Одеський національний університет імені І.І.Мечникова
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних систем та технологій
(повна назва кафедри (предметної, циклової комісії)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «магістр»
(освітньо-кваліфікаційний рівень)

« Аналіз та побудова систем моніторінгу споживання електроенергії »

« Analysis and Development of an Energy Consumption Monitoring App »

Виконав: здобувач денної форми навчання
спеціальності 123 Комп'ютерна інженерія
(код, назва спеціальності)
Вєтров Олексій Олександрович
(прізвище, ім'я, по-батькові здобувача)
Керівник канд. ф-м. н., доц. Шугайло Ю.Б. _____
(науковий ступінь, вчене звання, прізвище,ініціали) (підпис)
Рецензент д.т.н., проф. Гунченко Ю.О.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№ 4 від « 18 » грудня 2023 р.

Завідувач кафедри

_____ Гунченко Юрій
(підпис) (прізвище, ім'я)

Захищено на засіданні ЕК №_____
Протокол №___від «___» _____2023 р.
Оцінка_____/ /_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ Кобозєва Алла
(підпис) (прізвище, ім'я)

Одеса - 2023

АНОТАЦІЯ

У дипломній роботі розробляється тема «Аналіз та побудова систем моніторінгу споживання електроенергії».

Мета роботи

Порівняти різні методи моніторінгу живлення приватних господарств. На основі зібраних даних провести точкове та інтервальні передбачення та обрати найстабільніший для імплементації в додаток.

В результаті досліджень і заданих вимог додатку було обрано метод оптимізації споживання енергії через інтервальне передбачення та встановлений ліміти, перевищення яких відображається сповіщенням в додатку.

Додаток має функціональний інтерфейс і оброблену базу даних із аналізом результатів тижневих замірювань споживання. Додаток обирає ліміт, в залежності від відповідей користувача та інтервального передбачення та сповіщає про його перевищення.

Дослідження та проект було реалізовано за підтримки DAAD на базі HS-Anhalt.

ABSTRACT

The topic of the thesis is "Analysis and Development of an Energy Consumption Monitoring App".

Purpose of the work

To compare different methods of monitoring the power supply of private households.

Based on the collected data, make point and interval predictions and choose the most stable one for implementation in the application.

As a result of the research and the specified requirements of the application, we chose the method of optimizing energy consumption through interval prediction and set limits, the excess of which is displayed by a notification in the application.

The app has a functional interface and a processed database with an analysis of the results of weekly consumption measurements.

The application selects a limit depending on the user's answers and interval prediction and notifies about its exceeding.

The research and project were implemented with the support of the DAAD at HS-Anhalt.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

App	Додаток
DAAD	Німецька служба академічних обмінів
NILM	Неінтрузивний моніторинг навантаження
iL (Inductor current)	Індукторний струм
iC (Capacitor current)	Струм конденсатора
ООП	Об’єктне-орієнтовне програмування
Rx	Реактивне програмування
UI	User Interface
UX	User Experience
FR	Framework
PM	Рроект Менеджер
vg (Input line voltage)	Напруга вхідної лінії
vo (Output load voltage)	Напруга на вихідному навантаженні
vc (Compensator output voltage)	Вихідна напруга компенсатора
Tc(s) (Current control loop transfer function)	Функція передачі циклу поточного керування
Ti(s) (Current control current loop transfer function)	Функція передачі потоку циклу управління струмом
Tsw (Switching period)	Період перемикання

ВСТУП

Обґрунтування актуальності дослідження

Дослідження за темою «Аналіз та побудова систем моніторінгу споживання електроенергії» концентрується на актуальній та значущій тематиці - розробці та порівнянні передових методів прогнозування електроенергії для окремих домогосподарств. Це особливо важливо для встановлення лімітів споживання енергії, що стає критично важливим у контексті зростаючої частки відновлюваних джерел енергії та необхідності підвищення гнучкості енергетичних систем.

Однією з основних проблем, яку піднімає дослідження, є висока волатильність та стохастичність навантаження в окремих домогосподарствах.

Це викликає сумніви щодо ефективності традиційних методів точкового прогнозування та вимагає пошуку нових підходів. Важливість даної роботи полягає в розвитку ймовірнісних методів прогнозування, що дозволяють не лише визначити ймовірний діапазон споживання, але й встановити більш точні та реалістичні ліміти для кожного домогосподарства.

Аналіз, представлений у дослідженні, базується на даних, зібраних під час експерименту в Кьотені у 2023 році. Він охоплює оцінку використання енергії та часового навантаження як в будні, так і в вихідні дні.

Результати дослідження мають значний потенціал для формування більш точних та ефективних стратегій розподілу енергетичних ресурсів, а також для оптимізації системи енергопостачання в умовах зростаючої частки відновлюваних джерел енергії.

В рамках дослідження було розроблено мобільний додаток Waltio, що демонструє пряме застосування вивчених методів. Цей додаток спрямований на впровадження у повсякденне життя з метою формування ефективних

стратегій розподілу енергетичних ресурсів та оптимізації систем енергопостачання.

Дослідження використовує сучасний підхід до проектування, розробки та апробації мобільного додатку, що підкреслює його актуальність та значущість.

Мета дослідження :

оптимізація гнучкості споживання електричної енергії через прогнозування навантаження на рівні окремих домогосподарств за використанням розробки мобільного додатку.

Об’єкт дослідження :

методи оптимізації гнучкості споживання електричної енергії.

Предмет дослідження : методи безпосередніх інтервальних прогнозів ймовірнісного прогнозування електричного навантаження.

Базова гіпотеза :

Розробка мобільного додатку Waltio, та його впровадження у повсякденне буття допоможе у формуванні ефективних стратегій розподілу енергетичних ресурсів та оптимізує системи енергопостачання в умовах зростаючої частки відновлюваних джерел енергії.

Метод дослідження : проектування, розробка, апробація авторського мобільного додатку Waltio

Структура кваліфікаційної роботи : вступ, три розділи, шість підрозділів, висновки та перелік використаної літератури. У роботі побудовано 19 графічних зображень (у розділах). Загальний обсяг роботи – 77 стор.

1 ТЕХНОЛОГІЇ ПОБУДОВИ МОБІЛЬНОГО ДОДАТКУ ДЛЯ ОПТИМІЗАЦІЇ СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ

1.1 Теоретичні та прикладні аспекти розробки мобільних додатків

Мобільні додатки протягом десятиліття перетворилися з простих інструментів а невід'ємну частину щоденного життя.

З появою мобільних телефонів, які мали обмежені функції, до сучасних смартфонів, здатних виконувати різноманітні завдання, мобільні додатки пройшли довгий шлях розвитку. Ця еволюція спричинила глобальні зміни в способах спілкування, роботі, розвагах та багатьох інших сферах життя.

Впровадження App Store від Apple у 2008 році та Google Play Store (колишній Android Market) стало віхою у світі мобільних додатків. Ці платформи дозволили розробникам надавати свої додатки широкому загалу, відкриваючи нові можливості для інновацій та підприємництва. Різноманітність та доступність додатків зростали неймовірно швидко, охоплюючи кожну сферу від освіти до фінансів, від здоров'я до розваг. [38]

Розробка мобільних додатків є відповіддю на зростаючий попит сучасного суспільства на інноваційні, зручні та ефективні технологічні рішення. У світі, де переважна більшість населення використовує смартфони як щоденні інструменти для роботи, навчання, розваг та спілкування, мобільні додатки стають невід'ємною частиною повсякденного життя.

Компанії розробляють додатки, щоб вийти на широкий ринок потенційних клієнтів, використовуючи можливості, які надають мобільні платформи. Це включає в себе не тільки прямий продаж продуктів або послуг через додатки, але й створення брендового досвіду, підвищення пізнаванності бренду, залучення нових клієнтів та підтримання відносин із існуючими.

З іншого боку, мобільні додатки слугують потужним інструментом для рішення конкретних соціальних, освітніх та екологічних проблем.

Вони використовуються для надання важливої інформації, сприяння освітнім ініціативам, забезпечення доступу до охорони здоров'я та підтримки сталого розвитку.

В умовах швидкого технологічного розвитку, мобільні додатки є засобом інновацій, відкриваючи нові можливості для бізнесу та технологічних проривів. Вони дозволяють компаніям вводити передові технології, такі як штучний інтелект, машинне навчання та великі дані, для покращення взаємодії з клієнтами та оптимізації процесів.

Висновок : розробка мобільних додатків сьогодні є ключовим елементом в стратегії бізнесу чи організації, яка прагне залишатися актуальною, інноваційною та конкурентоспроможною в швидко змінюваному цифровому світі.

Створення мобільних додатків є складним і багатогранним процесом, починаючи від концептуалізації ідеї. Після визначення його цільової аудиторії, ключових функцій, а проведення ринкового аналізу і дослідження конкурентів, аби зрозуміти, як додаток буде вирізнятися на ринку.

Після визначення концепції, розробляється детальний план проекту, що включає тайм лайни, ресурси та вимоги.

На етапі проектування UX/UI, який вимагає від дизайнерів глибокого розуміння потреб і вподобань користувачів, а технічних можливостей платформи, для якої вони розробляють. Починаючи з дослідження та аналізу цільової аудиторії, процес включає в себе вивчення поведінкових моделей та вподобань користувачів. Це допомагає зрозуміти, як користувачі будуть взаємодіяти з додатком, та які функціональні можливості їм необхідні. [8]

Після збору інформації та визначення ключових вимог, наступає етап створення структури та логіки додатку. Це означає розробку карти сайту для веб-сайтів або архітектури додатку для мобільних програм. Цей етап є

критичним для визначення, як різні елементи і функції будуть організовані та як вони взаємодіятимуть між собою.

Далі, дизайнери розробляють провідники (wireframes) – базові каркаси сторінок, які візуалізують основні компоненти інтерфейсу (див.: рис.1.1) [10].

Провідники допомагають зосередитися на логікі розташування елементів, перш ніж переходити до візуального дизайну.

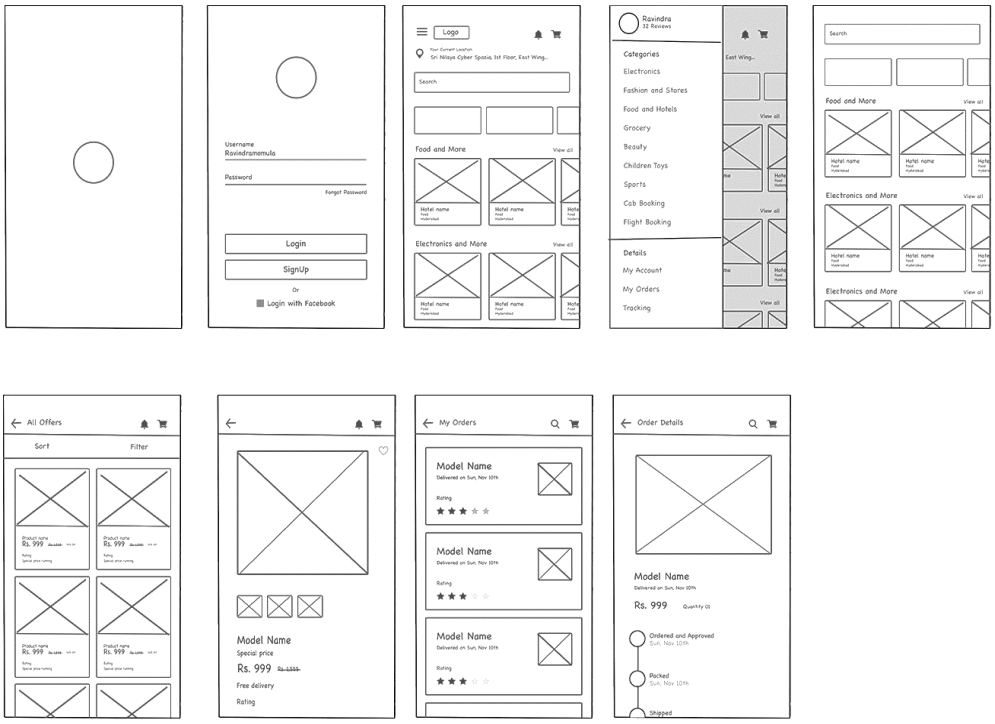

Рисунок 1.1. Проектування додатку

Після цього починається розробка власне інтерфейсу користувача (UI), де створюються детальні дизайні елементів інтерфейсу – кнопок, іконок, меню тощо. Цей етап включає вибір кольорової палітри, шрифтів та інших візуальних елементів, що формують загальний вигляд та стиль додатку.

Паралельно з UI, надзвичайно важливим є забезпечення високоякісного користувацького досвіду (UX). Це означає створення інтуїтивно зрозумілого та зручного використання інтерфейсу, який забезпечує легкість навігації та

доступу до необхідних функцій. Підхід до проектування UI/UX має прямий зв'язок з типом додатку, який розробляється, оскільки кожен тип додатку має свої унікальні вимоги та обмеження. [6]

Коли мова йде про різні типи додатків, такі як нативні, крос-платформні, гібридні або прогресивні веб-додатки, кожен з них вимагає специфічного підходу до UI/UX дизайну.

У нативних додатках дизайнери мають зосередитися на створенні інтерфейсів, що використовують можливості конкретної платформи та апаратного забезпечення.

У крос-платформних додатках слід забезпечити узгодженість та сумісність UI/UX на різних пристроях.

Гібридні та прогресивні веб-додатки вимагають знаходження балансу між веб- та мобільними стандартами дизайну, щоб забезпечити зручність та доступність для широкого спектру користувачів.

Висновок : вибір типу додатку має значення для визначення стратегії UI/UX дизайну, що в свою чергу впливає на загальний успіх та прийняття додатку користувачами.

Кожен тип додатку пропонує унікальні можливості та виклики для дизайнерів UI/UX, вимагаючи від них гнучкості, креативності та глибокого розуміння потреб цільової аудиторії.

Дослідники відрізняють наступні типи додатків.

Нативні додатки - розробляються спеціально для однієї платформи, як-от iOS або Android, використовуючи мови програмування та інструменти, призначені для цієї платформи. Для iOS, це Swift або Objective-C, а для Android - Kotlin або Java.

Нативні додатки можуть максимально використовувати можливості апаратного забезпечення пристрою, такі як камера, GPS, акселерометр, що дозволяє розробникам створювати високо функціональні додатки, оптимізовані під конкретну платформу.

Прикладом нативного додатку для iOS може бути “Photos” від Apple, який розроблений на Swift та має доступ до всіх функцій фотографії та відео, які пропонує iOS. Для Android, прикладом може бути “Google Maps”, розроблений на Kotlin, що забезпечує плавну інтеграцію з GPS та іншими функціями Android. [17]

Ці додатки використовують рідні бібліотеки та API платформ, як-от UIKit для iOS або Android SDK для Android, для створення інтуїтивно зрозумілих та реактивних інтерфейсів.

Однією з переваг нативних додатків є їх висока продуктивність. Вони швидко завантажуються та виконуються завдяки прямому доступу до апаратного забезпечення та оптимізованій процесорній потужності. Крім того, нативні додатки забезпечують кращу безпеку, оскільки вони мають доступ до різних рівнів безпеки, які пропонуються операційними системами. вони пропонують кращий користувацький досвід, оскільки повністю інтегровані з екосистемою платформи, включаючи дизайн, жести та загальну поведінку.

Розробка нативних додатків має свої недоліки. Вона може бути дорожчою та час витратнішою, оскільки потрібно розробляти окремі версії для кожної платформи. Це вимагає від команди розробників володіти набором різних навичок та знань специфіки кожної платформи. оновлення та підтримка додатків може бути складнішою, оскільки зміни треба впроваджувати окремо для кожної платформи. [2]

Висновок : нативні додатки є ідеальним вибором для проектів, де важливі висока продуктивність, специфічні функції платформи та найкращий користувацький досвід. Однак, потрібно враховувати вищу вартість та зусилля, необхідні для розробки та підтримки таких додатків.

Крос-платформні додатки є сучасним рішенням в мобільній розробці, оскільки вони дозволяють розробникам використовувати один і той же код для створення додатків, що працюють на різних операційних системах, таких як iOS і Android.

