МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри, канд. техн. наук,
доцент _____ Є. О. Давиденко
підпис

«___» _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

КРОСПЛАТФОРМНИЙ ЗАСТОСУНОК ВІДОБРАЖЕННЯ ПОГОДНИХ УМОВ

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 409. 21910919

Студента _____О. С. Сігєєв
підпис
«__» _____ 2023 р.

Керівник канд. пед. наук, доцент _____К. О. Кірей
підпис
«__» _____ 2023 р.

Консультант канд. техн. наук,
доцент кафедри екології _____А. О. Алексєєва
підпис
«__» _____ 2023 р.

м. Миколаїв – 2023 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ
Зав. кафедри
_____ Є. О. Давиденко
«_____» _____ 20__ р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано студенту групи 409 факультету комп'ютерних наук

Сігєєву Олександру Сергійовичу

(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи:

Кросплатформний застосунок відображення погодних умов

Затверджена наказом по ЧНУ від «17» березня 2023 р. № 60

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є прототип мобільного застосуноку із функціями відображення погодних умов, зміни мови застосунку та заданням бажаної локації

4. Перелік питань, що підлягають розробці

− дослідження предметної галузі та аналіз застосунків-аналогів;

− аналіз погодних сервісів та API для отримання даних про погоду;

− моделювання та проєктування ПЗ;

− розробка функціоналу мобільного застосунку;

− тестування застосунку;

− аналіз результатів розробки;

5. Перелік графічних матеріалів

Презентація.

6. Завдання до спеціальної частини

_____

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
доцент Алексєєва А. О.	Кафедра екології Медичного інституту ЧНУ ім. Петра Могили	Спеціальна частина з охорони праці

Керівник роботи канд. пед. наук, доцент Кірей Катерина Олександрівна
(посада, прізвище, ім'я, по батькові)

_____
(підпис)

Завдання прийнято до виконання
_____Сігєєвим Олександром Сергійовичем_____
(прізвище, ім'я, по батькові студента)

_____
(підпис)

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Кросплатформний застосунок відображення погодних умов

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	17.03.2023	20.13.2023	виконано
2.	Огляд літератури за темою роботи	21.03.2023	24.03.2023	виконано
3.	Складання календарного плану КРБ	27.03.2023	28.03.2023	виконано
4.	Аналіз предметної області	29.03.2022	31.03.2022	виконано
5.	Визначення проєктних проблем та пошук рішень	03.04.2023	06.04.2023	виконано
6.	Моделювання та проєктування ПЗ	07.04.2023	12.04.2023	виконано
7.	Розробка функціоналу застосунку, тестування та апробація розробленого ПЗ, аналіз результатів тестування	13.04.2023	21.05.2023	виконано
8.	Розробка спеціальної частини з охорони праці	21.05.2023	28.05.2023	виконано
9.	Відгук керівника КРБ	29.05.2023	31.05.2023	виконано
10.	Оформлення КРБ та презентації	01.05.2023	04.05.2023	виконано
11.	Попередній захист			
12.	Рецензування			
13.	Завершення оформлення КРБ та презентації			
14.	Захист кваліфікаційної роботи			

Розробив студент Сігєєв Олександр Сергійович _____

(*прізвище, ім'я, по батькові)* *(підпис)*

«__» _____ 20__ р.

Керівник роботи канд. пед. наук, доцент Кірей К. О. _____

(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«__» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Кросплатформний застосунок відображення погодних умов»

Студент 409 гр.: Сігєєв Олександр Сергійович

Керівник: канд. техн. наук, доцент Кірей К. О.

Кваліфікаційна робота бакалавра присвячена створенню кросплатформного мобільного застосунку для відображення погодних умов на IOS та Android платформах. Метою є підвищення мобільності отримання користувачами актуальної інформації про погоду шляхом розробки відповідного кросплатформного застосунку. У цій роботі було проведено аналіз предметної галузі та аналогічних програмних систем зі схожим функціоналом, визначені вимоги до функціональності та дизайну застосунку, обрані технології для розроблення кросплатформного застосунку та реалізований і протестований застосунок. Основним об'єктом кваліфікаційної роботи була організація та управління процесами отримання та надання користувачу інформації про погодні умови, а предметом - організація збору та надання користувачу інформації про погодні умови з використанням погодних сервісів та API.

КРБ викладена на 61 сторінки, вона містить 4 розділи, 17 ілюстрацій, 4 таблиці, 11 джерел в переліку посилань.

Ключові слова: кросплатформний застосунок, мобільний застосунок, React Native, React, Redux, IOS, Android, погодні умови, API, відображення погодних умов, прогноз погоди, місцезнаходження користувача.

ABSTRACT

of the Bachelor`s Thesis

«Cross-platform application for displaying weather conditions»

Student 409 group: Sihieiev Oleksandr Serhiyovich

Supervisor: PhD, associate professor Kirei K. O.

The bachelor's thesis is dedicated to creating a cross-platform mobile application for displaying weather conditions on IOS and Android platforms. The goal of this bachelor's thesis is to increase the mobility of users' access to up-to-date weather information through the development of a corresponding cross-platform application. In this work, an analysis of the subject area and similar software systems with similar functionality was carried out, requirements for the functionality and design of the application were determined, technologies were chosen for developing the cross-platform application, and the application was implemented and tested. The main object of the thesis was the organization and management of processes for obtaining and providing users with information about weather conditions, and the subject was the organization of the collection and provision of information about weather conditions to users using weather services and APIs.

The bachelor's qualification work consists of 61 pages, 4 chapters, 17 illustrations, 4 tables, and 11 sources in the list of references.

Keywords: cross-platform application, mobile application, React Native, React, Redux, IOS, Android, weather conditions, API, testing, functional requirements, application architecture, displaying weather conditions, weather forecast, user location, interface.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ	–	програмне забезпечення
ОС	–	операційна система
API	–	application programming interface
UML	–	unified modeling language
UI	–	user interface
UX	–	user experience

ВСТУП

Кваліфікаційну роботу бакалавра присвячено розробці кросплатформного мобільного застосунку для відображення погодних умов на IOS та Android платформах.

Актуальність теми зумовлена тим, що погодні умови є важливим елементом в повсякденному житті людей, тому з'являється необхідність у розробці кросплатформного застосунку, який дозволить користувачам отримувати актуальну інформацію про погоду на різних пристроях та платформах. Крім того, розробка такого застосунку відповідає вимогам сучасного ринку ПЗ, де велике значення має мобільність та доступність продукту на різних платформах.

Об’єктом кваліфікаційної роботи є методи організації та управління процесами отримання та надання користувачу інформації про погодні умови.

Предметом кваліфікаційної роботи є організація збору та надання користувачу інформації про погодні умови з використанням погодних сервісів та API.

Метою є спрощення отримання користувачами актуальної інформації про погоду шляхом розробки відповідного кросплатформного застосунку.

Практичне застосування: кросплатформний застосунок можна використовувати для отримання користувачами актуальної інформації про погоду.

Для досягнення мети необхідно виконати такі **завдання**:

− провести аналіз предметної галузі та аналогічних програмних систем зі схожим функціоналом;

− визначити вимоги до функціональності та дизайну застосунку відображення погодних умов;

− обрати технології для розроблення кросплатформного застосунку відображення погодних умов на різних пристроях та платформах;

− реалізувати та протестувати застосунок.

Кваліфікаційна робота бакалавра складається із 31 сторінки, містить 13 ілюстрацій, 4 таблиці та 10 використаних джерел.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ

1.1 Огляд застосунків-аналогів

Аналіз предметної області – важливий крок для вдалого створення будь-якого проєкту, оскільки це надає змогу визначити потрібний функціонал, основні недоліки та переваги на які варто звернути увагу. Для аналізу аналогів було обрано три застосунки: The Weather Channel (табл. 1.1), Live Weather: Radar & Forecast (табл. 1.2) та WeatherBug – Weather Forecast (табл. 1.3).

The Weather Channel

Застосунок для відстеження погодних умов на IOS та Android (табл. 1.1), розробляється командою The Weather Channel. Логотип застосунку наведено на рис. 1.1, вигляд інтерфейсу – на рис. 1.2.

Таблиця 1.1 – Опис застосунку «The Weather Channel»

Назва	The Weather Channel
Виробник:	The Weather Channel
Архітектура	Мобільний застосунок для iOS/Android
Функції	1. Перегляд погодних умов. 2. Застереження щодо небезпечних погодних умов. 3. Перегляд метереологічного відеоконтенту.
Переваги	1. Умовно безкоштовний застосунок. 2. Кросплатформний. 3. Сумісність з темною темою. 4. «Віджет» для домашньої сторінки.
Недоліки	1. Проблеми з точністю показань. 2. Занадто агресивна монетизація.
Вебсайт	https://apps.apple.com/us/app/weather-the-weather-channel/id295646461

Рисунок 1.1 – Логотип застосунку «The Weather Channel»

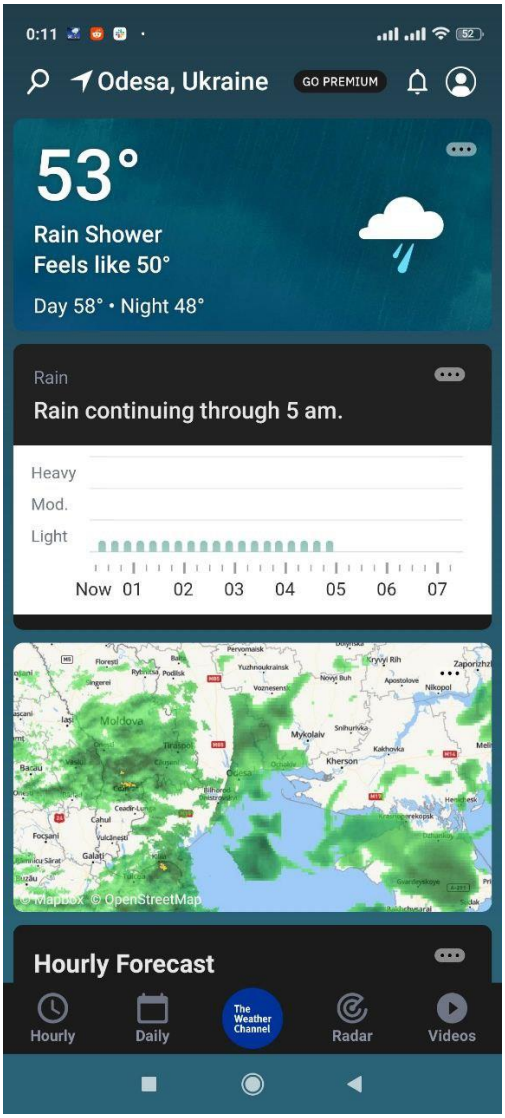

Рисунок 1.2 – Вигляд інтерфейсу «The Weather Channel»

The Weather Channel має можливість перегляду погодних умов на 15 днів уперед та застереження щодо небезпечних явищ. Окремо можна виділити «віджет» для домашньої сторінки та сумісність інтерфейсу з темною темою. Недоліками виступає агресивна монетизація (левова частка функцій схована за пейволом, тому застосунок сильно втрачає привабливість на фоні Live Weather: Radar & Forecast і WeatherBug – Weather Forecast, які без підписки надають більше можливостей та менше рекламних інтеграцій) та погана точність метеорологічних показань (неправильна температура тощо).

Live Weather: Radar & Forecast

Застосунок для відстеження погодних умов на Android (табл. 1.2). Логотип застосунку наведено на рис. 1.3, вигляд інтерфейсу – на рис. 1.4.

