ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: **«Вдосконалення мобільного вебзастосунку погодного інформаційного сервісу METEOTREND»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2023
Синенко Владислав Анатолійович
Керівник: Одарущенко Олег Миколайович
Рецензент: Муравльов Володимир Вячеславович

Полтава – 2024 року

ВСТУП

Актуальність теми дослідження. Тенденція підвищення «мобільності» суспільства, коли дедалі більша частина сервісів та послуг (як повсякденних, так і нечасто вживаних) переноситься у смартфони і доступна майже в будь-який зручний момент часу, не лише не сповільнюється після понад десяти років бурхливого росту. Більше того, експансія різних видів систем штучного інтелекту і можливість їх інтеграції в смартфони продовжує заохочувати дедалі більше постачальників послуг з різних галузей якщо не повністю переходити на мобільні платформи, то принаймні забезпечити там свою присутність за рахунок сторінок в соціальних мережах та сервісах, веб-застосунках та мобільних додатках. За досить короткий проміжок часу магазини техніки, одягу та продуктів стали продавати чи не більше товарів через інтернет замовлення, ніж безпосередньо покупцям на касі [1]. При цьому якщо ще зовсім недавно навіть для інтернет замовлення краще було використовувати браузер на ПК, бо громіздкі вебсайти були вкрай незручними для перегляду на мобільних пристроях, то сьогодні більшість сервісів мають окрему мобільну версію свого вебсайту, а то й повноцінний мобільний додаток, який забезпечує весь необхідний функціонал з урахуванням специфіки портативних пристроїв.

Втім, не варто забувати, що за кожним етапом еволюції інформаційних систем стоять нові технології їх розробки та тестування. За збільшенням числа поколінь мобільних платформ важливо розуміти, що число операційних систем, апаратних конфігурацій та протоколів обміну даними також зростає у загрозливій прогресії. Зосередитися на лише одній платформі означає суттєво спростити процес розробки та тестування, але також означає втрату потенційних клієнтів, які використовують інші платформи (в протистоянні мобільних платформ мова частіше за все йде про Android та iOS). Націлитися на обидві платформи – означає подвійні витрати на розробку та супровід програмного забезпечення [2].

Часто обирають компромісний варіант, коли програмне забезпечення створюється в якомусь абстрактному фреймворку, який при компіляції транслюється в виконуваний код для однієї або декількох підтримуваних платформ [3]. Таким чином, витрати по часу лише трохи перевищують аналогічні при розробці під одну native платформу, а результат охоплює стільки платформ, скільки підтримує цільовий фреймворк.

Саме такий підхід було обрано для розробки мобільного додатку для погодного інформаційного сервісу Meteotrend. На той момент у зв'язку з кращим знанням предметної області було використано фреймворк React Native з використанням мови програмування JavaScript. Мобільний додаток успішно пройшов стадії проектування, розробки та тестування, а також прийомку замовником. Проте на стадії розміщення його в Google Play довелося вносити суттєві зміни, пов'язані з інтеграцією сервісів Google, а також з'явилося побажання одночасно з запуском додатку для Android розмістити версію для iOS в Apple Store. Також виникли додаткові побажання до функціоналу.

Мета роботи – встановити необхідний обсяг змін для відповідності оновленим вимогам, обрати альтернативну технологію для переносу кодової бази, завершити розробку додатку з урахуванням вимог, та розмістити оновлену версію на онлайн-платформах з розповсюдження програмного забезпечення для мобільних пристроїв.

Завданнями кваліфікаційної роботи є:

1. Аналіз наявного функціоналу та змін, які потрібно внести.
2. Вибір фреймворку для модернізації додатку.
3. Модернізація існуючого додатку на новому фреймворку, адаптація існуючих ресурсів, додавання нового функціоналу та публікація.

Об'єкт дослідження – процеси вирішення задач бізнесу за рахунок використання мобільних додатків, задачі переносу програмного забезпечення між різними платформами.

Предмет дослідження – методи модернізації мобільного додатку.

Методи дослідження: системний аналіз (розділи 1, 2), моделювання (розділи 2, 3).

Інформаційна база, що використовувалася: публікації в періодичних виданнях, вебресурси, а також закордонні джерела про розробку програмного забезпечення.

Практична значущість: модернізовано та опубліковано мобільний додаток для отримання прогнозів для вебсервісу Meteotrends. Показано обсяг додаткової роботи, пов'язаної з неоптимальним вибором початкової технології для розробки. Окремо приділено увагу важливій темі підтримки функціоналу для людей з обмеженими можливостями.

Апробація результатів дослідження відбувалася шляхом оприлюднення доповідей на міжнародній та студентській конференціях.

Публікації. За результатами проведеного дослідження опубліковані тези: «Модернізація мобільного додатку погодного інформаційного сервісу METEOTREND». Матеріали XXI щорічного міждисциплінарного семінару «Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій ННІЕУПтаІТ ПДАУ», Полтава, 20 листопада 2024 р., С.25-29.

Структура та обсяг роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Обсяг роботи становить 75 сторінок, 6 таблиць, 10 рисунків та 2 додатки. Список використаних джерел налічує 52 найменування.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧОГО ФУНКЦІОНАЛУ ТА НЕОБХІДНИХ ЗМІН

1.1 Аналіз існуючого функціоналу мобільного застосунку

В рамках попередньої кваліфікаційної роботи вже було розглянуто історію розвитку мобільних пристроїв, показано динаміку їх бурхливого росту, особливо в період останніх двох десятиліть, та виділено ОС Android та iOS як провідні мобільні платформи протягом понад 10 років. При цьому мобільні пристрої на ОС Android суттєво переважають за чисельністю та варіативністю, але помітно поступаються основному конкуренту за прибутковістю на пристрій [4], (рис 1.1).

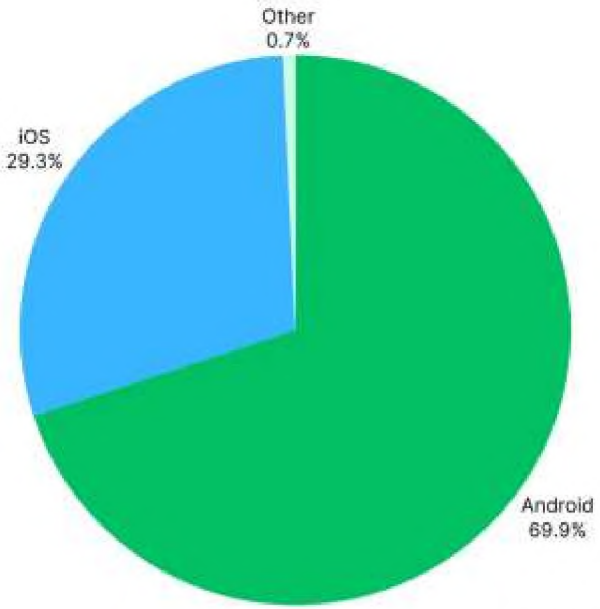

Рисунок 1.1 – Частка мобільних ОС у світовому ринку

Виходячи з попередньо наведеної інформації, отримавши свого часу замовлення на створення мобільного додатку для інформаційного погодного сервісу Meteotrend, платформи окрім Android та iOS не розглядалися, в силу того, що кожна додаткова платформа потребуватиме досить суттєвих витрат ресурсів

(у першу чергу – часових) на розробку, при цьому не даючи обсягу користувачів, відповідного цим витратам.

Особливості існуючої реалізації (як переваги, так і недоліки) буде розглянуто в наступних розділах. Нижче наведено функціональні можливості, які на даний момент доступні в останній версії додатку Meteotrend.

ФВ.01. Формування даних метеорологічного прогнозу для обраної місцевості виконується стороннім сервісом, обмін даними з яким відбувається через API.

ФВ.02. При першому запуску користувач повинен мати можливість ознайомитися з політикою приватності його даних, а також прийняти умови згоди на використання застосунку.

ФВ.03. При першому запуску користувач має ввести інформацію про поточне розташування через доступ до даних геолокації пристрою або ж через приблизне визначення локації на основі IP-адреси.

ФВ.04. Після виконання вимог Ф.В.02 та Ф.В.03, користувач повинен потрапити на домашній (основний) екран додатку.

ФВ.05. Якщо дані додатку не було видалено, при наступних запусках користувач одразу повинен переходити на домашній (основний) екран додатку.

ФВ.06. Домашній (основний) екран застосунку повинен відображати поточне метеорологічне становище для обраного розташування.

ФВ.07. Метеорологічне становище повинно включати в себе температуру, погодні умови, атмосферний тиск, швидкість та напрямок вітру, вологість та видимість.

ФВ.08. Для локацій, розташованих біля водойм, також повинна відображатися температура води і висота хвиль (якщо API відправляє – потрібно відображати).

ФВ.09. Додатково повинні відображатися відомості про час сходу і заходу сонця і місяця, а також фази місяця.

ФВ.10. Домашній (основний) екран повинен відображати погодинний прогноз (температура і погодні умови) на найближчі 24 години.

ФВ.11. На домашньому (основному) екрані повинен відображатися прогноз на найближчі 7 днів: мінімальна та максимальна температура, погодні умови.

ФВ.12. При натисканні на елемент прогнозу на наступні 7 днів користувач повинен переходити на екран, подібний до домашнього, але відомості про метеорологічний стан на ньому повинні бути поділені на 4 секції: ніч, ранок, обід, вечір.

ФВ.13. Бокове меню повинно бути доступне з будь-якого екрану (окрім зазначених у Ф.В.02 та Ф.В.03), забезпечуючи можливість навігації до наступних екранів: Обрані локації, Налаштування, Про додаток, Оцінити додаток, Поділитися з друзями.

ФВ.14. Бокове меню повинно бути доступне через іконку меню в заголовкові домашнього екрану, а також жестами конкретної платформи (наприклад, свайп з лівого краю екрану смартфону) на будь-якому екрані додатку.

ФВ.15. Користувач повинен мати можливість дізнатися погодні умови у будь-якому населеному пункті світу, включеному до бази сервісу, за допомогою функції пошуку.

ФВ.16. Екран пошуку повинен відображати список локацій, які відповідають введеному пошуку, та давати можливість обрати одну з локацій у цьому списку, після чого детальні метеорологічні дані мають бути завантажені на домашній екран.

ФВ.17. Після того, як зі списку пошуку обрано локацію, користувач повинен мати можливість додати її у обране, для подальшої можливості швидкого доступу до її погодних даних.

ФВ.18. Незалежно від того, чи додав користувач локацію з пошуку в обране, чи ні, якщо по ній було здійснено деталізоване відображення, вона повинна відображатися в спискові недавнього пошуку.

ФВ.19. Екран «Про додаток» повинен надавати користувачеві можливість переглянути політику приватності його даних, а також згоду на користування додатком.

ФВ.20. Екран налаштувань повинен надавати можливість обрати, в яких одиницях вимірювання повинні відображатися числові характеристики погодних умов.

ФВ.21. Екран пошуку додатково повинен надавати можливість встановити розташування пристрою користувача, за допомогою даних геолокації чи через приблизне розташування за IP-адресою (якщо геолокація заборонена користувачем).

1.2 Переваги та недоліки реалізації в React Native

На початку розробки першої версії мобільного додатку було проведено порівняльний аналіз наявних технологій програмування, які могли забезпечити необхідний функціонал, а також мали потенціал для реалізації додаткових можливостей (якби такі знадобилися). Було розглянуто зручність розробки, наявність достатньої бази знань та прикладів. Не в останню чергу враховувалося володіння мовою програмування, яка використовується у кожному з розглянутих фреймворків.

Зрештою, після того, як було відкинуто більшість варіантів, які не задовольняли одну чи більше вимог, залишалося зробити вибір між двома варіантами, що залишилися: React Native та Xamarin. На той момент було обрано перший варіант, але через понад рік розробки можна констатувати, що далеко не всі переваги, через які було зроблено такий вибір, виявилися суттєвими, а от недоліки зрештою привели до того, що простіше було виконати перенос кодової бази на інший фреймворк, ніж вирішувати всі проблеми, що склалися.

Почнемо з переваг. Швидка збірка коду – це справді так. Після першого запуску компілятора нова версія додатку розміщується майже миттєво, якщо зміни в код не вносилися, і все ще дуже швидко, якщо код було змінено. Оскільки

трансляція виконується в байт-код, внесені зміни зачіпають лише окремі фрагменти коду, що прискорює процес.

Чимало бібліотек, розроблених як спільнотою, так і самими розробниками – виявилося не такою суттєвою перевагою, як здавалося спочатку. Сам фреймворк React Native досить регулярно оновлюється, і, хоча кожне оновлення містить чіткий алгоритм дій [5, 41], які розробник має виконати для того, щоб оновити свої проекти до нової версії, іноді це не так легко зробити, а іноді – призводить до помилок, після яких доводиться відновлювати проект з резервної копії, якщо така є. За великим рахунком, у кожному проекті React Native у розробника є два варіанти: регулярно оновлювати свій проект до найновіших версій, або ж обрати одну версію LTS (Long-Term Support, довготривала підтримка), і робити проект виключно у цій версії. Обидва варіанти мають свої недоліки:

- якщо проект виконується в зафіксованій версії, його функціонал обмежується можливостями, які доступні на момент фіксації. Якщо в проекті використовуються якісь бібліотеки сторонніх розробників, вони оновлюються на розсуд їх авторів. На певному моменті автор бібліотеки може встановити обмеження, за якого бібліотеку можна оновити до наступної версії лише за умови оновлення самого фреймворку або ж пов'язаних із ним бібліотек. Також у оновленнях можуть міститися виправлення критичних багів, які також будуть недоступними без оновлення;
- якщо проект постійно оновлюється, деякі бібліотеки можуть не встигати за оновленнями фреймворку. В такому варіанті розробнику потрібно не тільки безпосередньо писати код проекту, але й регулярно стежити, аби між залежностями проекту не виникало конфліктів.

Як показує практика, більшість комерційних проектів, які пишуть на React Native, обирають варіант з фіксацією версії [6, 42], щоб уникнути додаткового навантаження, пов'язаного з контролем залежності. Хоча мобільний додаток для отримання прогнозу погоди, безперечно, не відноситься до складного програмного забезпечення, але за період його розробки було зроблено спробу тримати проект оновленим, яка не привела до успішного результату. Набір

бібліотек, які безпосередньо складають ядро React Native, мінімізовано, щоб забезпечити їх стабільну роботу, але як наслідок, навіть базові бібліотеки, необхідні для реалізації примітивних інтерфейсів, відстають у оновленні від фреймворку.

Можливість одразу відображати як функціональні так і візуальні зміни без повної перекомпіляції і повторного розміщення на пристрої чи емуляторі. Як і очікувалося, це суттєво економить час при активній розробці. Втім, як виявилося вже під час модернізації додатку, подібний функціонал (Hot Reload) надає і альтернатива React Native.

Принцип побудови ієрархії компонентів таким чином, що простіші елементи формують складніші, сам по собі дуже непоганий. Якщо будувати компоненти універсальними, а не націленими на єдине місце виконання, то можна суттєво скоротити число елементів, при цьому зберігши чітку впорядкованість. З іншого боку, кожний компонент потребує певні дані для своєї роботи, а їх може передати тільки батьківський компонент. Такі обмеження приводили до того, що кореневий компонент кожної сторінки передає першому рівню вкладеності всі дані, які можуть знадобитися не лише цьому рівню вкладеності, але й розташованим нижче [7]. При цьому перший рівень вкладеності частіше всього безпосередньо використовує лише частину даних, а всі інші просто передає далі, і так аж до останнього вкладеного компоненту. Хоча ці передачі даних не чинять різко негативного впливу на швидкодію програми, але регулярна передача наборів багатьох даних суттєво підвищує ймовірність помилки в коді.

Є можливість обійти модель поширення даних від батьківських компонентів до дочірніх, створена саме для ситуацій, описаних вище, за допомогою так званих «хуків» – функціональних елементів, які передають дані від основного відправника до крайнього отримувача без потреби в передачі даних через усю ієрархію. Але самі розробники React Native наголошують [8], що надмірне використання хуків – шкідлива практика, яка матиме негативний вплив на модель роботи з даними фреймворку.

