок 4.3 – Приклад помилок при реєстрації: обов'язкові поля (зліва) та некоректні значення (справа)

Після успішної реєстрації чи входу до облікового запису, користувач має доступ до інших екранів додатку, наприклад, до списку кімнат (рисунок 4.4).

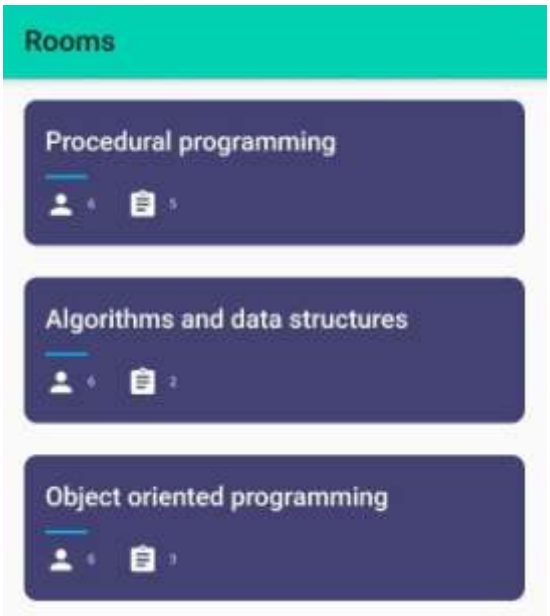

Рисунок 4.4 - Екран зі списком кімнат користувача

Після натискання на одну із карток зі списку, користувач переходить до самої кімнати, де розміщено тести для проходження та є можливість запросити інших користувачів у кімнату (рисунок 4.5).

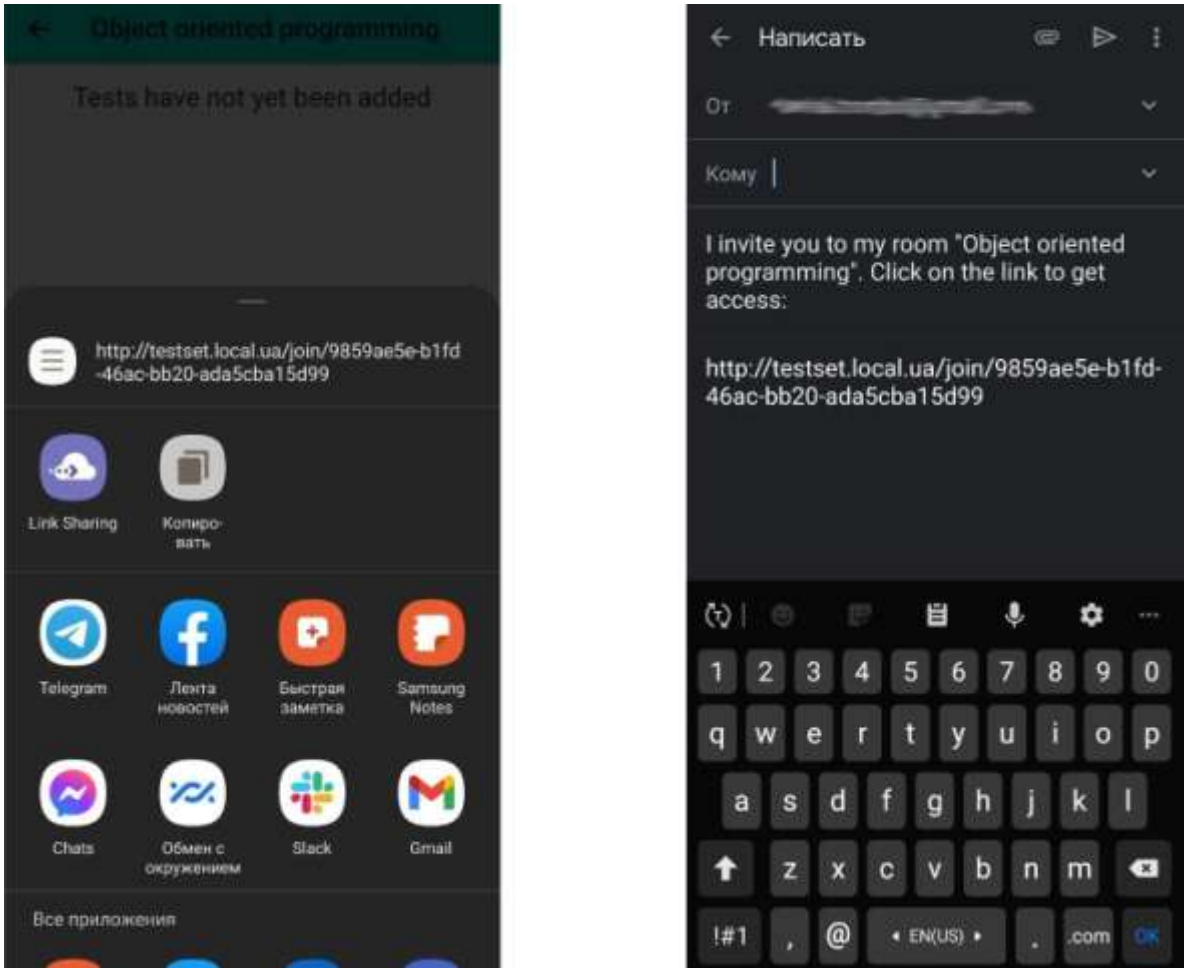

Рисунок 4.5 – Приклад функціоналу запрошення користувачів до кімнати

При наявних тестах у кімнати, можна перейти до їх проходження, де один за одним будуть відображатися питання (рисунок 4.6). Окрім цього, на екрані додатку буде відображатися назва тесту, номер поточного питання, загальна кількість питань та залишок часу на проходження, якщо тест обмежений у часі.