Таблиця 1.2 – Опис застосунку «Live Weather: Radar & Forecast»

Назва	Live Weather: Radar & Forecast
Виробник:	APPS INNOVA
Архітектура	Мобільний застосунок для Android
Функції	1. Погодинний та щоденний прогноз погоди. 2. Відображення часу сходу та заходу сонця. 3. Щохвилинне оновлення погодних умов. 4. Сповіщення про небезпечні погодні явища. 5. Показ погоди на 15 діб.
Переваги	1. Точність показань. 2. Швидке оновлення даних
Недоліки	1. Дизайн інтерфейсу.
Вебсайт	https://play.google.com/store/apps/details?id=com.appsinnova.weatherforecast.liveweather

Рисунок 1.3 – Логотип застосунку «Live Weather: Radar & Forecast»

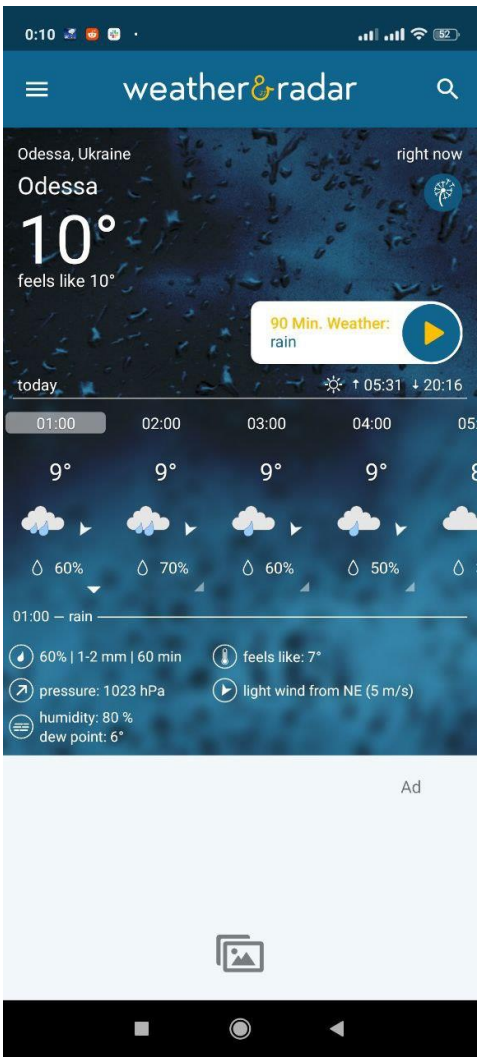

Рисунок 1.4 – Вигляд інтерфейсу «Live Weather: Radar & Forecast»

Live Weather: Radar & Forecast розробляється командою APPS INNOVA. Також має можливість перегляду погодних умов на 15 днів уперед та застереження щодо небезпечних явищ. Недоліками виступає інтерфейс з поганою читабельністю, а також не зручне розташування меню.

WeatherBug – Weather Forecast

Застосунок для відстеження погодних умов на IOS/Android (табл. 1.3). Логотип застосунку наведено на рис. 1.5, вигляд інтерфейсу – на рис. 1.6.

Таблиця 1.3 – Опис застосунку «WeatherBug – Weather Forecast»

Назва	WeatherBug – Weather Forecast
Виробник:	WeatherBug
Архітектура	Мобільний застосунок для iOS/Android
Функції	1. Інтерактивна мапа погодних умов. 2. Відображення кількості пилку в повітрі. 3. Сповіщення про небезпечні погодні умови. 4. Прогноз погоди на 2 тижні вперед.
Переваги	1. Інтуїтивно зрозумілий інтерфейс
Недоліки	*Недоліків не виявлено*
Вебсайт	https://apps.apple.com/us/app/weatherbug-weather-forecast/id281940292

Рисунок 1.5 – Логотип застосунку «WeatherBug – Weather Forecast»

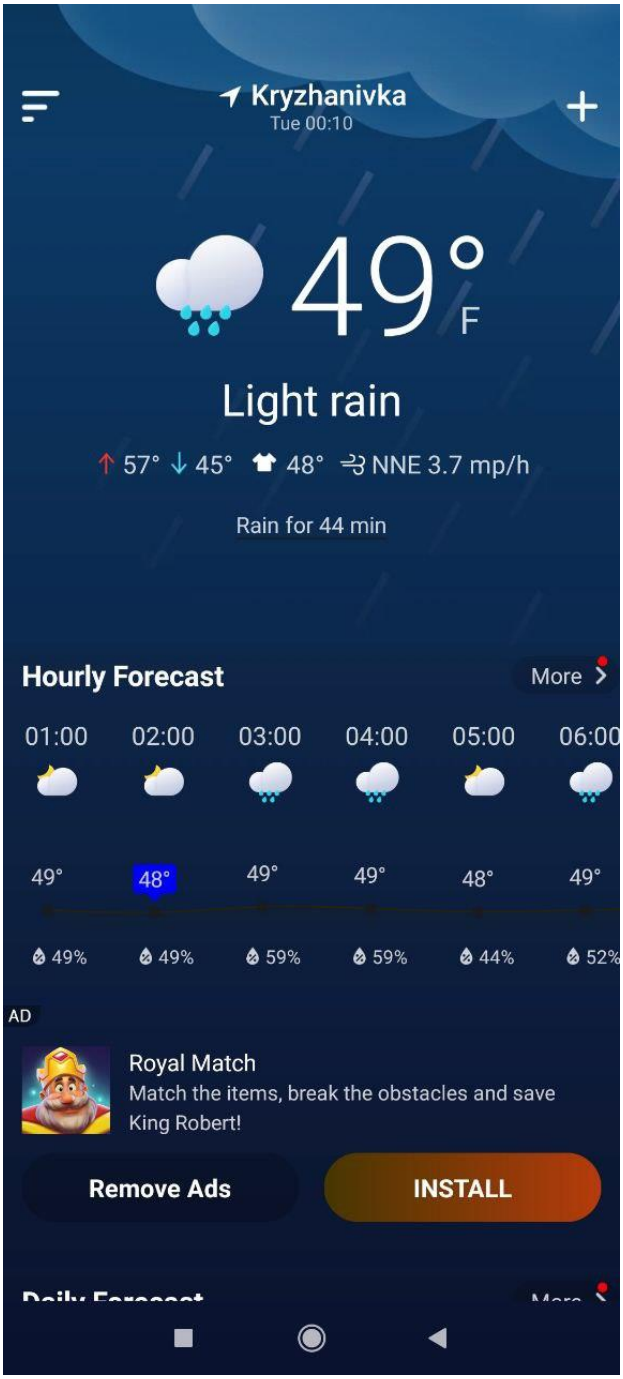

Рисунок 1.6 – Вигляд інтерфейсу «WeatherBug – Weather Forecast»

WeatherBug – Weather Forecast розробляється командою WeatherBug. Окрім прогнозу погоди має інтерактивну мапу погодних умов та відображення кількості пилку в повітрі, що цінно для алергиків. Недоліків у порівнянні з попередніми застосунками не виявлено.

1.2 Опис функціоналу застосунку відображення погодних умов

Ґрунтуючись на проведеному аналізі аналогів було сформовано перелік функціональних можливостей функціонал застосунку відображення погодних умов (табл. 1.4).

Таблиця 1.4 – Опис застосунку відображення погодних умов

Функції	1) погодинне відображення погодних умов; 2) прогноз погоди на 2 тижні; 3) автоматичне визначення локації користувача; 4) можливість обрати локацію для відображення погоди власноруч; 5) автоматична детекція мови користувача; 6) можливість обрати мову UI власноруч; 7) відображення поточної температури; 8) детальний опис погоди на певний день тижня.
Користувачі	1) користувач
Сценарії роботи	1. Користувач встановлює застосунок з метою отримати інформацію про погодні умови, йому пропонується надати доступ до геолокації та автоматично обрати мову застосунку. 2. Користувач запускає застосунок та опиняється на головному екрані, де відображено поточну температуру повітря для заданої локації. 3. Користувач відкриває список з погодинним прогнозом погоди, де йому показуються дані на кожну годину дня.

Кінець таблиці1.4

	4. Користувач закриває список та переходить до вибору локації, де використовує інтегровану мапу для обрання необхідного місця. 5. Користувач повертається на головний екран та переходить до меню вибору мови застосунку, де обирає з української та англійської локалізацій. 6. Користувач повертається до головного меню, де переходить до прогнозу на 14 діб. 7. Користувач відкриває детальний опис погодних умов для певної доби.

Застосунок відображення погодних умов має підтримувати iOS та Android системи. У самому застосунку мають бути реалізовані: динамічна зміна теми інтерфейсу, погодинне відображення погодних умов та прогноз на 14 діб, автоматичне визначення локації користувача та можливість вибору вручну. Окремо можна виділити інтеграцію Google Maps та автоматичну детекцію мови пристрою користувача для покращення UX.

1.3 Специфікація вимог до ПЗ

Призначення застосунку, для якого розробляється ПЗ

Призначення застосунку є підвищення мобільності отримання користувачами актуальної інформації про погоду шляхом розробки відповідного кросплатформного застосунку для систем iOS та Android.

Погодження, що ухвалені в програмній документації

Було погоджено, що для створення ПЗ та його роботи буде використано такі технології: фреймворк React Native та відповідно мова програмування JavsScript у поєднанні з типізованою мовою TypeScript, а також система для керуванням стану застосунку Redux.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування

Дане ПЗ не має суттєвих обмежень у сфері споживання, можна застосовувати у звичайному житті.

Характеристика користувачів

Основні характеристики користувача: наявність смартфону з підтримкою Інтернет-з’єднання.

Загальна структура і склад системи

Основна структура: мобільний застосунок

Загальні обмеження

Обмеження для роботи: наявність Інтернет-зв’язку

ОСНОВНІ ФУНКЦІЇ КРОСПЛАТФОРМНОГО ЗАСТОСУНКУ ВІДОБРАЖЕННЯ ПОГОДНИХ УМОВ

1. Відображення погодних умов

Опис функції

Ця функція дає можливість отримувати погодні дані (температуру, тиск, вологість повітря).

Вхідна і вихідна інформація

Вхідна – підключення до Інтернет-мережі та геолокації.

Вихідна – оновлювані дані про погодні умови.

Функціональні вимоги

Запит і отримання даних з API.

2. Обрання локації

Опис функції

Функція надає можливість обрати геолокацію для відображення прогнозу погоди

Вхідна і вихідна інформація

Вхідна – доступ до геолокації.

Вихідна – дані про локацію (широта та довгота).

Функціональні вимоги

Використання даних (широти та довготи) для отримання прогнозу погоди.

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Джерела і зміст вхідної інформації (даних)

В даному ПЗ джерелами вхідної інформації є користувач (надання дозволів, підключення нативних сервісів – Інтернет-з’єднання, геолокація, мова ОС.

Нормативно-довідкова інформація

Вимоги відсутні.

Вимоги до способів організації, збереження та ведення інформації

Отримання відбувається через протокол HTTP.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Вимоги до технічного забезпечення мають певні обмеження у вигляді відповідного версії ОС Android (вище 6.0) та iOS (вище 9). Необхідний Інтернет-зв’язок.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Архітектура включає питомі для мобільного застосунку частини.

Системне програмне забезпечення

Застосунок має бути побудований з використанням фреймворку React Native на мові TypeScript (JavaScript).

Мова і технологія розробки ПЗ

ПЗ має розроблюватись за допомогою мови TypeScript (JavaScript) на фреймворку React Native.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс має задовольняти вимоги UI/UX-дизайну для інтуітивного розуміння користувачем навігації застосунком. Інтерфейс має містити показання погодних умов, навігаційне меню, мапу, поле пошуку міста, список підтримуємих мов та список днів з прогнозом погоди, а також екран детального опису погоди на конкретну добу.

Апаратний інтерфейс

Мобільний девайс користувача (смартфон, планшет) на базі ОС iOS або Android.

Програмний інтерфейс

У якості програмного інтерфейсу використовується набір компонентів з React Native, які не залежать від платформи, таких як View, Text та Image, які безпосередньо представляють відповідні нативні елементах інтерфейсу користувацької платформи. Для розробки ПЗ використовується Visual Studio Code, Android Studio та XCode.

Комунікаційний протокол

Застосунок передбачає використання протоколу HTTPS.

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

ПЗ є доступним для користувачів, що мають на меті отримати актуальний прогноз погоди у конкретній локації за наявністю апаратного інтерфейсу і підтримки ним Інтернет-зв’язку та, бажано, геолокації.

Супроводженість

Застосунок потребує супроводу у разі оновлення API або необхідності їх заміни.

Переносимість

ПЗ може працювати на ОС iOS та Android. За потреби можлива адаптація до браузерного середовища та десктопних ОС (Windows, MacOS).

Продуктивність

Продуктивність ПЗ залежить від швидкості Інтернет-мережі.

Надійність

Персональні дані користувача не зберігаються на сервер, але інформація про геолокацію відправляється на API, а також локально зберігається мова UI та обрана локація.

Безпека

Застосунок використовує HTTP-запити.

Висновки до розділу 1

В першому розділі кваліфікаційної роботи бакалавра роглянуто та проаналізовано застосунки-аналоги з функціями відображення погодних умов, зроблено аналіз функціоналу аналогічних систем у сфері метеорології. Виокремлено переваги та недоліки аналізованих систем, проведено детальний аналіз застосунку, що розробляється, визначено функції, сценарії використання, складено специфікацію вимог до застосунку.

2 ПРОЄКТУВАННЯ КРОСПЛАТФОРМНОГО ЗАСТОСУНКУ ВІДОБРАЖЕННЯ ПОГОДНИХ УМОВ

1.3 Діаграма сценаріїв використання (Use Case)

Use Case, або сценарій використання, – це опис послідовності дій, які користувач може виконувати в проєктованому застосунку, взаємодіючи з його функціоналом [1]. Use Case діаграма наведена на рис. 2.1.

