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

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Танцюри Андрія Андрійовича*
	(ПІБ)
академічної групи	*121м-22-3*
	(шифр)
спеціальності	*121 Інженерія програмного забезпечення*
	(код і назва спеціальності)
освітньої програми	*Інженерія програмного забезпечення*
	(назва освітньої програми)
на тему:	*Розробка та дослідження ефективності впровадження мобільного кросплатформового додатка для ведення нотаток на Google Maps.*

_____*А.А. Танцюра*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*Проф. Бердник М.Г.*	74	добре	

Рецензент				

Нормоконтролер	*Доц. Гуліна І.Г.*			

Дніпро
2023

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

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)
_____ М. О. Алексеєв
(підпис) (прізвище, ініціали)
« » _____ 20 23 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121м-22-3* *Танцюрі Андрію Андрійовичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка та дослідження ефективності впровадження мобільного кросплатформового додатка для ведення нотаток на Google Maps.*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 09.10.2023 р. № 1227-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес ведення та управління нотатками з використанням мобільних додатків, інтегрованих з Google Maps.

Предмет досліджень – методи розробки кросплатформових мобільних додатків, зокрема, використання API Google Maps для створення інтуїтивно зрозумілих та ефективних інструментів ведення нотаток.

Мета НДР – підвищення ефективності функціонування кросплатформенних мобільних додатків для створення нотаток на Google Maps на різних мобільних операційних системах шляхом дослідження та вдосконалення процесу їх реалізації.

Методи дослідженя – для вирішення поставленої задачі використано такі методи: теорія системного аналізу, принципи функціонального та об'єктно-орієнтованого програмування.

Вихідні дані для проведення роботи – аналіз існуючих рішень у сфері мобільних додатків для ведення нотаток, основні концепції та принципи розробки кросплатформових додатків.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна – розробка методу інтеграції GPS-даних з нотатками, що дає користувачам можливість ефективно відстежувати та організовувати свої замітки за допомогою геолокації.

Практична цінність – додаток спрощує ведення нотаток з використанням географічних даних, забезпечуючи користувачам легкий доступ до інформації про різні місця та їх характеристики

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити ефективність використання обраного методу розробки кросплатформових додатків. В результаті роботи повинен бути розроблений додаток, який однаково ефективно працює на різних мобільних платформах з використанням єдиної кодової бази.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз існуючих рішень та постановка задачі	12.09.2023-30.09.2023
Дослідження методів для вирішення поставленої задачі	01.10.2023-31.10.2023
Розробка програмного забезпечення та дослідження ефективності запропонованих рішень.	01.11.2023-10.12.23

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації цього додатку полягає у зниженні витрат на розробку та підтримку через використання єдиної кодової бази для різних мобільних платформ, що забезпечує ефективне управління ресурсами.

Соціальний ефект від запровадження додатку полягає в оптимізації використання часу та ресурсів користувачів, оскільки впровадження інноваційного підходу до ведення нотаток на карті значно спрощує процес організації та відстеження важливої інформації.

Завдання видав _____ *Бердник М. Г.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Танцюра А.А.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *10.09.2023 р.*

Термін подання дипломного проекту до ЕК *20.12.2023*

РЕФЕРАТ

Пояснювальна записка: 92 стор., 23 рис., 2 додатка, 29 джерел.

Об’єкт досліджень – процес ведення та управління нотатками з використанням мобільних додатків, інтегрованих з Google Maps.

Предмет досліджень – методи розробки кросплатформових мобільних додатків, зокрема, використання API Google Maps для створення інтуїтивно зрозумілих та ефективних інструментів ведення нотаток.

Мета дослідження – підвищення ефективності функціонування кросплатформенних мобільних додатків для створення нотаток на Google Maps на різних мобільних операційних системах шляхом дослідження та вдосконалення процесу їх реалізації.

Методи дослідження – для вирішення поставленої задачі використано такі методи: теорія системного аналізу, принципи функціонального та об’єктно-орієнтованого програмування.

Наукова новизна – розробка методу інтеграції GPS-даних з нотатками, що дає користувачам можливість ефективно відстежувати та організовувати свої замітки за допомогою геолокації.

Практична цінність – додаток спрощує ведення нотаток з використанням географічних даних, забезпечуючи користувачам легкий доступ до інформації про різні місця та їх характеристики.

Область застосування – додаток охоплює особисте планування, туризм, урбаністичне планування та логістику, дозволяючи користувачам ефективно організовувати та відстежувати нотатки з використанням географічних даних

Прогнози щодо розвитку досліджень – дослідження можуть зосередитися на інтеграції адаптивних алгоритмів на основі машинного навчання для покращення взаємодії з користувачем.

Список ключових слів: кросплатформовий додаток, TypeScript, React, React Native, iOS, Android.

ABSTRACT

Explanatory Note: 92 pages, 23 figures, 2 appendices, 29 references.

Object of Research: The process of managing and maintaining notes using mobile applications integrated with Google Maps.

Subject of Research: Methods of developing cross-platform mobile applications, particularly the use of Google Maps API for creating intuitive and efficient note-taking tools.

Research Goal: Enhancing the efficiency of cross-platform mobile applications for creating notes on Google Maps across various mobile operating systems through the research and improvement of their implementation process.

Research Methods: The task was addressed using the following methods: systems analysis theory, principles of functional and object-oriented programming.

Scientific Novelty: Development of a method for integrating GPS data with notes, providing users with the ability to effectively track and organize their notes using geolocation.

Practical Value: The application simplifies note-taking using geographic data, offering users easy access to information about various places and their characteristics.

Application Scope: The application encompasses personal planning, tourism, urban planning, and logistics, enabling users to efficiently organize and track notes using geographic data.

Forecasts for Research Development: Future research may focus on integrating adaptive algorithms based on machine learning to enhance user interaction.

List of keywords: cross-platform application, TypeScript, React, React Native, iOS, Android.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ПЗ – програмне забезпечення;

ОС – операційна система;

Пін – місце на карті;

Нативний додаток – це програми, які розроблені специфічно для однієї платформи або операційної системи;

Рендеринг – це процес створення та оновлення візуального відображення компонентів в React, який відбувається при зміні стану або властивостей компонентів;

Стан – це об’єкт, що зберігає інформацію, яка може змінюватися протягом життєвого циклу компоненту. Стан використовується для реагування на взаємодію користувача, зберігання даних та управління поведінкою компоненту;

GPS – Global Positioning System (Глобальна Система Позиціонування);

JSON – JavaScript Object Notation (Нотація Об'єктів JavaScript);

SDK – Software Development Kit (Набір Засобів Розробки Програмного Забезпечення);

ЗМІСТ

ВСТУП

Актуальність теми: в умовах стрімкого розвитку мобільних технологій, важливість ефективного ведення та управління нотатками в мобільних додатках, інтегрованих з Google Maps, набуває особливої актуальності. Це обумовлено зростаючим використанням мобільних пристроїв і потребою в зручних інструментах для організації інформації.

Мета дослідження: метою дослідження є вивчення та вдосконалення процесу розробки кросплатформенних мобільних додатків для створення нотаток на Google Maps, щоб забезпечити ефективнішу роботу на різних мобільних операційних системах. Завдання включає аналіз методів розробки, визначення архітектури додатку, розробку інтерфейсу з використанням API Google Maps, і дослідження ефективності реалізації.

Для досягнення мети в роботі сформульовано та вирішено такі завдання:

1. Визначити існуючи методи створення кросплатформових мобільних додатків.
2. Спроектовати архітектуру додатку.
3. Розробити користувацький інтерфейс.
4. Інтегрувати картографічні сервіси.
5. Синхронізувати дані між різними пристроями.
6. Забезпечити захист персональних даних

Об’єкт дослідження: процес ведення та управління нотатками з використанням мобільних додатків, інтегрованих з Google Maps.

Предмет дослідження: методи розробки кросплатформових мобільних додатків, зокрема, використання API Google Maps для створення інтуїтивно зрозумілих та ефективних інструментів ведення нотаток.

Методи дослідження: для вирішення поставленої задачі використано такі методи: теорія системного аналізу, принципи функціонального та об’єктно-орієнтованого програмування.

Наукова новизна: розробка методу інтеграції GPS-даних з нотатками, що дає користувачам можливість ефективно відстежувати та організовувати свої замітки за допомогою геолокації.

Практична цінність: додаток спрощує ведення нотаток з використанням географічних даних, забезпечуючи користувачам легкий доступ до інформації про різні місця та їх характеристики.

Особистий внесок автора: включає вибір методів дослідження, розробку архітектури та реалізацію додатку, а також аналіз і оцінку отриманих результатів.

Структура та обсяг кваліфікаційної роботи: робота складається з вступу, трьох розділів, висновків, списку використаних джерел, і двох додатків. Загальний обсяг роботи становить 92 сторінки, із них 47 сторінок основного тексту, в тому числі 23 рисунків, і список використаних джерел із 29 найменувань на 1 сторінці

РОЗДІЛ 1
АНАЛІЗ ТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.2. Ведення нотаток про локації на мапі

Завдання, пов'язані з веденням нотаток про локації на мапі у мобільних додатках:

1. Розробка користувацького інтерфейсу:
   - розробка інтуїтивного та зручного інтерфейсу для додавання, редагування та видалення нотаток на мапі;
   - забезпечення можливості маркування конкретних точок на мапі та їх збереження як нотаток.
2. Інтеграція з картографічними сервісами:
   - глибока інтеграція з картами, такими як Google Maps, для відображення точних геолокацій та додаткової інформації про місця на мапі.
3. Синхронізація:
   - розробка механізму синхронізації, що дозволить користувачам отримувати доступ до своїх нотаток на різних мобільних пристроях;
   - забезпечення автоматичної синхронізації змін, які внесені на одному пристрої, на інших пристроях користувача.
4. Захист персональних даних:
   - розробка системи безпеки для забезпечення конфіденційності та цілісності персональних даних користувачів;
   - гарантування, що доступ до нотаток мають лише їх власники.

1.3. Існуючі аналоги систем ведення нотаток

Для кращого розуміння сучасного стану ринку та існуючих рішень у сфері ведення нотаток про локації на мапі, ми проаналізуємо деякі популярні додатки, такі як My Maps, Avenza Maps та Locus Map.

Розглянемо мобільний додаток My Maps. Він має такі переваги:

– зручність використання: My Maps використовує інтерфейс Google Maps, який вже відомий багатьом користувачам і вважається дуже зручним;
– спільний доступ: користувачі можуть ділитися своїми створеними картами з іншими користувачами, що дозволяє колективно працювати над проектами;
– інтеграція з Google-екосистемою: My Maps інтегрований з іншими Google-сервісами, такими як Google Drive і Google Photos, що робить його зручним для збереження та обміну даними.

Додаток My Maps має наступні недоліки:

– обмежена функціональність нотаток: My Maps має обмежену можливість додавання текстових нотаток до маркерів, і це може бути недостатньо для деяких користувачів;
– потреба в інтернет-підключенні: для роботи з My Maps потрібне активне Інтернет-підключення, що може бути не зручним у віддалених або обмежених місцях.

Розглянемо мобільний додаток Avenza Maps. Він має такі переваги:

– офлайн-режим: однією з найбільших переваг Avenza Maps є можливість завантажити географічні карти і працювати в офлайн-режимі, що корисно для користувачів без доступу до Інтернету;
– спеціалізація на геоданих: додаток спеціалізується на роботі з геоданими і може бути корисним для подорожуючих, геодетів та геологів;
– підтримка формату GeoPDF: Avenza Maps підтримує формат GeoPDF, що дозволяє використовувати спеціальні карти.

Додаток Avenza Maps має наступні недоліки:

– спеціалізація на геоданих: Додаток спеціалізується на геоданих і може бути менш зручним для користувачів, які шукають простий спосіб ведення нотаток;

– можливі обмеження в безкоштовній версії: Деякі функції можуть бути доступні тільки у преміум-версії Avenza Maps.

Розглянемо мобільний додаток Locus Map. Він має такі переваги:

– розширені можливості створення точок і шляхів: Locus Map дозволяє додавати текстові нотатки та фотографії до точок на мапі, що корисно для ведення записів;
– підтримка офлайн-карт: користувачі можуть завантажувати офлайн-карти для використання без Інтернет-підключення;
– підтримка різних картографічних форматів: Locus Map підтримує різні формати карт, що дозволяє використовувати різні джерела даних.

Додаток Locus Map має наступні недоліки:

– складність використання для новачків: Locus Map може виглядати складним для новачків через розширені можливості, і вимагатиме часу для освоєння;
– необхідність завантаження офлайн-карт: користувачам доведеться завантажити офлайн-карти перед використанням, що може займати додатковий час.

1.4. Постановка задачі

Розробити кросплатформовий мобільний додаток для ефективного управління нотатками про локації на мапі. Додаток призначений для забезпечення зручного доступу та організації інформації про різноманітні місця.

Функціональні вимоги додатка:

1. Реєстрація та вхід:
   – реалізація функцій реєстрації та входу з валідацією даних;
   – забезпечення безпеки та конфіденційності користувацьких даних.
2. Головний екран (мапа з пінами):
   – візуалізація мапи з можливістю додавання та відображення пінів, що вказують на збережені локації;

- функціонал пошуку та фільтрації пінів за різними критеріями.

3. Додавання та управління пінами:
   - інтерфейс для додавання нових пінів з деталями: назва, опис, координати;
   - можливість редагування та видалення існуючих пінів;
   - опція для відзначення пінів як "Улюблених".
4. Список пінів:
   - екран для перегляду усіх збережених пінів з детальною інформацією;
   - функція фокусування на певному піні на мапі через вибір у списку.
5. Кластеризація пінів:
   - кластеризація для групування близьких локацій на мапі;
   - візуальна фільтрація пінів за результатами пошуку;
   - інтеграція з картографічними сервісами: Google Maps для точного відображення геолокацій.
6. Динамічна зміна теми інтерфейсу:
   - легке впровадження нових тем;
   - розробка модульної системи для легкого додавання нових тем;
   - можливість переключення між світлою та темною темами.

1.5. Висновок

Визначено функціональні вимоги для розробки мобільного додатку. Вимоги включають розробку користувацького інтерфейсу для ефективного управління нотатками про локації, включаючи функції додавання, редагування та видалення пінів. Також акцент зроблено на кластеризацію пінів для зручної візуалізації та інтеграцію з картографічними сервісами. Додатково враховано динамічну зміну теми інтерфейсу та можливість легкого впровадження нових тем, що забезпечить адаптивність та персоналізацію користувацького досвіду. Ці вимоги формують основу для розробки додатку, який буде відповідати сучасним потребам користувачів у зручному управлінні геолокаційними даними.

РОЗДІЛ 2
МЕТОДИ РОЗРОБКИ МОБІЛЬНИХ КРОСПАТФОРМОВИХ ДОДАТКІВ

2.1. Мобільні операційні системи

Для розробки мобільних кросплатформових додатки використовуються дві основні мобільні операційних системи:

- операційна система iOS;
- операційна система Android.

iOS – це мобільна операційна система, створена та розроблена компанією Apple Inc. ексклюзивно для їхніх мобільних пристроїв. Це одна з найбільш популярних операційних систем у світі, яка відрізняється своєю стабільністю, високою безпекою та інтеграцією з іншими продуктами Apple.

Основні особливості iOS:

- безпека та приватність: iOS має суворі налаштування безпеки та приватності, включаючи двофакторну аутентифікацію та шифрування даних;
- екосистема Apple: тісна інтеграція з іншими продуктами Apple, такими як MacBook, iPad, та Apple Watch, забезпечує синхронізоване та безшовне використання;
- App Store: магазин додатків App Store відомий своєю високою якістю та безпекою додатків, завдяки суворій політиці модерації [27].
- інтерфейс користувача: iOS має чіткий та інтуїтивно зрозумілий інтерфейс, який легко адаптується до різних розмірів екранів.

Виклики при розробці для iOS:

- строгі вимоги до дизайну та розробки додатків;
- обмеження у використанні деяких API і функцій системи;
- необхідність використання мови програмування Swift або Objective-C.

Переваги кросплатформової розробки для iOS (рис. 2.1):

– можливість розробки додатків, які працюють як на iOS, так і на інших платформах, зменшує час та витрати на розробку;
– фреймворки для кросплатформової розробки, такі як React Native або Flutter, дозволяють використовувати єдину кодову базу для різних ОС.

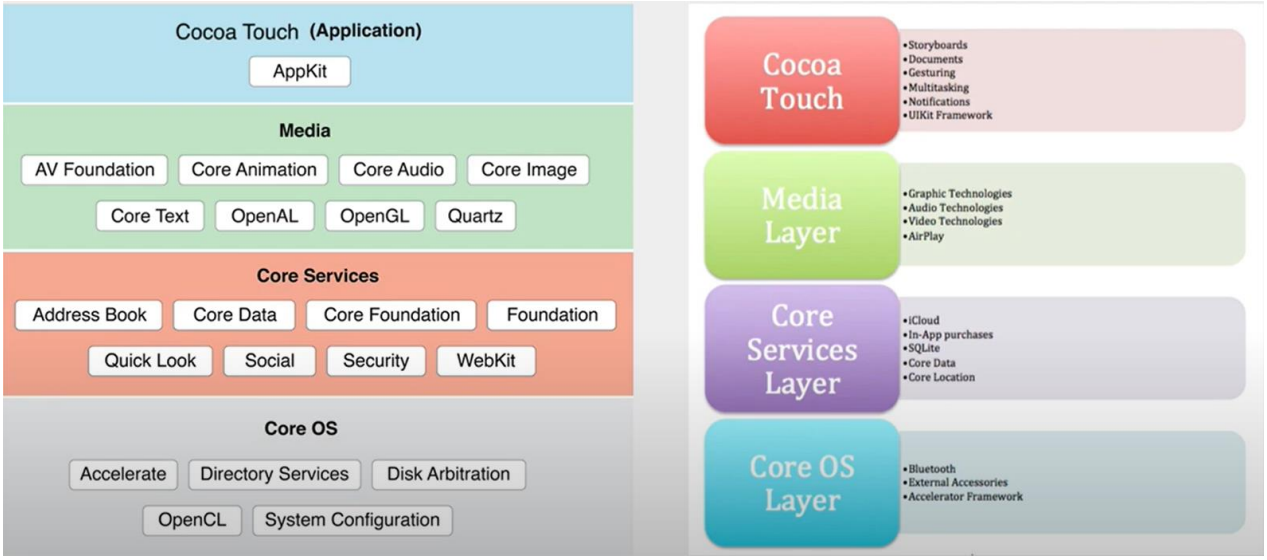

Рис. 2.1. Архітектура iOS

Android — це відкрита мобільна операційна система, розроблена Google, яка є найпоширенішою ОС для мобільних пристроїв у світі. Вона характеризується своєю гнучкістю, відкритістю та широким асортиментом пристроїв різних виробників [6].

Основні особливості Android:

– відкритість та гнучкість – Android базується на відкритому коді, що дозволяє розробникам більшу свободу в модифікації та налаштуванні системи [18];
– широкий вибір програмного забезпечення – Google Play Store містить мільйони додатків, доступних для завантаження;

– різноманітність апаратного забезпечення – Android працює на широкому спектрі пристроїв від різних виробників, пропонуючи різноманітність у виборі апаратного забезпечення;
– інтеграція з Google Services – тісна інтеграція з сервісами Google, включаючи Google Maps, Gmail та інші.

Виклики при розробці для Android:

– фрагментація ринку через велику кількість пристроїв з різними розмірами екранів та версіями ОС;
– необхідність оптимізації продуктивності та використання батареї;
– комплексність тестування через різноманітність апаратних конфігурацій.

Переваги кросплатформової розробки для Android:

– уніфікована розробка для різних пристроїв і версій ОС, що спрощує процес тестування та впровадження;
– використання кросплатформових фреймворків, таких як Xamarin або Flutter, дозволяє створювати додатки, які легко адаптуються до різних пристроїв та розмірів екрану.

