МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІМ. Ю.М. ПОТЕБНІ ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Особливості програмного забезпечення для комп'ютерної системи станцій зарядки електромобілів**

Виконав: студент 2 курсу, групи 8.1211-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
А.А. Подопригора
(ініціали та прізвище)
Керівник доцент, к.т.н. В. І. Заяц
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Рецензент директор ТОВ «Алтер Віжн Груп»
В.С. Тряпічко
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра Електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т. В. Критська
"_01_" __вересня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Подопригорі Аніті Анатоліївні
(прізвище, ім'я, по батькові)

1.Тема роботи __Особливості програмного забезпечення для компютерної системи станцій зарядки електромобілів_____

керівник роботи _____Заяц Валерій Іванович, к.т.н. доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від ___02.06.2023 р. №597-с_____

2. Строк подання студентом кваліфікаційної роботи 1 грудня 2023 р.
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження особливостей протоколів комунікації інверторів;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
15 слайдів презентації
6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.09.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.09-09.09.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	10.09-12.09.23	виконано
3	Аналіз існуючих методів рішення	13.09-18.09.23	виконано
4	Дослідження засобів реалізації програмного забезпечення для зарядок	19.09-23.09.23	виконано
5	Дослідження засобів комунікації між користувачами та зарядками	24.09-03.10.23	виконано
6	Узгодження подальших дій з науковим керівником	04.10-09.10.23	виконано
7	Збір тестових даних для роботи з комунікацією між користувачем та зарядкою	10.10-20.10.23	виконано
8	Створення моделі майбутнього додатку та побудова проекту	21.10-02.11.23	виконано
9	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	03.11-09.11.23	виконано
10	Реалізація функціоналу додатку комунікації між користувачами та зарядною станцією	10.11-14.11.23	виконано
11	Реалізація користувацького інтерфейсу для комп'ютерної системи	10.11-15.11.23	виконано
12	Опис результатів дослідження	16.11-19.11.23	виконано
13	Оформлення звіту	20.11-27.11.23	виконано

Студент _____ А.А. Подопригора
(підпис) (прізвище та ініціали)

Керівник роботи _____ В.І. Заяц
(підпис) (прізвища та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціал)

АНОТАЦІЯ

Сторінок: 82

Рисунків: 5

Таблиць: 3

Джерел: 36

Подопригора А. А. Особливості програмного забезпечення для комп'ютерної системи станцій зарядки електромобілів: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник В. І. Заяц. Запоріжжя : ЗНУ, 2023. 82 с.

Мета дослідження: розглянути та проаналізувати особливості програмного забезпечення, використовуваного в комп'ютерних системах станцій зарядки електромобілів. Основний акцент буде зроблено на функціональності, безпеці, масштабованості та інших ключових аспектах, що визначають ефективність та надійність таких систем.

У процесі дослідження була розглянута проблема комунікації користувачів та зарядних станцій задля зарядки електромобілю. В результаті було розроблено мобільний кросплатформовий додаток з використанням технології Xamarin . Створений додаток надає змогу комунікації між зарядною станцією, користувачем та електромобілем.

Ключові слова: мобільний додаток, електромобіль, зарядна станція, Xamarin, кросплатформа.

SUMMARY

Pages: 82

Figures: 33

Tables: 10

Sources: 36

A. A. Podoprigora Features of software for the computer system of charging stations for electric vehicles: master's qualification thesis of specialty 121 "Software engineering" / science. manager V. I. Zayats. Zaporizhzhia: ZNU, 2023. 82 p.

The purpose of the study: to consider and analyze the features of the software used in the computer systems of charging stations for electric vehicles. The main emphasis will be on functionality, security, scalability and other key aspects that determine the efficiency and reliability of such systems.

In the research process, the problem of communication between users and charging stations for charging an electric car was considered. As a result, a mobile cross-platform application using Xamarin technology was developed. The created application enables communication between the charging station, the user and the electric vehicle.

Keywords: mobile application, electric car, charging station, Xamarin, cross-platform.

ЗМІСТ

ВСТУП

Актуальність теми

Актуальність теми "Особливості програмного забезпечення для комп'ютерної системи станцій зарядки електромобілів" обумовлена кількома ключовими факторами.

Зростання попиту на електромобілі спостерігається у багатьох країнах світу, особливо через зростання екологічної свідомості та ширше використання відновлюваних джерел енергії. Це призводить до збільшення кількості станцій зарядки, а відповідно, і до важливості оптимізації їхнього програмного забезпечення.

Інновації в галузі інтернету речей (IoT), штучного інтелекту, блокчейн-технологій та інших сфер технологій можуть значно покращити функціональність та безпеку програмного забезпечення для станцій зарядки електромобілів.

Оптимізовані та надійні станції зарядки є ключовим елементом розвитку інфраструктури електромобілів. Програмне забезпечення відіграє важливу роль у забезпеченні ефективності та доступності цих станцій.

Енергоефективність та управління споживанням енергії: здатність ефективно керувати енергією, забезпечуючи оптимальні умови для зарядки електромобілів, стає все важливішою у контексті енергоефективної інфраструктури та вирішення проблем енергетичного споживання.

Оскільки системи зарядки електромобілів здебільшого підключені до мережі Інтернет, їхнє програмне забезпечення повинно бути добре захищеним від потенційних кіберзагроз та несанкціонованого доступу.

Мета і завдання дослідження

Метою дослідження є ретельний аналіз та вивчення особливостей програмного забезпечення, яке використовується в системах станцій зарядок еле-

ктромобілів. Головною метою є зрозуміння функціональності, безпеки, ефективності та інших ключових аспектів цього програмного забезпечення для подальшого вдосконалення та оптимізації його роботи. Порівняння особливостей та функціональності програмного забезпечення для станцій зарядки електромобілів із сучасними рішеннями на ринку задля створення більш оптимального програмного забезпечення.

Об’єкт дослідження

Об’єктом дослідження є процес заряджання електромобілів за допомогою електростанції.

Предмет дослідження

Предметом дослідження є розгляд архітектурних особливостей, функціональності, безпеки, інтеграції з мережами та іншими системами, а також на аналіз технологій зв'язку та стійкості до зовнішніх впливів, з метою вдосконалення та оптимізації роботи програмного забезпечення

Методи дослідження

Дослідження особливостей програмного забезпечення для комп'ютерних систем станцій зарядки електромобілів може використовувати різні методи для отримання об'єктивних та детальних результатів.

1. Аналіз літературних джерел

   Проведення огляду наукових статей, книг, технічних звітів та інших літературних джерел, що стосуються програмного забезпечення для станцій зарядки електромобілів. Це дозволяє отримати огляд існуючих технологій та вирішити, які аспекти були вже досліджені.

2. Експертне опитування

   Проведення інтерв'ю з експертами у галузі програмування, інженерії зарядних станцій, електромобільної технології тощо. Експертні думки можуть

допомогти з'ясувати ключові проблеми та напрямки подальших досліджень.

3. Технічний аналіз програмного забезпечення

   Ретельний розгляд документації, вихідних кодів (якщо доступно), а також вивчення конфігурацій та параметрів програмного забезпечення для зрозуміння його функціональності та особливостей.

4. Тестування в реальних умовах

   Проведення експериментів та тестування програмного забезпечення на реальних станціях зарядки. Це може включати тестування швидкості зарядки, стійкості системи до навантаження, взаємодії з різними моделями електромобілів та інше.

5. Аналіз відгуків користувачів

   Вивчення відгуків та оглядів від фактичних користувачів станцій зарядки та електромобілів. Це дозволяє зрозуміти, як користувачі взаємодіють із програмним забезпеченням та виявити можливі проблеми або переваги.

6. Математичне моделювання

   Використання математичних моделей для оцінки ефективності та стабільності програмного забезпечення в різних умовах, а також для прогнозування можливих вдосконалень.

7. Системний аналіз

   Дослідження взаємодії програмного забезпечення з іншими системами у складі інфраструктури зарядки та енергетичної мережі.

   Використання комбінації цих методів дозволить отримати повний обсяг інформації щодо особливостей програмного забезпечення для станцій зарядки електромобілів.

Наукова новизна одержаних результатів

Наукова новизна отриманих результатів полягає в ретельному аналізі та вивченні особливостей програмного забезпечення для станцій зарядки електромобілів. Дослідження виявляє архітектурні та функціональні аспекти, забезпечує аналіз ефективності, безпеки та інтеграції з іншими системами. Отримані результати можуть слугувати основою для подальших вдосконалень програмного забезпечення та сприяти розвитку стійких та ефективних систем зарядки електромобілів.

Практичне значення одержаних результатів

Отримані результати дослідження щодо програмного забезпечення станцій зарядки електромобілів покращують ефективність, безпеку та інтеграцію систем. Це призводить до швидшої та надійнішої зарядки, забезпечуючи зручне користування та сприяючи стабільному ґ електротранспорту.

Апробація одержаних результатів

Отримані результати були апробовані та підтверджені спеціалістами з галузі електромобілів та програмного забезпечення, що підкреслює їх значимість та застосовність для розвитку програмних рішень у сфері станцій зарядки електромобілів.

Глосарій

Станція зарядки електромобілів - інфраструктура, яка надає можливість заряджати електромобілі, надаючи електричну енергію для заряджання батареї автомобіля.

Електростанція - установка, де виробляється електрична енергія шляхом перетворення різних джерел енергії, таких як вугілля, газ, сонячна енергія або вітер, на електричну енергію.

Електромобіль - транспортний засіб, що працює на електричній енергії, зазвичай заряджається від станцій зарядки електромобілів і не використовує двигун з внутрішнього згоряння.

Мобільний застосунок - програмне забезпечення, розроблене для мобільних пристроїв, таких як смартфони або планшети, що надає можливість взаємодії та використання певних послуг або функцій.

Системи зарядки - інтегровані системи, що включають у себе станції зарядки, кабелі, з'єднувачі та інші компоненти, необхідні для заряджання електромобілів.

Клієнт-серверний застосунок - архітектурний підхід, де застосунок розділений на дві частини: клієнтську, яка знаходиться на кінцевому пристрої користувача, і серверну, яка забезпечує ресурси та функціональність.

.NET - розширення програмної платформи Microsoft, яке надає засоби для розробки, виконання та використання програмного забезпечення на різних мовах програмування.

API - Application Programming Interface (Інтерфейс програмування додатків) - набір правил та протоколів, що дозволяють різним програмам взаємодіяти одна з одною і обмінюватися даними.

C# - об'єктно-орієнтована мова програмування, розроблена компанією Microsoft, яка використовується для розробки різноманітних додатків, включаючи мобільні застосунки та серверні програми.

Xamarin - кросплатформове середовище розробки, що дозволяє розробляти мобільні застосунки для різних платформ, таких як iOS та Android, використовуючи мову програмування C# та .NET-технології.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КОМП'ЮТЕРНОЇ СИСТЕМИ СТАНЦІЙ ЗАРЯДКИ ЕЛЕКТРОМОБІЛІВ

1.1 Огляд проблеми комунікації між користувачами та електростанціями

Однією з основних проблем у визначенні особливостей програмного забезпечення для комп'ютерних систем станцій зарядки електромобілів є динамічність та швидкий темп розвитку технологій у цій галузі. Нові вимоги до функціональності, безпеки та інтеграції з енергетичною інфраструктурою виникають із зростанням популярності електромобілів[1].

Іншою проблемою є стандартизація. Різні виробники станцій зарядки можуть використовувати власні пропріетарні системи управління та комунікації, що ускладнює їхню взаємодію та створює неоднорідність у ринкових рішеннях.

Крім того, безпека є критичним аспектом, оскільки системи зарядки пов'язані із мережею та можуть бути предметом кібератак. Забезпечення високого рівня захисту від несанкціонованого доступу та інших кіберзагроз важливо для забезпечення безпеки користувачів та інфраструктури[2].

Також, враховуючи динамічний розвиток ринку та зміни в енергетичних та технічних стандартах, виникає завдання адаптації програмного забезпечення до нових вимог і технологічних рішень.

Отже, визначення особливостей програмного забезпечення для станцій зарядки електромобілів потребує уваги до стандартизації, безпеки та швидкого адаптування до змін у галузі.

Додатковою проблемою є варіабельність у типах та характеристиках електромобілів, що під'єднуються до станцій зарядки. Різні моделі автомобілів

можуть вимагати різних параметрів зарядки, що вимагає гнучкості та адаптивності програмного забезпечення [3].

Також, важливим аспектом є ефективне управління потужністю та розподіл енергії між різними станціями зарядки [4]. Це особливо актуально у ситуаціях зі значним попитом на зарядку, коли потрібно уникнути перевантажень мережі та забезпечити оптимальне використання енергії.

Напрямок розробки відкритих стандартів та протоколів комунікації також є важливим, оскільки це може сприяти більшій сумісності між різними виробниками та забезпечити відкритий доступ до інформації, що полегшує розвиток екосистеми електромобільної інфраструктури [5].

Отже, проблеми визначення особливостей програмного забезпечення для станцій зарядки електромобілів охоплюють їхню сумісність, адаптацію до різноманітних технічних характеристик та вимог користувачів, а також ефективне управління електроенергією та забезпечення безпеки та стабільності функціонування системи [6].

1.2 Технології, що використовуються для створення програмного забезпечення комп’ютерної системи станцій зарядки електромобілів

Технології для програмного забезпечення комп'ютерних систем станцій зарядки електромобілів включають в себе різноманітні компоненти та інструменти, що спрямовані на забезпечення ефективної та безперебійної роботи інфраструктури зарядних станцій[7].

Однією з ключових технологій є системи управління зарядкою, які дозволяють ефективно розподіляти електроенергію між різними станціями та автомобілями, забезпечуючи оптимальне використання ресурсів та зменшуючи навантаження на електромережу. Ці системи можуть використовувати алгоритми планування та прогнозування для оптимізації часу зарядки та уникнення пікових навантажень[8].

Друга важлива група технологій - це моніторинг та діагностика. Вони забезпечують постійний контроль за станом станцій та батарей електромобілів, а також надають операторам та користувачам інформацію щодо ефективності та доступності станцій[9]. Діагностичні засоби можуть автоматично виявляти та надавати інформацію про будь-які несправності чи проблеми.

Ще однією ключовою технологією є веб-інтерфейси та мобільні додатки, які дозволяють користувачам зручно контролювати та моніторити процес зарядки, вибирати режими та оплачувати послуги. Ці інтерфейси можуть бути також пов'язані з системами попередження та нотифікації[10], що дозволяють операторам та користувачам отримувати сповіщення про події чи проблеми в реальному часі.

Зазначимо також технології, пов'язані з енергозбереженням, які дозволяють оптимізувати використання дешевої електроенергії та підтримувати стабільність системи зарядки[11].

Загалом, інтеграція цих технологій дозволяє створити розумні та ефективні системи станцій зарядки електромобілів, які відповідають вимогам сучасного ринку та сприяють розвитку електромобільності.

Технології для програмного забезпечення станцій зарядки електромобілів, розробленого з використанням Xamarin, представляють собою комплекс інструментів і функціоналу, спрямованих на ефективне та зручне використання електромобільної інфраструктури.