Цей підхід використовує фреймворки, такі як React Native, Flutter від Google, або Xamarin від Microsoft, які дозволяють програмістам писати код на мовах, таких як JavaScript (React Native), Dart (Flutter) або C# (Xamarin), та потім компілювати його в рідний код для конкретної платформи.

Додаток, створений на Flutter, може виглядати та функціонувати майже ідентично на обох Android та iOS пристроях, забезпечуючи однорідний досвід для користувачів на обох платформах.

Flutter використовує власні віджети для забезпечення високої продуктивності та гнучкості в дизайні. React Native, з іншого боку, дозволяє інтегрувати рідні модулі, що може бути корисно для використання специфічних функцій платформи.

Однією з головних переваг крос-платформних додатків є економія часу та ресурсів. Оскільки код пишеться один раз і запускається на кількох платформах, це може значно знизити витрати на розробку та підтримку. Крім того, це спрощує управління кодом, оскільки вносяться зміни лише в одному репозиторію коду. [5]

Крос-платформні додатки мають недоліки.

По-перше, хоча вони і забезпечують відносно однаковий досвід на різних пристроях, іноді вони не можуть повною мірою використовувати унікальні функції та оптимізації конкретних платформ. Це може призводити до того, що додатки працюють повільніше, ніж їхні нативні аналоги, особливо в сценаріях, що вимагають інтенсивного використання ресурсів апаратного забезпечення.

По-друге, залежність від крос-платформних фреймворків та інструментів може бути ризикованою, оскільки ці технології швидко розвиваються і іноді можуть залишати за собою застарілі підходи або технічні борги.

Крім того, необхідність використання сторонніх плагінів для доступу до рідних функцій платформи може ускладнити розробку та підтримку.

Висновок : крос-платформні додатки пропонують значні переваги з точки зору швидкості розробки та зниження витрат, їх використання може бути обмежене у випадках, коли потрібна висока продуктивність, глибока інтеграція з платформою або використання специфічних функцій апаратного забезпечення. Рішення про вибір між крос-платформними та нативними додатками повинно базуватися на конкретних вимогах та цілях проекту.

Веб-додатки є програмами, які використовуються через інтернет-браузер і не вимагають завантаження та установки на пристрій користувача.

Вони розробляються з використанням веб-технологій, таких як HTML, CSS і JavaScript, та можуть бути доступні на будь-якому пристрої з інтернет-браузером. Такий підхід до розробки дозволяє веб-додаткам бути платформонезалежними, забезпечуючи користувачам доступ до додатку з різних пристроїв, включаючи настільні комп'ютери, ноутбуки, планшети та смартфони.

Одним з відомих прикладів веб-додатку є Google Docs, який дозволяє користувачам створювати, редагувати та спільно використовувати документи онлайн. Веб-додатки використовують веб-фреймворки та бібліотеки, такі як Angular, React або Vue.js, для створення інтерактивних та динамічних інтерфейсів користувача.

Переваги веб-додатків включають їх універсальність та легкість доступу. Користувачам не потрібно встановлювати додаток, що знижує бар'єр входу та спрощує оновлення. Веб-додатки можуть бути розроблені швидше та дешевше, оскільки вони вимагають однієї бази коду, яка працює на різних платформах.

Веб-додатки залежать від інтернет-з'єднання та можуть мати обмежений доступ до функцій апаратного забезпечення пристрою. Хоча деякі сучасні технології, такі як Progressive Web Apps (PWA), дозволяють веб-додаткам працювати офлайн та надають доступ до деяких функцій апаратного

забезпечення, вони все ще не можуть повністю конкурувати з можливостями нативних додатків.

Питання безпеки є важливими, оскільки веб-додатки взаємодіють із серверами через інтернет, що може створювати додаткові ризики. Веб-додатки можуть страждати від недоліків у продуктивності, особливо при інтенсивному використанні графіки або обробці даних.

Висновок : веб-додатки є ефективним вибором для проектів, які потребують широкої доступності, і сумісності з різними пристроями та операційними системами. Однак, для додатків, які потребують глибокої інтеграції з апаратним забезпеченням або високої продуктивності, нативні або крос-платформні рішення можуть бути більш підходящими.

Побудуємо схему підбору типу мобільного додатку, враховуючи всі фактори (див. : рис.1.2.)

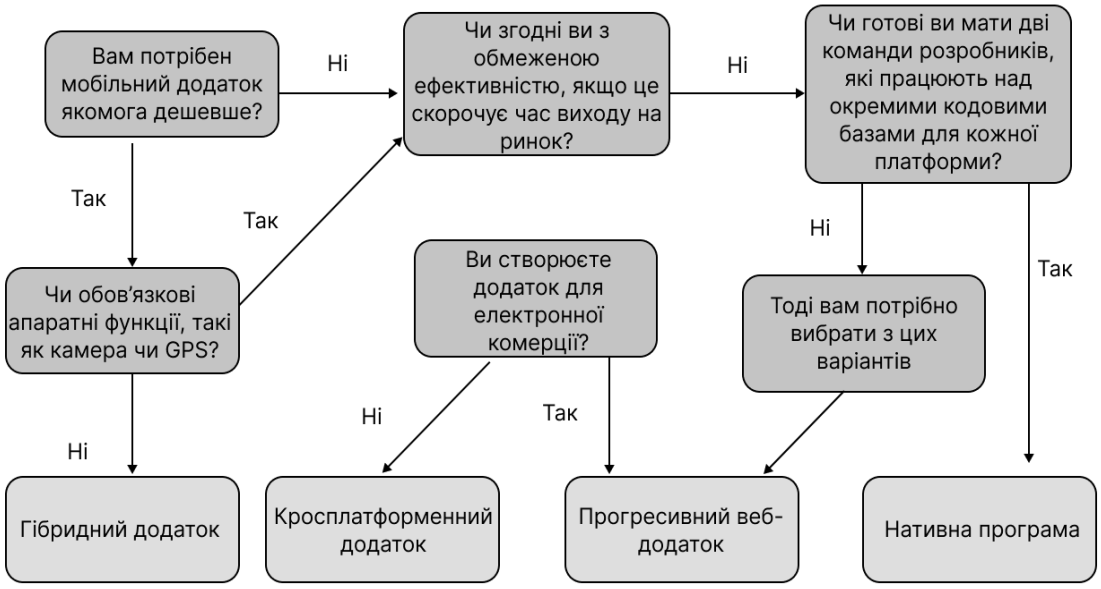

Рисунок 1.2 - Алгоритм вибору типу додатку

Для розробки проекту було обрано крос-платформний підхід, використовуючи Flutter та Dart.

Під час оцінювання можливих технологій, ми ретельно розглядали такі альтернативи, як React Native від Meta, Xamarin від Microsoft та Apache Cordova.

Кожна з цих технологій має свої сильні сторони, але для цілей кваліфікаційного дослідження Flutter був вибраний з наступних ключових аргументів.

Flutter, розроблений Google, використовує мову програмування Dart, яка забезпечує високу продуктивність та ефективне компілювання в нативний код.

На відміну від React Native, який використовує JavaScript, Flutter пропонує більш однорідний досвід розробки для обох основних мобільних платформ (iOS та Android), а забезпечує високу продуктивність завдяки своїй рендеринговій системі, заснованій на мові C++.

У порівнянні з Xamarin, який дозволяє розробляти крос-платформні додатки на C#, Flutter пропонує більшу гнучкість та інноваційний набір віджетів для UI. Xamarin, хоча і має сильну підтримку від Microsoft та велику спільноту, іноді може виявлятися обтяжливим у використанні для динамічного та складного інтерфейсу користувача.

Apache Cordova, який дозволяє розробляти додатки, використовуючи HTML, CSS та JavaScript, був відкинутий через його обмеження в продуктивності та виклики з інтеграцією з нативними функціями. Cordova є корисним для простих веб-додатків, він не забезпечує того рівня продуктивності та глибокої інтеграції з апаратним забезпеченням, який може забезпечити Flutter. [32]

Flutter вирізняється своєю здатністю до швидкого гарячого перезавантаження (hot reload), що дозволяє розробникам бачити зміни в реальному часі без необхідності повного перезапуску додатку. Ця функція значно спрощує процес розробки та тестування, зменшуючи час, необхідний для ітерацій.

Вибір Flutter для розробки крос-платформного додатку виявився вдалим рішенням, оскільки він забезпечує оптимальне співвідношення продуктивності, гнучкості та ефективності розробки.

Такий підхід дозволяє створювати високоякісні додатки, які пропонують однаковий користувацький досвід на різних мобільних платформах, відповідаючи сучасним вимогам ринку та очікуванням користувачів.

Використання Flutter як обраної платформи для розробки крос-платформного додатку, визначає роль мови програмування Dart, яка є ключовою складовою цього фреймворку. Dart, розроблений Google, є об'єктно-орієнтованою мовою програмування, оптимізованою для розробки мобільних та веб-додатків.

Переваги та недоліки мови програмування Dart в контексті її практичності, доступності ресурсів, продуктивності та впливу на ринок праці.

1. Легкість навчання : програмісти JavaScript, використовують Dart : оскільки основні принципи цих мов досить схожі. Це робить Dart раціональним вибором для широкої аудиторії розробників.

2. Доступність документації : завдяки підтримці Google, мова Dart має документовані можливості та особливості, що дозволяє знаходити відповіді на питання, які виникають під час навчання або кодування.

3. Висока продуктивність : програми, написані на Dart, виконуються швидше, ніж аналогічні програми на JavaScript, що робить Dart більш ефективним для певних типів додатків.

4. Стабільність та підтримка ООП : Dart є стабільною мовою програмування, яка підтримує об'єктно-орієнтоване програмування, включаючи наслідування, інтерфейси та опціональне типізування, ідеально підходячи для розробки великих та складних додатків.

5. AOT та JIT компіляція : Dart має здатність до обох типів компіляції : Ahead of Time (AOT) та Just in Time (JIT). AOT дозволяє конвертувати код Dart

безпосередньо в машинний код, в той час як JIT сприяє швидким циклам розробки.

1. Future та Async/Await : Використання “Future “ у Dart для представлення результату асинхронної операції. Синтаксис “async “ та “await “ полегшує читання та написання асинхронного коду, дозволяючи обробляти асинхронні відповіді в синхронному стилі.

2. Stream API : “Stream “ в Dart дозволяє обробляти асинхронні події, що відбуваються декілька разів. Це особливо корисно для обробки неперервних даних, наприклад, реального споживання електроенергії.

3. Контроль Над Потоками : Використання таких інструментів, як “StreamController “, для контролю над потоками даних, включаючи можливість паузи, відновлення та скасування подій.

4. Обробка Помилок : Впровадження механізмів обробки помилок в асинхронних потоках для забезпечення стабільності та надійності додатку.

5. Комбінування та Трансформація Потоків : Використання операцій, таких як “map “, “where “, “combineLatest “, для трансформації та комбінування декількох потоків даних, ефективно об'єднуючи інформацію з різних джерел.

Rx Dart є реалізацією бібліотеки Reactive Extensions для Dart, яка надає додаткові можливості для створення та управління реактивними потоками даних.

1. Основи rx dart : впровадження основних концепцій rx, таких як “observables“ (або “streams“), “subscribers“ (слухачі) та “operators“ (оператори).

2. Створення реактивних потоків : використання “observables“ для створення реактивних потоків, що динамічно реагують на зміни в даних, наприклад, на зміни у споживанні електроенергії.

3. Оператори трансформації : застосування операторів, таких як “map“, “filter“, “debounce“, “throttle“, для ефективного перетворення та контролю над потоками даних.

4. Управління підписками : контроль над підписками на потоки, включаючи їх активацію, паузу та скасування, щоб уникнути витоку пам'яті та інших проблем.

5. Комбінування потоків : використання таких конструкцій, як “combinelatest“ та “zip“, для ефективного поєднання даних з різних джерел.

6. Розширене управління помилками : впровадження складних механізмів обробки помилок в рамках реактивних потоків.

Впровадження цих технік дозволило створити ефективний, надійний та реактивний монітор електроенергії, що забезпечує точне відслідковування та аналіз споживання енергії.

Недоліки Dart :

1. Обмеженість ресурсів : Dart страждає від відсутності великої та згуртованої спільноти розробників, що ускладнює пошук рішень для конкретних проблем.

2. Низька поширеність на ринку : Dart є відносно новою мовою для багатьох програмістів і рідко використовується в індустрії, що може ускладнити пошук роботи для спеціалістів з Dart, оскільки мова ще не отримала розповсюдження.

3. Розвиток мови : Dart все ще перебуває в процесі розвитку, що, з одного боку, є позитивним, але з іншого – призводить до можливих змін у API та нестачі документації.

4. Відсутність нативної підтримки в браузерах : для демонстрації можливостей Dart його розробникам доводиться забезпечувати нативну

підтримку в усіх популярних браузерах, що вимагає додаткових зусиль та ресурсів.

Частково це спрощує Firebase, як хмарна платформа від Google, надає широкий спектр сервісів та функціоналів, які ідеально підходять для розробки мобільних додатків. У розробці додатку Waltio для моніторингу та оптимізації споживання електроенергії, Firebase використовувався для створення бази даних, авторизації користувачів, а обробки різних подій

Firebase надає два основних рішення для баз даних : Realtime Database та Cloud Firestore. Обидва ці рішення дозволяють зберігати та синхронізувати дані між користувачами в режимі реального часу.

- Realtime Database пропонує просту інтеграцію та високу швидкість синхронізації, ідеально підходить для додатків з високими вимогами до часу відгуку.

- Cloud Firestore надає більш гнучкі можливості запитів та індексації, а масштабується краще, що робить його вибором для більших та складніших додатків.

В дипломному дослідженні Firestore застосовано через його гнучкість у структуруванні даних та здатність ефективно обробляти складні запити, що важливо для аналізу великих обсягів даних про споживання електроенергії.

Firebase Authentication надає просте та безпечне рішення для управління користувачами. Воно підтримує різні способи входу, включаючи електронну пошту та пароль, вхід через соціальні мережі, та анонімний вхід.

Це дозволяє забезпечити гнучкий доступ до додатку, зберігаючи безпеку та конфіденційність персональних даних користувачів.

Firebase надає інструменти для збору аналітики та обробки подій в додатку.

Firebase Analytics дозволяє моніторинг відомостей про поведінку користувачів та використання додатку, що є критично важливим для оптимізації функціональності та інтерфейсу.

JSON (JavaScript Object Notation) є основним форматом для обміну даними у Firebase. Знання того, як працювати з JSON, є ключовим для ефективного використання Firebase, оскільки це дозволяє :

- структурувати дані : ефективне структурування даних у форматі JSON впливає на швидкість запитів та відгуків у Firebase.
- імпорт/експорт даних : можливість масштабування та міграції даних між сервісами.

  обробка даних : розуміння JSON дозволяє розробникам краще керувати даними, отримувати з них інформацію та виконувати складні запити.

Висновок : використання Firebase у додатку Waltio для оптимізації споживання електроенергії дозволило розробити стабільну, безпечну, і гнучку платформу.

Інтеграція Firebase спростила процес розробки, забезпечила потужними інструментами для аналізу даних та управління користувацькими взаємодіями.

Інтеграція Waltio з Firebase

У рамках проекту мобільний додаток “Waltio” інтегровано з Firebase, щоб забезпечити надійність, безпеку та ефективність управління даними з метою авторизації користувачів, обробки подій та аналітики.

На рисунку 1.3. представлені основні залежності, включені в наш “pubspec.yaml “ файл, які відображають інтеграцію з Firebase.