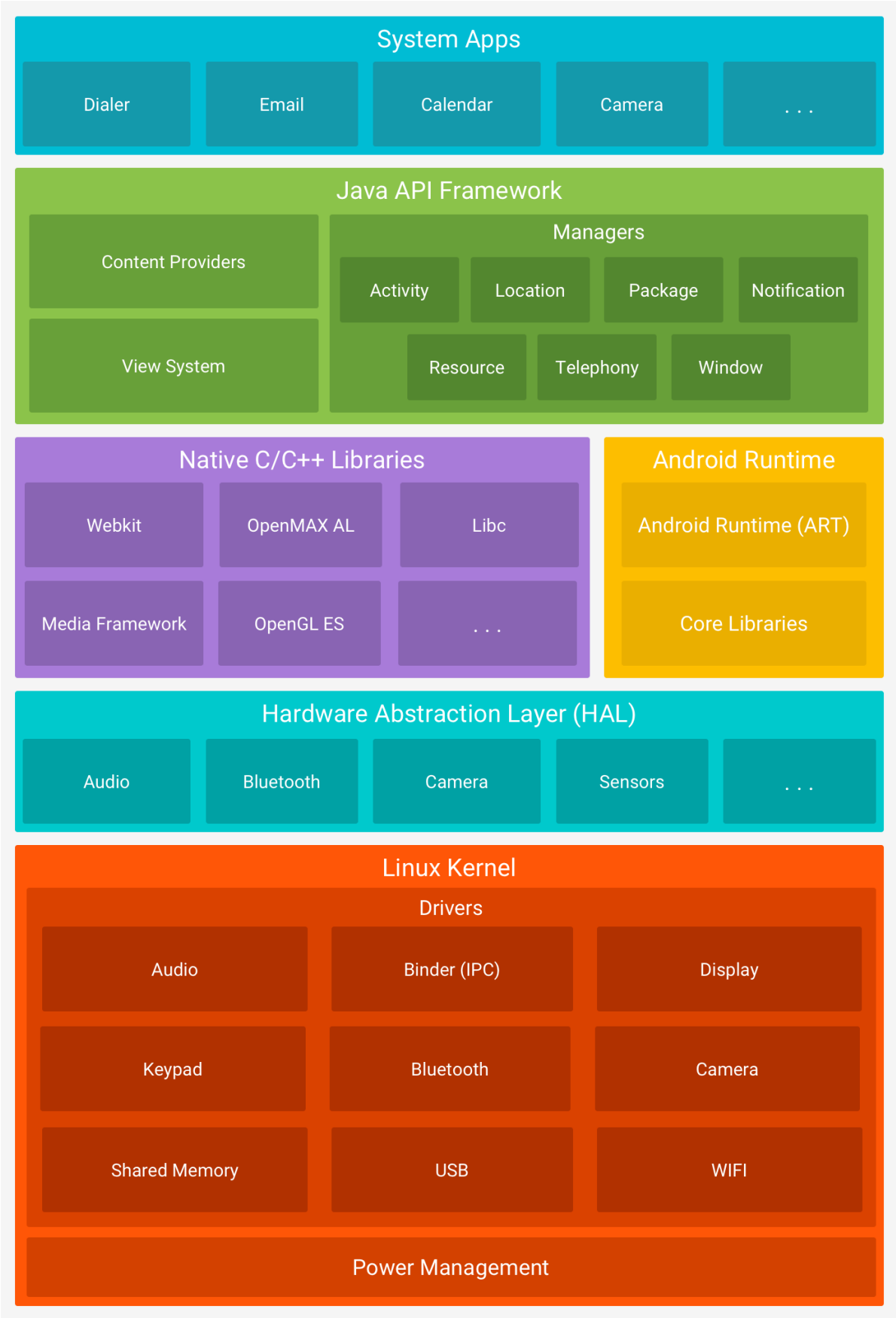

Рис. 2.2. Архітектура Android

2.2. Методи розробки кросплатформенних мобільних додатків

Кросплатформова розробка мобільних додатків дозволяє створювати програми, сумісні з кількома операційними системами, використовуючи одну кодову базу. Це значно економить час та ресурси. Нижче розглядаються кілька популярних фреймворків для кросплатформової розробки:

1. Xamarin – це фреймворк, що використовує C# та .NET для створення додатків для iOS, Android та Windows. Забезпечує високий рівень доступу до платформо-специфічних функцій [28].

Переваги Xamarin:

– можливість використання потужних бібліотек .NET;

– спільний код для бізнес-логіки.

Недоліки Xamarin: важчий у вивченні для розробників, які не знайомі з C#.

2. Flutter – це фреймворк, розроблений Google, що використовує мову Dart для створення нативної компіляції на iOS та Android. Відомий своїм швидким рендерингом UI [5].

Переваги Flutter:

– висока продуктивність;

– красивий дизайн інтерфейсу;

– широкий вибір віджетів.

Недоліки Flutter: спільнота Flutter ще зростає, і наявність ресурсів може бути обмеженою [16].

3. Фреймворк React Native – це фреймворк, розроблений Facebook, що використовує JavaScript і React для створення додатків. Дозволяє використовувати нативні компоненти під керуванням JavaScript.

Переваги React Native: велика спільнота, багато готових до використання компонентів, спільна кодова база для iOS та Android.

Недоліки React Native: час від часу потрібно писати платформо-специфічний код.

4. Ionic – фреймворк, що використовує HTML, CSS та JavaScript для створення додатків, підтримується на різних платформах.

Переваги Ionic:

– легкість у використанні для розробників з досвідом у веб-розробці;

– широкий вибір UI компонентів.

Недоліки Ionic: може мати меншу продуктивність порівняно з іншими фреймворками, особливо на складних інтерфейсах.

2.3. Методи розробки гібридних мобільних додатків

Гібридні мобільні додатки є компромісом між нативними та веб-додатками. Вони дозволяють розробникам використовувати веб-технології (HTML, CSS, JavaScript) для створення додатків, які потім загортаються в нативну оболонку і можуть бути розповсюджені через магазини додатків. Розглянемо два популярних фреймворки для гібридної розробки:

1. Apache Cordova (також відомий як PhoneGap) – фреймворк, що дозволяє розробникам створювати мобільні додатки, використовуючи стандартні веб-технології. Вона надає набір API для доступу до функцій мобільного пристрою, таких як камера, GPS, акселерометр тощо [11].

Переваги Apache Cordova:

– легкість у використанні для тих, хто вже знайомий з веб-розробкою;

– широка підтримка платформ.

Недоліки Apache Cordova – може мати обмеження з точки зору продуктивності та інтеграції з платформою [29].

2. AppMaster – що є більш новим гравцем у сфері гібридної розробки, який пропонує "безкодовий" підхід до створення додатків. Він дозволяє розробникам та бізнес-аналітикам створювати додатки за допомогою графічного інтерфейсу.

Переваги AppMaster:

– швидка розробка;

– легкість використання без потреби у глибоких знаннях програмування.

Недоліки AppMaster:

– обмежена гнучкість у порівнянні з традиційними методами розробки;
– може не підходити для складних або дуже специфічних проектів.

2.4. Фреймворк React Native

React Native — це популярний фреймворк для розробки кросплатформових мобільних додатків, створений і підтримуваний Facebook. Він дозволяє розробникам створювати додатки для iOS та Android, використовуючи JavaScript та React, одночасно надаючи доступ до нативних платформових можливостей [3].

React Native є такі базові компоненти:

– View – аналог у веб-розробці, використовується для створення контейнерів;
– Text – для відображення тексту;
– Image – для включення зображень;
– ScrollView – контейнер для прокручування вмісту;
– TextInput – для введення тексту користувачами.

React Native є такі інтерактивні компоненти, як:

– Button: простий компонент для створення кнопок;
– TouchableOpacity – контейнер, який змінює свою прозорість при торканні;
– TouchableHighlight – контейнер, що змінює виділення при торканні.

Списки та скролінг в React Native забезпечується такими компонентами:

– FlatList – високопродуктивний компонент для рендерингу простого списку;
– SectionList – схожий на FlatList, але для секційованих списків.

Для стилізація компонентів React Native використовує StyleSheet для створення стилів, схожих на CSS у веб-розробці, але з певними обмеженнями та специфічними властивостями

В React Native навігація між різними екранами і вікнами додатків зазвичай здійснюється за допомогою спеціальних бібліотек, таких як React Navigation або React Native Navigation [7].

В React Native отримати доступ до нативних функцій можна за допомогою модулів та API, які дозволяють взаємодіяти з нативними функціями пристрою, такими як камера, GPS, сенсори тощо.

Архітектура React Native є важливою частиною його успіху та популярності. Вона дозволяє розробникам писати модульний, легко читаний код, який може взаємодіяти з нативними елементами платформи.

Основні аспекти архітектури React Native включають (див. рис. 2.3):

- Bridge (Міст) – центральний компонент архітектури, який забезпечує взаємодію між JavaScript кодом та нативними модулями. Bridge використовує JSON-повідомлення для комунікації між двома частинами додатку [4];
- JavaScript Core – всі компоненти React Native, написані на JavaScript, виконуються в середовищі JavaScript Core. Це дозволяє React Native додаткам використовувати всі можливості JavaScript та React;
- Native Modules – React Native дозволяє розробникам інтегрувати нативний код (наприклад, написаний на Objective-C, Swift, або Java) з JavaScript кодом. Це забезпечує доступ до нативних функцій пристрою та платформи, таких як камера, геолокація тощо;
- Native Components – React Native використовує нативні компоненти замість веб-компонентів. Наприклад, він використовує UIView на iOS або android.view на Android, замість HTML елементів;
- React – реактивні компоненти та Virtual DOM, які є основою React, також є ключовими елементами в React Native. Вони дозволяють створювати інтерактивний UI з декларативним кодом;

– UI Thread & JavaScript Thread – в React Native є два основних потоки: UI Thread (для нативних елементів) та JavaScript Thread (де виконується JavaScript код). Bridge синхронізує взаємодію між цими потоками.

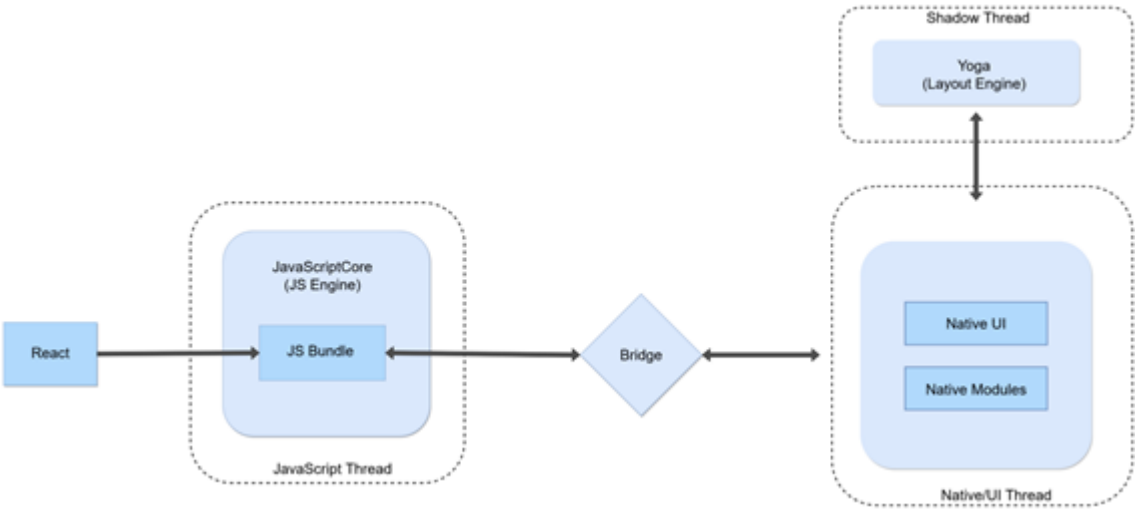

Рис. 2.3. Архітектура React Native

У React Native важливу роль відіграють потоки (threads), які забезпечують ефективну роботу та взаємодію між різними частинами додатку.

В React Native Є такі потоки:

– JavaScript Thread (JS Thread) – цей потік відповідає за виконання всього JavaScript коду. Це включає логіку додатку, обробку подій, виклик API та рендеринг компонентів React. Переважно, вся бізнес-логіка додатку виконується у цьому потоці [20];
– Native Thread (UI Thread) – нативний потік відповідає за рендеринг нативних UI компонентів та обробку користувацького вводу. Це основний потік, який взаємодіє з операційною системою та забезпечує відображення елементів інтерфейсу;
– Shadow Thread (Layout Thread) – відомий також як "Layout Thread", він відповідає за розрахунок макету (layout) в додатку. Цей потік використовується для розрахунку розмірів та позицій компонентів

інтерфейсу. Виконує обчислення, необхідні для відображення компонентів на екрані відповідно до їх стилів та розміщення;

– Bridge (Communication Bridge) – Bridge не є потоком у традиційному розумінні, але це ключовий механізм, який забезпечує взаємодію між JavaScript та нативними потоками. Через Bridge проходять повідомлення та команди між JS та нативним UI, дозволяючи синхронізувати стан та події між різними частинами додатку.

Expo є популярним інструментом та фреймворком у екосистемі React Native, який спрощує процес розробки та тестування мобільних додатків. Він надає набір інструментів та бібліотек, що дозволяють розробникам швидко створювати додатки без необхідності взаємодії з нативним кодом.

Основні особливості та переваги Expo:

– Expo дозволяє розробникам створювати додатки, використовуючи лише JavaScript та React, уникаючи прямого взаємодії з нативним кодом;
– посилює продуктивність розробки, особливо для новачків або тих, хто не має досвіду з нативним мобільним програмуванням;
– з Expo розробники можуть миттєво переглядати зміни у своєму коді на реальних пристроях або емуляторах без необхідності компіляції додатку;
– Expo SDK – набір готових до використання компонентів та API, які надають доступ до функцій пристрою, таких як камера, контакти, акселерометр тощо [25];
– Expo спрощує процес публікації додатків у магазинах додатків та дозволяє розробникам випускати оновлення без необхідності повторної публікації весь додаток.

Обмеження Expo:

– обмежений доступ до нативних функцій: не всі нативні можливості доступні через Expo, що може бути обмеженням для деяких розширених функцій додатків;

– розмір додатку – додатки, створені з Expo, можуть мати більший розмір через включені бібліотеки та функціонал.

Expo є ідеальним варіантом для тих, хто шукає швидкий та простий спосіб створення мобільних додатків на React Native, особливо для проектів, які не потребують глибокої інтеграції з нативними функціями. Однак, для розробки більш складних та інтегрованих з платформою додатків може бути кращим вибором використання React Native CLI.

React Native CLI (Command Line Interface) є офіційним інструментом для створення та управління проектами React Native. Він надає більшу гнучкість та контроль над розробкою додатків порівняно з Expo та ідеально підходить для розробників, які потребують повного доступу до нативних можливостей платформи [24].

Основні Особливості та переваги React Native CLI:

– повний контроль над проектом – React Native CLI дозволяє розробникам мати повний контроль над конфігурацією та структурою проекту. Вони можуть налаштовувати нативний код, використовувати власні плагіни, бібліотеки та залежності;
– нативна інтеграція – інструмент дозволяє безпосередньо взаємодіяти з нативними модулями та компонентами, що важливо для реалізації складних функцій та оптимізації додатків;
– гнучкість у виборі бібліотек – розробники можуть використовувати будь-які сторонні нативні бібліотеки без обмежень, що зустрічаються в Expo;
– налаштування потоку розробки – CLI надає більшу свободу у виборі та налаштуванні інструментів розробки, включаючи конфігурацію збірки, відлагодження та тестування.

Особливості роботи з React Native CLI:

- налаштування середовища – вимагає встановлення та налаштування нативних середовищ розробки, таких як Xcode для iOS та Android Studio для Android;
- команди для створення та управління проектом: використання командного рядка для створення нових проектів, зв'язування бібліотек, виконання додатків тощо;
- можливість оптимізувати додатки під конкретні особливості платформи.

React Native CLI є потужним інструментом для розробки мобільних додатків, який надає розробникам повний контроль над проектом та велику гнучкість. Цей інструмент ідеально підходить для більш досвідчених розробників та для проектів, що вимагають глибокої інтеграції з нативними функціями та оптимізації під конкретні платформи.

2.5. Бібліотека React

Життєвий цикл компонентів у React – це набір методів, які автоматично викликаються під час створення, оновлення та знищення компонентів. Розуміння цих методів є важливим для ефективної розробки React-додатків [19].

Основні етапи життєвого циклу компонентів в React:

- монтування (Mounting);
- оновлення (Updating);
- демонтування (Unmounting).

При монтуванні викликаються такі функції:

- constructor(): викликається перед монтуванням компонента;
- static getDerivedStateFromProps() викликається перед рендерингом компонента, як при початковому монтуванні, так і при подальших оновленнях;
- render(): викликається для рендерингу компонента;

– componentDidMount(): викликається один раз після першого рендерингу компонента.

При оновлення викликаються такі функції:

– static getDerivedStateFromProps();

– shouldComponentUpdate(): викликається перед рендерингом при оновленні компонента;

– render(): викликається для рендерингу оновленого компонента;

– getSnapshotBeforeUpdate(): викликається перед застосуванням змін до DOM;

– componentDidUpdate(): викликається після оновлення компонента.

При демонтуванні викликається функція componentWillUnmount() (до того, як компонент буде видалений з DOM).

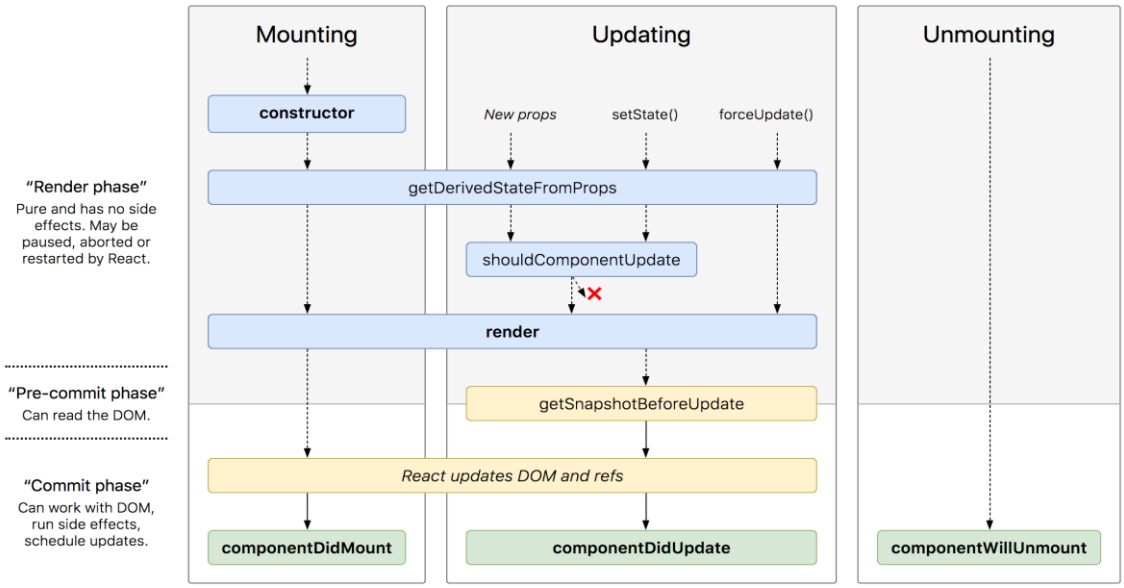

Рис. 2.4. Життєвий цикл React компонентів

Управління станом є ключовою концепцією в React, яка дозволяє компонентам реагувати на зміни даних та взаємодіяти з користувачем.

В React є такі підходи до управління станом:

– локальний стан компоненту:
– підняття стану (lifting state up):
– контекст (Context API):
– станові контейнери (Redux, MobX):

Локальний стан зберігається всередині компоненту і контролюється за допомогою this.setState у класових компонентах або хуків useState у функціональних компонентах. Він використовується для даних, які необхідні лише в межах одного компоненту.

Підняття стану (lifting state up) – це підхід, що дозволяє синхронізувати дані між різними компонентами шляхом "підняття" їх спільного стану до найближчого спільного предка.

Контекст (Context API) дозволяє передавати дані безпосередньо через дерево компонентів, уникаючи пропс-дрілінгу (передачі пропсів через кілька рівнів компонентів), що ідеально підходить для глобальних даних, таких як тема інтерфейсу, мовні налаштування тощо.

Для більш складних додатків часто використовуються зовнішні бібліотеки для управління станом, такі як Redux або MobX – ці інструменти надають ще більшу гнучкість та контроль над глобальним станом додатку.