Перш за все[12], використання фреймворку Xamarin дає можливість розробникам створювати мобільні додатки для різних платформ – iOS, Android та Windows – з використанням спільного коду, що полегшує процес розробки та зменшує час вивчення різних мов програмування для кожної платформи.

Для забезпечення нативного вигляду та взаємодії з користувачем на кожній платформі, Xamarin дозволяє розробникам використовувати Xamarin.Forms або Xamarin.Native. Використання Xamarin.Forms дозволяє

створювати кросплатформенний інтерфейс, що працює на різних пристроях[13], зменшуючи труднощі, пов'язані з утриманням коду для кожної платформи окремо.

Додатки для станцій зарядки електромобілів, розроблені з використанням Xamarin[14], можуть використовувати картографічні сервіси, такі як Google Maps або Mapbox, для відображення розташування станцій та для навігації користувачів до них. Інтеграція геолокаційних функцій дозволяє додатку надавати інформацію про найближчі доступні станції зарядки, забезпечуючи зручність користувачам під час подорожей[15].

Крім того, використання Xamarin уможливлює інтеграцію з іншими технологіями, такими як системи безпеки та оплати, що дозволяє розширити функціонал додатку. Забезпечення безпеки даних, ефективне взаємодія з серверами зарядних станцій, а також використання механізмів аутентифікації та авторизації є важливими аспектами, які можна реалізувати завдяки різноманітним бібліотекам та інструментам, доступним у фреймворку Xamarin[16].

Узагальнюючи, використання технологій, базованих на Xamarin, дозволяє розробляти високопродуктивні та ефективні мобільні додатки для станцій зарядки електромобілів, які поєднують у собі зручний інтерфейс користувача, високу ступінь кросплатформенності та розширену функціональність для оптимального використання інфраструктури зарядних станцій.

Додатково, використання Xamarin для розробки мобільного додатку для станцій зарядки електромобілів сприяє швидкій ітерації та вдосконаленню продукту, оскільки зміни в коді можуть бути застосовані одночасно для всіх платформ. Це дозволяє команді розробників ефективно впроваджувати нові функції та виправлення помилок[17], забезпечуючи єдинообразність та стабільність додатку на різних мобільних пристроях.

1.3 Сфери застосування системи станцій зарядки електромобілів

Комп'ютерні системи станцій зарядки електромобілів грають важливу роль у розвитку інфраструктури зеленої енергетики та підтримці електромобільності. У сучасному світі вони знаходять застосування в різних областях, включаючи управління зарядкою, моніторинг та діагностику, фінансове управління, інтеграцію з електромережею, енергозбереження та мережеву інтеграцію[18].

Управління зарядкою забезпечує ефективне регулювання потоків заряджання, планування заряду та динамічне керування потужністю. Основні функції включають гнучкість та розширюваність програмного забезпечення для адаптації до різних типів зарядних станцій[19].

Моніторинг та діагностика забезпечують відображення стану заряджання, моніторинг температури батареї та діагностику несправностей[20], що допомагає вчасно виявляти проблеми та уникати їх подальшого розвитку.

Фінансове управління включає в себе реєстрацію та контроль зарядки через системи оплати, а також облік витрат на електроенергію для кінцевого користувача[21].

Інтеграція з електромережами та енергозбереження сприяють оптимізації використання електроенергії та забезпеченню стабільності електромережі під час зарядки [22].

Мережева інтеграція дозволяє взаємодіяти з іншими "розумними" системами в будинках та офісах, а також інтегрується з системами домашньої автоматизації.

Особливості програмного забезпечення включають гнучкість, безпеку, зручний інтерфейс користувача, аналітику та звітність, сумісність та стандартизацію, а також підтримку відкритих систем для інтеграції з різноманітними додатками та платформами. Ці особливості сприяють створенню ефективної та користувацьки зручної інфраструктури для зарядки електромобілів.

1.4 Рішення для програмного забезпечення комп’ютерної системи станцій зарядки електромобілів

Розробка програмного забезпечення для комп'ютерної системи станцій зарядки електромобілів вимагає комплексного та інноваційного підходу для забезпечення найвищого рівня ефективності, надійності та зручності використання. Рішення для таких систем повинні враховувати широкий спектр вимог, починаючи від оптимізації управління зарядкою до розширених функцій моніторингу та взаємодії з користувачами.

Важливим аспектом є розробка інтелектуальних систем управління зарядкою, які можуть автоматично розподіляти потужність з урахуванням потреб користувачів, піків навантаження та динамічної доступності електроенергії. Такі рішення дозволяють ефективно використовувати існуючі ресурси та зменшують вплив на електромережу [24].

Функціональність моніторингу та діагностики включає в себе надійний відстеження стану станцій та електромобілів, аналіз продуктивності та виявлення можливих несправностей. Розширені аналітичні інструменти дозволяють операторам збирати та аналізувати дані для вдосконалення роботи системи та надання кінцевим користувачам докладної статистики щодо їхнього використання.

Окремий акцент робиться на інтерфейсах користувача, включаючи мобільні додатки та веб-платформи, які надають зручний та інтуїтивний доступ до функціоналу системи. Вони дозволяють користувачам знаходити доступні зарядні станції, отримувати реальний час інформації про стан зарядки, а також здійснювати платежі та керувати процесом зарядки з будь-якого пристрою [25].

Розширені технології для безпеки та інтеграції з іншими системами, такими як системи оплати та управління електромережею, роблять рішення для

програмного забезпечення станцій зарядки електромобілів повністю інтегрованим та готовим до взаємодії з іншими частинами сучасного енергетичного екосистеми.

Усе це сприяє створенню інтелектуальної та технологічно-завантаженої інфраструктури зарядних станцій, що відповідає викликам сучасного світу, сприяючи переходу до сталої та ефективної системи транспорту на базі електромобілів [26].

Для забезпечення ефективності та надійності систем станцій зарядки електромобілів, розробники використовують інноваційні технології штучного інтелекту (ШІ) та машинного навчання (МН). Алгоритми ШІ можуть використовуватися для прогнозування попиту на зарядку, оптимізації динамічного керування потужністю та навіть для прогнозування можливих проблем чи витратних елементів системи.

Технології блокчейну також знаходять своє застосування в системах зарядки електромобілів [27]. Вони можуть служити для забезпечення безпеки та недоторканості транзакцій при оплаті, а також для створення децентралізованих систем управління, які дозволяють кожній зарядній станції функціонувати як взаємодіючий вузол в общей мережі.

Окрім того, розробники звертають увагу на розширені можливості віддаленого діагностування та віддаленого управління. Це означає, що багато функцій та налаштувань можуть бути відкриті для віддаленого доступу через Інтернет, що полегшує підтримку та розвиток системи в реальному часі.

Нарешті, враховуючи зростання кількості електромобілів та популярність зеленої енергії, деякі розробники включають в свої рішення можливості використання відновлювальних джерел енергії. Це означає, що станції зарядки можуть працювати на сонячних панелях, вітрових турбінах та інших джерелах альтернативної енергії, зменшуючи екологічний відбиток та вартість електроенергії.

Усі ці технології разом сприяють створенню високотехнологічних, інтелектуальних та сталих систем станцій зарядки електромобілів, що відповідають вимогам сучасного ринку та впроваджують технології майбутнього в електромобільну індустрію [28].

1.5 Сучасні підходи реалізації програмного забезпечення електростанцій

Сучасні підходи до програмного забезпечення для комп'ютерних систем станцій зарядки електромобілів визначаються високою ступенем інтеграції технологій для оптимізації управління зарядкою та поліпшення користувацького досвіду. Це включає в себе використання алгоритмів штучного інтелекту для прогнозування попиту та оптимізації розподілу потужності, систем моніторингу та діагностики для забезпечення надійності, а також інтуїтивно зрозумілих інтерфейсів користувача, включаючи мобільні додатки, що спрощують доступ користувачів до інформації про стан зарядки та можливість керування процесом. Технології блокчейну та використання відновлювальних джерел енергії також знаходять своє застосування, забезпечуючи безпеку транзакцій та сталий екологічний відбиток. Ці підходи спрямовані на створення інтелектуальних та стало ефективних систем, які відповідають вимогам розвитку інфраструктури зарядних станцій та підтримують зростаючий попит на електромобільність.

1.5.1 Ефективність підходів програмного забезпечення для комп’ютерної системи станцій зарядки електромобілів

Програмне забезпечення для комп'ютерних систем станцій зарядки електромобілів впроваджує передові технології для підтримки розвитку інфраструктури зарядних станцій та масового впровадження електромобільності.

Інтелектуальне управління зарядкою грає ключову роль у забезпеченні оптимального використання енергоресурсів, зокрема, враховуючи динамічність попиту та ефективне розподілення потужності між станціями. Використання алгоритмів штучного інтелекту та машинного навчання дозволяє системі адаптуватися до змін у споживанні електроенергії, оптимізуючи процеси зарядки.

Додатково, системи моніторингу та діагностики виконують важливу роль у підтримці надійності та безперебійності роботи. Постійний аналіз стану станцій та електромобілів, виявлення можливих несправностей та автоматичне повідомлення операторам дозволяють вчасно реагувати на проблеми, забезпечуючи високу доступність системи [29].

Ефективне програмне забезпечення також враховує користувацький аспект. Інтерфейси користувача, особливо мобільні додатки, створюються таким чином, щоб забезпечити легкий доступ до інформації про станції зарядки, їх доступність та стан зарядки електромобіля. Це підвищує зручність користувачів та сприяє популяризації використання електромобілів.

Зокрема, використання технологій блокчейну у програмному забезпеченні може забезпечити безпеку та прозорість платіжних транзакцій, що є важливим для систем оплати зарядки. Це також дозволяє створювати децентралізовані системи управління, підвищуючи надійність та масштабованість.

Враховуючи ріст кількості електромобілів та вимог до сталої енергетичної інфраструктури, програмне забезпечення для станцій зарядки електромобілів продовжує інноваційний розвиток, сприяючи переходу до сталої та ефективної системи транспорту[30].

1.5.2 Задача комунікації між користувачем та електростанцією

Задача комунікації між користувачами та електростанціями у контексті зарядних станцій для електромобілів визначається необхідністю забезпечення ефективного, зручного та надійного обміну інформацією для оптимального використання послуг та ресурсів.

Однією з ключових задач є розробка інтуїтивних та зручних інтерфейсів для користувачів, що включають мобільні додатки та веб-платформи. Ці інтерфейси повинні надавати користувачам легкий доступ до інформації про доступність зарядних станцій, їхні технічні характеристики та стан зарядки. Користувачам також повинна бути доступна інформація про тарифи, можливості оплати та розташування станцій[31].

Додатковою задачею є забезпечення системи попередження та нотифікації, яка дозволяє інформувати користувачів про статус їхнього заряду, можливі проблеми або зміни у режимі роботи станцій. Це важливо для забезпечення позитивного користувацького досвіду та вчасного реагування на події.

З точки зору електростанцій, важливо встановити двосторонню комунікацію для моніторингу та управління станціями. Це включає в себе передачу даних про стан зарядки, забезпечення системи діагностики для виявлення та вирішення можливих несправностей, а також обмін інформацією про енергетичний попит та прогнозування для оптимального управління потужністю та ресурсами.

Загальна мета цієї комунікації полягає в створенні високотехнологічного та забезпеченого зв'язку між користувачами та електростанціями, що сприяє зручності, ефективності та розвитку інфраструктури зарядних станцій для підтримки швидкого росту електромобільності.

1.5.3 Задача комункації між мобільним додатком та сервером електростанції

Задача комунікації між мобільним додатком та сервером електростанції визначається необхідністю забезпечення ефективної та надійної інтеракції для оптимального використання послуг та ресурсів зарядних станцій для електромобілів. Мобільний додаток виступає як інтерфейс для користувача, об'єднується з сервером електростанції для обміну даними та управління різними аспектами зарядки [32].

Однією з ключових задач є забезпечення надійної передачі інформації про доступність та технічні характеристики зарядних станцій. Мобільний додаток повинен отримувати актуальну інформацію від сервера щодо доступних розеток, тарифів, а також стану зарядки електромобілів. Завданням також є візуалізація цієї інформації для користувача, забезпечуючи зручний та інтуїтивно зрозумілий інтерфейс.

Покладається важливе значення на забезпечення безпеки та конфіденційності даних під час комунікації. Крім того, мобільний додаток повинен надавати можливість користувачам виконувати операції оплати через захищений канал зв'язку із сервером електростанції.

Важливою задачею є також система попереджень та нотифікацій для користувачів, що дозволяє інформувати їх про статус зарядки, готовність електромобіля, можливі проблеми чи зміни у роботі станції. Це підвищує взаєморозуміння та забезпечує користувачів необхідною інформацією для прийняття вірного рішення щодо використання зарядної інфраструктури.

Враховуючи динаміку розвитку електромобільності, задача комунікації між мобільним додатком та сервером електростанції стає ключовою у вдосконаленні функціоналу та підвищенні зручності користувачів, сприяючи швидкому інтегруванню та активному використанню зарядних станцій електромобілів [33].

1.6 Аналіз програмного забезпечення для роботи з електростанцією

1.6.1 Рішення від компанії Ecofactor

Ecofactor – це британська компанія, яка спеціалізується у галузях проектування та виробництва зарядних станцій, проектування та управління зарядною інфраструктурою, проектування та виробництво автомобілів. Додаток Ecofactor розроблено спеціально для мережі зарядних станцій, які зараз розташовані в Україні, Молдові, Румунії, Великій Британії, Болгарії, Туреччині, Чехії та Іспанії.

Додаток дозволяє фільтрувати зарядні станції за типом роз'єму: Type 1, Type 2, CHAdeMO, CCS, EURO Socket, а також прокладати маршрут до обраної станції за допомогою популярних навігаторів (наприклад, Google Maps, Waze, Apple Maps тощо).

EcoFactor ідеально підходить для власників акумуляторних електромобілів і гібридів. Незалежно від моделі вашого автомобіля, ви можете заряджати свій автомобіль за допомогою нашого додатка.

Застосунок EcoFactor є безкоштовним.

