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

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Парастатова Ярослава Маріановича*
(ПІБ)

академічної групи *121-20-1*
(шифр)

напряму підготовки *121 Інженерія програмного забезпечення*
(код і назва напряму підготовки)

на тему: *Розробка програмного забезпечення мобільного застосунку обробки даних агромоніторингу*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Лактіонов І.С.*			
розділів:				
спеціальний	*проф. Лактіонов І.С.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Мартиненко А.А.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-20-1* (група) *Парастатова Ярослава Маріановича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка програмного забезпечення мобільного застосунку обробки даних агромоніторингу*

затверджена наказом ректора НТУ «ДП» від *23.05.2024 р.* № *469 -с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***10.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***14.06.2024 р.***

Завдання видав _____ *проф. Лактіонов І.С.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Парастатов Я.М.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *17.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 69 с., 15 рис., 3 дод., 20 джерел.

Об’єкт розробки: програмне забезпечення мобільного застосунку обробки даних агромоніторингу.

Метою розробки мобільного додатку системи агротехнічного моніторингу є надання власникам аграрного бізнесу можливості автоматизувати процеси та збирати дані для подальшого аналізу. Додаток надає можливість віддалено зчитувати показники численних сенсорів, розташованих у різних регіонах земельних угідь, візуалізувати та аналізувати їх для подальшого прийняття більш оптимальних сільськогосподарських рішень.

Основними методами, які використовувались при розробці, є: аналіз літературних джерел та технічної документації, визначення потреб та вимог українських аграрних аналітиків для створення програми, проектування та розробка архітектури програмного продукту, застосування провідних технологій в сфері розробки мобільних додатків, зокрема мови програмування Kotlin та інтегрованого середовища розробки Android Studio.

У вступі розглядається сучасний стан проблеми та її аналіз, визначається мета дипломної роботи, її галузь використання, наводиться обґрунтування актуальності теми та конкретизується постановка завдання.

У першому розділі був проведений аналіз існуючих застосунків агротехнічного моніторингу. На основі проведеного аналізу запропоновано розробити мобільний додаток для обробки даних, який буде простим у використанні, візуалізуватиме дані та матиме можливість експорту та імпорту даних.

У другому розділі визначено функціональні можливості розроблюваного застосунку, обрано архітектуру і шаблони проектування. Виконано розробку програмного продукту, описана робота програми, її структура та алгоритми функціонування, а також її виклик та завантаження, визначені вхідні та вихідні дані та склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробки програмного забезпечення, проведено розрахунок вартості робіт зі створення програми та встановлено необхідний час для її розробки.

Практичне завдання полягає у створенні мобільного застосунку, який отримує дані із системи агротехнічного моніторингу, оброблює та візуалізує їх, а також зберігає на локальному пристрої. Візуалізація даних включає графіки, діаграми та інші елементи, які допомагають швидко зрозуміти поточний стан полів та спрогнозувати майбутні зміни.

Список ключових слів: ANDROID, KOTLIN, МОБІЛЬНИЙ ЗАСТОСУНОК, АГРОТЕХНІЧНІ ДАНІ, АРХІТЕКТУРА, ШАБЛОН ПРОЕКТУВАННЯ, ДІАГРАМА, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

ABSTRACT

Explanatory note: 69 pages.,15 fig, 3 appendix, 20 sources

Object of development: Mobile application software for agricultural data processing and monitoring.

The purpose of developing a mobile application of the agrotechnical monitoring system is to provide agrarian business owners with the opportunity to automate processes and collect data for further analysis. The application provides an opportunity to remotely read values of numerous sensors located in different regions of farmsteads, visualize and analyze them for further adoption of more optimal agricultural decisions.

The main methods used in the development are: analysis of literary sources and technical documentation, determination of the needs and requirements of Ukrainian agricultural analysts for the creation of the program, design and development of the software product architecture, application of leading technologies in the field of mobile application development, in particular the Kotlin programming language and Android Studio integrated development environment.

The introduction examines the current state of the problem and its analysis, defines the purpose of the thesis, the field of its use, provides a justification for the relevance of the topic, and specifies the statement of the task.

In the first section, an analysis of existing applications of agrotechnical monitoring was done. Based on the analysis, it is proposed to develop a mobile application for data processing, which will be easy to use, visualize data and have the ability to export and import data.

In the second section the functionality of the developed application was defined, the architecture and design patterns were selected. The development of the software product has been carried out, the operation of the program, its structure and functioning algorithms, as well as its calling and downloading, the input and output data and the composition of technical means parameters have been described.

In the economic section, the labor intensity of software development is determined, the cost of work on creating the program is calculated, and the necessary time for its development is determined.

The practical task is to create a mobile application that receives data from the agrotechnical monitoring system, processes and visualizes them, and also stores them on a local device. Data visualization includes graphs, charts, and other elements that help the user quickly understand the current state of fields and predict future changes.

Keywords: ANDROID, KOTLIN, MOBILE APPLICATION, AGRITECH DATA, ARCHITECTURE, DESIGN PATTERN, DIAGRAM, SOFTWARE DEVELOPMENT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

MVVM – Model-View-ViewModel

IDE – Integrated Development Environment

SQL – Structured Query Language

HTTPS – HyperText Transfer Protocol Secure

JSON – JavaScript Object Notation

JVM – Java Virtual Machine

DAO – Data Access Object

СУБД – Система Управління Базою Даних

GPS – Global Positioning System

AVD – Android Virtual Device

ВСТУП

Інформаційні технології допомагають людству у всіх сферах його діяльності. Вони використовуються не лише в офісній роботі, але й у фізичній праці, зокрема в сільському господарстві. Впровадження сучасних технологій дозволяє фермерам отримувати корисні аналітичні дані у великому обсязі, складати прогнози та статистику. Завдяки цьому, продуктивність полів та врожайність значно збільшується. Однак, українські фермери часто стикаються з фінансовими обмеженнями, що ускладнює впровадження дорогих сучасних технологій. У зв’язку з цим, виникла потреба в розробці доступної технології, яка б виконувала всі необхідні функції для ефективного агромоніторингу, але при цьому була економічно вигідною.

Метою кваліфікаційної роботи є розробка мобільного застосунку, який отримує дані із системи агротехнічного моніторингу, оброблює та візуалізує їх, а також зберігає на локальному пристрої. Застосунок має допомогти землеробам приймати оптимальні рішення щодо управління своїми полями для підвищення продуктивності.

Візуалізаційна частина цієї роботи включає в себе створення зрозумілого інтерфейсу користувача, який у зручній формі подає необхідну інформацію. Завдяки цьому, фермери можуть легко аналізувати дані та робити обґрунтовані висновки. Візуалізація даних включає графіки, діаграми та інші елементи, які допомагають швидко зрозуміти поточний стан полів та спрогнозувати майбутні зміни.

Таким чином, розробка доступного та ефективного програмного забезпечення для агромоніторингу сприятиме підвищенню продуктивності українських фермерів, забезпечить їм доступ до сучасних технологій і допоможе у прийнятті обґрунтованих рішень, що позитивно вплине на аграрний сектор України.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Впровадження цифрових технологій у галузь агрокультури проходить повільніше, ніж в інші галузі. В останні роки ця ситуація змінюється завдяки зниженню вартості цифрових рішень, тож агрокультурні підприємства можуть користуватися перевагами агротехнічного моніторингу, використовуючи прості програмні та апаратні рішення [1]. Згідно з дослідженням Kurkul.com [2], проведеним у 2023 році, 42% опитаних фермерів використовують GPS-навігацію для ведення обліку полів, 38% використовують датчики для моніторингу вологості ґрунту, а 27% використовують супутникові зображення для відстеження вегетаційного індексу (NDVI) сільськогосподарських культур.

Сучасний міжнародний ринок інформаційних технологій пропонує багато систем агротехнічного моніторингу. За призначенням їх умовно можна розподілити на організаційні та функціональні. Здебільшого вони являють собою застосунки до мобільних пристроїв або веб-інтерфейси, для надання більшої зручності під час роботи у полі, проте є крос-платформні варіанти, розраховані на аналіз та вивід великого обсягу даних із використанням персональних комп’ютерів.

Наразі одними з найвідоміших інструментів для керування аграрними процесами вважаються John Deere Operations Center, FieldAlytics, Sirrus, FieldView Cab та AGMRI. Наведені застосунки мають різний набір функціоналу та можливостей, інтерфейс користувача а також призначення. Більшість із цих додатків є самостійними та можуть використовуватись автономно, проте деякі з них підтримують можливість комунікації із іншими системами, можливості імпорту\експорту даних, а також взаємодію із сторонніми девайсами та технікою, такою як різні моделі польової техніки John Deere та Climate FieldView™ Drive.

Особлива увага при розробці застосунку для агроменеджменту приділяється продумуванню та створенню дизайну інтерфейсу користувача, оскільки такі додатки мають візуалізувати велику кількість даних, та надавати користувачеві можливість точно їх модифікувати та отримувати до них доступ. Окрім цього, необхідно чітко встановити набір функцій, наявний у додатку, щоб не перевантажити його зайвим функціоналом та не дезорієнтувати користувача.

Було розглянуто організаційну систему агротехнічного моніторингу FieldAlytics [3]. Інтерфейс поділено на головний екран поточної вкладки та панель навігації у нижній частині екрана (рис. 1.1). Головний екран Dashboard надає користувачеві можливість обрати поточний обліковий запис, відображає інформацію про погодні умови в регіоні та розклад найближчих завдань. У вкладці Profiles користувач може продивитись візуалізацію даних та робіт, накладених на супутникове зображення місцевості. Вкладка Dispatch дозволяє менеджеру створити нові задачі, розподілити ресурси, техніку та людей на їх виконання. Вкладка Work Orders дозволяє робітнику продивитися свій список задач та їх стан виконання.