Класові компоненти – це компоненти у React, які створюються за допомогою ES6 класів. Вони надають більше можливостей, ніж функціональні компоненти, включаючи внутрішній стан, життєвий цикл компонентів та обробку подій [1].

Чому краще не використовувати класові компоненти в розробці:

– класові компоненти зазвичай мають більш складну структуру та синтаксис порівняно з функціональними компонентами;

– з появою хуків у React 16.8, функціональні компоненти отримали можливість використовувати стан та інші функції React без написання класу, що спрощує код та підвищує читабельність;
– React зосереджується на розвитку та оптимізації функціональних компонентів та хуків, що робить їх переважним вибором для сучасних додатків.

Враховуючи ці фактори, рекомендується використовувати функціональні компоненти та хуки для створення більш чистого, сучасного та легко підтримуваного коду в React-додатках.

Функціональні компоненти в React стали більш популярними з появою хуків, оскільки вони дозволяють використовувати стан та інші функції React у простішому та більш експресивному форматі.

Основні характеристики функціональних компонентів:

– декларативність – функціональні компоненти засновані на декларативному підході до визначення інтерфейсу користувача, що робить код більш чистим та зрозумілим;
– відсутність життєвого циклу та стану (до появи хуків): первісно функціональні компоненти не мали доступу до життєвого циклу та стану, але з появою хуків це змінилося;
– використання хуків – починаючи з React 16.8, хуки дозволяють використовувати стан, життєвий цикл та інші функції React у функціональних компонентах [26].

Переваги функціональних компонентів:

– простота та чистота коду – завдяки відсутності класових конструкцій, код стає простішим та легшим для читання;
– легкість тестування та підтримки – менше шансів на помилки та легше тестування через відсутність складної логіки та стану;
– краща підтримка в React: З часом React все більше фокусується на функціональних компонентах і хуках.

Хуки в React – це функції, які дозволяють використовувати стан та інші можливості React без написання класових компонентів. Запроваджені у React 16.8, хуки зробили розробку функціональних компонентів більш гнучкою та ефективною.

Основні Хуки в React:

- useState: дозволяє додавати стан до функціональних компонентів;
- useeffect: використовується для виконання побічних ефектів у компонентах, таких як запити до API, підписки або ручні зміни в DOM;
- useContext: дозволяє використовувати контекст для доступу до глобального стану без пропс-дрілінгу;
- useReducer: альтернатива useState для управління складнішим станом з використанням редуктора;
- useCallback: мемоїзує функції для оптимізації продуктивності, особливо при передачі функцій у дочірні компоненти;
- useMemo: мемоізує значення та допомагає уникнути непотрібних обчислень при рендерингу;
- useRef: використовується для доступу до DOM-елементів та зберігання змінних, які не викликають перерендеринг компоненту.

Переваги використання хуків:

- спрощення коду та підвищення читабельності;
- можливість використання стану та інших функцій React у функціональних компонентах;
- покращена повторне використання логіки між компонентами.

2.6. Висновок

Проведено всебічний аналіз ключових аспектів розробки мобільних додатків з використанням React Native. Це включало розгляд різних підходів та інструментів, які відіграють важливу роль у створенні ефективних та високопродуктивних мобільних додатків.

Проаналізовано основні мобільні операційні системи, такі як iOS та Android, та розглянуто, як їх особливості впливають на процес розробки. Особлива увага була приділена кросплатформовим фреймворкам, таким як Xamarin, Flutter, React Native та Ionic, а також гібридним методам розробки з використанням Apache Cordova та AppMaster. Висвітлено ключові особливості React Native, включаючи його архітектуру, хуки, віртуальний DOM, та інструменти як Expo та React Native CLI.

РОЗДІЛ 3
ТЕХНОЛОГІЯ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ

3.1. Огляд технологій

Для розробки обрано такі технології:

– TypeScript – мова програмування, яка надає строгу типізацію та об'єктно-орієнтовані можливості, що допомагають у підвищенні якості коду та запобіганні помилок [2];

– React Native Expo – фреймворк для розробки кросплатформових мобільних додатків;

– Redux-Toolkit – це ефективний інструмент для управління станом, який значно спрощує використання Redux у мобільних додатках, особливо при роботі зі складними станами та асинхронними операціями [12];

– Firebase Realtime Database – це хмарна база даних від Google, яка дозволяє зберігати та синхронізувати дані між користувачами в реальному часі. Застосування: Використовується для створення мобільних додатків із швидкою та ефективною синхронізацією даних, що забезпечує актуальність інформації для всіх користувачів [8];

– react-native-maps – це спеціалізована бібліотека, яка інтегрується з нативними API для Android та iOS, надаючи можливість ефективної взаємодії з картами в мобільних додатках на платформі React Native. Вона забезпечує розширені функції для роботи з картами, включаючи відображення маркерів, маршрутів та інші інтерактивні елементи, оптимізуючи процес створення геолокаційних функцій у кросплатформових мобільних додатках;

– react-native-map-clustering – це розширення для react-native-maps, призначене для оптимізації відображення великої кількості маркерів на карті. Вона автоматично групує маркери, що знаходяться близько один до одного, у кластери, тим самим покращуючи продуктивність та

зручність візуалізації даних на карті у мобільних застосунках, розроблених на базі React Native.

– Flipper – як інструмент для налагодження, Flipper сприяє більш швидкому та ефективному виявленню та вирішенню проблем, а також оптимізації продуктивності мобільних додатків [9].

3.2. Архітектура додатку

Архітектура мобільного додатку побудована на основі React Native, яка використовує сучасний підхід до створення кросплатформових мобільних застосунків. Код написано з використанням TypeScript, що забезпечує строгу типізацію та підвищує надійність коду. Загальну структуру папок проекту зображено на рис. 3.1.

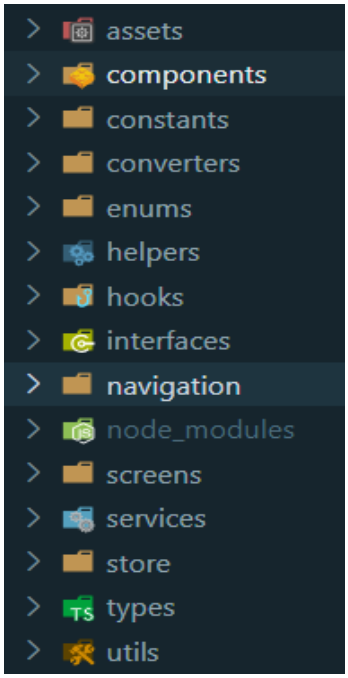

Рис. 3.1. Загальна структура проекту

Додаток розбито на модульні компоненти, що відповідають принципам React [24]. Кожен компонент виконує певну роль, будь то відображення інтерфейсу користувача, взаємодія з базою даних, або управління станом (рис. 3.2, рис. 3.3)

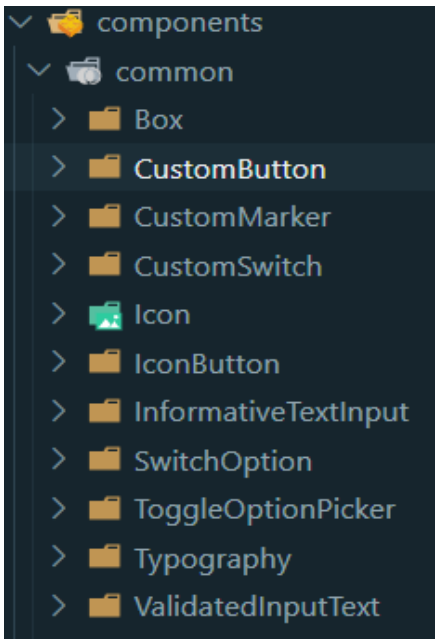

Рис. 3.2. Загальні компоненти додатку

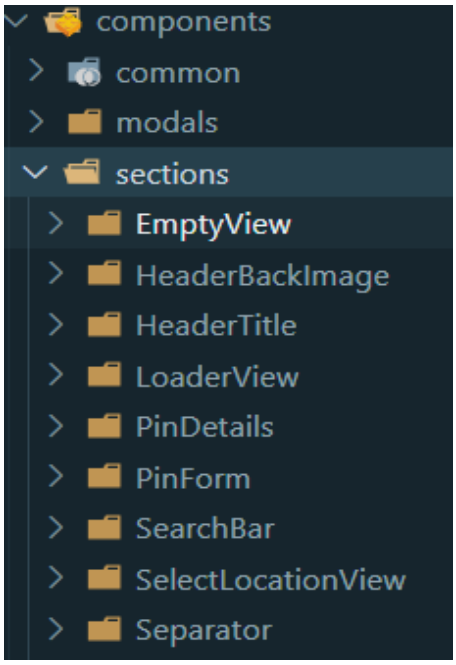

Рис. 3.3. Компоненти «секції»

Для управління станом додатку використовується Redux-Toolkit, який спрощує управління станом та організацію бізнес-логіки [14]. Redux-Toolkit допомагає уникнути багатьох складнощів, пов'язаних з традиційним Redux, завдяки вбудованим утилітам та методам (рис. 3.4, рис. 3.5) [2].

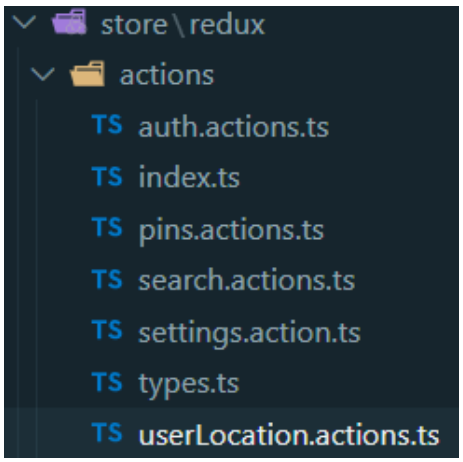

Рис. 3.4. Дії над сховищем redux

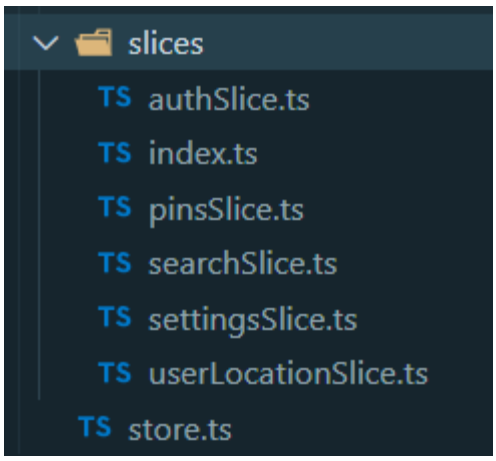

Рис. 3.5. Слайси сховища redux

3.3. База даних

Для розробки мобільного додатку обрано Firebase Realtime Database від Google. Це рішення базується на потребі в швидкій синхронізації даних між користувачами в реальному часі, а також на легкості інтеграції з мобільними додатками.

База даних організована у вигляді JSON об'єктів, що дозволяє інтуїтивно управляти даними. Основні вузли бази даних включають:

– pins: містить дані про місцезнаходження (піни), додані користувачами. Кожен пін включає опис, відмітку «Улюблений», заголовок та географічні координати;
– settings: зберігає налаштування користувачів, зокрема тему інтерфейсу;
– users: містить інформацію про користувачів, включаючи їх ідентифікатори, імена та електронні адреси.

Запроваджено наступні правила безпеки для захисту даних користувачів:

– основні правила: доступ до даних дозволений лише авторизованим користувачам;
– доступ до пінів: користувачі можуть працювати лише з власними пінами;
– валідація даних: застосовано правила для перевірки формату та типу даних, що зберігаються чи оновлюються.

Firebase Realtime Database інтегрована з додатком через бібліотеки react-native-firebase/app та react-native-firebase/database.

Це дозволяє користувачам отримувати оновлення даних в реальному часі, наприклад, при додаванні нових пінів або зміні налаштувань.

3.4. Інтерфейс додатку

Інтерфейс реєстрації користувача в додатку представлений на рис. 3.6 та рис. 3.7.

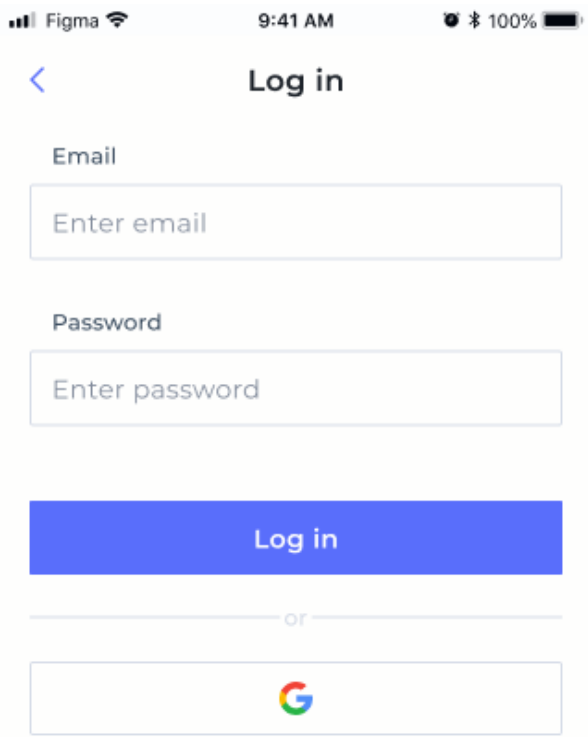

Рис. 3.6. Сторінка входу

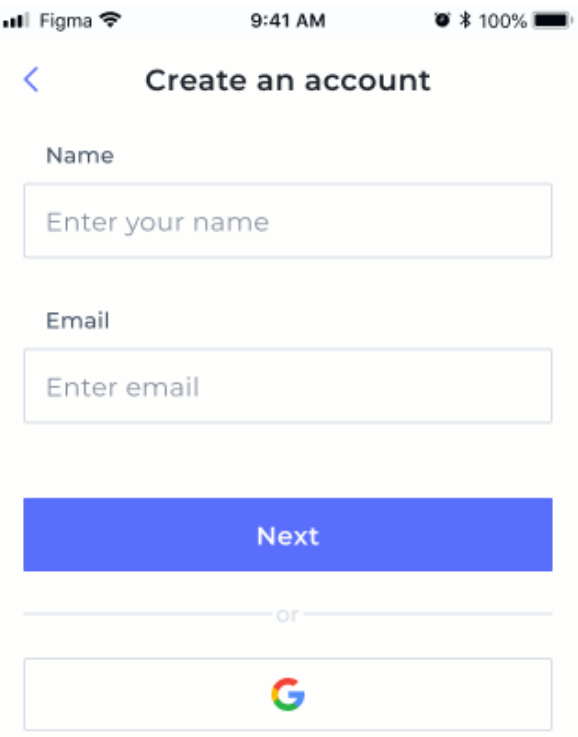

Рис. 3.7. Сторінка введення даних нового користувача

Інтерфейс створення паролю користувача та інтерфейс вкладки “Map” представлено на рис. 3.8 та рис. 3.9.

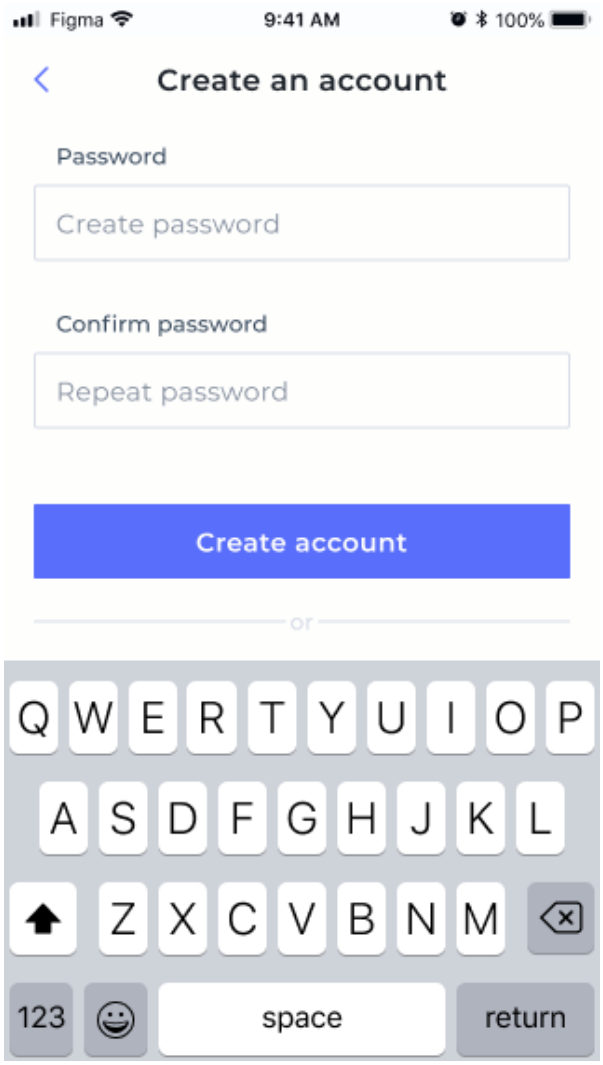

Рис. 3.8. Створення паролю

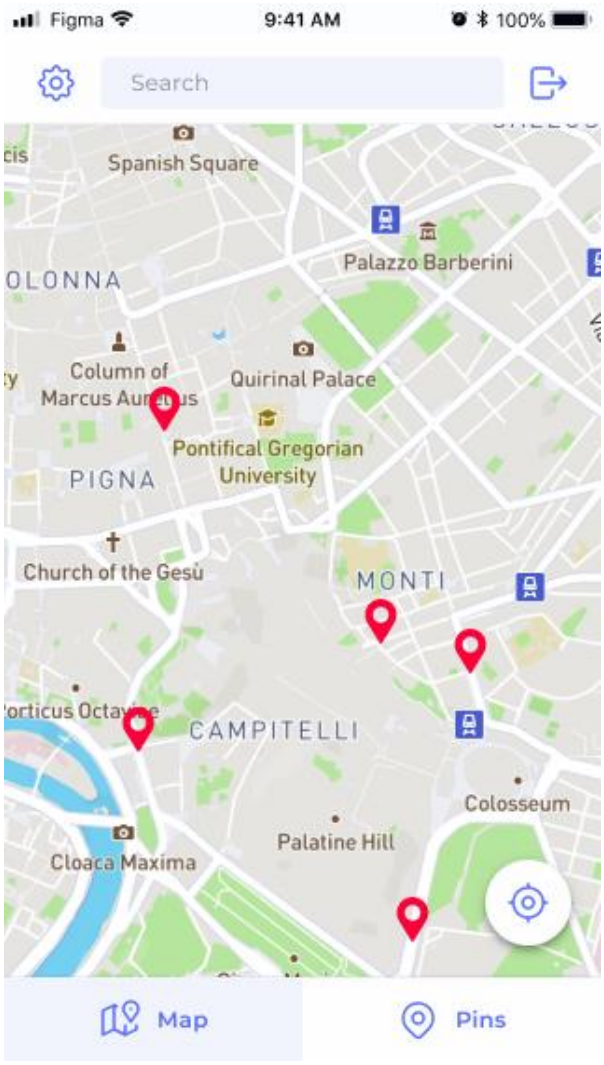

Рис. 3.9. Вкладка «Мапа»

Інтерфейс вкладки «Pins» та інтерфейс перегляду деталей піна представлено на рис. 3.10 та рис. 3.11.