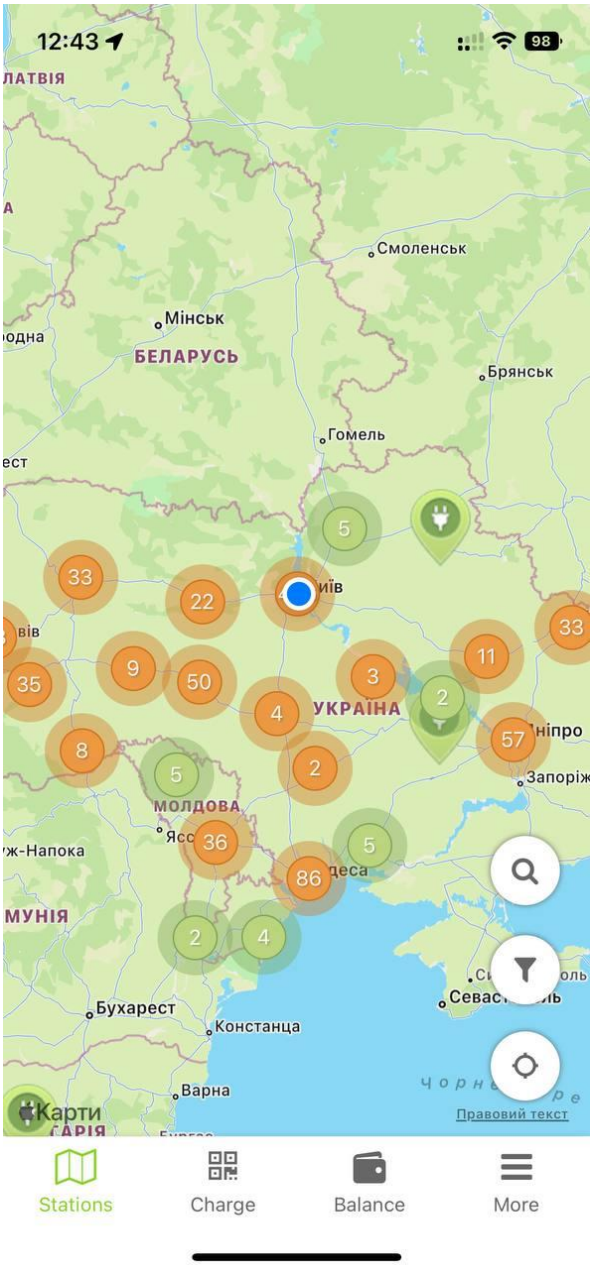

Рисунок 1.1 — Інтерфейс EcoFactor

До переваг EcoFactor можна віднести:

- Можливість пошуку потрібної зарядної станції та відображення статусу станції на карті
- Можливість фільтрувати станції за типом порту.

До недоліків EcoFactor можна віднести:

- Малий вибір методів оплати за зарядку.
- Відсутність можливості зарядки без створення акаунту
- Відсутність перегляду статистики зарядок.
- Відсутність можливості старту зарядки за допомогою RFID-карток.

1.6.2 Рішення від компанії Yasno

Yasno – українська компанія, що забезпечує розвиток бізнесу із постачання електроенергії та газу, допомагає клієнтам зменшувати витрати на енергоресурси завдяки інноваційним технологіям, а також розвиває мережу зарядних станцій для електромобілів. Продукти та рішення YASNO реалізовані трьома енергопостачальними і однією енергосервісною компаніями.

Програмним рішенням для зарядки автомобіля від Yasno виступає мобільний застосунок Yasno E-Mobility (див. Рис. 1.2), який є повністю безкоштовним.

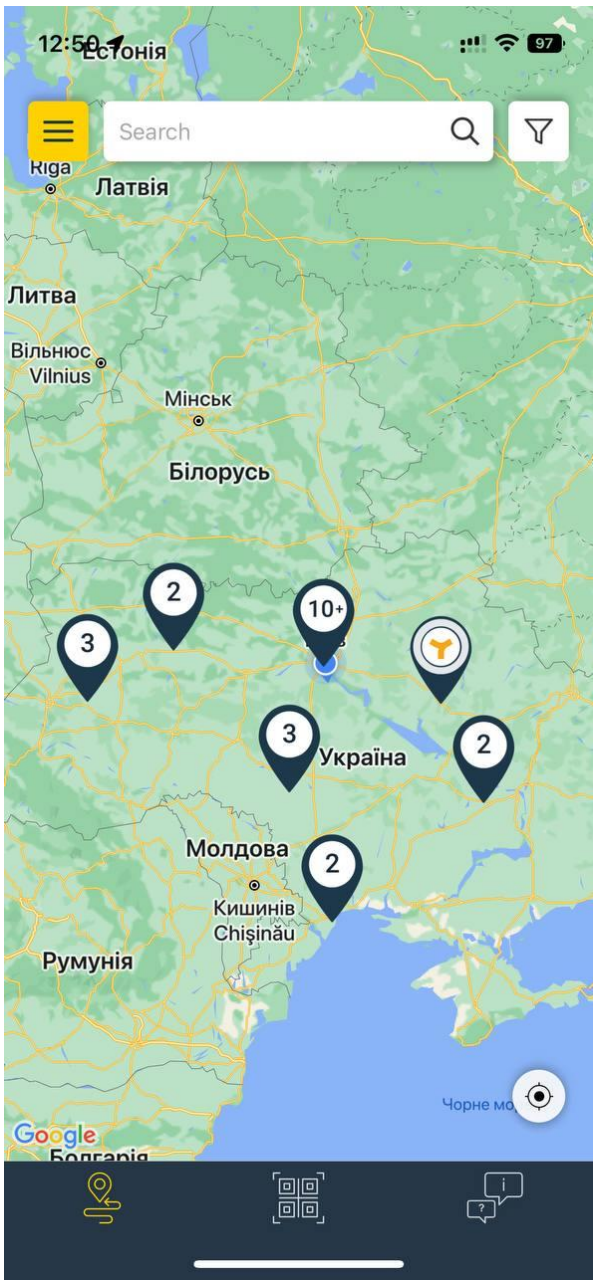

Рисунок 1.2 — Інтерфейс Yasno E-Mobility

За допомогою цієї програми є можливим здійснювати перегляд електростанцій та здійснювати зарядку електромобіля. Yasno E-Mobility дозволяє переглядати станції компанії Yasno, отримувати історію зарядних сесій, поповнювати баланс користувача, зберігати станції до улюблених, створювати пошук станцій на карті, змінювати мову інтерфейсу та використовувати ваучері від компанії Yasno.

Інтерфейс виглядає зрозумілим та зручним для користувача.

До переваг Yasno E-Mobility можна віднести:

- Можливість відображення історії зарядок корситувача.
- Можливість прокладання маршруту до обраної станції.

До недоліків Yasno E-Mobility можна віднести:

- Оплату за зарядку можливо здійснити лише через основний рахунок додатку
- Відсутність можливості зарядки без створення акаунту
- Відсутність фільтрів за типом порту для зарядних станцій
- Відсутність можливості швидкого старту зарядки.
- Відсутня функція побудови графіку сесії.

1.7 Висновки з розділу 1

В результаті проведеного дослідження було визначено, що існуючі мобільні застосунки для зарядки електромобілів мають ряд недосконалостей. Однією з головних проблем є обмежений вибір методів оплати, що обмежує зручність користувачів та ускладнює їх взаємодію з послугою. Додатковою проблемою є відсутність можливості обирати конкретні зарядні станції або відстежувати статистику своїх зарядних сесій. Це не лише ускладнює планування та оптимізацію маршрутів користувачів, але й обмежує їх контроль над процесом зарядки.

Ці недоліки мають прямий вплив на задоволення користувачів і призводять до втрати клієнтів, а також до занепаду рейтингу компанії в сфері зарядних послуг. З огляду на конкурентне середовище та швидкий розвиток галузі, важливо вирішити ці проблеми через впровадження нових функцій та покращення існуючих сервісів. Наприклад, розширення методів оплати, включення інструментів для вибору конкретних станцій та надання детальної статистики може сприяти підвищенню ефективності та конкурентоспроможності мобільних додатків для зарядки електромобілів.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ДОДАТКУ XAMARIN

2.1Фреймворки та бібліотеки для створення мобільного додатку

Під час написання мобільного додатку було використано фреймворк Xamarin. Xamarin - це фреймворк для крос-платформеної розробки мобільних додатків, який дозволяє розробникам використовувати один і той же код для створення додатків для різних мобільних платформ, зокрема Android та iOS. Заснований на технологіях .NET і C#, Xamarin надає можливість використовувати знайомі інструменти та мови програмування для створення мобільних застосунків.

Xamarin.Forms є фреймворком для створення крос-платформених інтерфейсів користувача. За допомогою Xamarin.Forms розробник може створити єдиний UI, який автоматично адаптується під різні платформи, зменшуючи таким чином потребу в повторному коду.

Xamarin.Native надає можливість розробляти нативні додатки для кожної платформи окремо. Це дозволяє використовувати всі особливості та можливості конкретної платформи. Фреймворк використовує мову програмування C#, яка є потужною та ефективною. Це дозволяє розробникам використовувати знайомий синтаксис та інструменти для створення мобільних додатків. Mono - це реалізація .NET для різних платформ, включаючи Android та iOS. Вона дозволяє виконувати C#-код на різних платформах.

Xamarin повністю інтегрований з Visual Studio, що робить розробку зручною та продуктивною для розробників, які вже працюють з цією інтегрованою розробкою середовищем.

Дозволяє розробникам використовувати однаковий код для створення додатків для різних платформ, а також забезпечує можливість реалізації нативного вигляду та поведінки. Це робить фреймворк популярним в середовищі розробки мобільних додатків.

2.2 Фреймворк Xamarin.Forms

Xamarin.Forms є крос-платформеним фреймворком для розробки мобільних додатків, який дозволяє створювати інтерфейс користувача один раз і використовувати його на різних платформах, таких як iOS, Android та UWP (Universal Windows Platform). Цей фреймворк використовує мову програмування C# та середовище .NET, забезпечуючи широкі можливості для розробників.

Xamarin.Forms використовує мову розмітки XAML для опису користувацького інтерфейсу. Це дозволяє розробникам легко створювати складні інтерфейси та зберігати їхню структуру у вигляді розмітки.

Xamarin.Forms надає широкий спектр елементів керування інтерфейсом, таких як кнопки, тексти, списки, таблиці та інші. Ці елементи легко налаштовуються та інтегруються у дизайн додатку.

Фреймворк пропонує вбудовану підтримку паттерну MVVM, що сприяє розділенню логіки додатку від інтерфейсу. ViewModel взаємодіє з моделлю та представляє дані для відображення на View.

Xamarin.Forms має вбудовані інструменти для навігації між сторінками додатку та обробки маршрутизації. Це дозволяє легко впроваджувати складну навігацію між різними частинами додатку.

З Xamarin.Forms можна використовувати векторні графічні елементи, такі як шляхи та фігури, що дозволяє створювати адаптивні та резинові інтерфейси.

Xamarin.Forms спрощує крос-платформену розробку, дозволяючи розробникам використовувати загальний код для реалізації бізнес-логіки та взаємодії з додатком на різних платформах.

Фреймворк дозволяє використовувати графічні бібліотеки для створення зображень та ефектів, що полегшує реалізацію креативного дизайну.

Xamarin.Forms надає можливості для легкої локалізації додатку, що дозволяє його адаптувати до різних мов та регіональних налаштувань.

У контексті дипломного проекту з розробки мобільного додатку для системи станцій зарядки електромобілів, використання Xamarin.Forms буде ключовим для забезпечення крос-платформеної сумісності та зручного інтерфейсу користувача. Розробник буде використовувати XAML для створення розмітки інтерфейсу, а підтримка MVVM дозволить структурувати логіку додатку та полегшити тестування. Навігація, підтримка векторної графіки та інші функції Xamarin.Forms будуть використані для створення зручного та функціонального додатку.

2.3Фреймворк Xamarin.Android та Xamarin.iOS

Розглянемо фреймворк Xamarin.Android. Xamarin.Android використовує мову програмування C# та .NET для створення мобільних додатків, призначених для операційної системи Android.

Розробники можуть визначити інтерфейс користувача за допомогою XML-розмітки (AXML). Це дозволяє легко створювати та конфігурувати UI елементи, які відповідають нативним елементам Android. Життєвий цикл Activity та Fragment контролює, як додаток взаємодіє з користувацьким інтерфейсом та як управляється станом додатку в процесі його виконання. Xamarin.Android надає API для взаємодії з різними функціями пристрою, такими як камера, геолокація, сенсори та інші.

Використання мови C# та .NET дозволяє розробникам використовувати лаконічний та типобезпечний код, спрощуючи розробку.

Розглянемо фреймворк Xamarin.iOS. Xamarin.iOS також використовує мову програмування C# та .NET, але для створення додатків для операційної системи iOS.

Взаємодія з CocoaTouch, фреймворками для iOS, та Interface Builder для розробки та конфігурації інтерфейсу користувача.

UIViewController та View Lifecycle. Використання життєвого циклу UIViewController та View для керування відображенням та поведінкою елементів у додатку. Взаємодія з iOS SDK дозволяє розробникам використовувати функції, які є характерними для iOS, такі як Touch ID, HealthKit, ARKit тощо.

Здатність інтеграції з Xcode та використання Swift або Objective-C для конкретних частин додатку, якщо це виправдано з точки зору функціональності.

У контексті дипломного проекту, вибір між Xamarin.Android та Xamarin.iOS буде залежати від стратегії підтримки платформ, функціональних особливостей додатку та специфікацій системи станцій зарядки електромобілів. Обидва фреймворки можуть використовуватися для реалізації крос-платформенного додатку, забезпечуючи спільний код для бізнес-логіки та ділової логіки.

2.4Фреймворк MVVMCross

MVVMCross є фреймворком для реалізації архітектури MVVM (Model-View-ViewModel) у Xamarin-додатках. Цей фреймворк дозволяє створювати крос-платформенні додатки з використанням одного коду для бізнес-логіки та подальшої реалізації на різних платформах, таких як Android, iOS, та UWP. Давайте розглянемо основні концепції та компоненти MVVMCross.

ViewModel відповідає за представлення бізнес-логіки та данних для View. Вона ізолює модель від інтерфейсу користувача та надає методи та властивості, необхідні для відображення даних на стороні View.

MVVMCross надає механізм для автоматичної синхронізації даних між View-Model та View за допомогою Data Binding. Це дозволяє оновлювати інтерфейс користувача автоматично при зміні даних у ViewModel.

Модель представляє собою бізнес-логіку та дані додатку. MVVMCross використовує моделі для управління та обробки даних.

Команди дозволяють обробляти дії користувача, такі як натискання кнопок або жестів. MVVMCross дозволяє легко визначати та обробляти команди у ViewModel.

Сервіси використовуються для взаємодії з зовнішніми ресурсами, такими як бази даних, мережеві запити, геолокація та інші. MVVMCross дозволяє використовувати сервіси для розділення логіки додатку.

Навігація в MVVMCross дозволяє переходити між різними сторінками (View) та передавати параметри між ними. Це корисно для реалізації навігаційної логіки в додатку.

MVVMCross використовує принцип внедрення залежностей для управління створенням та конфігурацією об'єктів. Це дозволяє вам ефективно використовувати сервіси та інші компоненти, не обов'язково створюючи їх у кожному місці.

MVVMCross підтримує використання плагінів для розширення функціональності. Це дозволяє вам легко включати та використовувати розширення для різноманітних задач, таких як камера, геолокація, картографія тощо.

View Binding дозволяє автоматично знаходити та зв'язувати елементи інтерфейсу користувача у View з відповідними властивостями у ViewModel.

У контексті дипломного проекту MVVMCross може бути використаний для створення структурованої та розширюваної архітектури для додатку системи станцій зарядки електромобілів. Використання ViewModel дозволить ефективно розділити логіку додатку та відокремити її від інтерфейсу користувача. Прив'язка даних спростить відображення даних у View, а використання сервісів та плагінів забезпечить ефективну взаємодію з різними функціональними можливостями, такими як геолокація та інші.