Нарешті, остання з особливостей, характерних для React Native – це механізм, який відповідає за зв'язок між даними та їх візуалізацією. Змінні, оголошені в коді компонента, можуть бути частиною його «стану» (state), або ж звичайними змінними. Якщо в результаті певних обчислень змінюється значення «звичайної» змінної, то візуалізація компонента не оновлюється, і змінене значення не буде передано дочірнім компонентам, навіть якщо якийсь із них використовує цю змінну. Якщо ж змінилося значення змінної стану, то буде виконано повторний рендер компонента на екрані, а також передано донизу всі дані, які використовують дочірні компоненти. Якщо якісь із цих даних модифікують змінні стану дочірніх компонентів, то в них пройде аналогічне оновлення, і так аж до останніх елементів у ієрархії. Помилку, внесену під час проектування, чи при рефакторингу існуючого коду, за таких обставин буде вкрай важко виявити, тим більше, що єдиний універсальний спосіб відладки програм на React Native – це розкидати по коду метод Console.log, і відстежувати події та значення змінних в консолі.

Наостанок варто підкреслити архітектурну проблему: компіляція проекту, розробленого під фреймворк React Native – це не монолітний механізм, а композиція багатьох технологій різних поколінь. Отримати при компіляції свіжого шаблону в найновішій версії фреймворку ряд попереджень про те, що декілька ланок конвеєру компіляції застаріли і не повинні використовуватися – цілком нормальна практика. Це, безперечно, не означає, що потрібно повністю припинити використовувати компілятор React Native, але ймовірність відмови складного неоднорідного механізму явно вища, ніж моноліту аналогічної, а то й меншої складності. При невдалому оновленні до нової версії фреймворку іноді простіше відновити проект з резервної копії, ніж намагатися встановити, де саме, та чим викликана помилка.

Ієрархія компонентів та пов’язана з нею модель поширення даних React Native працюють як чіткий механізм, поки програма складається з кількох компонентів, але по мірі її ускладнення викликає більше недоліків ніж переваг. Частота та обсяг оновлення змушують або ж виділяти окремі ресурси на підтримання залежностей проекту в актуальному стані, або ж заморожувати

версію фреймворку та бібліотек, обмежуючи програмне забезпечення в новому функціоналі, та іноді – у виправленні критичних багів.

1.3 Додаткові функціональні вимоги

Як уже було згадано, потреба в модернізації мобільного додатку виникла з двох основних причин: складність в подальшій підтримці існуючої реалізації, а також необхідність змін існуючого функціоналу. Перша причина досить детально пояснена у попередньому розділі, а друга складається з кількох частин:

- виправлення та коригування існуючого функціоналу – під час тестування роботи додатку на фізичних пристроях та емуляторах було виявлено деякі недоліки поточної реалізації;
- додаткові побажання від замовника – в процесі ознайомлення з поточною реалізацією замовник також висловив декілька покращень, які варто додати до нової версії додатку;
- коригування функціоналу, необхідне для успішного проходження модерації перед публікацією додатку на платформах розповсюдження мобільних додатків.

Всі зміни, перераховані у попередніх пунктах, вирішено оформити у вигляді додаткових функціональних вимог, для простішого контролю за їх реалізацією та тестуванням.

ФВ.22. Вміст компонентів користувацького інтерфейсу (як текстових, так і графічних) повинен коректно відображатися при зміні розміру екрану, а також при застосуванні користувацьких параметрів шрифтів (збільшений розмір тексту, жирний шрифт тощо) [43].

ФВ.23. У випадку виникнення помилок при обміні даними з API сервером, додаток повинен проінформувати користувача про наявність проблеми, по можливості конкретизувати її так, щоб користувач міг зрозуміти причину. Після цього додаток має повернутися до робочого стану.

ФВ.24. Екран налаштувань також повинен давати можливість змінити локалізацію додатку однією з доступних мов. Список мов, як і дані для локалізації, надаються сервером.

ФВ.25. Для мов, які пишуться справа наліво, весь користувацький інтерфейс також повинен бути адаптований до такого відображення. Допускається миттєвий перезапуск додатку для встановлення формату відображення LtR або ж RtL.

ФВ.26. З метою зменшення навантаження на сервер, кожен запит має випадковим чином обирати один із десяти доступних серверів, ім'я яких відрізняється лише цифрою.

ФВ.27. Для забезпечення коректної роботи на пристроях в режимі обмежених можливостей (голосовий набір / зчитування), кожен елемент користувацького інтерфейсу, призначений для взаємодії з користувачем, повинен мати семантичний опис, який може бути зчитаний програмою, яка надає режим обмежених можливостей.

Висновки до розділу 1

У першому розділі проаналізовано існуючі функціональні вимоги, а також проведено ретроспективний розбір недоліків та переваг фреймворку React Native, на якому розроблювалася перша версія додатку Meteotrend.

Наявний функціонал було систематизовано по функціональним вимогам з метою полегшення подальшого відстеження коректності реалізації функціоналу та тестування. Також було впорядковано нові вимоги до функціоналу, який потрібно додати – як зі сторони замовника, так і зі сторони платформи, на якій планується публікація додатку.

Зібрані за час використання недоліки обраного фреймворку будуть використані в наступному розділі при аналізі альтернативних засобів розробки.

РОЗДІЛ 2
ВИБІР ФРЕЙМВОРКУ ДЛЯ МОДЕРНІЗАЦІЇ ТА АДАПТАЦІЯ ІСНУЮЧИХ РЕСУРСІВ

2.1 Вибір фреймворку для модернізації

Як уже було згадано в розділі 1, наявне рішення у вигляді розробки мобільного додатку на крос-платформеному фреймворку React Native мало деякі недоліки, які, кожен окремо, напевно що можна було б вирішити, або ж принаймні пом'якшити їх негативний вплив на проект, але, взяті всі разом, потребували забагато ресурсів для вирішення, і простіше було перенести існуючу кодову базу на якесь альтернативне рішення, додавши нові функціональні зміни, також згадані в попередньому розділі.

Оскільки задача вибору фреймворку для модернізації постала знову, то потрібно було ще раз розглянути список можливих варіантів, як це було зроблено минулого року. Але при цьому потрібно було враховувати ще два додаткові критерії, які були відсутні під час попереднього аналізу:

- як найкраща сумісність із існуючим кодом – краще не ідеальний фреймворк, для якого можна з мінімальними втратами перенести існуючі ресурси, бізнес-логіку, макети екранів, ніж ідеальний фреймворк, для якого все доведеться розроблювати з нуля;
- відсутність або мінімальний вплив недоліків, з якими зіткнулися під час реалізації наявної версії, розробленої в React Native – хотілося б витрачати більше часу безпосередньо на розробку коду, а не на контроль версій та залежностей, але при цьому мати доступ до більшості потрібного функціоналу «з коробки» замість того, щоб шукати сумнівні самописні рішення, чи ще гірше, писати їх з нуля.

Коротко згадаємо ті варіанти, які було включено до переліку можливих кандидатів за результатами минулорічного аналізу. Як показав короткий огляд ситуації на ринку розробки мобільних додатків [9], за цей рік не з'явилося

якихось нових рішень, які б стрімко набирали популярність, тому можна використовувати попередні результати.

React Native (JavaScript). Варіант, обраний минулого разу. До переваг відносяться швидка збірка коду, широкий вибір бібліотек спільноти та розробників, результат компіляції є native мобільним додатком, оскільки код React Native транслюється в рідний для цільової платформи байт-код. Переваги і недоліки цього фреймворку [10] було розглянуто в першому розділі, і детальніше зупинятися на цьому варіанті не потрібно.

Cordova (JavaScript). Один із перших крос-платформених фреймворків [11], з підтримкою стандартних бібліотек JavaScript, для взаємодії доступні базові плагіни та API. Основний недолік – формально в результаті генерується мобільний додаток, який по суті є браузером, в який почергово завантажуються сторінки, що веде до зниження швидкодії і підвищеного використання ресурсів. Серед переваг для вирішення поставленої задачі – те, що фреймворк використовує ту ж мову програмування, що й попередній варіант реалізації. Але потрібно розуміти, що перенести вдасться тільки бізнес-логіку (backend-функції), бо проектування і розмітка екранів користувацького інтерфейсу будуть принципово різними (ReactNative використовує спеціальний XML-подібний синтаксис, розроблений Facebook, для проектування інтерфейсу [12], який називається JSX). В усьому іншому цей варіант ще менш прийнятний за існуючий, бо конвеєр компіляції чи не складніший за такий в React Native.

Qt (C++). Як і попередній варіант, результат транслюється в мобільний веб-додаток. Qt на даний момент використовується на дуже широкому наборі пристроїв, тому не дивно, що з трохи урізаними можливостями його можна запустити і на мобільних пристроях [13]. Недоліки ті ж самі, що й в Cordova (браузерна основа), з іншого боку обсяг бібліотек для C++ має повністю закривати потребу в додаткових ресурсах. В рамках теперішньої задачі цей варіант ще більш непрактичний в силу того, що інтерфейс будується за зовсім іншим принципом, та й адаптація коду JavaScript до C++ виглядає навіть складнішою, ніж написати весь необхідний код заново.

Flutter (Dart). На відміну від інших розглянутих варіантів має повноцінні набори компонентів, написані з нуля, для кожної з цільових платформ, які повторюють поведінку відповідних компонентів на пристроях [14]. Це наймолодший із розглянутих варіантів, але дуже динамічно розвивається, не в останню чергу через суттєву підтримку компанії Google. Варіант дуже перспективний, але потребує додаткових витрат часу на вивчення мови програмування Dart, і додаткової оцінки витрат на адаптацію наявних ресурсів.

MAUI (колишній Xamarin) (C#). Серед переваг – високий рівень популярності мови-носія, великий набір бібліотек та плагінів, які можна зручно встановлювати через пакетний менеджер NuGet. Весь процес розробки можна вести всередині інтегрованого середовища розробки Visual Studio, процес компіляції максимально однорідний. На відміну від реалізацій на JavaScript, код для конкретної платформи компілюється, а не інтерпретується. Є можливість реалізувати всю спільну логіку в одному місці, а інтерфейси, специфічні для цільових платформ – реалізувати окремо [15].

З огляду на перелік доступних варіантів, прийнятність того чи іншого рішення можна оцінити, скориставшись таблицею 2.1.

У цій таблиці наведено перелік характеристик кожного з розглянутих фреймворків, важливих для прийняття об’єктивної оцінки. Оцінка кожного фреймворку за тією чи іншою характеристикою надається за шкалою від 1 до 5, де 1 – мінімальна відповідність, а 5 – максимальна (повна) відповідність. Як бачимо, MAUI набрав найбільше балів, навіть більше, ніж початковий варіант React Native (справедливо сказати, що на момент вибору React Native частина проблем, пов'язаних із його використанням ще була невідома). На цих результатах можна було б зупинитися, але отримані числа є не зовсім об’єктивними, оскільки не всі характеристики є однаково важливими у прийнятті рішення, а сума оцінок не враховує важливість характеристик. Для більш зваженої оцінки введемо також коефіцієнт важливості тієї чи іншої характеристики безпосередньо для прийняття рішення, за шкалою від 0 до 1, де 0 – повністю неважлива, може бути проігнорована, а 1 – максимально важлива.

Таблиця 2.1 – Оцінка фреймворків за характеристиками

Характеристика	React Native	Cordova	Qt	Flutter	MAUI
Переносимість коду бізнес-логіки	5	4	2	3	3
Переносимість розмітки користувацького інтерфейсу	5	3	1	3	3
Адаптація існуючих ресурсів	5	3	3	3	4
Спорідненість готового виконуваного файлу з цільовою платформою	4	4	3	4	5
Варіативність доступних бібліотек	3	2	3	3	5
Знання мови програмування	4	4	3	1	5
Однорідність конвеєру компіляції	2	2	3	4	4
Загальний результат	28	22	18	21	29

Коротко опишемо кожну характеристику:

- переносимість коду бізнес-логіки, оцінює, наскільки легко буде адаптувати код, який відповідає за обчислення та прийняття рішень (відв'язаний від користувацького інтерфейсу), важливість – 0,8;

- переносимість розмітки користувацького інтерфейсу, оцінює, наскільки легко буде перенести існуючі макети та розмітку на новий фреймворк (стиль та розміри шрифтів, відступи між компонентами та контейнерами, важливість – 0,9;

- адаптація існуючих ресурсів, оцінює, наскільки швидко можна адаптувати існуючі ресурси (і який обсяг існуючих ресурсів може бути адаптований без потреби створювати їх заново) до нового фреймворку, важливість – 0,7;

- спорідненість готового виконуваного файлу з цільовою платформою, оцінює, чи результат компіляції буде додатково транслюватися перед виконанням, чи одразу готовий до виконання засобами цільової платформи, важливість – 0,2, оскільки безпосередньо на задачу, яку потрібно вирішити, він слабо впливає, в функціоналі додатку немає точок, де вимагається максимальна продуктивність системи, а отже навіть код, який буде інтерпретуватися на цільовій платформі, матиме задовільну ефективність;

- варіативність доступних бібліотек, оцінює різноманіття існуючих бібліотек і те, наскільки легко та зручно ними користуватися (включаючи пошук, встановлення, сумісність, частоту та легкість оновлень), важливість – 0,7;

- знання мови програмування, оцінює рівень володіння мовою програмування, з якою працює той чи інший фреймворк, на даний момент, важлива характеристика, оскільки потреба повного вивчення мови програмування з нуля займе чимало часу, а слабке знання мови програмування призведе до того, що багато часу буде витрачено на пошук інформації, а частина написаного коду буде досить низької якості, важливість – 0,8;

- однорідність конвеєру компіляції, загалом, вторинна характеристика, яка безпосередньо не впливає на вирішення поставленої задачі, оцінює монолітність конвеєру компіляції, через який проходить написаний код перед тим, як перетворитися на виконуваний код, зрозумілий цільовій платформі, чим складніший і неоднорідніший конвеєр, тим вище ймовірність відмови однієї з його складових, і важче виправити цю помилку, важливість – 0,4.

Тепер, коли всі характеристики означені, а також встановлено їх важливість, можна обчислити зважені оцінки кожного з фреймворків. Результати в оновленій таблиці 2.2:

Таблиця 2.2 – Зважені оцінки фреймворків

Характеристика	React Native	Cordova	Qt	Flutter	MAUI
Переносимість коду бізнес-логіки	4,00	3,20	1,60	2,40	2,40
Переносимість розмітки користувацького інтерфейсу	4,50	2,70	0,90	2,70	2,70
Адаптація існуючих ресурсів	3,50	2,10	2,10	2,10	2,80
Спорідненість готового виконуваного файлу з цільовою платформою	0,80	0,80	0,60	0,80	1,00
Варіативність доступних бібліотек	2,10	1,40	2,10	2,10	3,50
Знання мови програмування	3,20	3,20	2,40	0,80	4,00
Однорідність конвеєру компіляції	0,80	0,80	1,20	1,60	1,60
Загальний результат	18,90	14,20	10,90	12,50	18,00

Проаналізуємо результати зваженого порівняння фреймворків за характеристиками. З незначним відривом React Native випереджає MAUI. Це в дечому логічно, оскільки він має вищу оцінку в переносимості коду бізнес-логіки і інтерфейсу (очевидно, що простіше перенести код на той же самий фреймворк, ніж на будь-який інший). Але оскільки вже вирішено, що React Native більше не може використовуватися для розробки проекту Meteotrend, буде обрано наступний варіант, MAUI, який зайняв друге місце з незначним відривом від лідера, але суттєво випередив усі інші альтернативи.

2.2 Переваги та недоліки реалізації в MAUI

Отже, після проведеного аналізу альтернативних фреймворків для модернізації мобільного додатку Meteotrend, було обрано MAUI (раніше відомий як Xamarin). Коротко розглянемо основні переваги та недоліки нової реалізації проекту порівняно з наявною версією в React Native.

Переваги:

- єдина мова розробки (C#), весь процес розробки, а також весь програмний код написано мовою C#, яку можна охарактеризувати як потужну, безпечну відносно типів даних, і знайомою всім ,net розробникам, за рахунок однієї мови розробки спрощується інтеграція бізнес-логіки та користувацького інтерфейсу;

- підтримка крос-платформеності, одна кодова база дозволяє розробку для Android, Ios, Macos та Windows, з мінімальними коригуваннями, ефективність роботи виконуваного коду – на рівні native, а також доступ до арі окремих платформ [16, 44] (що дає можливість застосовувати додаткові функції залежно від цільової платформи);

- вбудовані Model-View-Update (MVU) та Model-View-Viewmodel (MVVM) патерни структурованого дизайну, які суттєво спрощують розробку UI за рахунок прив'язки даних (Data Binding);

- інтеграція в екосистему Microsoft, проста інтеграція з сервісами Azure, інструментами visual studio та іншими технологіями Microsoft;

- рендеринг користувацького інтерфейсу в native режимі, завдяки тому, що компоненти інтерфейсу трансформуються в native елементи цільової платформи, швидкодія виконуваного коду перевершує фреймворки на базі Javascript;

- широкий вибір .Net бібліотек, завдяки сумісності з усією екосистемою .Net, бібліотеки для роботи з мережею, безпекою та обробкою даних доступні у всіх додатках MAUI;

- «гаряче перезавантаження», фірмова особливість WPF, Xamarin, а отже й MAUI – зміни в коді під час запущеного в режимі відладки додатку відразу ж відображаються в запущеному додатку, без необхідності перекомпілювати код і заново його розміщувати на цільовій платформі, що суттєво прискорює процес розробки;

- уніфікований процес розробки для мобільних пристроїв та ПК з мінімальними зусиллями.