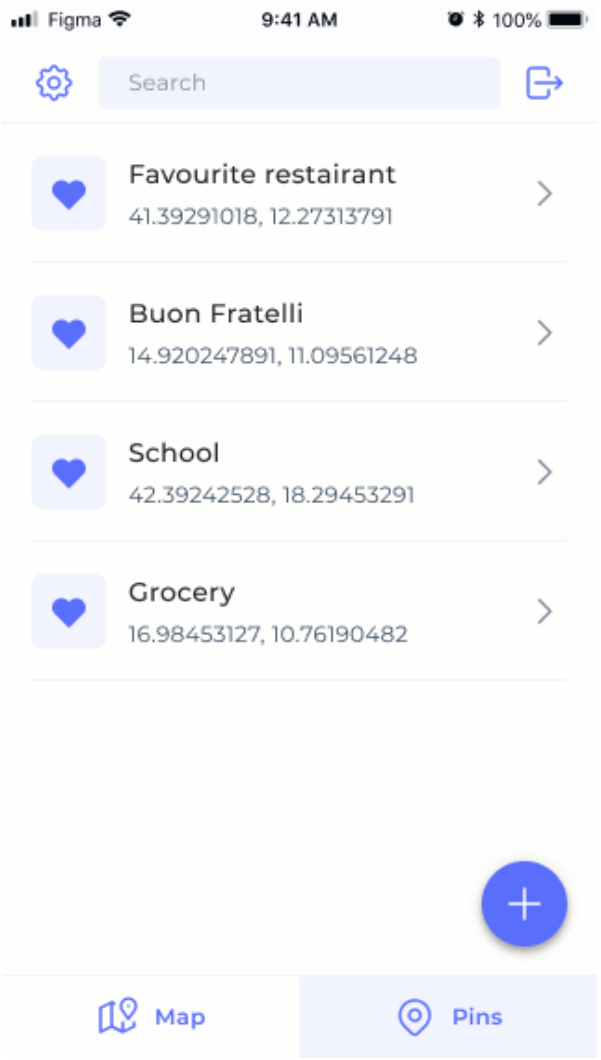

Рис. 3.10. Вкладка «Піни»

Рис. 3.11. Перегляд деталей піна

Інтерфейс пошуку пінів та інтерфейс контекстного меню піна представлено на рис. 3.12 та рис. 3.13.

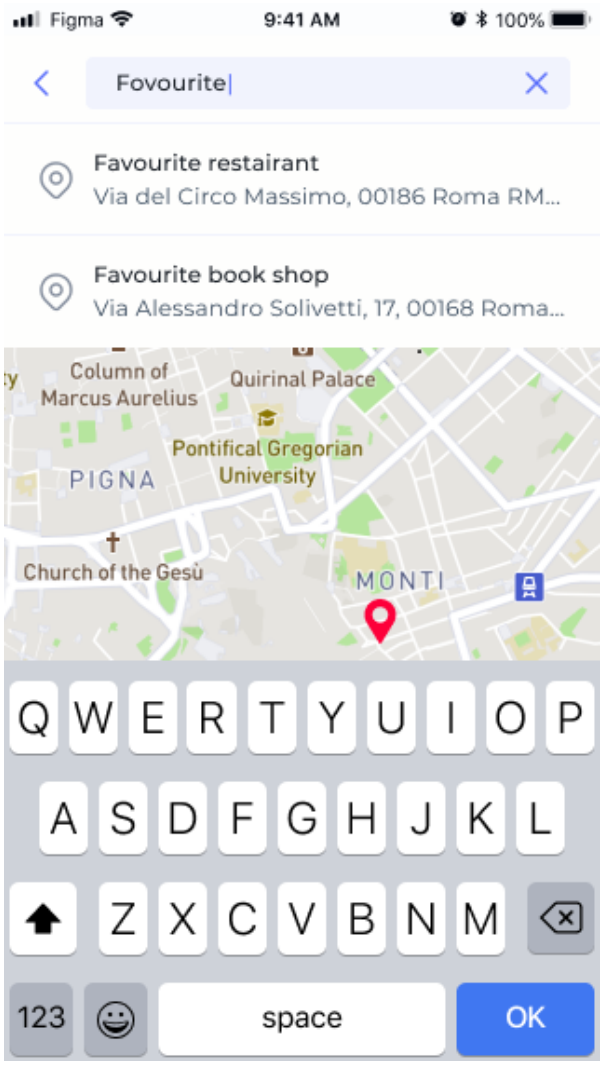

Рис. 3.12. Пошук пінів за ключовими словами

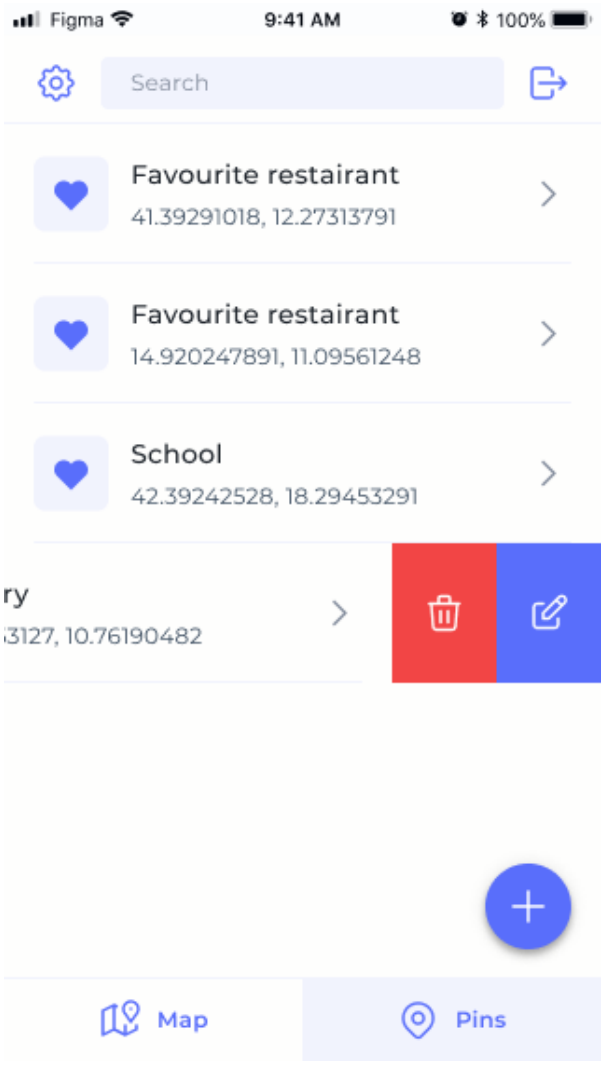

Рис. 3.13. Меню видалення та редагуванні піна

Інтерфейс створення нового піна та інтерфейс сторінки налаштувань представлено на рис. 3.14 та рис. 3.15.

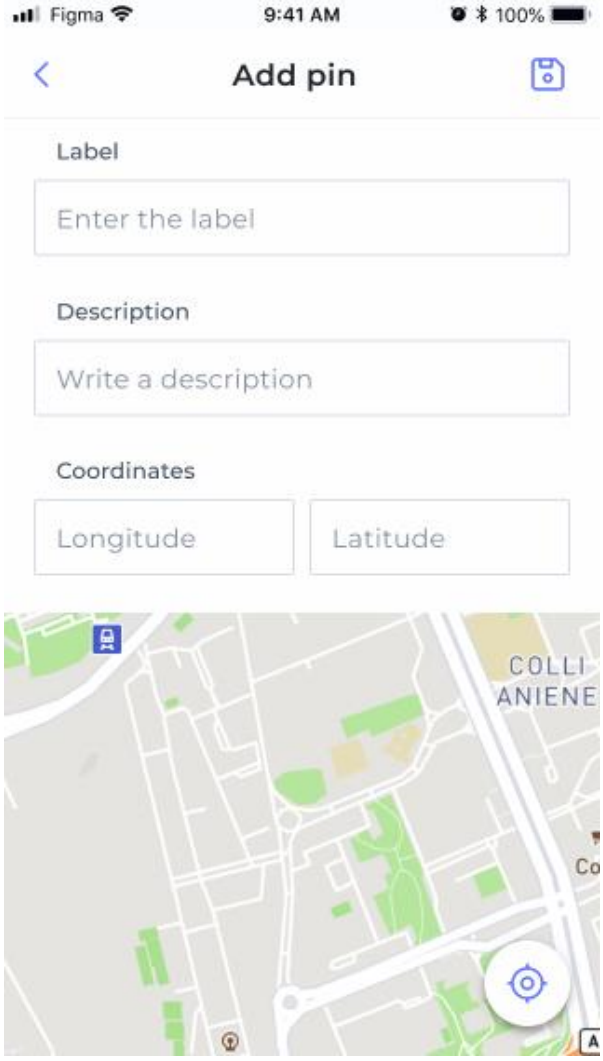

Рис. 3.14. Створення нового піна

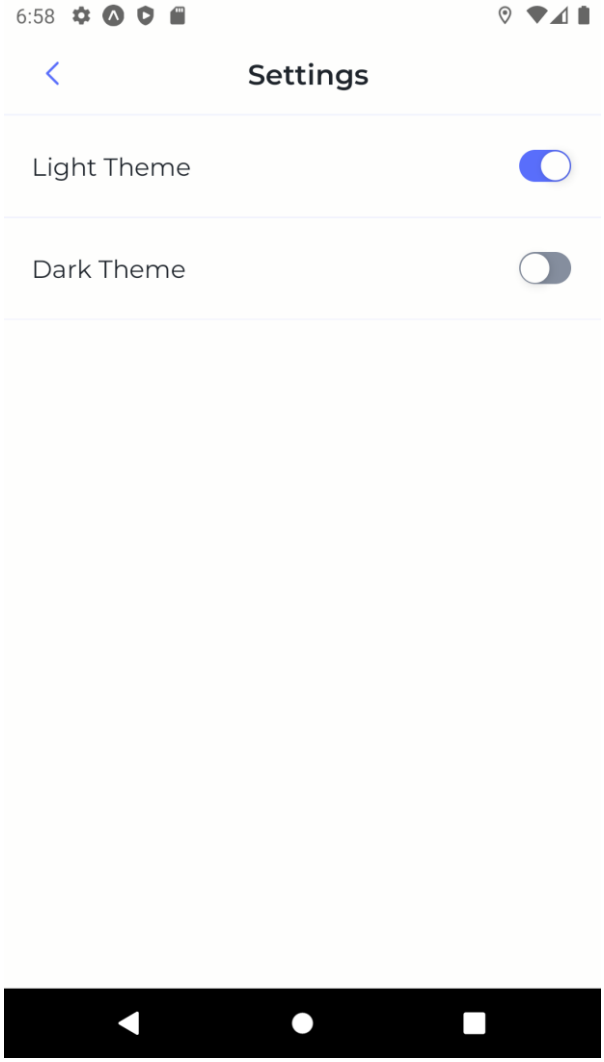

Рис. 3.15. Налаштування

Інтерфейс вкладки «Pins» та деталі піна, при умові що увімкнено темну тему, представлено на рис. 3.16 та рис. 3.17.

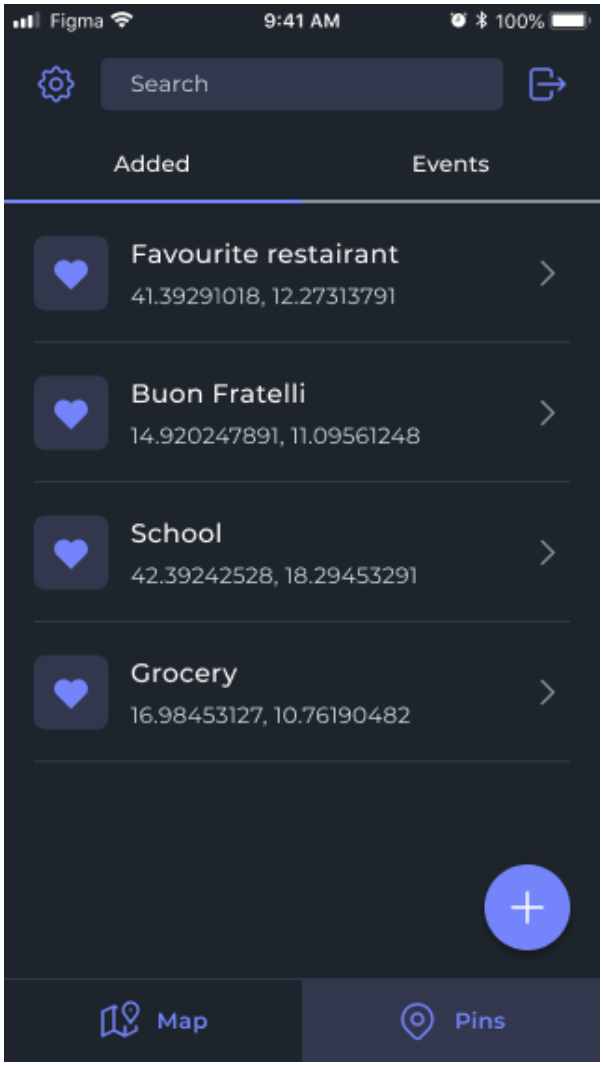

Рис. 3.16. Створення нового піна (темна тема)

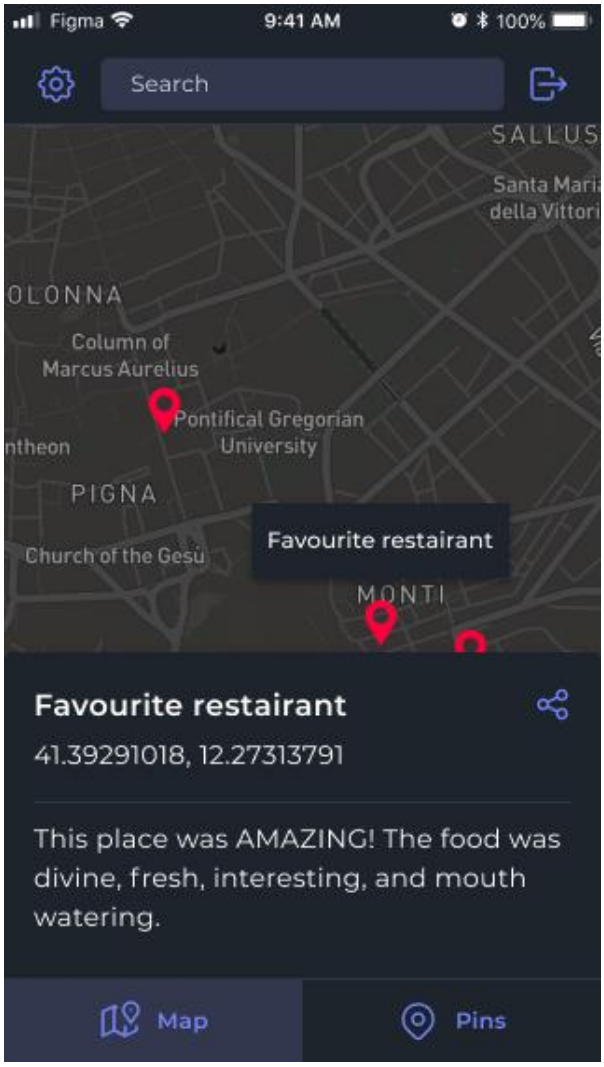

Рис. 3.17. Перегляд деталей піна (темна тема)

На рис. 3.18, рис. 3.19 зображено роботу додатку на iPhone 14 з операційною системою iOS 16.2

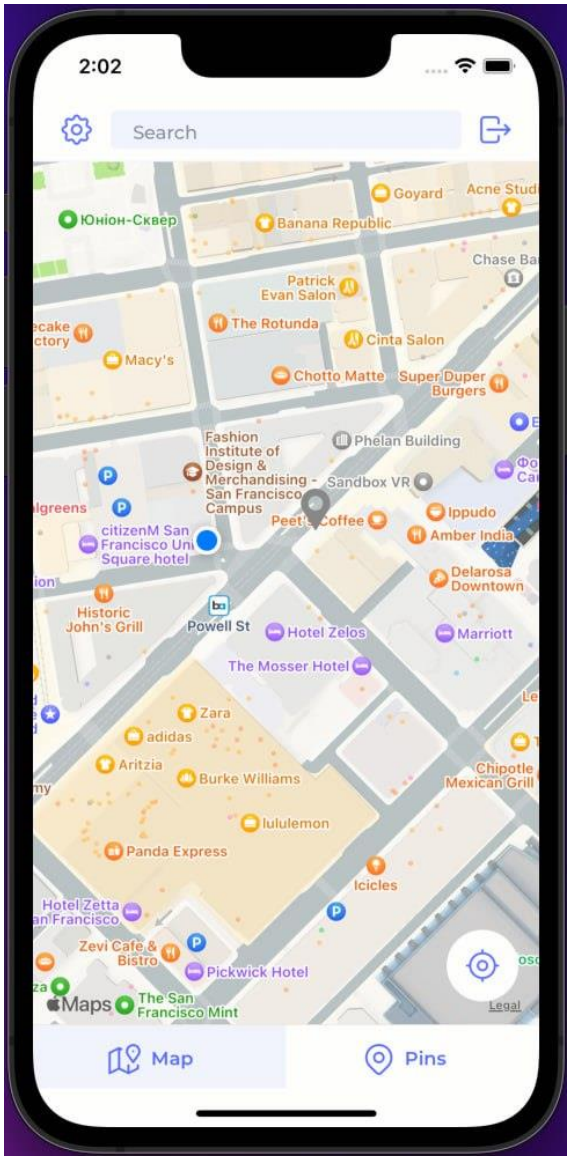

Рис. 3.18. Вкладка «Мапа» (iOS)

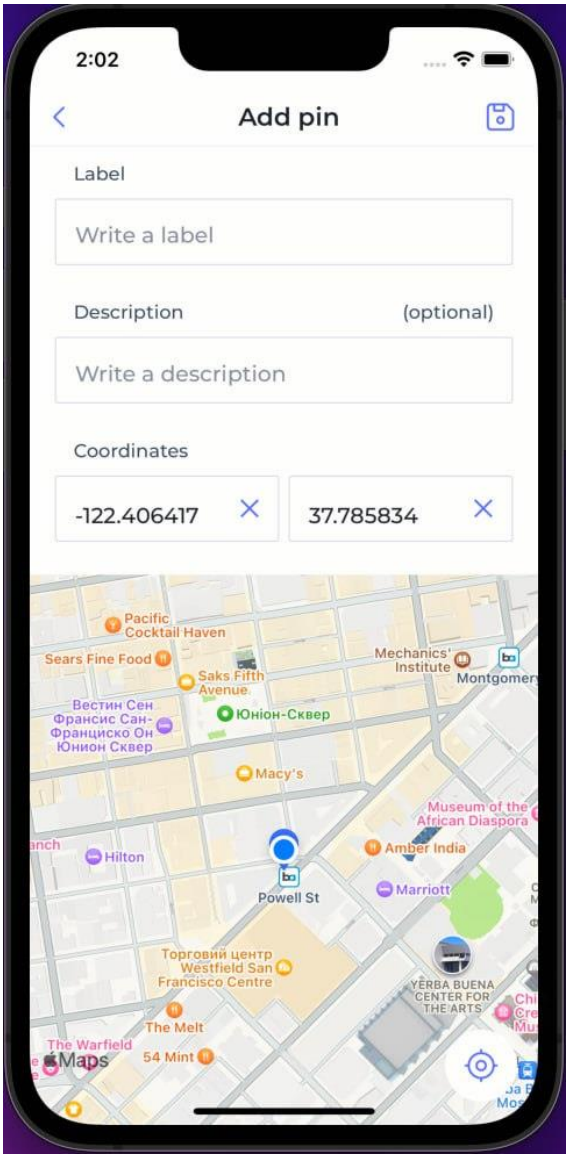

Рис. 3.19. Створення нового піна (iOS)

3.5. Навігація додатку

Використання React Navigation є стандартним підходом у спільноті React Native для реалізації навігації в мобільних додатках. Ця бібліотека є гнучкою, легко налаштовується та підтримує велику кількість навігаційних патернів, що робить її відмінним вибором для вашого додатку.

Використання NavigationContainer як кореневого компонента навігаційної структури є ключовим для управління станом навігації та контекстом в усьому додатку [15].

В додатку реалізовано такі Stack навігатори:

1. AuthStack – ефективно керує потоком аутентифікації користувача, забезпечуючи логічний перехід між екранами входу, реєстрації та створення паролю. Це сприяє поліпшенню користувацького досвіду та забезпеченню безпеки додатку.

   AuthStack містить такі сторінки:

   – сторінка входу;
   – сторінка введення даних користувача;
   – сторінка створення паролю користувача.

2. HomeStack – керує потоком аутентифікованого користувача, забезпечуючи переходи між вкладками, налаштуваннями, створенням та редагуванням пінів.

   HomeStack містить такі сторінки:

   – TabsNavigator – інтеграція навігації вкладок для основних екранів, таких як «Map» та «Pins». Забезпечує доступ до основних функцій додатку;
   – сторінка створення піна;
   – сторінка редагування піна;
   – сторінка налаштувань.

Висновок

Розглянуто ключові аспекти розробки мобільного додатку, включаючи вибір технологій, архітектуру додатку, базу даних, навігацію та інші важливі елементи.