Ці залежності вказують на версії бібліотек, які використано для інтеграції додатку з сервісами Firebase

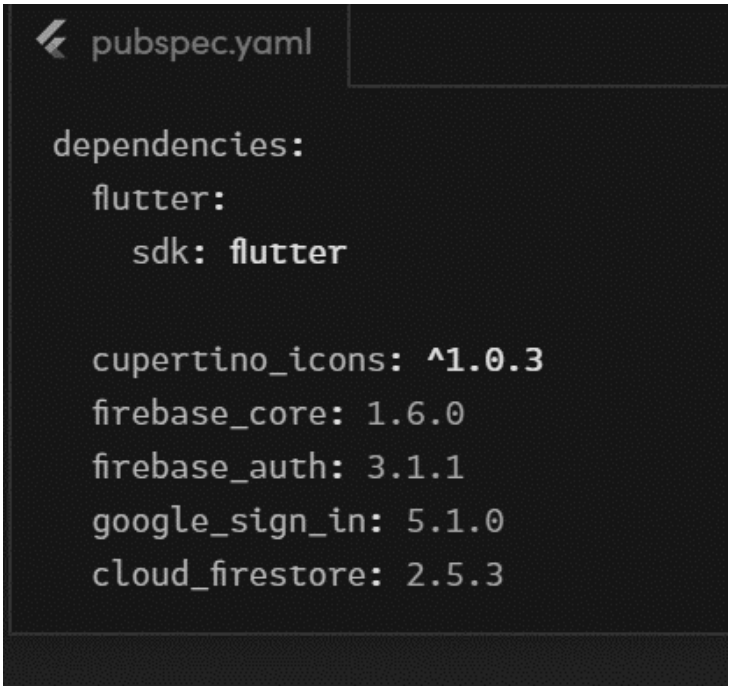

Рисунок 1.3 - Основні залежності інтеграції Waltio з Firebase, включені в “pubspec.yaml “ файл

Зокрема, “firebase_core “ є основою для всіх інших пакетів Firebase, “firebase_auth “ використовується для авторизації користувачів, “google_sign_in “ забезпечує інтеграцію з Google для входу в систему, а “cloud_firestore “ є нашим рішенням для бази даних.

Використання цих бібліотек дозволило нам створити додаток, який може взаємодіяти з хмарними сервісами, забезпечує швидкий доступ до даних та їх синхронізацію в реальному часі, а пропонує різні варіанти авторизації для користувачів.

У додатку Waltio використано Cloud Firestore в якості бази даних для зберігання та синхронізації даних в реальному часі.

На зображенні представлено, як в Cloud Firestore створюється та організовується колекція “energy_usage_records “, де кожен документ відповідає запису про споживання енергії від окремого користувача (див. :рис. 1. 4).

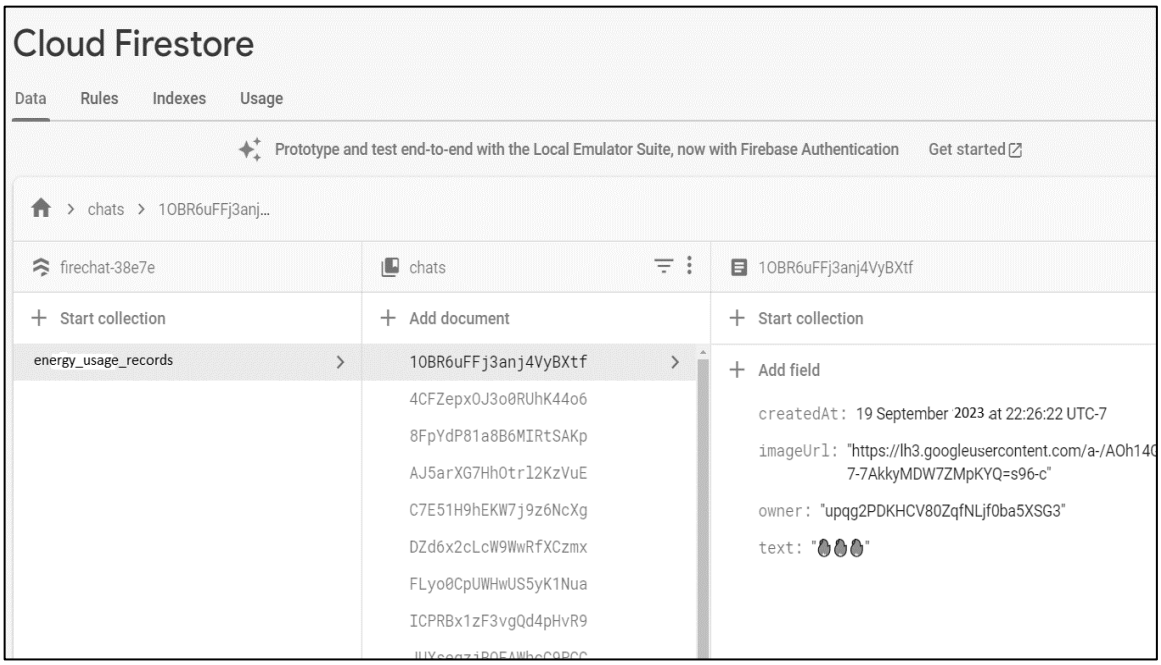

Рисунок 1.4 - Cloud Firestore у створенні колекції «energy_usage_records»

Налаштування безпеки

У контексті додатку, аналогічна структура може бути використана для зберігання даних про споживання електроенергії :

- “createdAt “ фіксує час здійснення виміру.

- “userId “ “ вказує на користувача, який зробив запис.

Використання Cloud Firestore дозволяє нам не тільки ефективно управляти даними споживання енергії, але й масштабувати додаток Waltio, забезпечуючи високу швидкість відгуку та зручність для користувачів при відстеженні їх енергетичного відбитку.

Ця інтеграція є ключовим компонентом додатку Waltio, що дозволяє не тільки оптимізувати процес розробки, але й забезпечити масштабованість та гнучкість сервісу для моніторингу та управління споживанням електроенергії.

1.2 Проектування додатку для моніторінгу споживання електроенергії

Для розробки системи моніторінгу та оптимизації енергоспоживання, розглянемо існуючі системи.

Hive - це система “розумного будинку”, що пропонує спектр пристроїв, включаючи термостати, розумні вимикачі, камери безпеки та ін.

Користувачі можуть контролювати ці пристрої через мобільний додаток або веб-інтерфейс, що надає можливість дистанційного керування домашнім опаленням, освітленням та безпекою. Hive дозволяє користувачам створювати гнучкі графіки та автоматизувати побутові процеси для підвищення комфорту та ефективності енергоспоживання.

Nest, який є частиною екосистеми Google, пропонує «розумні термостати», детектори диму та інші пристрої для «розумного будинку». Зокрема, їх термостати мають функцію навчання поведінки користувачів та автоматичної оптимізації налаштувань для економії енергії.

Додаток Nest надає детальний звіт про споживання енергії та рекомендації для зменшення витрат.

Аналізуючі ці системи виділимо основні труднощі цих додатків - встановлення фізичного обладнання.

На відміну від Hive і Nest, новий додаток Waltio зосереджений на програмному рішенні без необхідності фізичного встановлення додаткового обладнання. Він аналізує дані з “розумних лічильників” і забезпечує користувачів інструментами для моніторингу та оптимізації енергоспоживання.

Основні можливості включають :

- Авторизацію та профілі : Створення індивідуальних профілів з використанням Firebase.

- Локалізація : Підтримка української та німецької мов.

- Аналіз та прогнози : Алгоритми для аналізу відповідей користувачів та прогнозування споживання, в залежності від цих відповідей.

- Моніторинг : Інструменти для відстеження поточного споживання та рекомендацій з економії енергії.

- Сповіщення : Попередження про перевищення лімітів споживання.

- Інтеграція з обладнанням : Hive та Nest вимагають встановлення спеціалізованого обладнання, в той час як новий додаток Waltio результати цього дослідження і потребуватиме такої функції лише в майбутньому.

- Споживацький досвід : Хоча Hive та Nest пропонують широкий спектр автоматизованих функцій, новий зосереджений на простоті та зручності користувачів, які шукають менш складні рішення.

Покажемо взаємодію користувача із додатком :

На наступній схемі (рис. 1.5) представлений процес взаємодії користувача з мобільним додатком і сервером. Це виглядає як діаграма послідовностей, яка використовується для візуалізації логіки обміну даними між різними компонентами системи.

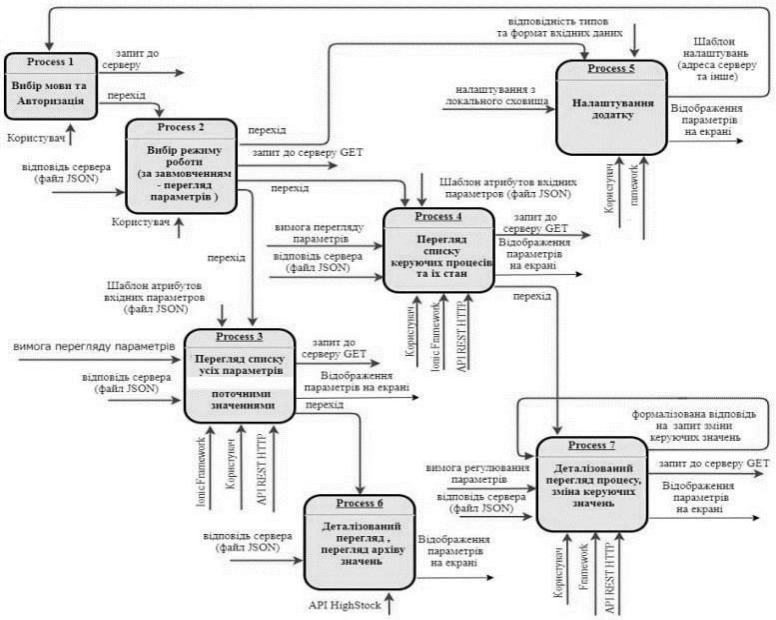

Рисунок 1.5 - Схема процесів додатку

Основні етапи процесів, зображених на схемі(див. рис. 5) :

1. Завантаження додатку :

- Користувач завантажує додаток і відкриває його.

2. Авторизація :

- Користувач вводить дані для входу.

- Дані відсилаються на сервер і здійснюється запит на авторизацію.

- Сервер повертає відповідь, чи була авторизація успішною.

3. Перегляд сторінок :

- Користувач переглядає різні сторінки додатку.

- Для цього він вибирає сторінку, і дані для відображення цієї сторінки запитуються з сервера.

4. Налаштування :

- Користувач входить у розділ налаштувань, де може змінити певні параметри.

- Змінені налаштування відправляються на сервер, де вони оновлюються та зберігаються.

5. Вихід :

- Коли користувач завершує сесію, він може вийти з додатку, і дані про вихід відправляються на сервер.

Ця схема демонструє типову взаємодію в рамках клієнт-серверної архітектури, де додаток (клієнт) взаємодіє з сервером через API для виконання різних функцій, таких як авторизація, перегляд контенту, налаштування та інших дій (рис.1.6).

Використання API дозволяє абстрагувати логіку обробки даних на сервері, забезпечуючи кращу безпеку та гнучкість системи.

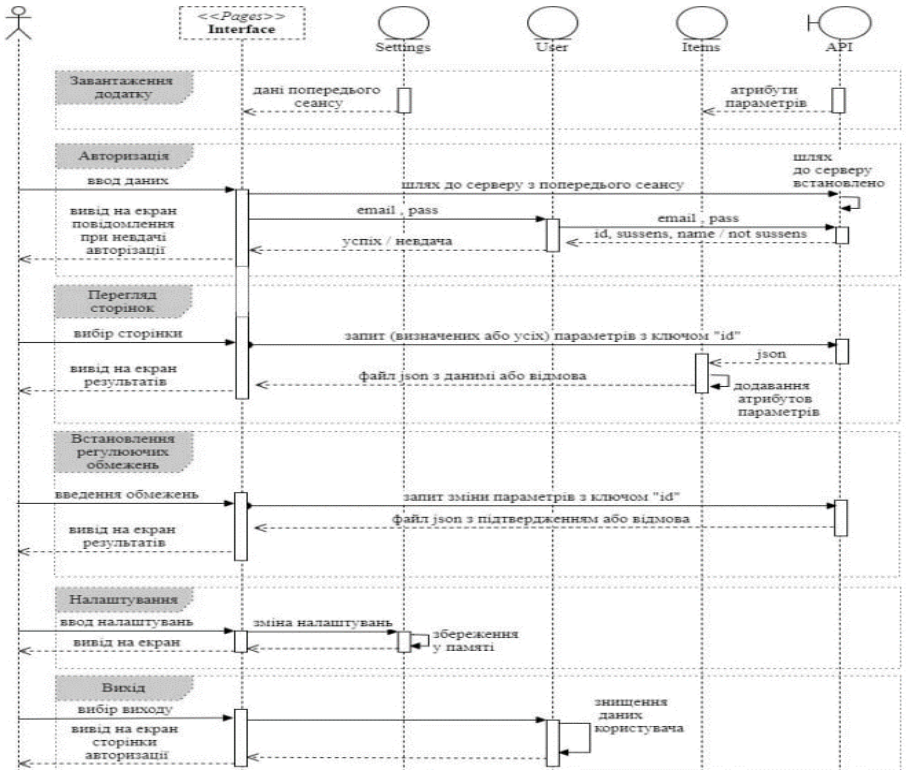

Рисунок 1.6 - Схема типової взаємодії в рамках клієнт-серверної архітектури

Дизайн додатку є комплексною роботою, яка включає в себе огляд ринку та його трендів, а навички роботи із різними редакторами, такими як Figma, Photoshop (див.: рис.1.7).

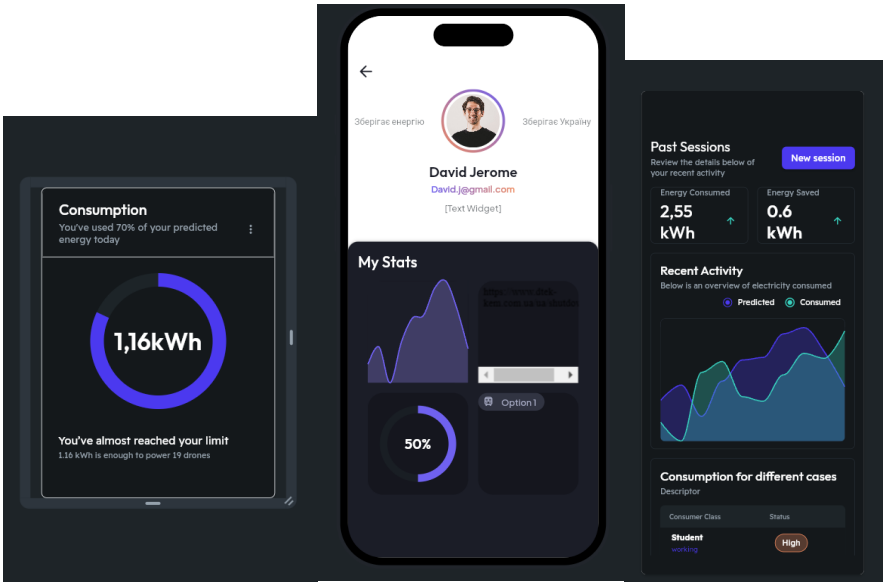

Рисунок 1.7 - Дизайн сторінок споживання, профілю та аналізу

Для розробки додатку Waltio особливу увагу було звернуто на ринки Німеччини та України, враховуючи їх культурні та лінгвістичні особливості.

Вибір на користь мінімалістичного дизайну було зроблено не випадково : це дозволяє створити інтерфейс, який не тільки виглядає сучасно, але й забезпечує зручність та ефективність користувацького досвіду (UX).

Стосовно програмної частини, розробка додатку ведеться на платформі Flutter використовуючи мову програмування Dart. Flutter було обрано через його високу продуктивність, гнучкість та здатність створювати крос-платформні додатки, які однаково добре функціонують на різних операційних системах. Використання Flutter сприяє швидкій та зручній роботі з дизайном, дозволяючи імплементувати складні анімації та інтерактивні елементи.

У контексті програмної реалізації, ключові аспекти дизайну, такі як інтуїтивна навігація, адаптивність, локалізація, та інтерактивність, втілюються за допомогою специфічних віджетів Flutter. Для навігації використовуються Drawer і BottomNavigationBar, для адаптивності - MediaQuery та Flexible, для локалізації - Localizations та Intl, а для анімацій і інтерактивності - AnimationController та GestureDetector.

Окрему увагу приділено відображенню даних та їх візуалізації, що особливо важливо для функцій, пов'язаних з аналізом споживання енергії. Для цього використовуються спеціалізовані бібліотеки, такі як fl_chart.

Кожен Flutter проект має файл pubspec.yaml, в якому зберігаються дані про всі бібліотеки в проекті.

Базовий список бібліотек, що застосовані у додатку “Waltio” :

1. firebase_core (1.6.0) - Основний пакет для використання Firebase в Flutter додатку.