Короткий Use Case

Користувач встановлює застосунок з метою отримати інформацію про погодні умови. Користувач запускає застосунок та опиняється на головному екрані, де відображено поточну температуру повітря для заданої локації.

Поверхневий Use Case

Головний сценарій (успішний)

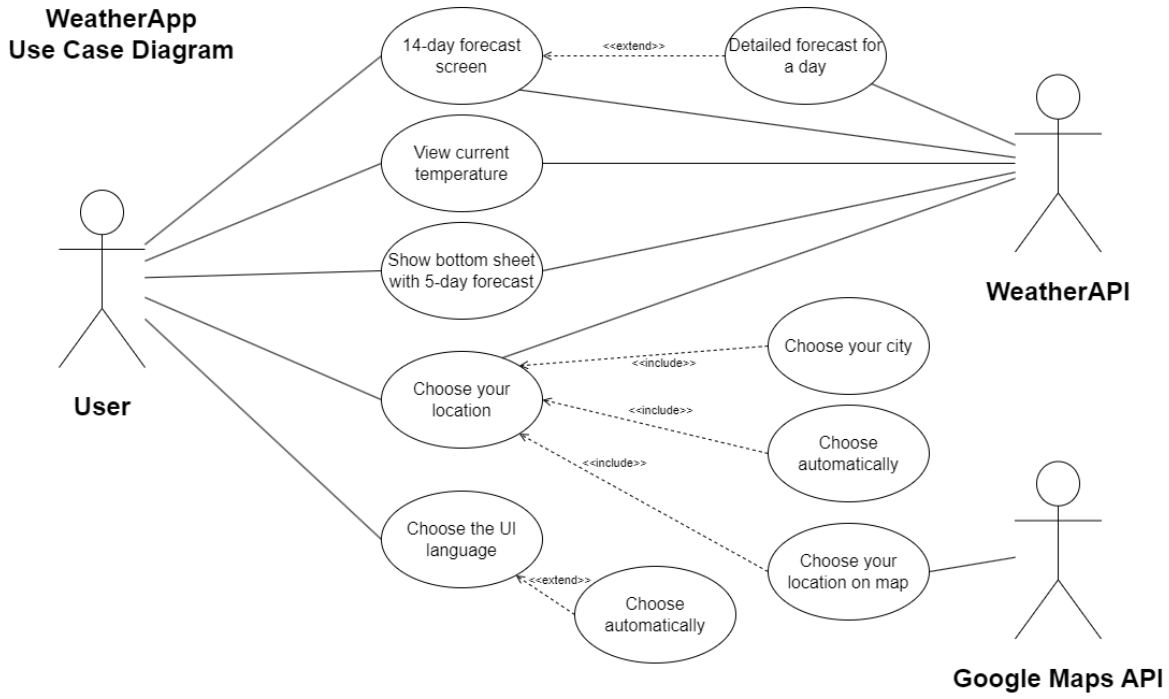

Рисунок 2.1 – Use Case діаграма застосунку

Користувач встановлює застосунок з метою отримати інформацію про погодні умови, йому пропонується надати доступ до геолокації та автоматично обрати мову застосунку. Користувач запускає застосунок та опиняється на головному екрані, де відображено поточну температуру повітря для заданої локації. Користувач відкриває список з погодинним прогнозом погоди, де йому показуються дані на кожну годину дня. Користувач закриває список та переходить до вибору локації, де використовує інтегровану мапу для обрання необхідного місця. Користувач повертається на головний екран та переходить до меню вибору мови застосунку, де обирає з української та англійської локалізацій. Користувач повертається до головного меню, де переходить до прогнозу на 14 діб. Користувач відкриває детальний опис погодних умов для певної доби.

Альтернативні сценарії:

1) користувач відмовляється надати доступ до даних про геолокацію. Застосунок повідомляє, що без доступу необхідно власноруч обрати бажану локацію. Користувач переходить до меню вибору локації;

2) користувач не увімкнув Інтернет-зв’язок. Виникає помилка, користувач має змогу натиснути кнопку для повторення запиту до API та перевірки стану мережі;

1.4 Діаграми діяльності

Застосунок із функцією показу погодних умов повинен мати зручний та інтуітивно зрозумілий інтерфейс. Щоб відображувати основні дії, які може здійснити користувач зазвичай використовують діаграми діальності [1]. Створені діаграми діяльності наведені нижче:

1) *Обирання локації* (рис. 2.2). Діграма відображає діяльність «**Обирання локації**». Коли користувач заходить у застосунок, його просять надати доступ до геолокації для автоматичного визначення координат, за якими буде надіслано запит для отримання інформації про погодні умови. Якщо користувач відмовляє у наданні доступу до сервісу геолокації, його перекидає до меню вибору локації. Звідси доступні два варіанти: пошук міста зі списку або пошук на мапі, при виборі локації налаштування буде збережено. Після вибору користувач потрапляє на головну сторінку, звідки може перейти до меню вибору локації;

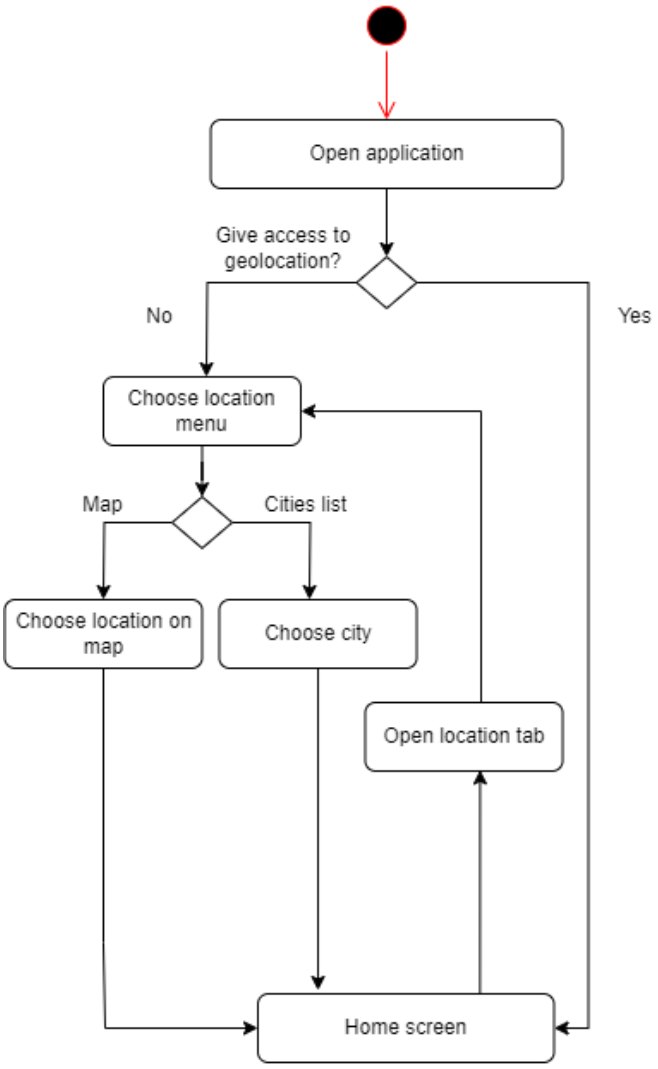

Рисунок 2.2 – Діаграма діяльності обираня локації

2) *Обирання мови інтерфейсу* (рис 2.3). Ця діаграма відображає діяльність, коли користувач переходить з головного меню до меню вибору мови застосунку. Тут відображено список доступних локалізацій, при виборі налаштування буде збережено.

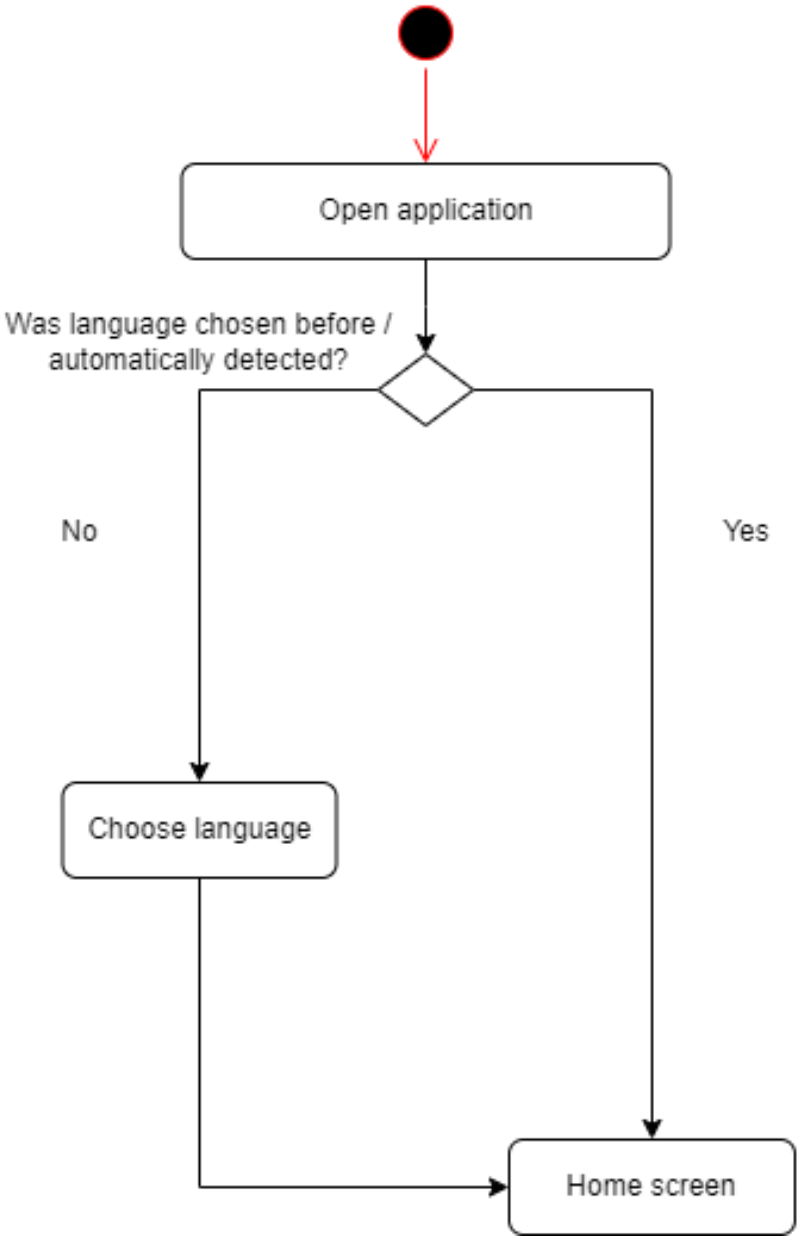

Рисунок 2.3 – Діаграма діяльності обирання мови інтерфейсу користувача

Діаграми було створені за допомогою спеціалізованого ПЗ створення діаграм draw.io для демонстрації деяких процесів дій користувача.

1.5 Діаграми послідовності

Діаграма послідовності (sequence diagram) – діаграма, на якій показані взаємодії об'єктів, впорядковані за часом їх прояву, що дозволяє візуалізувати тимчасові відносини між переданими повідомленнями. За допомогою діаграми послідовності можна уявити взаємодію елементів моделі як своєрідний часовий графік "життя" всієї сукупності об'єктів, пов'язаних між собою для реалізації варіанту використання програмної системи, досягнення бізнес-мети або виконання будь-якої задачі [1].

Для детальнішого розуміння того, яким чином відбувається ініціалізація та початковий запуск застосунку було розроблено діаграму послідовності (рис. 2.4).

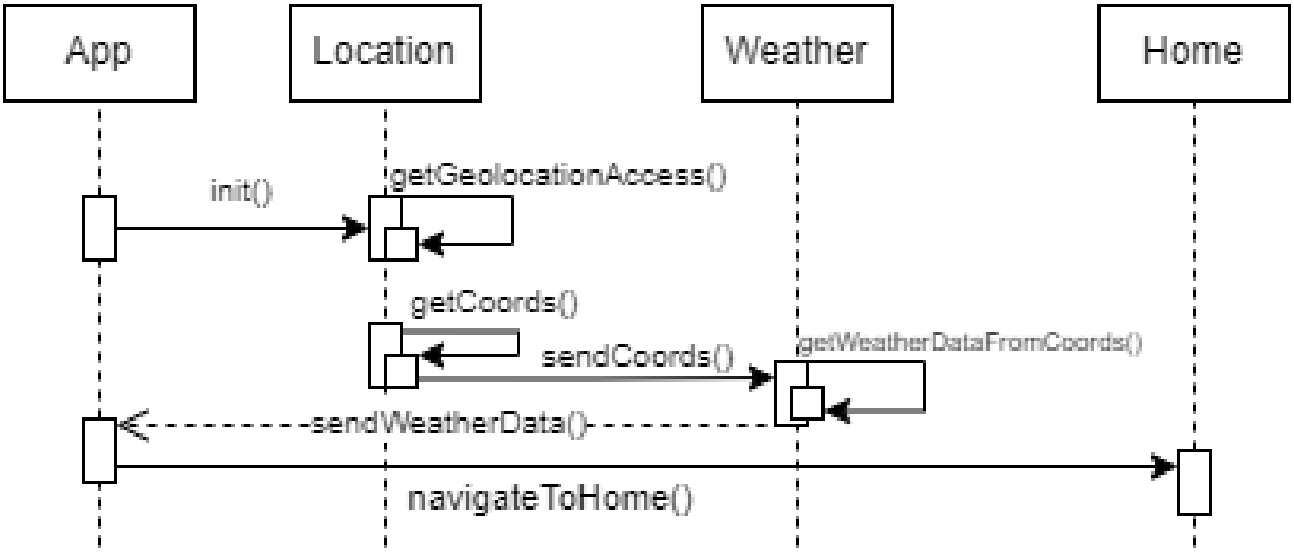

Рисунок 2.4 – Діаграма послідовності ініціалізації застосунку

Після запуску застосунку відбувається ініціалізація запиту геолокації, після чого дані про місцезнаходження користувача відправляються до метеорологічного API, яке повертає значення погодних показників.