Використання TypeScript як мови програмування сприяло підвищенню надійності та чистоти коду за рахунок строгої типізації та об'єктно-орієнтованих можливостей. Фреймворк React Native дозволив розробляти кросплатформовий додаток, що забезпечує консистентний користувацький досвід на різних пристроях.

З використанням Redux-Toolkit для управління станом додатку досягнуто високої організованості та ефективності управління даними. Firebase Realtime Database забезпечила надійне зберігання даних та їх синхронізацію в реальному часі, в той час як Flipper використовувався для оптимізації продуктивності та виявлення помилок.

Архітектура додатку була ретельно спланована, що забезпечило гнучкість та масштабованість додатку, а також сприяло легкості внесення змін і підтримки в майбутньому. Навігація додатку, реалізована за допомогою React Navigation, забезпечила плавне пересування між різними екранами та функціями.

Загалом, ретельний вибір технологій та методів розробки сприяв створенню стабільного, продуктивного та інтуїтивно зрозумілого мобільного додатку. Використання сучасних підходів та інструментів у розробці дозволило досягнути основних цілей проекту та забезпечити високий рівень задоволення користувачів.

ВИСНОВКИ

В кваліфікаційної роботи було розроблено мобільний додаток для ведення нотаток, інтегрований з Google Maps, з акцентом на ефективності та безпеку доступу до даних. Додаток, реалізований за допомогою React Native, використовує сучасні технології, включаючи TypeScript, React Native Expo, Redux-Toolkit та Firebase Realtime Database, доповнені інструментом відлагодження Flipper.

Однією з ключових особливостей додатку є розширене управління доступом до даних, яке забезпечується через правила Firebase, що дозволяє розділити потоки для авторизованих та неавторизованих користувачів. Це забезпечує, що конфіденційність інформації користувача захищена, а доступ до даних контролюється та обмежується залежно від статусу авторизації.

Додаток дозволяє користувачам створювати, редагувати та організовувати нотатки, які прив'язані до специфічних місць на карті. Інтеграція з Google Maps гарантує точність геолокаційних даних, а функції пошуку, відзначення улюблених локацій та персоналізації інтерфейсу підвищують зручність користування.

Завдяки єдиній кодовій базі, додаток забезпечує високу продуктивність на різних платформах, включаючи iOS та Android, що є важливою перевагою кросплатформового підходу. Це сприяє економії ресурсів під час розробки та підтримки, а також гарантує широкий доступ до додатку для різних груп користувачів.

Загалом, цей проект підкреслює важливість інтеграції сучасних технологій у мобільні додатки, з особливим акцентом на безпеку та конфіденційність даних, відкриваючи нові можливості для ефективного використання мобільних застосунків у повсякденному житті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Atencio, L.: "Functional Programming in JavaScript". – Manning Publications – 2020 – P.184-190.
2. Bennett, F.: "Pro TypeScript: Application-Scale JavaScript Development". – Apress – 2019 – P. 134-138.
3. Boduch, J.: "React and React Native: A complete hands-on guide to modern web and mobile development with React.js, 3rd Edition". – Packt Publishing – 2020 – P. 142.
4. Brown, M.: "Advanced Mobile App Development: Bridging Native and Web Applications". – Packt Publishing – 2020 – P. 164-169.
5. Carmine Zaccagnino: Programming Flutter: Native, Cross-Platform Apps the Easy Way. 1st Ed. – Chapter 1 – 2020 – P.11-12
6. Cross-Platform Mobile Development Best Practices – [Електронний ресурс] – Режим доступу: https://www.toptal.com/cross-platform/best-practices
7. Ellis, B.: "React Native in Action". – Manning Publications – 2018 – P. 73.
8. Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/database
9. Flipper: A React Native Debugger – [Електронний ресурс] – Режим доступу: https://fbflipper.com/
10. Friedman, A.: "Modern JavaScript: Develop and Design". – Peachpit Press – 2019 – P. 88-92.
11. Front-end Frameworks – [Електронний ресурс] – Режим доступу: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/
12. Hahn, B.: "Learning Redux". – Packt Publishing – 2018 – P. 18-21
13. Johnson, K.: "Effective Mobile Interface Design". – Wiley – 2020 – P. 45.
14. Mobile App Performance Optimization – [Електронний ресурс] – Режим доступу: https://developer.android.com/topic/performance

15. Mobile UI/UX Design Trends – [Електронний ресурс] – Режим доступу: https://www.smashingmagazine.com/2020/08/mobile-ui-ux-design-trends/
16. Moore, S.: "Flutter for Beginners: An Introductory Guide to Building Cross-Platform Mobile Applications". – Apress – 2021 – P. 76-80.
17. Noring, L.: "Fullstack React Native: The Complete Guide to React Native". – Fullstack.io – 2018 – P. 223.
18. Platform architecture – [Електронний ресурс] – Режим доступу: https://developer.android.com/guide/platform
19. React Lifecycle of Components – [Електронний ресурс] - https://www.geeksforgeeks.org/reactjs-lifecycle-components/
20. React Native Documentation – [Електронний ресурс] – Режим доступу: https://reactnative.dev/docs/getting-started
21. Redux Toolkit Essentials – [Електронний ресурс] – Режим доступу: https://redux-toolkit.js.org/introduction/getting-started
22. TypeScript: Scalable JavaScript Development – [Електронний ресурс] – Режим доступу: https://www.typescriptlang.org/docs/
23. Wargo, J. M.: "Practical Cross-Platform Mobile App Development". – Addison-Wesley – 2020 – P. 17.
24. Williams, J.: "Advanced React Native: Building Responsive and Performant Apps". – O'Reilly Media – 2021 – P. 56-60.
25. Wilson, T.: "Mastering React Native". – Packt Publishing – 2019 – P. 127.
26. White, R.: "Understanding React Hooks: A Comprehensive Guide". – O'Reilly Media – 2021 – P. 65-70.
27. Willis, S.: "iOS App Development Essentials: Concepts and Techniques". – Apress – 2020 – P. 78-83.
28. Xavier, P.: "Xamarin Mobile Application Development: From Concept to Market". – Packt Publishing – 2019 – P. 91-95.
29. Young, L.: "Apache Cordova in Action: Building Native Mobile Apps". – Manning Publications – 2018 – P. 102-106.

Додаток А

КОД ПРОГРАМИ

Файл App.tsx

```
import { useFonts } from "expo-font";
import React from "react";
import "react-native-gesture-handler";
import { Provider } from "react-redux";

import { FontWeightAliases } from "./constants";
import { useSplashScreen } from "./hooks/";
import AppRoutes from "./navigation/App.routes";
import store from "./store/redux/store";

export default function App() {
  const [isFontsLoaded] = useFonts({
    [FontWeightAliases.Bold]: require("./assets/fonts/Montserrat-Bold.ttf"),
    [FontWeightAliases.SemiBold]: require("./assets/fonts/Montserrat-
SemiBold.ttf"),
    [FontWeightAliases.Medium]: require("./assets/fonts/Montserrat-Medium.ttf"),
    [FontWeightAliases.Regular]: require("./assets/fonts/Montserrat-
Regular.ttf"),
  });

  const { SplashScreenContainer, onContentReady } =
    useSplashScreen(isFontsLoaded);

  return (
    <SplashScreenContainer>
      <Provider store={store}>
        <AppRoutes onReady={onContentReady} />
      </Provider>
    </SplashScreenContainer>
  );
}
```

Файл Box.tsx

```
import { FC } from "react";
import { View, ViewProps } from "react-native";

import { IAppColors } from "../../../constants/themes/types";
import { useAppTheme } from "../../../hooks";

export interface IBoxProps extends ViewProps {
  backgroundColor?: keyof IAppColors;
  borderColor?: keyof IAppColors;
}

export const Box: FC<IBoxProps> = ({
  backgroundColor,
  borderColor,
  style,
  ...rest
}) => {
  const { appColors } = useAppTheme();

  return (
    <View
```

```
      style={[
        backgroundColor && { backgroundColor: appColors[backgroundColor] },
        borderColor && { borderColor: appColors[borderColor] },
        style,
      ]}
      {...rest}
    />
  );
};
```

Файл AppRoutes.tsx

```
import { NavigationContainer } from "@react-navigation/native";
import { StatusBar } from "expo-status-bar";
import { useEffect } from "react";
import FlashMessage from "react-native-flash-message";

import {
  useAppTheme,
  useFirebaseAutoLogin,
  usePins,
  useSettings,
} from "../hooks";
import AuthStack from "./AuthStack";
import HomeStack from "./HomeStack";

type AppRoutesProps = {
  onReady: () => void;
};

const AppRoutes: React.FC<AppRoutesProps> = ({ onReady }) => {
  const { isAuthenticated, credentials } = useFirebaseAutoLogin();
  const { theme, statusBarStyle } = useAppTheme();
  const { fetchSettings } = useSettings();
  const { fetchPins } = usePins();

  useEffect(() => {
    fetchSettings().then(onReady);
    fetchPins();
  }, [credentials]);

  return (
    <>
      <StatusBar style={statusBarStyle} />
      <FlashMessage />
      <NavigationContainer theme={theme}>
        {isAuthenticated ? <HomeStack /> : <AuthStack />}
      </NavigationContainer>
    </>
  );
};

export default AppRoutes;
```

Файл HomeStack.tsx

```
import { StackNavigationProp, StackScreenProps } from "@react-navigation/stack";

import { IPinModel } from "../../types/models";

export type HomeStackParamList = {
  Home: undefined;
  AddPin: undefined;
  EditPin: { pin: IPinModel };
```

```
  Settings: undefined;
};

export type HomeScreenProps = StackScreenProps<HomeStackParamList>;

export type HomeScreenNavigationProp = StackNavigationProp<HomeStackParamList>;
import { createStackNavigator } from "@react-navigation/stack";

import { HeaderBackImage, HeaderTitle } from "../../components/sections";
import { useAppTheme } from "../../hooks";
import {
  AddPinScreen,
  EditPinScreen,
  SettingsScreen,
} from "../../screens/Home";
import TabsStack from "../TabStack";
import { HomeStackParamList } from "./types";

const Stack = createStackNavigator<HomeStackParamList>();

const HomeStack: React.FC = () => {
  const { appColors } = useAppTheme();

  return (
    <Stack.Navigator
      initialRouteName="Tabs"
      screenOptions={{
        headerShadowVisible: false,
        headerTitleAlign: "center",
        headerStyle: { backgroundColor: appColors.background },
        headerTitle: HeaderTitle,
        headerBackImage: HeaderBackImage,
      }}
    >
      <Stack.Screen
        name="Tabs"
        component={TabsStack}
        options={{
          headerShown: false,
        }}
      />

      <Stack.Screen
        name="AddPin"
        component={AddPinScreen}
        options={{
          title: "Add pin",
        }}
      />

      <Stack.Screen
        name="EditPin"
        component={EditPinScreen}
        options={{
          title: "Edit pin",
        }}
      />

      <Stack.Screen name="Settings" component={SettingsScreen} />
    </Stack.Navigator>
  );
};
```

```
export default HomeStack;
```

Файл AuthStack.tsx

```
import { StackScreenProps } from "@react-navigation/stack";

export type AuthStackParamList = {
  Startup: undefined;
  RegistrationStartup: undefined;
  RegistrationCompletion: { name: string; email: string };
  Login: undefined | { email: string };
};

export type AuthScreenProps = StackScreenProps<AuthStackParamList>;

import { createStackNavigator } from "@react-navigation/stack";
import React from "react";

import { HeaderBackImage, HeaderTitle } from "../../components/sections";
import {
  LoginScreen,
  RegistrationCompletionScreen,
  RegistrationStartupScreen,
  StartupScreen,
} from "../../screens/Auth";
import { AuthStackParamList } from "./types";

const Stack = createStackNavigator<AuthStackParamList>();

const AuthStack: React.FC = () => (
  <Stack.Navigator
    initialRouteName="Startup"
    screenOptions={{
      headerShadowVisible: false,
      headerTitleAlign: "center",
      headerTitle: HeaderTitle,
      headerBackImage: HeaderBackImage,
    }}
  >
    <Stack.Screen
      name="Startup"
      component={StartupScreen}
      options={{
        headerShown: false,
      }}
    />

    <Stack.Screen
      name="RegistrationStartup"
      component={RegistrationStartupScreen}
      options={{
        title: "Create an account",
      }}
    />

    <Stack.Screen
      name="RegistrationCompletion"
      component={RegistrationCompletionScreen}
      options={{
        title: "Create an account",
      }}
    />
```

```
    <Stack.Screen
      name="Login"
      component={LoginScreen}
      options={{
        title: "Log in",
      }}
    />
  </Stack.Navigator>
);

export default AuthStack;
```

Файл Tabs.tsx

```
import { BottomTabScreenProps } from "@react-navigation/bottom-tabs";

import { IPinModel } from "../../types/models";

export type TabStackParamList = {
  Map: undefined | { pin: IPinModel };
  Pins: undefined;
};

export type TabProps = BottomTabScreenProps<TabStackParamList>;

import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
import { useNavigation } from "@react-navigation/native";
import { useState } from "react";

import { MAP_ICON, PIN_ICON } from "../../assets/icons";
import { Icon } from "../../components/common";
import { ConfirmModal } from "../../components/modals";
import { SearchBar } from "../../components/sections";
import { textStyle_i3 } from "../../constants";
import { typographyStyleToTextStyle } from "../../helpers";
import { useAppTheme, useAuth } from "../../hooks";
import { MapScreen, PinsScreen } from "../../screens/Home";
import { HomeScreenNavigationProp } from "../HomeStack/types";
import styles from "./styles";
import { TabStackParamList } from "./types";

const Tabs = createBottomTabNavigator<TabStackParamList>();

const TabsStack: React.FC = () => {
  const navigation = useNavigation<HomeScreenNavigationProp>();
  const { appColors } = useAppTheme();
  const { signOut } = useAuth();
  const [isLogoutDialogOpened, setIsLogoutDialogOpened] = useState(false);

  return (
    <>
      <ConfirmModal
        title="Log Out"
        description="Are you sure you want to logout?"
        confirmText="Log Out"
        visible={isLogoutDialogOpened}
        onConfirm={() => {
          setIsLogoutDialogOpened(false);
          signOut();
        }}
        onCancel={() => {
          setIsLogoutDialogOpened(false);
```

```
        }}
      />

      <Tabs.Navigator
        initialRouteName="Map"
        screenOptions={{
          tabBarActiveBackgroundColor: appColors.variant,
          tabBarLabelPosition: "beside-icon",
          tabBarLabelStyle: typographyStyleToTextStyle(textStyle_i3, appColors),
          header: () => (
            <SearchBar
              style={styles.searchBarContainer}
              onLeftButtonPress={() => navigation.navigate("Settings")}
              onRightButtonPress={() => {
                setIsLogoutDialogOpened(true);
              }}
            />
          ),
        }}
      >
        <Tabs.Screen
          name="Map"
          component={MapScreen}
          options={{
            tabBarIcon: () => <Icon tintColor="primary" source={MAP_ICON} />,
          }}
        />

        <Tabs.Screen
          name="Pins"
          component={PinsScreen}
          options={{
            tabBarIcon: () => <Icon tintColor="primary" source={PIN_ICON} />,
          }}
        />
      </Tabs.Navigator>
    </>
  );
};

export default TabsStack;
```

Файл AddPinScreen.tsx

```
import { FC } from "react";
import { View } from "react-native";
import { LatLng } from "react-native-maps";

import { SAVE_ICON } from "../../../assets/icons";
import {
  PinForm,
  SelectLocationView,
  Separator,
} from "../../../components/sections";
import { DEFAULT_REGION } from "../../../constants";
import { pinFormToModel } from "../../../converters";
import {
  useHeaderRightButton,
  useHookForm,
  usePins,
  useUserLocation,
} from "../../../hooks";
import { HomeScreenProps } from "../../../navigation/HomeStack/types";
```

```
import { IPinForm } from "../../../types/forms";
import styles from "./styles";

export const AddPinScreen: FC<HomeScreenProps> = ({ navigation }) => {
  const { createPin } = usePins();

  const { userLocation } = useUserLocation();

  const { formController, watch, setValue, handleSubmit } =
    useHookForm<IPinForm>({
      defaultValues: {
        latitude: "",
        longitude: "",
        description: "",
      },
    });

  const setCoordinates = ({ latitude, longitude }: LatLng) => {
    setValue("latitude", String(latitude));
    setValue("longitude", String(longitude));

    formController.trigger("latitude");
    formController.trigger("longitude");
  };

  const savePinHandler = async (pinForm: IPinForm) => {
    const newPin = pinFormToModel(pinForm);

    const isPinCreated = await createPin(newPin);

    if (isPinCreated && navigation.canGoBack()) {
      navigation.goBack();
    }
  };

  const latitude = Number.parseFloat(watch("latitude"));
  const longitude = Number.parseFloat(watch("longitude"));

  useHeaderRightButton(navigation, SAVE_ICON, handleSubmit(savePinHandler));

  return (
    <>
      <Separator />

      <View style={styles.container}>
        <PinForm formController={formController} />

        <SelectLocationView
          initialRegion={{ ...DEFAULT_REGION, ...userLocation }}
          shouldRequestLocationInitially={false}
          latitude={latitude}
          longitude={longitude}
          onPickLocation={setCoordinates}
        />
      </View>
    </>
  );
};
```

Файл EditPinScreen.tsx

```
import { FC, useEffect } from "react";
import { View } from "react-native";
```

```
import { LatLng } from "react-native-maps";

import { SAVE_ICON } from "../../../assets/icons";
import {
  PinForm,
  SelectLocationView,
  Separator,
} from "../../../components/sections";
import { DEFAULT_REGION } from "../../../constants";
import { pinFormToModel } from "../../../converters";
import { useHeaderRightButton, useHookForm, usePins } from "../../../hooks";
import { HomeScreenProps } from "../../../navigation/HomeStack/types";
import { IPinForm } from "../../../types/forms";
import { IPinModel } from "../../../types/models";
import styles from "./styles";

export const EditPinScreen: FC<HomeScreenProps> = ({ navigation, route }) => {
  const { updatePin } = usePins();

  const editedPin = route.params?.pin;

  const { formController, watch, setValue, handleSubmit } =
    useHookForm<IPinForm>({
      defaultValues: {
        latitude: editedPin?.location.latitude.toString(),
        longitude: editedPin?.location.longitude.toString(),
      },
    });

  const setCoordinates = ({ latitude, longitude }: LatLng) => {
    setValue("latitude", String(latitude));
    setValue("longitude", String(longitude));

    formController.trigger("latitude");
    formController.trigger("longitude");
  };

  const savePinHandler = async (pinForm: IPinForm) => {
    if (editedPin) {
      const pinToUpdate: IPinModel = {
        ...pinFormToModel(pinForm),
        id: editedPin.id,
        isFavorite: editedPin.isFavorite,
      };

      const isPinUpdated = await updatePin(pinToUpdate);

      if (isPinUpdated && navigation.canGoBack()) {
        navigation.goBack();
      }
    }
  };

  const latitude = Number.parseFloat(watch("latitude"));
  const longitude = Number.parseFloat(watch("longitude"));

  useEffect(() => {
    if (editedPin) {
      setValue("label", editedPin.label);
      setValue("description", editedPin.description ?? "");
      setValue("latitude", editedPin.location.latitude.toString());
      setValue("longitude", editedPin.location.longitude.toString());
```

```
    }
  }, [editedPin]);

  useHeaderRightButton(navigation, SAVE_ICON, handleSubmit(savePinHandler));

  return (
    <>
      <Separator />

      <View style={styles.container}>
        <PinForm formController={formController} />

        <SelectLocationView
          initialRegion={{
            ...DEFAULT_REGION,
            latitude: editedPin?.location.latitude ?? 0,
            longitude: editedPin?.location.longitude ?? 0,
          }}
          latitude={latitude}
          longitude={longitude}
          shouldRequestLocationInitially={false}
          onPickLocation={setCoordinates}
        />
      </View>
    </>
  );
};
```