2. firebase_auth (3.1.1) - Пакет для імплементації системи авторизації через Firebase.

3. google_sign_in (5.1.0) - пакет для авторизації користувачів через Google.

4. cloud_firestore (2.5.3) - пакет для взаємодії з базою даних Cloud Firestore.

5. cupertino_icons (1.0.3) - набір іконок в стилі Apple’s Cupertino, який використовується у дизайні додатку.

6. flutter_localizations - підтримка локалізації та багатомовності в додатку.

7. provider (version) - пакет для управління станом в додатку, який допомагає з організацією бізнес-логіки.

8. http (version) - пакет для виконання HTTP запитів, якщо додаток Waltiовзаємодіє з веб-сервісами.

9. shared_preferences (version) - пакет для збереження простих даних користувача локально на пристрої.

10. flutter_svg (version) - пакет для відображення SVG зображень.

11. intl (version) - пакет для локалізації та форматування дат, чисел та ін.

12. charts_flutter (version) - бібліотека для візуалізації даних за допомогою графіків та діаграм.

13. flutter_bloc (version) - пакет для управління станом з використанням патерну BLoC (Business Logic Component).

14. firebase_messaging (version) - пакет для роботи з push-повідомленнями через Firebase Cloud Messaging.

15. firebase_storage (version) - пакет для зберігання файлів у Firebase Cloud Storage.

16. flutter_secure_storage (version) - Пакет для зберігання конфіденційних даних в безпечному сховищі.

17. path_provider (version) - пакет для знаходження правильного шляху до файлової системи пристрою.

- flutter_lints (2.0.1) - набір рекомендованих правил лінтера для Flutter.

- build_runner (2.1.11) - інструмент для генерації коду в проектах Flutter та Dart.
- json_serializable (6.2.0) - бібліотека для генерації коду для серіалізації та десеріалізації JSON.

Висновок : бібліотеки визначають функціональний набір інструментів для розробки додатку, зі спеціалізованими рішеннями для авторизації, роботи з даними та інтерфейсом користувача.

2 ІМОВІРНІСНИЙ ПРОГНОЗ НАВАНТАЖЕННЯ ДЛЯ ІНДИВІДУАЛЬНОГО СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ ТА СПОСОБИ ЙОГО ПІДТВЕРДЖЕННЯ В ДОДАТКУ

2.1 Прогнозування ймовірного навантаження на день для індивідуального споживання електроенергії за допомогою інтервальних методів

Актуальність дослідження

Це дослідження зосереджується на розробці та порівнянні передових методів ймовірнісного прогнозування навантаження на електроенергію для окремих домогосподарств, що має важливе значення для встановлення лімітів споживання енергії.

У контексті зростаючої частки відновлюваних джерел енергії та необхідності підвищення гнучкості енергетичних систем, точне прогнозування навантаження стає ключовим аспектом ефективного управління енергоресурсами.

Висока волатильність та стохастичність навантаження в окремих домогосподарствах ставить під сумнів ефективність традиційних методів точкового прогнозування, що спонукає до пошуку нових підходів.

Дослідження розглядає розвиток ймовірнісних методів прогнозування, що дозволяють не тільки визначити ймовірний діапазон споживання, але й дають змогу встановити більш точні та реалістичні ліміти для кожного домогосподарства.

Це має суттєве значення для інтеграції гнучких механізмів управління навантаженням в рамках виробництва енергії з відновлюваних джерел.

Аналіз базується на даних, зібраних під час експерименту в Кьотені 2023 року, та включає оцінку використання енергії та часового навантаження в будні та вихідні дні.

Результати дослідження можуть бути використані для формування більш точних та ефективних стратегій розподілу енергетичних ресурсів, а для оптимізації системи енергопостачання в умовах зростаючої частки відновлюваних джерел енергії.

Встановлення вдосконалених лімітів споживання енергії на основі даних прогнозування може сприяти більш ефективному та збалансованому використанню енергоресурсів у масштабах окремих домогосподарств.

Зростаюча частка відновлюваних джерел енергії у виробництві електроенергії вимагає підвищеної гнучкості в енергетичних системах. Точне прогнозування навантаження на рівні окремих домогосподарств є критично важливим для управління цією гнучкістю.

Мета дослідження :

оптимізація гнучкості споживання електричної енергії через прогнозування навантаження на рівні окремих домогосподарств за використанням розробки мобільного додатку.

Об’єкт дослідження :

методи оптимізації гнучкості споживання електричної енергії через передбачені ліміти.

Предмет дослідження : методи безпосередніх інтервальних прогнозів ймовірнісного прогнозування електричного навантаження.

Впровадження дослідження :

Україна та Німеччина разом із 194 країнами поставила перед собою мету зберігати “зростання глобальної середньої температури значно нижче 2°C вище докомп'ютерних рівнів, і докладати зусиль до обмеження зростання температури до 1.5°C вище докомп'ютерних рівнів, визнаючи, що це значно зменшить ризики та впливи зміни клімату” [1].

Це завдання передбачає збільшення встановленої чистої потужності відновлюваних нестабільних джерел енергії.

Щоб впоратися з розбіжностями між попитом і пропозицією, гнучкість стає ключовою.

Майбутній житловий сектор з відновувальними носіями та електромобілями може забезпечити частину необхідної гнучкості.

Для використання житлової гнучкості споживання важливим є детальне знання про загальне електричне навантаження конкретного домогосподарства.

За даними «розумних лічильників» впродовж експериментального двотижневого дослідження п’яті господарств Німеччини отримано доступні детальні знання про електричний попит споживачів.

У дослідженні застосовано систему за принципом автоматичного виявлення та класифікації електронних пристроїв домівки за допомогою єдиного точкового датчика вимірювання (NILM), розташованого відразу після основного енергометру (див. : рис. 2.1).

Система використовує той факт, що сучасні електронні пристрої та освітлювальні прилади використовують імпульсні джерела живлення (SMPS) для забезпечення високої ефективності

Джерела живлення вносять високо частотні електромагнітні завади (EMI) під час роботи, які поширюються по електропроводці.

Сигнали є стабільними та передбачуваними, виходячи з характеристик ти перемикання пристрою.

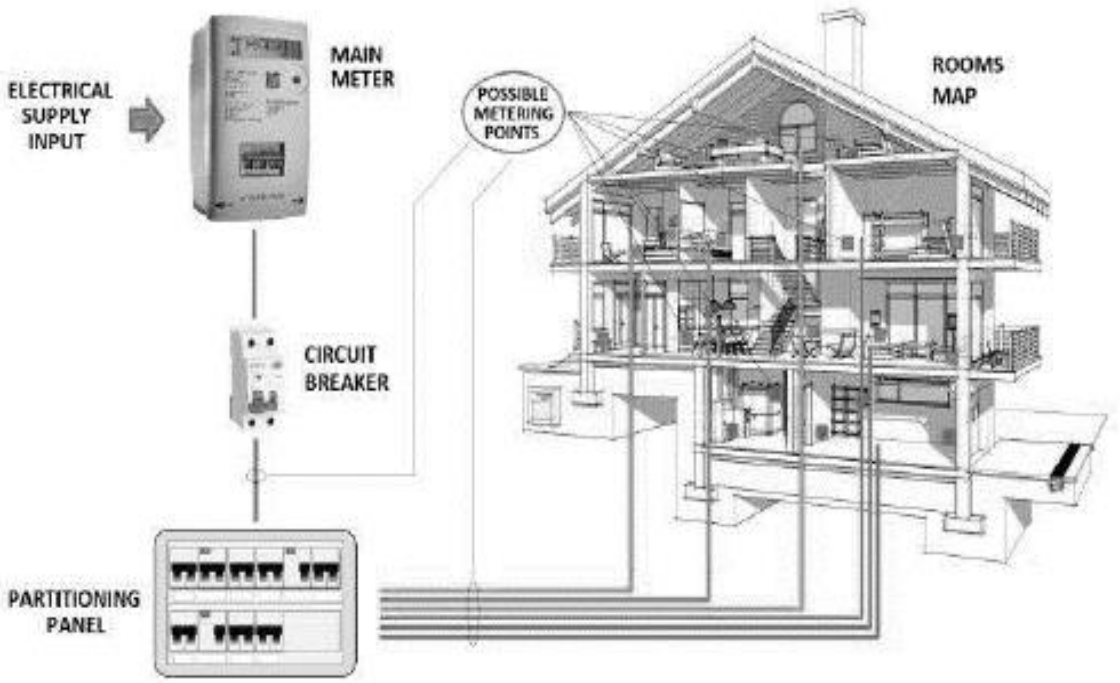

Рисунок 2.1 - Типова система електроживлення

Такий підхід дозволяє визначити EMI-сигнатуру кожного приладу і тим самим розрізняти між собою подібні пристрої;

Рисунок 2.2 та таблиця 2.1. показують графік у частотному діапазоні з включеними та виключеними приладами та результати вимірювання.

Ці дані надають можливість визначити майбутню потенційні обмеження додатку

Сучасна наука має численні дослідження щодо прогнозування навантаження.

Проте, більшість досліджень виконується для агрегованого навантаження, а не для окремих домогосподарств.

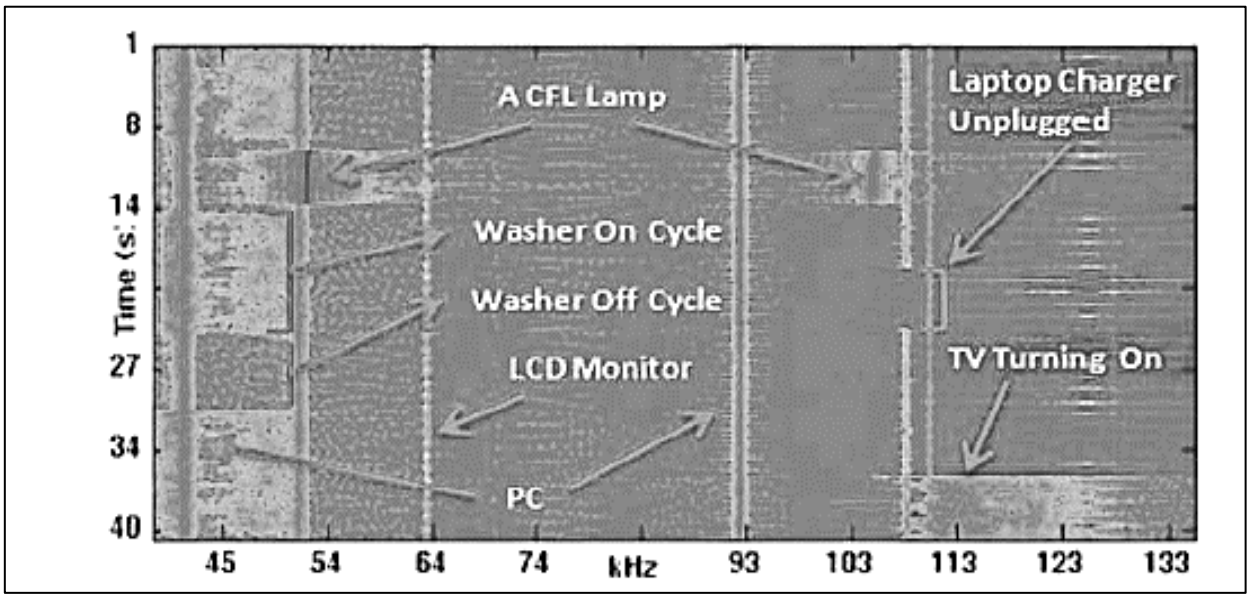

Рисунок 2.2 - Графік електроживлення у частотному діапазоні з включеними та виключеними приладами.

Таблиця 2.1.

Результати вимірювань електроживлення

State	Absolute Time	Relative Time	Sample No.	ON	
Start	21:45:30				
Incandescent ON	21:46:43	73.5	1470	I	
Refrigerator ON	21:49:47	257.1	5142	I, R	
Coffee machine ON	21:50:53	323.35	6467	I, R, C	
Toaster ON	21:53:27	476.7	9534	I, R, C, T	
Coffee machine OFF	21:55:54	623.7	12474	I, R, T	
Toaster OFF	21:56:10	639.75	12795	I, R	
Range hood ON	21:56:45	674.35	13487	I, R, Rh	Speed=3
Range hood	21:57:38	728.4	14568		Speed=2
Kettle ON	21:59:00	810.15	16203	I, R, Rh, K	
Range hood OFF	22:00:07	876.85	17537	I, R, K	
Kettle OFF	22:03:15	1064.9	21298	I, R	
Incandescent OFF	22:04:49	1158.85	23177	R	
Refrigerator OFF	22:09:27	1436.55	28731	-	
Dimmable light ON	22:10:20	1490	29800	D	Dimmest
	22:11:12	1542	30840		Brightest
Dimmable light OFF	22:13:50	1701.05	34021	-	
End	22:15:00				
Total	29.5 min	1770	35400		

Будучі детальнім, дослідження акцентовано на прогнозуванні навантаження окремих домогосподарств - у ньому виконується інтервальне прогнозування.

Оскільки навантаження окремих домогосподарств є високо волатильними та стохастичними через численні впливаючі фактори, точкове прогнозування несе ризик значного пере- або недооцінювання навантаження, що робить його проблематичними для використання в додатку.

Точкове прогнозування зосереджується на передбаченні конкретного значення навантаження, в той час як інтервальне прогнозування визначає діапазон можливих значень, оскільки дозволяє краще управляти ризиками пов'язаними з волатильністю та непередбачуваністю споживання енергії.

Тому в дослідженні замість чистого точкового прогнозування здійснено інтервальне прогнозування.

В дослідженні аналізовано підходи для визначення потенційного споживання в споживанні енергії в житлових будинках.

Це має значення у контексті інтеграції відновлюваних джерел енергії, оскільки вимагає від енергосистеми гнучкого та адаптивного управління навантаженням.

Використано значення показників, зібраних за допомогою «розумних лічильників», які забезпечують детальний огляд споживання електроенергії на рівні окремих домогосподарств (див.: рис.2.3)

Вживання цих даних дозволяє не тільки точно визначити поточне споживання, але й ефективно прогнозувати майбутнє споживання.

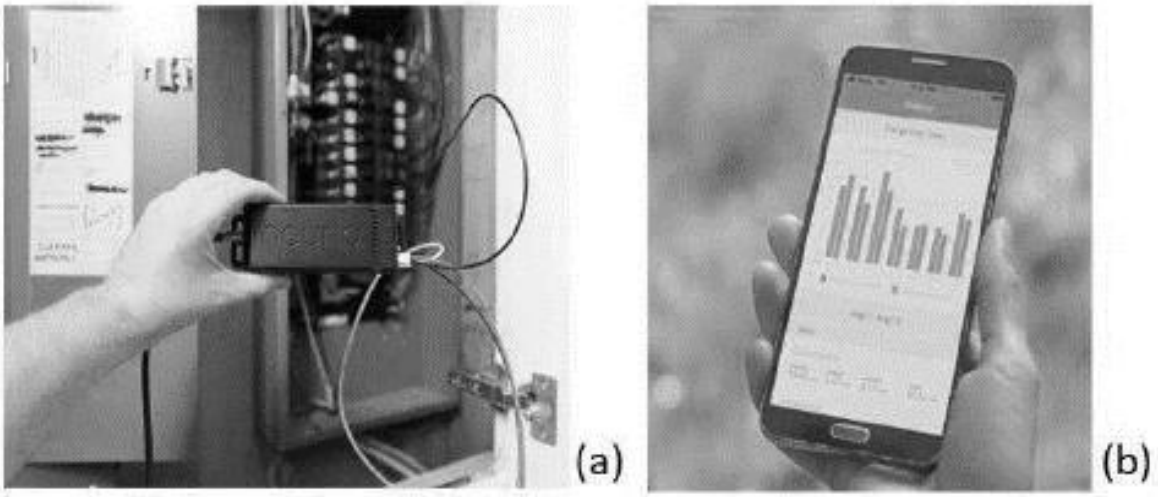

Рис 2.3. - Встановлення Electric

Отримані дані мають декілька параметрів, які будуть вирішальними при встановленні обмежень.

На прикладі, наведеному на рисунку 2.4, представлено графік виробництва фотовольтаїчної (PV) енергії впродовж дня.