Недоліки:

- менша спільнота, порівняно з React Native;

- складніше вивчення основ за умови браку знань про C# чи екосистему .NET;

- порівняно малий вік, перша стабільна версія була випущена всього 2 роки тому, тому в системі все ще можуть бути баги, проблеми з продуктивністю, а також окремі патерни розробки, які ще не були повністю перевірені спільнотою;

- кастомізація для конкретної платформи, хоча MAUI й надає native доступ до можливостей специфічних пристроїв, потрібні глибокі знання про безпосередні можливості платформ, в які транслюється універсальний код (Java/Kotlin для Android, Swift/Objective-C для iOS) [17];

- суттєво залежить від Visual Studio, складно працювати на більш «легких» середовищах розробки.

Якщо порівнювати особливості фреймворку, на якому реалізовано теперішню версію додатку, та фреймворку, який обрано для модернізації, можна узагальнити результати порівняння в наступній таблиці 2.3:

Таблиця 2.3 – Характеристики фреймворків

Характеристика	.NET MAUI	React Native
Основна мова	C#	JavaScript
Спільнота	Невелика, проте зростає	Велика і досвідчена
Продуктивність	Висока продуктивність, відображення користувацького інтерфейсу в native режимі	Проміжний транслятор JavaScript додає ще один рівень абстракції
Підтримувані платформи	Мобільні, ПК (Windows, macOS)	Мобільні, ПК (поки що в експериментальному варіанті)
Сторонні бібліотеки	Обмежені	Підтримується розширення
Складність навчання	Складніша для розробників без досвіду в .NET	Простіша для веб-розробників завдяки стеку JavaScript/React
Інструменти розробки	Visual Studio	Гнучкі (VS Code, Expo, CLI) [18]
Екосистема	Посилена інтеграція в екосистему Microsoft	Відкрите програмне забезпечення з підтримкою екосистеми npm
Кастомізація	Широкі можливості, проте потребує знання специфіки native платформи	Для базових потреб – безпосередньо з JavaScript, розширені можливості з native модулями
«Гаряче» перезавантаження	Підтримується	Підтримується

Таким чином, враховуючи, що розмір спільноти не впливає безпосередньо на розробку, і вже маючи досвід в .NET, можна обирати MAUI як платформу для модернізації існуючого проекту.

2.3 Адаптація ресурсів

Однією з важливих характеристик фреймворків, що були розглянуті в попередньому розділі була можливість швидкої адаптації якомога більшого числа існуючих ресурсів для економії часу, необхідного для модернізації

додатку. В цьому розділі коротко розглянемо особливості адаптації наявних ресурсів при переході до MAUI.

Візуальні ресурси, такі як зображення, піктограми тощо, можна перенести без потреби в додатковій модифікації, оскільки і React Native, і MAUI підтримують стандартні формати зображень. В проекті React Native зображення зберігалися за шляхом "/assets/images/", в новому проекті вони будуть розміщені за шляхом "/Resources/Images/".

Шрифти можна використовувати в їх теперішньому вигляді, додавши їх за шляхом "/Resources/Fonts/" та визначивши кожен в головному файлі MauiProgram.cs:

```
builder.ConfigureFonts(fonts =>
    {
        fonts.AddFont("CustomFont.ttf", "CustomFontAlias");
    });
```

Статичні файли з розширенням JSON або XML (з даними конфігурації, чи тестовими даними) можна використовувати без змін, розташувавши у потрібній папці проекту.

Точки API та фрагменти запитів можна використовувати частково [19]. REST API чи GraphQL запити можуть бути використані в MAUI аналогічним способом, з мінімальною адаптацією: якщо в ReactNative на рівні взаємодії з сервером найчастіше використовується fetch або Axios, то в MAUI ту ж функцію виконує HttpClient.

Файли локалізації в JSON можна використовувати в тому вигляді, в якому вони застосовуються в React Native (додавши динамічний парсинг), або ж включити їх до .resx файлу і працювати з ним далі як із одним із типів ресурсів (знадобиться конвертація з json до resx [45]).

Щодо інших ресурсів, які можна перенести з React Native до проекту в MAUI, то тут знадобиться більше додаткових дій. Компоненти користувацького інтерфейсу можуть бути частково адаптовані, але для більшості елементів потрібно буде переписувати їх для аналогу в новій платформі. Контейнери мають подібну поведінку, тому Grid, StackLayout та FlexLayout будуть працювати

приблизно так само, а стилі зі StyleSheet можна перенести у визначення Style в файлі xaml.

Логіка управління станами може бути частково адаптована, якщо додаток в React Native використовував Redux або Context API – для цих варіантів в MAUI є MVVM, Dependency injection, і навіть власна версія контейнерів станів Redux.NET. Втім, перша версія Meteotrends не використовувала жодного з цих фреймворків, тому цей код доведеться писати заново.

Логіка навігації – принципово різна. Бібліотеки навігації React Native використовують суттєво інакшу концепцію, ніж обидва варіанти, доступні в MAUI. Скоріше за все, для примітивних дерев навігації, на зразок того, який буде використовуватися в Meteotrend, буде достатньо використати NavigationPage, щоб не ускладнювати процес розробки роботою з Shell (більш популярний, але складніший спосіб переходу між сторінками) [20].

Логіка для окремих цільових платформ – може бути адаптована (розміщена у відповідних частинах проекту), але код доведеться переписувати для C#.

Таблиця 2.4 – Адаптація існуючих ресурсів

Ресурс	Повторне використання	Дії
Візуальні ресурси	Так	Розмістити у папці ресурсів проекту MAUI
Шрифти	Так	Додати до проекту та задекларувати в коді
Статичні JSON/XML файли	Так	Завантажити динамічно або конвертувати за потреби
Логіка API / Backend	Частково	Адаптувати мережевий код до HttpClient
Файли локалізації	Так	Завантажити динамічно або конвертувати до resx
Компоненти інтерфейсу	Ні	Переписати в XAML
Логіка управління станами	Частково	Переписати в ViewModels
Логіка навігації	Ні	Переписати в NavigationPage
Модулі окремих платформ	Частково	Переписати в платформах MAUI

Узагальнюючи, можна сказати, що модульна структура проекту першої версії додатку значно спрощує перенос на нову платформу, там де це можливо. Втім, через принципові відмінності між двома платформами найбільш важливі

ресурси все одно доведеться адаптувати вручну. Результуюча таблиця 2.4 коротко характеризує можливість/неможливість повторного використання ресурсу з проекту React Native у проекті MAUI та зазначає заплановані дії.

2.4 Підтримка функцій для користувачів з обмеженими можливостями

Тенденція дедалі більшого розширення функціоналу портативних пристроїв для користувачів з обмеженими можливостями набула такого рівня, що забезпечення коректної роботи мобільного додатку в режимі Accessibility було включено не просто до правил хорошого тону рекомендацій Google Play [21] та Apple Store [22], а знаходиться за крок від обов'язкового елементу, який має забезпечувати кожен додаток, який збирається публікуватися на цих платформах. У цьому розділі наведено загальні правила щодо підтримки додаткового функціоналу для користувачів з обмеженими можливостями, а також перераховано те, що буде використовуватися при модернізації додатку (саме в контексті використання MAUI). Після застосування релевантних змін в коді доцільно перевірити, наскільки ефективно реалізовано ці зміни, відкривши додаток за допомогою однієї з програм екранного читання і спробувавши взаємодіяти з основними функціями виключно за допомогою цієї програми.

2.4.1 Використання семантичних властивостей

Семантичні властивості – це додаткова текстова інформація про призначення того чи іншого візуального компоненту. У випадку з кнопкою, її текст буде автоматично розпізнано екранним читачем та озвучено користувачеві. Втім, якщо на кнопці буде зображення, чи якась комбінація символів (наприклад, імпровізована стрілка «-->»), то у екранного читача можуть виникнути складності з правильним розумінням призначення того чи іншого компоненту. Під екранним читачем (Screen reader) [23, 50] в даному випадку вживається загальне позначення компоненту операційної системи, який відповідає за розпізнавання візуальних елементів інтерфейсу, що відображаються на екрані, та

інформування користувача з обмеженими можливостями у зручний для нього спосіб. Різні операційні системи мають різні реалізації такого компоненту, але в тому чи іншому вигляді екранний читач присутній майже в кожній ОС, як мобільній, так і для ПК.

Для коректного розпізнавання призначення та вмісту візуальних компонентів використовуються декілька видів семантичних властивостей. Назви, наведені нижче, характерні для MAUI [24], але призначення таких властивостей відповідає аналогічному й в інших фреймворках.

SemanticProperties.Description. Описує візуальний контент для користувачів, які мають складність з розпізнаванням. Для кнопки це може бути її назва, для зображення – короткий опис того, що на ньому зображено.

SemanticProperties.Hint. Містить додаткову підказку для екранного читача стосовно призначення елементу. Для прикладу, кнопка «ОК» може бути недостатньо інформативною щодо розуміння того, що саме вона підтверджує. Можна вказати в якості підказки «Застосовує всі зміни та повертається до попереднього екрану».

SemanticProperties.HeadingLevel. Якщо інтерфейс містить чимало тексту, то на слух може бути складно зрозуміти, що з цього – заголовки, а що – безпосередньо основний контент. Завдяки цій властивості можна розмітити елементи інтерфейсу таким чином, щоб екранний читач розпізнавав, які компоненти відносяться до якої частини екрану, і забезпечував більш зрозумілу навігацію та розуміння тексту.

Ще одна важлива властивість своєчасно інформує екранного читача про те, що певна частина екрану щойно була оновлена, і потрібно повідомити про це користувача: SemanticProperties.LiveRegion вказує раніше ідентифіковану область екрану, яку потрібно буде розпізнати заново.

2.4.2 Використання властивостей автоматизації

Якщо семантичні властивості використовуються для коректного розпізнавання візуального контенту, то властивості автоматизації потрібні для

вибору правильної дії після того, як контент розпізнано. Встановлюючи правильні властивості автоматизації на тих елементах інтерфейсу, з якими можлива взаємодія, розробник покращує його інтерактивність, полегшуючи керування користувачам з обмеженими можливостями. Додатково ці властивості можуть використовуватися багатьма популярними засобами автоматизації тестування. Нижче наведено основні такі властивості.

AutomationProperties.Name. Вказує ім'я елементу для подальшої ідентифікації та взаємодії.

AutomationProperties.HelpText. Надає додаткову (допоміжну) інформацію про елемент (подібно до SemanticProperties.Hint).

AutomationProperties.IsInAccessibleTree. Бінарна властивість, що може бути в стані true або false. Визначає, чи включено елемент до дерева (ієрархії) Accessibility.

2.4.3 Використання контрастних кольорових схем

Навіть людям з хорошим зором складно розібрати інформацію, коли текст і фон мають подібні кольори, що вже казати про людей з ускладненнями зору. Існує загальноприйнятий набір рекомендацій по дизайну веб контенту WCAG – Web Content Accessibility Guidelines (в переважній більшості випадків він також застосовний до мобільних додатків), який визначає, що для тексту нормального розміру мінімальне співвідношення контрасту має бути 4.5:1, а для великого тексту – 3:1 [26, 47].

Також гарним тоном вважається підтримка динамічних тем. Більшість ОС для ПК, включаючи Windows, macOS та Ubuntu, а також мобільні ОС, включаючи Android та iOS, дають користувачеві можливість однією або декількома діями перемкнути пристрій на денну (світлу) чи нічну (темну) тему. Окрім того, що стандартні програми ОС автоматично розпізнають встановлену на даний момент тему, і адаптують свій зовнішній вигляд відповідно до неї (встановлюють світлий чи темний фон, змінюють колір тексту для більш контрастного відображення чи зменшення навантаження на очі), стороннє

програмне забезпечення також намагається розпізнавати поточний режим роботи ОС і адаптуватися під нього.

2.4.4 Використання динамічного масштабування тексту

Мобільні пристрої (особливо сімейство пристроїв на ОС Android) мають різні характеристики, які можуть безпосередньо впливати на відображення вмісту мобільних застосунків на екрані. Проектування інтерфейсу (включаючи стиль відображення, розміри шрифта, відступи тощо) виконане для одного типу пристроїв чи діагоналі екрану може дати зовсім неочікуваний результат на іншому. Тому для коректного відображення тексту на якомога більшій кількості пристроїв рекомендується уникати жорсткого задання розміщення елементів на екрані (наприклад явно вказувати, що певна кнопка повинна мати ширину 55 пікселів, чи бути розташована за координатами 16;25). Навпаки, якнайбільше компонентів повинні мати відносні параметри відображення та бути розміщеними в контейнерах, що підтримують динамічне масштабування (Grid, Stack тощо).

Там, де динамічного розміщення та автоматичного визначення розмірів компонентів недостатньо, можна також додатково визначати параметри візуалізації контенту (частіше всього – тексту) залежно від фактичних характеристик пристрою, де запущено додаток. Для прикладу, можна вказувати розмір шрифта компонента через `FontSize = “{DynamicResource TitleFontSize}”`.

2.4.5 Резервування мінімально допустимого розміру

Альтернативний спосіб збільшити ймовірність коректного відображення того чи іншого компонента на якомога більшому числі пристроїв – це вказувати мінімально допустимі розміри компонента. При відображенні сторінки середовище виконання постарається врахувати ці параметри (за винятком ситуації, коли бажані розміри неможливо встановити на даному пристрої в силу обмеження розміру або ж якщо це вступає в протиріччя з візуалізацією інших компонентів екрану [27, 46]).

Щоб встановити мінімально допустимі розміри компонента, потрібно вказати бажані властивості, які мають в назві “Request”, наприклад “HeightRequest”, “WidthRequest” тощо. Варто звернути увагу на те, що “Request” – це всього лише «запит», тобто компонент запитує у середовища виконання, чи можливо відобразити його з розмірами, не меншими ніж вказані. Дотримання цих параметрів, як уже було сказано вище, не гарантується.

Для більшості компонентів можна вказувати весь допустимий діапазон розмірів, використовуючи властивості “MinHeightRequest” – “MaxHeightRequest” і т.д.

2.4.6 Забезпечення навігації за допомогою клавіатури

Хоча на мобільних пристроях частіше всього взаємодія з інтерактивними компонентами додатку відбувається за допомогою дотику до сенсорного екрану в тому місці, де потрібно виконати певну дію, а на операційних системах, характерних для ПК, частіше за все використовується маніпулятор типу «мишка», але якщо мова заходить про найчастіше вживані засоби управління навігацією по сторінці для людей з обмеженими можливостями, навігація за допомогою клавіатури все ще одна з найбільш популярних [28, 48]. Вона налаштовується напрочуд легко, знайомим для розробників програмного забезпечення для ПК: вказуючи індекс табуляції для кожного елементу інтерфейсу, для якого планується взаємодія. Приклад такого індексу:

```
<Entry Placeholder=”Username” TabIndex=”1” />
```

Також наполегливо рекомендується надавати декілька варіантів виконання найбільш важливих дій на кожному екрані: через жести, екранні читачі та скорочення клавіш, та надавати зрозумілий зворотній зв'язок для важливих дій користувача (вібрація чи звук).

Ще один спосіб спростити взаємодію з додатком, як для користувачів з обмеженими можливостями, так і для всіх інших – це встановлювати фокус на

компонент екрану, взаємодія з яким є найбільш очікуваною в більшості сценаріїв. Так, для вікна налаштувань доцільно встановити фокус на кнопку підтвердження, а для вікна відомостей про програму – на кнопку «ОК».

Висновки до розділу 2

У другому розділі було проведено аналітичну роботу по збору альтернативних засобів розробки крос-платформенних мобільних додатків, зібрано та проаналізовано їх ключові переваги та недоліки. Варіанти для використання під час модернізації було оцінено за їх ключовими характеристиками, а також було проведено зважену оцінку з урахуванням важливості кожного критерію для поставленої задачі.