Файл SettingsScreen.tsx

```
import { FC } from "react";

import { IOptions, ToggleOptionPicker } from "../../../components/common";
import { Separator } from "../../../components/sections";
import { AppThemes } from "../../../enums";
import { useAppTheme, useSettings } from "../../../hooks";
import { HomeScreenProps } from "../../../navigation/HomeStack/types";

const THEME_OPTIONS: IOptions<AppThemes> = [
  {
    title: "Light Theme",
    value: AppThemes.Light,
  },
  {
    title: "Dark Theme",
    value: AppThemes.Dark,
  },
];

export const SettingsScreen: FC<HomeScreenProps> = () => {
  const { currentTheme } = useAppTheme();
  const { updateAppTheme } = useSettings();

  return (
    <>
      <Separator />

      <ToggleOptionPicker
        value={currentTheme}
        items={THEME_OPTIONS}
        onValueChanged={updateAppTheme}
      />
    </>
```

```
  );
};
```

Файл MapScreen.tsx

```
import React, { FC, useCallback, useEffect, useRef, useState } from "react";
import { View } from "react-native";
import ClusteredMap from "react-native-map-clustering";
import MapView from "react-native-maps";
import { useSelector } from "react-redux";

import {
  LOCATION_ICON,
  MARKER_GRAY_ICON,
  MARKER_ICON,
} from "../../../assets/icons";
import { CustomMarker, IconButton } from "../../../components/common";
import { PinDetailsModal } from "../../../components/sections";
import { DEFAULT_REGION } from "../../../constants";
import { IconButtonStyles } from "../../../constants/styles";
import {
  customMarkerModelToPinModel,
  pinItemModelToPinModel,
  pinModelToCustomMarkerModel,
} from "../../../converters";
import {
  animateToLocation,
  hideMarkerCalloutById,
  showMarkerCalloutById,
} from "../../../helpers";
import { useAppTheme, usePins, useUserLocation } from "../../../hooks";
import { TabProps } from "../../../navigation/TabStack/types";
import {
  setUserLocationAction,
  stopSearchAction,
} from "../../../store/redux/actions";
import { selectSearch } from "../../../store/redux/slices";
import { useAppDispatch } from "../../../store/redux/store";
import { ICustomMarkerModel, IPinItemModel } from "../../../types/components";
import { IPinModel } from "../../../types/models";
import { FoundPinsList } from "./components/FoundPinsList";
import styles from "./styles";

export const MapScreen: FC<TabProps> = ({ navigation, route }) => {
  const { appColors, mapStyles } = useAppTheme();
  const mapViewRef = useRef<MapView | null>(null);

  const dispatch = useAppDispatch();
  const { userLocation, requestUserLocation } = useUserLocation(true);
  const { getPinsBySearchQuery, pins } = usePins();
  const { searchQuery } = useSelector(selectSearch);

  const [markers, setMarkers] = useState<ICustomMarkerModel[]>([]);
  const [foundPins, setFoundPins] = useState<Array<IPinModel>>([]);
  const [selectedPin, setSelectedPin] = useState<IPinModel | null>(null);

  const showPinDetails = (pin: IPinModel) => {
    setSelectedPin(pin);

    animateToLocation(mapViewRef, pin.location);
    showMarkerCalloutById(markers, pin.id);
  };
```

```
const markerPressHandler = useCallback((pin: ICustomMarkerModel) => {
  setSelectedPin(customMarkerModelToPinModel(pin));
}, []);

const pinFoundPressHandler = useCallback(
  (pinItem: IPinItemModel) => {
    dispatch(stopSearchAction());

    showPinDetails(pinItemModelToPinModel(pinItem));
  },
  [mapViewRef]
);

const hidePinDetailsHandler = () => {
  hideMarkerCalloutById(markers, selectedPin?.id ?? "");
  setSelectedPin(null);
};

const regionChangeCompleteHandler = () => {
  if (selectedPin) {
    setTimeout(() => {
      showMarkerCalloutById(markers, selectedPin.id);
    }, 0);
  }
};

useEffect(() => {
  dispatch(setUserLocationAction(userLocation));
  animateToLocation(mapViewRef, userLocation);
}, [userLocation]);

useEffect(() => {
  const markers = pins.map(
    (pin): ICustomMarkerModel => ({
      ...pinModelToCustomMarkerModel(pin),
      icon: pin.isFavorite ? MARKER_ICON : MARKER_GRAY_ICON,
    })
  );

  setMarkers(markers);
}, [pins]);

useEffect(() => {
  setFoundPins(getPinsBySearchQuery(searchQuery));
}, [searchQuery, pins]);

useEffect(() => {
  if (route.params?.pin) {
    const { pin, ...restParams } = route.params;
    navigation.setParams(restParams);

    showPinDetails(pin);
  }
}, [route.params?.pin]);

return (
  <View style={styles.container}>
    {searchQuery && (
      <FoundPinsList pins={foundPins} onPinPressed={pinFoundPressHandler} />
    )}

    <ClusteredMap
```

```
        style={styles.map}
        customMapStyle={mapStyles}
        initialRegion={
          userLocation ? { ...DEFAULT_REGION, ...userLocation } : DEFAULT_REGION
        }
        clusterColor={appColors.primary}
        clusterTextColor={appColors.variant}
        showsUserLocation
        showsMyLocationButton={false}
        moveOnMarkerPress={false}
        ref={mapViewRef}
        onRegionChangeComplete={regionChangeCompleteHandler}
      >
        {markers.map((pin) => (
          <CustomMarker
            key={pin.key}
            marker={pin}
            coordinate={pin.location}
            onPress={markerPressHandler}
          />
        ))}
      </ClusteredMap>

      <IconButton
        style={IconButtonStyles.floating}
        backgroundColor="background"
        tintColor="primary"
        imageSource={LOCATION_ICON}
        onPress={requestUserLocation}
      />

      {selectedPin && (
        <PinDetailsModal pin={selectedPin} onClose={hidePinDetailsHandler} />
      )}
    </View>
  );
};
```

Файл FoundPin.tsx

```
import React, { FC } from "react";
import { TouchableOpacity, View } from "react-native";

import { PIN_ICON } from "../../../../../assets/icons";
import { Icon, Typography } from "../../../../../components/common";
import { textStyle_i11, textStyle_i13 } from "../../../../../constants";
import { IPinItemModel } from "../../../../../types/components";
import styles from "./styles";

type FoundPinProps = {
  pin: IPinItemModel;
  onPress: (pin: IPinItemModel) => void;
};

export const FoundPin: FC<FoundPinProps> = React.memo(({ pin, onPress }) => {
  return (
    <TouchableOpacity style={styles.container} onPress={() => onPress(pin)}>
      <Icon
        tintColor={pin.isFavorite ? "primary" : "systemGray"}
        source={PIN_ICON}
      />

      <View>
```

```
        <Typography
          style={textStyle_i13}
          lineBreakMode="tail"
          numberOfLines={1}
        >
          {pin.label}
        </Typography>

        {/* TODO: show actual address */}
        <Typography
          style={textStyle_i11}
          lineBreakMode="tail"
          numberOfLines={1}
        >
          "Via Alessandro Solivetti, 17, 00168 Roma..."
        </Typography>
      </View>
    </TouchableOpacity>
  );
});
```

Файл FoundPinsList.tsx

```
import { FC, useEffect, useState } from "react";
import { ViewStyle } from "react-native";
import { FlatList } from "react-native-gesture-handler";

import { Typography } from "../../../../../components/common";
import { Separator } from "../../../../../components/sections";
import {
  DISPLAYED_PINS_SEARCH_RESULTS_MAX_COUNT,
  textStyle_i13,
} from "../../../../../constants";
import { pinModelToPinItemModel } from "../../../../../converters";
import { useAppTheme } from "../../../../../hooks";
import { IPinItemModel } from "../../../../../types/components";
import { IPinModel } from "../../../../../types/models";
import { FoundPin } from "../FoundPin";
import { FOUND_PIN_HEIGHT } from "../FoundPin/styles";
import styles from "./styles";

const emptyListView = () => (
  <Typography style={[textStyle_i13, styles.nothingFoundText]}>
    Nothing found
  </Typography>
);

type FoundPinsListProps = {
  pins: Array<IPinModel>;
  onPinPressed: (pin: IPinItemModel) => void;
};

export const FoundPinsList: FC<FoundPinsListProps> = ({
  pins,
  onPinPressed,
}) => {
  const { appColors } = useAppTheme();
  const [displayedPins, setDisplayedPins] = useState<IPinItemModel[]>([]);
  const [pinsListStyle, setPinsListStyle] = useState<ViewStyle[]>();

  useEffect(() => {
    const displayedPins: IPinItemModel[] = pins.map((x) =>
      pinModelToPinItemModel(x)
```

```
    );

    setDisplayedPins(displayedPins);
  }, [pins]);

  useEffect(() => {
    const pinsListStyle: ViewStyle[] = [
      {
        height:
          pins.length > DISPLAYED_PINS_SEARCH_RESULTS_MAX_COUNT
            ? FOUND_PIN_HEIGHT * DISPLAYED_PINS_SEARCH_RESULTS_MAX_COUNT
            : "auto",
      },
    ];

    setPinsListStyle(pinsListStyle);
  }, [pins.length]);

  return (
    <FlatList
      style={[
        styles.container,
        { backgroundColor: appColors.background },
        pinsListStyle,
      ]}
      keyboardShouldPersistTaps="always"
      data={displayedPins}
      ItemSeparatorComponent={() => <Separator />}
      ListEmptyComponent={emptyListView}
      renderItem={({ item }) => <FoundPin pin={item} onPress={onPinPressed} />}
    />
  );
};
```

Файл PinsScreen.tsx

```
import { useNavigation } from "@react-navigation/native";
import React, { FC, useCallback, useEffect, useState } from "react";
import { ListRenderItemInfo, View } from "react-native";
import { RefreshControl } from "react-native-gesture-handler";
import { RowMap, SwipeListView } from "react-native-swipe-list-view";
import { useSelector } from "react-redux";

import { PLUS_ICON } from "../../../assets/icons";
import { IconButton } from "../../../components/common";
import { ConfirmModal } from "../../../components/modals";
import { EmptyView } from "../../../components/sections";
import { IconButtonStyles } from "../../../constants/styles";
import {
  pinItemModelToPinModel,
  pinModelToPinItemModel,
} from "../../../converters";
import { hideActionMenu } from "../../../helpers";
import { usePins } from "../../../hooks";
import { HomeScreenNavigationProp } from "../../../navigation/HomeStack/types";
import { TabProps } from "../../../navigation/TabStack/types";
import { stopSearchAction } from "../../../store/redux/actions";
import { selectSearch } from "../../../store/redux/slices";
import { useAppDispatch } from "../../../store/redux/store";
import { IPinItemModel } from "../../../types/components";
import { IPinModel } from "../../../types/models";
import {
  PIN_ACTION_MENU_WIDTH,
```

```
  PinActionMenu,
} from "./components/PinActionMenu";
import { PinItem } from "./components/PinItem";
import styles from "./styles";

export const PinsScreen: FC<TabProps> = ({ navigation }) => {
  const dispatch = useAppDispatch();
  const homeNavigation = useNavigation<HomeScreenNavigationProp>();

  const {
    pins,
    isPinsLoading,
    fetchPins,
    getPinsBySearchQuery,
    togglePinFavoriteStatus,
    deletePin,
  } = usePins();
  const { searchQuery } = useSelector(selectSearch);

  const [selectedPinRow, setSelectedPinRow] = useState<RowMap<IPinItemModel>>();
  const [selectedPinId, setSelectedPinId] = useState<string>();
  const [pinsItems, setPinsList] = useState<IPinItemModel[]>([]);
  const [isRemovePinConfirmationShown, setIsRemovePinConfirmationVisible] =
    useState(false);

  const togglePinFavoriteStatusHandler = useCallback((pin: IPinItemModel) => {
    const pinData: IPinModel = pinItemModelToPinModel(pin);

    togglePinFavoriteStatus(pinData);
  }, []);

  const pinPressedHandler = useCallback(
    (pin: IPinItemModel) => {
      dispatch(stopSearchAction());

      const pinData: IPinModel = pinItemModelToPinModel(pin);

      navigation.navigate("Map", { pin: pinData });
    },
    [navigation]
  );

  const deletePinHandler = (
    { item: pin }: ListRenderItemInfo<IPinItemModel>,
    row: RowMap<IPinItemModel>
  ) => {
    setSelectedPinId(pin.key);
    setSelectedPinRow(row);

    setIsRemovePinConfirmationVisible(true);
  };

  const confirmDeletePinHandler = () => {
    if (selectedPinId) {
      deletePin(selectedPinId);
    }

    setIsRemovePinConfirmationVisible(false);
  };

  const cancelDeletePinHandler = () => {
    setIsRemovePinConfirmationVisible(false);
```

```
  hideActionMenu(selectedPinRow, selectedPinId);
};

const editPinHandler = (
  { item: pin }: ListRenderItemInfo<IPinItemModel>,
  row: RowMap<IPinItemModel>
) => {
  hideActionMenu(row, pin.key);

  homeNavigation.navigate("EditPin", { pin: pinItemModelToPinModel(pin) });
};

const addPinHandler = () => homeNavigation.navigate("AddPin");

const renderPinItem = useCallback(
  ({ item: pin }: ListRenderItemInfo<IPinItemModel>) => (
    <PinItem
      pin={pin}
      onPress={pinPressedHandler}
      onPressFavoriteStatus={togglePinFavoriteStatusHandler}
    />
  ),
  [pinPressedHandler, togglePinFavoriteStatus]
);

const renderHiddenActionMenu = useCallback(
  (pin: ListRenderItemInfo<IPinItemModel>, row: RowMap<IPinItemModel>) => (
    <PinActionMenu
      onDelete={() => deletePinHandler(pin, row)}
      onEdit={() => editPinHandler(pin, row)}
    />
  ),
  []
);

useEffect(() => {
  const filteredPins = searchQuery ? getPinsBySearchQuery(searchQuery) : pins;
  const newPinsList = filteredPins.map((x) => pinModelToPinItemModel(x));

  setPinsList(newPinsList);
}, [pins, searchQuery]);

return (
  <View style={styles.container}>
    <ConfirmModal
      visible={isRemovePinConfirmationShown}
      title="Pin will be removed"
      description="Are you sure?"
      onConfirm={confirmDeletePinHandler}
      onCancel={cancelDeletePinHandler}
    />

    <SwipeListView
      keyboardShouldPersistTaps="always"
      refreshControl={
        <RefreshControl refreshing={isPinsLoading} onRefresh={fetchPins} />
      }
      onRowOpen={setSelectedPinId}
      onRowClose={setSelectedPinId}
      data={pinsItems}
      contentContainerStyle={
```

```
          pinsItems.length === 0 && styles.emptyListContainer
        }
        ListEmptyComponent={() => (
          <EmptyView>
            {searchQuery ? "Nothing found" : "There are no added pins yet"}
          </EmptyView>
        )}
        renderItem={renderPinItem}
        renderHiddenItem={renderHiddenActionMenu}
        stopLeftSwipe={60}
        stopRightSwipe={-200}
        rightOpenValue={-PIN_ACTION_MENU_WIDTH}
      />

      <IconButton
        style={IconButtonStyles.floating}
        backgroundColor="primary"
        imageSource={PLUS_ICON}
        onPress={addPinHandler}
      />
    </View>
  );
};
```

Файл SearchBar.tsx

```
import { FC, useEffect, useRef } from "react";
import { TextInput, View, ViewStyle } from "react-native";
import { useSelector } from "react-redux";

import {
  CLEAR_ICON,
  EXIT_ICON,
  LEFT_BLUE_ICON,
  SETTINGS_ICON,
} from "../../../assets/icons";
import { AppPalette, ImageSizes, textStyle_i13 } from "../../../constants";
import { typographyStyleToTextStyle } from "../../../helpers";
import { useAppTheme } from "../../../hooks";
import {
  setSearchQueryAction,
  startSearchAction,
  stopSearchAction,
} from "../../../store/redux/actions";
import { selectSearch } from "../../../store/redux/slices";
import { useAppDispatch } from "../../../store/redux/store";
import { Box, IconButton } from "../../common";
import { Separator } from "../Separator";
import styles from "./styles";

interface ISearchBarProps {
  style?: ViewStyle;
  onLeftButtonPress?: () => void;
  onRightButtonPress?: () => void;
}

export const SearchBar: FC<ISearchBarProps> = ({
  style,
  onLeftButtonPress,
  onRightButtonPress,
}) => {
  const { appColors } = useAppTheme();
```

```
const dispatch = useAppDispatch();
const { isActive, searchQuery } = useSelector(selectSearch);
const textInputRef = useRef<TextInput | null>(null);

const pinsSearchQueryChangeHandler = (text: string) => {
  dispatch(setSearchQueryAction(text));
};

const clearTextHandler = () => {
  dispatch(setSearchQueryAction(""));
};

const focusHandler = () => {
  dispatch(startSearchAction());
};

const stopSearchHandler = () => {
  textInputRef.current?.blur();

  dispatch(stopSearchAction());
};

useEffect(() => {
  if (!isActive) {
    textInputRef.current?.blur();
  }
}, [isActive]);

return (
  <>
    <View style={[styles.container, style]}>
      {isActive ? (
        <IconButton
          tintColor="primary"
          imageStyle={ImageSizes.medium}
          imageSource={LEFT_BLUE_ICON}
          onPress={stopSearchHandler}
        />
      ) : (
        <IconButton
          tintColor="primary"
          imageSource={SETTINGS_ICON}
          onPress={onLeftButtonPress}
        />
      )}

      <Box
        borderColor="variant"
        backgroundColor="variant"
        style={styles.inputContainer}
      >
        <TextInput
          ref={textInputRef}
          style={[
            styles.input,
            typographyStyleToTextStyle(textStyle_i13, appColors),
          ]}
          placeholder="Search"
          placeholderTextColor={AppPalette.systemGray}
          cursorColor={appColors.primary}
          selectionColor={appColors.primary}
          value={searchQuery}
```

```
            onFocus={focusHandler}
            onChangeText={pinsSearchQueryChangeHandler}
          />

          {isActive && (
            <IconButton
              imageStyle={ImageSizes.large}
              tintColor="primary"
              imageSource={CLEAR_ICON}
              onPress={clearTextHandler}
            />
          )}
        </Box>

        {!isActive && (
          <IconButton
            tintColor="primary"
            imageSource={EXIT_ICON}
            onPress={onRightButtonPress}
          />
        )}
      </View>

      <Separator />
    </>
  );
};
```

Файл StartupScreen.tsx

```
import React from "react";
import { Image, View } from "react-native";

import { ENTER_PAGE_PIC } from "../../../assets/icons";
import { CustomButton, Typography } from "../../../components/common";
import {
  CustomButtonStyles,
  textStyle_i1,
  textStyle_i4,
  textStyle_i5,
} from "../../../constants";
import { AuthScreenProps } from "../../../navigation/AuthStack/types";
import styles from "./styles";

export const StartupScreen: React.FC<AuthScreenProps> = ({ navigation }) => {
  const loginHandler = () => {
    navigation.navigate("Login");
  };

  const createAccountHandler = () => {
    navigation.navigate("RegistrationStartup");
  };

  return (
    <View style={styles.container}>
      <View style={styles.logoContainer}>
        <Image
          style={styles.logo}
          resizeMode="contain"
          source={ENTER_PAGE_PIC}
        />

        <Typography style={textStyle_i1}>Map Notepad</Typography>
```

```
      </View>

      <View style={styles.buttonsContainer}>
        <CustomButton
          style={CustomButtonStyles.base}
          backgroundColor="primary"
          textStyle={textStyle_i4}
          onPress={loginHandler}
        >
          Log in
        </CustomButton>

        <CustomButton
          style={CustomButtonStyles.base}
          backgroundColor="background"
          borderColor="primary"
          textStyle={textStyle_i5}
          onPress={createAccountHandler}
        >
          Create account
        </CustomButton>
      </View>
    </View>
  );
};
```

