

МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка мобільного застосунку для моніторингу відключення світла в мережі під операційну систему Android

Здобувача вищої освіти другого (магістерського) рівня,
II курсу, групи КМІ-23-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»



(підпис)

Мелешка Дмитра
Олександровича
(прізвище, ім'я, по батькові)

Науковий керівник



* (підпис)

Грищенко Світлана
Миколаївна, к.пед.н.,
ст. досл.
(прізвище, ім'я, по батькові,
науковий ступінь, вчене звання)

Керівник освітньо-
професійної програми



(підпис)

Горбовий Артур Юліанович,
д.т.н, професор
(прізвище, ім'я, по батькові, науковий
ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «10» січня 2025 №11

Завідувач кафедри



(підпис)

Філоненко Михайло
Миколайович, к.ф.-м.н.,
доцент
(прізвище, ім'я, по батькові,
науковий ступінь, вчене звання)

Ірпінь – 2025

Реєстрація № 2

27.01.2025

(дата)



(підпис відповідальної особи)

Результати перевірки

до захисту

(до захисту, на доопрацювання)

27.01.2025

(дата)



(підпис)

Результат захисту

4 (добре)

(оцінка)

04.02.2025

(дата)



(підпис)

Голова комісії:



(підпис)

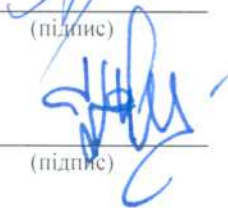
В.М. Франчук

Члени комісії:



(підпис)

А.Ю. Горбовий



(підпис)

М.М. Філоненко



(підпис)

І.Д. Погореловська

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	25		<div style="text-align: right;">04.02.2025</div> <div style="display: flex; justify-content: space-between;"> <div> Франчук В.М. Горбовий А.Ю. Філоненко М.М. Погореловська І.Д. Суховецька Л.П. </div> <div>      </div> </div>
2. Дотримання вимог щодо оформлення	від 0 до 10	8		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	42		

Оцінювання кваліфікаційної роботи науковим керівником - 43 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
85	B	4 (добре)	04.02.2025	<div style="display: flex; justify-content: space-between;"> <div> Франчук В.М. Горбовий А.Ю. Філоненко М.М. Погореловська І.Д. Суховецька Л.П. </div> <div>      </div> </div>

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую

Завідувач кафедри Філоненко М.М.

«23» грудня 2023



**Завдання на кваліфікаційну роботу
здобувачу вищої освіти**

Мелешка Дмитра Олександровича
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Розробка мобільного застосунку для моніторингу відключення світла в мережі під операційну систему Android

Затверджена наказом ректора від «20» грудня 2023 № 1701

2. Строк здачі закінченої роботи «07» січня 2025
3. Мета роботи Розробка мобільного додатку для моніторингу відключення електроенергії в мережі під операційною системою Android.

Об'єкт дослідження: Мобільний додаток для моніторингу відключення електроенергії.

Предмет дослідження Методи та інструменти розробки мобільного застосунку для моніторингу відключення електроенергії під операційною системою Android.

Орієнтовна база дослідження Підприємство ТОВ «Комтрейд Сервіс».

4. Вихідні дані до роботи методи та підходи до розробки та створений мобільний додаток
5. Перелік графічного матеріалу: 12 знімків екранних вікон
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ТА ІСНУЮЧИХ МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1. Аналіз літературних джерел з проблеми моніторингу відключення світла

1.2. Огляд існуючих мобільних застосунків для моніторингу відключення світла.

РОЗДІЛ 2. ТЕХНОЛОГІЯ РОЗРОБКИ ДОДАТКУ

2.1. Аналіз технологій для розробки додатку та створення архітектури

2.2. Програмна реалізація додатку

РОЗДІЛ 3. ОЦІНКА ЕФЕКТИВНОСТІ МОБІЛЬНОГО ЗАСТОСУНКУ

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2023	15.12.2023
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	27.05.2024	27.05.2024
3	II розділ (аналіз експериментальної частини про методики досліджень)	02.09.2024	04.09.2024
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	18.11.2024	18.11.2024
5	Висновки	10.12.2024	12.12.2024
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	23.12.2024	23.12.2024
7	Отримання відгуку керівника	28.01.2025	28.01.2025
8	Отримання зовнішньої рецензії	31.01.2025	31.01.2025

8. Дата видачі завдання «23» грудня 2023

9. Керівник кваліфікаційної роботи

10. Керівник освітньо-професійної програми

11. Завдання до виконання отримав здобувач вищої освіти


(підпис)

Грищенко С.М.
(прізвище, ініціали)


(підпис)

Горбовий А.Ю.
(прізвище, ініціали)


(підпис)

-
Мелешко Д. О.
(прізвище, ініціали)

АНОТАЦІЯ

Розробка мобільного додатку для моніторингу відключення світла в мережі під ОС Андроїд

Мета роботи полягає в розробці мобільного застосунку для моніторингу відключення електроенергії в мережі під операційною системою Android.

Завданнями роботи є:

провести аналіз наявних мобільних додатку для моніторингу відключення електроенергії; провести аналіз інструментів розробки для створення Android-додатків; дослідити технологічні аспекти розробки мобільного додатка під ОС Android; розробити прототип мобільного додатку для моніторингу відключення світла; провести тестування та впровадження додатку в реальних умовах.

Предметом дослідження є методи та інструменти розробки мобільного застосунку для моніторингу відключення електроенергії під операційною системою Android.

Об'єктом дослідження є мобільний додаток для моніторингу відключення електроенергії.

Базою дослідження виступає підприємство ТОВ «Комтрейд Сервіс».

За результатами дослідження сформульовані методика та етапи реалізації мобільного застосунку.

Одержані результати можуть бути використані при забезпеченню користувачів швидким доступом до інформації про планові та аварійні відключення електроенергії в їхньому регіоні, що дозволить їм краще планувати свої повсякденні дії.

Кваліфікаційна робота містить 54 сторінки, 0 таблиць, 12 рисунків, список використаних джерел 40 найменувань, 1 додаток.

Ключові слова: відключення світла, Android, Kotlin, операційна система, моніторинг, нативна розробка, Rest Api.

Рік виконання кваліфікаційної роботи 2025

Рік захисту роботи 2025

ABSTRACT

Development of a Mobile Application for Power Outage Monitoring in the Network on the Android OS

The purpose of this work is to develop a mobile application for monitoring power outages in the network under the Android operating system.

The objectives of the study are as follows:

1. Conduct an analysis of existing mobile applications for monitoring power outages.
2. Analyze development tools for creating Android applications.
3. Investigate the technological aspects of developing a mobile application for the Android OS.
4. Develop a prototype of a mobile application for monitoring power outages.
5. Test and implement the application in real-world conditions.

The subject of the study includes methods and tools for developing a mobile application for monitoring power outages under the Android operating system.

The object of the study is a mobile application for monitoring power outages.

The research base is the company LLC "Komtrade Service."

As a result of the study, a methodology and implementation stages for the mobile application were formulated.

The obtained results can be utilized to provide users with quick access to information about scheduled and emergency power outages in their region, enabling them to better plan their daily activities.

The qualification work comprises 54 pages, 0 tables, 12 figures, a list of 40 references, and 1 appendix.

Keywords: *power outage, Android, Kotlin, operating system, monitoring, native development, REST API.*

Year of work completion: 2025

Year of work defense: 2025

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ТА ІСНУЮЧИХ МОБІЛЬНИХ ЗАСТОСУНКІВ.....	10
1.1. Аналіз літературних джерел з проблеми моніторингу відключення світла.....	10
1.2. Огляд існуючих мобільних застосунків для моніторингу відключення світла.....	13
РОЗДІЛ 2. ТЕХНОЛОГІЯ РОЗРОБКИ ДОДАТКУ	23
2.1. Аналіз технологій для розробки додатку та створення архітектури	23
2.2. Програмна реалізація додатку	32
РОЗДІЛ 3. ОЦІНКА ЕФЕКТИВНОСТІ МОБІЛЬНОГО ЗАСТОСУНКУ	42
ВИСНОВКИ.....	47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	49
ДОДАТКИ.....	54

ВСТУП

У сучасних умовах, коли електроенергія стала невід’ємною частиною життя людей, проблема відключення світла набуває все більшої актуальності. Нестабільність енергопостачання може призвести до значних незручностей та збитків як для побутових споживачів, так і для підприємств.

Зокрема, сучасні конфлікти та військові дії в різних частинах світу підкреслюють важливість енергетичної безпеки. За останні роки вороги почали активно обстрілювати енергетичну інфраструктуру, включаючи електростанції та електричні мережі. Наприклад, в Україні стався блекаут, коли ворожі дії спричинили відключення електроенергії в значній частині країни.

Такі ситуації підкреслюють важливість розробки та впровадження ефективних інструментів для моніторингу відключень світла. Мобільний застосунок, що надає користувачам зручний інтерфейс для відстеження статусу електропостачання та отримання інформації про можливі відключення, стає невід’ємним інструментом для забезпечення безпеки та комфорту в умовах непередбачуваних ситуацій.

Актуальність розробки мобільного застосунку для моніторингу відключення світла в мережі під ОС Android зумовлена сучасними викликами, пов’язаними з нестабільністю енергопостачання та необхідністю забезпечення енергетичної безпеки в умовах кризових ситуацій, таких як військові конфлікти, природні катастрофи та технічні аварії. Мобільний додаток може стати важливим інструментом для споживачів електроенергії. Його функціонал включатиме можливість отримання актуальної інформації про стан електропостачання, сповіщення про планові та аварійні відключення, а також рекомендації щодо дій під час кризових ситуацій. Це дозволить мінімізувати ризики та незручності для користувачів, надаючи їм своєчасну та достовірну інформацію.

Різнманітні підходи до створення мобільних додатків для моніторингу відключень електроенергії та аналізу їх ефективності відображено у працях

зарубіжних авторів, таких як J. Nielsen, S. McConnell, B. Boehm, які акцентують увагу на важливості інтуїтивного дизайну, оптимізації продуктивності та забезпечення безпеки користувацьких даних. Їхні роботи слугують важливим теоретичним підґрунтям для розробки сучасних рішень, здатних адаптуватися до змінних умов і вимог користувачів.

Мета: Розробка мобільного застосунку для моніторингу відключення електроенергії в мережі під операційною системою Android.

Завданнями дослідження є:

1. Провести аналіз наявних мобільних застосунків для моніторингу відключення електроенергії.
2. Провести аналіз інструментів розробки для створення Android-додатків.
3. Дослідити технологічні аспекти розробки мобільного застосунку під ОС Android.
4. Розробити прототип мобільного застосунку для моніторингу відключення світла.
5. Провести тестування та впровадження застосунку в реальних умовах.

Об'єкт дослідження: Мобільний застосунок для моніторингу відключення електроенергії.

Предмет дослідження: Методи та інструменти розробки мобільного застосунку для моніторингу відключення електроенергії під операційною системою Android.

База дослідження: Підприємство ТОВ «Комтрейд Сервіс».

Методи дослідження: Для розробки мобільного застосунку використати методи, як: аналіз предметної області, що включає дослідження існуючих рішень для моніторингу відключення електроенергії, їх функціональності та відгуків користувачів; системний аналіз інструментів та платформ для розробки Android-додатків; створення макетів інтерфейсу та функціональних модулів додатка, та тестування додатку.

Наукова-практична важливість отриманих результатів: Мобільний додаток забезпечить користувачів швидким доступом до інформації про планові та аварійні відключення електроенергії в їхньому регіоні, що дозволить їм краще планувати свої повсякденні дії. Також сприятиме підвищенню обізнаності громадян про поточний стан електропостачання, мінімізуватиме незручності, пов'язані з відключеннями, та допоможе більш ефективно управляти енергоресурсами.

Для досягнення поставленої мети було використано комплексний підхід, що включає різноманітні методи дослідження. Зокрема, проведено аналіз існуючих наукових підходів до проблеми моніторингу відключень електроенергії, що дозволило визначити рівень її розробленості та актуальність. Використано структурно-функціональний метод для вивчення моделей і технічних рішень, які застосовуються в розробці мобільних додатків. Метод порівняння дав змогу оцінити якісні характеристики функціоналу додатків, що спрямовані на забезпечення зручності та ефективності для користувачів. Узагальнення результатів дослідження допомогло сформуванню цілісного уявлення про стан і перспективи впровадження сучасних технологій для моніторингу енергопостачання.

Практичне значення одержаних результатів. Запропоновано використовувати мобільний застосунок для моніторингу відключення світла для звичайних користувачів та компаній.

Апробація результатів роботи. Матеріали висвітлено в збірнику тез I Всеукраїнської конференції молодих вчених «Математичні методи та цифрові технології в економіці» (Ірпінь, 15 березня 2024 р.). Ірпінь, ДПУ, 2024.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ТА ІСНУЮЧИХ МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1. Аналіз літературних джерел з проблеми моніторингу відключення світла.

Тема енергетики в умовах відключень набула значної важливості для українського суспільства, викликаючи широку увагу та обговорення. Липневий моніторинг Інституту масової інформації виявив, що серед популярних українських онлайн-медіа більшість публікацій про стан енергосистеми мають збалансовану та нейтральну тональність(36,7%). Негативні матеріали(30,3%) лише трохи поступаються за кількістю позитивним та конструктивним(33%)[1].

Моніторинг відключення електроенергії є критично важливою складовою для забезпечення стабільного функціонування енергетичної системи, особливо в умовах нестабільних навантажень. Однак цей процес супроводжується численними проблемами, що впливають на точність, оперативність та ефективність контролю. Першою значною проблемою є нерівномірність розподілу енергетичних ресурсів між регіонами. Часто обмеження у передачі даних або технічні проблеми на рівні інфраструктури призводять до того, що оператори не можуть оперативно реагувати на відключення в реальному часі.

Також проблемою є технічні обмеження самого моніторингового обладнання. Використання застарілих або недостатньо розвинених систем унеможливорює повноцінне покриття території країни. Це створює інформаційні прогалини, через які важко передбачати тривалість і масштаб відключень, оскільки сучасні інтелектуальні системи моніторингу вимагають значних капіталовкладень для впровадження та підтримки.

Іншою важливою проблемою є відсутність можливостей моніторингу в реальному часі в багатьох системах моніторингу відключення електроенергії[3]. Можливість миттєвого отримання оновлень і сповіщень під час відключень електроенергії є важливою для ефективного реагування та відновлення. Однак багато систем не оснащені технологіями обробки даних у

реальному часі, що може сповільнювати час реагування під час критичних ситуацій. Цей недолік у можливостях моніторингу може призвести до неефективного розподілу ресурсів і тривалих відключень, що негативно вплине як на житлове, так і на комерційне середовище. Щоб підвищити ефективність систем моніторингу відключення електроенергії, життєво важливо інтегрувати передові технології, які сприяють збору та аналізу даних у реальному часі, дозволяючи швидше приймати рішення та діяти під час збоїв у електропостачанні.

Також ще одна важлива проблема пов'язана з обробкою великих обсягів даних. В умовах частих відключень інформація надходить у великих кількостях, що ускладнює її своєчасний аналіз. Алгоритми прогнозування можуть виявитися недостатньо точними, якщо дані надходять з затримками або містять похибки. Цей фактор також створює загрозу неефективного управління енергоресурсами в критичних ситуаціях. Крім того, труднощі в аналізі та інтерпретації даних становлять значну проблему для систем моніторингу відключення електроенергії[13]. Навіть якщо зібрано відповідні дані, складність інформації може ускладнити операторам отримання значущої інформації. Багато систем використовують методи, які недостатньо підтримують аналіз, необхідний для ефективного моніторингу та реагування. Це може призвести до суперечливих висновків і неефективних стратегій для управління збоями. Щоб подолати цю перешкоду, важливо впровадити надійні аналітичні інструменти, які можуть ефективно обробляти та візуалізувати дані, дозволяючи зацікавленим сторонам приймати обґрунтовані рішення на основі точної інтерпретації даних моніторингу затемнення[15]. Розширення можливостей аналізу даних зрештою сприятиме більш ефективному управлінню відключеннями електроенергії та покращить загальну надійність системи.