2.5 Діаграма станів та переходів

Діаграми станів та переходів це ще один вид UML-діаграм, які допомагають полегшити розуміння, які стани має застосунок та які переходи між ними існують [1]. Нижче показану узагальнену діаграму для ініціалізації застосунку та вибору мови інтерфейсу (рис. 2.5).

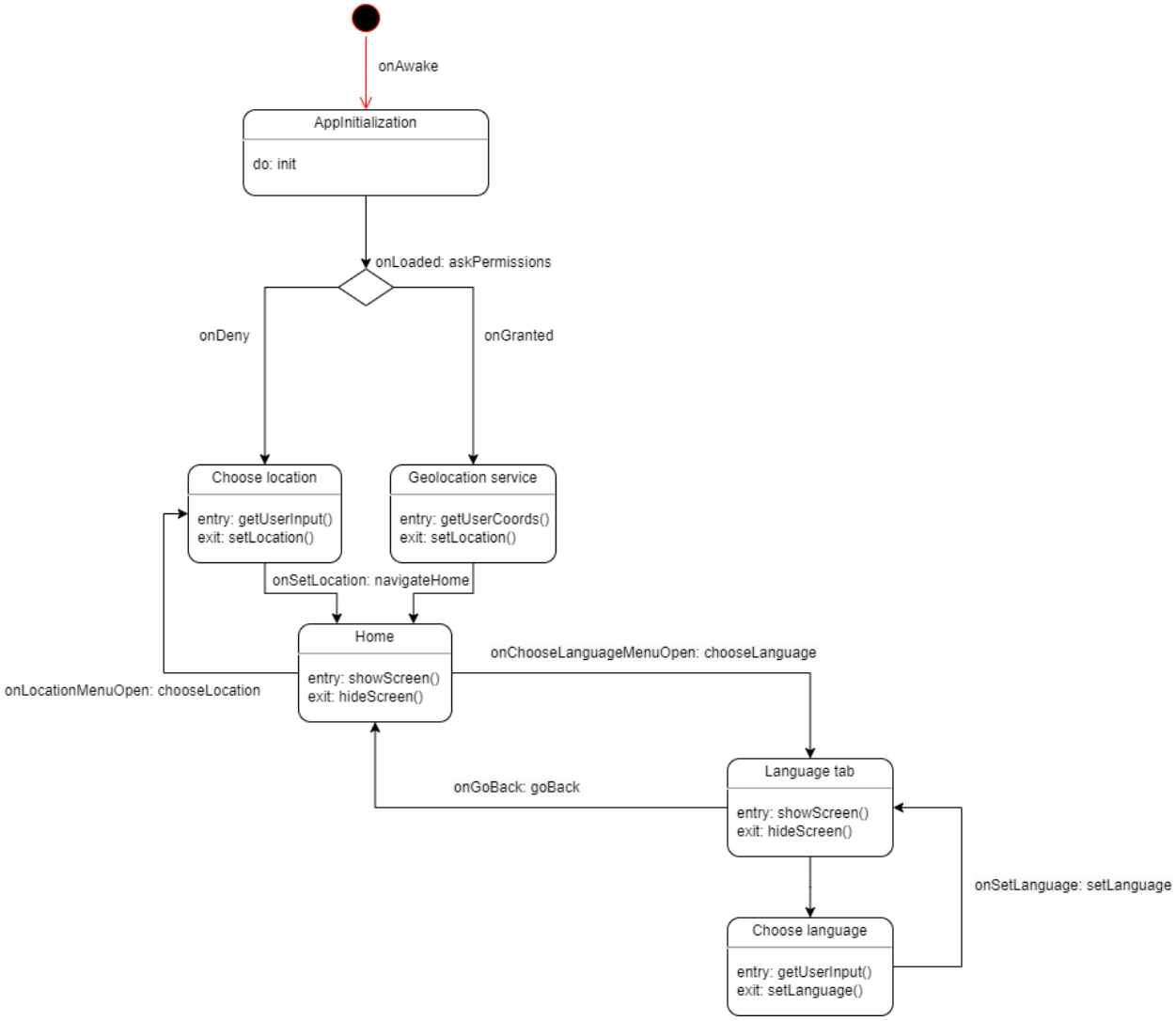

Рисунок 2.5 – Діаграма станів та переходів застосунку

Початковий стан гри запускає «маунт» або ініціалізацію головного компоненту. Після цього з’являється діалогове вікно з запитуванням доступу до геолокації користувача.

При наданні дозволів відкривається локація зберігається та відбувається перехід на головний екран застосунку. При відмові відбувається перехід до меню вибору локації власноруч, після вибору – аналогічно до попереднього.

Після переходу у стан *Home* також можливо перейти до *Language Tab*, звідки можна здійснити вибір мови інтерфейсу застосунку, після чого здійснити повернення до головного екрану.

2.6 Діаграма класів

Діаграма класів (рис. 2.6) є важливим шаблоном для переведення моделей у програмний код, є невід’ємною складовою для проєктування проєкту, що використовує об’єктно-орієнтований підхід програмування.

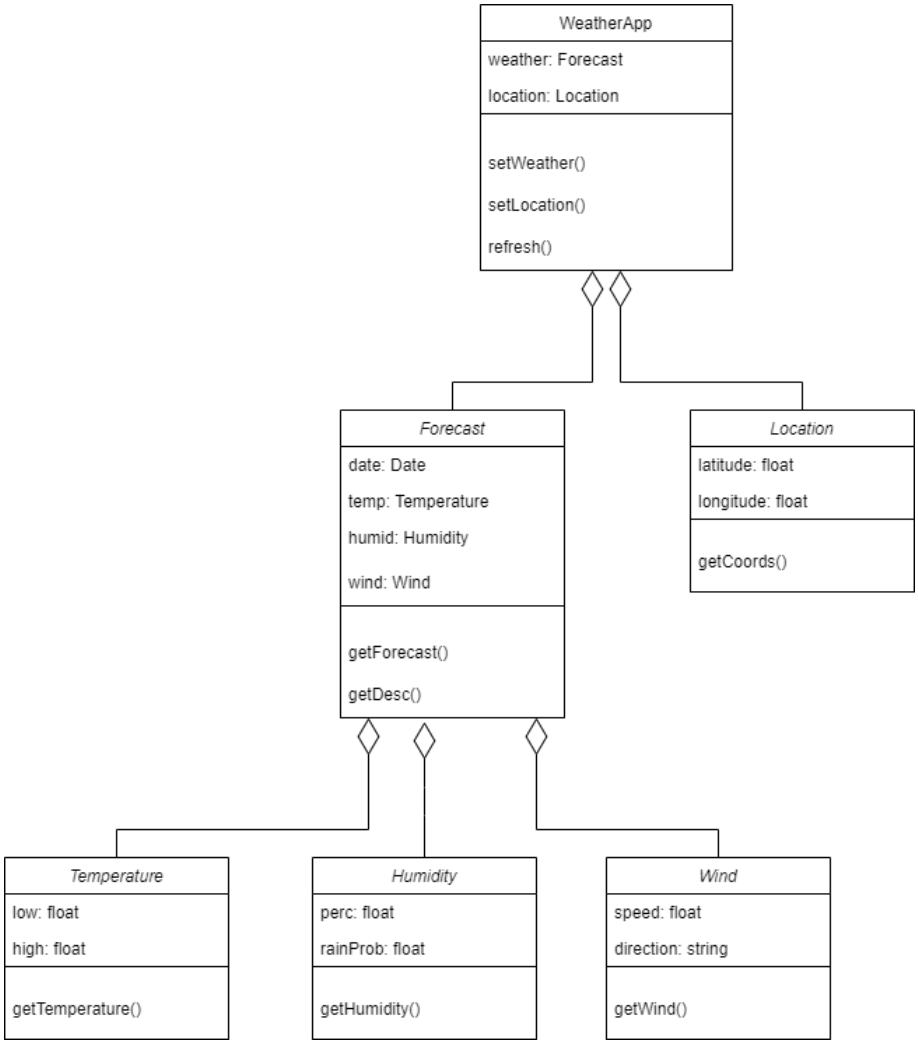

Рисунок 2.6 – Діаграма основних класів

На даній діаграмі (див. рис. 2.7) відображені основні класи, що використовуються для відображення прогнозу погоди.

Висновки до розділу 2

В другому розділі було описано основні етапи, сценарії використання та алгоритми роботи застосунку. Розроблено UML-діаграми, а саме діаграму використання, діаграми станів та переходів, діаграми діяльності та інші, завдяки яким було описано алгоритми та архітектуру застосунку.

3. АНАЛІЗ СТЕКУ ТЕХНОЛОГІЙ

2.1 Огляд основного стеку технологій

Для створення застосунку обрано такий стек технологій:

– мови програмування JavaScript;

– мова програмування TypeScript (яка базується на JavaScript);

– бібліотека React.js;

– фреймворк створення нативних застосунків React Native (на основі бібліотеки React);

– Google Maps для покращення UX вибору локації;

– OpenWeather API для отримання даних про погодні умови;

– Redux для керуванням глобального стану застосунку (рис. 2.2).

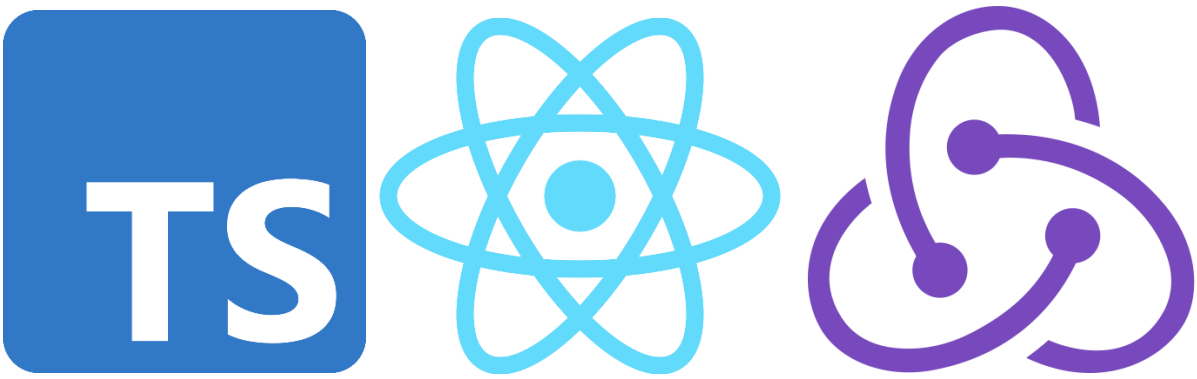

Рисунок 3.1 – Основний стек технологій

React Native поєднує найкращі аспекти розробки під конкретну платформу з React - кращою бібліотекою JavaScript для створення інтерфейсів користувача. React Native можна використовувати існуючих проєктах для Android та iOS або створити з нуля цілком новий застосунок.

React-примітиви рендеряться на нативний інтерфейс платформи, що означає, що застосунок використовує ті ж нативні API платформи, що й інші. З React Native одна команда може підтримувати кілька платформ і використовувати загальну технологію – React. Компоненти React обгортають існуючий нативний код та взаємодіють з нативними API через декларативну парадигму UI React та JavaScript. Це дозволяє розробляти нативні застосунки для нових команд розробників та дозволяє існуючим нативним командам працювати набагато швидше.

Facebook випустила React Native у 2015 році і з тих пір постійно його підтримує. У 2018 році React Native мав другу найбільшу кількість учасників серед усіх репозиторіїв на GitHub. Сьогодні React Native підтримується внесками окремих осіб та компаній з усього світу, включаючи Callstack, Expo, Infinite Red, Microsoft та Software Mansion.