Файл RegistrationStartupScreen.tsx

```
import React from "react";
import { View } from "react-native";

import { GOOGLE_ICON } from "../../../assets/icons";
import {
  CustomButton,
  IconButton,
  InformativeTextInput,
} from "../../../components/common";
import { Separator } from "../../../components/sections";
import { IconButtonStyles, textStyle_i4 } from "../../../constants/styles";
import { EMAIL_RULES, USERNAME_RULES } from "../../../helpers";
import { useHookForm } from "../../../hooks";
import { AuthScreenProps } from "../../../navigation/AuthStack/types";
import { ICreateUserForm } from "../../../types/forms";
import styles from "./styles";

export const RegistrationStartupScreen: React.FC<AuthScreenProps> = ({
  navigation,
}) => {

  const goToNextRegistrationStepHandler = ({
    name,
    email,
  }: ICreateUserForm) => {
    navigation.navigate("RegistrationCompletion", {
      name,
      email,
    });
  };

  const { formController, handleSubmit } = useHookForm<ICreateUserForm>({
    defaultValues: {
      name: "name",
```

```
      email: "test123@mail.com",
    },
  });

  return (
    <View style={styles.container}>
      <View style={styles.inputsContainer}>
        <InformativeTextInput
          formController={formController}
          name="name"
          rules={USERNAME_RULES}
          title="Name"
          placeholder="Enter name"
        />

        <InformativeTextInput
          formController={formController}
          name="email"
          rules={EMAIL_RULES}
          autoCapitalize="none"
          keyboardType="email-address"
          title="Email"
          placeholder="Enter email"
        />
      </View>

      <View style={styles.buttonsContainer}>
        <CustomButton
          backgroundColor="primary"
          textStyle={textStyle_i4}
          onPress={handleSubmit(goToNextRegistrationStepHandler)}
        >
          Next
        </CustomButton>

        <Separator>or</Separator>

        <IconButton
          style={IconButtonStyles.outline}
          borderColor="primary"
          imageSource={GOOGLE_ICON}
        />
      </View>
    </View>
  );
};
```

Файл StartupScreen.tsx

```
import React from "react";
import { Image, View } from "react-native";

import { ENTER_PAGE_PIC } from "../../../assets/icons";
import { CustomButton, Typography } from "../../../components/common";
import {
  CustomButtonStyles,
  textStyle_i1,
  textStyle_i4,
  textStyle_i5,
} from "../../../constants";
import { AuthScreenProps } from "../../../navigation/AuthStack/types";
import styles from "./styles";
```

```
export const StartupScreen: React.FC<AuthScreenProps> = ({ navigation }) => {
  const loginHandler = () => {
    navigation.navigate("Login");
  };

  const createAccountHandler = () => {
    navigation.navigate("RegistrationStartup");
  };

  return (
    <View style={styles.container}>
      <View style={styles.logoContainer}>
        <Image
          style={styles.logo}
          resizeMode="contain"
          source={ENTER_PAGE_PIC}
        />

        <Typography style={textStyle_i1}>Map Notepad</Typography>
      </View>

      <View style={styles.buttonsContainer}>
        <CustomButton
          style={CustomButtonStyles.base}
          backgroundColor="primary"
          textStyle={textStyle_i4}
          onPress={loginHandler}
        >
          Log in
        </CustomButton>

        <CustomButton
          style={CustomButtonStyles.base}
          backgroundColor="background"
          borderColor="primary"
          textStyle={textStyle_i5}
          onPress={createAccountHandler}
        >
          Create account
        </CustomButton>
      </View>
    </View>
  );
};
```

Файл RegistrationCompletionScreen.tsx

```
import React, { useState } from "react";
import { View } from "react-native";

import { GOOGLE_ICON } from "../../../assets/icons";
import {
  CustomButton,
  IconButton,
  InformativeTextInput,
} from "../../../components/common";
import { LoaderView, Separator } from "../../../components/sections";
import { IconButtonStyles, textStyle_i4 } from "../../../constants/styles";
import { FirebaseErrorMessages } from "../../../enums";
import {
  PASSWORD_RULES,
  extractErrorMessage,
  getConfirmPasswordRules,
```

```
} from "../../../helpers";
import { useAuth, useHookForm } from "../../../hooks";
import { AuthScreenProps } from "../../../navigation/AuthStack/types";
import AlertService from "../../../services/AlertService";
import AuthService from "../../../services/AuthService";
import FirebaseErrorTranslator from
"../../../services/ErrorTranslator/FirebaseErrorTranslator";
import { ICreatePasswordForm } from "../../../types/forms";
import styles from "./styles";

export const RegistrationCompletionScreen: React.FC<AuthScreenProps> = ({
  navigation,
  route,
}: AuthScreenProps) => {
  const [isLoading, setIsLoading] = useState(false);
  const { setCredentials } = useAuth();

  const { formController, watch, handleSubmit } =
    useHookForm<ICreatePasswordForm>();

  const createAccountHandler = async ({ password }: ICreatePasswordForm) => {
    setIsLoading(true);

    const registerResult = await AuthService.registerWithEmail(
      route.params?.email ?? "",
      password
    );

    setIsLoading(false);

    if (registerResult.isSuccess && registerResult.data) {
      setCredentials(registerResult.data);
    } else {
      const errorMessage = extractErrorMessage(registerResult);

      AlertService.warning(FirebaseErrorTranslator.translate(errorMessage));

      if (errorMessage === FirebaseErrorMessages.EMAIL_ALREADY_IN_USE) {
        navigation.goBack();
      }
    }
  };

  if (isLoading) {
    return <LoaderView />;
  }

  return (
    <View style={styles.container}>
      <View style={styles.inputsContainer}>
        <InformativeTextInput
          formController={formController}
          rules={PASSWORD_RULES}
          name="password"
          secureTextEntry
          autoCapitalize="none"
          title="Password"
          placeholder="Create password"
        />

        <InformativeTextInput
          formController={formController}
```

```
          rules={getConfirmPasswordRules(watch("password"))}
          name="confirmPassword"
          secureTextEntry
          autoCapitalize="none"
          title="Confirm password"
          placeholder="Repeat password"
        />
      </View>

      <View style={styles.buttonsContainer}>
        <CustomButton
          backgroundColor="primary"
          textStyle={textStyle_i4}
          onPress={handleSubmit(createAccountHandler)}
        >
          Create account
        </CustomButton>

        <Separator>or</Separator>

        <IconButton
          style={IconButtonStyles.outline}
          borderColor="primary"
          imageSource={GOOGLE_ICON}
        />
      </View>
    </View>
  );
};
```

Файл LoginScreen.tsx

```
import React, { useState } from "react";
import { View } from "react-native";

import { GOOGLE_ICON } from "../../../assets/icons";
import {
  CustomButton,
  IconButton,
  InformativeTextInput,
} from "../../../components/common";
import { LoaderView, Separator } from "../../../components/sections";
import { IconButtonStyles, textStyle_i4 } from "../../../constants/styles";
import {
  EMAIL_RULES,
  PASSWORD_RULES,
  extractErrorMessage,
} from "../../../helpers";
import { useAuth, useHookForm } from "../../../hooks";
import { AuthScreenProps } from "../../../navigation/AuthStack/types";
import AlertService from "../../../services/AlertService";
import AuthService from "../../../services/AuthService";
import FirebaseErrorTranslator from
"../../../services/ErrorTranslator/FirebaseErrorTranslator";
import { ILoginForm } from "../../../types/forms";
import styles from "./styles";

export const LoginScreen: React.FC<AuthScreenProps> = ({ route }) => {

  const { setCredentials } = useAuth();
  const [isLoading, setIsLoading] = useState(false);

  const { formController, handleSubmit } = useHookForm<ILoginForm>({
```

```
  defaultValues: {
    email: route.params?.email ?? "test@mail.com",
    password: "Test123@",
  },
});

const submitHandler = async ({ email, password }: ILoginForm) => {
  setIsLoading(true);

  const loginResult = await AuthService.loginWithEmail(email, password);

  if (loginResult.isSuccess && loginResult.data) {
    setCredentials(loginResult.data);
  } else {
    const error = extractErrorMessage(loginResult);

    AlertService.error(FirebaseErrorTranslator.translate(error));
  }

  setIsLoading(false);
};

if (isLoading) {
  return <LoaderView />;
}

return (
  <View style={styles.container}>
    <View style={styles.inputsContainer}>
      <InformativeTextInput
        formController={formController}
        rules={EMAIL_RULES}
        name="email"
        title="Email"
        autoCapitalize="none"
        placeholder="Enter email"
        keyboardType="email-address"
      />

      <InformativeTextInput
        formController={formController}
        rules={PASSWORD_RULES}
        name="password"
        title="Password"
        placeholder="Enter password"
        autoCapitalize="none"
        secureTextEntry
      />
    </View>

    <View style={styles.buttonsContainer}>
      <CustomButton
        backgroundColor="primary"
        textStyle={textStyle_i4}
        onPress={handleSubmit(submitHandler)}
      >
        Login
      </CustomButton>

      <Separator>or</Separator>

      <IconButton
```

```
          style={IconButtonStyles.outline}
          borderColor="primary"
          imageSource={GOOGLE_ICON}
        />
      </View>
    </View>
  );
};
```

Файл interfaces.ts

```
import { AsyncResult } from "../helpers/AOResult/types";

export interface IFirebaseRestService {
  getArray: <TResponse>(url: string) => AsyncResult<Array<TResponse>>;
  getObject: <TResponse>(url: string) => AsyncResult<TResponse>;
  delete: (url: string) => AsyncResult<void>;
  post: <TPayload, TResponse>(
    url: string,
    payload: TPayload
  ) => AsyncResult<TResponse>;
  put: <TPayload>(url: string, payload: TPayload) => AsyncResult<void>;
}

import { AsyncResult } from "../helpers/AOResult/types";
import { IPinModel } from "../types/models";

export interface IPinsService {
  getPins: () => AsyncResult<Array<IPinModel>>;
  filterPinsBySearchQuery: (
    pins: Array<IPinModel>,
    searchQuery: string
  ) => Array<IPinModel>;
  deletePin: (pinId: string) => AsyncResult<void>;
  createPin: (pin: IPinModel) => AsyncResult<string>;
  updatePin: (pin: IPinModel) => AsyncResult<void>;
  toggleFavoritePinStatus: (pin: IPinModel) => AsyncResult<void>;
}
```

Файл AuthService.tsx

```
import {
  UserCredential,
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
} from "firebase/auth";
import { FirebaseAuth } from "../../FirebaseConfig";
import { ErrorMessages } from "../../enums";
import { ExecuteAsync } from "../../helpers/AOResult";
import { AsyncResult } from "../../helpers/AOResult/types";
import { ICredentialsModel } from "../../types/models";

class AuthService {
  public registerWithEmail = async (
    email: string,
    password: string
  ): AsyncResult<ICredentialsModel | undefined> => {
    const requestResult = await ExecuteAsync<UserCredential>(async () =>
      createUserWithEmailAndPassword(FirebaseAuth, email, password)
    );

    return requestResult.convertTo<ICredentialsModel>({
      userId: requestResult.data?.user.uid ?? ErrorMessages.UNDEFINED_USER_ID,
    });
```

```
  };

  public loginWithEmail = async (
    email: string,
    password: string
  ): AsyncResult<ICredentialsModel | undefined> => {
    const requestResult = await ExecuteAsync<UserCredential>(async () =>
      signInWithEmailAndPassword(FirebaseAuth, email, password)
    );

    return requestResult.convertTo<ICredentialsModel>({
      userId: requestResult.data?.user.uid ?? ErrorMessages.UNDEFINED_USER_ID,
    });
  };
}

export default new AuthService();
```

Файл BaseErrorTranslator.tsx

```
export class BaseErrorTranslator {
  private _mapper: Record<string, string>;

  constructor(mapper: Record<string, string>, keys: string[]) {
    this._mapper = mapper;

    this.validateMapper(keys);
  }

  private validateMapper = (enumValues: string[]) => {
    const missingKeys = enumValues
      .filter((key) => !(key in this._mapper))
      .join(", ");

    if (missingKeys.length > 0) {
      throw new Error(`Missing error code mappings: [${missingKeys}]`);
    }
  };

  public translate = (errorCode: string): string =>
    this._mapper[errorCode] || errorCode;
}

import { FirebaseErrorMessages } from "../../enums";
import { BaseErrorTranslator } from "./BaseErrorTranslator";
```

Файл FirebaseErrorTranslator.tsx

```
class FirebaseErrorTranslator extends BaseErrorTranslator {
  constructor() {
    const mapper: Record<string, string> = {
      [FirebaseErrorMessages.USER_NOT_FOUND]: "User not found",
      [FirebaseErrorMessages.EMAIL_ALREADY_IN_USE]: "Email already in use",
      [FirebaseErrorMessages.INVALID_PASSWORD]: "Invalid password",
      [FirebaseErrorMessages.WRONG_PASSWORD]: "Wrong password",

      [FirebaseErrorMessages.INVALID_ID_TOKEN]: "Invalid token",
      [FirebaseErrorMessages.ID_TOKEN_EXPIRED]: "Token expired",
      [FirebaseErrorMessages.ID_TOKEN_REVOKED]: "Token revoked",

      [FirebaseErrorMessages.OPERATION_NOT_ALLOWED]: "Operation not allowed",
      [FirebaseErrorMessages.INTERNAL_ERROR]: "Internal error",
      [FirebaseErrorMessages.TOO_MANY_REQUESTS]: "Too many requests",
      [FirebaseErrorMessages.NETWORK_REQUEST_FAILED]: "Network request failed",
```

```
    };

    super(mapper, Object.values(FirebaseErrorMessages));
  }
}

export default new FirebaseErrorTranslator();
```

Файл FirebaseDatabaseService.tsx

```
import { get, push, ref, remove, update } from "firebase/database";

import { FirebaseDatabase } from "../FirebaseConfig";
import { firebaseNodesToArray } from "../helpers";
import { ExecuteAsync } from "../helpers/AOResult";
import { IFirebaseRestService } from "../interfaces";
import { IFirebaseNodes } from "../types/api/firebase";
import { AsyncResult } from "../helpers/AOResult/types";

export class FirebaseDatabaseService implements IFirebaseRestService {
  private getReference = (path: string) => ref(FirebaseDatabase, path);

  public getObject = <TResponse>(url: string): AsyncResult<TResponse> =>
    ExecuteAsync<TResponse>(async () => {
      const response = await get(this.getReference(url));

      return response.val() as TResponse;
    });

  public getArray = <TResponse>(url: string): AsyncResult<Array<TResponse>> =>
    ExecuteAsync<Array<TResponse>>(async () => {
      const response = await get(this.getReference(url));

      const firebaseNodes = response.val() as IFirebaseNodes;

      return firebaseNodesToArray(firebaseNodes);
    });

  public delete = (url: string): AsyncResult<void> =>
    ExecuteAsync(async () => remove(this.getReference(url)));

  public post = <TPayload, TResponse>(url: string, payload: TPayload) =>
    ExecuteAsync<TResponse>(async () => {
      const { key } = await push(this.getReference(url), payload);

      return key as TResponse;
    });

  public put = (url: string, payload: any): AsyncResult<void> =>
    ExecuteAsync<void>(() => update(this.getReference(url), payload));
}
```

Файл SettingsService.tsx

```
import { AppThemes } from "../enums";
import { AsyncResult } from "../helpers/AOResult/types";
import { IFirebaseRestService } from "../interfaces";
import { ISettingsPayload } from "../types/api/firebase";
import { FirebaseDatabaseService } from "./FirebaseDatabaseService";

export class SettingsService {
  private _database: IFirebaseRestService;
  private _pathToSettings: string;
```

```
  constructor(userId: string) {
    this._database = new FirebaseDatabaseService();
    this._pathToSettings = `settings/${userId}`;
  }

  public getAppTheme = (): AsyncResult<AppThemes> =>
    this._database.getObject<AppThemes>(this._pathToSettings + "/theme");

  public updateAppTheme = async (theme: AppThemes): AsyncResult<void> => {
    const payload: ISettingsPayload = {
      theme: theme,
    };

    return this._database.put(this._pathToSettings, payload);
  };
}
```

Файл PinsService.tsx

```
import { pinModelToPinPayload } from "../converters";
import { ErrorMessages } from "../enums";
import { AsyncResult } from "../helpers/AOResult/types";
import { IFirebaseRestService, IPinsService } from "../interfaces";
import { IPinPayload } from "../types/api/firebase";
import { ICredentialsModel, IPinModel } from "../types/models";
import { stringToKeywords } from "../utils";
import { FirebaseDatabaseService } from "./FirebaseDatabaseService";

export class PinsService implements IPinsService {
  private _restService: IFirebaseRestService;
  private _pathToPins: string;

  constructor(credentials: ICredentialsModel | null) {
    this._restService = new FirebaseDatabaseService();

    const userId = credentials
      ? credentials.userId
      : ErrorMessages.UNDEFINED_USER_ID;

    this._pathToPins = `pins/${userId}`;
  }

  public getPins = async (): AsyncResult<Array<IPinModel>> =>
    this._restService.getArray<IPinModel>(this._pathToPins);

  filterPinsBySearchQuery = (
    pins: Array<IPinModel>,
    searchQuery: string
  ): Array<IPinModel> => {
    const keywords = stringToKeywords(searchQuery);

    return pins.filter((pin) => {
      const label = pin.label.toLowerCase();
      const description = pin.description?.toLowerCase();
      const latitude = pin.location.latitude.toString();
      const longitude = pin.location.longitude.toString();

      return keywords.some(
        (key) =>
          label.includes(key) ||
          description?.includes(key) ||
          latitude.includes(key) ||
          longitude.includes(key)
```

```
      );
    });
  };

  public deletePin = async (pinId: string): AsyncResult<void> =>
    this._restService.delete(`${this._pathToPins}/${pinId}`);

  public createPin = async (pin: IPinModel): AsyncResult<string> => {
    const payload = pinModelToPinPayload(pin);

    return this._restService.post<IPinPayload, string>(
      this._pathToPins,
      payload
    );
  };

  public updatePin = async (pin: IPinModel): AsyncResult<void> => {
    const payload = pinModelToPinPayload(pin);

    return await this._restService.put<IPinPayload>(
      `${this._pathToPins}/${pin.id}`,
      payload
    );
  };

  toggleFavoritePinStatus = async (pin: IPinModel): AsyncResult<void> => {
    const newPin: IPinModel = { ...pin, isFavorite: !pin.isFavorite };

    return this.updatePin(newPin);
  };
}

import {
  MessageType,
  showMessage as showFlashMessage,
} from "react-native-flash-message";

import {
  ALERT_DISPLAY_DURATION_IN_MS,
  FlashMessageColors,
} from "../../constants";
import styles from "./styles";

const showMessage = (message: string, type: MessageType, duration: number) =>
  showFlashMessage({
    style: styles.container,
    titleStyle: styles.title,
    type,
    message,
    icon: type,
    position: "top",
    backgroundColor: FlashMessageColors[type],
    duration: duration,
  });
```

Файл AlertService.tsx

```
class AlertService {
  public info = (
    message: string,
    duration: number = ALERT_DISPLAY_DURATION_IN_MS
  ) => showMessage(message, "info", duration);
```

```
  public success = (
    message: string,
    duration: number = ALERT_DISPLAY_DURATION_IN_MS
  ) => showMessage(message, "success", duration);

  public warning = (
    message: string,
    duration: number = ALERT_DISPLAY_DURATION_IN_MS
  ) => showMessage(message, "warning", duration);

  public error(
    message: string,
    duration: number = ALERT_DISPLAY_DURATION_IN_MS
  ) {
    showMessage(message, "danger", duration);
  }
}

export default new AlertService();
```