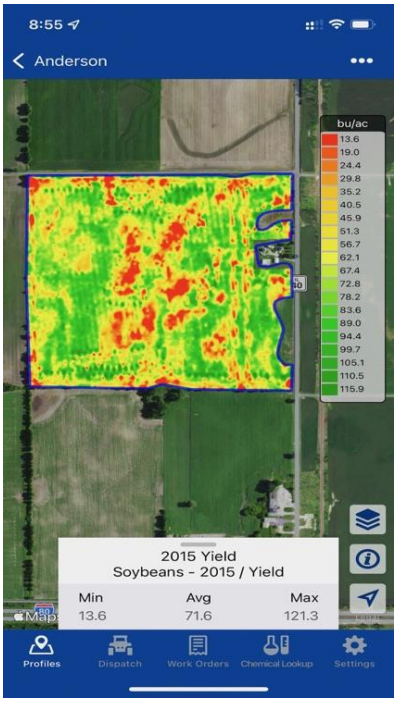

Рис. 1.1. Інтерфейс програми FieldAlytics

Додатково, у панелі навігації присутня вкладка Chemical Lookup, яка являє собою базу даних хімічних речовин таких як добрива, засоби захисту рослин від шкідників та гербіциди. Користувач може продивитись склад певного продукту, максимально допустиму норму його застосування та інші дані. Проте, цей функціонал не має прямого відношення до основних функцій системи агротехнічного моніторингу, тож постає питання стосовно доцільності його додавання до основного додатку.

Розглянемо іншу організаційну систему агроменеджменту John Deere Operational Center [4]. У застосунку також присутня нижня панель навігації. Головний екран (рис. 1.2) містить шкалу процесу виконання робіт у полі, список нових повідомлень, а також список задіяних у роботі ресурсів та їх стану. Вкладка Карта (Map) відображає розташування техніки та робочі процеси у реальному часі, що дуже важливо при роботі у великій команді. Вкладка Ресурси (Setup) містить інформацію про усі наявні на проекті ресурси: перелік робочої техніки, полів, агроміхікатів та робітників. Також на цьому екрані можна модифікувати ці списки або продивитись властивості кожного запису, що дуже зручно. На вкладках Планування (Plan) та Аналіз (Analyze) можна продивитися перелік запланованих задач та аналітику їх виконання у табличному форматі відповідно. Робітники мають можливість слідкувати за своїми робочими годинами та продуктивністю машин, що дозволяє визначити місця простою та проводити оптимізацію маршрутів.

Система надає користувачам систему повідомлень, що спрощує процес планування, видачі та відстежування завдань, допомагає співпрацівникам структурувати власний робочій день.

Інтерфейс John Deere Operational Center не перенавантажений зайвими вікнами та функціоналом, що робить його зручним при використанні працівником будь якої ролі. Окрім цього, система підтримує експорт та імпорт даних у інші системи агротехнічного моніторингу.

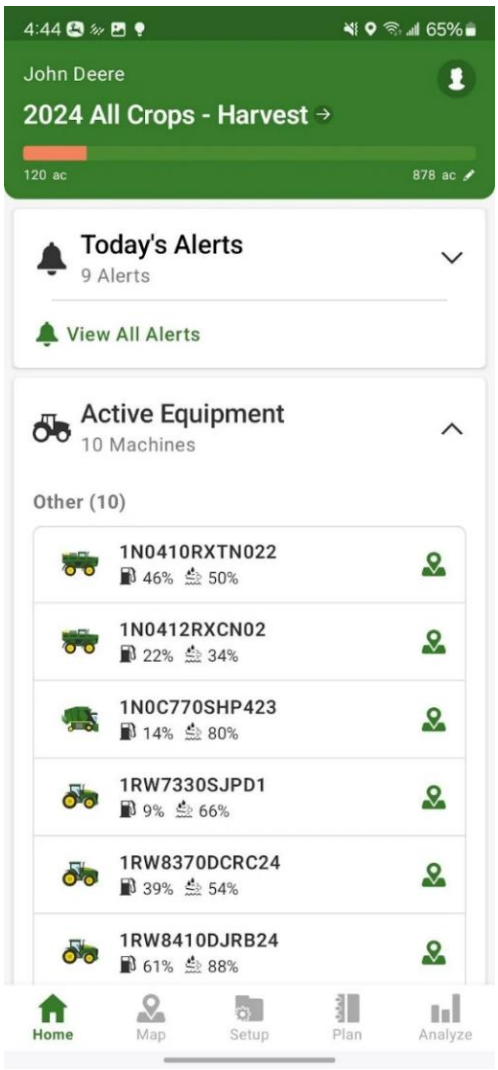

Рис. 1.2. Інтерфейс мобільного додатку John Deere Operational Center

Наступною розглянемо функціональну агромоніторингову систему Intelinair AGMRI та її мобільний застосунок для аналізу даних. Клієнти сервісу AGMRI отримують цифрову версію своїх полів, із повною детальною інформацією стосовно топографії місцевості, термального розподілу, індексу вегетації а також загального аграрного скаутингу [5]. Цифрова версія полів створюється за допомогою сукупності регулярних високоякісних супутникових знімків, зображень з літаків та дронів. Пізніше дані оброблюються та аналізуються за допомогою штучного інтелекту, що дозволяє прогнозувати потенційні проблеми, які можуть виникнути впродовж аграрного сезону, що допомагає приймати вірні своєчасні рішення та вживати відповідних заходів для їх запобігання. Сервіс гарантує підтримку та надання

корисних аналітичних даних не тільки під час аграрного сезону, а й до його початку та після його завершення. Окрім цього, AGMRI зберігає архівні дані полів користувача для подальшого прогнозування та прийняття рішень, що будуть підтверджені існуючими даними.

Важливою особливістю системи Intelinair AGMRI є широка можливість експорту аналітичних даних у інші системи. Наприклад, усі дані про цифрові поля можна експортувати у застосунок John Deere Operations Center, у якому команді агрономів буде легше координувати дії команди та розподіляти ресурси безпосередньо під час виконання робіт. Також, AGMRI за допомогою штучного інтелекту генерує оптимальні маршрути та алгоритми роботи польової техніки, такої як сівалки при посіві або комбайни при зборі врожаю, які пізніше можна імпортувати напряму до програмного забезпечення сумісної машини.

На рисунку 1.3 представлено інтерфейс головного екрана Поля (Fields) додатку AGMRI. Сторінка являє собою список полів користувача у спеціальному форматі «цифрових полів» із можливістю змінити представлення для перегляду різних поточних властивостей земельної ділянки. У верхній частині екрану знаходиться розбитий на категорії перелік повідомлень від системи стосовно потенціальних проблем, шо були помічені під час останнього моніторингу поля за допомогою знімків, а також ті, що були знайдені при проведенні детального аналізу отриманих раніше даних. Перелічені такі категорії проблем, як нові паростки бур’янів, низька схожість культур на певній ділянці, хвороби рослин та дефіцит поживних речовин в ґрунті. Вкладка Розвідки (Scout) дозволяє агрономам формувати та реєструвати нові запити на задачі по скаутингу певних полів для членів своєї команди, а також створювати детальні звіти стосовно проведеного скаутингу. Вкладка Погода (Weather) показує детальний прогноз погоди над обраним полем, включаючи температуру повітря, швидкість та напрям вітру, кількість опадів.

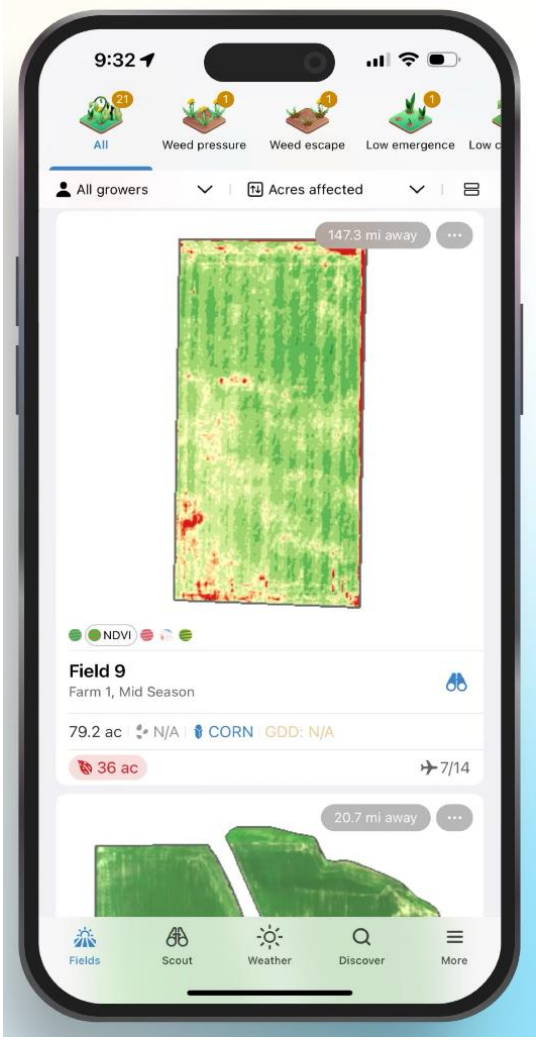

Рис. 1.3. Інтерфейс мобільного застосунку AGMRI

Як і розглянуті попередньо системи, Платформа цифрового землеробства Climate FieldView дозволяє візуалізувати дані та проводити аналітику, дозволяє налаштувати комунікацію команді агрономів, має широкі можливості експорту та імпорту даних з інших систем, а також автоматично оброблює отримані дані для створення порад для покращення продуктивності та оптимізації вкладених ресурсів.

Climate FieldView робить акцент на збір даних з полів у автоматичному режимі в реальному часі за допомогою їх зчитування напряму із сумісної польової техніки [6]. Для цього використовується окремий девайс FieldView Drive (рис. 1.4).

Рис. 1.4. Апаратне забезпечення FieldView Drive

На підставі проведеного аналізу було встановлено наступні недоліки у існуючих системах агротехнічного моніторингу:

- Масштабованість виключно під великі підприємства.