Ці проблеми можуть бути частково вирішені за допомогою мобільного додатку для моніторингу відключень електроенергії. Такий інструмент здатен забезпечити своєчасне збирання та аналіз даних, використовуючи сучасні

алгоритми прогнозування та обробки інформації, що дозволить значно підвищити ефективність моніторингу. Однак, навіть із такими технологічними рішеннями, варто враховувати можливі недоліки в існуючих системах моніторингу, які можуть суттєво ускладнювати роботу служб екстреного реагування в критичних ситуаціях[13].

При перебоях в електропостачанні, відсутність можливості точно відстежувати та повідомляти про масштаби знеструмлень може призвести до затримок у розгортанні аварійних служб і мобілізації необхідних ресурсів. Зокрема, Державна служба України з надзвичайних ситуацій акцентує увагу на важливості створення комплексних протоколів моніторингу для забезпечення оперативного реагування на техногенні надзвичайні ситуації[2]. Без надійної системи моніторингу, особам, що здійснюють первинне реагування, може бути складно оцінити ситуацію на місці події, що в результаті може поставити під загрозу безпеку постраждалих. Отже, подальший розвиток і вдосконалення таких систем є ключовим для підвищення ефективності заходів із реагування на надзвичайні ситуації, а мобільні додатки можуть стати важливим кроком у цьому напрямку.

Економічні наслідки неефективного моніторингу відключення електроенергії можуть бути глибокими, головним чином через незаплановані простої[3]. Компанії, які покладаються на безперервне електропостачання, зазнають значних фінансових втрат у разі несподіваних відключень. Наприклад, у виробників можуть зупинятися виробничі лінії, що призведе до затримок у виконанні замовлень і зниження прибутку. Крім того, сфери послуг, такі як ресторани та роздрібні магазини, можуть страждати від втрати можливостей продажу під час значних перерв. Кумулятивний ефект цих втрат може завдати шкоди місцевій економіці та порушити засоби до існування, що дає зрозуміти, що ефективний моніторинг відключення електроенергії має вирішальне значення для мінімізації економічних наслідків під час відключень електроенергії.

Окрім безпосереднього впливу на реагування на надзвичайні ситуації та економічну стабільність, неефективний моніторинг відключення світла може призвести до погіршення довіри населення до систем інфраструктури[1]. Коли громадяни стикаються з частими відключеннями електроенергії без чіткого зв'язку чи вирішення проблеми, їхня впевненість у надійності постачальників електроенергії та управління інфраструктурою падає. Таке руйнування довіри може призвести до суспільного розчарування та недовіри щодо здатності уряду ефективно керувати основними послугами. Як зазначалося в дискусіях навколо енергетичної галузі в Україні, прозорість і підзвітність є життєво важливими для підтримки суспільної довіри під час криз[2]. Таким чином, удосконалення практики моніторингу відключення електроенергії необхідне не лише для ефективності роботи, але й для відновлення та підтримки громадської довіри до систем критичної інфраструктури.

На основі проведеного аналізу літературних джерел було виявлено ключові напрями, досягнення та проблеми в цій галузі. У сучасних умовах надійність електропостачання є критично важливою для всіх категорій споживачів, а відключення електроенергії може мати серйозні наслідки. Тому питання моніторингу таких ситуацій викликає значний інтерес, і розробка відповідних інструментів та додатків стає все більш актуальною. Зокрема, мобільні додатки на платформі Android можуть стати дієвим інструментом для своєчасного інформування користувачів про перебої в електропостачанні, що, у свою чергу, сприятиме підвищенню прозорості та підзвітності постачальників послуг.

1.2. Огляд існуючих мобільних застосунків для моніторингу відключення світла.

Програми для звітування про перебої в режимі реального часу відіграють вирішальну роль у наданні негайної інформації про перебої в електропостачанні. Ці додатки відстежують стан електроенергії та сповіщають користувачів про будь-які зміни, гарантуючи, що вони завжди будуть

поінформовані про збої в роботі. Одним із яскравих прикладів є додаток «СвітлоБот», який розробили українські айтишники Андрій Буланов та Костянтин Медведенко[5]. Додаток пропонує постійний моніторинг електропостачання, надсилаючи сповіщення в режимі реального часу користувачам щоразу, коли відбувається зміна стану живлення[10]. Такі своєчасні сповіщення мають важливе значення для окремих осіб і підприємств, які значною мірою залежать від стабільного електропостачання, дозволяючи їм вживати необхідних запобіжних заходів і відповідно керувати своєю діяльністю.

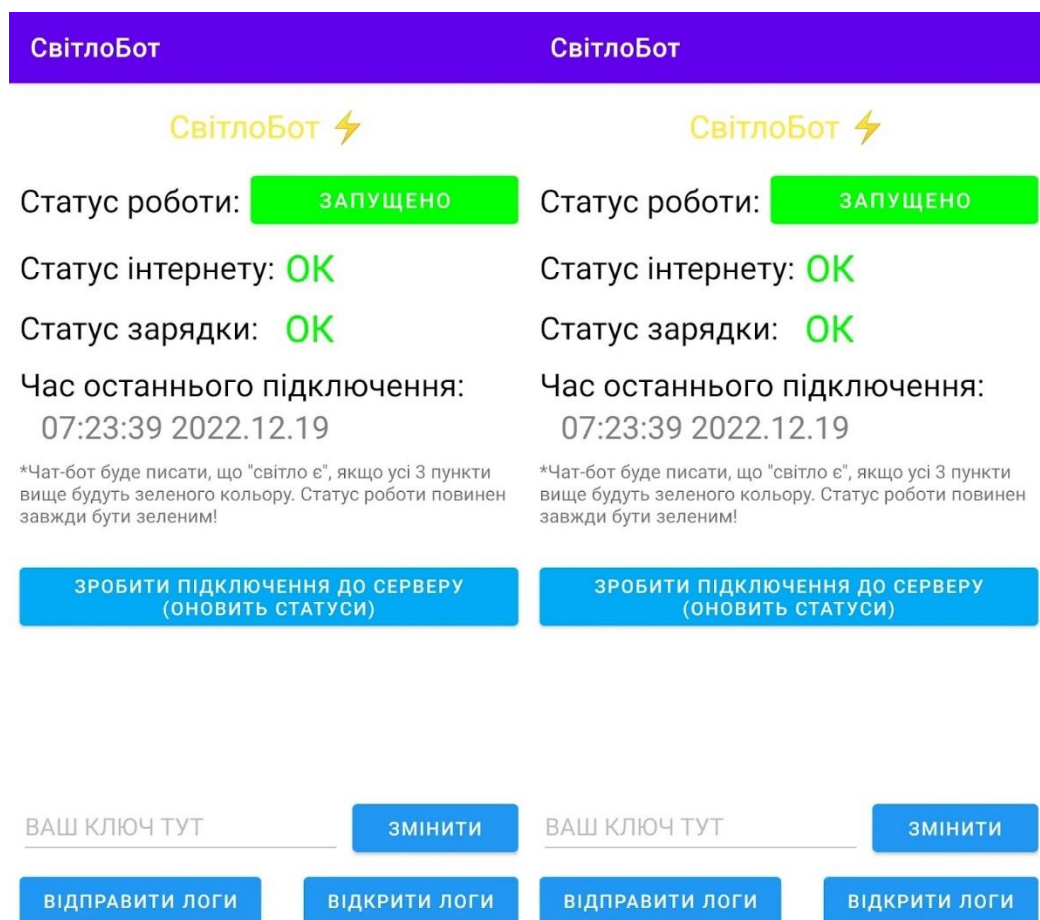


Рис. 1.1. Мобільний додаток СвітлоБот

Додаток «СвітлоБот» також оснащений функцією інтерактивної карти, яка надає користувачам можливість візуалізувати зони, де відбулися перебої в електропостачанні. Це сприяє не лише індивідуальним користувачам, але й підприємствам та організаціям у плануванні їхньої діяльності з урахуванням можливих перебоїв..

Ще однією важливою особливістю «Світлоботу» є його здатність інтегруватися з іншими системами та пристроями. Наприклад, додаток може бути підключений до систем розумного дому, що дозволяє автоматично активувати резервні джерела живлення або надсилати сповіщення на інші пристрої. Це робить «Світлобот» не лише інструментом для моніторингу, але й важливим компонентом у системі управління енергоспоживанням[7].

Програми для передбачення та прогнозування покращують здатність користувачів передбачати відключення електроенергії, надаючи інформацію про планове технічне обслуговування та надзвичайні ситуації. Наприклад, додаток «Світло» дозволяє користувачам зручно переглядати час і тривалість запланованих відключень електроенергії[28].

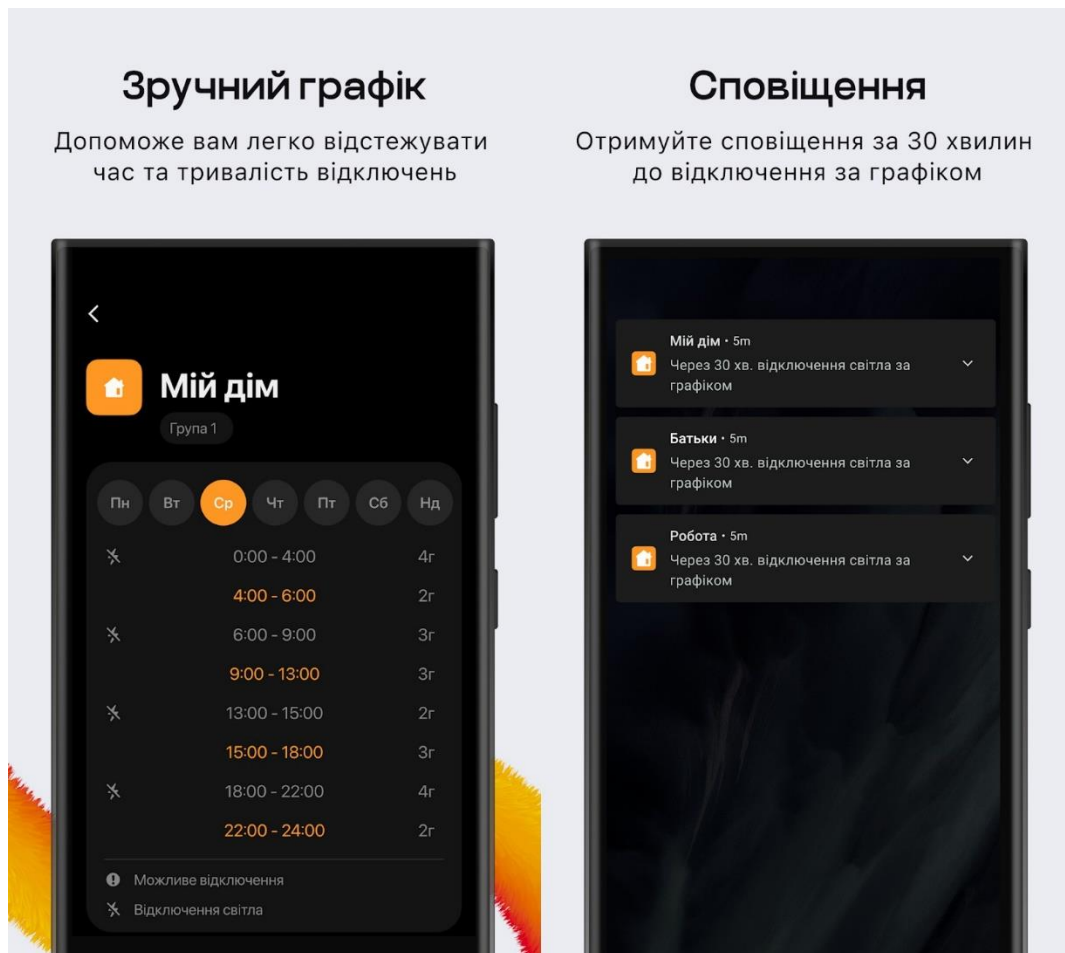


Рис. 1.2. Мобільний додаток Світло

Додаток також надає можливість встановлювати нагадування про майбутні відключення, що допомагає користувачам підготуватися до можливих

незручностей. Крім планових відключень, додаток інформує про екстрені відключення електроенергії, що дозволяє бути в курсі непередбачуваних ситуацій.

Цей додаток став популярним через часті планові відключення електроенергії, особливо під час енергетичних криз або надзвичайних ситуацій. Він допомагає користувачам ефективніше керувати своїм часом і уникати непередбачуваних ситуацій, пов'язаних із відключеннями світла.

«Світло» підтримує різні регіони України, включаючи Київ, Київську область, Львівську область, Дніпропетровську область, Одеську область, Харківську область, Миколаївську область, Чернігівську область, Тернопільську область та Вінницьку область. Дані про планові відключення отримуються з відкритих джерел обласних енергетичних провайдерів відповідного регіону.

Мобільний додаток «Мережі онлайн» від ДТЕК дозволяє користувачам отримувати вичерпну інформацію як про планові, так і про аварійні відключення, включаючи орієнтовні терміни відновлення[23].

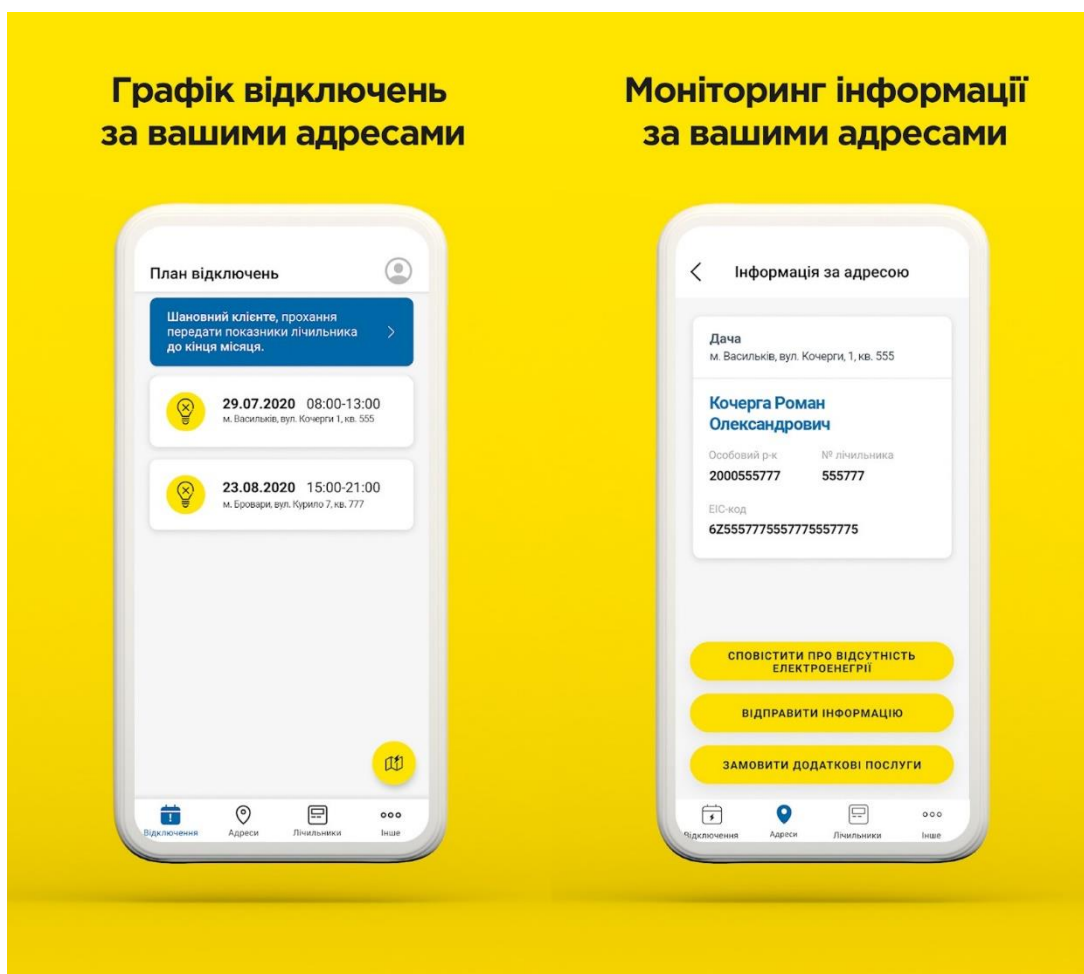


Рис. 1. 3. Мобільний додаток Мережі онлайн

Він охоплює як планові, так і аварійні відключення, допомагаючи користувачам оперативно дізнаватися про стан мереж та орієнтовний час відновлення електропостачання. У додатку можна отримати сповіщення про відключення в реальному часі, а також слідкувати за термінами виконання ремонтних робіт. Також цей додаток, можна поділити на 3 розділи:

1. Відключення. У цьому розділі представлено перелік аварійних та планових відключень електричної енергії, що дозволяє користувачам оперативно отримувати інформацію про відсутність електропостачання та орієнтовний час його відновлення. Користувачі можуть додавати кілька адрес для відстеження інформації про відключення.