TypeScript додає додатковий синтаксис до JavaScript, щоб забезпечити більш тісну інтеграцію з редактором та допомогти виявляти помилки на ранніх етапах розробки. Код TypeScript перетворюється на JavaScript, який запускається де завгодно запускається JavaScript: у браузері, на Node.js та у застосунках.

Redux допомагає писати застосунки, які поводяться послідовно, працюють у різних середовищах (клієнт, сервер та мобільний застосунок) і легко тестуються. Централізація стану та логіки застосунку дозволяє використовувати потужні можливості, такі як відміна/повторення дій, збереження стану та багато іншого.

Redux Toolkit — це офіційний стандартний підхід для написання логіки Redux. Він охоплює ядро Redux і містить пакети та функції, які необхідні для

створення програми з Redux. Redux Toolkit спрощує більшість завдань Redux, запобігає поширеним помилкам і полегшує написання ПЗ.

2.2 Огляд мови програмування JavaScript

JavaScript [2] – це високорівнева, інтерпретована мова програмування, яка використовується для розробки веб-додатків і динамічного веб-контенту. Вона була створена для реалізації можливостей взаємодії користувача з веб-сторінками і забезпечення валідації даних на клієнтській стороні. Проте, з часом JavaScript став універсальною мовою програмування, що використовується як на клієнтській стороні (у браузері), так і на серверній стороні (за допомогою платформи Node.js).

JavaScript має динамічний тип даних, що означає, що типи даних змінюються автоматично залежно від значень, що їм призначаються. Вона підтримує числа, рядки, логічні значення (true і false), масиви, об'єкти та null і undefined. JavaScript також має можливість роботи з функціями в якості об'єктів першого класу, що означає, що вони можуть бути передані як аргументи функцій, зберігатися в змінних та повертатися як значення функцій.

Однією з сильних сторін JavaScript є його здатність маніпулювати об'єктами на високому рівні. Об'єкти в JavaScript є колекціями властивостей, що містять пари "ключ-значення". Це дозволяє розробникам створювати складні структури даних і взаємодіяти з ними за допомогою методів, властивих об'єктам. JavaScript також підтримує прототипне спадкування, що дозволяє об'єктам успадковувати властивості та методи від інших об'єктів.

У JavaScript є широкий спектр вбудованих об'єктів і функцій, які дозволяють розробникам працювати з рядками, датами, масивами, математичними операціями, регулярними виразами і багато чим іншим. Крім

того, в JavaScript існують різноманітні стандартні бібліотеки і сторонні модулі, які додають додаткові функціональні можливості та спрощують розробку додатків.

JavaScript також підтримує асинхронне програмування, що дозволяє виконувати операції, які можуть зайняти багато часу, без блокування виконання інших операцій. Це досягається за допомогою зворотніх викликів (callbacks), промісів (promises) і асинхронних функцій (async/await).

У сучасному розробці веб-додатків, JavaScript широко використовується разом зі специфікаціями HTML5 та CSS3 для створення багатофункціональних інтерфейсів, а також з фреймворками і бібліотеками, такими як React, Angular і Vue.js, що дозволяють створювати великі, масштабовані додатки з багатою функціональністю.

Узагалі, JavaScript є потужним і гнучким інструментом для розробки веб-додатків, який дозволяє реалізувати різноманітні функціональність, працювати з динамічним контентом, взаємодіяти з користувачем і розробляти як клієнтську, так і серверну частини додатка.

2.3 Огляд мови програмування TypeScript

TypeScript [3] – це мова програмування, яка є розширенням мови JavaScript, що надає статичний типізацію та додаткові функції для покращення розробки ПЗ. Вона була розроблена компанією Microsoft і набула значної популярності серед розробників у світі JavaScript та веб-розробки.

Основні особливості TypeScript:

– Статична типізація: TypeScript додає статичну типізацію до JavaScript, що дозволяє визначати типи змінних, параметрів функцій, повернутих значень та інших елементів коду. Це дозволяє виявляти помилки під

час компіляції, перед тим як програма виконається, та полегшує відлагодження та підтримку коду.

– Об'єктно-орієнтоване програмування: TypeScript підтримує об'єктно-орієнтовану парадигму, включаючи класи, наслідування, інтерфейси, абстрактні класи та інші концепції. Це дозволяє створювати більш організований та перевикористовуваний код.

– Розширені можливості для типізації: TypeScript дозволяє визначати власні типи, створювати аліаси типів, перерахування та інші конструкції для точного визначення структури даних. Також вона підтримує поліморфізм, узагальнення (generics) та інші механізми для роботи з типами.

– Екосистема JavaScript: TypeScript базується на JavaScript і повністю сумісна з ним. Це означає, що весь наявний код на JavaScript може бути використаний безпосередньо в TypeScript-проектах. Крім того, TypeScript використовує сучасні стандарти JavaScript, такі як ES6 та ES7, та надає функціональні можливості, такі як стрілкові функції та розгалуження типів.

– Інструментарій розробки: TypeScript надає потужний набір інструментів для розробки, включаючи компілятор TypeScript, який перетворює TypeScript-код у JavaScript, засоби автодоповнення коду, перевірку типів під час розробки, інтеграцію з редакторами коду та інші інструменти, що спрощують процес розробки та полегшують підтримку коду.

– Підтримка веб-розробки та Node.js: TypeScript добре підходить для розробки веб-додатків, а також серверних додатків на Node.js. Вона підтримує роботу з фреймворками та бібліотеками, такими як React, Angular, Express та багатьма іншими, забезпечуючи більшу безпеку та продуктивність у процесі розробки.

Узагальнюючи, TypeScript - це розширення мови JavaScript, яке надає статичну типізацію та додаткові можливості для розробки ПЗ. Вона полегшує відлагодження та підтримку коду, забезпечує більшу безпеку та організованість, підтримує об'єктно-орієнтовану парадигму та надає потужний інструментарій для розробки веб- та серверних додатків.

2.4 Огляд бібліотеки React

React [4] – це відкрита JavaScript бібліотека для розробки користувацького інтерфейсу. Вона була розроблена компанією Facebook і широко використовується для створення веб-додатків і мобільних додатків. React дозволяє розробникам створювати ефективні, масштабовані та перевикористовувані інтерфейси, що забезпечують взаємодію користувача з додатком.

Центральним поняттям у React є компоненти. Компоненти - це відокремлені, самостійні блоки коду, які можуть містити HTML-подібний розмітку, логіку та стилі. Кожен компонент може мати свій власний стан (state), який може змінюватися з часом. Зміна стану компонента призводить до оновлення відображення на сторінці.

Одним з основних принципів React є "одноконтрольний" (one-way data flow) потік даних. Це означає, що дані передаються з батьківських компонентів до дочірніх через властивості (props). Дочірні компоненти не можуть безпосередньо змінювати дані, які отримують, але можуть сповіщати про зміни, викликаючи callback-функції, передані їм як властивості.

React також надає можливість використовувати JSX (JavaScript XML) - розширення синтаксису JavaScript, яке дозволяє включати HTML-подібний код

безпосередньо в JavaScript. Це спрощує роботу зі структурою компонентів і полегшує їх розуміння.

Однією з важливих особливостей React є віртуальний DOM (Virtual DOM). Він є представленням структури реального DOM у вигляді об'єкту JavaScript. Замість безпосередньо взаємодіяти з реальним DOM, React працює з віртуальним DOM, що дозволяє зменшити кількість операцій оновлення сторінки і забезпечує більш ефективну роботу додатка.

React також має широку екосистему допоміжних бібліотек, таких як React Router для навігації, Redux для керування станом додатку, Axios для взаємодії з сервером і багато інших. Ці бібліотеки поєднуються з React, щоб забезпечити більше функціональності і спростити розробку додатків.

У підсумку, React – потужна бібліотека для розробки веб-додатків, яка дозволяє створювати компонентну архітектуру, ефективно працювати зі станом додатку і ефективно взаємодіяти з віртуальним DOM. Завдяки своїй популярності і активній спільноті розробників, React залишається однією з найпопулярніших інструментів для фронтенд-розробки.

2.5 Огляд фреймворку React Native

React Native [5] – це відкрита платформа для розробки мобільних додатків, яка базується на бібліотеці React. За допомогою React Native розробники можуть створювати переносні мобільні додатки, які працюють як на iOS, так і на Android, використовуючи один код на JavaScript. За рахунок цього, React Native спрощує процес розробки мобільних додатків і дозволяє ефективно використовувати ресурси.

Основним принципом React Native є "один раз напиши, багато разів використовуй" (write once, use everywhere). Замість того, щоб розробляти окремі

версії додатків для кожної платформи (iOS і Android), React Native дозволяє використовувати спільний код на JavaScript для обох платформ. Він використовує нативні компоненти і API для кожної платформи, що забезпечує високу продуктивність і нативний вигляд додатків.

React Native забезпечує доступ до великої кількості готових компонентів, які включаються в стандартну бібліотеку React Native. Ці компоненти дозволяють розробникам швидко створювати інтерфейси додатків, включаючи кнопки, тексти, зображення, списки, навігацію та багато іншого. Крім того, React Native також підтримує сторонні компоненти, які можна використовувати для додаткової функціональності.

React Native також надає можливість оновлювати додатки безпосередньо на мобільних пристроях (over-the-air updates), що дозволяє розробникам внести зміни і виправити помилки без необхідності оновлювати додаток через магазини додатків.

Одним з великих переваг React Native є його взаємодія з нативним кодом. Розробники можуть використовувати нативні модулі для реалізації складної функціональності, яка не підтримується стандартними компонентами React Native. Також можна інтегрувати існуючі мобільні бібліотеки на Objective-C, Swift або Java.

У підсумку, React Native є потужною платформою для розробки переносних мобільних додатків. Вона дозволяє використовувати один код для розробки додатків на різних платформах, забезпечуючи високу продуктивність та нативний вигляд. З великим набором готових компонентів, підтримкою нативного коду і можливістю оновлення додатків, React Native стає вигідним вибором для швидкої та ефективної розробки мобільних додатків.

2.6 Огляд Google Maps

Google Maps [6] – це одна з найпопулярніших та найважливіших мапових платформ у світі, що надає розширені можливості для відображення карт, отримання маршрутів, міток, геолокації та багато іншого. У контексті React Native використання Google Maps дозволяє розробникам створювати мобільні додатки з географічними функціями на базі платформи Android та iOS.

Для роботи з Google Maps у React Native існує відкритий пакет React Native Maps, який надає зручний інтерфейс для інтеграції з Google Maps API. Цей пакет забезпечує доступ до всіх основних функціональних можливостей Google Maps, таких як відображення карт, робота з маркерами, додавання полігонів та поліліній, отримання маршрутів та багато іншого.

Щоб почати використовувати Google Maps у React Native, необхідно спочатку встановити та налаштувати пакет React Native Maps. Це включає додавання залежностей до проекту та налаштування конфігурації Android та iOS.

Після встановлення пакету React Native Maps, можна почати використовувати його компоненти для відображення карт та інших географічних елементів. Наприклад, компонент MapView дозволяє відображати карту на екрані додатку. Можна налаштовувати параметри відображення, такі як центр карти, зум, максимальний та мінімальний зум, тип карти та багато іншого.

Крім відображення карти, React Native Maps також надає можливості додавання маркерів на карту. Компонент Marker дозволяє створювати та налаштовувати маркери з власними значками, текстом, додатковими даними та обробниками подій. Маркери можуть бути використані для позначення

конкретних місць на карті, таких як місцезнаходження користувача, місця інтересу або будь-які інші точки.

Крім маркерів, React Native Maps дозволяє додавати полігони та полілінії на карту. Полігони складаються з набору координат і можуть бути використані для виділення областей на карті. Полілінії також складаються з набору координат і можуть бути використані для відображення лінійних об'єктів, таких як маршрути або кордони.

Крім базового відображення карт, React Native Maps надає можливості отримання маршрутів за допомогою Google Maps Directions API. За допомогою компонента DirectionsRenderer можна відображати маршрути на карті, передаючи початкову та кінцеву точки, а також додаткові параметри маршрутизації, такі як режим переміщення, обмеження шляхопроводів та інші.

У контексті React Native також можна використовувати різноманітні додаткові бібліотеки та розширення, які спрощують використання Google Maps. Наприклад, є бібліотеки, які додають можливості для роботи з місцями, геозонами, розширеною маршрутизацією та іншими функціями Google Maps API.

Узагальнюючи, використання Google Maps у React Native дозволяє розробникам створювати потужні мобільні додатки з географічними функціями. Завдяки пакету React Native Maps та інших додаткових бібліотек, розробники можуть легко взаємодіяти з Google Maps API, відображати карти, створювати маркери, полігони, полілінії, отримувати маршрути та багато іншого. Це робить Google Maps універсальним інструментом для розробки мобільних додатків з маповою функціональністю у React Native.