Господарства малого та середнього обсягу не мають достатньо коштів для користування великими аналітичними системами які пропонує сучасний ринок інформаційних технологій, до того ж такі підприємства не зможуть отримати вигоди з такої глибокої та масштабної аналітики, яка пропонується.

- Неактуальність систем в нашій місцевості.

Деякі з наведених платформ недоступні в Україні, або їх алгоритми не враховують місцевих умов, таких як особливості ґрунту, ландшафту, клімату та флори.

- До недоліків мобільних додатків можна віднести зайве навантаження інтерфейсу функціями, які рідко використовуються.

До таких функцій можуть належати бази даних та сторінки із прогнозами погоди. Якщо сторінка не використовується так активно як інші, то буде

доцільно прибрати її с головної навігаційної панелі, або взагалі винести її у окремий додаток.

Провівши аналіз існуючих рішень на ринку, можемо сформувати вимоги до цифрового продукту:

- Лаконічний інтерфейс користувача: одна з переваг мобільного застосунку у тому, що мобільний девайс можна використовувати під час виконання роботи. Для поліпшення доступу до функцій застосунку потрібен неперевантажений інтерфейс для більш швидкого та зручного користування додатком.
- Зрозуміла візуалізація та обробка даних: коли у системі наявні численні сенсори, що відправляють дані, важливо ефективно відобразити їх користувачеві. Для цього необхідно продумати розташування таблиць, графіків та діаграм а також можливість формування звітів.
- Комунікація з іншими системами: У додатку необхідно передбачити експорт та імпорт даних у широко використовуваних форматах, щоб дозволити користувачу обробляти дані іншими способами за потреби.

1.2. Призначення розробки та галузь застосування

Метою розробки мобільного додатку системи агротехнічного моніторингу є надання власникам аграрного бізнесу можливості автоматизувати процеси та збирати дані для подальшого аналізу. Додаток надає можливість віддалено зчитувати показники численних сенсорів, розташованих у різних регіонах земельних угідь, візуалізувати та аналізувати їх для подальшого прийняття більш оптимальних сільськогосподарських рішень.

Напрями сільського господарства, де використовується система моніторингу стану навколишнього середовища:

- Відстеження стану посівів: своєчасне виявлення та оцінка проблем зі здоров’ям посаджених культур дозволяє агрономам зупинити процес розповсюдження хвороби, щоб запобігти втратам врожаю.

- Точне землеробство: агротехнічний моніторинг допомагає агрономам впровадити практики точного землеробства, які спрямовані на максимізацію продуктивності посівів при мінімізації впливу на навколишнє середовище [7]. Завдяки точному аналізу фермери знатимуть які ділянки полів потребують певних ресурсів, що призводить до більш ефективного їх використання.
- Прийняття рішень на основі існуючих даних: маючи можливість збирати та аналізувати як поточні так і архівні дані, агрономи можуть приймати обґрунтовані рішення на основі існуючих даних, що зазвичай призводить до підвищення врожайності посівів та зниженню загальних втрат.

1.3. Підстава для розробки

В кінці навчання, студент виконує кваліфікаційну роботу (проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою.

Підставою для розробки кваліфікаційної роботи на тему «Розробка програмного забезпечення мобільного застосунку обробки даних агромоніторингу» є наказ по Національному технічному університету «Дніпровська політехніка» №469-с від 23.05.2024.

1.4. Постановка завдання

Метою кваліфікаційної роботи є розробка мобільного застосунку, який отримує дані із системи агротехнічного моніторингу, оброблює та візуалізує їх, а також зберігає на локальному пристрої. Застосунок має допомогти землеробам приймати оптимальні рішення щодо управління своїми полями для підвищення продуктивності.

Основна ідея полягає в створенні неперевантаженого інтерфейсу користувача, який у зручній формі подає користувачу необхідну інформацію, залишаючись простим у використанні. Для досягнення цієї мети

використовуються такі графічні елементи як діаграми, карти, графіки, випадаючі списки та інші інтерактивні елементи.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для програмного забезпечення, розробленого під операційну систему Android, встановлюється наступний ряд функціональних вимог:

- Отримання даних з віддаленого серверу: Застосунок повинен мати змогу встановити з’єднання із віддаленим джерелом даних та завантажити дані на пристрій для подальшої обробки.
- Зберігання даних: Застосунок повинен мати організоване внутрішнє локальне сховище даних для їх зберігання.
- Візуалізація даних: Застосунок має представляти дані у зручному та зрозумілому користувачеві вигляді, для цього можуть використовуватися діаграми, карти, графіки, зображення, таблиці та інші методи візуалізації.
- Обробка даних: Застосунок повинен надавати користувачу необхідні інструменти та можливості з обробки даних моніторингової системи для знаходження тенденцій, аномалій та закономірностей. Це включає в себе функції підрахунку індексів NDVI, NDMI та інших показників; фільтрацію та сортування даних.
- Звітність: Застосунок має вміти генерувати короткі інформативні звіти для спрощення користувачу процесів управління та прийняття рішень.

1.5.2. Вимоги до інформаційної безпеки

Для програмного забезпечення, яке локально зберігає файли і має змогу звертатися до мережі Інтернет, встановлюються наступні вимоги до інформаційної безпеки:

- Захист локально збережених файлів: Для запобігання несанкціонованого доступу до приватних даних, усі дані під час збереження та використання всередині додатку мають бути зашифровані.
- Безпека передачі даних мережею: Для запобігання різних видів мережевих атак під час передачі даних мережею необхідно використовувати зашифровані протоколи, наприклад, HTTPS, під час комунікації із віддаленим сервером.

1.5.3. Вимоги до складу та параметрів технічних засобів

Встановлюються наступні вимоги до складу та параметрів технічних засобів:

- Клієнтська машина, працююча на базі операційної системи Android 11 або вище, із широкосмуговим доступом до мережі Інтернет, для забезпечення безперервного зв’язку із віддаленим сервером.

1.5.4. Вимоги до інформаційної та програмної сумісності

До програмного продукту встановлюються наступні вимогу до інформаційної та програмної сумісності:

- Сумісність із програмним забезпеченням: Застосунок повинен бути сумісним та встановлюватись на операційній системі Android версії 11 та вище.
- Сумісність із форматами даних: Застосунок повинен мати змогу імпортувати дані з файлів у форматах XML, XLSX, JSON, CSV та TXT та експортувати дані у відповідних форматах
- Сумісність із хмарними сервісами: Застосунок повинен мати змогу завантажувати дані з сторонніх хмарних сервісів та на них.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Провівши аналіз існуючих програмних продуктів подібної тематики та галузі використання, робимо висновок, що функціональним призначенням розроблюваного застосунку є налаштування системи хмарних джерел інформації, для її подальшої візуалізації та аналізу.

Розглянемо основні функції програми:

1. Завантаження та зберігання даних: Застосунок працює сумісно із відкритою аналітичною хмарною платформою ThingSpeak, яка дозволяє користувачу зберігати дані, які були отримані з агротехнічних сенсорів та відправлені на хмарне середовище. Мобільний застосунок зчитує ці дані та зберігає локально для подальшої обробки.
2. Організація та управління інформаційною системою: Користувач має змогу вести облік своїх приладів для зчитування даних. Їх ідентифікаційні дані уводяться користувачем вручну, та зберігаються локально у сховищі даних на пристрої, для спрощення подальшого доступу.
3. Візуалізація та обробка даних: Застосунок дозволяє побачити зміну значення показників із часом у зручному форматі, такому як графіки та чарти. Окрім цього, додаток проводить базову статистичну обробку даних.

2.2. Опис застосованих математичних методів

Під час розробки даного програмного продукту були використані базові математичні та логічні оператори, такі як: додавання, множення та порівняння. Також використовувались статистичні методи із знаходження середнього значення вибірки, медіани, а також максимального та мінімального значення.

2.3. Опис використаної архітектури та шаблонів проектування

Технології розробки Android – застосунку включають в себе рекомендовану архітектуру, що включає в себе набір нестандартних шаблонів проектування [8]. На зображеній на рис 2.1. діаграмі можна побачити схему взаємодії елементів архітектури Android – застосунку один із іншим.

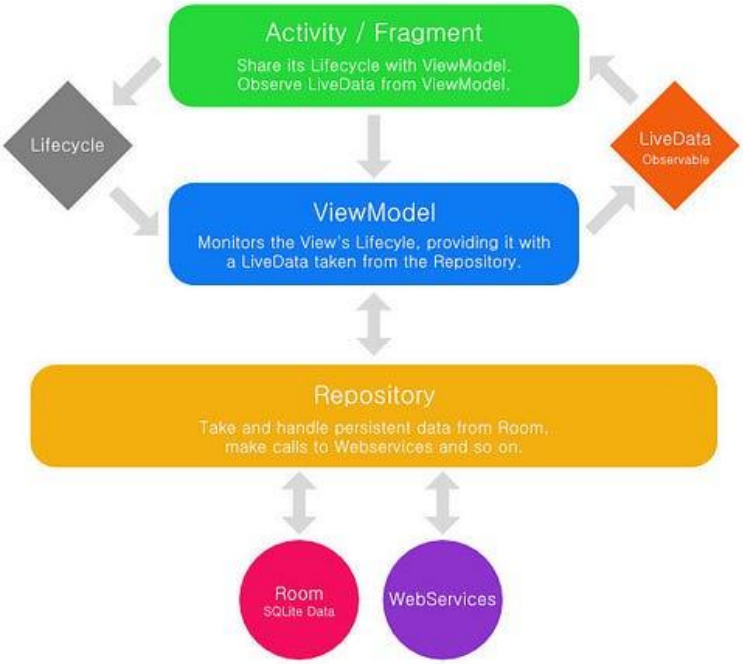

Рис. 2.1. Діаграма архітектури Android – застосунку

Не дивлячись на деякі відмінності, архітектура стандартної програми являє собою варіацію шаблону проектування MVVM (Model-View-ViewModel) [9], елементи якої розглянуті нижче:

4. View: Цей компонент відповідає за конструювання інтерфейсу користувача (UI), та таких його елементів як текстові блоки, кнопки, зображення, поля вводу даних та інші графічні елементи. Ці елементи не повинні бути пов’язані із бізнес-логікою застосунку, натомість беручі усі дані для відображення із компоненту-прошарку ViewModel. Для реалізації View у додатку використовуються класи Активностей (Activity) та Фрагментів (Fragments).