2. Адреси. У цьому розділі користувачі можуть повідомляти компанію про відсутність електроенергії та замовляти додаткові послуги, такі як обслуговування приладів обліку або підключення. Важливо зазначити, що

програма здійснює пряму переадресацію з додатку на відповідну сторінку вебсайту компанії.

3. Лічильники. Цей розділ призначений для передачі показів приладів обліку. У разі, якщо користувач забув вчасно передати покази, йому буде надіслано нагадування. Для підтвердження інформації про використану електроенергію необхідно сфотографувати лічильник і надіслати фото разом із показами[23].

Ці програми використовують аналітику даних для прогнозування можливих збоїв, допомагаючи користувачам підготуватися заздалегідь і пом'якшити вплив перебоїв з електропостачанням на їхнє повсякденне життя.

Платформи оповіщення та спілкування на основі спільнот стали ефективними інструментами для сприяння з'єднанню між користувачами під час відключень електроенергії. Наприклад, додаток Yasno Mobile дозволяє мешканцям Києва отримувати сповіщення про відключення електроенергії безпосередньо від постачальника комунальних послуг, підвищуючи обізнаність громади та реагуючи на відключення[6].

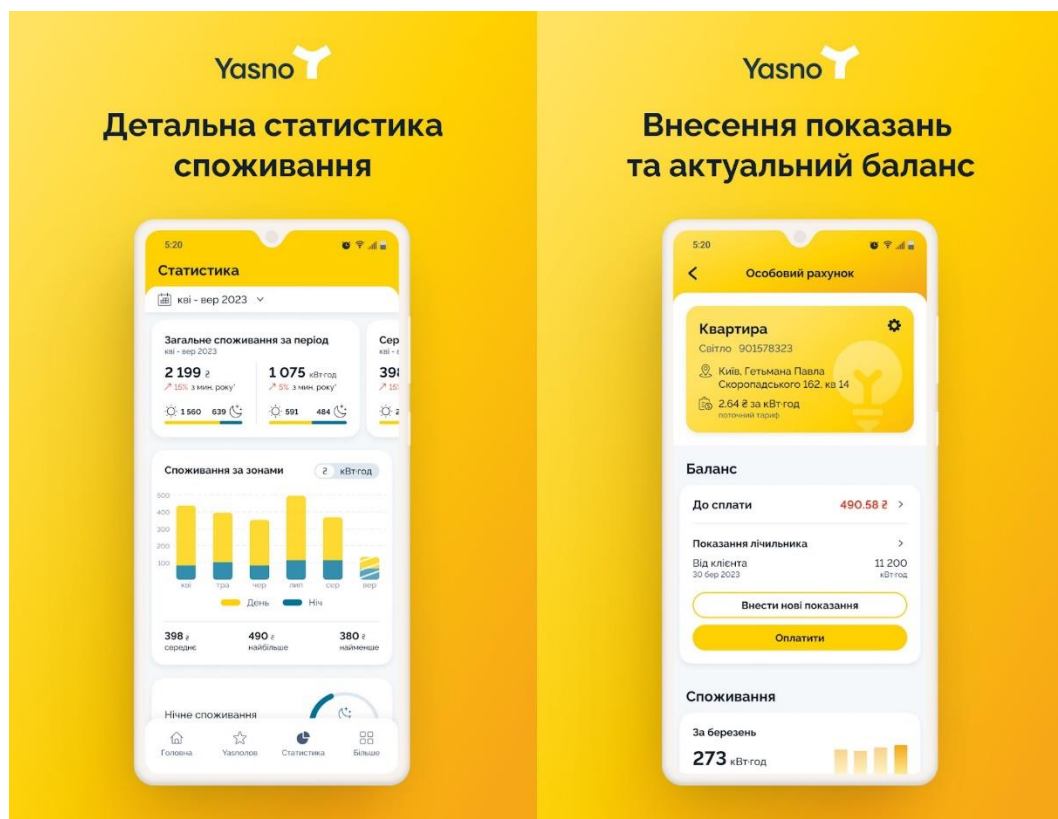


Рис. 1. 4. Мобільний додаток Yasno Mobile

Мобільний додаток забезпечує оперативний доступ до інформації про суму до сплати за електроенергію, можливість передачі показань лічильника та здійснення платежів без комісії. Користувачі можуть аналізувати споживання електроенергії або газу на основі рахунків, що сприяє оптимізації витрат. Додаток також дозволяє створювати рахунки для кількох об'єктів та здійснювати їх оплату одним натисканням.

У сучасних умовах, крім спеціалізованих мобільних додатків, зростає популярність використання платформ для обміну повідомленнями, таких як Telegram і Viber. Вони полегшують спілкування між користувачами, дозволяючи їм обмінюватися інформацією та оновленнями щодо проблем з електропостачанням у режимі реального часу. Ці сервіси не тільки інформують людей, але й зміцнюють зв'язки між громадами, допомагаючи підтримувати один одного під час відключень і ділитися критичними новинами про місцеву ситуацію.

Функціональні вимоги для таких систем і додатків відіграють ключову роль, визначаючи, які саме дії система повинна виконувати для задоволення потреб користувачів. Наприклад, функціонал додатку може включати можливість інтеграції з платформами обміну повідомленнями для автоматичного сповіщення або передачі даних про стан енергопостачання. При цьому ці вимоги повинні відповідати принципам SMART, щоб забезпечити точність, досяжність і актуальність запропонованих рішень.

Визначення функціональних та нефункціональних вимог до мобільного додатку включає кілька ключових аспектів. Функціональні вимоги визначають, що саме додаток повинен робити, наприклад, можливість користувачів повідомляти про нові акції, послуги, отримувати оновлення в режимі реального часу, переглядати прогнози щодо певних можливостей та отримувати рекомендації щодо дій для зручного користування додатку.

Функціональні вимоги для мобільного додатку моніторингу відключення включатимуть:

- Відстеження відключень електропостачання за адресою проживання. Функціональна вимога щодо відстеження відключень електропостачання за адресою проживання передбачає, що користувач повинен мати можливість ввести свою адресу в додатку. Після введення адреси додаток повинен відображати поточний статус електропостачання за цією адресою. Це включає інформацію про наявність або відсутність електропостачання, а також будь-які заплановані або поточні відключення. Інформація повинна оновлюватися в режимі реального часу або з мінімальною затримкою, щоб користувач завжди мав актуальні дані про стан електропостачання за своєю адресою[17].

- Перегляд графіків відключень за іншими адресами. Функціональна вимога щодо перегляду графіків відключень за іншими адресами передбачає, що користувач повинен мати можливість вводити будь-які інші адреси, окрім своєї основної, для отримання інформації про заплановані відключення електропостачання. Додаток повинен відображати графіки відключень для введених адрес, надаючи користувачу доступ до детальної інформації про дати та часи запланованих відключень. Інформація повинна бути представлена у зручному форматі, наприклад, у вигляді календаря або списку, з можливістю фільтрації за датою або іншими параметрами. Це дозволить користувачам легко знаходити необхідну інформацію та планувати свої дії відповідно до графіків відключень електропостачання за різними адресами.

- Сповіщення користувача про відключення електропостачання. Функціональна вимога щодо сповіщення користувача про відключення електропостачання передбачає, що додаток повинен автоматично надсилати сповіщення користувачу у разі відключення електропостачання за його адресою. Сповіщення повинні містити детальну інформацію про причину відключення, орієнтовний час відновлення електропостачання та будь-які інші важливі деталі. Користувач повинен мати можливість налаштовувати типи сповіщень, які він бажає отримувати, такі як push-сповіщення, SMS або email.

Це забезпечить своєчасне інформування користувача про будь-які зміни в стані електропостачання, дозволяючи йому вжити необхідних заходів для мінімізації незручностей.

- Сповіщення про найближче відключення згідно графіку. Функціональна вимога щодо сповіщення про найближче відключення згідно графіку передбачає, що додаток повинен надсилати користувачу попереджувальні сповіщення про заплановані відключення електропостачання. Ці сповіщення повинні надходити заздалегідь, наприклад, за 30 хвилин до запланованого відключення, щоб користувач мав достатньо часу для підготовки. Сповіщення повинні містити інформацію про точний час початку та закінчення відключення, а також будь-які інші важливі деталі, такі як причини відключення або рекомендації щодо підготовки. Користувач повинен мати можливість налаштовувати час попередження та гучність сповіщень відповідно до своїх потреб, що забезпечить максимальну зручність та ефективність використання додатку[7].

Нефункціональні вимоги стосуються якості роботи додатку, таких як швидкість завантаження, безпека даних, зручність користування, сумісність з різними операційними системами та пристроями, а також здатність додатку витримувати високі навантаження під час пікових періодів використання. Важливо також врахувати вимоги до інтерфейсу користувача, щоб забезпечити інтуїтивно зрозумілий та привабливий дизайн, який сприятиме легкому доступу до всіх функцій додатку.

Серед нефункціональних вимог для мобільного додатку моніторингу відключення світла можна виділити кілька ключових аспектів. По-перше, продуктивність додатку є критично важливою, оскільки він повинен швидко реагувати на запити користувачів і оперативно оновлювати дані в реальному часі. Це забезпечить користувачам своєчасну інформацію про відключення світла[4].

Надійність додатку також має велике значення. Він повинен бути стійким до збоїв і забезпечувати безперервну роботу навіть при високих навантаженнях.

Це означає, що додаток повинен мати механізми для відновлення після збоїв і мінімізації часу простою.

Зручність використання є ключовим фактором для успіху будь-якого мобільного застосунку. Інтерфейс повинен бути інтуїтивно зрозумілим і зручним для користувачів різного рівня технічної підготовки, що забезпечить позитивний користувацький досвід.

Енергоефективність додатку є важливою для збереження заряду батареї пристрою користувача. Додаток повинен мінімізувати споживання енергії, щоб користувачі могли використовувати його протягом тривалого часу без необхідності частого заряджання пристрою.

Функціональні вимоги є критичними для успіху будь-якого проєкту розробки програмного забезпечення. Вони забезпечують чітке розуміння того, що необхідно створити, дозволяючи команді правильно розставляти пріоритети та ефективно виконувати свої завдання.

Нефункціональні вимоги, з іншого боку, визначають внутрішні характеристики системи, такі як продуктивність, безпека, масштабованість та надійність. Вони забезпечують, що система працює належним чином під різними умовами та відповідає очікуванням користувачів щодо якості та стабільності.

РОЗДІЛ 2. ТЕХНОЛОГІЯ РОЗРОБКИ ДОДАТКУ

2.1. Аналіз технологій для розробки додатку та створення архітектури

На сьогоднішній день найбільш поширеними мобільними операційними системами є Android(близько 60%) та iOS(близько 38%). Інші 2% припадають на операційні системи, які поступово втрачають популярність, такі як Windows Phone, KaiOS та Samsung OS. У цьому матеріалі ми розглянемо розробку додатків для Android та iOS[12].

Нативна розробка передбачає створення програм для мобільних пристроїв на конкретній мові програмування для конкретної платформи. Нативні додатки відзначаються високою продуктивністю та відсутністю обмежень у розробці (Java і Kotlin для Android, Swift для iOS). Перевагами такого підходу є швидка реакція на дії користувача, прямий доступ до апаратної частини пристрою та можливість створення інтерфейсу, максимально зручного для користувачів конкретної платформи. Недоліками є висока вартість розробки та підтримки, а також тривалий час, необхідний для розробки. Створення одного і того ж додатка для різних платформ(iOS та Android) є тривалим і дорогим процесом, а їх тестування також займає значний час (наприклад, компіляція оновлень для Android займає близько 30 секунд). Тому, якщо необхідно створити простий додаток для двох платформ одночасно, часто використовують гібридний підхід до розробки[19].

Кросплатформенна розробка здійснюється за допомогою веб-технологій, таких як HTML, CSS та JavaScript, що дозволяють створювати додатки для кількох платформ одночасно. Однак, для того щоб додаток працював відповідно до вимог платформи, його потрібно “перекласти” на зрозумілу для платформи мову або додати проміжний шар-перекладач. Перевагами цього підходу є низька вартість розробки, оскільки іноді достатньо одного фахівця. Недоліками є можливі труднощі у роботі всіх функцій, затримки у реакції на дії

користувача (додаток може бути повільним), а також необхідність додаткового доопрацювання інтерфейсу.

Розробка додатків для кожної з операційних систем здійснюється з використанням відповідного середовища програмування та мов програмування. Програми для ОС Android розробляються в середовищі Android Studio мовами Kotlin або Java. Для iOS розробка ведеться в середовищі Xcode мовою Swift. Такий підхід до розробки називають “рідним” (Native), оскільки розробники використовують “рідні” середовища та мови програмування для створення додатків [20].

Для розробки мобільного додатку для моніторингу відключення світла мною вибрано саме операційну систему Android та мову програмування Kotlin, оскільки Android є найпоширенішою мобільною платформою в світі, що дозволяє охопити широку аудиторію користувачів. Kotlin, як сучасна мова програмування, офіційно підтримується Google для розробки Android-додатків, має простий і зрозумілий синтаксис, високу продуктивність і надає розширені можливості для написання чистого та безпечного коду.

Операційна система Android, спочатку розроблена для смартфонів та планшетів, а тепер і для ПК, бере свій початок у 2003 році як проект американської технологічної компанії Android Inc. з метою створення операційної системи для цифрових камер. У 2004 році проект був переорієнтований на розробку операційної системи для смартфонів. Та у 2005 році компанія Android Inc. була придбана американською компанією Google Inc. Команда Android у Google вирішила базувати свій проект на Linux, операційній системі з відкритим кодом для персональних комп’ютерів.

5 листопада 2007 року Google оголосила про створення Open Handset Alliance, консорціуму, з метою розробки та просування Android як безкоштовної операційної системи з відкритим кодом, що підтримує сторонні програми[12].

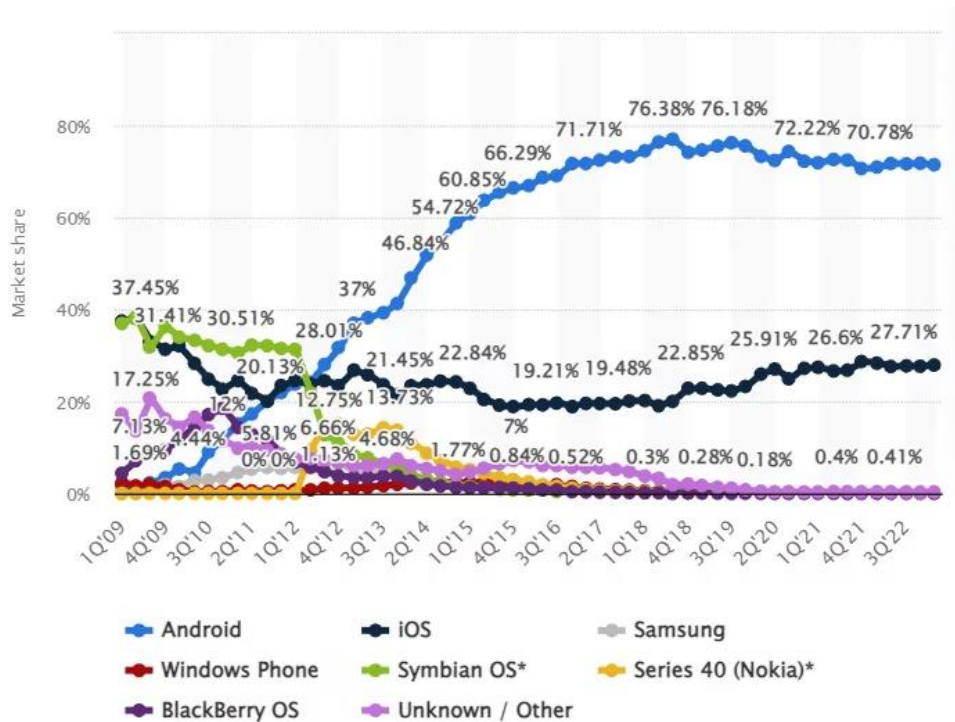


Рис. 2.1. Рейтинг операційної системи станом на 2022 р.