Після того, як було обрано найбільш підходящий альтернативний варіант, його переваги та недоліки було розглянуто в контексті дій, необхідних для міграції з існуючого становища. Було сформовано алгоритм дій по адаптації наявних ресурсів.

Заключна частина розділу присвячена збору засобів, що спрощують використання мобільних додатків особами з обмеженими можливостями. Розглянуто найбільш часто вживані підходи, пов'язані з екранними читачами та полегшенням взаємодії з навігацією, з прикладами того, як це може бути реалізовано засобами обраного інструменту.

РОЗДІЛ 3
МОДЕРНІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ ПОГОДНОГО ІНФОРМАЦІЙНОГО СЕРВІСУ METEOTREND

3.1 Вибір засобів розробки

Коротко узагальнимо рішення, прийняті в попередніх розділах. Проводиться модернізація існуючого мобільного додатку погодного інформаційного сервісу Meteotrend, наявна на даний момент версія якого написана мовою JavaScript з використанням крос-платформенного фреймворку ReactNative. В результаті аналізу наявних варіантів було обрано модернізацію мовою програмування C# з використанням фреймворку MAUI (раніше відомого як Xamarin).

Обсяг роботи включає в себе як відтворення існуючого функціоналу, так і деякі додаткові зміни, пов'язані з виправленням знайдених недоліків та побажаннями від замовника. Зокрема потрібно додати можливість динамічної зміни мови інтерфейсу в налаштуваннях (із застосуванням обраної мови до користувацького інтерфейсу), коректну обробку помилок (особливо пов'язаних з обміном даними з сервером), а також реалізувати підтримку основних функцій, що стосуються зручності використання додатку особами з обмеженими можливостями.

Як уже було згадано в розділі 2, якщо розробка в React Native, за великим рахунком, не обмежена чіткими рамками інструментів (за винятком, хіба що, тієї частини, яка відповідає за компіляцію, та й то за потреби її можна замінити альтернативними програмами), і можна використовувати той текстовий редактор, який подобається (головне, щоб була підтримка синтаксису JavaScript, і бажано діалекту React), то з MAUI такого широкого вибору не буде. Звісно, писати код все так же можна в тому текстовому редакторі, який більше подобається, але оскільки проекти Xamarin за своєю структурою більш складні і взаємопов’язані, то фактично альтернатив основному інтегрованому середовищу

розробки Visual Studio від Microsoft немає. На момент модернізації додатку актуальною версією була Visual Studio 2022 Community Edition (безкоштовна для індивідуального використання).

Хоча в Visual Studio, за умови встановлення інструментарію для мобільної та кросплатформенної розробки, вже є емулятор мобільного пристрою за замовчуванням, його використання має певні обмеження, і за наявності Android Studio краще скористатися її емуляторами. Перед початком відладки достатньо запустити емулятор пристрою, дочекатися його завантаження та обрати правильний мобільний пристрій у списку Local Android Devices в меню запуску відладки [29, 49].

Не зважаючи на те, що розробка наявної на даний момент версії додатку велася одним розробником, і модернізація також буде проводитися тими ж силами, було вирішено цього разу в процесі розробки використовувати систему контролю версій. Для цього було одразу декілька причин. По-перше, Visual Studio має вбудований клієнт Git, який дозволяє практично без додаткових зусиль створити репозиторій (чи клонувати вже існуючий) та виконувати всі базові операції системи контролю версій над проектом: отримати актуальний стан гілки коду, відправити внесені зміни, переключитися на іншу гілку коду, створити тег, подивитися історію тощо. По-друге, використання системи контролю версій приносить багато користі навіть при індивідуальній розробці. Можна буквально в декілька кліків перейти до потрібної гілки розробки, перемикатися між гілками, кожна з яких вирішує окрему задачу, обєднувати гілки коду (попередньо перевіривши, чи не виникає конфліктів при обєднанні), а також переглядати розбіжності між гілками. По-третє, досвід роботи з Git (і зокрема з GitHub та Bitbucket, один із яких буде використовуватися в цій роботі) вже багато років є або обов'язковою вимогою або дуже бажаним умінням у переважній більшості вакансій розробника, тому здобути додатковий досвід під час роботи над даним проектом теж було не зайвим.

Як і раніше, для коректної роботи наявної версії додатку потрібні Java Development Kit та NodeJS (для компіляції проекту в React Native, якщо

знадобиться додаткова відладка), а для взаємодії з NodeJS буде використовуватися вбудована Windows PowerShell. Список всього необхідного програмного забезпечення та причина його використання приведені в табл. 3.1.

Таблиця 3.1 – Перелік програмного забезпечення

Назва	Тип	Призначення
Microsoft Visual Studio 2022 Community Edition	Інтегроване середовище розробки	Робота з проектом модернізації, редагування коду, компіляція
Android Studio Ladybug | 2024.2.1	Інтегроване середовище розробки	Емуляція пристроїв Android за допомогою вбудованого менеджера віртуальних пристроїв
Bitbucket плагін для Visual Studio	Плагін – розширення функціоналу	Застосування системи контролю версій до файлів проекту
Java Development Kit (JDK) 23	Набір розробника Java	Використання середовища виконання Java, а також деяких інструментів з цього набору для коректної роботи Node.JS та Android Studio
Node.JS v22.11.0	Середовище виконання JavaScript	Необхідне для запуску наявної версії Meteotrend для порівняння коректності роботи

Остаточний вибір засобів розробки враховує як технічні аспекти, так і потреби ефективної роботи розробника. Використання Microsoft Visual Studio 2022 Community Edition забезпечує зручну інтеграцію з фреймворком MAUI та системою контролю версій Git, що суттєво полегшує управління кодом і тестування. Таким чином, обраний інструментарій не лише відповідає технічним вимогам, але й сприяє підвищенню продуктивності та якості розробки.

3.2 Створення та конфігурація проєкту

Хоча загалом процес створення проєктів різних видів в Visual Studio виглядає однаково, але від правильного вибору опцій буде залежати коректність створеного проєкту та його відповідність нашим очікуванням. Тому на процедурі створення проєкту та його початковій конфігурації варто зупинитися трохи детальніше.

Перш за все, варто відзначити, що Visual Studio – це середовище розробки для багатьох продуктів, так чи інакше пов'язаних з сервісами Microsoft: тут є і веб-розробка та хмарні сервіси (ASP.NET, Azure, NodeJS), різні види мобільної та десктопної розробки, розробка ігор на Unity та на ігрових рушіях на основі C++, а також чимало додаткових інструментів. Найчастіше весь набір існуючих інструментів одному користувачеві не потрібен, а в повному наборі Visual Studio 2022 займе на диску понад 100 ГБ [30, 52]. Тому при встановленні Visual Studio користувач може обрати ті компоненти, які йому потрібні (за потреби пізніше можна видалити чи встановити додаткові). Отже, для того, щоб можна було вести розробку мобільного додатку з використанням фреймворку MAUI, потрібно встановити компоненти Visual Studio, які для цього необхідні (рис. 3.1).

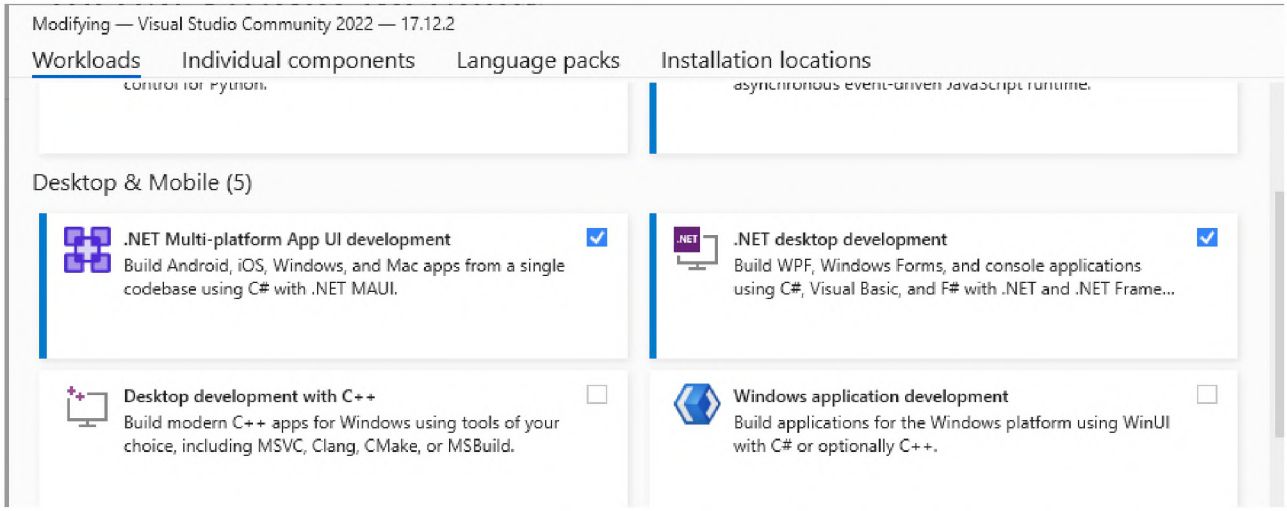

Рисунок 3.1 – Встановлення компонентів для мобільної розробки

Діалог створення нового проекту за замовчуванням відображає всі доступні види проектів, які можна створити з використанням встановлених компонентів Visual Studio, і щоб не помилитися, можна скористатися фільтрами над списком проектів. Перший фільтр – мови програмування, тут потрібно встановити C#, другий – платформи, тут можна обмежити вибір тими платформами, які планується використовувати. Останній фільтр найважливіший, тут можна встановити тип проекту, в нашому випадку – MAUI. Навіть після того, як ми виберемо та встановимо необхідні фільтри, все одно залишається чотири

види проекту. Потрібно обрати .NET MAUI App, оскільки саме цей тип проекту створює єдиний додаток для платформ, які заплановано як цільові для модернізації Meteotrend.

Далі вказуємо назву проекту та обираємо місце, де він буде розташований (рис. 3.2.). За потреби можна вказати окреме ім'я для рішення, але в нашому випадку це не потрібно. «Рішення» – це додаткова абстракція всередині проекту Visual Studio, яка дозволяє в рамках одного проекту і одного програмного коду отримувати різні варіанти виконуваних артефактів. Частіше всього у проекті знаходиться одне рішення (як у нашому випадку), яке називається так само, як і проект, і знаходиться в одній із ним папці. Проте у випадках розробки складних проектів, може бути декілька рішень, з кожним із яких можна працювати в Visual Studio окремо. Прикладом такого проекту може бути мобільна гра, яка має декілька версій (безкоштовну та платну), а також декілька доповнень до неї, які встановлюються окремо. Розробник доповнення може займатися лише ним, і при цьому не боятися змінити код класів, що не відносяться безпосередньо до його роботи, а тестувальник безкоштовної версії може запустити юніт-тести для відповідного рішення і бути впевненим, що перевіряє лише функціонал безкоштовної версії.

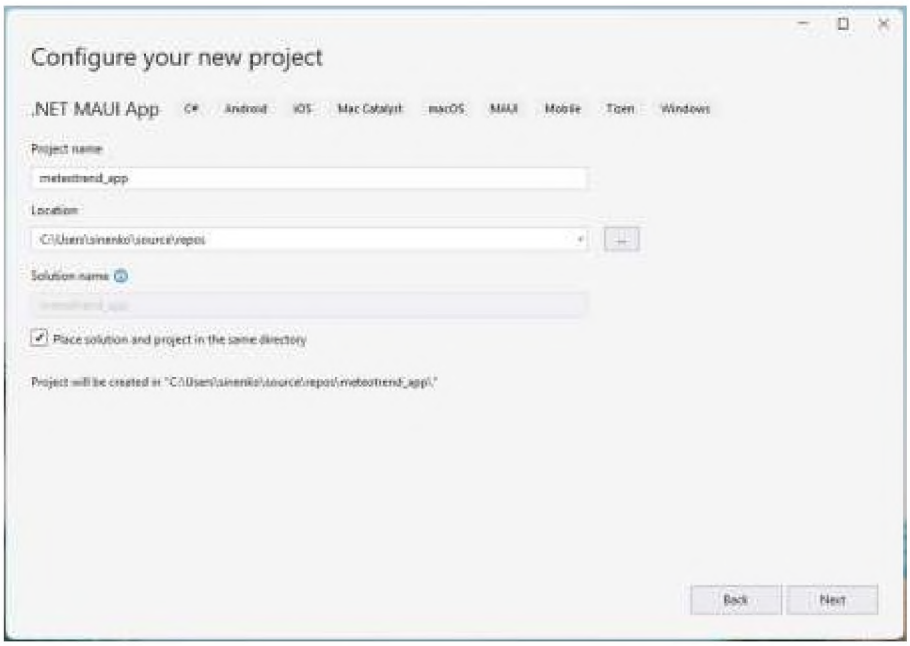

Рисунок 3.2 – Створення нового проекту

На наступному кроці потрібно обрати тип та версію фреймворку, на якому буде розроблюватися проект. Для десктопних додатків вибір значно більший, а для MAUI на даний момент є всього дві опції: .NET 8.0 (Long Term Support) та .NET 9.0 (Standard Term Support). Як уже зазначалося раніше, одна з переваг MAUI забезпечується в першу чергу тим, що цей фреймворк базується на .NET, навколо якого вибудовано цілу систему сумісності з багатьма іншими технологіями. В нашому випадку буде достатньо старішої версії з довготривалою підтримкою.

Після натискання на кнопку “Create” пройде трохи часу, доки буде створено базову структуру проекту та завантажено необхідні шаблони. Розглянемо призначення кожного елемента структури проекту (рис. 3.3), оскільки в подальшому з ним потрібно буде працювати.

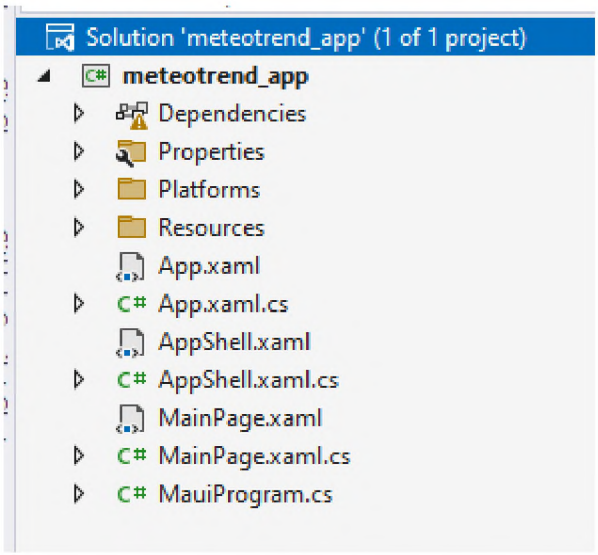

Рисунок 3.3 – Структура нового проекту

Dependencies. Тут зберігається список пакетів NuGet (перевірені сторонні бібліотеки, які можна підключити до проекту), а також залежності, на які спирається проект. В цьому списку можуть бути фреймворки MAUI, сторонні пакети, а також додаткові бібліотеки .NET. Напряму цей список як правило не редагується, але ті бібліотеки, які будуть додаватися в процесі розробки, будуть тут відображатися.

Properties. Тут зберігаються конфігураційні файли, а також метадані для проекту. По мірі того, як в проект будуть додаватися нові об'єкти, а також поступово будуть вноситися зміни, необхідні для публікації проекту, в цій папці також будуть додаватися нові файли.

Platforms. У списку відмінностей між React Native та MAUI в розділі 2 згадувалося про те, що останній дозволяє виділити весь платформо-специфічний код, і виокремити його в окремі папки, залишаючи в загальній кодовій базі лише спільний для всіх платформ код. Папка Platforms містить вкладені папки для всіх підтримуваних платформ, і по мірі того, як буде виникати потреба зробити якісь зміни, характерні для окремих платформ (наприклад, редагування маніфесту чи додавання специфічних ресурсів), саме в ці папки ми будемо звертатися.

Resources. Зберігає спільні для всіх платформ ресурси: стилі, зображення, шрифти тощо. Містить вкладені папки для різних видів ресурсів, зокрема окрема папка Raw для тих ресурсів, які не обробляються напряму, наприклад JSON, XML та інші допоміжні файли.