Ці дані є ілюстрація зміни споживання електроенергії протягом доби, що є ключовим для інтервального прогнозування навантаження.

Згідно з графіком, прогнозоване виробництво енергії змінюється в залежності від часу дня, із піковими значеннями удень (див. : рис. 2.4)

Вертикальні смуги на графіку відображають інтервали ймовірності виробництва енергії, де більш темні кольори вказують на вищу ймовірність/

За допомогою цих даних побудовано стратегію для встановлення лімітів споживання або для управління гнучкістю споживання в домогосподарствах, зокрема, шляхом змінення часу використання важливих побутових пристроїв на моменти низького навантаження на енергосистему.

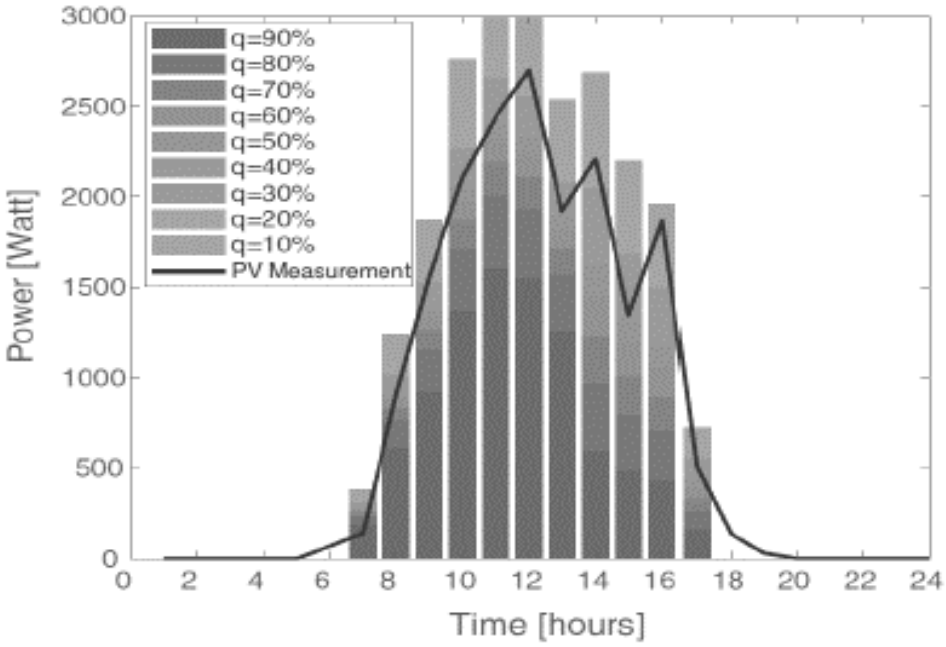

Рисунок 2.4 - Споживання в залежності від часу дня

Наступним параметром аналізу є продовжність світлового дня.

На рисунку 2.5. представлено розсіяний графік, який ілюструє залежність між кількістю світлового дня і споживанням електроенергії.

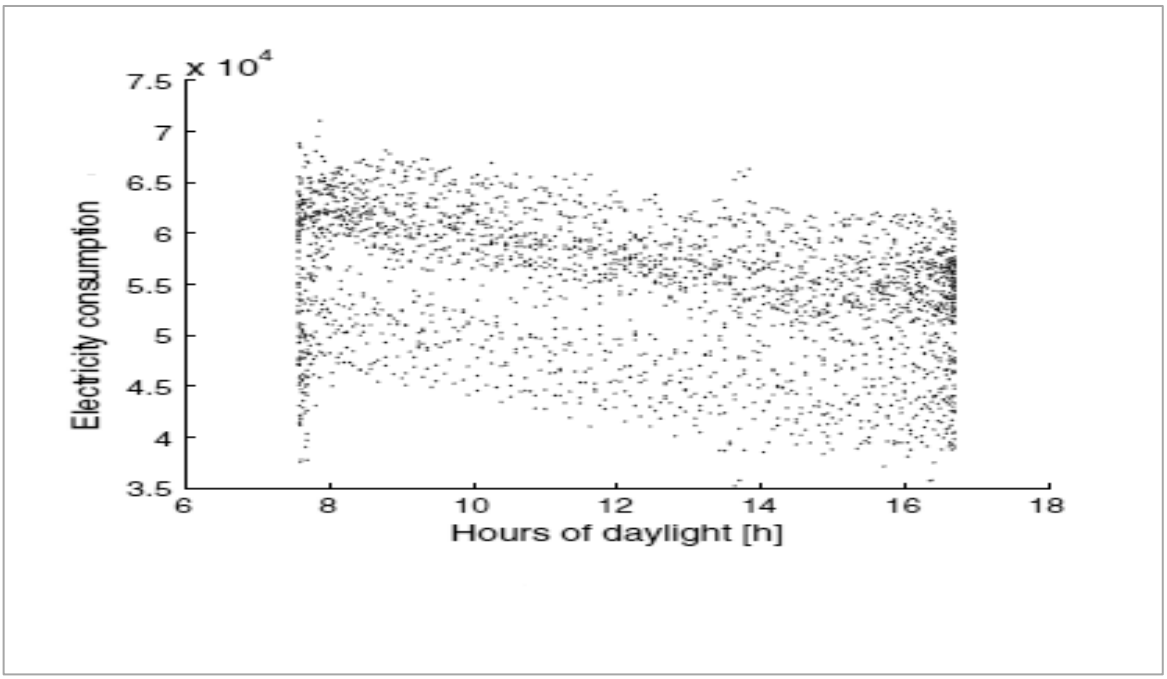

Рисунок 2.5 - Залежність споживання електроенергії від світлового дня

Кожна точка на графіку представляє окремий випадок споживання електроенергії (у ватах) в залежності від тривалості світлового дня (у годинах).

Зі збільшенням кількості годин світлового дня можна спостерігати зменшення споживання електроенергії, що, ймовірно, пов'язано з меншою потребою в освітленні приміщень протягом денних годин.

Маючи всі дані, можна перейти до прогнозування.

Побудуємо схему інтервального прогнозування (рис.2.6).

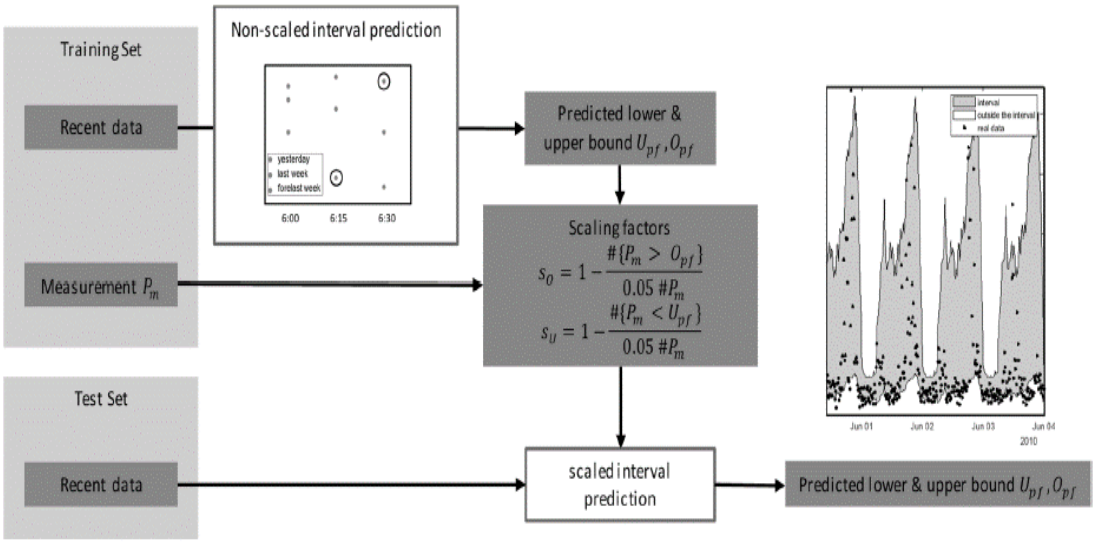

Рисунок 2.6. – Схема інтервального прогнозування

У таблиці 2.2 представлено процес прямого інтервального прогнозування навантаження (Direct Interval Load Forecast), який складається з кількох етапів.

Етапи Прямого Інтервального Прогнозування

1. Training Set (набір даних для навчання) :

вибірка недавніх даних з історичних записів споживання електроенергії для навчання моделі прогнозування.

2. Non-scaled Interval Prediction (немасштабований інтервальний прогноз) :

на основі навчального набору даних визначаються нижня та верхня границі інтервалу прогнозування. вони обираються таким чином, що береться друга найменша та друга найбільша значення з набору даних як “Op “ (нижня межа) і “Up “ (верхня межа).

3. Measurement P_m (вимірювання) :

фактичне вимірювання споживання в момент часу, яке потім використовується для масштабування інтервалу прогнозу.

Таблиця 2.2

Процес прямого інтервального прогнозування навантаження (Direct Interval Load Forecast)

Case	Points per day	Day selection	# data points	lower & upper bound
1	i-1, i, i+1	d-1, d-7, d-14	9 (7)	second smallest / second largest
2	i-2, i-1, i, i+1, i+2	d-1, d-7	10 (8)	second smallest / second largest
3	i-1, i, i+1	d-1, d-2, d-7	9 (7)	second smallest / second largest
4a	i-2, i-1, i, i+1, i+2	d-1, d-2, d-7, d-14	20 (16)	third smallest / third largest
4b	i-2, i-1, i, i+1, i+2	d-1, d-2, d-7, d-14	20 (14)	fourth smallest / fourth largest
4c	i-2, i-1, i, i+1, i+2	d-1, d-2, d-7, d-14	20 (12)	fifth smallest / fifth largest

4. Scaling Factors (коефіцієнти масштабування) :

розрахунок коефіцієнтів масштабування S_o та S_u - використовуються для коригування прогнозованих інтервалів залежно від фактичного вимірювання, формули для коефіцієнтів забезпечують корекцію інтервалу, якщо фактичне споживання виходить за межі прогнозованого.

5. Test Set (тестовий набір даних) :

використання нового набору даних (не включеного в навчальний набір) для перевірки точності масштабованих інтервалів прогнозу.

6. Scaled Interval Prediction (масштабований інтервальний прогноз) :

застосування коефіцієнтів масштабування до немасштабованого інтервалу для отримання кінцевих прогнозованих меж U_p та O_p, так, коефіцієнти масштабування можна вказати самостійно, але для того щоб вони були ефективними, вони повинні бути обґрунтованими на основі аналізу даних та моделі прогнозування.

Коефіцієнти масштабування S_oта S_u розраховуються за допомогою формул, заснованих на співвідношенні між фактичним вимірюванням P_m та прогнозованими межами нижнього O_{pl} та верхнього U_{pl} значень навантаження :

$$S_o = 1 - \frac{|P_m - O_{pl}|}{0{,}05 \times P_m} \tag{2.1}$$

де :

S_o – коефіцієнт масштабування;

P_m – фактичне вимірювання;

O_{pl} – межа нижнього значення навантаження.

$$S_u = 1 - \frac{|P_m - U_{pl}|}{0{,}05 \times P_m} \tag{2.2}$$

де :

S_u – коефіцієнт масштабування;

P_m – фактичне вимірювання;

U_{pl} – межа нижнього значення навантаження.

Формули 2.1. та 2.2. показують, що коефіцієнти масштабування визначаються на основі відхилення прогнозованих меж від фактичного вимірювання, нормалізованого до певної частини вимірювання (у цьому випадку, 5%).

Висновок :

1. Фактичні дані споживання : вимірювано фактичного споживання енергії в даний час.

2. Прогнозовані інтервали : визначено прогнозовані нижні та верхні межі споживання енергії з використанням моделі прогнозування.

3. Розрахунок коефіцієнтів : використовуючи фактичні дані та прогнозовані інтервали, розраховано коефіцієнти масштабування за вищенаведеними формулами.

4. Коригування інтервалів : після визначення коефіцієнтів скориговано прогнозовані інтервали, через множення їх на відповідні коефіцієнти масштабування.

У таблиці 2.3. представлені сценарії вибору даних для прогнозування :

- case (випадок) : різні стратегії вибору даних для прогнозу.
- points per day (кількість точок на день) : які точки споживання в день беруться для аналізу.
- day selection (вибір днів) : з яких днів вибираються дані.
- number of data points (кількість точок даних) : загальна кількість точок даних, які використовуються.
- lower & upper bound (нижня та верхня межа) : які точки даних вибираються як нижні та верхні межі інтервалу.

Таблиця 2.3

Порівняння точкових та інтервальних прогнозів

	Point-based prediction methods					Direct interval methods					
	P_{SLP}	P_5	P_1	P_6	P_7	*case 1*	*case 2*	*case 3*	*case 4a*	*case 4 b*	*case 4 c*
I_{median}	1.99	1.76	1.95	1.97	1.97	1.92	1.85	1.92	1.86	1.85	1.87
I_{mean}	2.14	2.00	2.11	2.12	2.13	3.02	2.76	2.94	2.72	2.72	2.61
I_{max}	8.31	9.63	8.7	8.45	8.62	24.63	18.49	23.24	17.41	17.8	17.57
f_{mean}	90.1 %	90.1 %	90.1 %	90.1%	90.1 %	89.7 %	89.7 %	89.7 %	89.8 %	89.7 %	89.7 %
Δf	16.0 %	9.9 %	14.8 %	15.5 %	15.8 %	6.0 %	6.4 %	7.2 %	7.4 %	6.9 %	6.9 %

Ця інформація важлива для визначення того, як формувати навчальний набір даних та які точки використовувати для встановлення границь інтервалу прогнозування.

Кожен випадок може бути адаптований до конкретних умов та потреб прогнозування.

Зображення включає опис результатів оцінювання точності різних методів інтервального прогнозування споживання електроенергії.

У розділі 3 описано дослідження, яке було проведено для оцінювання цих методів, використовуючи дані з частотою в 15 хвилин протягом одного року.

Мета полягала в тому, щоб порівняти прогнозовані інтервали ширини (I_{median} , I_{mean}, I_{max}) та ступеня виконання (f_{mean}, Δ_f) для оцінювання точності прогнозів.

Таблиця надає порівняння між точковими методами прогнозування та прямими інтервальними методами.

Для кожного методу розраховано та порівняно показники :

- I_{median} , I_{mean} , I_{max} - медіанна, середня та максимальна ширина інтервалів;
- f_{mean} - середній відсоток випадків, коли реальне споживання потрапляло в прогнозований інтервал;
- Δ_f - відсоткова різниця між прогнозованим і фактичним виконанням.

Додаткові параметри, такі як середня абсолютна помилка (MAE), корінь з середньоквадратичної помилки (RMSE) і максимальна абсолютна помилка E_{max} які дають змогу оцінити загальну точність прогнозів.

Ця таблиця корисна для визначення обмежень у споживанні електроенергії, оскільки вона дозволяє порівняти різні методи та вибрати найбільш точний для планування та управління енергоспоживанням.

Інтервали з меншими ширинами та вищими значеннями f_{mean} вказують на більшу точність та на більшу надійність прогнозів для реалізації ефективного управління навантаженням, встановлення обмежень електроенергії. [21]

Середнє навантаження в домогосподарствах коливається від 160 Вт до майже 1 кВт, з явними відмінностями між ними.

Річне споживання електроенергії варіюється від 1400 до 8635 кВт·год, з середнім значенням 4685 кВт·год, що переважно відповідає споживанню електроенергії типовим домогосподарством з показником близько 3350 кВт·год.

Це значення вище, адже в аналізі враховано лише індивідуальні будинки, які споживають більше енергії порівняно з багатоквартирними будинками.

Для оцінювання використано нормалізовані відхилення через різноманіття навантажень між домогосподарствами.

Дані поділено на навчальну та тестову вибірки, з приблизно 6 днями для навчання та 6 днями для тестування; перші два дні не враховано через недостатність історичних даних.

Для порівняння передбачених інтервалів аналізовано ширину інтервалів та ступінь їх виконання.

Точність різних прогнозів перевірено за допомогою таких показників, як :

- середня абсолютна відсоткова помилка (MAPE),
- середня абсолютна помилка (MAE),
- середньоквадратична помилка (RMSE) та максимальна абсолютна помилка (E_{max}).