5. ViewModel [20]: Цей компонент архітектури не містить жодних залежностей від елементів відображення, а лише зберігає та оновлює в собі дані для інтерфейсу користувача, у зручному для View-елементів вигляді. Окрім цього, задачею цього прошарку є збереження цілісності та актуальності даних впродовж життєвого циклу застосунку. Для досягнення цих цілей може використовується механізм Прив'язки даних (Data Binding), але більш популярним на даний момент є використання шаблону проектування Спостерігач (Observer).

6. Model: Інкапсулює внутрішні моделі даних застосунку, їх бізнес логіку, математичні обчислення та валідацію. Класи цього рівня зазвичай взаємодіють із віддаленими сервісами або локальними сховищами даних. Так як дані до застосунку можуть надходити із різних джерел, та змінюватись впродовж життєвого циклу застосунку, для забезпечення стабільності та валідності даних прийнято використовувати шаблон проектування Репозиторій (Repository).

Розглянувши архітектуру застосунку, можемо проаналізувати шаблони проектування, які є складовими описаних вище елементів архітектури:

7. Шаблон проектування Спостерігач (Observer Pattern): Згідно із цим шаблоном, суб'єкт має список залежних від нього спостерігачів, які нотифікуються суб'єктом при кожній зміні даних, для проведення їх своєчасного оновлення [10]. У Android для цього використовується клас LiveData.

8. Шаблон проектування Repository: Виконує роль прошарку між бізнес логікою застосунку та сховищами даних (як віддалених так і локальних). Таким чином створюється додатковий рівень абстракції, що приховує деталі доступу та взаємодії із даними від інших компонентів системи. Це призводить до спрощення обслуговування, розуміння та редагування системи загалом, а також сприяє повторному використанню коду.

2.4. Опис використаних технологій та мов проектування

Для розробки мобільного застосунку були використані наступні новітні продукти інформаційних технологій, для забезпечення стабільної роботи програми, ефективного впровадження необхідного функціоналу у повному обсязі, а також забезпечення довгострокової підримки програмного продукту:

1. Kotlin [11]: Об’єктно-орієнтована крос-платформна мова програмування загального призначення та високого рівня абстракції із статичною типізацією даних, на якій написано програму. Kotlin працює на принципі, подібному до мови Java (використання JVM, як середи виконання), та за дизайном є повністю сумісною із мовою Java. Kotlin розробляється та підтримується компанією Google, завдяки чому має регулярні оновлення, покращення, інструментарій для роботи а також інтеграцію у IDE Android Studio. Володіє зручним синтаксисом, забезпечує захист від вказівників на порожні значення а також підтримує Лямбда-функції.

2. Android Studio [12]: Інтегроване Середовище Розробки (IDE) мобільних додатків для системи Android. Підтримує редагування коду із динамічною синтаксичною та логічною перевіркою у різних мовах, таких як Java, Kotlin, XML, SVG та інші. Має вбудовані технології зборки проектів та систему контролю версій, а також потужний емулятор віртуальних девайсів на системі Android.

3. SQLite: Вбудована безпосередньо у операційну систему Android система управління базою даних (СУБД). Надає простий та швидкий доступ до структурованих даних, що зберігаються у локальному внутрішньому середовищі девайсу. Володіє базовими захисними механізмами, такими як автентифікація застосунків за паролем, для захисту конфіденційної та особистої інформації.

4. SQL (Structured Query Language): Мова програмування, що використовується для збереження та обробки даних у реляційних базах даних [13]. Принцип роботи полягає у формуванні складних запитів до бази даних за

допомогою команд та операторів. Ці запити, при виконанні, безпечно та швидко модифікують базу даних або дані, що у ній зберігаються.

5. Git: Розподілена система контролю версій (VCS) [14], що надає розробникам програмного забезпечення набір можливостей, таких як підтримка декількох версій програмного коду одночасно, відстеження внесених до коду змін та версій, та можливість збереження програмних файлів віддалено у форматі репозиторію. Усі ці функції полегшують керування проектом, а також значно спрощують роботу фахівців у команді для більшої ефективності та швидкості розробки.

6. JSON (JavaScript Object Notation): Зручний формат передачі даних між програмними застосунками незалежно від їх мов програмування, який до того ж легко сприймається людиною. Стандарти формату два три види даних: об’єкт, масив та значення. Значення зберігаються у вигляді ключ-значення. Цих простих операторів достатньо, щоб описати будь яку структуру даних та передати її у текстовому вигляді, що робить цю технологію дуже ефективною та популярною.

7. GSON: Розроблена компанією Google бібліотека з відкритим вихідним кодом, що спрощує серіалізацію та десеріалізацію даних виду JSON (перетворення у формат JSON та з нього).

8. Room [15]: Object-Relational Mapper (ORM) – технологія, що налаштовує зв’язок баз даних із об’єктно орієнтовними мовами програмування. Вона робить це шляхом створення «Віртуальної бази даних», опис моделі якої проводиться у форматі об’єктів мови програмування.

9. Volley [16]: HTTP-бібліотека, розроблена компанією Google. Її призначенням є спрощення процесу проведення мережевих запитів та обробки їх результатів та підвищення ефективності та продуктивності у порівнянням із традиційними методами проведення подібних операцій. Основний функціонал цієї бібліотеки проаналізовано нижче:

- Асинхронний доступ до мереж: Volley виконує усі мережеві запити у асинхронному режимі, задля запобігання блокуванню

головного потоку програми (ще називається UI поток, який відповідає за графічний інтерфейс користувача застосунку).

- Автоматична обробка черги запитів: Бібліотека самостійно надає пріоритети запитам та організує їх в черги, що дозволяє оптимізувати пропускну здатність мережі та використання ресурсів девайсу.
- Вбудоване кешування: Volley надає механізми кешування даниї мережевих відповідей на запити, що покращує продуктивність застосунку в цілому, та полегшує отримання доступу до завантажених файлів та даних.

10. MPAndroidChart [17]: Безкоштовна бібліотека з відкритим вихідним кодом, що дозволяє візуалізувати дані у вигляді графіку в Android застосунку. Дана бібліотека має чисельні переваги над аналогами, серед яких:

- Підтримка створення стовпчастих, лінійних, секторних, кільцевих, точкових, бульбашкових та ін. діаграм.
- Дозволяє гнучке налаштування графіків, що відображаються, а саме такі елементи як кольори, шрифти, легенду діаграми, анімації, осі та інші візуальні елементи.
- Підтримка різних типів даних, від звичайних списків до власних комплексних типів об’єктів.
- Взаємодія із дотиками: Візуальні елементи, що надаються цією бібліотекою, оснащені обробкою дотиків, що дозволяють масштабування, наближення та навігацію графіком.

2.5. Опис структури програми та алгоритмів її функціонування

Структура мобільного застосунку для візуалізації та обробки даних агротехнічного моніторингу зображена на рис. 2.2.