App.xaml. Це основна точка входу для програми (аналогічно до Program.cs чи Main.c), для визначення ресурсів, стилів, тем та глобальних шаблонів даних. Містить визначення класу Application та є оптимальним місцем для того, що задавати спільні стилі, кольори та інші ресурси. В React Native було досить проблематично задати стилі для спільного використання всіма компонентами, бо все потрібно було б передавати по ієрархії викликів разом з іншими параметрами. В MAUI можна задати все це в основній точці входу (можна й в іншому місці, але тоді буде складніше шукати ці місця в разі потреби внести зміни), і далі використовувати в усьому проекті просто називаючи стиль по його ідентифікатору.

App.xaml.cs. Є частиною описаного вище класу, дозволяє задавати події життєвого циклу додатку та логіку ініціалізації. Може бути використаний для того, щоб налаштувати налаштування та сервіси рівня всієї програми.

AppShell.xaml. Визначає структуру та зовнішній вигляд програмної навігації (у випадку, якщо використовується навігація на основі Shell, а не NavigationPages [20, 51].

AppShell.xaml.cs. Аналогічно, містить логіку для файлу, описаного в попередньому абзаці. Також в цьому файлі можна задавати прив'язку подій до елементів оболонки.

MainPage.xaml. Початковий користувацький інтерфейс для додатку. Включає в себе XAML розмітку для першої сторінки, яку побачить користувач (за умови, що до цього не буде якогось початкового встановлення налаштувань або іншого разового переходу).

MainPage.xaml.cs. Містить логіку та обробники подій для цієї сторінки.

MauiProgram.cs. Ініціалізує та налаштовує додаток .NET MAUI. Реєструє сервіси, обробники, ресурси та інші дані конфігурації для подальшого використання (зокрема в dependency injection).

3.3 Використання особливостей MAUI для забезпечення виконання додаткових вимог

Користувацький інтерфейс крос-платформенного застосунку .NET MAUI складається з абстрактних об'єктів, які надалі переводяться в native компоненти кожної з цільових платформ. Основні групи компонентів, з яких зазвичай складають користувацький інтерфейс – сторінки (pages), розміщення (layouts) та відображення (views).

Сторінки можуть бути чотирьох основних типів (рис. 3.4): звичайні (ContentPage), бічне меню (FlyoutPage), панель навігації (NavigationPage) та вкладки (TabbedPage) [31].

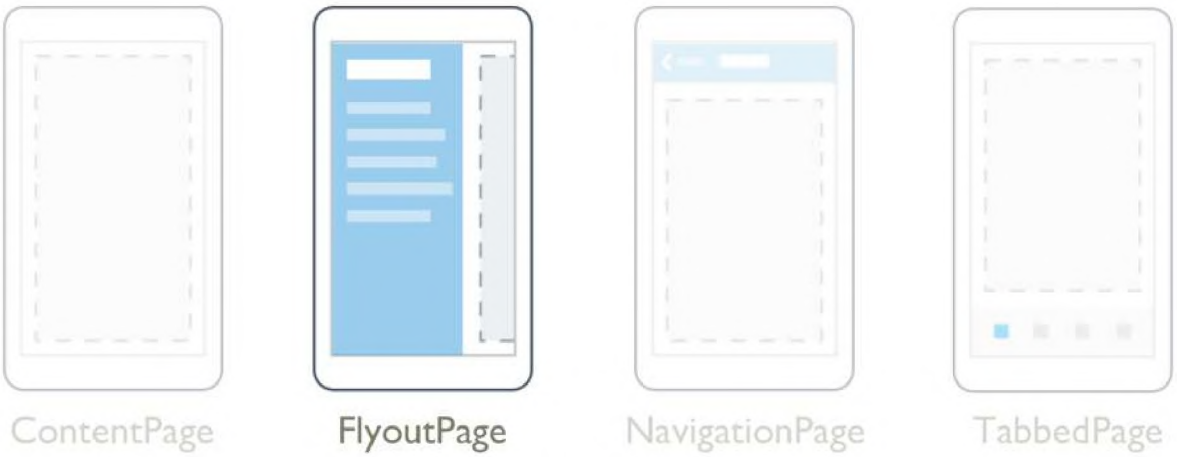

Рисунок 3.4. – Типи сторінок

Розміщення мають більше варіантів:

- абсолютне (absolutelayout) – коли всі елементи розміщуються по координатам відносно їх батьківського контейнера, чи не найбільш рідкісний варіант в силу своєї обмеженої адаптивності;

- зв’язуюче (bindablelayout) – дає можливість генерувати вміст, використовуючи динамічну прив'язку до колекції об'єктів, така прив’язка частіше всього працює в обидві сторони, тобто якщо змінюється порядок чи якісь елементи в колекції, вони одразу ж відображаються в контенті, а якщо користувач змінює певним чином порядок елементів через інтерфейс (наприклад, перетягуючи елементи і змінюючи їх порядок), оновлюється пов'язана колекція;

- гнучке (flexlayout) – дозволяє впорядковувати свої елементи використовуючи визначений набір варіантів вирівнювання та орієнтації, він базується на подібному модулі гнучкого розміщення з CSS, відомому як flex-box чи flex layout;

- сітка (grid) – відображає компоненти у вигляді таблиці з визначеним числом рядків та стовпців, причому один елемент може розширюватися на декілька рядків чи стовпців, що дозволяє будувати досить складні, але при цьому адаптовані інтерфейси, один з найбільш вживаних елементів, не в останню чергу тому, що в ньому можна задавати автоматичне встановлення ширини для

колонки, і вона буде займати весь вільний простір, що допомагає правильно розставити елементи на різних типах пристроїв;

- горизонтальний та вертикальний стеки (horizontalstacklayout, verticalstacklayout) – складають елементи один за другим в рядок чи стовпчик, хоча таке розміщення є чи не найбільш часто вживаним, але на практиці ці два варіанти використовуються порівняно рідко, бо існує stacklayout, який може бути як горизонтальним так і вертикальним, і зазвичай для спрощення використовують його, хоча в проектах з відкритим кодом вважається кращою практикою використовувати окремий клас залежно від типу розміщення, що покращує читаємість коду.

Всі ці варіанти розміщення досить корисні самі по собі, але якщо враховувати, що їх можна вільно комбінувати між собою (наприклад, зробити кореневе розміщення у вигляді сітки, у верхні комірки додати вертикальні стеки, а в нижні – горизонтальні стеки, причому в горизонтальних стеках зробити гнучке розміщення з вирівнюванням), отримуємо універсальний інструмент, який дозволяє відобразити інтерфейс будь-якої складності. Головне при цьому не забувати, що чим складніший інтерфейс, тим важче буде користувачеві правильно з ним взаємодіяти. Декількох промахів повз елементи керування може бути достатньо для того, щоб ваш додаток було видалено з пристрою, яким би корисним і функціональним він не був.

Перераховувати всі елементи категорії View тут не будемо, в силу того, що їх тільки в стандартному наборі MAUI майже 50 [31], а за рахунок розширення додатковими бібліотеками може бути набагато більше. Достатньо знати, що все, що наслідується від класу View, може бути відображено на екрані. Відповідно, при створенні власних елементів інтерфейсу, їх потрібно наслідувати саме від цього класу. Сюди відносяться як стандартні елементи керування (кнопка, перемикач, повзунок, текстове поле тощо), так і геометричні примітиви, і деякі додаткові компоненти, які будуть згадані пізніше.

Якщо згадувати розробку Meteotrend в React Native, то можна пригадати, що кожен файл з кодом (за винятком утилітарних, що містили виключно

JavaScript функції без жодної візуалізації) містив як візуальну складову, так і програмний код за нею, причому результат кожного такого файлу повертав JSX структуру – спеціальну версію XML [12], яка в подальшому включалася до батьківського компоненту. Але при цьому безпосередньо код кожного файлу змішував візуалізацію та виконання. MAUI пропонує іншу концепцію, де кожен клас, що підтримує візуалізацію (фактично, кожна сторінка) складається з двох частин: опису користувацького інтерфейсу та реалізації його функціональної частини. Друга частина пишеться на досить традиційному C# з використанням широких можливостей, які надає .NET. Перша частина – так званий XAML (eXtensible Application Markup Language), мова розмітки на основі XML. Це не обов'язкова вимога, в MAUI можна створювати сторінки й на чистому C#, але тоді втрачається основна перевага підходу – чітке розділення візуалізації та логіки [32].

XAML – фактично XML, і кожен валідний з точки зору XAML файл буде також валідним з точки зору XML. Але синтакс XAML надає чимало унікальних можливостей. Окрім уже згаданого вище розділення візуальної та функціональної складових, XAML надає більш зрозумілу і читаєму структуру інтерфейсу, відображаючи ієрархію контейнерів та компонентів.

Як і більшість мов розмітки, XAML має свої обмеження, зокрема:

- не може містити виконуваний код, всі обробники подій повинні бути визначені у відповідному файлі ,cs;
- не може містити цикли, це обмеження можна обійти використовуючи компоненти, що працюють з колекціями даних (наприклад listview та collectionview);
- не може містити умовну обробку, тим не менш, завдяки використанню прив'язки даних і використанню конвертерів можна досягнути аналогічного результату;
- загалом не може створювати класи, у яких немає конструктора без параметрів, втім за деяких умов це обмеження можна обійти;

- в цілому не може викликати методи, хоча це обмеження також можна обійти в деяких випадках (переважно за рахунок прив'язки даних).

Таким чином, кожна окрема сторінка може прив’язувати дані до окремих елементів на сторінці без необхідності передавати дані від компонента до компонента вниз по ієрархії, як це робиться в React Native.

3.4 Розробка набору екранів Onboarding

Оскільки об'єм коду всього додатку явно виходить за обмеження даної роботи, то продемонструємо особливості підходів та концепцій фреймворку MAUI на серії екранів Onboarding, які відображаються при першому запускові додатку. Запустимо наявну версію додатку (React Native) для порівняння візуальної реалізації (рис. 3.5).

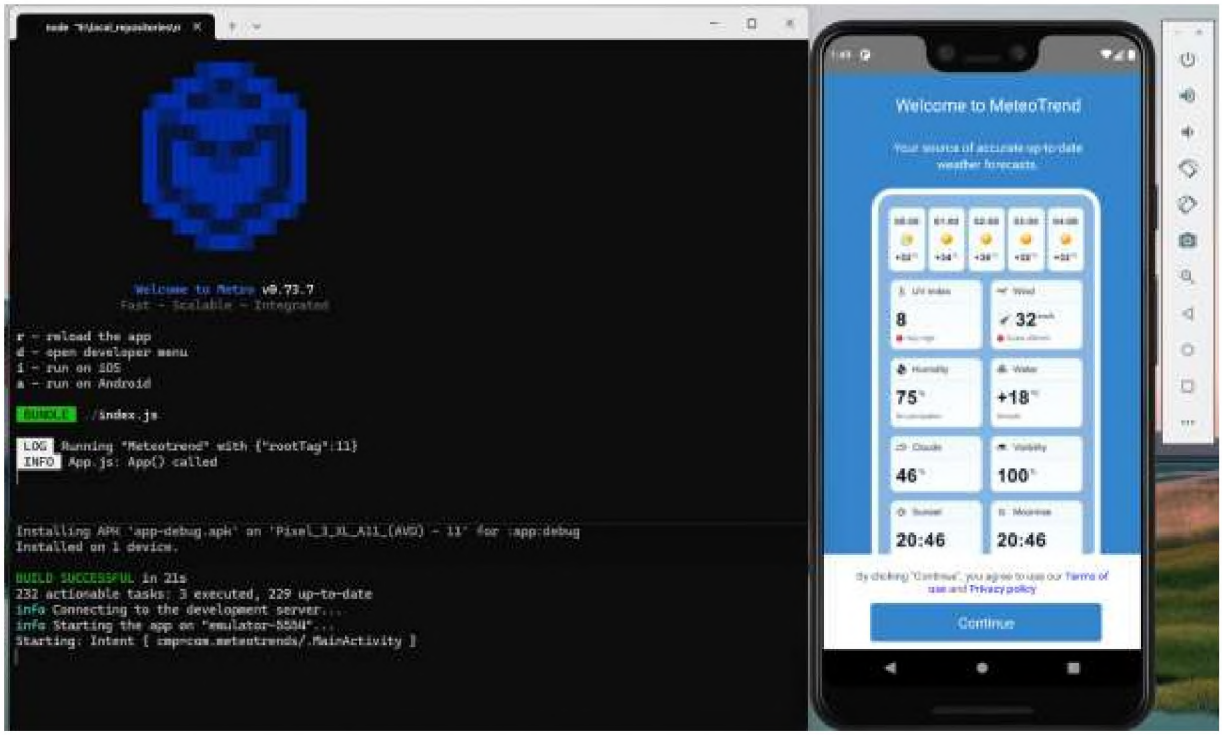

Рисунок 3.5 – Запуск Meteotrend версії React Native

Перед тим, як розбирати екран, зображений на рисунку вище, потрібно передбачити, щоб набір екранів Onboarding відображався користувачеві лише під час першого запуску (оскільки основне призначення цих екранів – задати початкові налаштування користувача, а також визначити його поточне розміщення). Для цього потрібно додати код в App.xaml.cs (початкова точка виконання логіки). Повна версія зміненого коду має наступний вигляд:

```
using meteotrend.Pages;

namespace meteotrend
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            bool hasCompletedOnBoarding =
Preferences.Get("HasCompletedOnBoarding", false);

            if (hasCompletedOnBoarding)
            {
                // User has completed onboarding, load the main page
                // MainPage
                MainPage = new NavigationPage(new MainPage());
            }
            else
            {
              // User has not completed onboarding, load the onboarding page
                // Onboarding1Page
                MainPage = new NavigationPage(new Onboarding1Page());
            }
        }
    }
}
```

Оскільки це перший лістинг коду проекту, розберемо його якомога детальніше. В подальших лістингах ми вже не будемо повертатися до пояснення фрагментів, які були описані раніше.

```
using meteotrend.Pages;
```

Оскільки сторінки (екрани) додатку знаходяться в папці Pages, а файл, який зараз редагується, знаходиться в корені проекту, то потрібно додати простір імен, який відповідає місцезнаходженню папки.

```
namespace meteotrend
```

Оскільки теперішня сторінка знаходиться в корені проекту, то її простір імен відповідає базовому простору імен проекту “meteotrend”.

```
public partial class App : Application
```

Клас App вказано як основний клас запуску, унаслідуваний від Application. Модифікатор partial означає, що даний файл з кодом не повністю описує даний клас, і він може бути додатково доповнений у інших файлах.

```
public App()
        {
            InitializeComponent();

            bool hasCompletedOnBoarding =
Preferences.Get("HasCompletedOnBoarding", false);

            if (hasCompletedOnBoarding)
            {
                // User has completed onboarding, load the main page
                // MainPage
                MainPage = new NavigationPage(new MainPage());
            }
            else
            {
            // User has not completed onboarding, load the onboarding page
                // Onboarding1Page
                MainPage = new NavigationPage(new Onboarding1Page());
            }
        }
```

Метод App викликається при створенні об'єкту цього класу (фактично, буде виконано під час запуску додатку). Він починається з виклику InitializeComponent(), що є хорошою практикою для конструктора похідного класу – завжди спочатку викликати конструктор або аналогічний метод базового класу [33].

Далі ми перевіряємо, чи є в збережених параметрах для цього додатку на даному пристрої запис із ключем “HasCompletedOnBoarding”. Для цього використовується клас Preferences, який відповідає за роботу з даними, які мають зберігатися між запусками додатку (але все одно видаляються, якщо додаток видаляється з пристрою). Preferences має два методи, важливі для нас: Get(key, valueIfFalse) – отримує значення ключа, або ж повертає вказане значення, якщо такого ключа не існує, та Set(key, value) – записує в пам'ять пристрою значення за вказаним ключем. Фактично, з точки зору програмування клас Preferences

можна сприймати як словник із асинхронним доступом. На різних платформах native реалізація цього класу буде відрізнятися, але, як і стверджує вся концепція крос-платформенних фреймворків, нас це не повинно хвилювати, оскільки ми працюємо з абстракцією. В даному випадку ми перевіряємо, чи є в збережених ключах для нашого додатку ключ з ідентифікатором «HasCompletedOnBoarding», і якщо такого ключа не знайдено (або якщо такий ключ є, але його значення “false”), то ми переходимо до першого екрану з серії Onboarding, якщо ж такий ключ існує і в нього записано значення “true”, то можна переходити до головного екрану.