Час обчислення також розглядається як додатковий параметр продуктивності.

Обраний метод дозволяє зробити наступне :

Для застосування методу прямого інтервального прогнозування, вирішено завдання прогнозування добового споживання електроенергії холодильником.

На основі дослідницьких даних встановлено, що середнє споживання холодильником становить 1,5 кВт·год на добу, з коливаннями від 1 кВт·год до 2 кВт·год.

Використовуючи ці дані визначено інтервали прогнозування :

1. Збір даних : на основі даних за 14 днів встановлено коливання споживання.

2. Визначення меж : вибрано друге найнижче (1,2 кВт год) та друге найвище (1,8 кВт год) значення з історичних даних як прогнозовані межі.

3. Розрахунок коефіцієнтів масштабування : використовуючи останнє вимірювання 1,4 кВт год, розраховано коефіцієнти S_oта S_u для корекції прогнозованих інтервалів, згідно з формулами 2.1. та 2.2.

4. Прогноз : Застосовано коефіцієнти для коригування інтервалів. Якщо, наприклад, коефіцієнт масштабування

S_o виходить 0,95, а S_u 1,05, то скориговані межі інтервалу стануть 1,14 кВт·год (1,2 0,95) та 1,89 кВт·год (1,8 та 1,05) відповідно.

Ці скориговані інтервали використано для встановлення лімітів на добове споживання електроенергії холодильником, що дозволило не тільки планувати енергетичні витрати, але й вжити заходи для оптимізації енергоспоживання.

Завдяки цьому підходу підвищено енергоефективність і знижено витрати на електроенергію.

2.2 Створення IoT-частини

Концепція Інтернету речей (IoT) бере свій початок з 1980-х років, коли перші пристрої почали обмінюватися даними через вбудовані процесори.

Термін “Internet of Things” вперше був використаний Кевіном Ештоном в 1999 році. Він висловив ідею, що комп'ютери можуть управляти об'єктами в реальному світі, забезпечуючи автоматизацію та розумніше прийняття рішень. [25]

В основі IoT лежить ідея RFID-технології (радіочастотна ідентифікація), яка дозволяє електронним тегам відправляти інформацію про об'єкти.

З розвитком бездротових технологій, зниженням вартості сенсорів і збільшенням обчислювальної потужності, IoT почав розвиватися і знайшов застосування в інформаційній індустрії.

У додатку Nest IoT працює через «розумний термостат» Nest, де використовує ряд сенсорів, алгоритмів машинного навчання та інтерфейсів користувача для оптимізації опалення та охолодження в будинках.

Алгоритм Nest:

- збір даних через датчики температури, вологості, активності та освітлення, щоб визначити стан дому і прийняти рішення про найбільш ефективне управління кліматом;
- автоматизація з використанням цих даних, Nest самонавчається впродовж кількох тижнів, запам'ятовуючи звички та вподобання споживачів та автоматично налаштовуючи температуру для комфорту та ефективності;
- дистанційне керування через мобільний додаток, що дозволяє змінювати налаштування і переглядати історію споживання енергії;
- енергозбереження автоматично знижує температуру, коли користувачі відсутні, що допомагає економити енергію і витрати;
- інтеграція з іншими пристроями Nest або сторонніми продуктами для створення більш об'єднаного і інтелектуального дому.

Nest визнається одним з інноваційних прикладів використання IoT для споживачів.

Застосування IoT в контексті вкладки “Вольтметр” додатку, що проектується у кваліфікаційному проекті, має особливості.

У додатку вкладка “Вольтметр” є прикладом застосування IoT для моніторингу електричних параметрів.

Використання ESP32, мікроконтролера з можливістю підключення до інтернету, дозволяє зчитувати дані з під'єднаних датчиків або вимірювальних пристроїв і передавати ці дані до додатку (див.: рис. 2.7.).

На даний момент ESP32 використовується для передачі сигналу на світлодіод, що може демонструвати статус (включено/виключено) або інші характеристики електричної системи.

Побудовано схему яка передбачає:

- світлодіоди, що вказуватимуть на статус або вимірювання, здійснювані пристроєм;
- резистори, що обмежують струм до світлодіодів, запобігаючи їх пошкодженню;
- фоторезистор (LDR - light-dependent resistor), що змінює свій опір залежно від освітленості, може використовуватися для вимірювання рівня освітленості в приміщенні.

Показники фоторезистору (показник ldr_data) та напруга (voltage), можуть бути використані для моніторингу умов навколишнього середовища або для контролю використання енергії.

Ця інформація, за допомогою ESP32, передається до хмарної бази даних (Firebase у вашому випадку), де її можна зберігати, аналізувати та відображати в додатку.

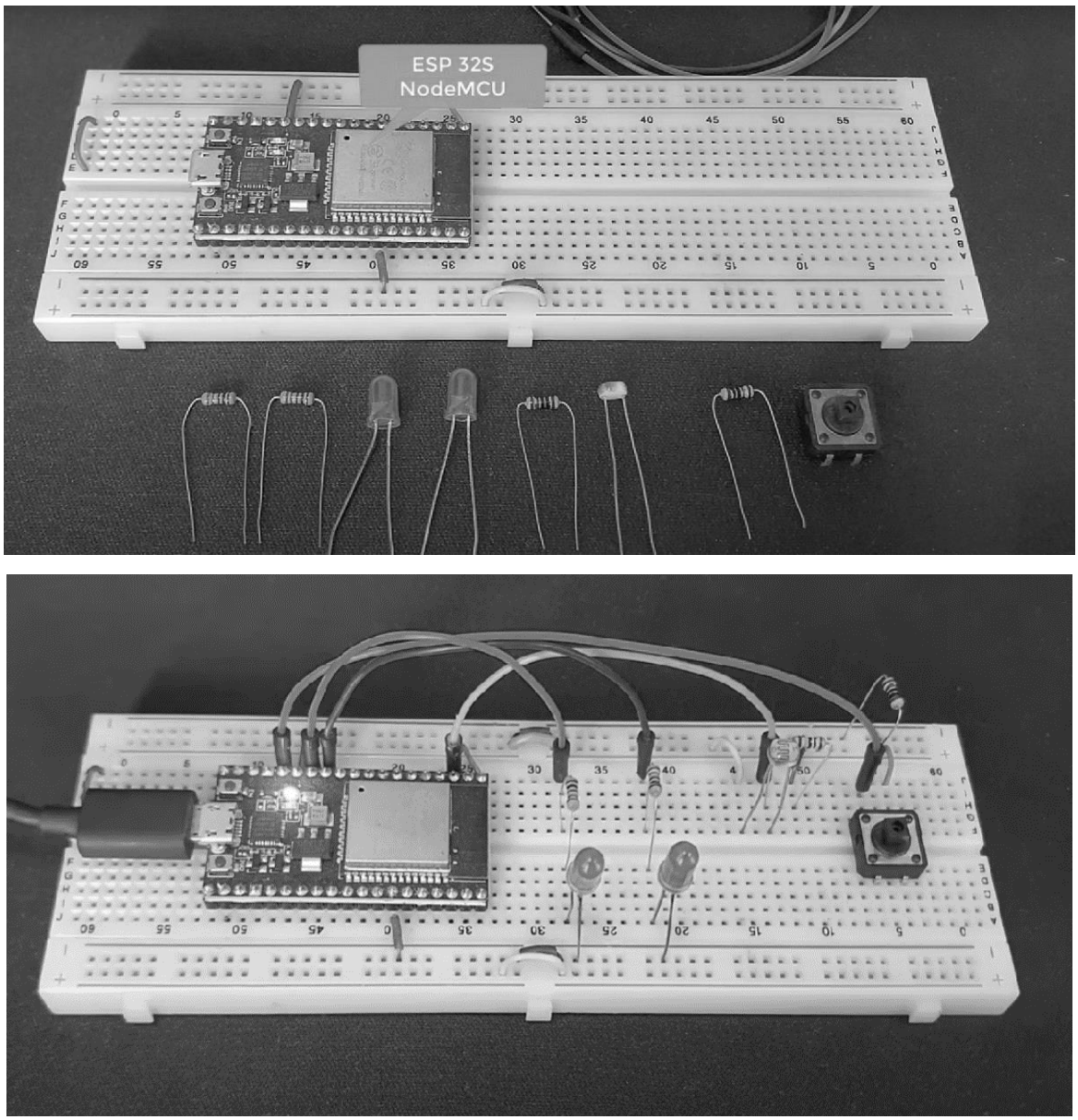

Рисунок 2.7 - Схема підключення компонентів

1. Підключення бібліотек та налаштування констант :

- Підключаються необхідні бібліотеки для роботи з Wi-Fi та Firebase.
- Визначаються константи для Wi-Fi (SSID та пароль) та Firebase (API ключ та URL бази даних).

```
#include <Arduino.h>
#if defined(ESP32)
#include <WiFi.h>
#elif defined(ESP8266)
#include <ESP8266WiFi.h>
#endif
#include <Firebase_ESP_Client.h>
```

2. Ініціалізація об'єктів :

- FirebaseData fbdo; створює об'єкт для взаємодії з Firebase.
- FirebaseAuth auth; та FirebaseConfig config; використовуються для аутентифікації та конфігурації.

3. Функція setup :

- Ініціалізує серійний порт для дебагу.
- Підключається до Wi-Fi.
- Налаштовується конфігурація Firebase з використанням API ключа та URL бази даних.
- Проходить аутентифікацію на Firebase.
- Ініціалізується підключення до Firebase і встановлюється автоматичне підключення до Wi-Fi у разі втрати з'єднання.

```
Serial.begin(115200);
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting to Wi-Fi");
while (WiFi.status() != WL_CONNECTED){
  Serial.print(".");
  delay(300);
}
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());
Serial.println();
```

4. Функція loop :

- Перевіряється, чи готовий Firebase і чи було успішно здійснено реєстрацію.
- Періодично (кожні 15 секунд) відправляє два типи даних до Firebase : ціле число, яке збільшується на 1 з кожним циклом (count), і випадкове дробове число (0.01 + random(0,100)).

```
if (Firebase.RTDB.setInt(&fbdo, "test/int", count)) {
  Serial.println("PASSED");
  Serial.println("PATH: " + fbdo.dataPath());
  Serial.println("TYPE: " + fbdo.dataType());
}
else {
  Serial.println("FAILED");
  Serial.println("REASON: " + fbdo.errorReason());
}
```

Кожен виклик Firebase :

- Записує дані на вказаний шлях у базі даних.
- Виводить у серійний порт статус операції (успішно чи ні), шлях до даних та тип даних.

Після цих кроків, наш пристрій під’єднано

Перевіряємо доступність даних :

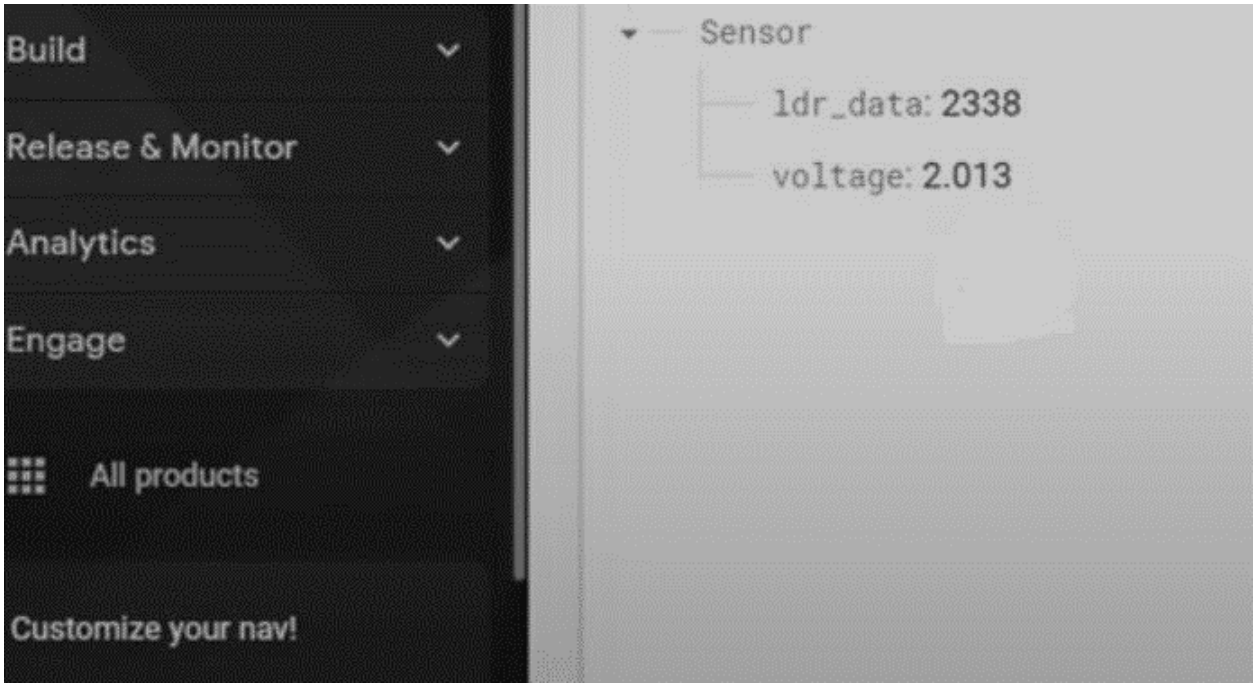

Рисунок 2.8 - Перевірка під’єднання пристрою

- ldr_data : 2338 - дані фоторезистору, що вимірює освітленість.
- voltage : 2.013 - дані про напругу, зібрані з сенсора

Ці дані відображаються у Firebase Realtime Database і можуть бути використані для моніторингу в реальному часі або для подальшого аналізу.

У майбутньому, ця система буде розширена для включення додаткових датчиків та вимірювальних пристроїв, таких як ваттметри, що дозволить користувачам отримувати більш повну картину свого споживання енергії.

Це може сприяти кращому управлінню енергетичними ресурсами і наданню рекомендацій щодо покращення енергоефективності.

3 ІМПЛЕМЕНТАЦІЯ ДОСЛІДЖЕНЬ В ДОДАТОК WALTIO

3.1. Додавання можливостей Firebase до класу та розробки його API

В розділі 1.2, ми вже встановили бібліотеки Firestore, та створили новий проект. Продовжимо побудову додатка створивши .dart файли на необхідні нам екрани.

На рисунку 3.1. представлено імплементацію аналітичних сервісів Firebase.

Рисунок 3.1.- Екрани додатку та Імплементація Firebase Analytics

Firebase Analytics дозволяє глибоко зрозуміти, як користувачі взаємодіють з вашим додатком. [35] Ви можете відслідковувати такі дії, як частота відкриття додатку, найпопулярніші функції, тривалість сесій та багато іншого. Це буде необхідно для пізнішої реалізації функціоналу додатку і

взагалом є дуже корисною функцією. Оскільки все в Flutter є віджетом, орієнтуватися серед них дуже складно, тому додамо Navigation Bar внизу.

```
import 'package:flutter/material.dart';
import
'package:font_awesome_flutter/font_awesome_flutter.dart';
class AppBottomNav extends StatelessWidget {

 Widget build(BuildContext context) {
   return BottomNavigationBar(
     items: [
       BottomNavigationBarItem(
           icon: Icon(FontAwesomeIcons.graduationCap, size:
20),
           title: Text('Consumption')),
       BottomNavigationBarItem(
           icon: Icon(FontAwesomeIcons.bolt, size: 20),
           title: Text('About')),
       BottomNavigationBarItem(
           icon:  Icon(FontAwesomeIcons.userCircle,  size:
20),
           title: Text('Profile')),
     ].toList(),
     fixedColor: Colors.deepPurple[200],
     onTap: (int idx) {
       switch (idx) {
         case 0:
           // do nuttin
           break;
         case 1:
           Navigator.pushNamed(context, '/about');
        break;
         case 2:
           Navigator.pushNamed(context, '/profile');
           break;
```

Оскільки далі починається роботи із активними частинами додатку, треба перетворити наш Firestore Class в Dart Class.

Попередньо створено базу даних із результатами досліджень. Результатом вимірювань є таблиця формату .xls (Excel).