Рисунок 4.6 – Екран проходження тесту з питанням, яке має один варіант відповіді

Після завершення тесту, користувач бачить кількість правильних відповідей, загальну кількість питань та відсоток правильних відповідей (рисунок 4.7).

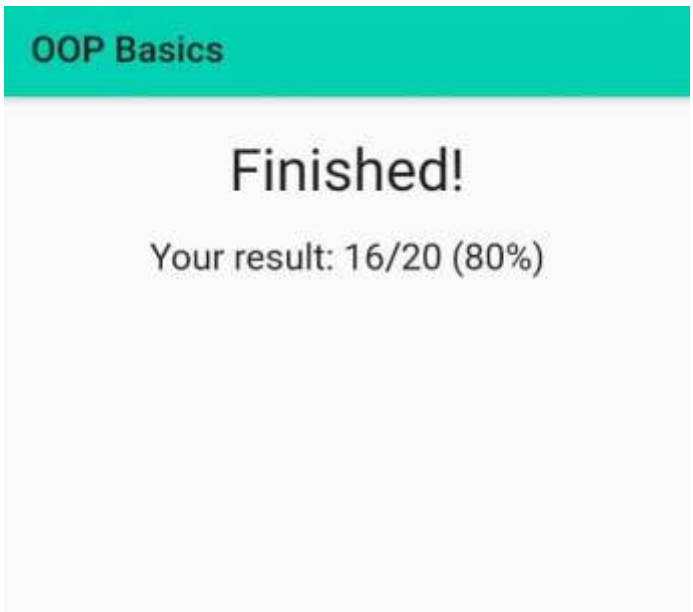

Рисунок 4.7 – Приклад екрану з результатами тесту

4.2 Результати тестування

У результаті тестування додатку не було виявлено помилок у роботі додатку. Різні екрани додатку завантажуються достатньо швидко, як і дані з серверного API. Використання застосунку не викликає складнощів, інтерфейс є інтуїтивно зрозумілим та зручним для користування.

4.3 Висновок

Як стало зрозуміло з тестування додатку, вдалося розробити зручний та функціональний застосунок для проведення тестів.

ВИСНОВКИ

У даній кваліфікаційній роботі було розроблено мобільний додаток для тестування з використанням фреймворку Flutter. Окрім цього, було розглянуто найпопулярніші фреймворки для розробки кросплатформенних мобільних додатків, існуючі підходи до реалізації кросплатформенності у обраних фреймворках, а також, мобільні додатки для розробки та проходження тестів.

Кросплатформенність – здатність програмного забезпечення працювати з кількома апаратними платформами або операційними системами. Забезпечується завдяки використанню високорівневих мов програмування, середовищ розробки та виконання, які підтримують умовну компіляцію, компоновку та виконання коду для різноманітних платформ.

Для розробки кросплатформенних мобільних додатків існує багато підходів та фреймворків. Кожен має свої переваги та недоліки: деякі з них легкі в розробці, оскільки використовують web-технології, інші показують кращі результати продуктивності, також, деякі дозволяють розробляти застосунки для більшої кількості ОС.

У результаті виконання даної роботи, було розроблено кросплатформенний мобільний додаток для проведення тестувань серед користувачів. Даний мобільний додаток показує гарні результати продуктивності та зручний у користуванні.

При подальшій розробці додатку, будуть додані нові можливості для користувачів: такі як перегляд результатів тестів адміністратором кімнати, можливість перегляду власних результатів тестування, статистика по проходженню тестів, а також отримання push- та email-повідомлень і нові типи запитань.

ПЕРЕЛІК ПОСИЛАНЬ

1. Biessek A. Flutter for Beginners. *Packt*. 2019. С. 40–281. URL : https://www.packtpub.com/product/flutter-for-beginners/9781788996082. (дата звернення: 31.10.2020).

2. Eisenman B. Learning React Native, 2nd Edition. *O'Reilly Media, Inc.* 2017. С. 42–86. URL : https://www.oreilly.com/library/view/learning-react-native/9781491989135/. (дата звернення: 09.10.2020).

3. Hermes D., Mazloumi N. Building Xamarin.Forms Mobile Apps Using XAML. *Apress*. 2019. С. 53–107. URL : https://www.apress.com/gp/book/9781484240298. (дата звернення: 17.09.2020).

4. Napoli M. L. Beginning Flutter A Hands On Guide to App Development. *Wiley*. 2019. С. 11–304. URL : https://www.wiley.com/en-ua/Beginning+Flutter% 3A+A+Hands+On+Guide+to+App+Development-p-9781119550853. (дата звернення: 10.09.2020).

5. Payne R. Beginning App Development with Flutter. *Apress*. 2019. С. 50–246. URL : https://www.apress.com/gp/book/9781484251805. (дата звернення: 07.11.2020).

6. Windmill E. Exploring Cross-Platform Development with Flutter, React Native, and Xamarin. *Manning*. 2020. С. 12–51. URL : **Ошибка! Недопустимый объект гиперссылки.**. (дата звернення: 17.11.2020).

7. Windmill E. Flutter in Action. *Manning*. 2020. С. 50–78. URL : https://www.manning.com/books/flutter-in-action. (дата звернення: 20.10.2020).

8. Ravulavaru A. Learning Ionic. *Packt*. 2015. С. 8–41. URL : https://www.packtpub.com/product/learning-ionic/9781783552603. (дата звернення: 29.09.2020).