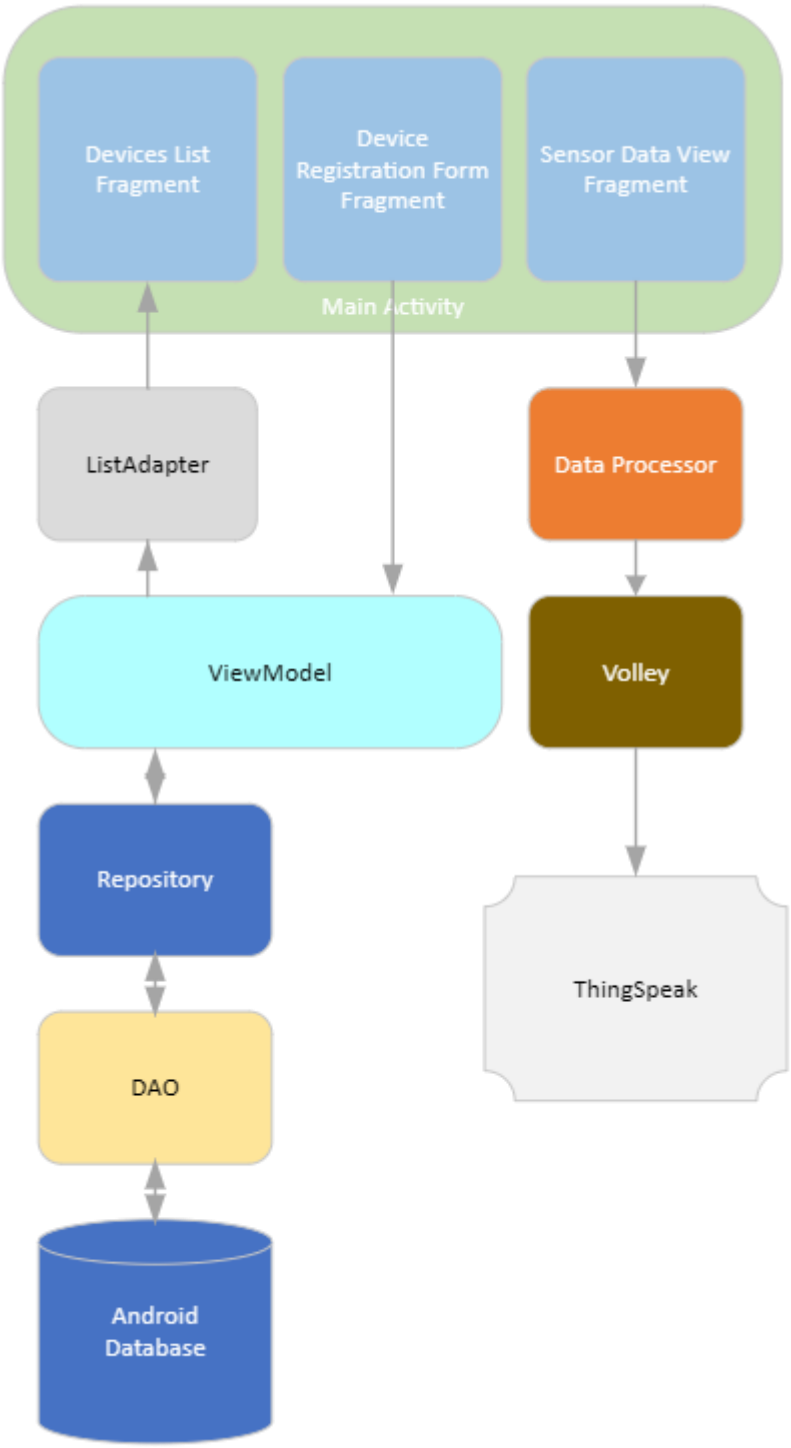

Рис. 2.2. Структура застосунку

Програма створена за архітектурою MVVM (Model-View-ViewModel), яка дозволяє розподілити обов'язки між головними класами програми та інкапсулювати методи та дані, які не потрібні певному шару застосунку. Розберемо кожний структурний елемент, та алгоритми, за які він відповідає:

1. Main Activity: Виконує роль представлення, та точки входу у програму. Головна активність відповідає за підтримання контексту застосунку та глобальних змінних і полів (Таких як ViewModel), а також надає користувачу навігацію додатком. Містить у собі три фрагменти: екрани з різною розміткою та функціональними кнопками:

- Device List Fragment – відображає користувачу поточний список зареєстрованих давачів інформації. Під час завантаження фрагменту, він підписує свій графічний елемент RecyclerView на ViewModel активності, таким чином при кожному оновлені даних графічний елемент оновлюється самостійно, завдяки класу Адаптер, як показано на рис. 2.3. Окрім цього, розмітка цього екрану має в собі навігаційний функціонал: при натисканні кнопки розмітка змінюється на екран реєстрації нового девайсу у систему, а при натисканні на елемент списку відкривається екран перегляду даних.

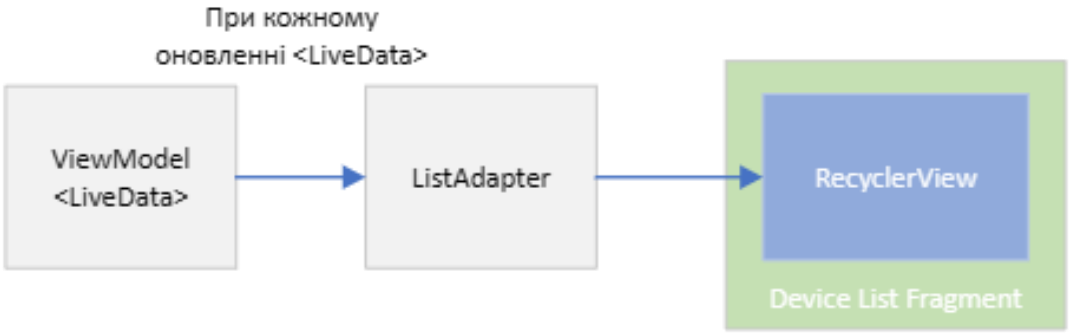

Рис. 2.3. Схема оновлення графічних даних із використанням класу ViewModel

- Device Registration Fragment – відображає користувачу форму реєстрації, після заповнення якої по натисканню функціональної кнопки відбувається валідація уведених даних, та у разі її проходження відбувається запит до ViewModel на реєстрацію нового девайсу до бази даних. Після цього екран повертається на попередній.

- Sensor Data View Fragment – При створені, фрагмент отримує параметри із даними обраного користувачем сенсору від List Fragment. Далі він конструює HTTPS запит та надсилає його до ViewModel, щоб отримати дані з хмарного середовища. Наступним кроком, клас DataProcessor проводить обробку даних, що дозволяє представленню розмістити графічні елементи для користувача у вигляді графіків, створених бібліотекою MPAndroidChart.

2. ViewModel: Прошарок між бізнес-логікою застосунку та представленнями. Як було описано вище, кожен екран представлення повинен звертатися до ViewModel для отримання або обробки даних. Таким чином, ViewModel виконує наступні функції:

- Взаємодія із базою даних – ViewModel звертається до репозиторію програми, із запитом на запис, видалення чи отримання даних
- Взаємодія із віддаленим сервером – ViewModel формує запит до віддаленого серверу, чекає на відповідь та оброблює її, надсилаючи представленню інформацію для відображення клієнту.
- Зберігання даних, що спостерігаються – View Model зберігає в собі дані виду Observable, які при зміні даних відсилають повідомлення зі змінами кожному підписнику.

3. Repository: Компонент, який відповідальний за зв’язок із базою даних. Завдяки методам бібліотеки Room, репозиторію достатньо звернутися до об’єкту DAO (Data Access Object), який зберігає в собі необхідні запити, написані мовою SQL.

4. Data Processor: Оброблює дані, отримані з ViewModel математичними та статистичними формулами перед їх візуалізацією у графіках.

Структура бази даних внутрішнього середовища операційної системи Android, до якого надається доступ за допомогою СУБД SQLite має вигляд, зображений на рис. 2.4.

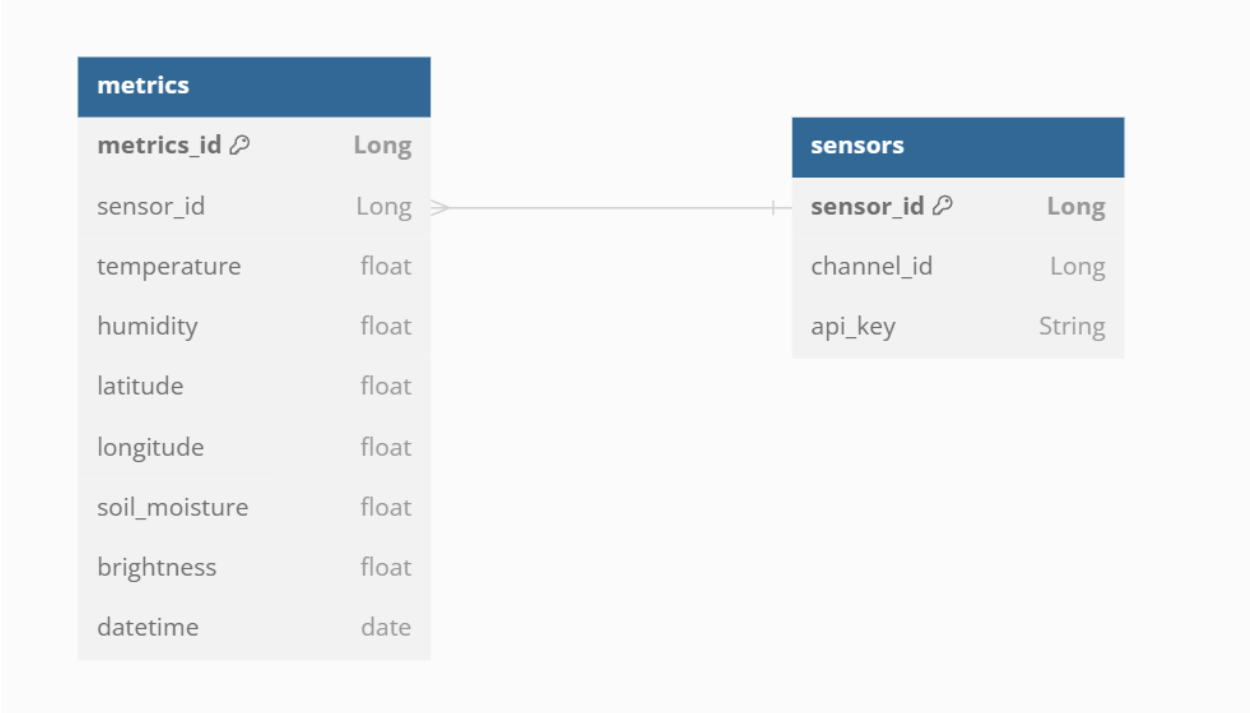

Рис. 2.4. Структура бази даних

Застосунку необхідно зберігати лише два типи даних у атомарному виді. Основним із них є тип зареєстрованого у додатку девайса, для якого створена таблиця “sensors”, та також додаток зберігає локально дані, що передаються з хмарного середовища у вигляді записів у таблиці “metrics”. Ці таблиці пов’язані між собою зовнішнім ключем таблиці “metrics”, яка зберігає ідентифікатор девайсу, до якого цей запис належить.

Таким чином, можемо виділити чотири основних алгоритми, які забезпечують ефективну, стабільну та зручну для користувача роботи програмного продукту:

1) Алгоритм навігації та візуалізації: Відповідно до базових потреб UX (User eXperience) [18] використовується алгоритм розміщення графічних елементів на розмітках екранів та реалізація методів переміщення між ними для клієнта

2) Алгоритм взаємодії з базою даних: Так як частина персональних даних, що використовуються при роботі застосунку, розташовані у базі даних, використовується алгоритм взаємодії з нею, для забезпечення стабільного та безпечного CRUD (Create Read Update Delete) [19].

3) Алгоритм завантаження даних з хмарного середовища: При отриманні даних з віддалених сервісів, використовується алгоритм завантаження даних, що дозволяє швидко та безпечно отримати інформацію для подальшої обробки.

4) Алгоритм обробки даних: Використовується для трансформації та обробки даних агротехнічного моніторингу.