Firebase не може звертатися до такого формату файлу, тому використовано xceliser, який допоміг серіалізувати документи Excel у формат JSON або десеріалізувати JSON у документ Excel.

Використовано встановлену JSON_seziable для автоматизації процесу.

Зіставимо типи даних з їхніми конструкторами, щоб їх можна було динамічно інстанціювати:

```
import 'services.dart';
import 'package:firebase_analytics/firebase_analytics.dart';
/// Static global state. Immutable services that do not care about
build context.
class Global {
   // Data Models
 static final Map models = {
   Topic: (data) => Topic.fromMap(data),
   Quiz: (data) => Quiz.fromMap(data),
   Report: (data) => Report.fromMap(data),
```

“Quiz” - це клас, який зберігає відповіді користувача на наступні питання:

З чим ви працюєте зараз? (Студент, Блогер, Працівник, Пенсія)

Чи є сьогодні свято?

Чи є прилади або обладнання, які мають бути враховані як значні споживачі електроенергії? (кондиціонер влітку, обігрівач взимку)

Чи є в вашому домогосподарстві будь-які зміни, що вплинули б на звичайний розподіл споживання енергії? (Нові предмети)

Квіз сторінку побудовано наступним чином:

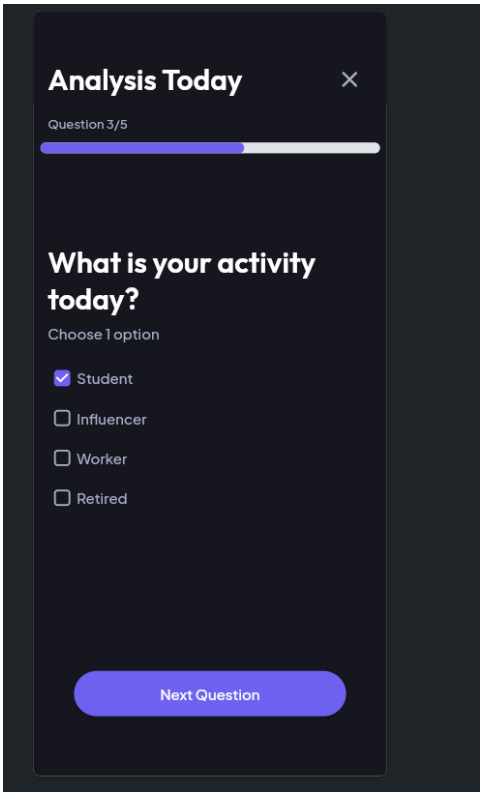

Рисунок 3.2 - Квіз-Сторінка Waltio

Імплементація вимірювань у додаток Waltio розпочинається з додавання залежностей Firebase до класу та розробки його API.

Префікс _ використовується для назв властивостей, щоб створити приватні члени в Dart. Ми зробимо це для клієнтських бібліотек, які використовує клас.

Потоки, надані Firebase, транслюються для кількох підписників (це називається гарячим Observable в Rx). У цьому проекті користувач є записом аутентифікації, тоді як профіль представляє користувацькі дані, які будуть збережені про цього користувача в Firestore.

Також, PublishSubject loading - це як гарячий observable, за винятком того, що значення можна відправляти до нього вручну. Ми використаємо його для перемикання стану завантаження в нашому коді.

auth.dart

```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:rxdart/rxdart.dart';

class AuthService {
  // Dependencies
  final GoogleSignIn _googleSignIn = GoogleSignIn();
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final Firestore _db = Firestore.instance;

  // Shared State for Widgets
  Observable<FirebaseUser> user; // firebase user
  Observable<Map<String, dynamic>> profile; // custom user data in Firestore
  PublishSubject loading = PublishSubject();

  // constructor
  AuthService() {
```

Рисунок 3.3 – Библиотека авторізації додатку

Flutter не нав'язує конкретної архітектури для управління спільним глобальним станом у додатку. Серед популярних підходів можна виділити Redux та використання вбудованих віджетів, але вони вимагають певної підготовчої роботи і можуть бути обтяжливими під час прототип формування UX.

Простий підхід до управління станом полягає в наданні глобальної змінної як екземпляру класу, яка містить потоки та/або Rx Observables.

Це дозволило відокремити логіку аутентифікації від віджетів, одночасно дозволяючи будь-якому віджету слухати та реагувати на зміни у стані аутентифікації.

Алгоритм аутентифікації

- Observables/Streams використовується лише для даних із станом;
- клас AuthService створити лише один раз (хоча технічно в цьому випадку він не є синглтоном);

- створити новий файл під назвою lib/auth.dart;
- створення Класу AuthService.

Розпочнемо з додавання функцій Firebase до класу та розробки його API.

Префікс використовується для назв властивостей для створення приватних членів у Dart.

В кваліфікаційному дослідженні це виконано для клієнтських бібліотек, застосованих класом.

Потоки надані Firebase транслюються для кількох підписників (це називається гарячим Observable в Rx).

У демонстрації користувач є записом аутентифікації, тоді як профіль представляє користувацькі дані, які будуть збережені в Firestore.

PublishSubject loading працює як гарячий Observable, але значення можуть бути відправлені до нього вручну.

Ми використали його для перемикання стану завантаження у нашому коді.

```
import 'package : firebase_auth/firebase_auth.dart';
import 'package : google_sign_in/google_sign_in.dart';
import 'package : cloud_firestore/cloud_firestore.dart';
import 'package : rxdart/rxdart.dart';
class AuthService {
// Dependencies
final GoogleSignIn _googleSignIn = GoogleSignIn();

final FirebaseAuth _auth = FirebaseAuth.instance;

final Firestore _db = Firestore.instance;
Observable<FirebaseUser> user; // firebase user
Observable<Map<String, dynamic>> profile; // custom user
data in Firestore
PublishSubject loading = PublishSubject();
// constructor
AuthService()
}
Future<FirebaseUser> googleSignIn() async {
}
```

```
void updateUserData(FirebaseUser user) async {
void signOut()
```

Стан Ініціалізується в конструкторі цього класу.

Користувач - це просто «обгортка» Observable над потоком Firebase «onAuthStateChanged».

Ця частина критично важлива, оскільки ми можемо тоді застосовувати оператори Rx для неймовірного контролю над потоком.

Для отримання даних профілю з бази даних спочатку потрібно отримати UID залогіненого користувача - ось де RxDart починає бути дійсно чудовим.

Оператор «switchMap» бере користувача з зовнішнього Observable, а потім використовує його для перемикання на інший Observable запису бази даних Firestore.

Якщо користувач не ввійшов у систему, додаток просто повертає Observable порожню мапу.

Ця логіка використовує потужність реактивного програмування з RxDart для створення гнучкої та ефективної системи управління.

За допомогою «switchMap» можна переключатися між різними станами, входу та виходу користувача, та обробляти зміни стану, гарантуючи, що віджети будуть оновлені з найновішою інформацією про стан користувача (див.: рис.3.2.)

```
AuthService() {
  user = Observable(_auth.onAuthStateChanged);

  profile = user.switchMap((FirebaseUser u) {
    if (u != null) {
      return _db
          .collection('users')
          .document(u.uid)
          .snapshots()
          .map((snap) => snap.data);
    } else {
      return Observable.just({});
    }
  });
}
```

Рисунок 3.3 – Приклад запитання користувачу

3.2. Управління додатком за допомогою Google

Вхід за допомогою Google.

Цей процес показує нативний екран входу Google і надає idToken та accessToken.

1. Вхід за допомогою Google :

- показується нативний екран входу Google, на якому користувач вводить свої дані;
- після успішного входу, Google надає “idToken “ та “accessToken “, які є ключами для аутентифікації користувача.

2. Вхід у Firebase :

- на цьому етапі користувач вже ввійшов у систему Google, але ще не у Firebase.

- ми можемо просто передати токени (idToken та accessToken) до Firebase, щоб виконати вхід.

- Firebase використовує токени для аутентифікації користувача та створення сесії входу.

3. Оновлення Firestore :

- після успішного входу у Firebase, оновлюється база даних Firestore будь-якими користувацькими даними, які слід використовувати в UI.

- оновлюється профіль користувача, налаштування або будь-яких інших даних, які пов'язані з користувацьким обліковим записом.

Цей процес забезпечує плавний та безпечний вхід користувача через Google, а також інтеграцію з Firebase, дозволяючи управляти користувацькими даними та аутентифікацією в межах додатку.

```
Future<FirebaseUser> googleSignIn() async {
 // Start
 loading.add(true);
 // Step 1
 GoogleSignInAccount googleUser = await _googleSignIn.signIn();
 // Step 2
 GoogleSignInAuthentication      googleAuth      =      await
googleUser.authentication;
 FirebaseUser user = await _auth.signInWithGoogle(
 accessToken    :    googleAuth.accessToken,    idToken    :
googleAuth.idToken)
```

Рисунок 3.4. – Приклад логіки аутентифікації

Логіку аутентифікації виконано.

UI-елементів відображаються на основі Observable стану - StatefulWidget та StreamBuilder.

StatefulWidget, який пов’язано з Observable профілю та відображає його дані у UI. Ми слухаємо потік, а потім використовуємо отримані значення для виклику “setState “ на віджеті, чим спричиняємо зміни.

Використання StreamBuilder є більш гнучким та лаконічним підходом. Це просто віджет, який залежить від значення потоку, тому коли значення потоку змінюється, ви можете умовно відображати різні віджети.

Це ідеальний випадок для використання кнопки входу. Наприклад, ми не хочемо показувати входу через Google у користувацькому інтерфейсі, якщо користувач вже увійшов у систему.

StreamBuilder приймає Observable користувача як потік, потім у додатку Waltio реалізується функція побудови, яка повертає UI віджети.

Snapshot - це дані, випущені потоком/observable.

У додатку Waltio, ми знаємо, що користувач увійшов у систему, якщо snapshot.hasData == true (див. рис. 3.4.)

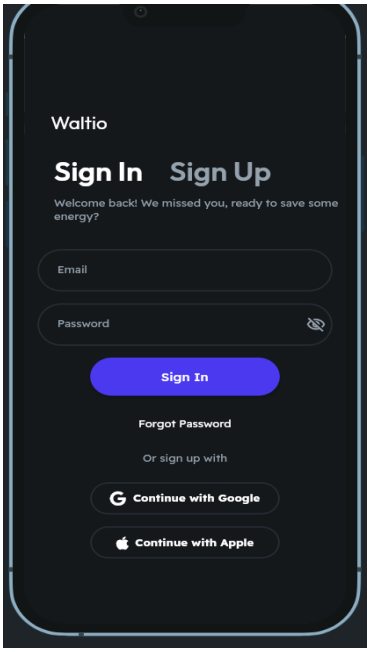

Рисунок 3.4 - Створення Сторінки Waltio

Далі створимо клас EnergyConsumptionPredictor з методом calculateScaledIntervals, який використовує коефіцієнти масштабування S_oта S_u розраховуються за допомогою формул, заснованих на співвідношенні між фактичним вимірюванням P_m(історичні дані) та прогнозованими межами нижнього O_{pl} та верхнього U_{pl} значень навантаження :

$$S_o = 1 - \frac{|P_m - O_{pl}|}{0{,}05 \times P_m} \tag{3.1}$$

$$S_u = 1 - \frac{|P_m - U_{pl}|}{0{,}05 \times P_m} \tag{3.2}$$

для розрахунку масштабованих меж інтервалу.

```
class EnergyConsumptionPredictor {
  double lastMeasurement; // Останнє вимірювання
  double lowerBound; // Нижня межа
  double upperBound; // Верхня межа

  EnergyConsumptionPredictor({
    required this.lastMeasurement,
    required this.lowerBound,
    required this.upperBound,
  });

  Map<String, double> calculateScaledIntervals() {
    double scalingFactorLower = 1 - ((lastMeasurement -
lowerBound).abs() / (0.05 * lastMeasurement));
    double scalingFactorUpper = 1 - ((lastMeasurement -
upperBound).abs() / (0.05 * lastMeasurement));

    double scaledLowerBound = lowerBound * scalingFactorLower;
    double scaledUpperBound = upperBound * scalingFactorUpper;

    return {
      'scaledLowerBound': scaledLowerBound,
      'scaledUpperBound': scaledUpperBound
    };
  }
}

void main() {
  // Приклад використання класу
  var predictor = EnergyConsumptionPredictor(
    lastMeasurement: 1.4,
    lowerBound: 1.2,
    upperBound: 1.8,
  );

  var scaledIntervals = predictor.calculateScaledIntervals();
  print('Scaled Lower Bound:
${scaledIntervals['scaledLowerBound']}');
  print('Scaled Upper Bound:
${scaledIntervals['scaledUpperBound']}');
```

Отримали інші параметри, такі як lowerBound та upperBound для використання та звернулися до відповідей із історічними даними.

– для lowerBound (нижня межа) знайдемо мінімальне добове значення споживання за цей період.

– для upperBound (верхня межа) знайдемо максимальне добове значення споживання за цей період.

```
import 'dart:convert';
import 'dart:io';

Future<void> main() async {

  var filePath = '/D/ReseachData/Limits.json';
  var file = File(filePath);
  var jsonString = await file.readAsString();
  var jsonData = jsonDecode(jsonString) as List;

  var energyValues = jsonData.map((e) => e['energy'] as double).toList();
  var minValue = energyValues.reduce((value, element) => element < value ? element : value);
  var maxValue = energyValues.reduce((value, element) => element > value ? element : value);

  print(': $minValue');
  print(': $maxValue');
}
```

Використаємо множувач в залежності від відповіді на Квіз.

Зробимо запит до бази даних, щоб отримати ці значення, маємо функція fetchConsumptionData для отримання даних з бази даних:

```
Future<Map<String, double>> fetchBounds() async {
  List<double> consumptionData = await fetchConsumptionData();

  double lowerBound = consumptionData.reduce(min);
  double upperBound = consumptionData.reduce(max);

  return {
    'lowerBound': lowerBound,
    'upperBound': upperBound,
  };
}
```

Щоб додати фактор дня тижня, додамо наступне:

```
import 'dart:convert';
```

```
import 'dart:io';

void main() async {
  var file = File('path_to_your_json_file.json');
  var jsonString = await file.readAsString();
  var jsonData = jsonDecode(jsonString) as List;
```

Вибірка даних за певний день тижня, наприклад, четвер:

```
  var dayOfWeek = 'Thursday';
  List<double> energyValues = jsonData.where((item) {
    return item['day_of_week'] == dayOfWeek;
  }).map<double>((item) {
    return item['energy_consumption'] as double;
  }).toList();

      // Перевірка, чи є дані за вказаний день
  if (energyValues.isEmpty) {
    print('Дані за вказаний день не знайдено.');
    return;
 }
```

// Знаходження мінімального та максимального значень

```
  double minValue = energyValues.reduce(min);
  double maxValue = energyValues.reduce(max);

print('Найменше значення споживання енергії у $dayOfWeek:
$minValue');
print('Найбільше значення споживання енергії у $dayOfWeek:
$maxValue');
}
```

Пояснення:

- читаємо JSON файл і перетворюємо його в масив об'єктів;
- використовуємо where() для фільтрації записів, що відповідають вказаному дню тижня;

– застосовуємо map() для отримання списку значень споживання енергії.reduce() використовується для знаходження мінімального та максимального значень у відфільтрованому списку.