Перш ніж почати створювати сторінки, варто потурбуватися про те, щоб усі сторінки мали однакову кольорову схему. Для цього варто визначити іменовані кольори для окремих об'єктів, і або ж створити стилі для кожного компоненту, або ж принаймні встановити кольори для більшості елементів інтерфейсу, щоб за потреби їх можна було швидко змінити. Для цього підходить Resources/Styles/Colors.xaml. За замовчуванням в ньому вже визначено набір стандартних кольорів:

```
<!-- Default colors -->
<Color x:Key="Primary">#512BD4</Color>
<Color x:Key="PrimaryDark">#ac99ea</Color>
<Color x:Key="PrimaryDarkText">#242424</Color>
<Color x:Key="Secondary">#DFD8F7</Color>
<Color x:Key="SecondaryDarkText">#9880e5</Color>
<Color x:Key="Tertiary">#2B0B98</Color>

<Color x:Key="White">White</Color>
<Color x:Key="Black">Black</Color>
<Color x:Key="Magenta">#D600AA</Color>
<Color x:Key="MidnightBlue">#190649</Color>
<Color x:Key="OffBlack">#1f1f1f</Color>

<Color x:Key="Gray100">#E1E1E1</Color>
<Color x:Key="Gray200">#C8C8C8</Color>
<Color x:Key="Gray300">#ACACAC</Color>
<Color x:Key="Gray400">#919191</Color>
<Color x:Key="Gray500">#6E6E6E</Color>
<Color x:Key="Gray600">#404040</Color>
<Color x:Key="Gray900">#212121</Color>
<Color x:Key="Gray950">#141414</Color>

<SolidColorBrush x:Key="PrimaryBrush" Color="{StaticResource Primary}"/>
<SolidColorBrush x:Key="SecondaryBrush" Color="{StaticResource Secondary}"/>
```

```
<SolidColorBrush x:Key="TertiaryBrush" Color="{StaticResource Tertiary}"/>
<SolidColorBrush x:Key="WhiteBrush" Color="{StaticResource White}"/>
<SolidColorBrush x:Key="BlackBrush" Color="{StaticResource Black}"/>

<SolidColorBrush x:Key="Gray100Brush" Color="{StaticResource Gray100}"/>
<SolidColorBrush x:Key="Gray200Brush" Color="{StaticResource Gray200}"/>
<SolidColorBrush x:Key="Gray300Brush" Color="{StaticResource Gray300}"/>
<SolidColorBrush x:Key="Gray400Brush" Color="{StaticResource Gray400}"/>
<SolidColorBrush x:Key="Gray500Brush" Color="{StaticResource Gray500}"/>
<SolidColorBrush x:Key="Gray600Brush" Color="{StaticResource Gray600}"/>
<SolidColorBrush x:Key="Gray900Brush" Color="{StaticResource Gray900}"/>
<SolidColorBrush x:Key="Gray950Brush" Color="{StaticResource Gray950}"/>
```

Додамо декілька кольорів, які будуть використовуватися в додатку:

```
<!-- App specific colors and brushes-->
<Color x:Key="MT_MainBlue">#3485CB</Color>
<Color x:Key="MT_HeaderBlue">#1063AB</Color>
<Color x:Key="MT_OnboardingButtonBackgroundSelected">#FFFFFF</Color>
<Color x:Key="MT_OnboardingButtonBackgroundNotSelected">#3485CB</Color>
<Color x:Key="MT_OnboardingButtonTextSelected">#1063AB</Color>
<Color x:Key="MT_OnboardingButtonTextNotSelected">#FFFFFF</Color>
```

Тепер можна створити клас сторінки Onboarding1. Розберемо її код (вже модифікований нами) по частинам.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="meteotrend.Pages.Onboarding1Page"
             NavigationPage.HasBackButton="False"
             NavigationPage.HasNavigationBar="False"
             BackgroundColor="{StaticResource MT_MainBlue}">
```

Перший рядок – невід'ємна частина валідного XML файлу (як уже згадувалося раніше, XAML – також підвид XML.

ContentPage – це один із видів сторінки, як також згадувалося раніше. Це кореневий елемент ієрархії сторінки, в який будуть додаватися всі подальші елементи. Властивість xmlns визначає правила, за якими ці схеми мають бути валідовані. Властивість x:Class має співпадати з іменем класу (і таким же у .cs доповненні до xaml). Ця частина файлу генерується автоматично.

Як можна бачити з рисунку вище, перша сторінка Onboarding не має ані панелі навігації вгорі, а ні кнопки «назад», бо в принципі ще нема куди

повертатися. Тому ці дві властивості встановлюються в “false”. Остання властивість встановлює колір фону для всієї сторінки в заданий нами.

```
<Grid RowDefinitions="Auto, *, Auto">
    <!-- Top Section: Welcome Text -->
    <StackLayout Grid.Row="0" Padding="20, 30, 20, 10" VerticalOptions="Start">
        <Label
            Text="Welcome to MeteoTrend"
            FontSize="24"
            TextColor="White"
            HorizontalOptions="Center"
            FontAttributes="Bold"/>
        <Label
            Text="Your source of accurate up-to-date weather forecasts."
            FontSize="16"
            TextColor="White"
            HorizontalOptions="Center"
            HorizontalTextAlignment="Center"
            Padding="40,0,40,0"
            Margin="0,10,0,0"/>
    </StackLayout>

    <!-- Middle Section: Image -->
    <Image Grid.Row="1"
           Source="onboarding_welcome.png"
           Aspect="AspectFit"
           VerticalOptions="Center"
           HorizontalOptions="Center"/>
```

Наступна частина коду XAML задає верхній текст сторінки а також зображення по центру сторінки. В кореневу ієрархію сторінки додається розміщення Grid, яке складається з трьох рядів, де перший та останній займають розмір автоматично, з огляду на те, скільки місця займає їх вміст, а рядок посередині займає все місце, яке залишилося.

Всередині першого рядка додається контейнер стек, щоб розмістити в цьому рядку декілька компонентів. Також вказується відступ (padding) для того, щоб вміст контейнеру не займав усе відведене місце. В середину цього контейнеру додаються два компоненти Label, задача яких – відображати статичний текст. Їх властивості досить очевидні, тому зауважимо тільки, що навіть для двох компонентів Label довелося вводити додатковий контейнер у вигляді стеку, бо інакше вони були б розміщені за однаковими координатами, іншими словами, наклалися б один на одного. Загальне правило розміщення Grid: в кожній комірці має бути тільки один компонент. Якщо в комірці потрібно

встановити більше ніж один компонент, їх потрібно обернути в додатковий контейнер. Позиція елементу в комірці визначається його властивостями Grid.Row та за потреби – Grid.Column, починаючи рахунок з нульового індексу.

Так, зображення в центрі – єдиний елемент в цій комірці, а отже, його не потрібно додатково загортати у контейнер. Тут вказується адреса цього зображення (в даному випадку зображення знаходиться в проекті за шляхом за замовчуванням для всіх зображень – /Resources/Images/, тому достатньо вказати його назву – onboarding_welcome.png. Властивість Aspect визначає те, як середовище виконання має розміщувати це зображення, в даному випадку значення AspectFit означає, що зображення повинно адаптуватися до вільного місця в контейнері так, щоб воно повністю туди помістилося. Дві останні опції визначають, як вирівнювати зображення в межах свого контейнера.

```
<!-- Bottom Section: Terms, Privacy Policy and Continue Button -->
<StackLayout Grid.Row="2"
             Padding="20"
             BackgroundColor="White"
             VerticalOptions="End">

    <!-- Terms and Privacy Policy Text -->
    <Label HorizontalOptions="Center"
           Padding="10, 5, 10, 5"
           Margin="0, 0, 0, 10"
           HorizontalTextAlignment="Center">
        <Label.FormattedText>
            <FormattedString>
                <Span Text="By clicking " />
                <Span Text="&quot;Continue&quot;"/>
                <Span Text=", you agree to use our "/>
                <Span Text="Terms of use" TextColor="Blue"
              SemanticProperties.Hint="Opens the Terms of Use in a new page">
                    <Span.GestureRecognizers>
                        <TapGestureRecognizer Tapped="OnTermsOfUseTapped" />
                    </Span.GestureRecognizers>
                </Span>
                <Span Text=" and " />
                <Span Text="Privacy policy" TextColor="Blue"
            SemanticProperties.Hint="Opens the Privacy Policy in a new page">
                    <Span.GestureRecognizers>
                      <TapGestureRecognizer Tapped="OnPrivacyPolicyTapped" />
                    </Span.GestureRecognizers>
                </Span>
                <Span Text="."/>
            </FormattedString>
        </Label.FormattedText>
    </Label>
```

Ця секція багато в чому подібна до попередньої, тому не будемо повторно пояснювати особливості вже розглянутих елементів. З того, на що варто звернути увагу в цьому фрагменті – компонент Label, який містить у собі форматований рядок. За замовчуванням налаштування шрифту застосовуються до всього вмісту компоненту. В нашому випадку потрібно в рамках одного тексту вставити два посилання, за якими мають відображатися інші екран, а тому потрібно частину тексту відформатувати по іншому, до того ж додати обробник події, пов'язаної саме з цим фрагментом тексту. Подія Click, звична для десктопного програмування, в мобільних пристроях залишилася лише для компонентів типу Button, у всіх інших випадках частіше за все подія більше пов'язана з сенсорним екраном, в нашому випадку – TapGestureRecognizer, що можна перекласти як розпізнавач жесту дотику.

```
            <!-- Continue Button -->
            <Button Text="Continue"
                    BackgroundColor="{StaticResource MT_MainBlue}"
                    TextColor="White"
                    FontSize="20"
                    HeightRequest="50"
                    CornerRadius="5"
                    VerticalOptions="End"
                    Padding="10"
                    Clicked="OnContinueClicked"
             SemanticProperties.Hint="Proceed to the next onboarding page"/>
        </StackLayout>
    </Grid>
</ContentPage>
```

Заключна частина першого екрану – кнопка, яка переводить користувача на наступний екран. Як можемо бачити, більшість властивостей відносяться безпосередньо до візуалізації компонента, але є ще одна, згадана раніше в контексті підтримки людей з обмеженими можливостями. Мова йде про SemanticProperties.Hint, яка надає додаткову інформацію про те, що буде після кліка на кнопку, в разі якщо цей екран буде розпізнано екранним читачем.

Тепер, коли ми розглянули візуальну частину сторінки, розберемо файл з логікою, яка опрацьовує взаємодію з компонентами користувацького інтерфейсу. За цей функціонал відповідає файл Onboarding1Page.xaml.cs.

```
namespace meteotrend.Pages
```

```
{
    public partial class Onboarding1Page : ContentPage
    {

        public Onboarding1Page()
        {
            InitializeComponent();
        }

        private void OnContinueClicked(object sender, EventArgs e)
        {
            _ = NavigateToNextOnboarding();
        }

        private void OnTermsOfUseTapped(object sender, EventArgs e)
        {
            _ = NavigateToTermsAsync();
        }

        private void OnPrivacyPolicyTapped(object sender, EventArgs e)
        {
            _ = NavigateToPrivacyAsync();
        }
```

Заголовок файлу подібний до розглянутого раніше. Серед методів маємо конструктор, задача якого обмежена викликом ініціалізації батьківського класу, а також три методи, які обробляють взаємодію з інтерактивними компонентами цієї сторінки. Кожен із них виконує єдину задачу – викликає метод, який і проводить операцію переходу на потрібну сторінку.

```
        private async Task NavigateToNextOnboarding()
        {
            // perform the async navigation
            try
            {
                await Navigation.PushAsync(new Onboarding2Page());
            }
            catch (Exception)
            {
                await DisplayAlert("Error", "Unable to navigate to the next
page", "OK");
            }
        }

        private async Task NavigateToTermsAsync()
        {
            // perform the async navigation
            try
            {
                await Navigation.PushAsync(new TermsOfUsePage());
            }
            catch (Exception)
            {
```

```
                    await DisplayAlert("Error", "Unable to navigate to the Terms
of use page", "OK");
                }
            }

            private async Task NavigateToPrivacyAsync()
            {
                // perform the async navigation
                try
                {
                    await Navigation.PushAsync(new PrivacyPolicyPage());
                }
                catch (Exception)
                {
                    await DisplayAlert("Error", "Unable to navigate to the
Privacy policy page", "OK");
                }
            }

        }

    }
```

Всі три методи дуже схожі між собою. Вони намагаються викликати асинхронну операцію з переходу до наступної сторінки (для чого створюється новий екземпляр відповідного класу), а в разі виникнення виключної ситуації відображається повідомлення про помилку. Єдине додаткове пояснення, яке можна надати до двох останніх лістингів коду, це конструкція виклику функції NavigateTo. Також може виникнути питання, чи потрібно з трьох функцій обробки події на сторінці викликати додаткові методи, чому не можна відразу включити в обробник події конструкцію try/catch. Причина в асинхронності методів, які нам потрібні. За правилами, спільними як для C# так і JavaScript, якщо в якомусь методі використовується асинхронна конструкція await, то такий метод потрібно позначати спеціальним ключовим словом async. Але додавання такого ключового слова змінює сигнатуру функції, після чого вона не може вважатися коректним обробником події, яку вона повинна спіймати. З іншого боку, ми можемо викликати з обробника подій інші асинхронні функції, не чекаючи результату їх роботи. Іншими словами, ми просто викликаємо асинхронну функцію, і не переймаємося тим, чим вона закінчиться. Але якщо просто написати в коді функції виклик асинхронної функції без await і змінної, куди буде присвоєно результат, отримаємо помилку при компіляції. Тому для

таких ситуацій використовується спеціальне позначення: _= functionName. Таким чином ми явно вказуємо компілятору, що хочемо викликати асинхронну функцію і нас не турбує, що її результат буде втрачено [34].

Після того, як створено візуальну та функціональну частини сторінки, можна запустити другий емулятор, і спробувати завантажити додаток для перевірки (рис. 3.6).

Якщо додаток на основі фреймворку MAUI потрібно запустити на ОС Windows, можна обрати пункт Framework (net8.0-android). Для пристроїв на iOS потрібно обирати один із трьох доступних варіантів, залежно від того, ведеться відладка на фізичному пристрої, підключеному до ПК, на віддаленому пристрої, чи на симуляторі. Для ОС Android варіантів менше, фактично лише один: Android Local Devices. Сюди потрапляють як фізичні пристрої, під'єднані до ПК, так і емулятори.

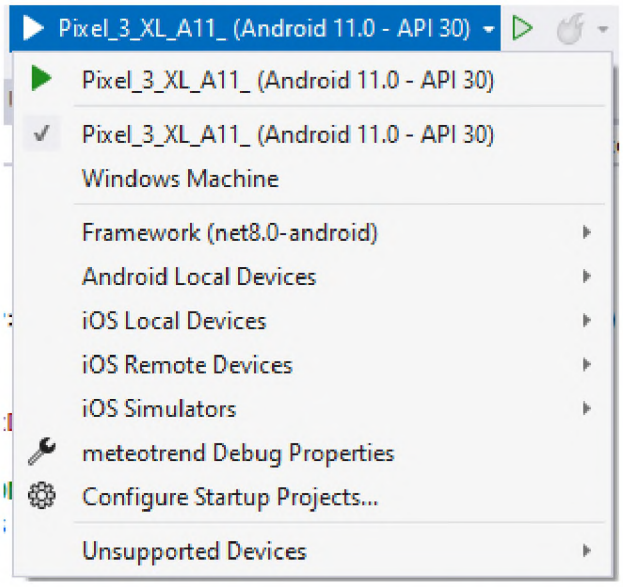

Рисунок 3.6 – Вибір пристрою для відладки

Після того, як обрано правильний пристрій, можна запускати компіляцію. Якщо в процесі не буде помилок, то в емуляторі буде відображено перший екран Onboarding (рис. 3.7):

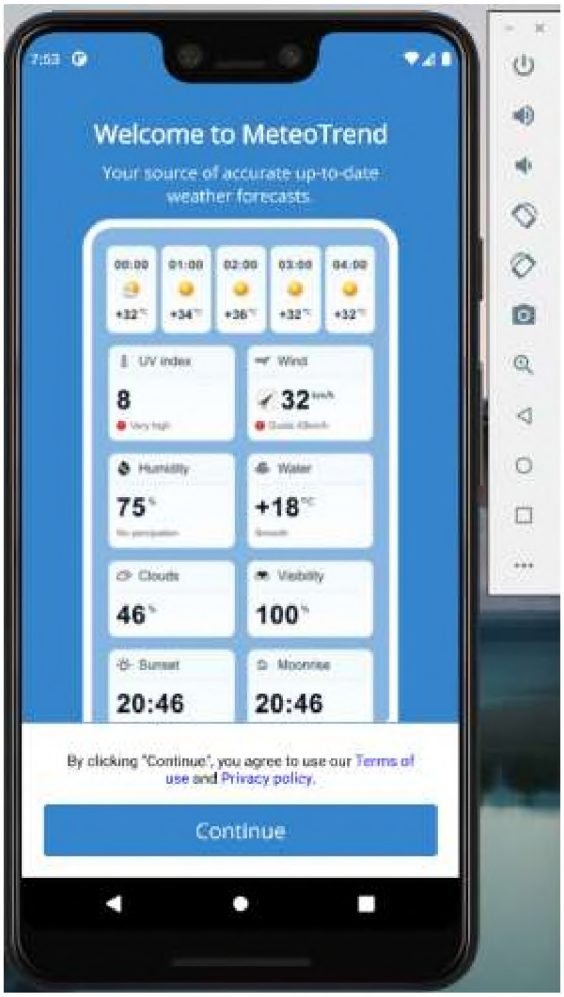

Рисунок 3.7 – Оновлений екран Onboarding 1

Продовжимо з наступними екранами. Простіше всього буде підготувати екрани для угоди користувача та політики приватності. Вони мають подібну і дуже просту структуру: шапка сторінки з заголовком та кнопкою-зображенням стрілки вліво, а також вміст сторінки – просте відображення великого обсягу тексту з прокруткою. Вміст сторінки пізніше буде встановлюватися динамічно, тому при створенні в коді можна залишити якийсь стандартний текст, все одно потім він буде замінюватися.

Хоча задача, на перший погляд, порівняно проста, але в процесі реалізації виникли певні складнощі. В першу чергу це пов’язано з тим, що за замовчуванням шапка навігації має інший вигляд кнопки повернення до попередньої сторінки. При використанні навігації через Shell можна налаштувати зовнішній вид і поведінку, але, як виявилося в процесі розробки

подальших сторінок, навігація через Shell в нашому випадку не дуже підходить, тому що стек сторінок у ній повністю контролюється системою, і деякі види переходів між сторінками приводять до втрати маршруту навігації (що, у свою чергу, приводить до того, що команди переходу вперед та назад не працюють до перезапуску додатку). Альтернативний спосіб навігації через Navigation Page дозволяє напряму маніпулювати зі стеком сторінок, більше того, побудова маршруту навігації повністю визначається логікою додатку. Але можливості зміни зовнішнього вигляду та поведінки меню навігації в цьому випадку набагато менш гнучкі [20]. Зрештою, знаючи, що в процесі розробки нам доведеться реалізувати декілька різних видів шапки навігації з різною поведінкою, було вирішено робити навігацію через Navigation Page, але повністю відмовитися від її стандартного компоненту і замінити його на самописну версію. Тому в лістингові нижче у властивостях сторінки параметри HasBackButton і HasNavigationBar встановлені в false, хоча звісно ж на цих сторінках і панель навігації, і клавіша повернення назад мають бути присутніми.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="meteotrend.Pages.PrivacyPolicyPage"
             Title="Privacy policy"
             NavigationPage.HasBackButton="False"
             NavigationPage.HasNavigationBar="False"
             BackgroundColor="White">
```