2.5 Фреймворки Prism та Refit

Prism пропонує модульний підхід до розробки, де функціональність додатку поділяється на модулі. Це дозволяє легко розширювати та змінювати

функціональність додатку. Інверсія керування допомагає зменшити залежності між компонентами. Prism базується на патерні MVVM, забезпечуючи чітку відокремленість між інтерфейсом користувача та бізнес-логікою. Це полегшує тестування та розширення додатку.Prism надає ефективні засоби навігації між сторінками (View) додатку. Це дозволяє легко впроваджувати та керувати навігаційною логікою.Механізм EventAggregator в Prism дозволяє взаємодіяти між модулями, обмінюючи подіями. Це створює слабкі зв'язки та полегшує розширення додатку. Prism інтегрується з DI-контейнерами для внедрення залежностей. Це полегшує створення та керування об'єктами та сприяє їх відокремленню.

Refit надає простий та зручний HTTP-клієнт для взаємодії з веб-серверами. Декларативні запити використовують атрибути C# для визначення структури запитів. Refit використовує інтерфейси C# для визначення API, що робить код чистим та легко зрозумілим. Це сприяє взаємодії з API та забезпечує типобезпечний код. Refit автоматично виконує серіалізацію та десеріалізацію даних між додатком та веб-сервером. Це зменшує кількість ручного коду для обробки даних. Refit дозволяє легко модифікувати запити за допомогою атрибутів, наприклад, для додавання заголовків чи параметрів. Refit підтримує асинхронні запити, що дозволяє ефективно взаємодіяти з API, не блокуючи основний потік додатку.

В процесі роботи Prism може бути використаний для побудови структурованої архітектури додатку системи станцій зарядки електромобілів, забезпечуючи ефективну роботу з інтерфейсом користувача та бізнес-логікою. Refit використовуватиметься для взаємодії з серверами системи станцій зарядки через REST API, надаючи простий та декларативний спосіб виконання HTTP-запитів та обробки даних.

2.6 Фреймворки SQLite.NET

SQLite.NET використовує ORM для взаємодії з базою даних SQLite. ORM дозволяє вам взаємодіяти з базою даних, використовуючи об'єктно-орієнтовану модель даних, а не SQL-запити безпосередньо. SQLite.NET дозволяє визначати моделі даних, які відображаються на таблиці в базі даних. Це називається TableMappings. Моделі представляють об'єкти, які будуть зберігатися в базі даних. SQLite.NET надає зручні методи для виконання операцій створення, читання, оновлення та видалення даних в базі даних. Це дозволяє легко взаємодіяти з даними в коді додатку. SQLite.NET може працювати з локальною базою даних, що дозволяє зберігати дані безпосередньо на пристрої. Це особливо корисно для додатків, які потребують доступу до даних навіть без Інтернет-з'єднання. SQLite.NET підтримує транзакції, що дозволяє групувати кілька операцій в одну транзакцію. Це важливо для забезпечення консистентності даних при взаємодії з базою даних.

SQLite.NET дозволяє визначати індекси для полів таблиць, що сприяє оптимізації запитів та прискорює пошук даних. Бібліотека підтримує асинхронні операції, що дозволяє виконувати операції з базою даних асинхронно, що може покращити відзивчивість додатку. SQLite.NET надає підтримку міграцій бази даних, що дозволяє оновлювати схему бази даних при змінах у моделях даних без втрати даних. Для забезпечення безпеки, SQLite.NET підтримує шифрування бази даних, що дозволяє захистити конфіденційні дані в додатку.

У дипломному проекті SQLite.NET може бути використаний для реалізації локального сховища даних для інформації про станції зарядки електромобілів. Він дозволяє ефективно зберігати, оновлювати та отримувати дані про зарядні станції на пристрої користувача. Можливість використання транзакцій, оптимізація запитів та інші функції допоможуть забезпечити швидку та надійну роботу з базою даних у вимогливому середовищі мобільних додатків.

2.7 Бібліотека Xamarin.Essentials

Xamarin.Essentials - це набір надзвичайно корисних бібліотек та інструментів, які призначені для спрощення розробки мобільних додатків на платформі Xamarin. Ця бібліотека надає простий та консистентний API для доступу до основних функцій пристроїв, таких як камера, геолокація, датчики, акумулятор, мережа та багато інших. Завдяки Xamarin.Essentials розробники можуть швидко і легко інтегрувати функціональність пристроїв в свої додатки з мінімальними зусиллями і безпекою, що дозволяє покращити взаємодію з апаратурою та надавати користувачам зручний інтерфейс, оптимізований для їхніх пристроїв. За допомогою Xamarin.Essentials розробники можуть прискорити процес розробки, збільшити переносимість коду між різними платформами та забезпечити стабільні та продуктивні додатки для платформ Android, iOS та UWP.

Xamarin.Essentials - це не тільки зручний, але й потужний інструмент для розробників мобільних додатків на платформі Xamarin. Вона включає в себе більше 30 модулів, які охоплюють широкий спектр можливостей пристроїв. Серед найбільш важливих функцій можна виділити доступ до камери (фото та відео), геолокації (з отриманням поточних координат та маршрутів), акумулятора (інформація про рівень заряду та статус зарядження), мережі (перевірка доступності мережі, тип мережі), датчиків (акселерометр, компас, гіроскоп), вібрації, текстовий до зображення (OCR), системний звук, календар та інші. Крім того, Xamarin.Essentials надає інтерфейси для роботи з системним кліпбордом, прискоренням, браузером, клавіатурою та управлінням енергоспоживанням. Це робить бібліотеку важливим інструментом для розробників, які хочуть ефективно використовувати функціонал мобільних пристроїв у своїх додатках, забезпечуючи високий рівень зручності та продуктивності для кінцевих користувачів.

2.8 Бібліотека Newtonsoft.Json

Newtonsoft.Json, часто відома просто як Json.NET, є однією з найпопулярніших бібліотек для роботи з форматом JSON в середовищі .NET. Розроблена Лемаром Гіттінгзом, вона надає потужні та гнучкі інструменти для серіалізації та десеріалізації об'єктів .NET у формат JSON та навпаки. Json.NET підтримує багато функцій, таких як вкладені об'єкти, атрибути для керування процесом серіалізації, валідація та власні конвертери для роботи зі складними типами даних. Ця бібліотека дозволяє легко та ефективно взаємодіяти з веб-сервісами, обмінюватися даними між додатками та зберігати конфігураційні параметри у структурі JSON, що робить її важливим інструментом для розробників .NET.

Основні процеси, які виконує ця бібліотека, включають серіалізацію та десеріалізацію об'єктів .NET у формат JSON та навпаки. Розуміння роботи Json.NET може бути розділене на кілька ключових аспектів.

З .NET об'єктів у JSON: Коли потрібно перетворити об'єкти .NET додатку в формат JSON для передачі або збереження, можна використовувати процес серіалізації. Json.NET дозволяє автоматично перетворювати об'єкти у валідний JSON-рядок.

З JSON у .NET об'єкти: Якщо є JSON-рядок, і потрібно відновити об'єкт .NET з цього рядка, можна використовувати процес десеріалізації. Json.NET розуміє структуру JSON і може автоматично відновити об'єкти вашого класу.

Json.NET підтримує використання атрибутів для контролю над процесами серіалізації та десеріалізації. Наприклад, можна використовувати атрибут [JsonProperty] для вказівки імені поля чи властивості в JSON. Також доступні різноманітні конфігураційні параметри для налаштування різних аспектів роботи Json.NET.

Json.NET легко справляється з вкладеними об'єктами та колекціями. Він автоматично рекурсивно обробляє структуру даних та виконує серіалізацію та десеріалізацію.

2.9 Бібліотека Xamarin.Auth

Бібліотека Xamarin.Auth є інструментом для роботи з автентифікацією в Xamarin-проектах. Вона дозволяє зручно виконувати процеси аутентифікації та авторизації в додатках, спрощуючи взаємодію з різними сервісами, такими як соціальні мережі, веб-сайти, або інші автентифікаційні системи.

Основні функції Xamarin.Auth включають можливості описані далі.

Xamarin.Auth дозволяє легко взаємодіяти з різними сервісами, такими як Facebook, Twitter, Google, Microsoft, Dropbox, і багатьма іншими. Вона надає готові реалізації для роботи з OAuth та OpenID Connect.

Забезпечує шаблони для використання стандартних протоколів аутентифікації, таких як OAuth та OAuth2. Це робить процес автентифікації простішим та консистентним для різних провайдерів.

Дозволяє зберігати та відновлювати сесії автентифікації, щоб користувач не повинен був знову вводити свої дані при кожному вході.

Xamarin.Auth інтегрується з нативними браузерами та іншими додатками для зручної автентифікації та отримання доступу до ресурсів.

2.10 Credential Manager огляд та застосування

Credential Manager є інструментом для безпечного зберігання та керування обліковими даними користувача на операційній системі. Основна його функція - забезпечити безпечний доступ до різних ресурсів та послуг, не вимагаючи повторного введення інформації користувачем.

Credential Manager дозволяє зберігати логіни та паролі для різних облікових записів, полегшуючи процес авторизації та уникнення необхідності запам'ятовування кожного пароля. Використовуючи дані, збережені в Credential Manager, операційна система може автоматично заповнювати форми на вебсайтах або в інших додатках, що вимагає введення облікових даних.

Цей інструмент використовує різні механізми шифрування для захисту збережених облікових даних, забезпечуючи високий рівень безпеки. Його основна перевага полягає в тому, що він дозволяє користувачам уникнути несанкціонованого доступу до важливої інформації.

Credential Manager може пропонувати можливість синхронізації збережених облікових даних між різними пристроями користувача, забезпечуючи зручний доступ до них з різних платформ. Це робить його особливо зручним для тих, хто використовує декілька пристроїв.

Загалом, використання Credential Manager сприяє безпеці та зручності використання облікових даних, полегшуючи процес авторизації та забезпечуючи ефективний доступ до різних ресурсів.

2.11 Особливості платформ Android та iOS

Розгортання UI для Android і iOS у фреймворку Xamarin може відрізнятися в різних аспектах через особливості кожної з платформ

2.11.1 Нативний вигляд Android

Нативний вигляд Android вимагає ретельного врахування особливостей платформи для створення інтерфейсу користувача, який відповідає стандартам та естетичним вимогам Android. У Xamarin для Android, можна використовувати XML-розмітку для опису вигляду та розміщення елементів інтерфейсу. Розглянемо ключові аспекти нативного вигляду Android.

Використання XML-розмітки для опису інтерфейсу дозволяє легко розділити дизайн та логіку. Кожен макет може містити різні види елементів інтерфейсу, такі як Button, TextView, ImageView тощо.

Використання ресурсів для визначення кольорів, розмірів, рядків тексту тощо. Директиви розмітки (dp - пікселі з незалежністю від щільності пікселів) дозволяють створювати адаптивний дизайн для різних розмірів екранів.

Android використовує різні менеджери розміщення (layout managers) для керування розташуванням та виглядом елементів. Наприклад, LinearLayout, RelativeLayout, ConstraintLayout. Використання нативних елементів Android,

таких як Button, TextView, ImageView для забезпечення консистентності з іншими додатками на платформі, використання тем та стилів для задання зовнішнього вигляду та підтримки рекомендацій по дизайну Android, використання ресурсів для підтримки різних мов та культур.

Розгортання нативного вигляду Android у Xamarin дозволяє реалізовувати інтерфейс, який гармонійно вписується в екосистему Android та надає користувачам знайомий та комфортний досвід взаємодії.

2.11.2 Нативний вигляд iOS

Створення нативного вигляду для iOS у Xamarin включає в себе використання інструментів та підходів, характерних для розробки на цій платформі. Нижче наведено ключові аспекти створення нативного вигляду для iOS у фреймворку Xamarin.

Використання Storyboards та Interface Builder для графічного створення інтерфейсу користувача. У візуальному редакторі можна додавати елементи інтерфейсу, встановлювати зв'язки між елементами та визначати їх властивості.

Використання Auto Layout для автоматичного адаптування інтерфейсу до різних розмірів екранів та орієнтацій пристрою. Система Auto Layout дозволяє створювати гнучкі та адаптивні інтерфейси. Використання стандартних нативних елементів управління iOS, таких як UIButton, UILabel, UITextField, UIImageView та інших, для підтримки звичних відчуттів та стилів iOS. Використання XIB-файлів (Nib-файлів) для визначення окремих екранів чи компонентів інтерфейсу. XIB-файли дозволяють розділити інтерфейс та логіку.

Визначення стилів та кольорів за допомогою Asset Catalog. Asset Catalog дозволяє організувати графічні ресурси та дотримуватися рекомендацій по дизайну iOS. Підтримка локалізації та мультиязичності за допомогою файлів .strings та Asset Catalog. Це дозволяє забезпечити переклади для різних мов та регіонів. Застосування тем та стилів для зміни вигляду додатку в цілому. До-

датки iOS можуть використовувати теми App Appearance для зміни зовнішнього вигляду за налаштуваннями користувача (наприклад, світла або темної теми). Можливість додавання та налаштовування елементів інтерфейсу безпосередньо в коді за допомогою мови програмування Swift або Objective-C.

Створення нативного вигляду для iOS в Xamarin дозволяє розробникам ефективно використовувати інструменти та практики, що властиві розробці під цю платформу, та забезпечує гармонійний інтерфейс для користувачів iOS-пристроїв.

2.12 Розміщення та розміри екрану

2.12.1 Розміщення та розміри екрану для Android

Використовуючи LinearLayout, можна визначити орієнтацію (вертикальну чи горизонтальну) та розміщення елементів відносно один одного.

RelativeLayout дозволяє розміщувати елементи відносно інших елементів або в краях батьківського елемента.

ConstraintLayout дозволяє використовувати обмеження для точного визначення положення та розмірів елементів. Вказання конкретних розмірів та координат для кожного елементу в макеті. Для адаптації до різних розмірів екранів використовуються різні значення розмірів та розміщення. Визначення розмірів у dimens.xml та використання різних розмірів для різних розмірів екрану.

2.12.2 Розміщення та розміри екрану для iOS

Використання Storyboards та Interface Builder для графічного розміщення елементів інтерфейсу та налаштування їх розмірів.

Використання Auto Layout для створення гнучких інтерфейсів, які адаптуються до різних розмірів екранів.

Можливість програмного розміщення та розмірів елементів через мови програмування Swift або Objective-C. Використання констрейнтів для точного визначення положення та розміру елементів.

2.12.3 Порівняльна характеристика

Зазвичай розробка для Android включає в себе більше розмірів екранів та пристроїв з різними параметрами. Необхідно управляти розмірами в пікселях та логікою для підтримки різних екранів.

Для iOS використання Auto Layout дозволяє створювати гнучкі та адаптивні інтерфейси для різних розмірів iPhone та iPad. Розміщення елементів може здійснюватися за допомогою інтерфейсу та коду, що робить розміщення більш гнучким.

Обидві платформи мають засоби для розміщення та розмірів екрану, але підхід інший через використання різних інструментів та підходів.

Використання ресурсів та констрейнтів дозволяє розробникам створювати адаптивні та красиві інтерфейси для різних пристроїв.

2.13 Особливості проектування для Android та iOS

Під час проектування мобільного додатка для платформ Android та iOS важливо враховувати особливості кожної з них. Для Android властивий Material Design, який використовує характерні для платформи матеріальні ефекти, водяні знаки та тіні. Основний принцип - створення інтуїтивно зрозумілого та візуально привабливого інтерфейсу.