У 2012 році Android стала найпопулярнішою операційною системою для мобільних пристроїв, випередивши iOS від Apple, і станом на 2024 рік займає близько 68 відсотків ринку мобільних пристроїв.

Для розробки під операційною системою Android першою переважною мовою є Kotlin, що було підтверджено на конференції Google I/O у 2019 році. Kotlin — це статично типізована, сучасна, але вже зріла мова програмування з відкритим вихідним кодом загального призначення, розроблена для підвищення задоволення розробників. Вона підтримує як об'єктно-орієнтоване, так і функціональне програмування. Kotlin працює на JVM і може використовуватися скрізь, де сьогодні застосовується Java.

На сьогоднішній день понад 90% професійних розробників Android використовують Kotlin, а 95% з 100% найкращих додатків Android містять код на Kotlin[26].

Переваги Kotlin включають:

- Лаконічність: Kotlin зменшує обсяг додаткового коду, роблячи його більш компактним.

- Безпека від нульових значень: Kotlin прагне усунути `NullPointerException` (нульові посилання) з коду.
- Інтєроперабельність: Kotlin легко викликає код Java, а код на Kotlin може використовуватися в Java.
- Розумне приведення типів: Kotlin автоматично приводить незмінні значення до безпечних типів.
- Час компіляції: Kotlin забезпечує кращу продуктивність і швидший час компіляції.
- Зручні інструменти: Програми на Kotlin можна створювати за допомогою командного рядка або будь-якої Java IDE.
- Функції розширення: Kotlin підтримує функції та властивості розширення, що дозволяє розширювати функціональність класів без зміни їх коду[26].

Для створення додатку буде використовуватися API компанії ДТЕК електромережі з метою отримання даних про графіки відключень. API (Application Programming Interface) — це механізми, що дозволяють двом програмним компонентам взаємодіяти між собою, використовуючи набір визначень та протоколів. Наприклад, система метеослужби містить щоденні дані про погоду, а програма погоди на телефоні взаємодіє з цією системою через API, відображаючи щоденні оновлення погоди[9].

Принцип роботи API зазвичай виражається через зв'язок запит-відповідь між клієнтом і сервером. Клієнт — це будь-яка зовнішня програма, з якою взаємодіє користувач, тоді як сервер відповідає за серверну логіку та операції з базою даних. У цьому сценарії API функціонує як проміжний рівень між клієнтом і сервером, що дозволяє надсилати запити даних і отримувати відповіді.

Існує чотири основні способи функціонування API, які залежать від часу та причин їх створення.

- SOAP (Simple Object Access Protocol) — це простий протокол доступу до об'єктів. Клієнт і сервер обмінюються повідомленнями за допомогою XML. Цей тип API менш гнучкий і був більш популярним у минулому.

- RPC (Remote Procedure Call) API — це система віддаленого виклику процедур. Клієнт виконує функцію (або процедуру) на сервері, а сервер надсилає результат назад клієнту.

- WebSocket API — це сучасна розробка web API, яка використовує об'єкти JSON для передачі даних. WebSocket API підтримує двосторонній зв'язок між клієнтськими програмами та сервером. Сервер може надсилати повідомлення зворотного дзвінка підключеним клієнтам, що робить його ефективнішим порівняно з REST API.

- REST API є найпопулярнішими та найгнучкішими API-інтерфейсами в Інтернеті. Клієнт надсилає запити на сервер у вигляді даних, сервер використовує ці дані для виконання внутрішніх функцій і повертає результати назад клієнту. Нижче розглянемо REST API докладніше.

REST (Representational State Transfer) — це архітектурний стиль, що визначає набір функцій, таких як GET, PUT, DELETE тощо, які клієнти можуть використовувати для доступу до даних сервера. Обмін даними між клієнтами та серверами здійснюється за протоколом HTTP.

Основною особливістю REST API є те, що передача даних виконується без збереження стану. Це означає, що сервери не зберігають дані клієнта між запитами. Клієнтські запити до сервера подібні до URL-адрес, які вводяться в браузері для відвідування веб-сайту. Відповідь від сервера містить прості дані без типового графічного відображення веб-сторінки[17].

REST API має чотири основні переваги:

1. Інтеграція: API використовуються для інтеграції нових програм із існуючими програмними системами, що підвищує швидкість розробки, оскільки кожен функцію не потрібно писати з нуля. API можна використовувати для посилення наявного коду.

2. Інновації: Цілі галузі можуть змінитися з появою нової програми. Компанії повинні швидко реагувати та підтримувати швидке розгортання інноваційних послуг. Це можливо завдяки внесенню змін на рівні API без необхідності переписувати весь код.

3. Розширення: API-інтерфейси надають компаніям унікальну можливість задовольняти потреби своїх клієнтів на різних платформах. Наприклад, API для карт дозволяє інтегрувати інформацію про карти через веб-сайти, Android, iOS тощо. Будь-яка компанія може надати аналогічний доступ до своїх внутрішніх баз даних, використовуючи безкоштовні або платні API.

4. Простота обслуговування: API діє як шлюз між двома системами. Кожна система зобов'язана вносити внутрішні зміни таким чином, щоб це не вплинуло на API. Таким чином, будь-які майбутні зміни коду однією стороною не вплинуть на іншу сторону.

Для розробки архітектури мобільного додатку було використано середовище Android Studio та Lucidchart. Lucidchart, створений компанією Lucid Software, є провідним інструментом для створення діаграм, що дозволяє командам ефективно вирішувати складні проблеми, узгоджувати ідеї та планувати майбутнє. Android Studio забезпечує всі необхідні засоби для розробки та тестування додатків на платформі Android. Завдяки вбудованим інструментам, таким як Layout Editor та Design Tools, можна створювати складні інтерфейси користувача, візуалізувати архітектурні компоненти та керувати залежностями через систему збірки Gradle.

Основою архітектури програми є підхід MVVM (Model-View-ViewModel), який дозволяє чітко розмежувати логіку представлення та бізнес-логіку програми. Це забезпечує гнучкість у підтримці та розширенні функціоналу, а також сприяє повторному використанню компонентів. Для керування станом додатку використовувалися інструменти Jetpack Compose та StateFlow, що забезпечують реактивність інтерфейсу та спрощують роботу з потоками даних у режимі реального часу[35].

Додатково, використання Jetpack Compose надає розробникам можливість реалізовувати інтерфейси користувача за допомогою декларативного підходу, що суттєво скорочує кількість шаблонного коду. Завдяки цьому підходу розробники можуть сфокусуватися на логіці та поведінці додатка, а не на дрібних деталях управління станом компонентів. Це особливо важливо в умовах сучасних високонавантажених проєктів, де швидкість розробки та надійність функціоналу є критичними факторами успіху.

Інтеграція StateFlow забезпечує реактивний зв'язок між даними та інтерфейсом, мінімізуючи затримки та виключаючи можливість невідповідності між станами системи, що є критично важливим для обробки асинхронних подій у додатках із високим навантаженням. Завдяки своїй гнучкості та інтеграції зі співпрограмами, StateFlow стає оптимальним вибором для сценаріїв, що вимагають оперативного оновлення інформації, наприклад, у системах моніторингу енергопостачання в реальному часі. Водночас у розробці Android LiveData залишається поширеним інструментом, особливо в проєктах, які дотримуються архітектурних рекомендацій Android[27]. Вибір між цими підходами залежить від вимог конкретного випадку використання та особливостей архітектури додатку.

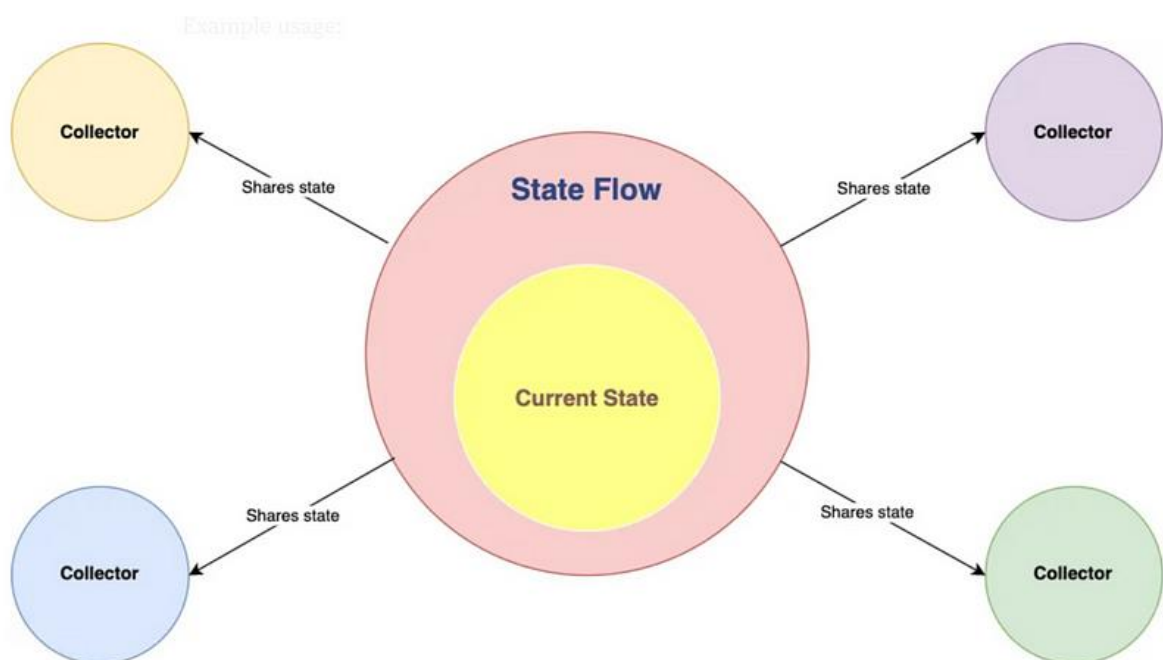


Рис 2.2. Діаграма StateFlow

Stateflow забезпечує можливість проектування та розробки систем диспетчерського керування, планування завдань, керування несправностями, протоколів зв'язку, користувацьких інтерфейсів та гібридних систем. У середовищі Android StateFlow є оптимальним для класів, що потребують підтримки спостережуваного змінного стану.

Життєвий цикл StateFlow у додатках для Android зазвичай відповідає життєвому циклу компонентів, таких як Activity або Fragment, які його містять. Основне призначення StateFlow полягає в забезпеченні реактивного потоку даних, причому його поведінка тісно пов'язана з життєвим циклом компонентів, які його спостерігають[17].

StateFlow та LiveData виконують схожі функції у розробці Android, дозволяючи спостерігати за змінами в даних і відповідно реагувати на них. Незважаючи на корисність обох, StateFlow має певні переваги, що роблять його більш ефективним та кращим вибором у певних сценаріях:

StateFlow повністю інтегрований зі співпрограмами Kotlin для обробки асинхронних операцій, що дозволяє працювати з асинхронними потоками даних у більш природний і лаконічний спосіб. StateFlow є більш потужним та гнучким API для роботи з асинхронними потоками даних порівняно з LiveData, надаючи можливість застосування різних операторів, таких як map, filter і transform, для легкої маніпуляції та перетворення потоків даних. StateFlow має вбудовану підтримку обробки зворотного тиску, що дозволяє контролювати швидкість видачі та споживання даних, чого LiveData не пропонує, роблячи її менш придатною для сценаріїв, де контроль зворотного тиску є важливим[22].

StateFlow доступний як у змінному, так і в незмінному варіантах: MutableStateFlow дозволяє оновлювати стан, тоді як незмінний StateFlow може лише читатися. StateFlow не залежить від життєвого циклу Android. У той час як LiveData розроблено з урахуванням життєвого циклу та автоматично очищається після знищення пов'язаного компонента, StateFlow не має цієї вимоги, що є перевагою у випадках спостереження за даними в компонентах, не

пов'язаних з інтерфейсом користувача, або під час виконання тривалих фонових завдань. StateFlow легко тестується за допомогою тестових утиліт співпрограм Kotlin, що спрощує написання модульних тестів для коду, який використовує StateFlow.

LiveData також піддається тестуванню, але може вимагати додаткових налаштувань і шаблонів тестування. Вибір між StateFlow та LiveData залежить від специфічних вимог проєкту та бажання використовувати Kotlin Flow і співпрограми як частину розробки. У багатьох випадках StateFlow є кращим завдяки своїй гнучкості та ефективності при роботі з асинхронними потоками даних[40].

Для оптимізації роботи з залежностями було обрано Hilt, що не лише автоматизує процес їх впровадження, але й зменшує ризики помилок у структурі додатку. Це дозволяє легко підтримувати та розширювати функціонал програми, адже модулі стають більш автономними й тестованими. Використання Hilt також сприяє створенню уніфікованої архітектури, яка забезпечує стабільність навіть при додаванні нових функцій чи інтеграції складних сервісів.

Dagger Hilt є бібліотекою для ін'єкцій залежностей у Android, побудованою на основі Dagger 2. Hilt призначений для спрощення впровадження ін'єкцій залежностей у додатках для Android шляхом скорочення шаблонного коду та забезпечення інтеграції з компонентами Android, такими як дії, фрагменти, служби та ViewModels. Використовуючи потужність Dagger, Hilt пропонує розробникам більш стислий та інтуїтивно зрозумілий API[37].

Dagger Hilt застосовує набір анотацій для визначення залежностей та їхніх областей, що полегшує налаштування точок ін'єкції у коді. Завдяки спрощеним анотаціям і вбудованій інтеграції з компонентами фреймворку Android, налаштування ін'єкції залежностей у проєкті Android стає простішим. Hilt забезпечує готову підтримку для впровадження залежностей у ViewModels, що дозволяє ефективно керувати залежностями в компонентах архітектури Android. Hilt пропонує попередньо визначені області, такі як @Singleton,

@ActivityScoped і @FragmentScoped, для керування життєвим циклом залежностей у компонентах Android, забезпечуючи належне управління пам'яттю та можливість повторного використання. Для проєктів, які вже використовують Dagger, Hilt пропонує плавний шлях міграції, дозволяючи поступове впровадження без значних змін до існуючого коду[29].

У поєднанні з іншими компонентами Android-екосистеми, такими як Room для роботи з базами даних і Retrofit для виконання мережевих запитів, Hilt і Jetpack Compose створюють потужне середовище для розробки ефективних і масштабованих мобільних додатків. Зокрема, бібліотека Retrofit спрощує створення HTTP-запитів, уніфікуючи процес незалежно від кінцевих точок чи параметрів. Завдяки використанню анотацій для визначення запитів ця бібліотека зменшує обсяг шаблонного коду, забезпечуючи високий рівень автоматизації та контроль над мережею[40].

Хоча обробка анотацій часто пов'язана з більш тривалим часом компіляції, цей підхід виправдовує себе завдяки спрощенню розробки та покращенню читабельності коду. Альтернативою може бути використання проксі-серверів інтерфейсу, що дозволяє додатково оптимізувати виконання однотипних операцій із різними параметрами. Незважаючи на те, що такий підхід може викликати занепокоєння щодо впливу на продуктивність, затримки під час виконання мережевих запитів значно перевищують будь-які додаткові витрати, пов'язані з використанням проксі-сервера або згенерованого коду [32]. Це робить бібліотеки, які застосовують такі механізми, невід'ємним інструментом сучасної Android-розробки, дозволяючи розробникам зосередитися на інноваціях та задоволенні потреб користувачів, водночас забезпечуючи високий рівень надійності та продуктивності.