Описані вище алгоритми необхідні для функціонування та забезпечення працездатності розроблюваного мобільного застосунку.

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними програми є записи про стан навколишнього середовища, зроблені сенсорними девайсами та передані на хмарне середовище. Ці записи включають у себе такі метрики як: температура та вологість повітря, вологість ґрунту, ширина та довгота (GPS data). Вони завантажуються у додаток під час його роботи. Окрім цього, вхідними даними вважаються зареєстровані через спеціальну форму користувачем давачі.

Вихідними даними програми являються:

- Графіки із візуалізацією зміни отриманих даних із часом,
- Оброблені алгоритмами дані,
- Згенеровані звіти.

2.7. Опис розробленого програмного продукту

2.7.1. Використані технічні засоби

Для розробки програмного забезпечення мобільного застосунку обробки даних агротехнічного моніторингу були використані наступні технічні засоби:

- Персональний комп’ютер на операційній системі Windows;
- Клавіатура та маніпулятор «Миша»;
- Мобільний девайс на операційній системі Android;

- Широкосмугове підключення до мережі Інтернет.

2.7.2. Використані програмні засоби

Для розробки мобільного застосунку було використано наступні програмні засоби:

- Веб-браузер Google Chrome: Для доступу до технічної документації та літератури;
- Інтегрована середа розробки (IDE) Android Studio: надає повний функціонал, який необхідний Android-розробнику. Дозволяє та допомагає писати зрозумілий та чистий код, вирішує логічні помилки, має розширений редактор XML файлів розмітки, має вбудовану підтримку інструментів управління та збирання проектів Maven та Gradle а також систему контролю версій.
- Android Virtual Device (AVD): Емулятор девайсу на базі системи Android. Необхідний для тестування програмного продукту під час його розробки. До того ж AVD інтегрується напряму до середовища розробки Android Studio, дозволяючи відлагоджувати програму та швидко встановлювати нові версії зборки.

2.7.3. Виклик та завантаження програми

Для запуску програмного продукту необхідно:

- У налаштуваннях мобільного девайсу на системі Android увімкнути дозвіл на встановлення сторонніх пакетів (Так як .apk файл не було підписано)
- Завантажити встановлюваний файл на систему
- Встановити застосунок з пакету .apk

2.7.4. Опис інтерфейсу користувача

При першому запуску програми користувач бачить порожній список зареєстрованих девайсів (рис. 2.5).

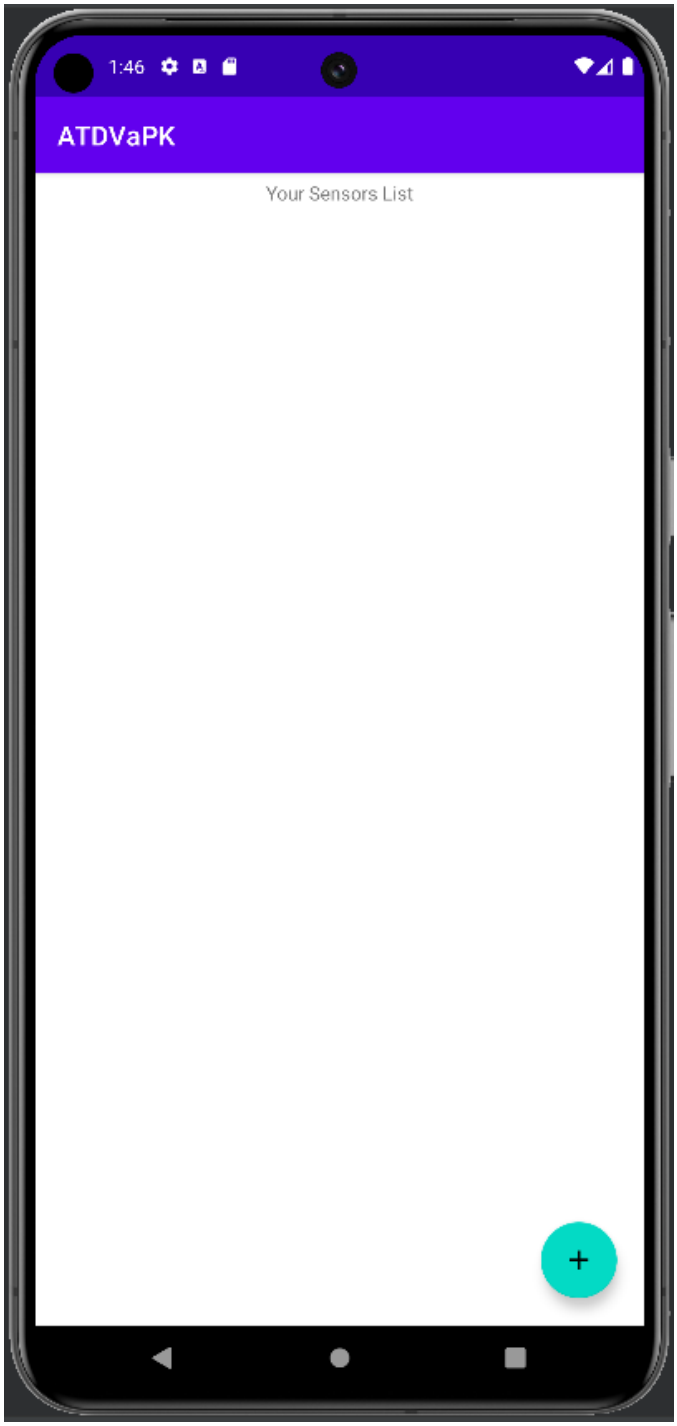

Рис. 2.5. Вигляд порожнього списку зареєстрованих сенсорів при першому запуску програми

При натисканні на кнопку у правому нижньому куті екрану, відкривається форма реєстрації нового давача, як показано на рис. 2.6.

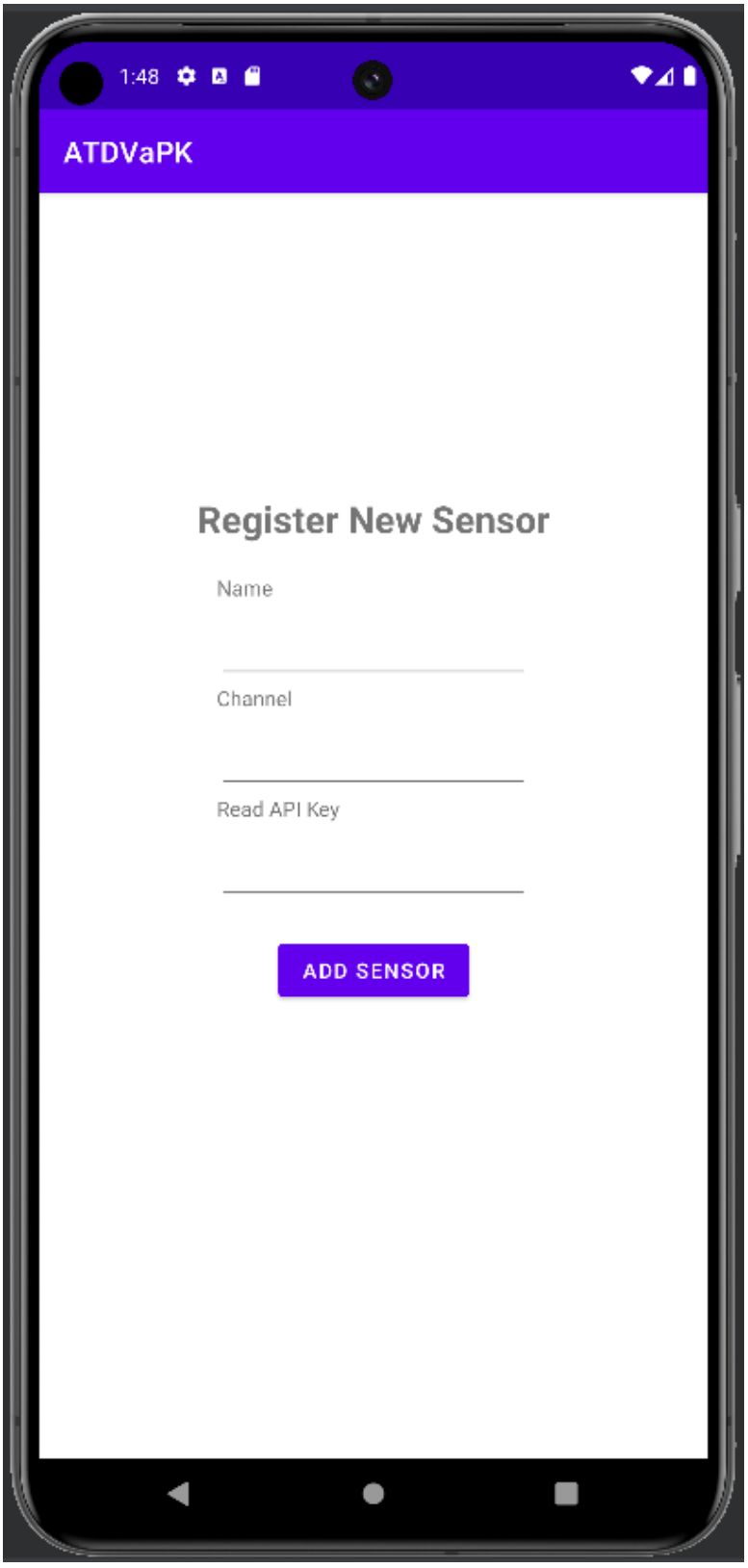

Рис. 2.6. Форма реєстрації нового девайсу

Форма реєстрації проводить валідацію уведених даних при натисканні кнопки «Додати Сенсор», що не дозволить користувачеві ввести некоректну інформацію. Помилку при валідації можна побачити на рис. 2.7.