Файл useUserLocation.tsx

```
import {
  PermissionStatus,
  getCurrentPositionAsync,
  requestForegroundPermissionsAsync,
} from "expo-location";
import { useCallback, useEffect, useState } from "react";
import { LatLng } from "react-native-maps";
import { useSelector } from "react-redux";

import { ErrorMessages } from "../enums";
import { ExecuteAsync } from "../helpers/AOResult";
import AlertService from "../services/AlertService";
import { selectUserLocation } from "../store/redux/slices";

const requestLocationPermissions = async (): Promise<void> => {
  const { status } = await requestForegroundPermissionsAsync();

  switch (status) {
    case PermissionStatus.DENIED:
      throw new Error(ErrorMessages.LOCATION_PERMISSION_DENIED);
    case PermissionStatus.UNDETERMINED:
      throw new Error(ErrorMessages.SOME_WENT_WRONG);
  }
};

const getCurrentLocation = async (): Promise<LatLng> => {
  const { coords } = await getCurrentPositionAsync();
  return {
    latitude: coords.latitude,
    longitude: coords.longitude,
  };
};

type UseCurrentLocationReturn = {
  userLocation: LatLng | null;
  requestUserLocation: () => void;
};

export const useUserLocation = (
  shouldRequestLocationInitially: boolean = false
): UseCurrentLocationReturn => {
```

```
  const location = useSelector(selectUserLocation);

  const [userLocation, setUserLocation] = useState<LatLng | null>(location);

  const requestUserLocation = useCallback(async () => {
    const permissionResult = await ExecuteAsync(requestLocationPermissions);

    if (permissionResult.isSuccess) {
      const location = await getCurrentLocation();

      setUserLocation(location);
    } else {
      AlertService.info(permissionResult.getMessage());
    }
  }, [setUserLocation]);

  useEffect(() => {
    if (shouldRequestLocationInitially) {
      requestUserLocation();
    }
  }, [shouldRequestLocationInitially]);

  return { userLocation, requestUserLocation };
};

import { useMemo, useState } from "react";
import { useSelector } from "react-redux";

import { IPinsService } from "../interfaces";
import { PinsService } from "../services";
import AlertService from "../services/AlertService";
import {
  addPinAction,
  deletePinAction,
  setPinsAction,
  toggleFavoritePinStatusAction,
  updatePinAction,
} from "../store/redux/actions";
import { selectAuth, selectPins } from "../store/redux/slices";
import { useAppDispatch } from "../store/redux/store";
import { IPinModel } from "../types/models";

type UsePinsReturn = {
  pins: Array<IPinModel>;
  isPinsLoading: boolean;
  fetchPins: () => void;
  getPinsBySearchQuery: (searchQuery: string) => Array<IPinModel>;
  createPin: (pin: IPinModel) => Promise<boolean>;
  updatePin: (pin: IPinModel) => Promise<boolean>;
  togglePinFavoriteStatus: (pin: IPinModel) => void;
  deletePin: (pinId: string) => void;
};

export const usePins = (): UsePinsReturn => {
  const dispatch = useAppDispatch();

  const { credentials } = useSelector(selectAuth);
  const pins = useSelector(selectPins);
  const pinsService: IPinsService = useMemo(
    () => new PinsService(credentials),
    [credentials]
  );
```

```
const [isPinsLoading, setIsPinsLoading] = useState<boolean>(false);

const fetchPins = async () => {
  setIsPinsLoading(true);

  const result = await pinsService.getPins();

  setIsPinsLoading(false);

  if (result.isSuccess && result.data) {
    dispatch(setPinsAction(result.data));
  } else {
    AlertService.error(result.getMessage());
  }
};

const getPinsBySearchQuery = (searchQuery: string): Array<IPinModel> =>
  pinsService.filterPinsBySearchQuery(pins, searchQuery);

const createPin = async (pin: IPinModel): Promise<boolean> => {
  const result = await pinsService.createPin(pin);

  if (result.isSuccess && result.data) {
    const newPin: IPinModel = {
      ...pin,
      id: result.data,
    };

    dispatch(addPinAction(newPin));
  } else {
    AlertService.error(result.getMessage());
  }

  return result.isSuccess;
};

const updatePin = async (pin: IPinModel) => {
  const result = await pinsService.updatePin(pin);

  if (result.isSuccess) {
    dispatch(updatePinAction(pin));
  } else {
    AlertService.error(result.getMessage());
  }

  return result.isSuccess;
};

const togglePinFavoriteStatus = async (pin: IPinModel) => {
  const result = await pinsService.toggleFavoritePinStatus(pin);

  if (result.isSuccess) {
    dispatch(toggleFavoritePinStatusAction(pin.id));
  } else {
    AlertService.error(result.getMessage());
  }
};

const deletePin = async (pinId: string) => {
  const result = await pinsService.deletePin(pinId);
```

```
    if (result.isSuccess) {
      dispatch(deletePinAction(pinId));
    } else {
      AlertService.error(result.getMessage());
    }
  };

  return {
    pins,
    isPinsLoading,
    fetchPins,
    getPinsBySearchQuery,
    createPin,
    updatePin,
    togglePinFavoriteStatus,
    deletePin,
  };
};
```

Файл useAppTheme.tsx

```
import { DefaultTheme, Theme } from "@react-navigation/native";
import { StatusBarStyle } from "expo-status-bar";
import { useEffect, useState } from "react";
import { MapStyleElement } from "react-native-maps";
import { useSelector } from "react-redux";

import { IAppColors } from "../constants/themes/types";
import { AppThemes } from "../enums";
import { selectSettings } from "../store/redux/slices";

type UseAppThemeReturn = {
  theme: Theme | undefined;
  currentTheme: AppThemes;
  appColors: IAppColors;
  statusBarStyle: StatusBarStyle;
  mapStyles: MapStyleElement[];
};

export const useAppTheme = (): UseAppThemeReturn => {
  const {
    currentTheme,
    themeResource: { colors, statusBarStyle, mapStyles },
  } = useSelector(selectSettings);

  const [theme, setTheme] = useState<Theme>();

  useEffect(() => {
    setTheme({
      ...DefaultTheme,
      colors: {
        ...DefaultTheme.colors,
        card: colors.background,
        text: colors.systemGray,
        background: colors.background,
      },
    });
  }, [colors]);

  return {
    theme,
    currentTheme,
    appColors: colors,
```

```
    statusBarStyle,
    mapStyles,
  };
};
```

Файл useSettings.tsx

```
import { useMemo } from "react";

import { AppThemes, ErrorMessages } from "../enums";
import { extractErrorMessage } from "../helpers";
import AlertService from "../services/AlertService";
import FirebaseErrorTranslator from
"../services/ErrorTranslator/FirebaseErrorTranslator";
import { SettingsService } from "../services/SettingsService";
import { setAppThemeAction } from "../store/redux/actions";
import { useAppDispatch } from "../store/redux/store";
import { useAuth } from "./useAuth";

interface UseSettingsReturn {
  fetchSettings: () => Promise<void>;
  updateAppTheme: (theme: AppThemes) => Promise<void>;
}

export const useSettings = (): UseSettingsReturn => {
  const { credentials } = useAuth();
  const dispatch = useAppDispatch();

  const settingsService = useMemo(
    () =>
      new SettingsService(
        credentials?.userId ?? ErrorMessages.UNDEFINED_USER_ID
      ),
    [credentials]
  );

  const fetchSettings = async (): Promise<void> => {
    const result = await settingsService.getAppTheme();

    if (result.isSuccess) {
      if (result.data) {
        dispatch(setAppThemeAction(result.data));
      } else {
        AlertService.error(ErrorMessages.DATA_COULD_NOT_BE_RETRIEVED);
      }
    } else {
      const error = extractErrorMessage(result);

      AlertService.error(FirebaseErrorTranslator.translate(error));
    }
  };

  const updateAppTheme = async (theme: AppThemes): Promise<void> => {
    const result = await settingsService.updateAppTheme(theme);

    if (result.isSuccess) {
      dispatch(setAppThemeAction(theme));
    } else {
      const error = extractErrorMessage(result);

      AlertService.error(FirebaseErrorTranslator.translate(error));
    }
  };
```

```
  return {
    fetchSettings,
    updateAppTheme,
  };
};
```

Файл useHookForm.ts

```
import { UseFormProps, UseFormReturn, useForm } from "react-hook-form";

import { IFormController } from "../components/common";
import { IBaseForm } from "../types/forms";

type UseHookFormReturn<T extends IBaseForm> = {
  formController: IFormController;
} & Omit<UseFormReturn<T>, "control" | "resetField" | "trigger">;

export const useHookForm = <T extends IBaseForm>(
  props?: UseFormProps<T>
): UseHookFormReturn<T> => {
  const { control, trigger, resetField, ...rest } = useForm<T>(props);

  const formController: IFormController = {
    control,
    resetField,
    trigger,
  };

  return {
    formController,
    ...rest,
  };
};
```

Файл useHeaderRightButton.ts

```
import { StackNavigationProp } from "@react-navigation/stack";
import { useLayoutEffect } from "react";
import { ImageProps } from "react-native";

import { IconButton } from "../components/common/IconButton";
import { HomeStackParamList } from "../navigation/HomeStack/types";
import { scaleSize } from "../utils";

export const useHeaderRightButton = <T extends keyof HomeStackParamList>(
  navigation: StackNavigationProp<HomeStackParamList, T>,
  imageSource: ImageProps["source"],
  onPress: () => void
) => {
  useLayoutEffect(() => {
    navigation.setOptions({
      headerRight: () => (
        <IconButton
          style={{
            margin: scaleSize(12),
          }}
          tintColor="primary"
          imageSource={imageSource}
          onPress={onPress}
        />
      ),
    });
  }, [navigation]);
```

```
};
```

Файл useAuth.ts

```
import { useCallback } from "react";
import { useSelector } from "react-redux";

import { FirebaseAuth } from "../FirebaseConfig";
import { loginAction, logoutAction } from "../store/redux/actions";
import { selectAuth } from "../store/redux/slices";
import { useAppDispatch } from "../store/redux/store";
import { ICredentialsModel } from "../types/models";

type UseAuthReturn = {
  credentials: ICredentialsModel | null;
  isAuthenticated: boolean;
  setCredentials: (credentials: ICredentialsModel) => void;
  signOut: () => Promise<void>;
};

export const useAuth = (): UseAuthReturn => {
  const dispatch = useAppDispatch();
  const { isAuthenticated, credentials } = useSelector(selectAuth);

  const setCredentials = useCallback(
    (credentials: ICredentialsModel) => dispatch(loginAction(credentials)),
    []
  );

  const signOut = useCallback(async (): Promise<void> => {
    await FirebaseAuth.signOut();

    dispatch(logoutAction());
  }, []);

  return {
    credentials,
    isAuthenticated,
    setCredentials,
    signOut,
  };
};
```

Файл useFirebaseAutoLogin.ts

```
import { useEffect, useState } from "react";

import { FirebaseAuth } from "../FirebaseConfig";
import { ICredentialsModel } from "../types/models";
import { useAuth } from "./useAuth";

type UseLoginReturn = {
  credentials: ICredentialsModel | null;
  isAuthenticated: boolean;
  isLoginInProcess: boolean;
};

export const useFirebaseAutoLogin = (): UseLoginReturn => {
  const { isAuthenticated, setCredentials, credentials } = useAuth();
  const [isLoginInProcess, setIsLoginInProcess] = useState(true);

  useEffect(() => {
    const unsubscribe = FirebaseAuth.onAuthStateChanged(async (user) => {
      if (user) {
```

```
        setCredentials({
          userId: user.uid,
        });
      }

      setIsLoginInProcess(false);
    });

    return unsubscribe;
  }, []);

  return {
    credentials,
    isLoginInProcess,
    isAuthenticated,
  };
};
```

Файл useSplashScreen.ts

```
import * as SplashScreen from "expo-splash-screen";
import React, {
  FC,
  ReactNode,
  memo,
  useCallback,
  useEffect,
  useState,
} from "react";

type ISplashScreenWrapperProps = {
  children: ReactNode;
};

type UseSplashScreenReturn = {
  onContentReady: () => void;
  SplashScreenContainer: FC<ISplashScreenWrapperProps>;
};

export const useSplashScreen = (
  canHideSplashScreen: boolean
): UseSplashScreenReturn => {
  const [isContentReady, setIsContentReady] = useState(false);

  const onContentReady = useCallback(async () => {
    if (canHideSplashScreen) {
      await SplashScreen.hideAsync();
    }
  }, [canHideSplashScreen]);

  const SplashScreenContainer: FC<ISplashScreenWrapperProps> = memo(
    ({ children }) => <>{isContentReady ? children : null}</>
  );

  useEffect(() => {
    SplashScreen.preventAutoHideAsync();

    setIsContentReady(canHideSplashScreen);
  }, [canHideSplashScreen]);

  return { SplashScreenContainer, onContentReady };
};
```

Файл types.ts

```
import { IBaseModel } from "../../models";

export interface IFirebaseNodes {
  [index: string]: IBaseModel;
}

import { LatLng } from "react-native-maps";

export interface IPinPayload {
  userId: string;
  label: string;
  description: string;
  location: LatLng;
  isFavorite: boolean;
}

import { AppThemes } from "../../../enums";

export interface ISettingsPayload {
  theme?: AppThemes;
}

export interface IBaseItemModel {
  key: string;
}

import { LatLng, MapMarkerProps } from "react-native-maps";

import { IBaseItemModel } from "./baseItemModel";

export interface ICustomMarkerModel extends IBaseItemModel {
  label: string;
  description: string;
  location: LatLng;
  icon?: MapMarkerProps["icon"];
  showCallout?: () => void;
  hideCallout?: () => void;
}
import { LatLng } from "react-native-maps";

import { IBaseItemModel } from "./baseItemModel";

export interface IPinItemModel extends IBaseItemModel {
  label: string;
  description: string;
  location: LatLng;
  isFavorite: boolean;
}

export interface IBaseForm {
  [key: string]: string;
}


import { IBaseForm } from "./baseForm";

export interface ICreatePasswordForm extends IBaseForm {
  password: string;
  confirmPassword: string;
}
```

```
import { IBaseForm } from "./baseForm";

export interface ICreateUserForm extends IBaseForm {
  name: string;
  email: string;
}

import { IBaseForm } from "./baseForm";

export interface ILoginForm extends IBaseForm {
  email: string;
  password: string;
}

import { IBaseForm } from "./baseForm";

export interface IPinForm extends IBaseForm {
  label: string;
  description: string;
  latitude: string;
  longitude: string;
}

export interface IBaseModel {
  id: string;
}

export interface ICredentialsModel {
  userId: string;
}
import { LatLng } from "react-native-maps";

import { IBaseModel } from "./baseModel";

export interface IPinModel extends IBaseModel {
  label: string;
  description: string;
  location: LatLng;
  isFavorite: boolean;
}
```

Файл typographyStyleToTextStyle.ts

```
import { StyleProp, TextStyle } from "react-native";

import { ITypographyStyle } from "../components/common/Typography/types";
import { FontHeights, FontSizes, FontWeights } from "../constants";
import { IAppColors } from "../constants/themes/types";

export const typographyStyleToTextStyle = (
  style: StyleProp<ITypographyStyle>,
  appColors: IAppColors
): TextStyle => {
  let customStyle: TextStyle = {};

  if (style) {
    let unionStyles: ITypographyStyle = {};

    if (Array.isArray(style)) {
      unionStyles = Object.assign({}, ...style);
    } else if (typeof style === "object") {
      unionStyles = style;
    }
```

```
    const { lineHeight, fontWeight, fontSize, color, ...restStyleAttributes } =
      unionStyles;

    const specificStyle: TextStyle = {
      lineHeight: lineHeight && FontHeights[lineHeight],
      fontFamily: fontWeight && FontWeights[fontWeight],
      fontSize: fontSize && FontSizes[fontSize],
      color: color && appColors[color],
    };

    customStyle = Object.assign(restStyleAttributes, specificStyle);
  }

  return customStyle;
};
```

Файл mapHelpers.ts

```
import { RefObject } from "react";
import MapView, { LatLng } from "react-native-maps";

import { DEFAULT_REGION } from "../constants";
import { ICustomMarkerModel } from "../types/components";

export const animateToLocation = (
  mapViewRef: RefObject<MapView | null>,
  location: LatLng | null
) => {
  if (location) {
    mapViewRef.current?.animateToRegion({
      ...DEFAULT_REGION,
      ...location,
    });
  }
};

export const showMarkerCalloutById = (
  markers: ICustomMarkerModel[],
  pinId: string
) => {
  markers.find((x) => pinId === x.key)?.showCallout?.();
};

export const hideMarkerCalloutById = (
  markers: ICustomMarkerModel[],
  pinId: string
) => {
  markers.find((x) => pinId === x.key)?.hideCallout?.();
};

import { LatLng } from "react-native-maps";

export const formatCoordinate = ({ latitude, longitude }: LatLng) =>
  latitude.toFixed(8) + ", " + longitude.toFixed(8);
```

Файл firebaseHelpers.ts

```
import { IFirebaseNodes } from "../types/api/firebase";
import { AOResult } from "./AOResult";
```

```
export const firebaseNodesToArray = <T>(nodes: IFirebaseNodes): Array<T> => {
  const array: T[] = [];

  for (let key in nodes) {
    nodes[key].id = key;

    array.push(nodes[key] as T);
  }

  return array;
};

export const extractErrorMessage = <T>(result: AOResult<T>): string =>
  result.exception?.code ?? result.getMessage();
```

Файл firebaseHelpers.ts

```
import { UseControllerProps } from "react-hook-form";

import {
  EMAIL_REGEX,
  LATITUDE_REGEX,
  LONGITUDE_REGEX,
  PASSWORD_REGEX,
  USERNAME_REGEX,
} from "../../constants";
import { ValidationErrorMessages } from "../../enums";

export type RulesType = UseControllerProps["rules"];

export const USERNAME_RULES: RulesType = {
  required: ValidationErrorMessages.REQUIRED,
  minLength: { value: 2, message: ValidationErrorMessages.USERNAME_TOO_SHORT },
  maxLength: { value: 50, message: ValidationErrorMessages.USERNAME_TOO_LONG },
  pattern: {
    value: USERNAME_REGEX,
    message: ValidationErrorMessages.USERNAME_INVALID,
  },
};

export const EMAIL_RULES: RulesType = {
  required: ValidationErrorMessages.REQUIRED,
  pattern: {
    value: EMAIL_REGEX,
    message: ValidationErrorMessages.EMAIL_INVALID,
  },
};

export const PASSWORD_RULES: RulesType = {
  required: ValidationErrorMessages.REQUIRED,
  minLength: { value: 8, message: ValidationErrorMessages.PASSWORD_TOO_SHORT },
  pattern: {
    value: PASSWORD_REGEX,
    message: ValidationErrorMessages.PASSWORD_INVALID,
  },
};

export const PIN_LABEL_RULES: RulesType = {
  required: ValidationErrorMessages.REQUIRED,
  minLength: { value: 2, message: ValidationErrorMessages.PIN_LABEL_TOO_SHORT },
  maxLength: { value: 40, message: ValidationErrorMessages.PIN_LABEL_TOO_LONG },
};
```

```
export const LONGITUDE_RULES: RulesType = {
  required: ValidationErrorMessages.REQUIRED,
  pattern: {
    value: LONGITUDE_REGEX,
    message: "Incorrect format",
  },
};

export const LATITUDE_RULES: RulesType = {
  required: ValidationErrorMessages.REQUIRED,
  pattern: {
    value: LATITUDE_REGEX,
    message: "Incorrect format",
  },
};

import { ValidationErrorMessages } from "../../enums";
import { RulesType } from "./validationRules";
export const getConfirmPasswordRules = (
  comparedPassword: string
): RulesType => ({
  required: ValidationErrorMessages.REQUIRED,
  validate: (value) =>
    value === comparedPassword || ValidationErrorMessages.PASSWORD_MISMATCH,
});
```

Файл firebaseHelpers.ts

```
import { ErrorMessages } from "../../enums";
import { AsyncFunc, AsyncResult } from "./types";

export class AOResult<T = null> {
  public isSuccess: boolean = false;
  public data?: T;
  public exception?: any;
  public message?: string;

  public setSuccess(data: T) {
    this.isSuccess = true;
    this.data = data;
  }

  public setFailure(message: string | undefined) {
    this.isSuccess = false;
    this.message = message;
    this.exception = undefined;
  }

  public setException(exception: any) {
    this.isSuccess = false;
    this.message = undefined;
    this.exception = exception;
  }

  convertTo<TNewType = undefined>(data?: TNewType) {
    let newResult = new AOResult<TNewType>();

    newResult.data = data;
    newResult.isSuccess = this.isSuccess;
    newResult.exception = this.exception;
    newResult.message = this.message;

    return newResult;
```

```
  }

  getMessage(): string {
    return (
      this.exception?.message ?? this.message ?? ErrorMessages.SOME_WENT_WRONG
    );
  }
}

export async function ExecuteAsync<T>(func: AsyncFunc<T>): AsyncResult<T> {
  const result = new AOResult<T>();

  let isOnFailureExecuted: boolean = false;

  const onFailure = (message: string): void => {
    isOnFailureExecuted = true;
    result.setFailure(message);
  };

  try {
    const funcResult = await func(onFailure);

    if (!isOnFailureExecuted) {
      result.setSuccess(funcResult);
    }
  } catch (ex: any) {
    result.setException(ex);
  }

  return result;
}
```

Додаток Б

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	**Опис**
Пояснювальні документи	
Кваліфікаційна робота.doc	Пояснювальна записка. Документ Word.
Кваліфікаційна робота.pdf	Пояснювальна в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і відкомпільовану програму
Презентація	
Презентація.ppt	Презентація