2.2. Програмна реалізація додатку

Після детального аналізу технологій розробки було прийнято рішення використовувати мову програмування Kotlin, зважаючи на її високу інтеграцію з операційною системою Android та можливість застосування сучасних

бібліотек і фреймворків, що сприяють оптимізації процесу розробки. На даному етапі ми переходимо до програмної реалізації. Для забезпечення високої ефективності функціонування додатку та можливості реалізації відгуків у реальному часі обрано архітектуру MVVM (Model-View-ViewModel). Це дозволяє чітко розділити логіку бізнес-процесів, інтерфейс користувача та управління даними, що значно полегшує підтримку коду, його тестування та масштабування[17].

В основі додатку лежить обробка даних з зовнішніх API, які надають інформацію про графік відключення в певних регіонах. API-запити надсилаються з певною періодичністю, щоб забезпечити постійне оновлення інформації в реальному часі. Для цього використовуються асинхронні методи роботи з мережею, зокрема бібліотека Retrofit, яка дозволяє спрощено виконувати HTTP-запити та обробляти відповіді. Додатково застосовувалася бібліотека OkHttp для покращення управління мережею та забезпечення коректної обробки можливих мережевих помилок.

Для отримання графіків відключень використовується API компанії ДТЕК, яке надає актуальну інформацію про планові та аварійні відключення електроенергії Інформація включає в себе:

- Планові відключення, які заплановані заздалегідь для проведення технічного обслуговування, модернізації електромереж або зменшення навантаження на мережі.
- Аварійні відключення, які виникають несподівано через аварії або інші непередбачувані обставини.

API компанії ДТЕК надає можливість отримувати ці дані у форматі, зручному для інтеграції в мобільні додатки. Це дозволяє додатку автоматично оновлювати інформацію про відключення, забезпечуючи користувачів актуальними даними в реальному часі.

```

24 object RetrofitClient {
25     private const val BASE_URL = "https://api.dtek.com/"
26
27     private val loggingInterceptor = HttpLoggingInterceptor().apply { this: HttpLoggingInterceptor
28         level = HttpLoggingInterceptor.Level.BODY
29     }
30
31     private val okHttpClient = OkHttpClient.Builder()
32         .addInterceptor(loggingInterceptor)
33         .build()
34
35     val instance: DtekService by lazy {
36         val retrofit = Retrofit.Builder()
37             .baseUrl(BASE_URL)
38             .client(okHttpClient)
39             .addConverterFactory(GsonConverterFactory.create())
40             .build()
41
42         retrofit.create(DtekService::class.java)
43     }
44 }
45

```

Рис. 2.2. Реалізація API інтерфейсу застосунку

Важливим компонентом додатку є робота з даними в режимі офлайн, коли користувачі можуть не мати доступу до Інтернету через тривалі відключення електропостачання. Для цього було реалізовано локальне кешування даних за допомогою бібліотеки Room, яка є частиною Jetpack і надає простий інтерфейс для роботи з базою даних SQLite. Бібліотека Room значно спрощує роботу з базою даних Android SQLite, забезпечуючи інтерфейс високого рівня абстракції. Її використання дозволяє зменшити обсяг коду, зробити його більш структурованим та полегшити процес розробки[24].

Завдяки Room розробникам не потрібно витратити значний час на опис базових взаємодій, таких як SQL-запити, у межах програми. Достатньо додати необхідні компоненти бібліотеки, щоб отримати готові інструменти, які значно спрощують та прискорюють роботу.

Використання Room в проєкті дозволяє зберігати останню отриману інформацію про графіки відключення та відображати їх користувачам навіть за відсутності з'єднання з мережею.

Інтерфейс користувача був розроблений з урахуванням принципів UX-дизайну, щоб забезпечити простоту та зручність використання додатку в умовах стресових ситуацій, коли користувачеві необхідно швидко отримати важливу інформацію[27]. На головному екрані користувач має можливість перевірити наявність електропостачання у своєму помешканні. Функціонал реалізовано таким чином, що при натисканні на кнопку перевірки, здійснюється запит до IP-адреси роутера, яку користувач попередньо вказав у своєму особистому кабінеті. У разі отримання відповіді від роутера, система інформує користувача про наявність електропостачання.

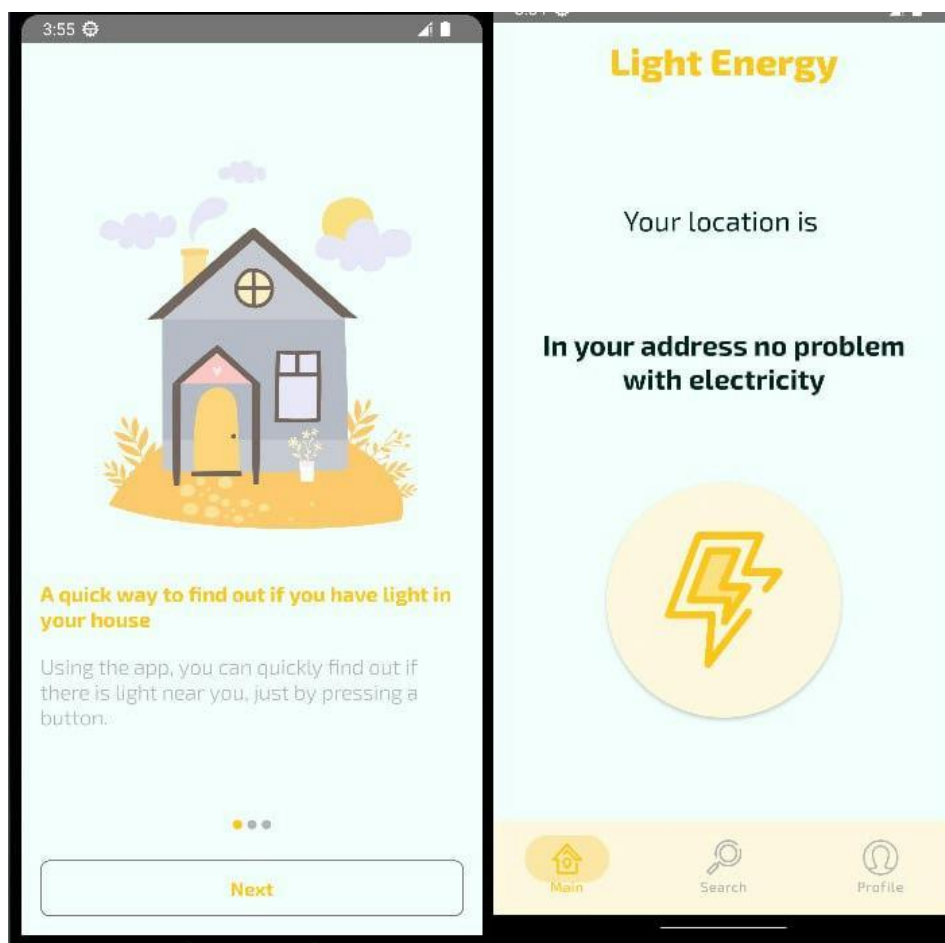


Рис. 2.3. Головний екран мобільного додатку та екран інструкції

На інших екранах користувач має можливість здійснювати пошук певної адреси та переглядати графік відключень електропостачання. Цей функціонал дозволяє користувачеві дізнатися, коли за вказаною адресою планується відключення електроенергії. Інформація про планові відключення надається у

вигляді зручного для сприйняття графіка, що дозволяє користувачеві завчасно підготуватися до можливих перебоїв у електропостачанні.

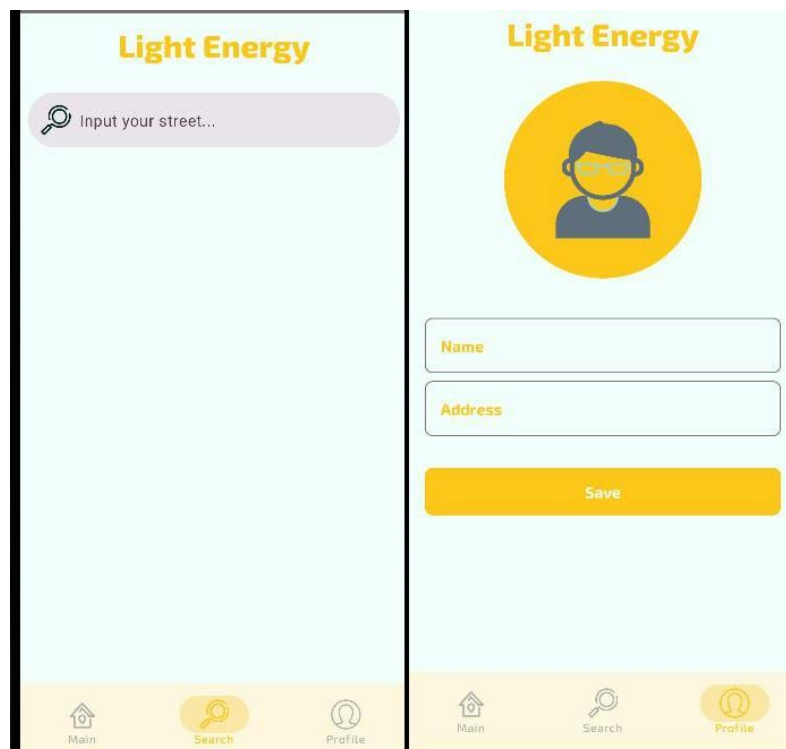


Рис. 2.4. Екран пошуку адреси та особистий кабінет

Для реалізації інтерфейсу користувача використовувалася бібліотека Jetpack Compose, яка дозволяє створювати адаптивні інтерфейси з меншими витратами на підтримку та оновлення[25]. Також, була впроваджена система push-сповіщень для негайного інформування користувачів про нові події, пов'язані з відключенням або відновленням електропостачання.

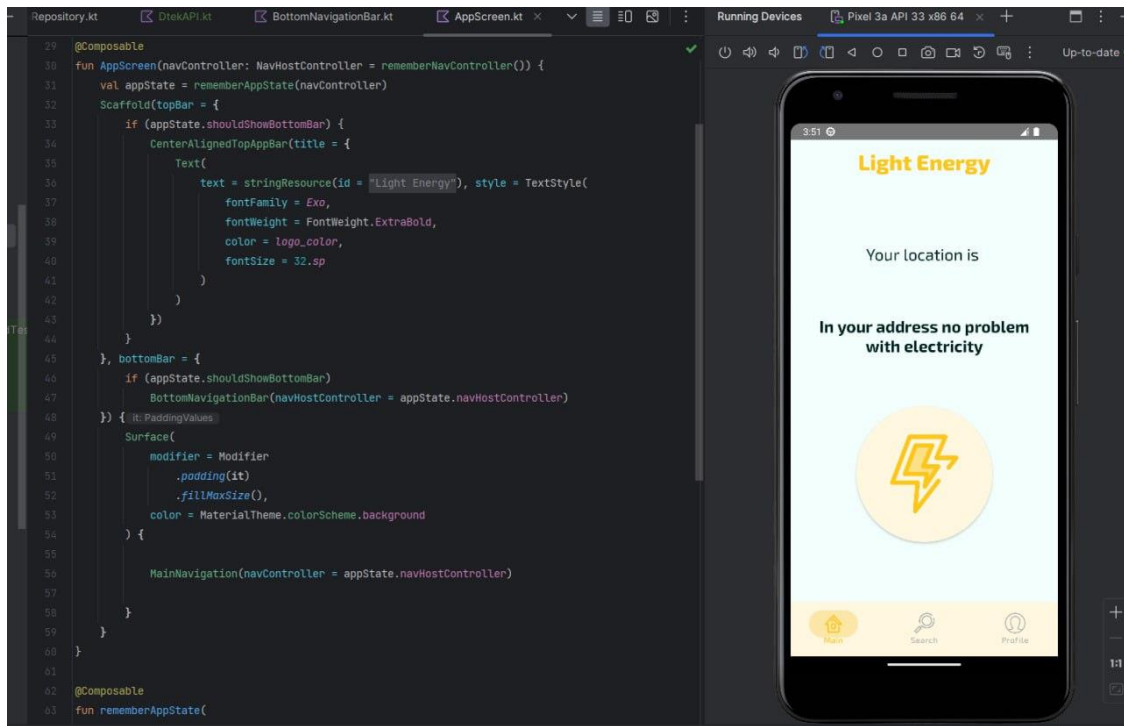


Рис. 2.5. Екран додатку, та вигляд його в коді

Також, для забезпечення стабільної та безперебійної роботи додатку на різних пристроях Android було проведено тестування як на реальних пристроях, так і на віртуальних емуляторах. Крім того, враховано особливості різних версій операційної системи Android для забезпечення сумісності додатку з якомога ширшою аудиторією користувачів. Завдяки цьому вдалося створити функціональний і надійний продукт, який відповідає сучасним вимогам до якості програмного забезпечення.

Оцінка ефективності мобільного додатку є наступним важливим етапом, що дозволяє визначити, наскільки успішно продукт виконує свої завдання та задовольняє потреби користувачів. Цей процес базується на аналізі ключових метрик, які допомагають оцінити популярність додатку, його функціональність та вплив на користувачів. Наприклад, такі показники, як кількість завантажень, реєстрацій та активних користувачів, дозволяють сформулювати повне уявлення про ефективність впровадженого рішення, забезпечуючи розробникам дані для подальшого вдосконалення продукту.

Активність користувачів аналізується за допомогою метрик DAU (Daily Active Users), WAU (Weekly Active Users) та MAU (Monthly Active Users), що

дозволяють оцінити частоту використання застосунку в різних часових проміжках. Ці дані потрібні для визначення залученості користувачів та довгострокової ефективності застосунку[8].

Залучення користувачів відображає рівень їхньої зацікавленості та тривалості використання мобільного додатку. Цей показник дає змогу оцінити частку користувачів, які залишаються активними протягом певного часу, їхню щоденну активність, а також тривалість взаємодії із функціоналом програми.

Для оцінки залучення застосовується співвідношення активних користувачів за короткий період (наприклад, день) до активних користувачів за триваліший інтервал (тиждень або місяць). У теорії максимальний рівень залучення становить 100%, що вказує на постійну активність усіх користувачів, однак у реальних умовах такі значення досягаються рідко. Метою розробників є досягнення максимально можливого рівня залучення для забезпечення стійкого інтересу до додатку.

Коефіцієнт утримання також є важливою метрикою, що визначає, як довго користувачі залишаються активними після встановлення програми. Цей показник дозволяє відстежити динаміку користувацької активності та момент її зниження. Високий коефіцієнт утримання, як правило, свідчить про успішність додатку, однак для програм із вузькою цільовою метою (наприклад, одноразового використання) він може бути менш значущим. У таких випадках акцент зміщується на ефективність просування програми, а не на тривалість її використання.

Ще одним важливим параметром для оцінки ефективності застосунку є його віральність. Віральність мобільного додатку характеризує здатність користувачів залучати нових учасників, зокрема друзів чи знайомих, через реферальні механізми. У додатках із соціальною складовою або залежністю від розширення аудиторії цей параметр має критичне значення для вдосконалення маркетингових стратегій. Для оцінки віральності використовуються аналітичні системи, які дозволяють визначити ефективність залучення нових користувачів.

Водночас у програмах, де соціальна взаємодія не є основною функцією, віральність може бути малозначущою.

Монетизація є ключовим аспектом успішності мобільного додатку. Основними метриками для її оцінки є середній дохід з одного користувача (ARPU – Average Revenue Per User) та середній дохід з користувача, що здійснює покупки (ARPPU – Average Revenue Per Paying User). Перший показник дозволяє оцінити загальну ефективність монетизації, тоді як другий дає уявлення про середній чек транзакцій. Співвідношення між цими показниками відображає ефективність воронки продажів і може слугувати основою для вдосконалення стратегії монетизації[8].

Вартість залучення одного користувача (CAC – Customer Acquisition Cost) є показником для аналізу окупності рекламних кампаній мобільного додатку. Вона розраховується як співвідношення витрат на маркетинг до кількості залучених користувачів і дозволяє оцінити ефективність витрачених ресурсів. Цей показник потрібен для прогнозування рентабельності інвестицій та оптимізації стратегії просування.