Наступний крок – це можливість гарно відображати заголовок: кнопка з зображенням завжди має знаходитися по краю, а текст заголовка має знаходитися посередині тієї ширини рядка, що залишилася. Для цього в контейнері заголовка (до речі, зверніть увагу: макет сторінки досить примітивний, і основним розміщенням сторінки є звичайний вертикальний стек) оголошуються дві колонки, одна з автоматичним вибором ширини, а друга – з вибором усієї ширини, що залишилася.

```
    <StackLayout>
        <!-- Header Content -->
        <Grid HeightRequest="50"
```

```
            BackgroundColor="{StaticResource MT_HeaderBlue}">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <!-- For the image -->
                    <ColumnDefinition Width="*" />
                    <!-- For the label -->
                </Grid.ColumnDefinitions>

                <!-- Back Button Image -->
                <Image Grid.Column="0"
                    Source="menu_back.png"
                    HeightRequest="24"
                    Margin="10, 0, 0, 0"
                    VerticalOptions="Center"
                    HorizontalOptions="Start">
                    <Image.GestureRecognizers>
                        <TapGestureRecognizer
Tapped="TapGestureRecognizer_Tapped" />
                    </Image.GestureRecognizers>
                </Image>

                <!-- Centered Label -->
                <Label Grid.Column="1"
                    Text="Privacy policy"
                    FontSize="20"
                    FontAttributes="Bold"
                    TextColor="White"
                    HorizontalOptions="Center"
                    VerticalOptions="Center"/>
            </Grid>
            <!-- Main Content -->
            <StackLayout Padding="20" Spacing="20">
                <ScrollView>
                    <Label Text="Privacy policy. Long text"
                   FontSize="14"
                   TextColor="Black"/>
                </ScrollView>

            </StackLayout>
        </StackLayout>

    </ContentPage>
```

Знаючи ширину тексту заголовку, можна було б встановити його положення таким чином, щоб він знаходився посередині. Але при зміні локалізації ширина тексту для однієї і тієї ж сторінки може змінитися як в більшу, так і в меншу сторони. Підхід, обраний в лістингові вище, забезпечує коректне відображення заголовка в максимально можливому діапазоні пристроїв. Практично така ж структура сторінки застосовується і для умов використання.

Код, який міститься в файлі .xaml.cs, досить лаконічний, і має бути зрозумілий після розбору попереднього класу.

```
namespace meteotrend.Pages;

public partial class PrivacyPolicyPage : ContentPage
{
    public PrivacyPolicyPage()
    {
        InitializeComponent();
    }

    private async void GoBack()
    {
        await Navigation.PopAsync();
    }

    private void TapGestureRecognizer_Tapped(object sender, TappedEventArgs e)
    {
        GoBack();
    }
}
```

Остаточний вигляд сторінки показано на рис.3.8:

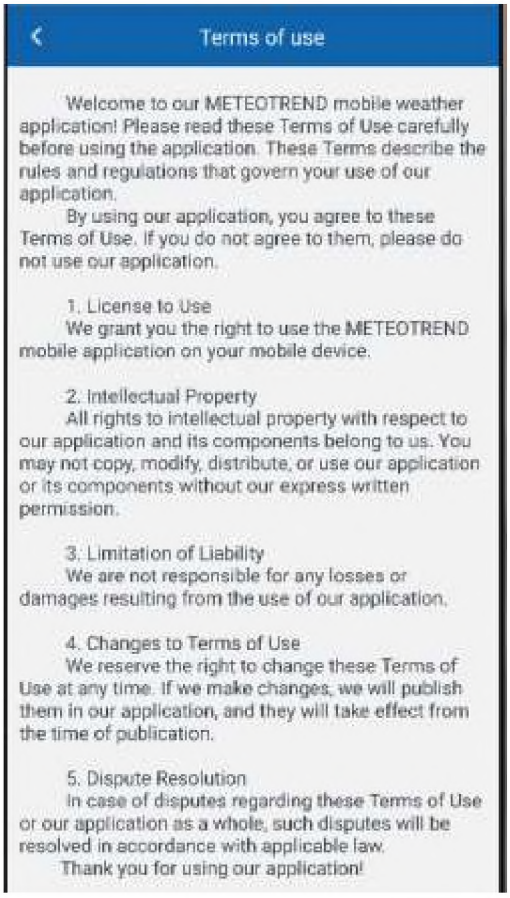

Рисунок 3.8 – Екран умов використання

Наступний екран має складнішу структуру. Користувач переходить до нього після того, як ознайомлюється з політиками приватності та умовами використання та погоджується продовжувати використовувати додаток. На цій сторінці користувач має обрати одиниці вимірювання для основних погодних характеристик: швидкості вітру, тиску та температури. Кожен із варіантів погодної характеристики має бути окремою кнопкою, а поведінка групи кнопок, що відносяться до однієї й тієї ж самої характеристики, має бути подібною до радіо кнопок (група елементів з текстом, де вибір одного з елементів знімає вибір іншого). Оскільки візуальна складова, з одного боку, має багато повторів, а з іншого – досить об'ємна, не будемо приводити весь програмний код (з ним можна ознайомитися в додатках), а вкажемо найбільш суттєві та значущі з точки зору новизни застосування в рамках цього додатку фрагменти.

На білому фоні потрібно розмістити кнопку з зображенням стрілки назад. Але стрілка також на білому фоні, тому користувачеві може бути складно зрозуміти, де знаходяться межі кнопки. Щоб позначити межі, кнопка з зображенням вкладається в інший візуальний компонент – Frame.

```
<!-- Back Button -->
<Frame Padding="0"
       CornerRadius="5"
       BorderColor="LightBlue"
       BackgroundColor="Transparent"
       HasShadow="False">

<Button Grid.Column="0"
    Text=""
    BackgroundColor="White"
    ImageSource="onboarding_back.png"
    Clicked="OnBackClicked" />

</Frame>
```

За рахунок того, що властивість Padding елемента Frame встановлена в «0», він займає стільки ж місця, скільки й сама кнопка. А за рахунок властивості BorderColor, крайні пікселі цього елемента будуть помітними на білому фоні.

Тепер розглянемо основну логіку роботи компонентів цього екрану – кнопки, що перемикають одиниці вимірювання.

```
            <!-- Temperature Selection -->
            <StackLayout Orientation="Horizontal" HorizontalOptions="Center"
Spacing="10" Margin="0, 15, 0, 0">
                <Image Source="onboarding_temperature.png" HeightRequest="24" />
                <Label Text="Temperature" FontSize="14" TextColor="White"/>
            </StackLayout>
            <StackLayout Orientation="Horizontal" HorizontalOptions="Center"
Spacing="20">
                <Button Text="C°"
                        x:Name="cmdTemperatureC"
                        BackgroundColor="{StaticResource
MT_OnboardingButtonBackgroundSelected}"
                        TextColor="{StaticResource MT_OnboardingButtonTextSelected}"
                        FontSize="18"
                        MinimumHeightRequest="50"
                        MinimumWidthRequest="75"
                        MaximumWidthRequest="150"
                        SemanticProperties.Description="Select Celsius as the unit of
temperature for weather forecasts."
                        Clicked="cmdTemperatureC_Clicked"/>
                <Button Text="Fa"
                        x:Name="cmdTemperatureF"
                        BackgroundColor="{StaticResource
MT_OnboardingButtonBackgroundSelected}"
                        TextColor="{StaticResource MT_OnboardingButtonTextSelected}"
                        FontSize="18"
                        MinimumHeightRequest="50"
                        MinimumWidthRequest="75"
                        MaximumWidthRequest="150"
                        SemanticProperties.Description="Select Fahrenheit as the unit
of temperature for weatherer forecasts."
                        Clicked="cmdTemperatureF_Clicked"/>
            </StackLayout>
```

З того, на що варто звернути увагу в цьому лістингу – кнопки використовують в якості кольору тексту та заднього фону не колір у форматі RGB, а StaticResource. Це якраз ті ідентифіковані кольори, які ми додавали вручну в App файл. Якщо в подальшому буде вирішено змінити колір усіх таких компонентів, то це достатньо буде змінити в одному місці, а не редагувати кожен файл. Також кнопки на цьому екрані використовують властивість SemanticProperties.Description, щоб спростити використання додатку для людей з обмеженими можливостями. Ще одна особливість, відображена на цьому лістингу, пов'язана з забезпеченням адаптивності інтерфейсу. Для обох кнопок встановлюються мінімальна висота кнопки, а також мінімальна і максимальна ширина. Як уже згадувалося, ці значення не гарантовані, але поки це можливо, система буде намагатися витримувати вказані параметри.

Перш ніж переходити до реалізації коду, який буде відповідати за перемикання параметрів, підготуємо допоміжні класи, які будуть використовуватися реалізацією. Для того, щоб було простіше передавати параметри налаштувань між класами, а також використовувати їх в API запитах, створимо для кожного параметру універсальну структуру SettingsItem.

```
namespace meteotrend.Sources.Types
{
    public class SettingsItem
    {
        public string Name { get; set; }
        public string TextValue { get; set; }
        public string APIValue { get; set; }

        public SettingsItem(string pName, string pTextValue, string pAPIValue)
        {
            Name = pName;
            TextValue = pTextValue;
            APIValue = pAPIValue;
        }

    }
}
```

Поле Name буде містити назву окремого екземпляру – окремого параметру конкретних налаштувань. Поле TextValue – текстову назву цього параметру, яка буде відображатися за потреби в користувацькому інтерфейсі. Поле APIValue міститиме умовне позначення для цього параметру, визначене схемою відповідного API запиту.

Далі створимо статичний клас Default, єдине призначення якого полягатиме в зберіганні статичних даних, загальнодоступних у межах додатку. Тут будуть зберігатися різні константи та інші параметри, які не варто зберігати десь окремо. Для прикладу, додамо сюди набір параметрів для визначення температури, а також функцію, яка дозволить отримати правильний об'єкт за його назвою.

```
    // Setting constants
    public static SettingsItem C_TEMPERATURE_CELSIUS = new SettingsItem("C°",
"C°", "c");

    public static SettingsItem C_TEMPERATURE_FAHRENHEIT = new SettingsItem("F°",
"Fa", "f");
```

```
public static SettingsItem GetTemperatureByName(string name)
{
    return name switch
    {
        "C°" => C_TEMPERATURE_CELSIUS,
        "F°" => C_TEMPERATURE_FAHRENHEIT,
        _ => C_TEMPERATURE_CELSIUS
    };
}
```

Повторимо аналогічні дії для одиниць вимірювання атмосферного тиску, а також для швидкості вітру. Тепер, коли у нас є типи даних для реалізації функціоналу, можна додавати весь код, необхідний для цього.

```
public partial class Onboarding2Page : ContentPage
{

    private SettingsItem CurrentTemperature;
    private SettingsItem CurrentPressure;
    private SettingsItem CurrentWindSpeed;

    public Onboarding2Page()
    {
        InitializeComponent();

        /* the following 3 rows were added to avoid compilation warning about
unitialized values of the corresponding variables.
           of course they will be initialized either by default values (in
the nearest "if" section) or by setting corresponding values
           depending on the preferences, but since compiler thinks there
might be any chance of null, better to be sure */
        CurrentTemperature = Defaults.C_TEMPERATURE_CELSIUS;
        CurrentPressure = Defaults.C_PRESSURE_MM;
        CurrentWindSpeed = Defaults.C_WINDSPEED_MS;

        // Checking preferences, initializing the buttons states

        var StoredTemperatureSettings =
Preferences.Get("Settings_TemperatureUnits", null);
        var StoredPressureSettings =
Preferences.Get("Settings_PressureUnits", null);
        var StoredWindSpeedSettings =
Preferences.Get("Settings_WindSpeedUnits", null);
        if (StoredTemperatureSettings == null)
        {
            StoredTemperatureSettings = "C°";
            CurrentTemperature = Defaults.C_TEMPERATURE_CELSIUS;
            Preferences.Set("Settings_TemperatureUnits",
Defaults.GetTemperatureByName(StoredTemperatureSettings).Name);
        }
        if (StoredPressureSettings == null)
        {
            StoredPressureSettings = "mm";
            CurrentPressure = Defaults.C_PRESSURE_MM;
```

```
                Preferences.Set("Settings_PressureUnits",
Defaults.GetPressureByName(StoredPressureSettings).Name);
            }
            if (StoredWindSpeedSettings == null)
            {
                StoredWindSpeedSettings = "m/s";
                CurrentWindSpeed = Defaults.C_WINDSPEED_MS;
                Preferences.Set("Settings_WindSpeedUnits",
Defaults.GetWindSpeedByName(StoredWindSpeedSettings).Name);
            }
```

Тут створюються об'єкти для кожного виду налаштувань, щоб зберігати поточне значення параметрів, і одразу ж після цього в конструкторі ініціалізуємо їх значеннями за замовчуванням. Далі ми намагаємося зчитати збережені раніше значення параметрів, і у випадку, якщо раніше ці параметри не зберігалися, також зберігаємо їх.

```
    switch (StoredTemperatureSettings)
    {
        case "C°":
            {
                CurrentTemperature = Defaults.C_TEMPERATURE_CELSIUS;
                SetButtonState(cmdTemperatureC, [cmdTemperatureF]);
                break;
            }
        case "F°":
            {
                CurrentTemperature = Defaults.C_TEMPERATURE_FAHRENHEIT;
                SetButtonState(cmdTemperatureF, [cmdTemperatureC]);
                break;
            }
        default:
            {
                if (Application.Current != null)
                {
                    SetButtonState(null, [cmdTemperatureC, cmdTemperatureF]);
                }
                break;
            }
    }
```

Далі, залежно від того, яке значення ми зрештою визначили як встановлене на даний момент, присвоюємо відповідний об'єкт з класу Defaults в теперішнє значення параметру. Повторюємо це для всіх трьох видів параметрів. Залишилося реалізувати функцію SetButtonState, яка виконує оновлення стану кнопок на сторінці.

```
    /* The following method is used to set the state of the buttons. It is used
to set the state of the buttons
```

```
        * when the user clicks on them. The selected button is highlighted, while
the other buttons are not. */
       private void SetButtonState(Button? selectedButton, Button[]? otherButtons)
       {
           if (Application.Current != null)
           {
               if (selectedButton != null)
               {
                   selectedButton.BackgroundColor =
(Color)Application.Current.Resources["MT_OnboardingButtonBackgroundSelected"];
                   selectedButton.TextColor =
(Color)Application.Current.Resources["MT_OnboardingButtonTextSelected"];
               }

               if (otherButtons != null)
               {
                   foreach (var button in otherButtons)
                   {
                       button.BackgroundColor =
(Color)Application.Current.Resources["MT_OnboardingButtonBackgroundNotSelected"];
                       button.TextColor =
(Color)Application.Current.Resources["MT_OnboardingButtonTextNotSelected"];
                   }
               }
           }
       }
```

Синтаксис цієї функції може виглядати трохи незрозумілим, особливо її сигнатура. Ми передаємо один об'єкт класу Button, який потрібно відобразити як вибраний (але символ «?» після назви класу позначає, що в якості цього об'єкту може бути передано null, тобто жодної кнопки не потрібно вибирати). Далі ми передаємо масив елементів класу Button (який теж може бути null), для того, щоб відобразити їх як не вибрані. Така реалізація дозволяє використовувати одну й ту ж функцію для всіх видів параметрів, замість того, щоб писати схожий код для кожної групи параметрів.