На Android використовуються фрагменти, які дозволяють розділити екран на компоненти для більшої гнучкості при адаптації до різних розмірів та орієнтацій пристроїв. Однак розробники повинні враховувати широкий спектр пристроїв і їх різноманітність, що може вимагати додаткового стилевого та функціонального налаштування.

Для iOS характерний Human Interface Guidelines, який націлений на створення мінімалістичного та лаконічного дизайну. Використання Storyboard та

Auto Layout дозволяє створювати універсальні інтерфейси для різних розмірів екранів, пристроїв та орієнтацій. Тут основна ідея - створення єдинообразного інтерфейсу для різних пристроїв.

Обидві платформи мають свої унікальні вимоги та принципи, і розробники повинні бути свідомі цих особливостей при проектуванні та виробництві мобільних додатків. Забезпечення високоякісного користувацького досвіду вимагає ретельного вивчення та використання рекомендаційних інструментів та дизайнерських стандартів для кожної конкретної платформи.

2.14 Інтеграція з бібліотеками

Під час інтеграції з бібліотеками на платформі Android, використовується система управління залежностями Gradle. Цей інструмент надає можливість додавання та керування залежностями за допомогою спеціальних файлів конфігурації, таких як build.gradle. Розробники можуть отримувати доступ до багатьох готових бібліотек через репозиторії, наприклад, JCenter або Maven Central.

У свою чергу, на платформі iOS інтеграція з бібліотеками відбувається через CocoaPods або Carthage. CocoaPods дозволяє автоматизувати процес встановлення та оновлення бібліотек, роблячи цей процес більш простим та зручним для розробників. З іншого боку, Carthage надає більше гнучкості та контролю над процесом інтеграції, дозволяючи розробникам керувати завантаженням та оновленням бібліотек.

Одна з ключових відмінностей полягає у виборі мов програмування. Android підтримує Java та Kotlin, що дозволяє використовувати різні бібліотеки для обох мов. У той час як iOS використовує Swift та Objective-C, і розробники повинні враховувати цю різницю під час вибору бібліотек.

Таблиця 1 — Порівняння Xamarin.Android та Xamarin.iOS

Характеристика	**Xamarin.Android**	**Xamarin.iOS**
Дизайн та Архітектура Інтерфейсу	Дотримання дизайну матеріалу, характерного для Android.	Акцент на дизайні, типовому для iOS, з використанням CocoaTouch та інструментів для ефективної розробки.
Мапінг на Нативні Елементи	Використання XML-розмітки для мапінгу на Android елементи.	Можливість використовувати XIB або програмний код для визначення інтерфейсу на iOS.
Життєвий Цикл Компонентів	Використання життєвого циклу Activity та Fragment.	Використання життєвого циклу UIViewController та View.
Специфічні Функції Програмування	Взаємодія з Java API та Android SDK.	Взаємодія з CocoaTouch та iOS SDK.
Мовлення Програмування	Використання C# та .NET для розробки.	Використання C# та .NET для розробки.

У підсумку, обидві платформи пропонують зручні інструменти для інтеграції залежностей, і вибір між ними може залежати від конкретних вимог проекту та особистих вподобань розробника.

2.15 Порівняння продуктивності розробки мобільних додатків в Xamarin з іншими крос-платформовими рішеннями

Порівняння продуктивності розробки мобільних додатків в Xamarin з іншими крос-платформовими рішеннями може бути складним завданням, оскільки багато факторів можуть впливати на вибір та ефективність різних інструментів. Однак ось кілька ключових аспектів, які можна порівняти:

Xamarin: Xamarin дозволяє використовувати спільний код для реалізації бізнес-логіки та взаємодії з сервером. Можна поділяти до 90% коду між Android і iOS. Використовує нативні компоненти і вигляд на кожній платформі, що забезпечує більш автентичний вигляд додатку. Однак потрібно слідкувати за оновленнями відповідних бібліотек. З використанням Xamarin можна легко інтегрувати функції нативної платформи, оскільки можна використовувати бібліотеки на різних мовах програмування (C#, Swift, Objective-C, Java). Має активну спільноту та підтримку від Microsoft.

Flutter: Flutter використовує Dart і дозволяє створювати крос-платформові додатки з однієї кодової бази. Він використовує власний рендеринг для створення відмінного вигляду на різних платформах. Flutter має власний движок відображення і використовує власні віджети. Це може дозволити більшу гнучкість у дизайні, але може потребувати більше зусиль для досягнення нативного вигляду. Має велику кількість власних плагінів та підтримує інтеграцію з різними функціями платформ. Швидко росте та має активну спільноту, спонсоровану Google.

React Native: React Native дозволяє використовувати JavaScript і React для створення крос-платформових додатків. Застосування нативних компонентів допомагає досягти високої швидкості розробки. React Native використовує нативні компоненти для відображення на екрані, що дозволяє досягти високого ступеня нативного вигляду. Є широкий вибір сторонніх бібліотек та модулів для React Native, що полегшує інтеграцію нових функцій. Дозволяє

використовувати компоненти React Native на обох платформах Має велику та активну спільноту, підтримується Facebook.

Враховуючи, що вибір крос-платформеного рішення також буде залежати від специфічних потреб проекту, навичок команди розробників та інших факторів. Кожна з цих технологій має свої переваги та обмеження, тому важливо враховувати контекст проекту при виборі.

Таблиця 2 — Порівняння продуктивності розробки мобільних додатків в Xamarin з іншими крос-платформовими рішеннями.

Характеристика	**Xamarin**	**Flutter**	**React Native**
Мова Програмування	C#	Dart	JavaScript (React)
Швидкість Розробки	Висока, до 90% спільного коду	Висока, один код для обох платформ	Висока, можливість використання React Native
Дизайн та Вигляд	Використання нативних компонентів	Власний рендеринг, гнучкість у дизайні	Використання нативних компонентів
Широкий Спектр Функцій	Легка інтеграція нативних функцій	Широкий вибір власних та сторонніх плагінів	Широкий вибір сторонніх бібліотек та модулів
Спільні Компоненти	Можливість використовувати спільні бібліотеки	Має велику кількість власних віджетів	Дозволяє використовувати компоненти React Native
Спільнота та Підтримка	Активна спільнота, підтримка від Microsoft	Швидко ростуча спільнота, підтримка від Google	Велика та активна спільнота, підтримується Facebook

2.12 Висновки з розділу 2

1. Процес розробки мобільних додатків на платформі Xamarin для станцій зарядки електромобілів виявився завданням, яке об'єднує в собі численні технічні та дизайнерські виклики. Використання Xamarin дозволяє розробникам створювати крос-платформенні додатки, що ефективно працюють як на Android, так і на iOS, знижуючи витрати часу та ресурсів.
2. Однією з ключових складових розробки є використання Xamarin.Forms, яке дозволяє створювати єдиний код для інтерфейсу та бізнес-логіки для обох платформ. Засоби реалізації, такі як Xamarin.Essentials, спрощують взаємодію з функціоналом пристроїв та покращують загальний досвід користувача.
3. Інтеграція з бібліотеками на кожній з платформ (Android та iOS) відбувається за допомогою Gradle та CocoaPods або Carthage відповідно. Кожен з цих інструментів має свої особливості, і вибір може залежати від конкретних потреб проекту та уподобань розробника.
4. При розробці для Android та iOS важливо враховувати їхні унікальні особливості та дизайнерські принципи, такі як Material Design для Android та Human Interface Guidelines для iOS. Інтеграція нативного вигляду та оптимізація для різних розмірів екранів є важливими етапами розробки для забезпечення максимального комфорту користувачів на обох платформах.

Усі ці аспекти важливі для успішного розгортання мобільного додатка для станцій зарядки електромобілів, і вони відображають інтегрований та виважений підхід до розробки, який враховує технічні, дизайнерські та платформозалежні вимоги.

РОЗДІЛ 3 РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ СТАНЦІЙ ЗАРЯДКИ ЕЛЕКТРОМОБІЛІВ

3.1Створення мобільного додатку за допомогою фреймворка Xamarin

3.1.1 Налаштування середовища для розробки

Створення мобільного додатка за допомогою Xamarin вимагає налаштування спеціалізованого середовища для розробки. Розробку додатку було проведено на комп'ютерах з операційними системами Windows та MacOS з наступними характеристиками.

Комп'ютер з операційною системою Windows:

1. Чотирьохядерний процесор Intel Core i7-8550U з тактовою частотою 1.8 ГГц.
2. Графічний процесор NVIDIA GeForce GTX 1050.
3. 16 ГБ оперативної пам'яті.
4. Операційна система Windows 10.

Комп'ютер з операційною системою MacOS:

1. Процесор Apple M1.
2. Графічний процесор Apple M1.
3. 8 ГБ оперативної пам'яті.
4. Операційна система MacOS Sonoma 14.1.

Для розгортання додатку необхідно встановити фреймворк Xamarin та його залежності. Для встановлення фреймворка Xamarin були виконані наступні кроки.

Створення мобільного додатка за допомогою Xamarin вимагає налаштування спеціалізованого середовища для розробки. Нижче наведено детальні кроки налаштування:

1. Встановлення Visual Studio.

Завантажено та встановлено Visual Studio, була обрана опція "Mobile development with .NET" під час встановлення.

2. Встановлення Xamarin.
   У Visual Studio, оберати "Tools" -> "Get Tools and Features".
3. Встановлено "Mobile development with .NET" та "Xamarin" у списку робочих навичок.
4. Встановлення Android SDK
   Було обрано "Tools" -> "Android" -> "Android SDK Manager".
   Встановлені необхідні компоненти для Android розробки.
5. Встановлення Android Emulator
   Було обрано "Tools" -> "Android" -> "Android Device Manager".
   Додано та налаштовано віртуальний пристрій.
6. Встановлення Xcode (для розробки для iOS)
   Завантажено та встановлено Xcode з App Store.
7. Встановлення Xamarin.iOS
   На шлянху "Tools" -> "Get Tools and Features" встановлено "iOS development with .NET" у списку робочих навичок.

Після перевірки встановленого фреймворка Xamarin та необхідних залежностей було створено та розгорнуто новий проект. Приклад початкового проекту наведено на рисунку нижче

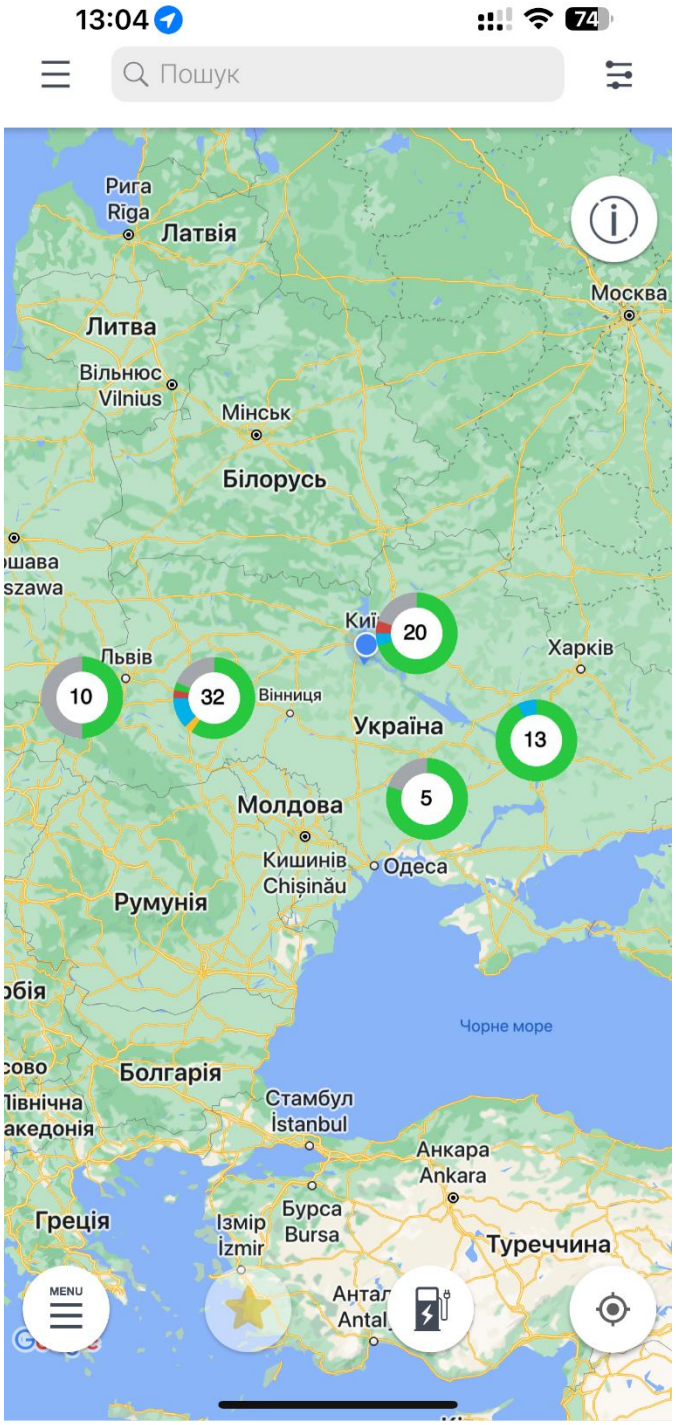

Рисунок 3.1 — Ініціалізація початкового екрану додатку для зарядки електростанцій

3.1.2 Підготовка застосунку для роботи з даними користувача

Наступним кроком після налаштування та розгортання додатку на різних платформах було реалізовано основний функціонал системи зарякди. Були додані основні запити до API, які дозволили реалізувати базовий функціонал для користувачів, такі як початок зарядної сесії, резерв портів, відображення станцій на мапі, загрузка та відображення статистики користувача, додані різноманітні методи оплати та можливість зберігання даних користувача в додатку. Для коректної роботи додатку компанією «Infocom LTD» була надана база даних для перевірки роботи та тестування додатку.

Проект був розпочатий із проектування та аналізу вимог. Визначалися функціональні та нефункціональні вимоги для розробки мобільного додатка для зарядних станцій електромобілів. Обрано тип додатку (Xamarin.Forms) та встановлено необхідні пакети NuGet для роботи з мережевими запитами та іншими складовими.

Далі була проведена реалізація основних запитів до API. Використовуючи бібліотеки для роботи з HTTP-запитами, були налаштовані запити для різних функціональностей, таких як початок зарядної сесії, резерв портів, отримання інформації про станції та інше.

Для відображення станцій на мапі була використана інтеграція з картами, така як Google.Maps.

Також були реалізовані функції для відображення статистики користувача та різноманітні методи оплати. Дані користувача були збережені в додатку, використовуючи вбудовані механізми зберігання даних або відповідні бібліотеки, такі як Xamarin.Essentials Secure Storage чи інші для Xamarin.Native.

Весь розроблений функціонал був підданий тестуванню. Забезпечено коректну роботу всіх функцій та вирішено всі виявлені під час тестування проблеми.

Лістинг 1 демонструє приклад реалізації завантаження та підписки на оновлення даних станцій з серверу . Функція `SubscribeToAllStations`

створює новий ConcurrentDictionary, завантажує та додає у нього об’єкт кожної станції. Якщо словник вже має об’єкт такої станції, то відбувається лише оновлення даних у словнику. У функції також реалізований механізм очікування даних від серверу, що надає змогу слідкувати за змінами, що відбуваються на станції задля того, щоб користувач завжди отримував актуальний стан станції.