2.7 Огляд OpenWeather API

OpenWeather API [7] – це публічний API (інтерфейс програмування додатків), який надає доступ до широкого спектру метеорологічних даних і прогнозів погоди. Це один з найпопулярніших сервісів погоди, який дозволяє розробникам інтегрувати погодну інформацію в свої додатки, веб-сайти або сервіси.

OpenWeather API пропонує різноманітні методи для отримання погодних даних. Основні можливості API включають:

– Поточну погоду: API дозволяє отримати дані про поточну погоду для певного місцезнаходження. Це може включати такі дані, як температура, вологість, швидкість вітру, тиск, хмарність і багато іншого.

– Прогноз погоди: API надає доступ до прогнозів погоди на кілька днів вперед. Розробники можуть отримати прогнози на різні часові проміжки, включаючи деталізовану інформацію про погоду на кожен з них.

– Геолокація: API надає можливість отримати погодні дані для певного місцезнаходження за допомогою географічних координат (широта і довгота) або назви міста. Це дозволяє розробникам отримувати погоду для будь-якої області світу.

– Додаткові дані: API також надає доступ до додаткових метеорологічних даних, таких як індекси UV-випромінювання, індекси якості повітря, прогнози дощу, снігопаду, температурний графік і багато іншого.

Для використання OpenWeather API розробникам потрібно отримати API-ключ, який є унікальним ідентифікатором, що дозволяє отримувати доступ до сервісу. Цей ключ передається в запитах до API для аутентифікації та обмеження використання.

API OpenWeather має різні типи планів та обмеження використання, включаючи безкоштовний план з обмеженими можливостями та платні плани з більш високими лімітами та розширеними функціями.

Для отримання даних з OpenWeather API розробники можуть використовувати HTTP-запити, такі як запити GET або POST, і відповіді повертаються у форматі JSON або XML, залежно від вибору розробника.

Важливо зазначити, що OpenWeather API підлягає умовам використання та ліцензійним обмеженням. Розробники повинні ознайомитися з умовами використання та дотримуватися їх при інтеграції погодних даних в свої додатки.

2.8 Огляд Redux та Redux Toolkit

Redux [8] – це популярна бібліотека для керування станом додатків у JavaScript, яка широко використовується в контексті React [9] та React Native. Redux дозволяє ефективно організовувати та управляти станом додатку, що дозволяє зменшити складність, покращити швидкодію та зручність розробки.

Redux працює на принципі однорядкової деревоподібної структури стану, яку називають "store". Весь стан додатку зберігається у цьому store як не змінний об'єкт. Компоненти можуть отримувати доступ до стану за допомогою функції mapStateToProps, а також змінювати стан за допомогою функції mapDispatchToProps. Оновлення стану відбуваються за допомогою спеціальних функцій, які називаються "actions", та "reducers", які визначають, які зміни потрібно внести у стан додатку.

Однак, використання Redux в базовому вигляді може бути трохи незручним і вимагати великої кількості коду для налаштування та управління станом. Щоб спростити цей процес, було створено Redux Toolkit - надбудову

над Redux, яка надає зручні інструменти та патерни для розробки додатків на основі Redux.

Redux Toolkit надає такі основні компоненти:

– createSlice: Це функція, яка допомагає організувати код, пов'язаний зі станом, у зручний спосіб. Вона автоматично генерує відповідні actions та reducers, що спрощує розробку.

– configureStore: Ця функція використовується для створення Redux store з підтримкою різних додаткових функцій, таких як middleware та DevTools Extension. Вона дозволяє налаштувати store з мінімальним зусиллям.

– createAsyncThunk: Цей інструмент допомага

– є обробляти асинхронні запити та взаємодіяти з сервером, спрощуючи створення відповідних actions та reducers.

– createEntityAdapter: Цей компонент допомагає управляти колекціями даних, такими як списки або таблиці, надаючи зручний спосіб додавання, видалення та оновлення об'єктів.

Однією з найбільших переваг Redux Toolkit є зменшення кількості написаного власного коду. Він надає шаблони, які автоматично генерують багато необхідного коду за вас, забезпечуючи при цьому чистоту та ефективність коду.

Узагальнюючи, Redux Toolkit - це розширення для Redux, яке надає зручні інструменти для розробки додатків на основі Redux. Використання Redux Toolkit допомагає знизити складність розробки та полегшує управління станом, забезпечуючи при цьому чистоту та ефективність коду. Він надає шаблони та утиліти, які спрощують створення actions, reducers, middleware та store, що забезпечує більш ефективний процес розробки у контексті React та React Native додатків.

Висновки до розділу 3

Використання стеку технологій, який включає фреймворк React Native, мову програмування TypeScript та менеджер стану Redux, є обґрунтованим з кількох причин:

– Кросплатформеність: React Native дозволяє розробляти мобільні застосунки для платформ iOS та Android з використанням одного кодової бази. Це ефективно зменшує затрати на розробку та підтримку додатків для кількох платформ.

– Швидкість розробки: React Native забезпечує швидку розробку завдяки можливості використання компонентів з многоразовим використанням, гарячої перезавантаження компонентів та інструментам для швидкого розгортання змін.

– TypeScript: Використання TypeScript надає переваги статичного типізації, що дозволяє виявляти помилки на етапі розробки та поліпшує підтримку коду. Це допомагає зменшити кількість помилок, покращити продуктивність розробників та зробити код більш надійним.

– Redux: Redux є потужним менеджером стану, який дозволяє ефективно керувати станом застосунку та його змінами. Використання Redux спрощує управління станом застосунку, полегшує реактивність і взаємодію компонентів, а також дозволяє легко відновлювати стан після перезавантаження додатка.

– Велике співтовариство: React Native, TypeScript та Redux є популярними технологіями з великою кількістю активних розробників та співтовариств. Це означає, що є багато доступних ресурсів, бібліотек, допоміжних інструментів та знань, які можна використовувати для підтримки та розширення проекту.

Узагальнюючи, використання стеку технологій, що включає React Native, TypeScript та Redux, дозволяє швидко розробляти кросплатформні мобільні застосунки, забезпечуючи високу продуктивність, надійність та зручне управління станом. Такий стек технологій є популярним серед розробників і має потужну спільноту, що підтримує розвиток та поширення цих технологій.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ВІДОБРАЖЕННЯ ПОГОДНИХ УМОВ

4.1 Методологія розробки дизайну застосунку

Дизайн інтерфейсу мобільного застосунку на React Native залежить від багатьох факторів, включаючи вимоги проекту, цільову аудиторію, брендовий стиль і особливості платформи. Однак, існують загальні принципи та практики [10][11], які можуть бути застосовані при розробці дизайну інтерфейсу на React Native.

Нижче розглянуто деякі з них:

– Консистентність: Важливо забезпечити консистентність дизайну в усьому застосунку. Використовуйте однаковий стиль, кольори, шрифти і компоненти на всіх сторінках та екранах.

– Простота і зручність використання: Дизайнуйте інтерфейс таким чином, щоб він був простим і зрозумілим для користувача. Мінімізуйте кількість кроків і навігаційних елементів для досягнення мети користувача.

– Відповідність платформі: Враховуйте особливості платформи (iOS або Android) та рекомендації щодо дизайну, щоб забезпечити нативний вигляд і взаємодію з користувачем.

– Використання компонентів: React Native надає багато готових компонентів, таких як кнопки, тексти, зображення, списки та інші. Використовуйте їх для побудови інтерфейсу і спрощення розробки.

– Розташування елементів: Розташуйте елементи інтерфейсу таким чином, щоб було забезпечено зручну та логічну взаємодію користувача. Розгляньте розташування елементів за допомогою контейнерів, грідів або флекс-боксів для досягнення бажаного макету.

– Використання анімацій: Використовуйте анімації для покращення візуального враження та взаємодії з користувачем. React Native надає можливості для створення різних типів анімацій, таких як переміщення, зміна розміру та прозорості елементів.

– Брендовий стиль: Дотримуйтесь брендового стилю вашої компанії або проекту. Використовуйте відповідні кольори, шрифти та елементи дизайну, щоб відображати унікальність вашого бренду.

– Тестування і зворотній зв'язок: Перевірте дизайн інтерфейсу за допомогою тестування користувачами або отримайте зворотній зв'язок від користувачів, щоб покращити його ефективність та зрозумілість.

Загалом, розробка дизайну інтерфейсу на React Native вимагає уважного планування, дотримання дизайнерських принципів та адаптації до потреб вашого проекту та цільової аудиторії.

4.2 Дизайн застосунку

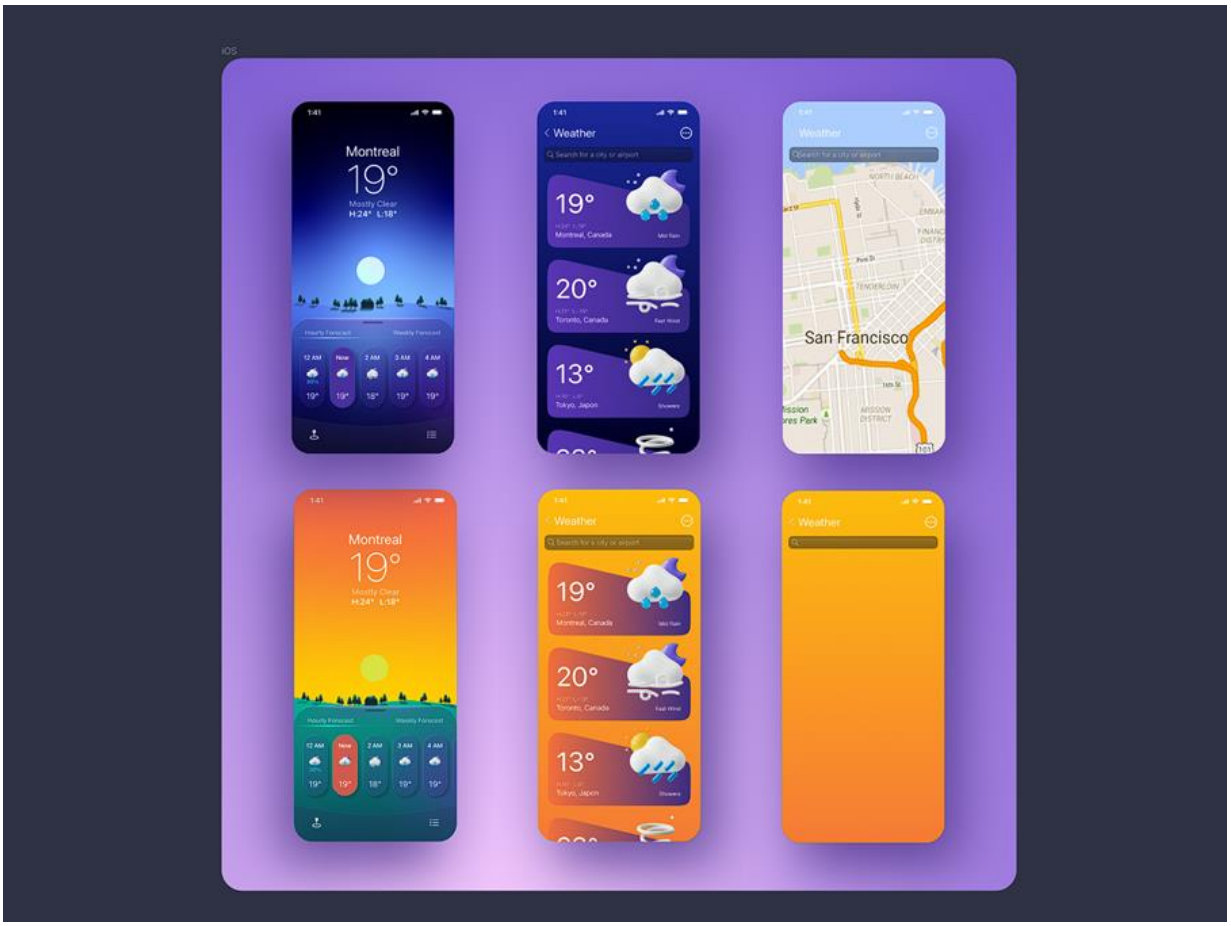

Рисунок 4.1 – Початковий дизайн застосунку

Під час розробки дизайну (рис. 4.1) застосунку було враховано попередньо наведені рекомендації щодо проектування інтерфейсу, особливо простоту та зручність використання.

Усього планувалось розробити 4 екрани:

– Головний екран з відображенням поточних погодних умов та модальним вікном з коротким описом погоди на тиждень.

– Екран з переліком найбільших населених пунктів, коротким описом погоди для них.

– Екран з мапою для вибору локації власноруч.