Закінчивши зміни в коді для всіх трьох груп параметрів, і додавши функції для збереження вибраних параметрів, обробки переходу назад та вперед, можна порівняти, як виглядає ця сторінка в новій версії додатку, та в попередній. Як бачимо на рисунку 3.9 нижче, візуально сторінки досить подібні, хоч і є деякі відмінності. Втім, нова реалізація (на рисунку зліва) є більш стабільною в роботі на екранах різних пристроїв.

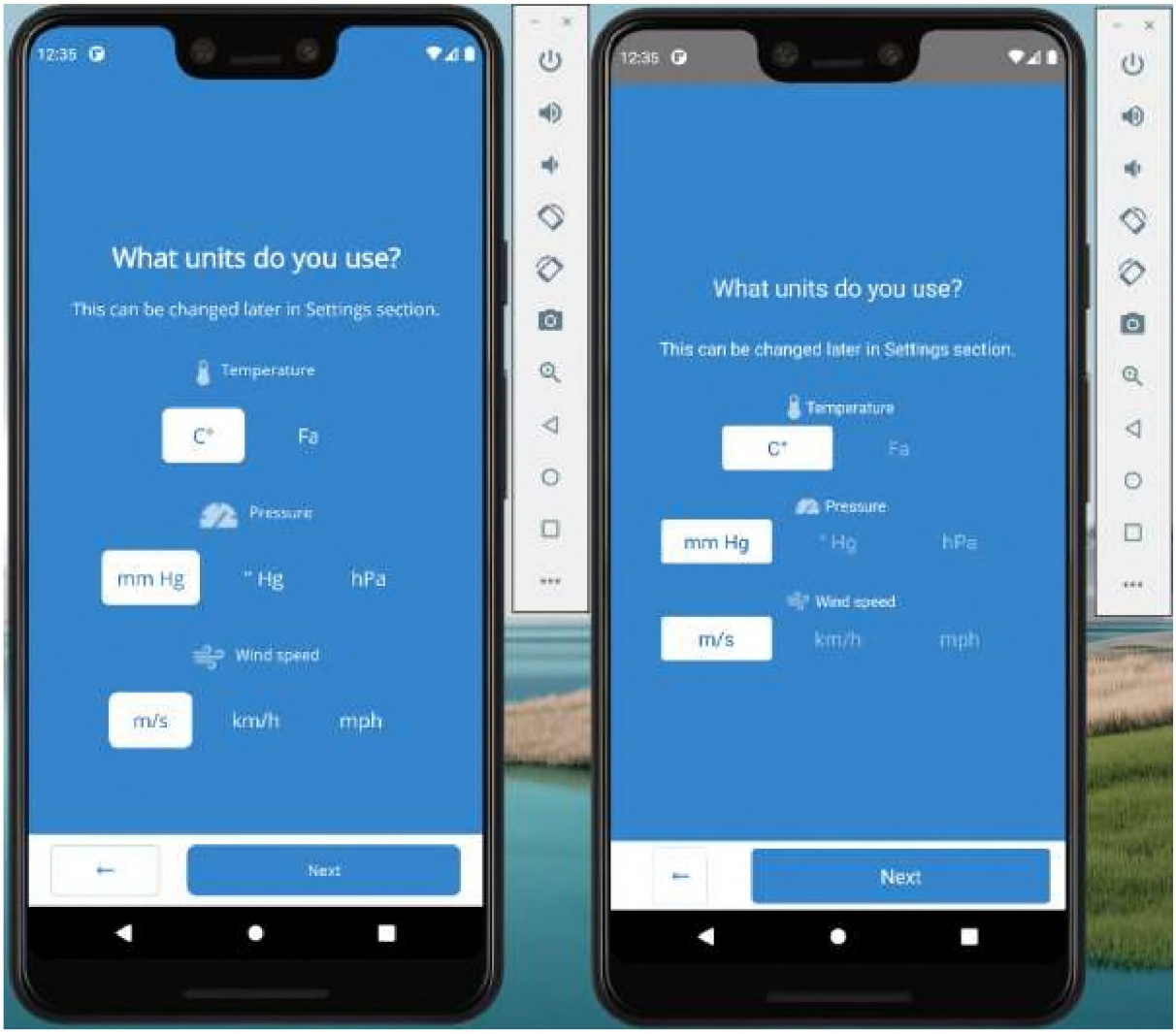

Рисунок 3.9 – Порівняння старої та нової версії Onboarding2

Для закінчення розробки екранів Onboarding потрібно додати третю сторінку. На цій сторінці буде попередження для користувача про те, що йому потрібно надати інформацію про своє розташування для того, щоб можна було отримати метеорологічні дані для цієї локації. Після натискання на кнопку потрібно запитати дозвіл на використання даних геолокації, і в разі погодження – відправити на сервер запит з даними, отриманими від пристрою. В разі відмови – відправити на сервер без даних геолокації, щоб спробувати визначити приблизне розташування за допомогою розпізнавання IP-адреси.

Як і раніше, уникатимемо фрагментів коду, які вже були пояснені раніше. В візуальній частині все досить просто: трохи тексту, зображення майже на весь

екран, і така ж панель навігації вперед і назад, як і на попередній сторінці. Програмна частина трохи складніша.

```
        private async Task<Location?> GetLocationAsync()
        {
            try
            {
                var location = await Geolocation.GetLastKnownLocationAsync();
                if(location == null)
                {
                    location = await Geolocation.GetLocationAsync(new
GeolocationRequest
                    {
                        DesiredAccuracy = GeolocationAccuracy.Medium,
                        Timeout = TimeSpan.FromSeconds(10)
                    });
                }
                return location;
            }
            catch (Exception ex)
            {
                await DisplayAlert("Error", "Failed to get location data" +
ex.Message, "OK");
                return null;
            }
        }
```

Ця функція безпосередньо запитує дані геолокації в асинхронному запиті, і або ж повертає отримані дані, або ж відображає повідомлення, що не вдалося отримати дані. Функція, яка веде до цього запиту, дещо складніша, тому її краще розібрати по частинам.

```
        private async Task RequestLocationAndProceed()
        {

            double LocationLatitude = 200;  // initialize with invalid values
            double LocationLongitude = 200; // initialize with invalid values

            string LocationID = string.Empty;   // initialize with empty string

            // Check and request location permission

            var status = await
Permissions.CheckStatusAsync<Permissions.LocationWhenInUse>();

            if (status != PermissionStatus.Granted)
            {
                status = await
Permissions.RequestAsync<Permissions.LocationWhenInUse>();
            }
```

Для уникнення виключення, яке виникає тоді, коли ми намагаємося отримати дані геолокації в ситуації, коли додаток не має відповідного дозволу, нам потрібно спочатку перевірити, чи є дозвіл на використання геоданих. Якщо такого дозволу немає, ми запитуємо у користувача такий дозвіл [35].

```
        if (status == PermissionStatus.Granted)
            {
                // Permission granted, get location data
                var location = await GetLocationAsync();
                if (location != null)
                {
                    // await DisplayAlert("Location", $"Latitude:
{location.Latitude}, Longitude: {location.Longitude}", "OK");
                    LocationLatitude = location.Latitude;
                    LocationLongitude = location.Longitude;
                }
            }
```

У випадку, якщо користувач дозволяє це, ми отримуємо дані геолокації використовуючи попередньо описану функцію, після чого готуємо об'єкт з даними, який пізніше буде використано в запиті.

```
        else
            {
                string? result = await APIRequests.FirstPlaceSelectorAsync(null,
null, null);

                if (result != null)
                {
                    Debug.WriteLine($"APIRequests.FirstPlaceSelectorAsync: Result:
{result}");
                }
                else
                {
                    await DisplayAlert("Error", "Failed to get location data. Try to
allow use of geolocation data.", "OK");
                }

                //mock data
                LocationID = "416";
            }
```

Інакше ми відправляємо на сервер запит щодо визначення координат через IP адресу, і перевіряємо, чи надійшли такі дані.

```
        // Coordinates validation
        if ((LocationLatitude < -90.0 || LocationLatitude > 90.0 || LocationLongitude
< -180.0 || LocationLongitude > 180.0) && LocationID == string.Empty)
```

```
        {
            await DisplayAlert("Error", $"Invalid coordinates: {LocationLatitude},
{LocationLongitude}", "OK");
            return;
        }
        else
        {

            if (LocationID == string.Empty)  // valid coordinates but no ID yet -
need to fetch the ID using longitude and latitude
            {
                string? result = await
APIRequests.FirstPlaceSelectorAsync(LocationLatitude, LocationLongitude, null);

                if (result != null)
                {
                    Debug.WriteLine($"APIRequests.FirstPlaceSelectorAsync: Result:
{result}");
                }
                else
                {
                    await DisplayAlert("Error", "Failed to get location data. Try to
allow use of geolocation data.", "OK");
                }

                //mock data
                LocationID = "416";
            }

            Preferences.Set("CurrentLocationLatitude", LocationLatitude);
            Preferences.Set("CurrentLocationLongitude", LocationLongitude);

            // Onboarding settings and location are defined, proceed to the main page
            Preferences.Set("HasCompletedOnBoarding", true);
            await Navigation.PushAsync(new MainPage());
        }
```

Нарешті, заключна частина цієї функції. Одне з найбільш фундаментальних правил – завжди перевіряти дані перед використанням, особливо якщо вони отримані від ненадійного джерела, а тим більше – через інтернет. Для координат проводиться попередня валідація: широта не повинна бути за межами -90 .. 90 градусів, а довгота – за межами -180 .. 180 градусів. Якщо якась із цих умов не виконується, і ми ще не маємо визначеного ідентифікатора локації (координати потрібні тільки для визначення ідентифікатора локації, тому якщо він уже є, перевіряти координати не потрібно), тоді ми повідомляємо користувача про некоректні дані геолокації. Інакше, якщо координати валідні, а ідентифікатора локації ще немає, ми робимо запит на сервер про визначення найближчої до переданих координат локації, на забуваючи перевірити при цьому

дані, отримані від сервера (не забуваємо про правило!). Після успішного отримання локації, так чи інакше, ми записуємо координати, ідентифікатор локації. Також записуємо в пам'ять пристрою відмітку про те, що процес задання початкових параметрів успішно виконано (щоб при наступному запуску користувач одразу переходив до основного екрану), і переходимо до основного екрану.

3.5 Вартість розробки

Вартість розробки розрахована в таблиці. Вартість часу, витраченого на розробку, взята з середніх показників погодинної вартості роботи на тематичних вебресурсах [36, 37, 38].

Таблиця 3.2 – Вартість розробки

Складова розробки	Вартість	Пояснення
Вартість програмного забезпечення, використаного для розробки	0 грн.	Програмне забезпечення, яке використовувалося при розробці – безкоштовне
Вартість часу, витраченого на розробку	25 год х 600 грн.	Усереднене значення рівня розробника нижче середнього на тематичних сайтах
Вартість активації облікового запису розробника на платформі Google Play	1000 грн (25 USD)	[39]
Вартість активації облікового запису розробника на платформі Apple Store	3960 грн (99 USD)	Є безкоштовна ознайомча ліцензія, але вона обмежена по можливостям, і не дає можливості безпосередньо розповсюджувати (публікувати) програмне забезпечення. [40]

Враховуючи представлені дані, загальна вартість розробки проекту є відносно невисокою, що обумовлено використанням безкоштовного програмного забезпечення для створення продукту. Основна частина витрат припадає на оплату роботи розробника та активацію облікових записів на відповідних платформах для розповсюдження програмного забезпечення. Це

підкреслює важливість врахування витрат на інструменти публікації та робочий час у процесі планування бюджетів подібних проектів.

Висновки до розділу 3

У розділі 3 проводяться практичні дії по модернізації мобільного додатку з урахуванням проведених раніше досліджень та порівняльного аналізу. Проведено вибір допоміжного програмного забезпечення для розробки. Показано детальний розбір процесу створення проекту в Visual Studio з поясненням варіантів вибору. Нарешті, безпосередньо виконано покроковий процес створення першого набору екранів функціоналу Onboarding, як візуальної так і функціональної частини. Показано відмінності між старим та новим варіантами екранів (відмінності мінімальні або відсутні).

Розглянуто та внесено основні зміни, заплановані в першому та другому розділах, включаючи підтримку додаткових властивостей компонентів для полегшення взаємодії з додатком особами з обмеженими можливостями. Також забезпечено можливість зміни локалізації користувацького інтерфейсу та включено правки, необхідні для успішного проходження модерації перед публікацією додатку в платформах поширення мобільного програмного забезпечення.

ВИСНОВКИ

При виконанні кваліфікаційної роботи було проведено аналіз наявного функціоналу мобільного додатку, розробленого раніше та аналіз змін, необхідних для відповідності оновленим вимогам замовника. Також було перераховано основні враження від розробки першої версії додатку, переваги та недоліки існуючої технології. Зібрано та систематизовано додаткові функціональні вимоги, як отримані від замовника, так і пов'язані з подальшою публікацією.

При підготовці розділу 2 було обрано альтернативну технологію для переносу існуючого функціоналу, та проведено адаптацію ресурсів, які можна було з мінімальними змінами використовувати в новій версії. Сформовано список змін, необхідних для повноцінної підтримки користувачів з обмеженими можливостями.

У третьому розділі, використовуючи раніше зібрану інформацію, було підібрано набір інструментів, що забезпечили модернізацію мобільного додатку для нових вимог, адаптовано структуру проекту для нового фреймворку. Наведено порівняльні приклади того, як структурно виглядають візуально ідентичні сторінки додатку, реалізовані в технологіях React Native та MAUI. Коротко згадано про особливості публікації в Google Play та App Store.

Проведено розрахунок вартості виконаних робіт. Як і раніше, все програмне забезпечення, використане в роботі, є безкоштовним, але публікація на обох платформах розповсюдження мобільних додатків потребує наявності платного облікового запису розробника, що враховано у вартість виконаних робіт.

На даний момент мобільний додаток повністю перенесено з реалізації під React Native на MAUI, проходить перевірка додатку в Google Play перед його інтеграцією та розміщенням. Порівнюючи досвід розробки аналогічного функціоналу в двох різних фреймворках, варто зазначити, що врахування особливостей компонування інтерфейсу, а також обмеження на операції з

даними – важливі аспекти, специфічні для кожного фреймворку. Тому, хоча всі можливі перешкоди на початку розробки передбачити складно, та найбільш принципові відмінності між інструментами розробки обов'язково потрібно врахувати, інакше можна зіткнутися з ситуацією, коли реалізація запланованого функціоналу суперечить обмеженням фреймворку.