Лістинг 1 — Вміст функціїї `SubscribeToAllStations`

```
public static async Task SubscribeToAllStations()
{
    try
        {
          string changes = "first";
          bool isStationsNumberChanged = false;
          var stationsData = await
App.DataManager.GetAllStationsLongPolling("first",
App.CancelGettingAllStation?.Token);
if(stationsData != default(List<StationData>)&&
stationsData != null)
        {
          App.AllStations.Clear();
    foreach(StationData stationData in stationsData)
    {
    Station station = new Station();
    station.CopyParams(stationData);
    App.AllStations[station.Id] = station;
                        }

    Xamarin.Forms.Device.BeginInvokeOnMainThread(() =>
App.StationsUpdated?.Invoke(isStationsNumberChanged));
    }
```

```
    while (true)
    {
    CancelGettingAllStation?.Token.ThrowIfCancellationR
equested();

    if (stationsData != null)
    {
    foreach (var newStation in stationsData)
    {
    if (App.AllStations.ContainsKey(newStation.Id))
    {
    foreach (var newPort in newStation.Ports)
    {
    var currPort =
App.AllStations[newStation.Id].Ports.FirstOrDefault(p
=> p.Id == newPort.Id);
    if (currPort != null)
    {
    currPort.Status = newPort.Status;
currPort.Connected = newPort.Connected;
currPort.EstimatedStopDateEpochtime =
newPort.EstimatedStopDate?.ToEpochtime();

currPort.ConnectionChangeDateEpochtime                =
newPort.ConnectionChangeDate?.ToEpochtime();
currPort.ChargedPercent = newPort.ChargedPercent;
var session = App.ChargingSessions.FirstOrDefault(s =>
s.StationId == newStation.Id && s.PortId == currPort.Id);
if (session != null)
```

```
    {
    session.Port = currPort;
    //Xamarin.Forms.Device.BeginInvokeOnMainThread(()
=> App.ChargingSessionsUpdated?.Invoke(changes));
    Xamarin.Forms.Device.BeginInvokeOnMainThread(()   =>
App.StationsUpdated?.Invoke(isStationsNumberChanged));
    }
    else
    {
      await App.CheckInternetConnection();
    }
    var buffStations = App.AllStations.Values;
    await                              Task.Delay(2000,
CancelGettingAllStation.Token);
    isStationsNumberChanged = false;
    changes = "next";
    stationsData                =                 await
App.DataManager.GetAllStationsLongPolling("next",
CancelGettingAllStation?.Token);
    }
    }
    catch (TaskCanceledException ex)
    {
    Debug.WriteLine($"SubscribeGetStationsLongPolling
task was canceled: {ex?.Message}");
    }
    catch (OperationCanceledException ex)
    {
    Debug.WriteLine($"SubscribeGetStationsLongPolling
task was canceled: {ex?.Message}");
```

```
    }
    catch (Exception ex)
    {
    Debug.WriteLine($"SubscribeGetStationsLongPolling
exception: {ex?.Message}");
    await                                Task.Delay(2000,
CancelGettingAllStation.Token);
    Task.Run(App.SubscribeToAllStations).Forget();
    Crashes.TrackError(ex);
    }
    }
```

На рисунку 3.2 зображені завантажені станції та їх стан на поточний період

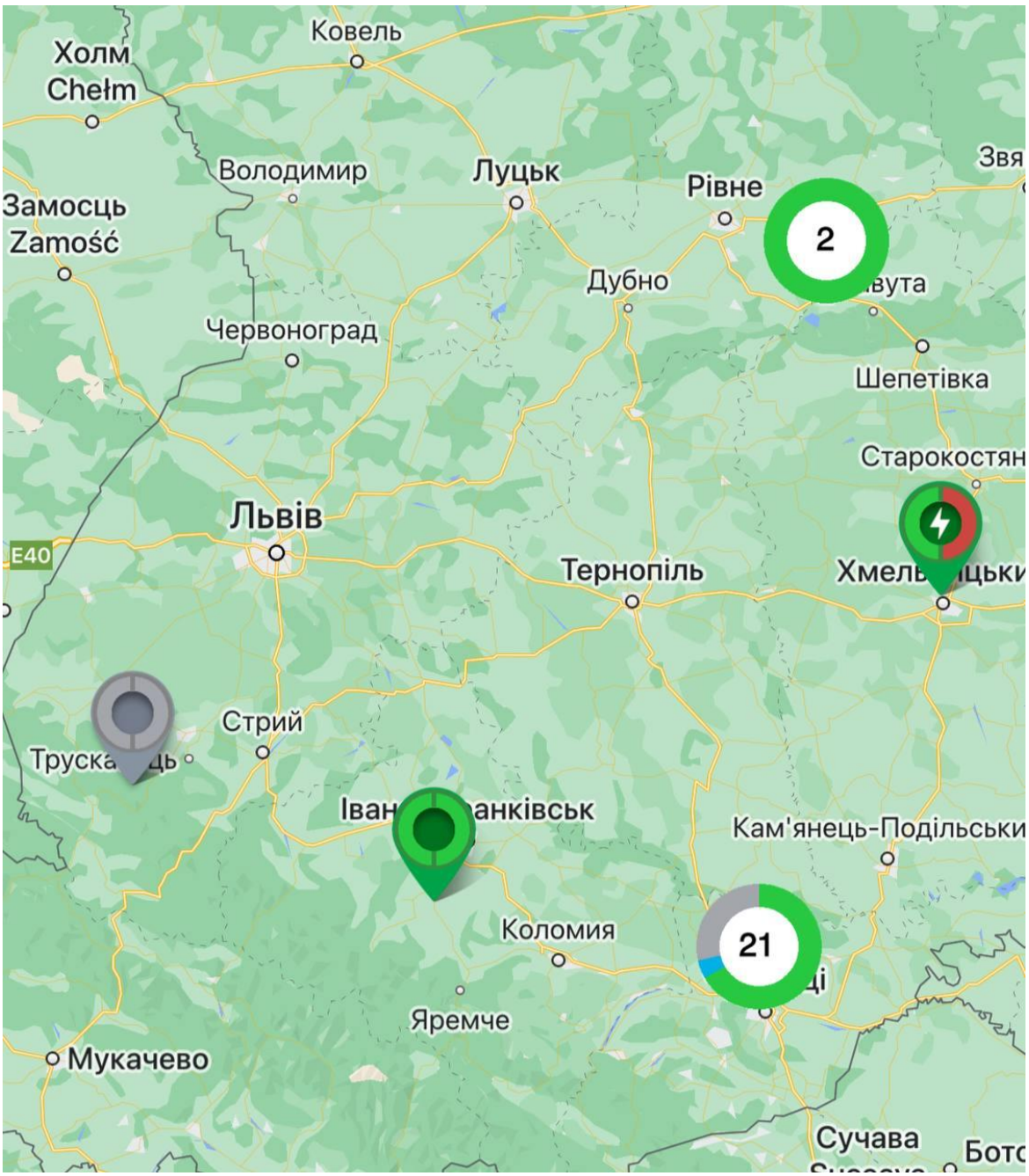

Рисунок 3.2 — Завантажені станції та їх стан

Для коректного зберігання інформації отриманої від сервера, були реалізовані моделі даних, для роботи з моделлю додаток використовує контролери, для відображення даних користувачу були реалізовані View.

3.1.3 Робота зі зберіганням даних у пам’яті телефону

Під час створення мобільного додатку виникла потреба реалізувати можливість зберігання даних користувача у внутрішній пам’яті телефону. Чому виникла така потреба? По-перше, користувач повинен мати змогу починати зарядку не маючи акаунту, так звані «гостьові» сесії. По-друге, користувач повинен мати можливість додати платіжну картку до системи, при цьому не поширювати персональні дані до серверу. Саме тому було запропоновано розглянути роботу з бібліотекою Xamarin.Auth.

Для реалізації механізму безпечного зберігання даних було реалізовано інтерфейс ICredentialsManager у спільному для обох платформ проекті. Приклад реалізації інтерфейсу наведений у лістингу 2.

Лістинг 2 — Вміст інтерфейсу ICredentialsManager

```
namespace UGVChargers.Helpers
{
    public interface ICredentialsManager
    {
        string Login { get; }
        CookieContainer Cookies { get; }
        CookieContainer GuestCookies { get; }
        Task SaveCredentials(string login,
CookieContainer cookies);
        Task SaveGuestCredentials(CookieContainer
cookies);
        Task DeleteCookies();
        Task DeleteGuestCookies();
    }
}
```

У проекті Xamarin.iOS може бути реалізований клас CredentialsManager, що втілює інтерфейс ICredentialsManager. В цьому класі реалізовано методи для зберігання та отримання логіну та cookies.

Клас має зчитувати та зберігати дані логіну та cookies в безпечний спосіб. Для збереження використовуються безпечні механізми, можливо, використовуючи KeyChain або подібні засоби для збереження конфіденційних даних.

Клас повинен мати метод для зчитування логіну та cookies збережених раніше. Це може включати в себе використання безпечних механізмів для отримання конфіденційних даних.

Реалізація методів для збереження логіну та cookies. Збереження повинно відбуватися в безпечному сховищі, такому як KeyChain на iOS. Метод для видалення гостьових cookies, якщо вони також зберігаються окремо. Приклад реалізації для iOS проекту наведено у лістингу 3.

Лістинг 3 — Вміст класу `CredentialsManager`

```
public class CredentialsManager : ICredentialsManager
{
string ICredentialsManager.Login
{
get
{
var account = this.GetAccount().Result;
return account?.Username;
}
}

CookieContainer ICredentialsManager.Cookies
{
get
{
var account = this.GetAccount().Result;
```

```
return account?.Cookies;
}
}

CookieContainer ICredentialsManager.GuestCookies
{
get
{
var account = this.GetAccount(true).Result;
return account?.Cookies;
}
}

async Task ICredentialsManager.SaveCredentials(string login,
CookieContainer cookies)
{
if (!string.IsNullOrWhiteSpace(login))
{
var oldAccount = await this.GetAccount();
if (oldAccount != null)
{
await AccountStore.Create().DeleteAsync(oldAccount, App.Ap-
pName);
}

Account account = new Account(login, new Dictionary<string,
string>() { { "General", "true" } }, cookies);
await AccountStore.Create().SaveAsync(account, App.AppName);
}
}

async Task ICredentialsManager.SaveGuestCredentials(Cook-
ieContainer cookies)
```

```
{
var oldAccount = await this.GetAccount(true);
if (oldAccount != null)
{
await AccountStore.Create().DeleteAsync(oldAccount, App.Ap-
pName);
}

Account account = new Account("Guest", new Dictionary<string,
string>() { { "Guest", "true" } }, cookies);
await AccountStore.Create().SaveAsync(account, App.AppName);
}

async Task ICredentialsManager.DeleteCookies()
{
var account = await this.GetAccount();
if (account != null)
{
await AccountStore.Create().DeleteAsync(account, App.Ap-
pName);
Account updatedAccount = new Account(account.Username, ac-
count.Properties, null);
await AccountStore.Create().SaveAsync(updatedAccount,
App.AppName);
}
}

async Task ICredentialsManager.DeleteGuestCookies()
{
var account = await this.GetAccount(true);
if (account != null)
{
```

```
await AccountStore.Create().DeleteAsync(account, App.Ap-
pName);
Account updatedAccount = new Account(account.Username, ac-
count.Properties, null);
await AccountStore.Create().SaveAsync(updatedAccount,
App.AppName);
}
}

private async Task<Account> GetAccount(bool isGuest = false)
{
var accounts = await AccountStore.Create().FindAc-
countsForServiceAsync(App.AppName);
try
{
string key = isGuest ? "Guest" : "General";
return accounts?.Find(account => account.Properties.Con-
tainsKey(key));
}
catch
{
return null;
}
}
}
```

У лістингу 4 представлена реалізація інтерфейсу ICredentialsManager для управління обліковими записами користувача та кукі в Xamarin.Android. Давайте розглянемо його детально.

Статичне поле Password містить константний рядок, який виглядає як конкатенація різних рядків. Це використовується доступу до даних за статчи-

ним паролем. Зчитує дані про логін користувача з облікового запису, використовуючи метод GetAccount().Зчитує дані про куки користувача з облікового запису, використовуючи метод GetAccount().

Метод SaveCredentials зберігає облікові дані користувача та куки. Видаляє старий обліковий запис, якщо він вже існує, і створює новий обліковий запис для користувача.

Метод DeleteCookies видаляє кукі користувача. Оновлює обліковий запис, замінюючи його куки на null.

Знаходить обліковий запис за вказаним ключем (Guest або General) серед всіх збережених облікових записів.

Цей код дозволяє керувати обліковими записами користувача та гостя, а також їхніми куками, зберігаючи їх у безпечному сховищі. Він використовує AccountStore для роботи із збереженням та отриманням облікових записів.

Ця структура дозволить зберігати та отримувати конфіденційні дані, такі як логін та cookies, в безпечний спосіб на обох платформах Xamarin.Android та Xamarin.iOS. Приклад реалізації для Android проекту наведено у лістингу 4.

Лістинг 4 — Вміст класу `CredentialsManager` у проекті Android

```
public class CredentialsManager : ICredentialsManager
{
string ICredentialsManager.Login
{
get
{
var account = this.GetAccount().Result;
return account?.Username;
}
}

CookieContainer ICredentialsManager.Cookies
{
```

```
    get
    {
    var account = this.GetAccount().Result;
    return account?.Cookies;
    }
    }

    CookieContainer ICredentialsManager.GuestCookies
    {
    get
    {
    var account = this.GetAccount(true).Result;
    return account?.Cookies;
    }
    }

    async Task ICredentialsManager.SaveCredentials(string
login, CookieContainer cookies)
    {
    if (!string.IsNullOrWhiteSpace(login))
    {
    var oldAccount = await this.GetAccount();
    if (oldAccount != null)
    {
    await
AccountStore.Create(Android.App.Application.Context,
Password).DeleteAsync(oldAccount, App.AppName);
    }

    Account account = new Account(login, new
Dictionary<string, string>() { { "General", "true" } },
cookies);
```

```
    await
AccountStore.Create(Android.App.Application.Context,
Password).SaveAsync(account, App.AppName);
    }
    }
    async                                                Task
ICredentialsManager.SaveGuestCredentials(CookieContainer
cookies)
    {
    var oldAccount = await this.GetAccount(true);
    if (oldAccount != null)
    {
    await
AccountStore.Create(Android.App.Application.Context,
Password).DeleteAsync(oldAccount, App.AppName);
    }

    Account   account   =   new   Account("Guest",   new
Dictionary<string,  string>()  {  {  "Guest",  "true"  }  },
cookies);
    await
AccountStore.Create(Android.App.Application.Context,
Password).SaveAsync(account, App.AppName);
    }

    async Task ICredentialsManager.DeleteCookies()
    {
    var account = await this.GetAccount();
    if (account != null)
    {
    await
AccountStore.Create(Android.App.Application.Context,
Password).DeleteAsync(account, App.AppName);
```

```
    Account updatedAccount = new Account(account.Username,
account.Properties, null);
    await
AccountStore.Create(Android.App.Application.Context,
Password).SaveAsync(updatedAccount, App.AppName);
    }
    }

    async Task ICredentialsManager.DeleteGuestCookies()
    {
    var account = await this.GetAccount(true);
    if (account != null)
    {
    await
AccountStore.Create(Android.App.Application.Context,
Password).DeleteAsync(account, App.AppName);
    Account updatedAccount = new Account(account.Username,
account.Properties, null);
    await
AccountStore.Create(Android.App.Application.Context,
Password).SaveAsync(updatedAccount, App.AppName);
    }
    }

    private async Task<Account> GetAccount(bool isGuest =
false)
    {
    var accounts = await
AccountStore.Create(Android.App.Application.Context,
Password).FindAccountsForServiceAsync(App.AppName);
    try
    {
    string key = isGuest ? "Guest" : "General";
```

```
    return accounts?.Find(account =>
account.Properties.ContainsKey(key));
    }
    catch
    {
    return null;
    }
    }
    }
```

Наведена реалізація надає змогу користувачу розпочинати гостьові сесії та звичанйні. На малюнку зображені приклади роботи застосунку у користувацькому режимі та в гостьовому.