– Екран обирання мови застосунку.

4.3 Головний екран застосунку

Розглянемо головний екран застосунку (рис. 4.2).

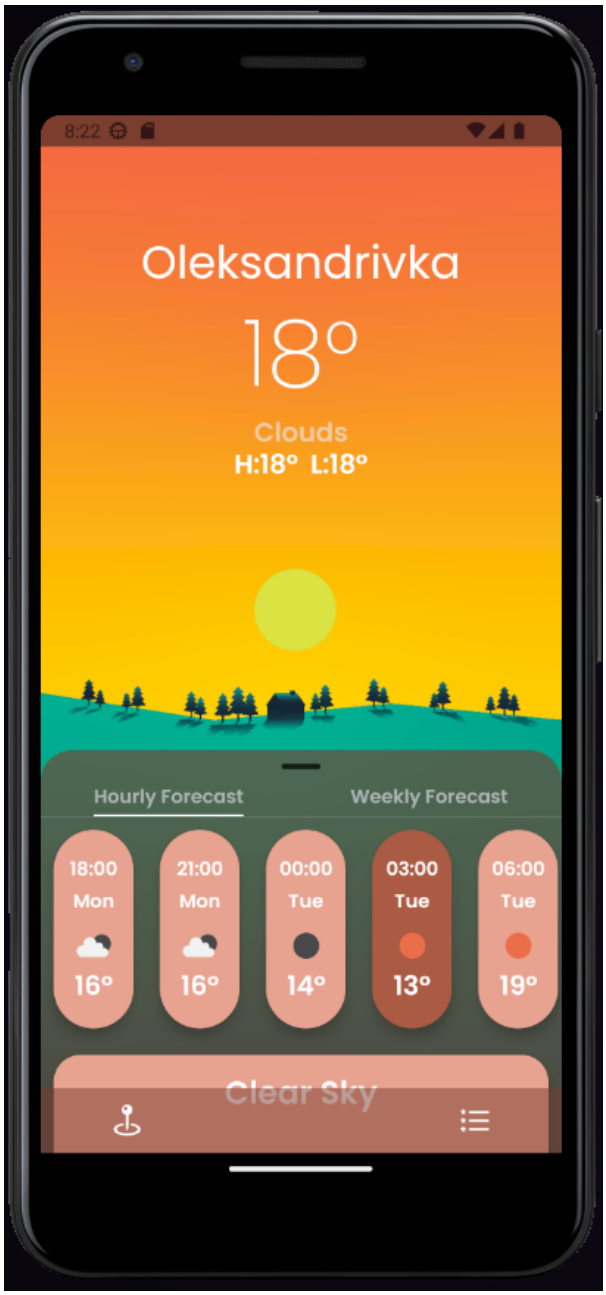

Рисунок 4.2 – Головний екран застосунку

Лістинг коду 4.1 – Головний екран застосунку

```
const HomeScreen = () => {
    const theme = useThemeContext();
    const styles = useThemedStyles(style);

    const bg = theme.dark
        ? require('./../../assets/img/nightbg.png')
        : require('./../../assets/img/daybg.png');

    const forecast = useSelector(
        (state: RootState) => state.weather.forecastNow,
    );
    const isLoading = useSelector(
        (state: RootState) => state.weather.isLoading,
    );

    const dispatch = useDispatch();

    useEffect(() => {
        dispatch(getForecastNow());
        dispatch(getForecastWeek());
    }, [dispatch]);

    return (
        <View style={styles.container}>
            <ImageBackground style={styles.bgImg} source={bg}>
                {isLoading ? (
                    <Suspense />
                ) : (
                    <>
                        <View style={styles.mainInfo}>
                            <Text
                                style={[styles.mainInfoFont,
styles.cityText]}
                            >
                                {forecast?.name}
                            </Text>
                            <Text
                                style={[
                                    styles.mainInfoFont,
                                    styles.temperatureText,
                                ]}
                            >
                                {forecast?.main.temp
```

```
                                    ? Math.round(forecast?.main.temp) +
'°'
                                    : '-'}
                            </Text>
                            <Text style={[styles.mainInfoFont,
styles.skyText]}>
                                {forecast?.weather[0].main}
                            </Text>
                            <Text
                                style={[styles.mainInfoFont,
styles.loHiText]}
                            >
                                {'H:' +
                                    Math.round(forecast?.main.temp_max ??
0) +
                                    '°'}
                                {'  L:' +
                                    Math.round(forecast?.main.temp_min ??
0) +
                                    '°'}
                            </Text>
                        </View>

                        <BottomSheetComponent>
                            <Weather />
                        </BottomSheetComponent>
                    </>
                )}
            </ImageBackground>
        </View>
    );
};
```

На екрані бачимо наступні елементи:

– Назва населеного пункту (локації).

– Температура повітря.

– Стислий опис погоди.

– Найвищий та найнижчий температурні показники.
– Частково приховане модальне вікно.

Код реалізації екрану наведено у ліст. 4.1.

Також розглянемо модальне вікно з інформацією про стан погодних умов (рис. 4.3).

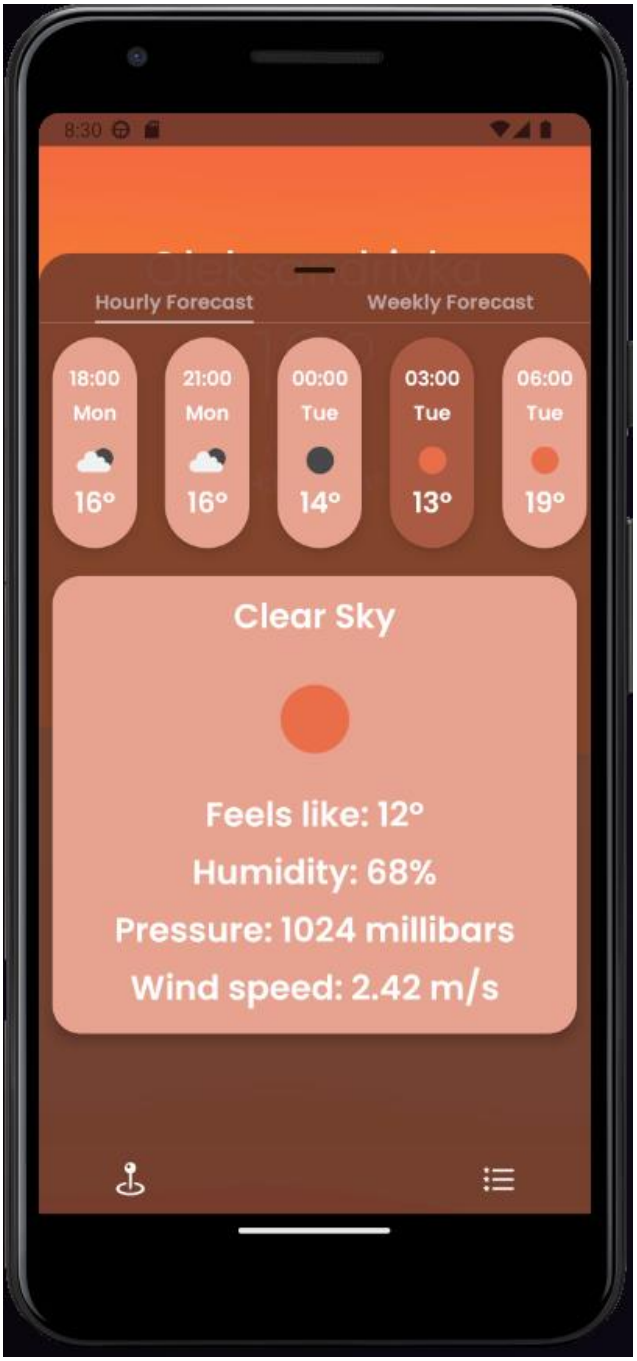

Рисунок 4.3 – Відкрите модальне вікно

У модалці присутні дві вкладки:

– Hourly Forecast – тут відображається список з прогнозом погоди на кожні три години на два тижня з можливістю вибору потрібного проміжку часу та відображенням коротких відомостей про обраний день.

– Weekly Forecast – тут відображається список з прогнозом погоди на кожен день впродовж двох тижнів з можливістю вибору потрібного дня та відображенням коротких відомостей про нього.

У ліст. 4.2-4.4 наведено код реалізації обгортки для модального вікна, анімований компонент для заднього фону модального вікна та наповнення відповідно.

Лістинг коду 4.2 – Обгортка модального вікна

```
import { useMemo, useRef } from 'react';
import BottomSheet from '@gorhom/bottom-sheet';
import CustomBackground from './Background/Background';

interface Props {
    children: React.ReactNode;
}

const BottomSheetComponent = ({ children }: Props) => {
    const bottomSheetRef = useRef<BottomSheet>(null);
    const snapPoints = useMemo(() => ['40%', '90%'], []);

    return (
        <BottomSheet
            ref={bottomSheetRef}
            snapPoints={snapPoints}
            backgroundComponent={CustomBackground}
            backgroundStyle={{ borderRadius: 24 }}
        >
            {children}
        </BottomSheet>
    );
};

export default BottomSheetComponent;
```

Лістинг коду 4.3 – Анімований компонент заднього фону

```
import React, { useMemo } from 'react';
import { BottomSheetBackgroundProps } from '@gorhom/bottom-sheet';
import Animated, {
    useAnimatedStyle,
    interpolateColor,
} from 'react-native-reanimated';
import useThemeContext from '../../../hooks/useThemeContext';

const hexTransparencyValues = ['A0', 'F0'];

const Background: React.FC<BottomSheetBackgroundProps> = ({
    style,
    animatedIndex,
}) => {
    const theme = useThemeContext();

    const containerAnimatedStyle = useAnimatedStyle(() => ({
        backgroundColor: interpolateColor(
            animatedIndex.value,
            [0, 1],
            [
                `${theme.colors.bgdark}${hexTransparencyValues[0]}`,
                `${theme.colors.bgdark}${hexTransparencyValues[1]}`,
            ],
        ),
    }));
    const containerStyle = useMemo(
        () => [style, containerAnimatedStyle],
        [style, containerAnimatedStyle],
    );

    return <Animated.View pointerEvents="none" style={containerStyle} />;
};

export default Background;
```

Лістинг коду 4.4 – Наповнення модального вікна

```
const Weather = () => {
    const styles = useThemedStyles(style);
    const tabBarHeight = useBottomTabBarHeight();
    const screenHeight = Dimensions.get('window').height;
```

```
const forecast = useSelector(
    (state: RootState) => state.weather.forecastsWeek,
);

const [selectedTab, setSelectedTab] = useState<Tab>(tabs[0]);
const [selectedForecastDay, setSelectedForecastDay] =
    useState<ForecastWeek>(forecast[0]);
const [selectedIndex, setSelectedIndex] = useState<number>(0);

const handleSelectTab = (tab: Tab) => {
    setSelectedTab(tab);
};

const handleSelectForecastDay = (index: number) => {
    setSelectedForecastDay(forecast[index] ?? forecast[0]);
    setSelectedIndex(index ?? 0);
};

const weekly = forecast.filter(
    item => new Date(item.dt_txt).getHours() === 12,
);

weekly.some(
    item => new Date(item.dt_txt).getDate() === new Date().getDate(),
) || weekly.unshift(forecast[0]);

const showedForecast = selectedForecastDay || forecast[0];

return (
    <BottomSheetScrollView
        showsVerticalScrollIndicator={false}
        style={[
            styles.container,
            {
                height: screenHeight - 2 * tabBarHeight,
            },
        ]}
    >
        <WeatherHeader
            onTabSelect={handleSelectTab}
            selectedTab={selectedTab}
        />
        <BottomSheetFlatList
            horizontal
            showsHorizontalScrollIndicator={false}
```

```
                data={selectedTab === 'hourly' ? forecast : weekly}
                keyExtractor={(item, index) => (item.dt +
index).toString()}
                renderItem={({ item, index }) => {
                    const hours = new Date(item.dt_txt).getHours();
                    const isSelected = index === selectedIndex;
                    return (
                        <WeatherItem
                            forecast={item}
                            day={weekday[new Date(item.dt_txt).getDay() ||
0]}
                            date={formatDate(item.dt_txt)}
                            time={hours}
                            isSelected={isSelected}
                            showDate={selectedTab === 'weekly'}
                            onSelectForecastDay={() =>
                                handleSelectForecastDay(index)
                            }
                        />
                    );
                }}
            />
            <View style={styles.detailedContainer}>
                <Text
                    style={[styles.descText, { textTransform: 'capitalize'
}]}
                >
                    {showedForecast?.weather[0].description}
                </Text>
                <Image
                    style={styles.descImage}
                    source={{
                        uri:
`https://openweathermap.org/img/wn/${showedForecast?.weather[0].icon}@4x.p
ng`,
                    }}
                />
                <Text style={styles.descText}>
                    Feels like:
{Math.round(showedForecast?.main.feels_like)}°
                </Text>
                <Text style={styles.descText}>
                    Humidity: {showedForecast?.main.humidity}%
                </Text>
                <Text style={styles.descText}>
```

```
                    Pressure: {showedForecast?.main.pressure} millibars
                </Text>
                <Text style={[styles.descText]}>
                    Wind speed: {showedForecast?.wind.speed} m/s
                </Text>
            </View>
        </BottomSheetScrollView>
    );
};
```

4.4 Мапа для вибору локації

На рис. 4.4 показано мапу вибору локації для відображення погодних умов, а також дві кнопки: для очищення збереженої локації (і автоматичного її визначення) та для збереження поточної локації на мапі.