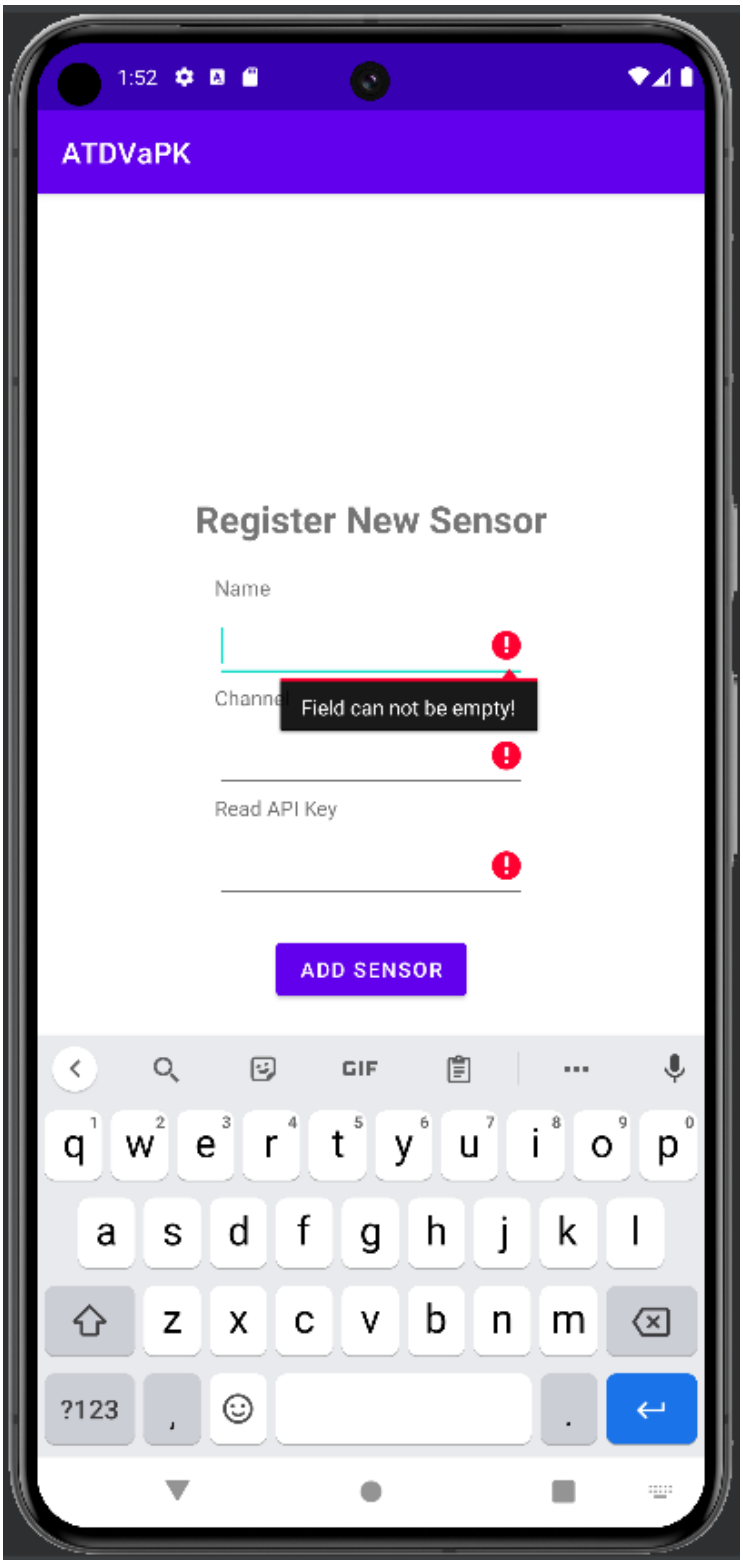

Рис. 2.7. Помилка при валідації через введення некоректних даних

Вигляд заповненого списку можна побачити на рис. 2.8.

Рис. 2.8. Заповнений список сенсорів

При необхідності видалити девайс із списку, можна зажати на нього (LongClick), у разі чого буде викликаний діалог для підтвердження, як зображено на рис. 2.9.

Рис. 2.9. Діалог із підтвердженням про видалення девайсу зі списку

При звичайному натисканні на девайс зі списку, відкриється екран візуалізації даних, як показано на рис. 2.10

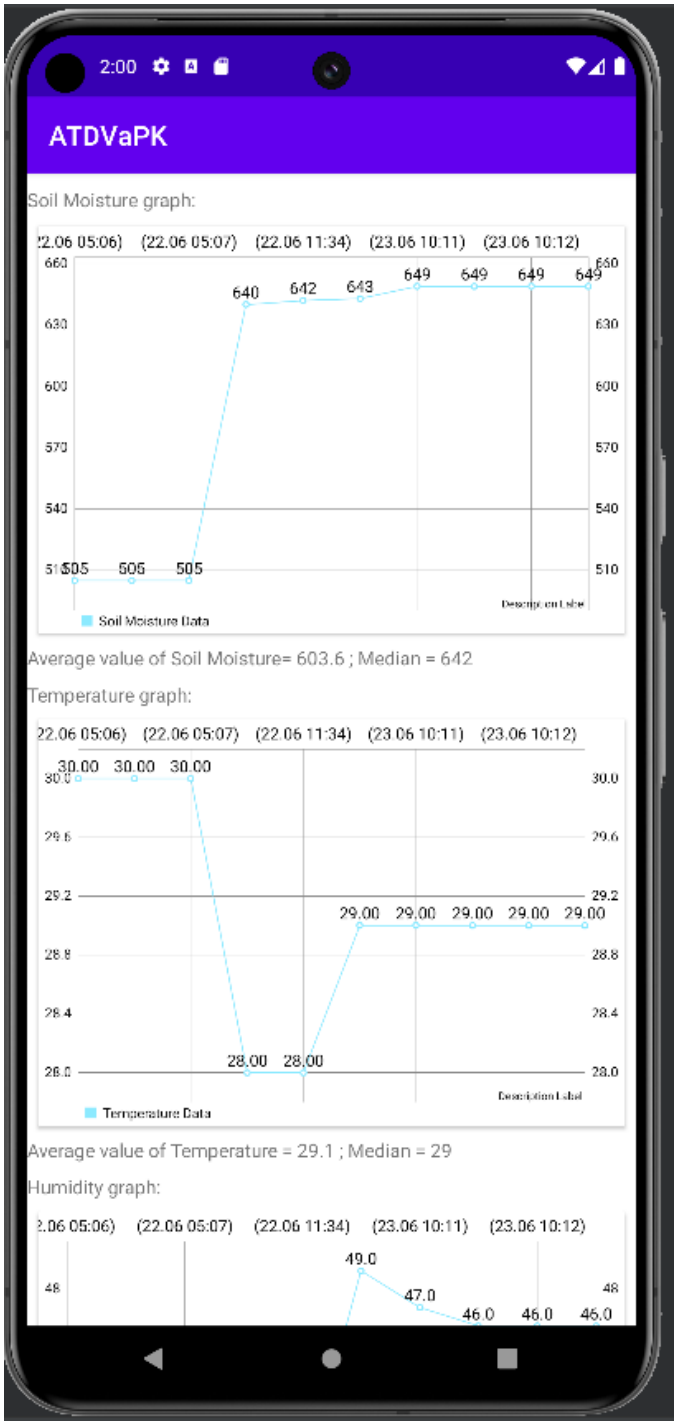

Рис. 2.10. Екран візуалізації даних сенсору

При необхідності змінити масштаб графіку (наприклад, при великій кількості записів), можна наблизити його, або віддалити за допомогою спеціального жесту, як показано на рис. 2.11.

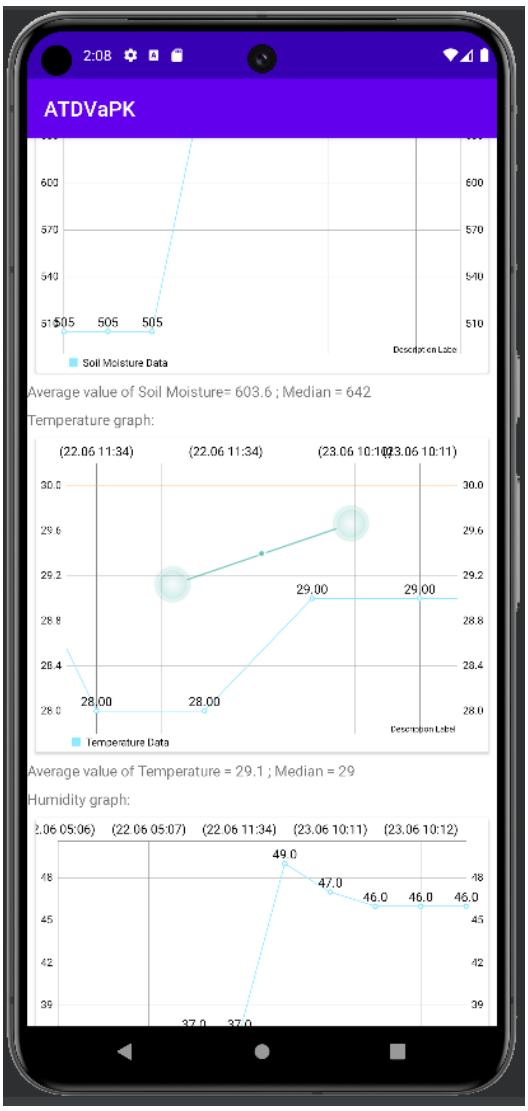

Рис. 2.11. Спеціальний жест для зміну масштабу графіку

Розроблений мобільний застосунок дозволяє користувачу організовувати та адмініструвати базу даних сенсорів агротехнічних даних, а також легко та швидко отримувати та візуалізувати оброблені дані зареєстрованих пристроїв у аналітичному вигляді.