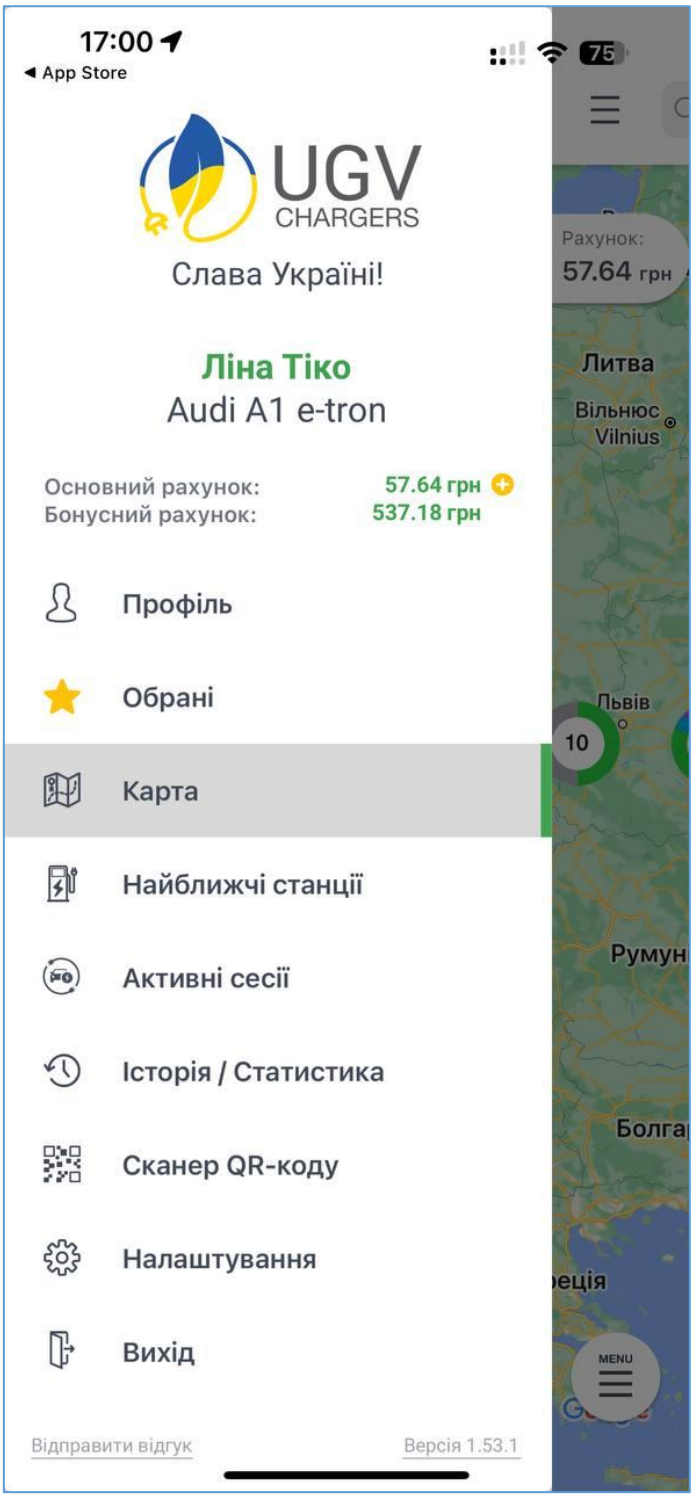

Рисунок 3.3 — Відображення користувацького акаунту

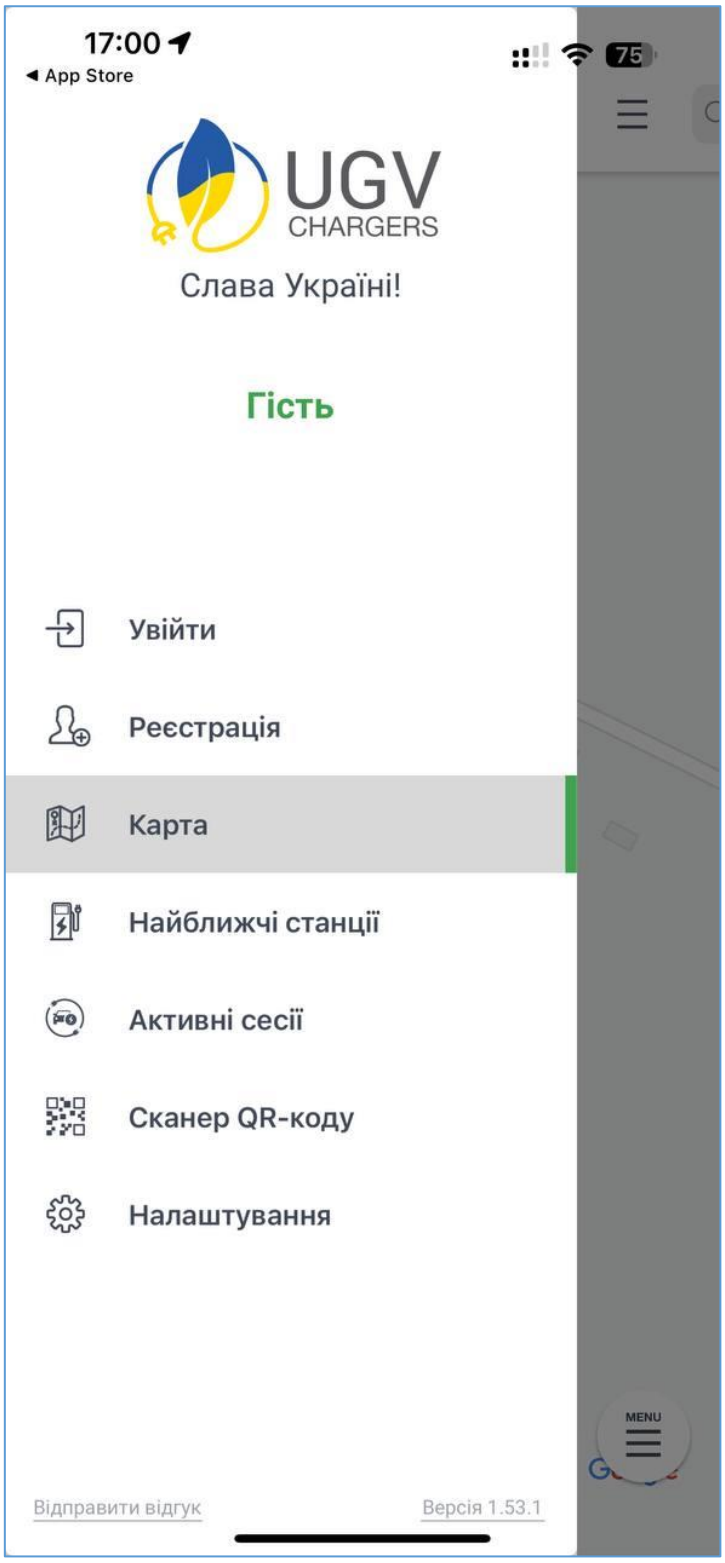

Рисунок 3.4 — Відображення гостьового акаунту

3.1.4 Реалізація зберігання даних платіжних карток користувачів

Для реалізації механізму збереження даних карток було реалізовано інтерфейс ICardsManager, який включає в себе методи, що дозволять зберігати, видаляти та отримувати інформацію щодо наявних карток у користувача. Реалізацію інтерфейсу можна побачити у лістингу 5.

Лістинг 5 — Вміст інтерфейсу `ICardsManager`

```
public interface ICardsManager
{
Task<IReadOnlyList<Card>> GetCards(CustomerData customerData);
Task<string> GetMainCardNumber(CustomerData customerData);
Task SetMainCardNumber(CustomerData customerData, string cardNumber);
Task AddCard(CustomerData customerData, Card card);
Task RemoveCard(CustomerData customerData, Card card);
}
```

Для правильного збереження інформації про картку було створено модель Card. Реалізація моделі зображена у лістингу 6

Лістинг 6 — Вміст класу `Card`

```
public class Card
{
private string card = BankCardsResource.BankCard;
public static string MainCardNumber { get; set; } = string.Empty;
public string Number
{
get
{
```

```
return this.Value;
}

set
{
this.Value = value;
}
}

[JsonProperty("ExpMonth")]
public string ExpMonth { get; set; } = string.Empty;

[JsonProperty("ExpYear")]
public string ExpYear { get; set; } = string.Empty;
public string SecureCardNumber
{
get
{
if (this.Number == BankCardsResource.AddCard || this.Number
== BankCardsResource.PayByCard)
{
return this.Number;
}
else
{
return "xxxx-xxxx-xxxx-" + this.Number.Substring(12, 4);
}
}
}

public bool IsMain
{
get
```

```
{
return this.Number == Card.MainCardNumber;
}
}
public static string CardTypeByFirstNumber(char firstNumber,
bool unkCardBlack = true)
{
switch (firstNumber)
{
case '3':
return "american_express";
case '4':
return "visa";
case '5':
return "master_card";
case '6':
return "maestro";
default:
if (unkCardBlack)
{
return "add_card_b";
}
else
{
return "add_card";
}
}
}

public static bool IsStringFromDigits(string str)
{
foreach (char symbol in str)
{
```

```
if (symbol < '0' || symbol > '9')
{
return false;
}
}

return true;
}
}
```

За аналогією було створео два класи для Android та iOS, що наслідують цей інтерфейс та реалізують функціонал для коректної роботи зберігання даних .

Приклад доданих карток у застосунок можна побачити на малюнку

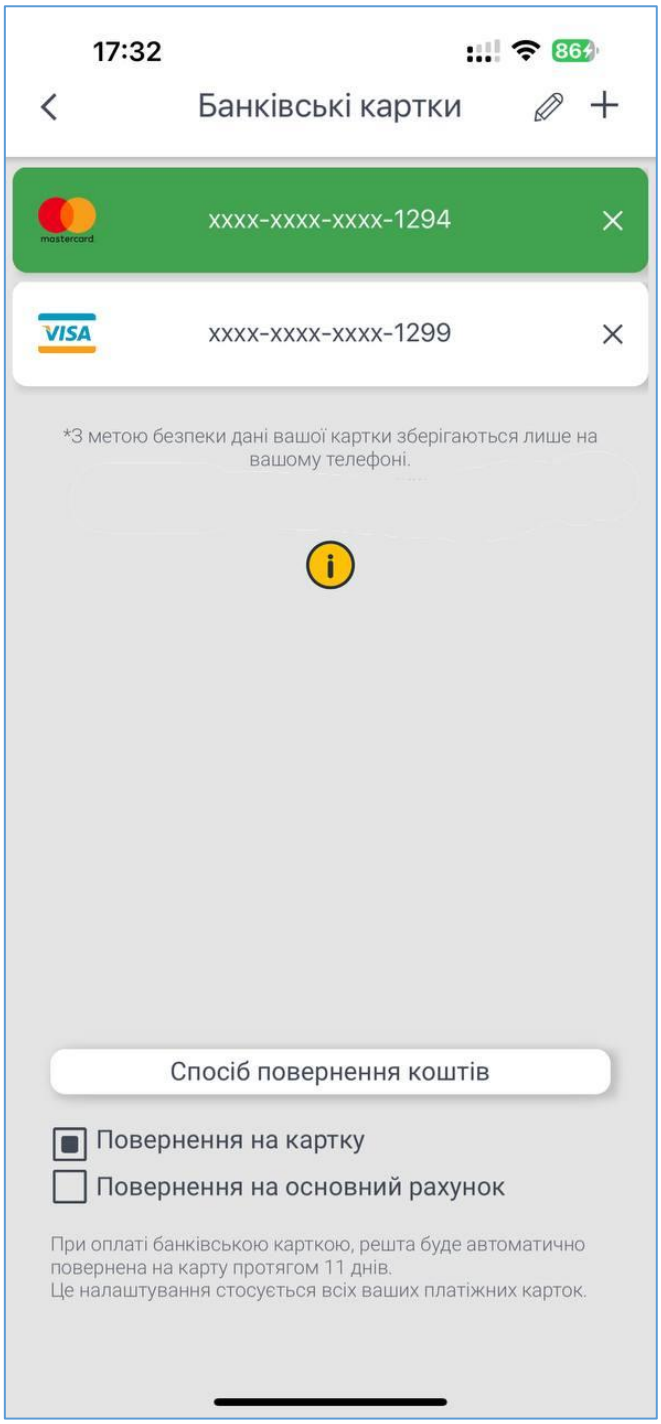

Рисунок 3.5 — Відображення карток користувача

3.2 Висновки з розділу 3

Під час розробки мобільного додатку для станцій зарядки електромобілів була виявлена необхідність в реалізації безпечного зберігання даних кори-

стувача у внутрішній пам'яті телефону. Це обумовлено двома основними вимогами: можливість почати зарядку без обов'язкового наявності акаунта (гостьові сесії), а також можливість додати платіжну картку до системи, не розголошуючи особисті дані на сервері. Для вирішення цих завдань було запропоновано використовувати бібліотеку Xamarin.Auth.

Для реалізації безпечного зберігання даних був створений інтерфейс ICredentialsManager, який був імплементований в обох платформах. У проекті Xamarin.iOS був реалізований клас CredentialsManager, використовуючи механізми безпечного сховища, такі як KeyChain, для збереження логіну та cookies користувача. Аналогічно, в Xamarin.Android використовувався Shared Preferences для зберігання цих даних.

Загальна структура класу CredentialsManager дозволяє ефективно керувати обліковими записами користувача та гостя, забезпечуючи безпечне зберігання та отримання конфіденційних даних на обох платформах. Особливості реалізації на кожній платформі відрізняються через особливості їхніх механізмів зберігання.

На Android, Shared Preferences використовується для простого зберігання невеликих обсягів даних, таких як логін та невелика кількість cookies. Важливо враховувати контекст Android та коректно взаємодіяти з SharedPreferences.

У випадку iOS, для зберігання конфіденційних даних, таких як паролі та cookies, використовується KeyChain. KeyChain забезпечує безпечне та зашифроване зберігання даних, а його взаємодія з Secure Enclave забезпечує високий рівень безпеки.