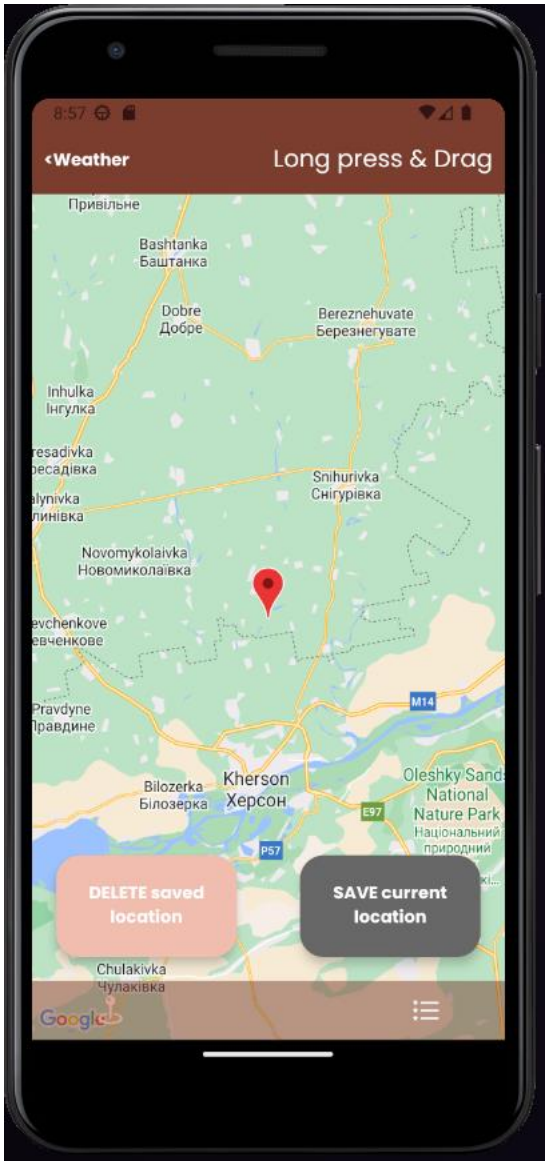

Рисунок 4.4 – Екран з мапою

Лістинг 4.5 – Екран мапи

```
const MapScreen = () => {
    const coords = useSelector((state: RootState) =>
state.location.coords);
    const dispatch = useDispatch();

    const [newCoords, setNewCoords] = useState<LatLng | null>(null);
    const tabBarHeight = useBottomTabBarHeight();

    const handleSaveCoords = () => {
```

```
        newCoords && dispatch(setLocationCoords(newCoords));
    };

    const handleClearSavedCoords = () => {
        Alert.alert(
            'Clear saved location?',
            'If you clear your saved location, you will need to select it
again or allow the application to set it automatically',
            [
                {
                    text: 'Cancel',
                    style: 'cancel',
                },
                {
                    text: 'I understand it. Clear the location',
                    onPress: () => dispatch(clearLocationCoords()),
                    style: 'destructive',
                },
            ],
        );
    };

    return (
        <View style={styles.container}>
            <MapView
                style={styles.mapStyle}
                initialRegion={{
                    latitude: coords.latitude ?? 37.78825,
                    longitude: coords.longitude ?? -122.4324,
                    latitudeDelta: 1,
                    longitudeDelta: 1,
                }}
                // customMapStyle={mapStyle}
            >
                <Marker
                    draggable
                    coordinate={{
                        latitude: coords.latitude ?? 37.78825,
                        longitude: coords.longitude ?? -122.4324,
                    }}
                    onDragEnd={e =>
setNewCoords(e.nativeEvent.coordinate)}
                    title={'Current location for the weathercast'}
                    description={
```

```
                        'Long press and drag the marker to select a new
location'
                    }
                />
            </MapView>
            <View
                style={[
                    styles.button,
                    { right: 10, bottom: tabBarHeight + 10 },
                ]}
            >
                <Button
                    title="SAVE current location"
                    onPress={handleSaveCoords}
                    disabled={!newCoords}
                />
            </View>
            <View
                style={[styles.button, { left: 10, bottom: tabBarHeight +
10 }]}
            >
                <Button
                    title="DELETE saved location"
                    onPress={handleClearSavedCoords}
                />
            </View>
        </View>
    );
};
```

У ліст. 4.5 наведено код реалізації мапи для вибору локації відображення погодних умов.

4.5 Налаштування зв’язку з API

У ліст. 4.6-4.7 наведено код для отримання даних та відправлення запитів до API (redux-saga middleware).

Лістинг 4.6 – «Саги» для отримання даних геолокації

```
export function* locationUpdateCoordsSaga() {
    try {
        yield put(setIsLoading());
```

```
        // @ts-ignore
        const isManuallySet = yield AsyncStorage.getItem('isManuallySet');
        // @ts-ignore
        const savedCoords = JSON.parse(yield
AsyncStorage.getItem('coords'));
        if (isManuallySet === 'true') {
            yield put(updateLocation(savedCoords));
        } else {
            // @ts-ignore
            const result = yield GetLocation.getCurrentPosition({
                enableHighAccuracy: true,
                timeout: 60000,
            });
            if (!savedCoords) {
                yield AsyncStorage.setItem('coords',
JSON.stringify(result));
                yield put(updateLocation(result));
            }

            const savedToNewDeltaLatitude = Math.abs(
                +result.latitude - +savedCoords.latitude,
            );
            const savedToNewDeltaLongitude = Math.abs(
                +result.longitude - +savedCoords.longitude,
            );
            if (
                savedToNewDeltaLatitude > 0.3 &&
                savedToNewDeltaLongitude > 0.35
            ) {
                yield AsyncStorage.setItem('coords',
JSON.stringify(result));
                yield put(updateLocation(result));
            }
        }
        yield put(stopLoading());
    } catch (error) {
        yield put(setError(error));
        console.error(error);
    }
}

export function* locationUpdateUpdateCoordsWatcherSaga() {
    yield takeEvery(mapActions.GET_LOCATION_COORDS,
locationUpdateCoordsSaga);
}
```

```
export function* locationSaveCoordsSaga(action: {
    type: string;
    payload: {
        latitude: number;
        longitude: number;
    };
}) {
    try {
        yield put(setIsLoading());
        yield AsyncStorage.setItem('coords',
JSON.stringify(action.payload));
        yield AsyncStorage.setItem('isManuallySet', 'true');
        yield put(updateLocation(action.payload));
        yield put(stopLoading());
    } catch (error) {
        yield put(setError(error));
        console.error(error);
    }
}

export function* locationSaveCoordsWatcherSaga() {
    yield takeEvery(mapActions.SET_LOCATION_COORDS,
locationSaveCoordsSaga);
}

export function* locationClearSavedCoordsSaga() {
    try {
        yield put(setIsLoading());
        yield AsyncStorage.removeItem('coords');
        yield AsyncStorage.removeItem('isManuallySet');
        yield put(clearLocation());
        yield put(stopLoading());
    } catch (error) {
        yield put(setError(error));
        console.error(error);
    }
}

export function* locationClearSavedCoordsWatcherSaga() {
    yield takeEvery(
        mapActions.CLEAR_LOCATION_COORDS,
        locationClearSavedCoordsSaga,
    );
}
```

Лістинг 4.7 – «Саги» для отримання даних з OpenWeatherAPI

```
const API = 'https://api.openweathermap.org/data/2.5';
const city = 'Kyiv';

export function* weatherUpdateDaySaga() {
    try {
        yield put(setIsLoading());
        // @ts-ignore
        const stateCoords = yield select(
            (state: RootState) => state.location.coords,
        );
        const coords = JSON.parse(yield AsyncStorage.getItem('coords'));
        let api =
`${API}/weather?q=${city}&units=metric&appid=${Config.API_KEY}`;
        if (stateCoords && stateCoords.latitude && stateCoords.longitude)
{
            api =
`${API}/weather?lat=${stateCoords.latitude}&lon=${stateCoords.longitude}&u
nits=metric&appid=${Config.API_KEY}`;
        } else if (coords && coords.latitude && coords.longitude) {
            api =
`${API}/weather?lat=${coords.latitude}&lon=${coords.longitude}&units=metri
c&appid=${Config.API_KEY}`;
        }
        // @ts-ignore
        let result = yield call(() => fetch(api, {}).then(res =>
res.json()));
        yield put(updateForecastDay(result));
    } catch (error) {
        yield put(setError(error));
        console.error(error);
    }
}

export function* weathersUpdateWeekWatcherSaga() {
    yield takeEvery(weatherActions.GET_FORECAST_WEEK,
weatherUpdateDaySaga);
}

export function* weatherUpdateWeekSaga() {
    try {
        yield put(setIsLoading());
        // @ts-ignore
        const stateCoords = yield select(
```

```
            (state: RootState) => state.location.coords,
        );
        const coords = JSON.parse(yield AsyncStorage.getItem('coords'));
        let api =
`${API}/forecast?q=${city}&units=metric&appid=${Config.API_KEY}`;
        if (stateCoords && stateCoords.latitude && stateCoords.longitude)
{
            api =
`${API}/forecast?lat=${stateCoords.latitude}&lon=${stateCoords.longitude}&
units=metric&appid=${Config.API_KEY}`;
        } else if (coords && coords.latitude && coords.longitude) {
            api =
`${API}/forecast?lat=${coords.latitude}&lon=${coords.longitude}&units=metr
ic&appid=${Config.API_KEY}`;
        }
        // @ts-ignore
        let result = yield call(() => fetch(api, {}).then(res =>
res.json()));

        yield put(updateForecastWeekArray(result));
    } catch (error) {
        yield put(setError(error));
        console.error(error);
    }
}
```

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра завершено створення кросплатформного застосунку відображення погодних умов.

Для досягнення визначеної мети було виконано поставлені завдання:

– здійснено аналіз предметної області та застосунків-аналогів;

– визначено основні функції застосунку, що проєктується;

– створено специфікацію вимог до ПЗ;

– створено діаграму сценаріїв використання, описано основні та альтернативні use cases;

– створено діаграми діяльності, послідовності, класів;

– обрано та проаналізовано стек технологій;

– реалізовано застосунок.

Завдяки проведеному аналізу аналогів було визначено актуальність та доцільність створення мобільного застосунку відображення погодних умов. За результатами дослідження аналогів сформовано вимоги, визначено основні функції та вимоги до застосунку.

Для проєктування застосунку було створено декілька видів UML-діаграм для більш структурованої та легкої для розуміння демонстрації роботи. Згідно з технічним завданням було обрано найбільш зручні інструменти та технології розробки, а саме: фреймворк React Native, мова програмування TypeScript та open-source бібіотеки для реалізації певного функціоналу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Patrick Grässle, Henriette Baumann, Philippe Baumann UML 2.0 in Action : Birmingham , 2005 . 225 p.

2. JavaScript Documentation : вебсайт. URL: developer.mozilla.org/en-US/docs/Web/javascript (дата звернення: 25.04.2023).

3. TypeScript Documentation : вебсайт. URL: www.typescriptlang.org/docs (дата звернення: 25.04.2023).

4. React Documentation : вебсайт. URL: react.dev (дата звернення: 26.04.2023).

5. React Native Documentation : вебсайт. URL: reactnative.dev (дата звернення: 27.04.2023).

6. Google Maps Services React Native repository: вебсайт. URL: github.com/hysan/react-native-google-maps-services (дата звернення: 03.05.2023).

7. OpenWeather Docs: вебсайт. URL: openweathermap.org/api (дата звернення: 03.05.2023).

8. Redux Documentation : вебсайт. URL: redux.js.org (дата звернення: 03.05.2023).

9. Alex Banks, Eva Porcello Learning React: Functional Web Development with React and Redux Second Edition : Sebastopol, 2020 . 289 p.

10. 10 Usability Heuristics for User Interface Design: вебсайт. URL: www.nngroup.com/articles/ten-usability-heuristics (дата звернення: 03.05.2023).

11. User Interface Design patterns: вебсайт. URL: ui-patterns.com/patterns (дата звернення: 03.05.2023).