Створимо Push-Notifications на випадок перевищення ліміту:

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Notification History',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity:
VisualDensity.adaptivePlatformDensity,
      ),
      home: NotificationHistoryPage(),
    );
  }
}

class NotificationHistoryPage extends StatelessWidget {
  final List<Map<String, dynamic>> notifications = [
    {
      'title': 'Limit evaluated',
      'type': 'normal',
    },
    {
      'title': 'Limit exceeded, Achtung!',
      'type': 'warning',
    },
    // Add more notifications as needed
  ];

  @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Notifications'),
      ),
      body: ListView.builder(
        itemCount: notifications.length,
        itemBuilder: (context, index) {
          final item = notifications[index];
          return ListTile(
            title: Text(item['title']),
            subtitle: Text(item['date']),
            tileColor:  item['type']  ==  'warning'  ?
Colors.red[100] : null,
          );
        },
      ),
    );
  }
}
```

Перевіримо наявність повідомлень – див.: рис. 3.5:

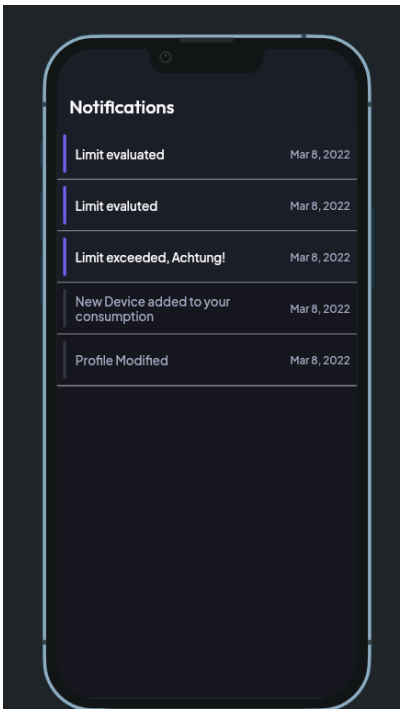

Рисунок 3.5 - Повідомлення системи

ВИСНОВКИ

У дипломній роботі розробляється тема «Аналіз та побудова систем моніторінгу споживання електроенергії».

У представленій роботі розглядається актуальна тема використання інформаційних технологій для оптимізації енергоспоживання та підвищення енергоефективності у житлових будинках. Робота має значення у контексті зростаючого використання відновлюваних джерел енергії та необхідності гнучкого управління навантаженням у енергосистемах.

Робота фокусується на аналізі даних, зібраних за допомогою "розумних лічильників", що дозволяють точно визначати поточне та прогнозувати майбутнє споживання електроенергії.

Важливою частиною є розробка та застосування методів інтервального прогнозування навантаження з використанням статистичних даних.

Виконано ряд експериментів для визначення точності різних методів інтервального прогнозування, що має велике значення для планування та управління енергоспоживанням.

Використання розроблених методів дозволяє оптимізувати використання електроенергії, планувати енергетичні витрати та сприяти підвищенню енергоефективності.

Розглянуто практичне застосування інтернету речей (IoT) в контексті енергоменеджменту, демонструючи ефективність сучасних технологій у моніторингу та управлінні споживанням енергії в домогосподарствах.

За результатами дослідження можна зробити висновок про високий потенціал використання інформаційних технологій для оптимізації енергоспоживання та розробки ефективних методів управління енергетичними ресурсами.

Робота відкриває шлях для подальших досліджень у цій галузі і може бути використана як основа для реалізації практичних рішень у сфері енергетики.

На мові Dart та завдяки фрейм-ворку Flutter добудовано додаток, який допомагає рядовому користувачу оптимізувати своє енергоспоживання

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. . Алгоритми та структури даних у розробці програм: монографія / Київ: Наукова думка, 2019. – 291 с.
2. Веб-додатки: від концепції до реалізації: монографія / Львів: Львівська Політехніка, 2019. – 134 с.
3. Енергоменеджмент: стратегії та технології / Львів: Львівська Політехніка, 2020. – 150 с.
4. Інноваційні системи моніторингу енергоспоживання: монографія. / Харків: ХНУРЕ, 2019. – 148 с.
5. Інтелектуальне управління енергією: монографія. / Одеса: Одеська Політехніка, 2021. – 146 с.
6. Основи UI/UX дизайну для розробників / Харків: ХНУРЕ, 2020. – 206 с.
7. Офіційна документація Firebase : веб-сайт [Електронне видання]. - Режим доступу URL : https : //firebase.google.com/docs 2
8. Офіційна документація Flutter : веб-сайт. [Електронне видання] Режим доступу URL : https : //flutter.dev/docs
9. Офіційна документація по сумісній роботі Flutter та Firebase : веб-сайт. [Електронне видання]- Режим доступу URL : https : //firebase.flutter.dev/
10. Програмування мобільних додатків: монографія. / Київ: ІТ-Прес, 2019 – 235 с.
11. Розробка крос-платформних додатків / Дніпро: Арт Прес, 2021. – 89 с.
12. Сучасні системи контролю енергоспоживання: навчальний посібник. / Київ: КНУБА, 2021. – 208 с.
13. Шеннон К. Работы по теории информации и кибернетике / М.: Изд-во иностранной литературы, 1963. – 832
14. Akopkokhyants S. Mastering Dart // Sergey Akopkokhyants, 2022. – 90 p.
15. Balbaert I. Dart Cookbook // Ivo Balbaert, 2014. – 78 p.

16. Belchin M. Web Programming with Dart //M.Belchin, P.Juberias, 2014. – 107 p.
17. Bracha G. The Dart Programming Language // Gilad Bracha, 2021. –65 p.
18. Buckett C. Dart in Action // Chris Buckett, 2023. – 105 p.
19. Carney, John; Cunningham, Pádraig; Bhagwan, Umesh : Confidence and prediction intervals for neural network ensembles - International Joint Conference on Neural Networks. Washington : IEEE, 2019.
20. Chen, Bo-Juen; Chang, Ming-Wei; Lin, Chih-Jen : Load forecasting using support vector machines : A study on EUNITE competition 2021. In : IEEE transactions on power systems / Piscataway Township : IEEE, 2021.
21. Clow M. Learn Google Flutter Fast : 65 Example Apps // Mark Clow, 2022. – 191 p.
22. De Vieaux, Richard; Shumi, Jennifer; Schweinsberg, Jason; Ungar, Lyle : Prediction intervals for neural networks via nonlinear regression. In : Technimetrics 40/2020. / Abingdon : Taylor & Francis, 2020.
23. El-Baz, Wessam; Seufzger, Michael; Lutzenberger, Sandra; Tzscheutschler, Peter; Wagner, Ulrich : Impact of probabilistic small-scale photovoltaic generation forecast on energy management systems. In : Solar Energy 165/2021. / Amsterdam : Elsevir, 2021.
24. El-Baz, Wessam; Tzscheutschler, Peter; Wagner, Ulrich : Dayahead probabilistic PV generation forecast for buildings energy management systems. In : Solar Energy 171/2021/. Amsterdam : Elsevir, 2021.
25. Freitas E. Flutter Succinctly // Ed Freitas, 2022. – 87 p.
26. Götz, Manuel et al. : Renewable Power-to-Gas : A technological and economic review / Karlsruhe : Karlsruhe Instituts für Technologie, 2019.
27. Hinterstocker, Michael; von Roon, Serafin; Dufter, Christa : Smart Meter Benefits / Munich : FfE GmbH, 2022. – 178 p.

28. Hsu, Y-Y; Ho, K.-L. : Fuzzy expert systems : an application to short-term load forecasting. Stevenage : IEE Proceedings C(Generation, Transmission and Distribution), 2022.
29. Humeau, Samuel, Wijaya, Tri Kurniawan; Vasirani, Matteo, Aberer, Karl : Electricity Load Forecasting for Residential, Customers : Exploiting Aggregation and Correlation between Households / Lausanne : École Polytechnique Fédérale de Lausanne, 2023. – 312 p.
30. Mainkar P. Google Flutter Mobile Development Quick Start Guide : Get up and running with iOS and Android mobile app development // P. Mainkar, S. Giordano, 2022. – 106 p.
31. Miola A. Flutter Complete Reference : Create beautiful, fast and native apps for any device // Miola Albero, 2020. – 104 p.
32. Moore K. Flutter Apprentice // K. Moore, M. Katz, V. Ngo, 2019. – 123 p.
33. Papadopoulos, Georgios; Edwards, Peter; Murray, Alan : Confidence estimation methods for neural networks : A practicalcomparison. Piscataway Township : IEEE transactions on neural networks, 2021
34. Rap P. Beginning App Development with Flutter : Create Cross-Platform Mobile Apps / Rap Payne, 2019. – 210 p.
35. Regett, Anika; Conrad, Jochen; Fattler, Steffen : project «Документація до пакету flutter_bloc. веб-сайт». [Електронне видання] Режим доступу
36. Sikore M. Dart Essentials // Martin Sikora, 2015. – 89 p.
37. Strom C. Dart for Hipsters // Chris Strom, 2021. – 108 p.
38. Walrath K. Dart : Up and Running // K. Walrath, S. Ladd, 2022. – 204 p.
39. Windmill E. Flutter in Action // Eric Windmill, 2019. –112 p.
40. Zaccagnino C. Programming Flutter : Native, Cross-Platform Apps the Easy Way // Carmine Zaccagnino, 2020.
41. Zammetti F. Practical Flutter : Improve your Mobile Developemnt with Google’s Latest Open-Source SDK // Frank Zammetti, 2019.

42. Zapranis, Achilleas; Livanis, Efstratios : Prediction intervals for neural network models - Proceedings of the 9th WSEAS International Conference on Computers. Wisconsin : World Scientific and Engineering Academy and Society (WSEAS) , 2022.

ДОДАТОК А

Довідка про участь у студентської програмі FIT4UKRAINE від DAAD

DAAD Deutscher Akademischer Austauschdienst
German Academic Exchange Service

HOCHSCHULE ANHALT University of Applied Sciences

Stipendienprogramm „Future Innovation Talents 4 Ukraine“

Stipendienvereinbarung

zwischen

der Hochschule Anhalt

-Stipendiengeber-

und

Nachname, Vorname:	Vlatrov, Oleksii
Geburtsdatum und -ort:	16.03.1999, Chishinau, Moldova
Status:	Master-Studierende/r
E-Mail-Adresse:	oleksii.vietrov@student.hs-anhalt.de

-Geförderte Person-

§ 1 Vollstipendium

Der Stipendiengeber vergibt an die geförderte Person im Rahmen einer Projektförderung des Deutschen Akademischen Austauschdienstes (DAAD) aus Mitteln des Auswärtigen Amts im Förderprogramm „Zukunft Ukraine – Stipendienprogramm für Geflüchtete aus der Ukraine“ ein Vollstipendium in Höhe von insgesamt **931 Euro pro Monat**

für ein Studium im Studiengang: Elektro- und Informationstechnik

mit dem **Stipendienzweck einen Studienabschluss zu erreichen.**

in dem Förderzeitraum:

vom: 01.04.2023

bis: 31.12.2024

Das Stipendium steht unter dem Vorbehalt der Mittelzuweisung durch den Deutschen Akademischen Austauschdienst (DAAD) an den Stipendiengeber.

Stipendienvereinbarung – FIT4Ukraine – Stand: 24.03.2023 Seite 1 von 3

DAAD Deutscher Akademischer Austauschdienst
German Academic Exchange Service

§ 2 Mitteilungs- und Mitwirkungspflichten der geförderten Person

Die geförderte Person ist verpflichtet, der Hochschule alle Änderungen von Tatsachen, die für die Vergabe des Stipendiums relevant sind (z.B. BAföG, Sozialleistungen, andere Förderung) unverzüglich anzuzeigen.

Die geförderte Person ist verpflichtet ihren Studienfortschritt jedes Semester an das International Office zu melden. Die Anmeldung zur Bachelor- bzw. Masterarbeit ist dem International Office zeitnah anzuzeigen, da das Stipendium mit Erreichung des Abschlusses endet, auch wenn der maximale Förderzeitraum noch nicht ausgeschöpft wurde.

Die geförderte Person versichert an den Maßnahmen des Projektes und Förderprogramms „Future Innovation Talents 4 Ukraine" in Abstimmung mit dem International Office teilzunehmen.

§ 3 Kündigung des Stipendiums aus wichtigem Grund

Bei Vorliegen eines wichtigen Grundes ist das Stipendium seitens des Stipendiengebers durch Kündigung der Stipendienvereinbarung zu beenden. Die Stipendienleistungen werden unverzüglich eingestellt. Ein wichtiger Grund liegt insbesondere vor, wenn

a) die geförderte Person das Stipendium durch vorsätzliche oder grob fahrlässige Täuschung über erhebliche Tatsachen erschlichen hat (falsche bzw. unvollständige Angaben oder Verschweigen),

b) das Stipendium nicht zweckentsprechend verwendet worden ist und die geförderte Person dies kannte oder nur infolge grober Fahrlässigkeit nicht kannte,

c) Tatsachen erkennen lassen, dass die geförderte Person sich nicht im erforderlichen und zumutbaren Umfang um die Zweckerreichung bemüht,

d) der Zweck des Stipendiums (Studienabschluss) nicht mehr erreicht werden kann.

§ 4 Rückzahlung des Stipendiums

Im Falle der Kündigungsgründe gemäß § 4 a) und b) dieser Stipendienvereinbarung sind die bereits ausgezahlten Beträge zurückzuzahlen und zu verzinsen.

Bricht die geförderte Person ihr Studium aus Gründen, die sie selbst vorsätzlich oder grob fahrlässig zu vertreten hat vorzeitig ab, muss sie das Stipendium grundsätzlich zurückzahlen.

§ 5 Salvatorische Klausel

Sollten einzelne Bestimmungen dieses Vertrages unwirksam oder undurchführbar sein oder nach Vertragsschluss unwirksam oder undurchführbar werden, bleibt davon die Wirksamkeit des Vertrages im Übrigen unberührt. An die Stelle der unwirksamen oder undurchführbaren Bestimmung soll diejenige wirksame und durchführbare Regelung treten, deren Wirkungen der wirtschaftlichen Zielsetzung am nächsten kommen, die die Vertragsparteien mit der unwirksamen bzw. undurchführbaren Bestimmung verfolgt haben. Die vorstehende Bestimmung gilt entsprechend für den Fall, dass sich der Vertrag als lückenhaft erweist.

Änderungen und Ergänzungen dieser Vereinbarung bedürfen der Schriftform.

ДОДАТОК Б

Програмний код файлу MAIN.DART

```
import 'dart:async';
import 'dart:math';

import 'package:flutter/material.dart';
import   'package:flutter/services.dart'   show   SystemChrome,
DeviceOrientation;
import 'package:intl/intl.dart' show DateFormat;

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Clock(),
      ),
}
// Utility class to access values as binary integers
class BinaryTime {
  List<String> binaryIntegers;

  BinaryTime() {
    DateTime now = DateTime.now();
    String hhmmss = DateFormat("Hms").format(now).replaceAll(':',
'');

    binaryIntegers = hhmmss
        .split('')
        .map((str)  =>  int.parse(str).toRadixString(2).padLeft(4,
'0'))
        .toList();
  }

  get hourTens => binaryIntegers[0];
  get hourOnes => binaryIntegers[1];
  get minuteTens => binaryIntegers[2];
  get minuteOnes => binaryIntegers[3];
```

```
  get secondTens => binaryIntegers[4];
  get secondOnes => binaryIntegers[5];
}
/// Column to represent a binary integer.
class ClockColumn extends StatelessWidget {
  String binaryInteger;
  String title;
  Color color;
  int rows;
  List bits;

  ClockColumn({this.binaryInteger,    this.title,    this.color,
this.rows = 4}) {
    bits = binaryInteger.split('');
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        ...[
          Container(
            child: Text(
              title,
              style: Theme.of(context).textTheme.display1,
            ),
          )
        ],
        ...bits.asMap().entries.map((entry) {
          int idx = entry.key;
          String bit = entry.value;

          bool isActive = bit == '1';
          int binaryCellValue = pow(2, 3 - idx);

          return AnimatedContainer(
            duration: Duration(milliseconds: 475),
            curve: Curves.ease,
            height: 40,
            width: 30,
            decoration: BoxDecoration(
              borderRadius: BorderRadius.all(Radius.circular(5)),
              color: isActive
                  ? color
```

```
                    : idx < 4 - rows
                        ? Colors.white.withOpacity(0)
                        : Colors.black38,
                ),
                margin: EdgeInsets.all(4),
                child: Center(
                  child: isActive
                      ? Text(
                          binaryCellValue.toString(),
                          style: TextStyle(
                            color: Colors.black.withOpacity(0.2),
                            fontSize: 18,
                            fontWeight: FontWeight.w700,
                          ),
                        )
                      : Container(),
                ),
              );
            }),
            ...[
              Text(
                int.parse(binaryInteger, radix: 2).toString(),
                style: TextStyle(fontSize: 30, color: color),
              ),
              Container(
                child: Text(
                  binaryInteger,
                  style: TextStyle(fontSize: 15, color: color),
                ),
              )
            ]
          ],
        );
      }
    }
```