Сарафанність (Word-of-Mouth) також відіграє важливу роль у залученні нових користувачів, навіть за відсутності реферальних механізмів у застосунку. Відстеження джерел залучення, включно з рекомендаціями користувачів, дозволяє оцінити рівень їхньої задоволеності. Для цього можна використовувати опитування або інтеграцію з аналітичними системами, що фіксують походження користувачів.

Коефіцієнт віральності (K-Factor) визначає співвідношення між користувачами, залученими через рекламні кампанії, та тими, хто дізнався про додаток завдяки рекомендаціям. Високий K-Factor сприяє зменшенню залежності від платного просування, збільшуючи аудиторію без додаткових витрат.

Ціна ефективної установки (eCPI – Effective Cost Per Install) враховує не лише витрати на залучення користувачів, але й ефективність вірусного поширення додатку. Вона забезпечує точніший аналіз реальних витрат на залучення кожного

нового користувача, допомагаючи оптимізувати маркетингову стратегію та оцінювати успішність кампаній.

Окупність інвестицій (ROI – Return on Investment) це ще один ключовий показником, який відображає співвідношення між чистим прибутком та витратами на розробку і просування мобільного додатку. Високий показник ROI свідчить про ефективність застосунку та його здатність приносити прибуток, підтверджуючи доцільність інвестицій. Хоча ця метрика не виявляє слабкі місця продукту, вона забезпечує стратегічне бачення його успішності. Для отримання більш детальної оцінки доцільно використовувати опитування користувачів, стимулювати надання відгуків та регулярно аналізувати робочі метрики[8].

Firebase Analytics від Google пропонує розробникам інтегрований аналітичний інструментарій, що дозволяє комплексно оцінювати поведінкові патерни користувачів та ефективність застосунку під ОС Android. Сервіс забезпечує доступ до широкого спектру показників, включно з даними про активність користувачів, тривалість сеансів та ефективність маркетингових кампаній. Основні метрики, доступні в сервісі, включають:

- трафік і кількість нових та загальних користувачів;
- щоденну, щотижневу та щомісячну активність;
- коефіцієнт утримання користувачів;
- середню тривалість сеансу;
- віральність та конверсію;
- транзакції та середній дохід на одного користувача;
- окупність інвестицій.

Серед додаткових можливостей Firebase Analytics:

- push-сповіщення для підвищення активності користувачів;
- креш-аналітика для відстеження помилок;
- інструменти для тестування додатків;
- функція відстеження шляхів користувачів до цільових дій.

Однією з ключових переваг застосування аналітичних інструментів, зокрема Firebase Analytics, є можливість прогнозування поведінки користувачів,

що сприяє ефективнішому плануванню оновлень та маркетингових заходів. Завдяки хмарному зберіганню даних результати аналізу залишаються постійно доступними, забезпечуючи гнучкість у прийнятті управлінських рішень[22].

Використання таких сервісів у процесі розробки мобільного додатку для моніторингу відключень електроенергії в мережі під ОС Android дозволяє оптимізувати функціональність, вдосконалювати стратегії просування та підвищувати рентабельність продукту. Базовий набір інструментів можна розширити платними функціями, що поглиблюють аналітичні можливості.

Системний аналіз статистичних даних допомагає визначати сильні та слабкі сторони продукту, оцінювати конкурентоспроможність, підвищувати точність планування та орієнтувати стратегії розвитку на реальні показники. Зрештою це позитивно впливає на фінансові результати проєкту та збільшує прибутковість застосунку.

РОЗДІЛ 3. ОЦІНКА ЕФЕКТИВНОСТІ МОБІЛЬНОГО ЗАСТОСУНКУ

Проведення дослідження ефективності розробленого мобільного додатку для моніторингу відключень електроенергії в мережі під ОС Android із залученням інструментів аналітики Firebase передбачає комплексний підхід, що охоплює збір, обробку та інтерпретацію даних про користувацьку взаємодію та стабільність роботи продукту.

Firebase Analytics виступає як основний інструмент для збору та опрацювання статистичних даних про додаток. Інтеграція із сервісом здійснюється шляхом впровадження офіційної бібліотеки Firebase в код Android-застосунку. Ключові компоненти:

- Firebase Analytics: для фіксації подій (events), параметрів (parameters) та користувацьких властивостей (user properties).
- Firebase Crashlytics: для відстеження помилок та збоїв.
- Firebase Remote Config: для оперативного оновлення конфігураційних параметрів без необхідності перереінсталювати додаток.
- Firebase Predictions: для побудови моделей поведінки користувачів та прогнозування тенденцій[21].

Для інтеграції Firebase Analytics у мобільний застосунок Light Energy необхідно виконати низку попередніх кроків. Спершу слід завантажити файл google-services.json та розмістити його у відповідній директорії проєкту. Після цього у конфігураційних файлах проєкту вносяться зміни, а до вихідного коду додаються додаткові налаштування. Наведений нижче фрагмент демонструє приклад підключення аналітичних інструментів Firebase[16]:

```
id("com.google.gms.google-services")  
implementation(platform("com.google.firebase:firebase-bom:33.7.0"))  
implementation("com.google.firebase:firebase-analytics")
```

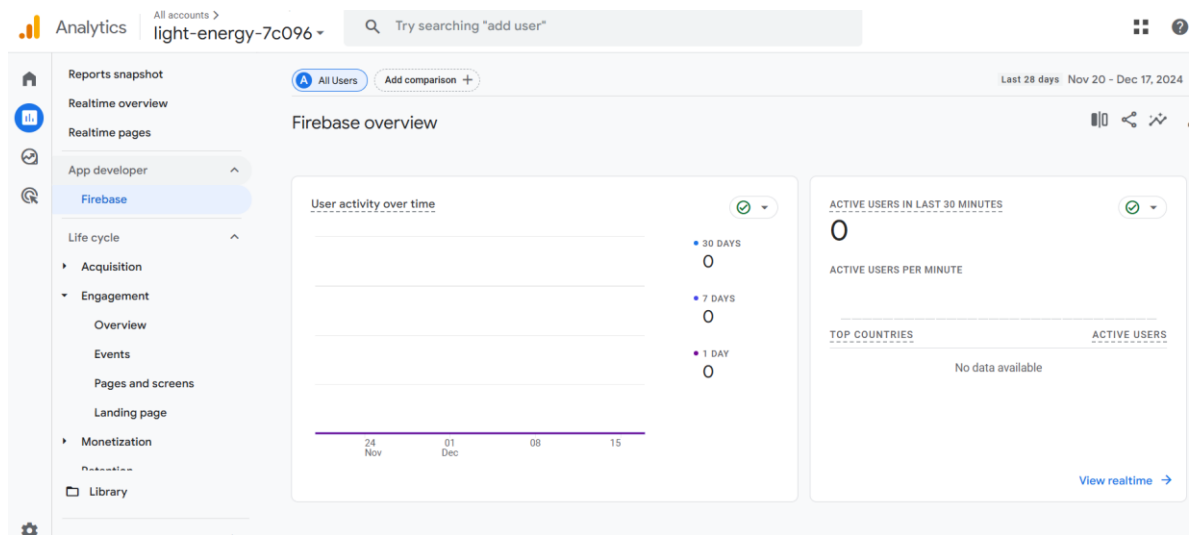


Рис. 3.1. Вікно Firebase Analytics

Дослідження ефективності мобільного застосунку Light Energy проводилося з використанням комплексного підходу, що охоплював технічний аналіз взаємодії користувачів із застосунком.

Для отримання релевантних даних та максимально реалістичних умов використання було створено тестову групу з 10 осіб. Ця група складалася з двох основних підгруп:

- Друзі та знайомі розробника (8 осіб): цю підгрупу було обрано завдяки особистому доступу до людей, котрі були мотивовані надати детальний та щирий зворотний зв'язок.
- Співробітники ТОВ «Комтрейд Сервіс» (2 особи): дана підгрупа дозволила оцінити ефективність застосунку в умовах бізнес-середовища. Їх внесок був важливим для розуміння практичного впливу сервісу на робочі процеси підприємства.

Учасникам дослідження було надано посилання на тестову версію застосунку Light Energy, призначену для пристроїв на базі операційної системи Android. Після отримання посилання користувачі власноруч встановлювали застосунок на свої смартфони. Це забезпечувало реалістичні умови, оскільки учасники не отримували жодних спеціальних інструкцій, окрім базового пояснення функціоналу. Їм було запропоновано використовувати додаток так, як вони це робили б у повсякденному житті.

Тестовий період тривав 7 днів. Такий термін був обраний з огляду на Реальну динаміку відключень електроенергії, які могли статися впродовж цього часу, також на можливість відстежити, як користувачі адаптуються до функціоналу застосунку, змінюють налаштування та формують певні моделі поведінки за кілька днів.

Протягом цього тижня середовище було досить мінливим (через нестабільну ситуацію з електропостачанням в Україні), що дало змогу оцінити застосунок в умовах реальних планових та аварійних відключень. Завдяки цьому учасники могли перевірити не лише корисність основних функцій, але й оперативність оновлення даних та доречність сповіщень.

З метою отримання максимально об'єктивної інформації застосовувався Firebase Analytics. За допомогою цього сервісу фіксувалися поведінкові події користувачів у застосунку. Вимірювалися кількість відкриттів застосунку, тривалість сесій, перегляди окремих екранів, використання функцій налаштування сповіщень та інші статистичні параметри.

Також для отримання реальних відгуків користувачі могли написати в довільному форматі (наприклад, у вигляді повідомлень електронною поштою чи в месенджері). Це дозволило виявити глибші інсайти, які не завжди проглядаються крізь призму автоматичного збору даних.

У дослідженні були визначені основні метрики для оцінки результативності використання застосунку, це:

- Кількість відкриттів застосунку за день: дає уявлення про частоту звернень до сервісу. Середній показник склав 3 відкриття на користувача на добу, що говорить про регулярну потребу в отриманні актуальної інформації.
- Тривалість сесії: у середньому 40–60 секунд. Цей час свідчить, що користувачі швидко отримували потрібні дані (інформація про відключення, перевірка стану мережі) та не потребували тривалого перебування у застосунку.
- Налаштування сповіщень: близько 80% учасників активували push-сповіщення про майбутні відключення. Такий високий показник підтверджує практичну цінність проактивного інформування.

- Перевірка електропостачання за IP-адресою роутера: приблизно 90% користувачів протестували цю функцію, а 70% визнали її практично корисною. Це підтвердило попередню гіпотезу, що можливість дізнатися про наявність світла дистанційно – значна перевага для цільової аудиторії.

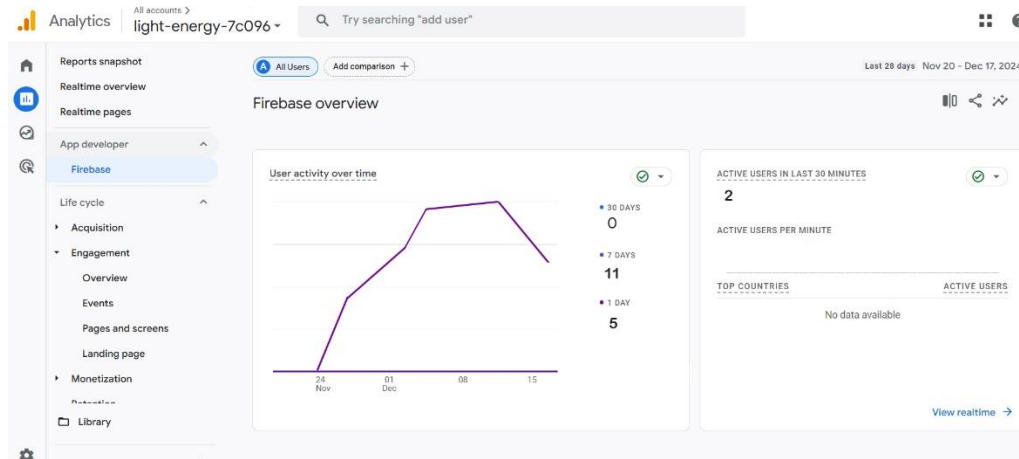


Рис. 3.2. Результати дослідження Firebase

Крім сухих статистичних даних, було проаналізовано відгуки користувачів, згідно з якими 7 з 10 учасників оцінили інтерфейс як інтуїтивний та простий. Це свідчить про те, що дизайн прийнятих рішень щодо навігації та розташування елементів інтерфейсу загалом відповідає потребам користувачів.

Згідно з оцінкою якості даних та актуальності інформації, 70 відсотків користувачів залишилися задоволеними актуальністю інформації про відключення. Однак було зазначено, що деякі затримки в оновленні даних потребують покращення.

Враховуючи унікальну функцію перевірки відключення вдома згідно з IP-адресою роутера, 8 користувачів виокремили цю опцію як особливо цінну. Вона дозволяє зрозуміти, чи є в помешканні електропостачання, не виходячи з дому та не вмикаючи додаткові прилади.

На основі отриманих даних було визначено кілька напрямів для вдосконалення застосунку. Першим є оновлення графіків відключень, де головним недоліком є затримки в оновленні даних, що негативно впливає на точність та своєчасність інформації. Вирішення цієї проблеми є ключовим кроком для підвищення довіри користувачів.

Другим важливим аспектом є розширення функціоналу та налаштувань. Запит користувачів на розширення можливостей персоналізації, таких як фільтри за регіоном, типом відключення, та інтеграція з іншими сервісами, свідчить про потенціал ускладнення продукту за рахунок нових опцій.

Третім напрямом є візуалізація та інтерфейсні поліпшення. Користувачі бажають бачити більш наочні карти та чітке розмежування зон відключень, що може поліпшити розуміння ситуації та спростити навігацію. Окрему увагу слід приділити впливу використання застосунку на діяльність підприємства ТОВ «Комтрейд Сервіс». Моделювання показало, що своєчасна інформація про відключення дозволяла зменшити середньодобовий простій обладнання на 15–20%, прогнозувала економію коштів за тиждень у розмірі близько 1000–1500 грн. Оскільки компанія займається в основному ремонтом техніки, де в середньому обробляє 100 замовлень на тиждень, то знаючи про майбутні відключення, компанія може планувати графік роботи працівників так, щоб вони займалися завданнями, які не потребують електроенергії (наприклад, діагностика окремих деталей, підготовка документації, пакування, сортування комплектуючих) у періоди без електропостачання. Це мінімізує "холості" години, коли працівники не можуть виконувати свою основну роботу.

Також використання додатка підвищує ефективність управління ресурсами, оскільки дає змогу планувати робочий час, підготувати резервні джерела живлення та мінімізувати зупинки робочого процесу.

Дослідження було проведено системно, поєднуючи інструменти аналітики (Firebase Analytics) та опитування користувачів. Такий підхід забезпечив повне розуміння реальної поведінки користувачів, їхніх потреб та болючих точок у роботі застосунку. Отримані дані стали основою для подальшого вдосконалення Light Energy, як з точки зору функціональності, так і з точки зору покращення користувацького досвіду.

ВИСНОВКИ

Під час виконання кваліфікаційного проєкту, присвяченого розробці мобільного додатку для моніторингу відключення світла в мережі під ОС Android, було здійснено комплексний аналіз предметної області та обґрунтовано актуальність такого рішення. В сучасних умовах, коли енергетична інфраструктура України зазнає регулярних пошкоджень внаслідок ворожих ракетних обстрілів, питання стабільного електропостачання стає вкрай важливим. Масовані атаки на об'єкти критичної інфраструктури призводять до планових і позапланових відключень світла, що суттєво ускладнює побут, бізнес-процеси та роботу комунальних служб. Було проведено аналітичний огляд літературних джерел та існуючих мобільних застосунків, спрямованих на інформування користувачів про перебої в електропостачанні. Зокрема, було розглянуто сучасний стан проблематики, ідентифіковано основні недоліки поточних рішень, а також визначено напрямки покращення. У процесі огляду було сформульовано функціональні та нефункціональні вимоги до застосунку, що дозволило чітко окреслити цілі та критерії його ефективності.