Важливо відзначити, що обидві платформи дозволяють зберігати дані в файловій системі, але на iOS можна використовувати більш безпечні методи завдяки KeyChain. Такий підхід дозволяє ефективно реалізувати функціонал безпечного зберігання даних для мобільного додатку, що взаємодіє зі станціями зарядки електромобілів.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ КОМП'ЮТЕРНОЇ СИСТЕМИ СТАНЦІЙ ЗАРЯДКИ ЕЛЕКТРОМОБІЛІВ

4.1 Результати створення внутрішнього сховища для платформ Android та iOS

На кожній платформі (Android і iOS) реалізація буде трошки відрізнятися через особливості кожної з них, але загальна структура дозволяє забезпечити збереження та отримання конфіденційних даних в безпечний спосіб на обох платформах.

Під час реалізація в Xamarin.Android використовується SharedPreferences для зберігання даних в парі "ключ-значення" на Android. Цей механізм призначений для простого збереження невеликого обсягу даних, таких як логін та невеликі кількості cookies.

Для зберігання великих обсягів даних або для більш продуктивного управління файлами може використовуватися файлова система.

Важливо враховувати контекст Android при взаємодії з різними компонентами системи, такими як SharedPreferences. Для шифрування та розшифрування даних можна використовувати власні методи або стандартні засоби криптографії.

На iOS для зберігання конфіденційних даних, таких як паролі або cookies, зазвичай використовується KeyChain. KeyChain надає безпечний і зашифрований спосіб зберігання таких даних. З KeyChain можна взаємодіяти через Secure Enclave, що додає додатковий рівень безпеки.

Для зберігання інших типів даних можна використовувати файлову систему iOS. В iOS важливо враховувати особливості обробки даних, щоб вони відповідали вимогам платформи.

iOS має вищий рівень безпеки завдяки KeyChain та можливості використання Secure Enclave.Android використовує SharedPreferences, яке менше безпечне, але може бути використано для простих випадків.На Android важливо

взаємодіяти з контекстом та коректно використовувати SharedPreferences. На iOS важливо використовувати KeyChain та враховувати особливості інтеграції зі системою. Обидві платформи дозволяють зберігати дані в файловій системі, але на iOS можна використовувати більш безпечні методи через KeyChain.

Під час реалізації механізму зберігання було створено порівняльну таблицю з характеристиками для двох платформ.

Таблиця 3 — Порівняльна характеристика зберігання даних на різних платформах

Характеристика	**Android**	**iOS**
Засіб Зберігання Даних	SharedPreferences, файлова система	KeyChain, файлова система
Безпека Даних	Забезпечується через SharedPreferences, можливо, використовуючи шифрування.	Використання KeyChain, забезпечення Secure Enclave для додаткового рівня безпеки.
Інтеграція з Системою	Важливо взаємодіяти з контекстом Android та коректно використовувати SharedPreferences.	Важливо використовувати KeyChain та враховувати особливості інтеграції з iOS.
Робота з Файлами	Можливо використовувати файлову систему Android.	Використання файлової системи iOS.
Криптографія	Можна використовувати власні методи або стандартні API для криптографії.	Важливо використовувати інструменти, що надає сама платформа iOS.

Обираючи між цими платформами для зберігання конфіденційних даних, важливо враховувати особливості інтеграції та вимоги до безпеки в вашому додатку.

Таким чином вдалося реалізувати можливість зберігання гостьових сесій на сервері та забезпечити підтримку оновлення даних для користувачів, що не мають зареєстрованого акаунту в додатку.

Загалом, реалізація механізму зберігання даних за допомогою бібліотеки Xamarin.Auth успішно враховує вимоги до безпеки та приватності користувачів на обох мобільних платформах.

В ході вивчення та реалізації можливості зберігання даних у внутрішній пам'яті мобільного додатку, виявилася нагальна потреба в створенні гостьових сесій та безпечного управління конфіденційними даними користувачів. Першочерговими завданнями були забезпечення можливості починати користуватися додатком без автентифікації, а також забезпечення приватного та безпечного додавання платіжних карток. Саме тому вирішено було обрати для реалізації механізму зберігання даних бібліотеку Xamarin.Auth, яка дозволяє ефективно управляти обліковими записами та кукі на обох платформах - Xamarin.iOS та Xamarin.Android.

Результатом впровадження було створення інтерфейсу ICredentialsManager, який спрощує роботу з управлінням конфіденційними даними та дозволяє використовувати єдиний функціонал для обох платформ. Клас CredentialsManager на платформі Xamarin.iOS реалізує методи для безпечного зберігання та отримання логіну та кукі з використанням KeyChain. Це забезпечує високий рівень безпеки завдяки зашифрованому зберіганню в безпечному сховищі та можливості використання Secure Enclave.

Для платформи Xamarin.Android обрано використання Shared Preferences для зберігання облікових даних та кукі. Цей механізм призначений для простого зберігання невеликого обсягу даних, але дозволяє ефективно вирішити задачу зберігання для даного контексту, де обсяг інформації є обмеженим.

Важливим етапом було створення порівняльної таблиці характеристик для двох платформ. Ця таблиця надає зрозумілу картину про те, як обрані механізми працюють на кожній з платформ, дозволяючи визначити відмінності та особливості кожного підходу.

У підсумку, вдалося ефективно забезпечити можливість зберігання гостьових сесій на сервері та надати підтримку оновлення даних для користувачів, які не мають зареєстрованого акаунту в додатку. Вибір між різними механізмами зберігання, такими як KeyChain на iOS та Shared Preferences на Android, дозволяє врахувати особливості та вимоги до безпеки на обох мобільних платформах. Цей підхід дозволяє досягти балансу між високою безпекою зберігання та ефективністю використання ресурсів на кожній з платформ.

4.2 Результати роботи розробленої комп’ютерної системи системи станцій зарядки електромобілів

Результати дослідження вказують на успішну реалізацію основного функціоналу системи зарядки для електромобілів. На етапі налаштування та розгортання додатку на різних платформах були вирішені ключові завдання, пов'язані з взаємодією з API, початком зарядної сесії, резервуванням портів, відображенням станцій на мапі, завантаженням та відображенням статистики користувача, а також іншими аспектами.

Процес розробки був відділений на етапи, розпочавши від аналізу вимог та визначення необхідностей. Обрано Xamarin.Forms як тип додатку, і встановлено необхідні пакети для роботи з мережевими запитами та іншими складовими. Реалізовано основні запити до API, інтеграцію з картами для відображення станцій, а також функції зберігання даних користувача.

Проект був підданий тестуванню, під час якого було виявлено та вирішено проблеми, що могли виникнути під час роботи додатку.

4.3 Варіанти покращення точності роботи системи

Після налаштування та розгортання додатку на різних платформах було виконано наступні кроки. Реалізовано основний функціонал системи зарядки, включаючи початок зарядної сесії, резерв портів, відображення станцій на мапі, завантаження та відображення статистики користувача, різноманітні методи оплати та можливість зберігання даних користувача в додатку.

В рамках проекту почали з проектування та аналізу вимог, визначаючи функціональні та нефункціональні вимоги для мобільного додатка зарядних станцій електромобілів. Обрано тип додатку (Xamarin.Forms) та встановлено необхідні пакети NuGet для роботи з мережевими запитами.

Проведено реалізацію основних запитів до API, використовуючи бібліотеки для роботи з HTTP-запитами. Налаштовано запити для функціональностей, таких як початок зарядної сесії, резерв портів, отримання інформації про станції та інше.

Для відображення станцій на мапі використано інтеграцію з картами, Google.Maps. Також реалізовано функції для відображення статистики користувача та різноманітних методів оплати. Дані користувача збережено в додатку за допомогою вбудованих механізмів зберігання або відповідних бібліотек, таких як Xamarin.Auth Розроблений функціонал підданий тестуванню для забезпечення коректної роботи всіх його частин та вирішення виявлених проблем. Механізм очікування даних від серверу впроваджено для слідкування за змінами станцій.

Для подальшого вдосконалення додатку можна врахувати такі аспекти як поліпшення інтерфейсу користувача, оптимізація продуктивності, розширення функціоналу, підвищення безпеки, локалізація та мультиплатформенність, вдосконалення аналітики, взаємодія із зовнішніми системами, тестування та виправлення помилок, оновлення та підтримка.

4.4 Висновки з розділу 4

У висновках до даного розділу важливо відзначити, що процес розробки механізму зберігання конфіденційних даних на платформах Android і iOS був успішно реалізований. При цьому, несмотря на те, що реалізація на кожній платформі має свої особливості, загальна структура дозволяє забезпечити безпечне збереження та отримання конфіденційних даних на обох платформах.

На платформі Android використовується механізм SharedPreferences для простого зберігання обмеженого обсягу даних, таких як логін та невеликі кількості cookies. Для великих обсягів даних або продуктивного управління файлами може бути використана файлова система. При цьому, важливо враховувати контекст Android під час взаємодії з різними компонентами системи, такими як SharedPreferences. Також можна використовувати власні методи або стандартні засоби криптографії для шифрування та розшифрування даних.

У випадку iOS, для зберігання конфіденційних даних, таких як паролі або cookies, типово використовується KeyChain. KeyChain надає безпечний та зашифрований спосіб зберігання таких даних, і може взаємодіяти через Secure Enclave, що додає додатковий рівень безпеки. Для інших типів даних може використовуватися файлова система iOS. Важливо враховувати особливості обробки даних, щоб вони відповідали вимогам платформи. Зауважимо, що iOS має вищий рівень безпеки завдяки використанню KeyChain та можливості використання Secure Enclave.

Описана порівняльна таблиця характеристик для платформ Android і iOS допомагає зрозуміти відмінності та специфіку кожної з платформ у контексті збереження конфіденційних даних. Враховуючи ці особливості при розробці додатків, можна досягти ефективного та безпечного збереження конфіденційних даних користувачів на обох платформах.

ВИСНОВКИ

1. У висновках слід відзначити, що процес розробки механізму зберігання конфіденційних даних на платформах Android і iOS пройшов успішно. Незважаючи на те, що реалізація на кожній платформі має свої особливості, загальна структура дозволяє забезпечити безпечне збереження та отримання конфіденційних даних на обох платформах.
2. На платформі Android використовується механізм SharedPreferences для простого зберігання обмеженого обсягу даних, таких як логін та невеликі кількості cookies. Для великих обсягів даних або продуктивного управління файлами може бути використана файлова система. При цьому, важливо враховувати контекст Android під час взаємодії з різними компонентами системи, такими як SharedPreferences. Також можна використовувати власні методи або стандартні засоби криптографії для шифрування та розшифрування даних.
3. У випадку iOS, для зберігання конфіденційних даних, таких як паролі або cookies, типово використовується KeyChain. KeyChain надає безпечний та зашифрований спосіб зберігання таких даних, і може взаємодіяти через Secure Enclave, що додає додатковий рівень безпеки. Для інших типів даних може використовуватися файлова система iOS. Важливо враховувати особливості обробки даних, щоб вони відповідали вимогам платформи. Зауважимо, що iOS має вищий рівень безпеки завдяки використанню KeyChain та можливості використання Secure Enclave.
4. Описана порівняльна таблиця характеристик для платформ Android і iOS допомагає зрозуміти відмінності та специфіку кожної з платформ у контексті збереження конфіденційних даних. Враховуючи ці особливості при розробці додатків, можна досягти ефективного та безпечного збереження конфіденційних даних користувачів на обох платформах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Electric Vehicle Charging Infrastructure - A Review" (2020), С.102-107.
2. GitHub та Open Source проекти: Вивчення відкритих проектів на GitHub може допомогти вам зрозуміти практичне застосування Xamarin у реальних проектах.
3. Статті та ресурси від Microsoft: Відвідайте ресурси Microsoft для розробників, такі як Microsoft Learn та Microsoft Developer Blog, для отримання додаткової інформації.
4. Дослідницькі роботи та тези в галузі мобільної розробки та електромобільності.
5. "Xamarin Mobile Application Development" by Dan Hermes, с.25-27.
6. "Xamarin.Forms: Cross-platform UI Toolkit" - офіційна книга документації Xamarin.Forms, с.47-52.
7. "Building Cross-Platform iOS/Android Apps with Xamarin, Visual Studio, and C# - YouTube Tutorial" by Xamarin University, с.102-110.
8. Stack Overflow: Xamarin Tag on Stack Overflow, с.145-152.
9. "Xamarin.Forms: Official YouTube Channel" - Xamarin YouTube Channel.
10. "Mobile App Development with Xamarin.Forms" - Coursera Course by Microsoft, с 123-132.
11. "Xamarin Development for the Mac".
12. "Exploring Xamarin.Forms" - Pluralsight Course by Gill Cleeren.
13. "Xamarin.Forms Kickstarter 2.0" by Dino Esposito, с.256.
14. "Introduction to Xamarin.Android" - Xamarin University Course on YouTube, с.34-45.
15. "Xamarin.Forms and the MVVM Pattern" - Blog Post by James Montemagno, с.39-42.
16. "Xamarin.Forms: Working with Maps" - Blog Post by Charlin Agramonte.
17. "Getting Started with Xamarin.Forms" - Udemy Course by Mosh Hamedani.

18."Xamarin.Forms DependencyService: Getting Started" - Blog Post by Gerald Versluis, c.178.
19."Xamarin.Forms Official Samples on GitHub" - Xamarin.Forms Samples.
20."Xamarin.Forms: Official API Documentation".
21."Xamarin.Essentials: Cross-platform APIs for Mobile Apps".
22."Xamarin.Forms: Tips and Tricks" - Blog Post Series by David Ortinau.
23."Xamarin.Forms: Styling a UI with XAML".
24."Xamarin.Forms: Building Cross-platform Mobile Apps".
25."Xamarin.Forms: Official Blog".
26."Xamarin.Forms: Best Practices" .
27."Xamarin.Forms: Introduction to Shell" - Microsoft Learn Module: Microsoft Learn Module: Офіційний модуль Microsoft Learn, який допоможе вам зрозуміти та використовувати Xamarin.Forms Shell.
28."Xamarin.Forms: Working with ListView" - Blog Post by Leomaris Reyes: Blog Post.
29."Xamarin.Forms: Behaviors" - Blog Post by Charlin Agramonte:.
30."Xamarin.Forms: Working with Images" - Blog Post by Gerald Versluis:.
31."Xamarin.Forms: Accessing Native Features with DependencyService".
32."Xamarin.Forms: Data Binding" - Tutorial by GeeksforGeeks.
33."Xamarin.Forms: Working with Firebase Realtime Database" - Blog Post by Adam Pedley.
34."Xamarin.Forms: Custom Renderers" - Blog Post Series by Xamarin University:
35."Xamarin.Forms: Performance Tips & Tricks" - Blog Post by André Marques:

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Подопригора Аніта Анатоліївна, студентка 2 курсу, денної форми навчання,

Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz118bd-12@stu.zsea.edu.ua,

- підтверджую, що написана мною кваліфікаційна робота на тему: «**Особливості програмного забезпечення для компютерної системи станцій зарядки електромобілів»** відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений;

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою Інтернет-системи, в також архівування моєї роботи у базі даних цієї системи.

Дата _____ Підпис _____ Подопригора Аніта Анатоліївна
(студент)

Дата _____ Підпис _____Заяц Валерій Іванович
(науковий керівник)