Також увага була приділена технологічним аспектам розробки додатку. Зокрема, основною мовою програмування було обрано Kotlin, яка є офіційно підтримуваною мовою розробки під Android та забезпечує підвищену надійність, чіткість коду й скорочення кількості помилок порівняно з традиційним Java-кодом. На основі проведеного аналізу інструментів та фреймворків для Android-розробки (зокрема, Android Jetpack компонентів) було створено архітектуру програмного продукту, яка забезпечує високу продуктивність, гнучкість та масштабованість. При цьому застосовано архітектурний патерн MVVM для спрощення підтримки та розширюваності коду. Реалізація додатку включала розробку інтерфейсу користувача з використанням сучасних UI-компонентів Android, налаштування підключення до сервісів для оперативного отримання даних про відключення електроенергії, а також інтеграцію інструментів аналітики (Firebase Analytics), що дозволило

відстежувати поведінку користувачів та приймати обґрунтовані рішення щодо подальшого вдосконалення продукту.

В кінці було визначено критерії оцінки ефективності мобільного застосунку, проведено дослідження за участі тестової групи користувачів та проаналізовано результати цього дослідження. Збір статистичних даних про частоту використання, зручність інтерфейсу, швидкість оновлення інформації та корисність унікальних функцій (зокрема, дистанційної перевірки наявності електроенергії) дав змогу оцінити відповідність застосунку визначеним вимогам та цільовим індикаторам. Результати дослідження підтвердили актуальність і практичну цінність розробленого рішення, а також виявили напрями для його подальшого вдосконалення, як-от покращення алгоритмів оновлення графіків відключень чи розширення функціоналу через інтеграції з іншими сервісами.

Завдяки проведеному аналізу сучасного стану предметної області, визначенню вимог, раціональному вибору технологій та оцінці ефективності прототипу, було створено мобільний додаток для ОС Android, який надає користувачам інструменти для моніторингу відключення світла та проактивного планування своїх дій. Це рішення є важливим внеском у підвищення поінформованості споживачів та ефективність управління ресурсами як для пересічних користувачів, так і для підприємств, що працюють в умовах нестабільного електропостачання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гурцак Є. О., Геселева Н. В. Інформаційні системи управління виробництвом : thesis. 2016.
URL: <https://er.knutd.edu.ua/handle/123456789/4451> (дата звернення: 04.01.2025).
2. Два місяці без світла: як український бізнес виживає на тлі блекаутів. *Mind.ua*. URL: <https://mind.ua/publications/20250610-dva-misyaci-bez-svitla-yak-ukrayinskij-biznes-vizhivae-na-tli-blekautiv> (дата звернення: 20.12.2024).
3. Економічна правда. Як бізнесам пережити тотальний блекаут. *Економічна правда*.
URL: <https://epravda.com.ua/columns/2022/11/15/693806/> (дата звернення: 20.12.2024).
4. Економічна правда. Як довго триватимуть вимкнення світла і чи можливий блекаут: питання-відповіді. *Економічна правда*.
URL: <https://epravda.com.ua/publications/2022/11/10/693656/> (дата звернення: 20.12.2024).
5. З'явився новий додаток для моніторингу відключень світла у Києві та області | нашкіїв.ua. *НашКиїв.UA*. URL: <https://nashkiev.ua/news/zyavivsysya-novii-dodatok-dlya-monitoringu-vidklyuchen-svitla-u-kievi-ta-oblasti> (дата звернення: 20.12.2024).
6. Інтернет-моніторинг сонячної станції - Solar Garden. *Solar Garden*.
URL: <https://www.solargarden.com.ua/service/monitoryng-generatsiyi-sonyachnoyi-elektrostantsiyi/> (дата звернення: 04.01.2025).
7. Інформаційні технології : Норм. база. Київ : КНТ, вид-во, 2015. 500 с.
8. Маймур І. Українські айтівці створили застосунок “Світлобот”, який моніторить відключення світла. *Журнал великого міста*.
URL: <https://bzh.life/ua/lyudi/1719390484-ukrayintsi-stvorili-zastosunok-svitlobot/> (дата звернення: 20.12.2024).

9. Мобільний застосунок YASNO. *Офіційний сайт YASNO.*
URL: <https://kyiv.yasno.com.ua/mobile-application> (дата звернення: 20.12.2024).
10. Моніторинг відключень світла в Home Assistant. *Dou.ua.*
URL: <https://dou.ua/forums/topic/49537/>.
11. Найкращі інструменти для розробки мобільних додатків Artjoker. *Створення та розробка сайтів Київ, Харків. Замовити створення веб сайту під ключ з нуля - Веб студія Artjoker.* URL: <https://artjoker.ua/blog/luchshie-instrumenty-dlya-sozdaniya-mobilnykh-prilozheniy/> (дата звернення: 04.01.2025).
12. Основні метрики мобільного додатка – Wezom. *Wezom.ua.*
URL: <https://wezom.com.ua/ua/blog/osnovnye-metriki-mobilnogo-prilozheniya> (дата звернення: 20.12.2024).
13. Перевірка поточних відключень. *ДТЕК Київські електромережі | Оператор системи розподілу.* URL: <https://www.dtek-kem.com.ua/ua/shutdowns> (дата звернення: 20.12.2024).
14. Польська А. 7 додатків, чатботів і карт, які допоможуть при відключеннях світла. *Журнал великого міста.* URL: <https://bzh.life/ua/gorod/5-dodatkov-chatbotiv-i-kart-yaki-dopomozhut-pri-vimknennyah-svitla-spisok/> (дата звернення: 20.12.2024).
15. Про затвердження Методичних рекомендацій застосування критеріїв визначення ефективності управління об'єктами державної власності. *Офіційний вебпортал парламенту України.*
URL: <https://zakon.rada.gov.ua/rada/show/v0314665-07> (дата звернення: 20.12.2024).
16. РОЗРОБКА МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ OS Android / М. Л. Дворецький та ін. Миколаїв : ЧНУ, 2021. 150 с.
17. Рясна О. В. Проектування системи автоматизації : thesis. 2017.
URL: <http://essuir.sumdu.edu.ua/handle/123456789/66638> (дата звернення: 04.01.2025).
18. Системи моніторингу якості електричної енергії SMART GRID | Інтернет-магазин Electrovmir. *Інтернет-магазин «Електровимір».*

URL: <https://www.evm.ua/novosti/sistemy-monitoryngu-yakosyi-ee-smart-grid> (дата звернення: 04.01.2025).

19. Системи накопичення електроенергії для дому та офісу: Starlight. *Логістична компанія, Inter Trans Logistics*. URL: <https://itl-ltd.com/blog/view/sistemi-nakopichennya-elektroenergiyi-dlya-domu-ta-ofisu-starlight/> (дата звернення: 04.01.2025).

20. Чи вб'є економіку України блекаут і як бізнес виживає без світла - BBC News Україна. *BBC News Україна*. URL: <https://www.bbc.com/ukrainian/features-64082615> (дата звернення: 20.12.2024).

21. Що таке IP-адреса і для чого вона потрібна? – Trium. *Trium*. URL: <https://trium.com.ua/43-shcho-take-ip-adresa-i-dlia-choho-vona-potribna> (дата звернення: 20.12.2024).

22. Як відключення електроенергії впливає на роботу бізнесу?. «Дебет-Кредит» - Бухгалтерські новини. URL: <https://news.dtki.ua/society/economics/80478-iak-vidkliucennia-elektroenergiyi-vplivaje-na-robotu-biznesu> (дата звернення: 20.12.2024).

23. Як зв'язати додатки з платформою Firebase - Google AdMob Довідка. *Google Help*. URL: <https://support.google.com/admob/answer/6383165?hl=uk> (дата звернення: 20.12.2024).

24. Як створити мобільний додаток: кроки та вартість | Wezom. *Wezom.ua*. URL: <https://wezom.com.ua/ua/blog/7-krokov-yak-stvoriti-mobilniy-dodatok> (дата звернення: 20.12.2024).

25. Як я розробив ElectricityOff – застосунок для моніторингу вимкнень світла. *Dou.ua*. URL: <https://dou.ua/forums/topic/49946/> (дата звернення: 17.10.2024).

26. 7 important advantages of Android app development. *DECODE*. URL: <https://decode.agency/article/android-app-development-benefits/> (date of access: 20.12.2024).

27. Android vs iOS App Development: Which One To Go For?. *London App Development (LAD)*. URL: <https://londonappdevelopment.co.uk/blog/android-vs-ios-development> (date of access: 20.12.2024).
28. AWS Storage category icon Storage - Overview of Amazon Web Services. URL: <https://docs.aws.amazon.com/whitepapers/latest/aws-overview/storage-services.html> (date of access: 04.01.2025).
29. Development L. S. iOS Developer vs Android Developer: A Comparison. *Medium*. URL: <https://leeddev.medium.com/ios-developer-vs-android-developer-a-comparison-ab9dd4ec40f6> (date of access: 20.12.2024).
30. Firebase | Google's Mobile and Web App Development Platform. *Firebase*. URL: <https://firebase.google.com/> (date of access: 20.12.2024).
31. Google Analytics for Firebase. *Firebase*. URL: <https://firebase.google.com/docs/analytics> (date of access: 20.12.2024).
32. Group C. W. Мережі онлайн – Додатки в Google Play. *Android Apps on Google Play*. URL: <https://play.google.com/store/apps/details?id=com.touchcard.ewallet&hl=uk> (дата звернення: 20.12.2024).
33. Hanifshaikh. What is Kotlin Programming Language? History, Benefits, Why Kotlin, etc. *Medium*. URL: <https://medium.com/@hanifshaikh8153/what-is-kotlin-programming-language-history-benefits-why-kotlin-etc-0b4d3b6632df> (date of access: 20.12.2024).
34. Jetpack Compose UI App Development Toolkit - Android Developers. *Android Developers*. URL: <https://developer.android.com/compose> (date of access: 20.12.2024).
35. Kotlin Programming Language. *Kotlin*. URL: <https://kotlinlang.org/> (date of access: 20.12.2024).
36. Mobile UX Design - The Ultimate Guide 2025. *Bluespace*. URL: <https://uxcam.com/blog/mobile-ux/> (date of access: 20.12.2024).
37. Somethingtoldme. Ін'єкція залежностей на Android за допомогою Hilt. *Medium*. URL: <https://medium.com/nuances-of-programming/инъекция->

[зависимостей-на-android-с-помощью-hilt-f2c4ac05cb57](#) (дата звернення: 04.01.2025).

38. Svitlo lab. Світло – Додатки в Google Play. *Android Apps on Google Play*.

URL: <https://play.google.com/store/apps/details?id=com.svitlo.app&hl=uk> (дата звернення: 20.12.2024).

39. The Importance of UX/UI Design in Mobile App Development | SmartDev. *SmartDev*. URL: [https://smartdev.com/the-importance-of-ux-ui-design-in-mobile-app-development/#:~:text=User%20Experience%20\(UX\)%20design%20and,their%20interaction%20with%20the%20app](https://smartdev.com/the-importance-of-ux-ui-design-in-mobile-app-development/#:~:text=User%20Experience%20(UX)%20design%20and,their%20interaction%20with%20the%20app). (date of access: 20.12.2024).

40. What is Android? | Android. *Android*. URL: <https://www.android.com/what-is-android/> (date of access: 20.12.2024).

ДОДАТКИ

Додаток А

```

package com.dev.meleshko.lightenergy.search.ui

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.SearchBar
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import com.dev.meleshko.lightenergy.R
import com.dev.meleshko.lightenergy.ui.theme.LightEnergyTheme

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SearchScreen(modifier: Modifier = Modifier, viewModel: SearchViewModel =
hiltViewModel()) {
    var query by remember {
        mutableStateOf("")
    }
    var mainList by remember {
        mutableStateOf(emptyList<String>())
    }
    var onActive by remember {
        mutableStateOf(false)
    }

    Column(modifier = modifier.fillMaxSize()) {
        SearchBar(

```

```

        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        placeholder = {
            Text(text = "Input your street...")
        },
        leadingIcon = {
            Icon(
                painter = painterResource(id = R.drawable.ic_search),
                contentDescription = stringResource(
                    id = R.string.search
                )
            )
        },
        query = query,
        onQueryChange = {
            query = it
            mainList = viewModel.findStreet(it)
        },
        onSearch = {
            onActive = false
        },
        active = onActive,
        onActiveChange = {
            onActive = it
        }
    ) {
        LazyColumn {
            items(items = mainList) {
                Box(
                    modifier = Modifier.clickable {
                        onActive = false
                    }
                ) {
                    Text(text = it)
                }
            }
        }
    }
}

@Preview
@Composable
fun SearchScreenPreview() {
    LightEnergyTheme {
        SearchScreen()
    }
}

```

```

    }
}

package com.dev.meleshko.lightenergy

import androidx.compose.material3.Icon
import androidx.compose.material3.NavigationBar
import androidx.compose.material3.NavigationBarItem
import androidx.compose.material3.NavigationBarItemDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.res.painterResource
import androidx.navigation.NavGraph.Companion.findStartDestination
import androidx.navigation.NavHostController
import androidx.navigation.compose.currentBackStackEntryAsState
import com.dev.meleshko.lightenergy.navigation.BottomScreens
import com.dev.meleshko.lightenergy.ui.theme.Exo
import com.dev.meleshko.lightenergy.ui.theme.light_gray
import com.dev.meleshko.lightenergy.ui.theme.logo_color
import com.dev.meleshko.lightenergy.ui.theme.logo_color_container
import com.dev.meleshko.lightenergy.ui.theme.logo_color_with_alfa

@Composable
fun BottomNavigationBar(
    navHostController: NavHostController
) {
    val navigationItems = listOf(
        BottomScreens.Main, BottomScreens.Search, BottomScreens.Profile
    )

    val navBackStackEntry by navHostController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route

    NavigationBar(containerColor = logo_color_container) {
        navigationItems.forEach { screen ->
            NavigationBarItem(
                colors = NavigationBarItemDefaults.colors(
                    indicatorColor = logo_color_with_alfa,
                    selectedIconColor = logo_color,
                    selectedTextColor = logo_color,
                    unselectedIconColor = light_gray,
                    unselectedTextColor = light_gray
                ),
                selected = currentRoute == screen.route,
                onClick = {
                    navHostController.navigate(screen.route) {

                        popUpTo(navHostController.graph.findStartDestination().id) {

```

```

        saveState = true
    }
    launchSingleTop = true
    restoreState = true
}
},
icon = {
    Icon(
        painter = painterResource(id = screen.iconRes),
        contentDescription = screen.name
    )
},
label = { Text(text = screen.name, fontFamily = Exo) }
}
}
}

```

```
package com.dev.meleshko.lightenergy
```

```

import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.CenterAlignedTopAppBar
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.Stable
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
import androidx.navigation.NavHostController
import androidx.navigation.compose.currentBackStackEntryAsState
import androidx.navigation.compose.rememberNavController
import com.dev.meleshko.lightenergy.navigation.BottomScreens
import com.dev.meleshko.lightenergy.navigation.MainNavigation
import com.dev.meleshko.lightenergy.ui.theme.Exo
import com.dev.meleshko.lightenergy.ui.theme.logo_color

```

```

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AppScreen(navController: NavHostController = rememberNavController()) {
    val appState = rememberAppState(navController)
    Scaffold(topBar = {
        if (appState.shouldShowBottomBar) {

```

```

        CenterAlignedTopAppBar(title = {
            Text(
                text = stringResource(id = R.string.light_energy), style = TextStyle(
                    fontFamily = Exo,
                    fontWeight = FontWeight.ExtraBold,
                    color = logo_color,
                    fontSize = 32.sp
                )
            )
        })
    }, bottomBar = {
        if (appState.shouldShowBottomBar)
            BottomNavigationBar(navHostController = appState.navHostController)
    }) {
        Surface(
            modifier = Modifier
                .padding(it)
                .fillMaxSize(),
            color = MaterialTheme.colorScheme.background
        ) {

            MainNavigation(navController = appState.navHostController)

        }
    }
}

@Composable
fun rememberAppState(
    navHostController: NavHostController = rememberNavController()
) = remember(navHostController) {
    AppState(navHostController)
}

@Stable
class AppState(
    val navHostController: NavHostController
) {
    private val navigationItems = listOf(
        BottomScreens.Main, BottomScreens.Search, BottomScreens.Profile
    )
    private val routes = navigationItems.map { it.route }

    val shouldShowBottomBar: Boolean
        @Composable get() =
            navHostController.currentBackStackEntryAsState().value?.destination?.route in routes
}

```

```
package com.dev.meleshko.lightenergy.splash.ui
```

```
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.size
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.dev.meleshko.lightenergy.R
import com.dev.meleshko.lightenergy.core.presentation.AppProgressIndicator
import com.dev.meleshko.lightenergy.ui.theme.Exo
import com.dev.meleshko.lightenergy.ui.theme.logo_color
import kotlinx.coroutines.delay
```

```
@Composable
```

```
fun SplashScreen(modifier: Modifier = Modifier, onNavigationToMain: () -> Unit) {
```

```
    Column(
        modifier = modifier.fillMaxSize(), horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
```

```
        Image(
            modifier = Modifier.size(200.dp),
            painter = painterResource(id = R.drawable.bolt_thunder_svgrepo_com),
            contentDescription = stringResource(R.string.logo)
        )
```

```
        Spacer(modifier = Modifier.height(32.dp))
```

```
        Text(
            text = stringResource(R.string.light_energy), style = TextStyle(
                color = logo_color,
                fontSize = 48.sp,
                fontFamily = Exo,
                fontWeight = FontWeight.ExtraBold
            )
        )
```

```

    )
    Spacer(modifier = Modifier.height(64.dp))
    AppProgressIndicator()
}
LaunchedEffect(key1 = Unit) {
    delay(2000L)
    onNavigationToMain()
}
}

```

```

@Preview(showBackground = true)
@Composable
fun SplashScreenPreview() {
    SplashScreen(onNavigationToMain = {})
}

```

```
package com.dev.meleshko.lightenergy.search.ui
```

```

import androidx.lifecycle.ViewModel
import com.dev.meleshko.lightenergy.data.DataRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import javax.inject.Inject

```

```

@HiltViewModel
class SearchViewModel @Inject constructor(
    private val repository: DataRepository
) : ViewModel() {

    fun findStreet(name: String): List<String> {
        return repository.getStreets().filter {
            it.lowercase().startsWith(name.lowercase())
        }
    }
}

```

```
package com.dev.meleshko.lightenergy.profile.ui
```

```

import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape

```

```

import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.dev.meleshko.lightenergy.R
import com.dev.meleshko.lightenergy.ui.theme.Exo
import com.dev.meleshko.lightenergy.ui.theme.LightEnergyTheme
import com.dev.meleshko.lightenergy.ui.theme.logo_color

```

@Composable

```

fun ProfileScreen(modifier: Modifier = Modifier) {
    var name by remember {
        mutableStateOf("")
    }
    var address by remember {
        mutableStateOf("")
    }

    Column(
        modifier = modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Top
    ) {
        Box(
            modifier = Modifier
                .size(200.dp)
                .clip(CircleShape)
                .background(logo_color)
        ) {
            Image(

```

```

        painter = painterResource(id = R.drawable.profile_default),
        contentDescription = stringResource(R.string.profile_image),
        modifier = Modifier
            .fillMaxSize(),
        contentScale = ContentScale.Fit
    )
}

Spacer(modifier = Modifier.height(32.dp))
Column(modifier = Modifier.fillMaxWidth()) {
    OutlinedTextField(
        modifier = Modifier.fillMaxWidth(), shape = RoundedCornerShape(8.dp),
        value = name, onValueChange = { name = it }, textStyle = TextStyle(
            fontFamily = Exo,
            fontWeight = FontWeight.Bold,
            color = logo_color
        ),
        label = {
            Text(
                text = stringResource(id = R.string.name), style = TextStyle(
                    fontSize = 16.sp,
                    fontFamily = Exo,
                    fontWeight = FontWeight.Bold,
                    color = logo_color
                )
            )
        }
    )
    OutlinedTextField(
        modifier = Modifier.fillMaxWidth(), shape = RoundedCornerShape(8.dp),
        value = address, onValueChange = { address = it }, textStyle = TextStyle(
            fontFamily = Exo,
            fontWeight = FontWeight.Bold,
            color = logo_color
        ),
        label = {
            Text(
                text = "Address", style = TextStyle(
                    fontSize = 16.sp,
                    fontFamily = Exo,
                    fontWeight = FontWeight.Bold,
                    color = logo_color
                )
            )
        }
    )
}
Spacer(modifier = Modifier.height(32.dp))
Button(
    onClick = { /*TODO*/ },
    modifier = Modifier

```

```

        .fillMaxWidth()
        .height(48.dp),
        shape = RoundedCornerShape(8.dp),
        colors = ButtonDefaults.buttonColors(
            containerColor = logo_color
        )
    ) {
        Text(
            text = "Save", style = TextStyle(
                fontSize = 16.sp,
                fontFamily = Exo,
                fontWeight = FontWeight.Bold,
                color = Color.White
            )
        )
    }
}

@Preview(showBackground = true)
@Composable
fun ProfileScreenPreview() {
    LightEnergyTheme {
        ProfileScreen()
    }
}

package com.dev.meleshko.lightenergy.onboarding.ui

import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.pager.PagerState
import androidx.compose.foundation.pager.rememberPagerState
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.OutlinedButton
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.TextStyle

```

```

import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.dev.meleshko.lightenergy.R
import com.dev.meleshko.lightenergy.onboarding.domain.OnboardingPage
import com.dev.meleshko.lightenergy.ui.theme.Exo
import com.dev.meleshko.lightenergy.ui.theme.light_gray
import com.dev.meleshko.lightenergy.ui.theme.logo_color
import kotlinx.coroutines.launch

```

```

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun OnboardingPageScreen(
    onboardingPage: OnboardingPage,
    modifier: Modifier = Modifier,
    pagerState: PagerState,
    onNextButtonClicked: () -> Unit = {}
) {
    val scope = rememberCoroutineScope()
    Column(
        modifier = modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Column(
            modifier = Modifier.weight(1f),
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Image(
                painter = painterResource(id = onboardingPage.imageRes),
                contentDescription = null
            )
            Spacer(modifier = Modifier.height(16.dp))
            Text(
                text = stringResource(id = onboardingPage.titleRes), style = TextStyle(
                    fontSize = 18.sp,
                    fontFamily = Exo,
                    fontWeight = FontWeight.Bold,
                    color = logo_color
                )
            )
            Spacer(modifier = Modifier.height(16.dp))

```

```

Text(
    text = stringResource(id = onboardingPage.descriptionRes), style = TextStyle(
        fontSize = 18.sp,
        fontFamily = Exo,
        color = light_gray,
        fontWeight = FontWeight.Light,
    )
)
}
Column(modifier = Modifier.weight(0.15f), verticalArrangement = Arrangement.Bottom) {
    RowPagerIndicator(pagerState, count = 3)
    Spacer(modifier = Modifier.height(16.dp))
    OutlinedButton(modifier = Modifier
        .fillMaxWidth()
        .height(48.dp),
        shape = RoundedCornerShape(8.dp),
        onClick = {
            if (pagerState.currentPage != 2) scope.launch {
                pagerState.animateScrollToPage(pagerState.currentPage + 1)
            } else {
                onNextButtonClicked()
            }
        }) {
        Text(
            text = stringResource(R.string.next), style = TextStyle(
                fontSize = 16.sp,
                fontFamily = Exo,
                fontWeight = FontWeight.Bold,
                color = logo_color
            )
        )
    }
}
}
}
}

```

```

@OptIn(ExperimentalFoundationApi::class)
@Preview(showBackground = true)
@Composable
fun OnboardingPageScreenPreview() {
    OnboardingPageScreen(
        onboardingPage = OnboardingPage(
            titleRes = R.string.onboarding1_title,
            imageRes = R.drawable.home,
            descriptionRes = R.string.onboarding1_description
        ),
        pagerState = rememberPagerState {
            1
        }
    )
}

```

```

        }, onNextButtonClicked = {

        }
    )
}

package com.dev.meleshko.lightenergy.onboarding.ui

import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.pager.HorizontalPager
import androidx.compose.foundation.pager.rememberPagerState
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.dev.meleshko.lightenergy.R
import com.dev.meleshko.lightenergy.onboarding.domain.OnboardingPage

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun OnboardingScreen(
    modifier: Modifier = Modifier,
    onNextButtonClick: () -> Unit
) {

    val pages = listOf(
        OnboardingPage(
            titleRes = R.string.onboarding1_title,
            descriptionRes = R.string.onboarding1_description,
            imageRes = R.drawable.home
        ),
        OnboardingPage(
            titleRes = R.string.onboarding2_title,
            descriptionRes = R.string.onboarding2_description,
            imageRes = R.drawable.location
        ),
        OnboardingPage(
            titleRes = R.string.onboarding3_title,
            descriptionRes = R.string.onboarding3_description,
            imageRes = R.drawable.prapare
        ),
    )

    val pagerState = rememberPagerState {
        pages.size
    }

    HorizontalPager(state = pagerState) {
        OnboardingPageScreen(
            onboardingPage = pages[it],
            pagerState = pagerState,

```

```

        modifier = modifier,
        onNextButtonClicked = {
            onNextButtonClick()
        }
    )
}

}

package com.dev.meleshko.lightenergy.onboarding.ui

import androidx.compose.foundation.ExperimentalFoundationApi
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.wrapContentHeight
import androidx.compose.foundation.pager.PagerState
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.unit.dp
import com.dev.meleshko.lightenergy.ui.theme.light_gray
import com.dev.meleshko.lightenergy.ui.theme.logo_color

@OptIn(ExperimentalFoundationApi::class)
@Composable
fun RowPagerIndicator(pagerState: PagerState, count: Int) {
    Row(
        Modifier
            .wrapContentHeight()
            .fillMaxWidth()
            .padding(bottom = 8.dp),
        horizontalArrangement = Arrangement.Center
    ) {
        repeat(count) { iteration ->
            val color =
                if (pagerState.currentPage == iteration) logo_color else light_gray
            Box(
                modifier = Modifier
                    .padding(2.dp)
                    .clip(CircleShape)
                    .background(color)
                    .size(8.dp)
            )
        }
    }
}

```

```

    }
}
}

```

```
package com.dev.meleshko.lightenergy.navigation
```

```

import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.navigation.NavHostController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import com.dev.meleshko.lightenergy.onboarding.ui.OnboardingScreen
import com.dev.meleshko.lightenergy.splash.ui.SplashScreen

```

```
@Composable
```

```

fun MainNavigation(modifier: Modifier = Modifier, navController: NavHostController) {
    NavHost(
        modifier = modifier,
        navController = navController,
        startDestination = Screens.Splash.route
    ) {
        composable(route = Screens.Splash.route) {
            SplashScreen(onNavigationToMain = {
                navController.navigate(Screens.OnBoarding.route)
            })
        }
        bottomNavigationBuilder(navController = navController)
        composable(route = Screens.OnBoarding.route) {
            OnboardingScreen(onNextButtonClick = {
                navController.navigate(Screens.Bottom.route)
            })
        }
    }
}
}

```

```

sealed class Screens(val route: String) {
    data object Splash : Screens("splash")
    data object OnBoarding : Screens("onboarding")
    data object Bottom : Screens("bottom")
}

```

```
package com.dev.meleshko.lightenergy.navigation
```

```

import androidx.compose.ui.Modifier
import androidx.navigation.NavGraphBuilder
import androidx.navigation.NavHostController
import androidx.navigation.compose.composable
import androidx.navigation.navigation
import com.dev.meleshko.lightenergy.R

```

```
import com.dev.meleshko.lightenergy.main.ui.MenuScreen
import com.dev.meleshko.lightenergy.profile.ui.ProfileScreen
import com.dev.meleshko.lightenergy.search.ui.SearchScreen
```

```
fun NavGraphBuilder.bottomNavigationBuilder(
    modifier: Modifier = Modifier,
    navController: NavHostController
) {
    navigation(
        startDestination = BottomScreens.Main.route,
        route = Screens.Bottom.route
    ) {
        composable(route = BottomScreens.Main.route) {
            MenuScreen()
        }
        composable(route = BottomScreens.Search.route) {
            SearchScreen()
        }
        composable(route = BottomScreens.Profile.route) {
            ProfileScreen()
        }
    }
}
```

```
sealed class BottomScreens(val route: String, val name: String, val iconRes: Int) {
    data object Main :
        BottomScreens("Main", name = "Main", iconRes = R.drawable.ic_home)

    data object Search : BottomScreens("Search", name = "Search", iconRes =
R.drawable.ic_search)
    data object Profile : BottomScreens(
        "Profile",
        name = "Profile",
        iconRes = R.drawable.ic_person
    )
}
```

```
package com.dev.meleshko.lightenergy.main.ui
```

```
import android.util.Log
import android.util.Size
import android.view.MotionEvent
import androidx.compose.animation.core.animateFloatAsState
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
```

```

import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.ExperimentalComposeUiApi
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.draw.scale
import androidx.compose.ui.draw.shadow
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.input.pointer.pointerInteropFilter
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import com.dev.meleshko.lightenergy.R
import com.dev.meleshko.lightenergy.ui.theme.logo_color_container

@OptIn(ExperimentalComposeUiApi::class)
@Composable
fun ButtonComponent(onClick: () -> Unit, size: Dp = 200.dp) {
    val selected = remember { mutableStateOf(false) }
    val scale = animateFloatAsState(if (selected.value) 1.2f else 1f, label = "")

    Box(
        modifier = Modifier
            .scale(scale.value)
            .shadow(2.dp, shape = CircleShape)
            .size(size)
            .clip(CircleShape)
            .pointerInteropFilter {
                when (it.action) {
                    MotionEvent.ACTION_DOWN -> {
                        selected.value = true
                    }

                    MotionEvent.ACTION_UP -> {
                        onClick()
                        selected.value = false
                    }
                }
            }
        true
    )
    .background(logo_color_container)
    .padding(42.dp)

) {
    Image(
        painter = painterResource(id = R.drawable.bolt_thunder_svgrepo_com),

```

```

        contentDescription = null
    )
}

}

@Preview(showSystemUi = true)
@Composable
fun ButtonComponentPreview() {
    ButtonComponent({})
}

package com.dev.meleshko.lightenergy.main.ui

import android.Manifest
import android.util.Log
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.Button
import androidx.compose.material3.Icon
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import com.dev.meleshko.lightenergy.R
import com.dev.meleshko.lightenergy.ui.theme.Exo
import com.dev.meleshko.lightenergy.ui.theme.LightEnergyTheme

@Composable
fun MenuScreen(modifier: Modifier = Modifier, viewModel: MainViewModel = hiltViewModel())
{

```

```

Column(
    modifier = modifier
        .fillMaxSize()
        .padding(16.dp),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.SpaceEvenly
) {
    var text by remember {
        mutableStateOf("")
    }
    val locationPermissionRequest = rememberLauncherForActivityResult(
        ActivityResultContracts.RequestMultiplePermissions()
    ) { permissions ->
        when {
            permissions.getDefault(Manifest.permission.ACCESS_FINE_LOCATION, false) -> {
                // Precise location access granted.
                text = "Granded"
            }

            permissions.getDefault(Manifest.permission.ACCESS_COARSE_LOCATION, false) ->
        {
            // Only approximate location access granted.
            text = "Granded not all"

        }

            else -> {
                // No location access granted.
                text = "Not Granded"
            }
        }
    }
}

```

```

Text(text = "Your location is $text", style = TextStyle(
    fontWeight = FontWeight.Normal,
    fontFamily = Exo,
    fontSize = 24.sp,
    textAlign = TextAlign.Center
))

```

```

Text(
    text = "In your address no problem with electricity", style = TextStyle(
        fontWeight = FontWeight.Bold,
        fontFamily = Exo,
        fontSize = 24.sp,
        textAlign = TextAlign.Center
    )
)

```

```

    )
)

ButtonComponent(onClick = {
    locationPermissionRequest.launch(
        arrayOf(
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION
        )
    )
    text = viewModel.getRandomStreet()
    viewModel.pingRouter("192.168.0.1") { result ->
        // Виведення результату
        println(result)
        Log.d("Main", result)
    }
})

}

}

@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    LightEnergyTheme {
        MenuScreen()
    }
}

```