Програмний продукт відповідає стандартам новітньої розробки мобільних застосунків на платформі Android: містить шаблонну розмітку та дизайн інтерфейсу користувача із використанням рекомендованих елементів, таких як FloatActionButton та список RecyclerView, впроваджує оптимальне використання функціоналу сучасних бібліотек для стабільної роботи, та використовує шаблон проектування MVVM для зв’язку візуального представлення із бізнес-логікою.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. Передбачуване число операторів програми: 1721;
2. Коефіцієнт складності програми: 1,4;
3. Коефіцієнт корекції програми в ході розробки: 0,3;
4. Годинна заробітна плата розробника – 198,34 грн/год.

За статистикою платформи найму “DOU”, на початок 2023 року медіана заробітної плати розробника програмного забезпечення рівня Junior складає 830$ на місяць [1]. У середньому, графік роботи розробника складається з 21 робочого дня по 8 годин. Отримуємо погодинну ставку, як 830$/(21 день * 8 годин) = 4,94$ / година. Станом на початок червня 2024 року, за офіційним курсом валют НБУ, один долар США дорівнює 40,15 грн [2]. Отже, розрахована погодинна ставка еквівалентна 198,34 грн. на годину.

5. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі: 1,3;
6. Коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності: 1;
7. Вартість машино-години ЕОМ: 4 грн/год;

Для розробки мобільного додатку використовувався персональний комп’ютер із широкосмуговим інтернет-підключенням. Вартість машино-години ЕОМ складається з вартості амортизації та обслуговування обладнання, вартості електроенергії та інтернет-підключення. Виходячи з характеристик ПК та монітору, дізнаємось, що приблизне споживання електроенергії за годину становить 300 Вт за годину. Згідно із тарифним планом постачальника електроенергії Yasno, 1 кВт/год коштує 4,32 грн [3]. Таким чином рахуємо

вартість машино-години, як 0,3 кВт/год * 4,32 грн/кВт = 1,3 грн/год. Щомісячна плата за підключення до інтернету від провайдера Kyivstar становить 200 грн. Отже, вартість робочої години інтернету = 200 грн / 168 год = 1,19 грн/год. Загальну вартість машино-години ЕОМ, враховуючи вартість амортизації та обслуговування, встановимо, як 4 грн/год.

Оцінка трудомісткості розробки програмного забезпечення ускладняється через творчий характер роботи програмістів. Для цього існує ряд моделей з різним ступенем точності.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

t_n-витрати праці на програмування по готовій блок-схемі;

$t_{отл}$-витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) \qquad (3.2)$$

де q - передбачуване число операторів (1721);

C - коефіцієнт складності програми (1,4);

p - коефіцієнт корекції програми в ході її розробки (0,3).

Отже, умовне число операторів в програмі має значення:

$$Q = 1721 \cdot 1{,}4 \cdot (1 + 0{,}3) = 3132{,}22$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \text{ , людино-годин} \qquad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі, враховуючи роботу із сторонніми джерелами даних він становить 1.3;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. Оцінюється в значення 1.

Підставивши значення у формулу (3.3), отримуємо результат:

$$t_u = \frac{3132{,}22 \cdot 1{,}3}{80 \cdot 1} = 50{,}9 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25) \cdot k} \text{ , людино-годин} \qquad (3.4)$$

де Q – умовне число операторів програми (3132,22);

k – коефіцієнт кваліфікації програміста (1).

Підставивши відповідні значення в формулу (3.4), отримуємо:

$$t_a = \frac{3132,22}{23 \cdot 1} = 136,18 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25) \cdot k} \text{ , людино-годин} \qquad (3.5)$$

Підставимо значення у формулу (3.5) та отримаємо витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{3132,22}{25 \cdot 1} = 125,29 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k} \text{ , людино-годин} \qquad (3.6)$$

$$t_{отл} = \frac{3132,22}{5 \cdot 1} = 626,44 \text{ людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1,5 \cdot t_{отл} \text{ , людино-годин} \qquad (3.7)$$

$$t^{k}_{отл} = 1,5 \cdot 626,44 = 939,67 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до} \text{ , людино-годин} \quad (3.8)$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ , людино-годин} \quad (3.9)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др} \text{ , людино-годин.} \quad (3.10)$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = = \frac{3132{,}22}{18 \cdot 1} = 174{,}01 \text{ людино-годин.}$$

$$t_{до} = 0{,}75 \cdot 174{,}01 = 130{,}5 \text{ людино-годин.}$$

$$t_д = 174{,}01 + 130{,}5 = 304{,}51 \text{ людино-годин.}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 50{,}9 + 136{,}18 + 125{,}29 + 626{,}44 + 304{,}51 = 1293{,}32 \text{ людино-години.}$$

3.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на

налагодження програми на ЕОМ:

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн} \quad (3.11)$$

Заробітна плата виконавців $З_{ЗП}$ визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн} \quad (3.12)$$

де: t - загальна трудомісткість, людино-годин;

$C_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 198,34 грн / год, отримуємо:

$$З_{ЗП} = 1293{,}32 \cdot 198{,}34 = 256\,517 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot C_{мч}, \text{ грн,} \quad (3.13)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (4 грн/год).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 626{,}44 \cdot 4 = 2505{,}76 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$K_{ПО} = 256\,517 + 2505{,}76 = 259\,022{,}76 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.} \quad (3.14)$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Очікуваний період створення ПЗ:

$$T = \frac{1293{,}32}{1 \cdot 176} \approx 7{,}34 \text{ міс.}$$

Висновок: Розробка мобільного додатку для візуалізації даних агротехнічного моніторингу потребує 1293,32 людино-години. З урахуванням курсу долару НБУ та медіани заробітної плати розробника ПЗ на момент написання роботи, для розробки знадобиться 259 022,76 грн. Запланований термін процесу розробки складає приблизно 7,34 місяця при роботі над проектом одного спеціаліста рівня Junior із тривалістю робочого часу не більше 40 годин на тиждень.

ВИСНОВКИ

У даній бакалаврській роботі було розроблено мобільний застосунок, який отримує дані із системи агротехнічного моніторингу, оброблює та візуалізує їх, а також зберігає на локальному пристрої. Застосунок має на меті допомогти землеробам приймати оптимальні рішення щодо управління своїми полями для підвищення продуктивності.

Мобільний застосунок має практичне застосування, оскільки дозволяє користувачам візуалізувати та обробляти дані важливих показників агротехнічного напряму, які напряму впливають на врожайність та продуктивність праці фахівців даної галузі.

Архітектура розробленого програмного продукту включає в себе найкращі практики створення надійних та високоякісних застосунків. Це включає в себе використання таких технологій та компонентів системи Android, як активності, фрагменти, файли розмітки, репозиторії та моделі. Основним шаблоном проектування, що використовувався під час розробки програмного забезпечення, є Model-View-ViewModel, який включає в себе використання шаблонів Observer для надання даних до представлень та Repository для зв’язку із базою даних.

Для розробки мобільного застосунку використовувалися мова розмітки XML для опису графічних елементів, та мова програмування загального призначення Kotlin для опису представлень та бізнес-логіки. У якості робочої середи було використане IDE Android Studio. Програмний продукт стабільно функціонує на девайсах із операційною системою Android 10 та вище.

Економічна частина роботи містить розрахунок трудомісткості та вартості розробки програмного забезпечення. Було визначено, що загальна трудомісткість проекту склала 1293,32 людино-годин, а загальні витрати на створення програмного продукту 259 022,76 грн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Agritech Startup Ecosystem in Ukraine: Ideas and Realization / URL https://link.springer.com/chapter/10.1007/978-981-16-2275-5_19 (дата звернення 02.05.2024)
2. Дослідження Kurkul.com 2023-го року стосовно використання агротехнічних технологій у комерції. / URL https://kurkul.com/news/32436-indormatsiyni-technologiyi-i-agropromislovosti / (дата звернення 02.05.2024)
3. FieldAlytics Enable introducing page / URL https://info.ever.ag/fieldalyticsenable.html (дата звернення 03.05.2024)
4. John Deere informational page / URL https://www.deere.ua/uk/campaigns/ag-turf/operations-center/ (дата звернення 03.05.2024)
5. AGMRI site home page / URL https://www.intelinair.com/ (дата звернення 03.05.2024)
6. Climate FieldView site home page / URL https://www.climatefieldview.com.ua/ (дата звернення 03.05.2024)
7. Precision Agriculture: Reviewing the Advancements, Technologies, and Applications in Precision Agriculture for Improved Crop Productivity and Resource Management / URL https://www.researchgate.net/publication/372766466_Precision_Agriculture_Reviewing_the_Advancements_Technologies_and_Applications_in_Precision_Agriculture_for_Improved_Crop_Productivity_and_Resource_Management (дата звернення 05.05.2024)
8. Android Application Architecture / URL https://medium.com/oceanize-geeks/android-application-architecture-189b4721c7c5 (дата звернення 16.06.2024)
9. Design Patterns and Architecture: The Android Developer Roadmap / URL https://getstream.io/blog/design-patterns-and-architecture-the-android-developer-roadmap-part-4/ (дата звернення 17.06.2024)

10. Observer pattern : Wikipedia / URL https://en.wikipedia.org/wiki/Observer_pattern (дата звернення 17.06.2024)

11. Kotlin docs / URL https://kotlinlang.org/docs/home.html (дата звернення 19.06.2024)

12. Android Studio home page / URL https://developer.android.com/studio (дата звернення 11.05.2024)

13. AWS article on what is SQL / URL https://aws.amazon.com/what-is/sql/?nc1=h_ls (дата звернення 19.06.2024)

14. Git home page / URL https://git-scm.com/ (дата звернення 20.06.2024)

15. Android documentation on Room library / URL https://developer.android.com/training/data-storage/room (дата звернення 20.06.2024)

16. Volley overview and documentation / URL https://google.github.io/volley/ (дата звернення 20.06.2024)

17. MPAndroidChart GitHub Repository / URL https://github.com/PhilJay/MPAndroidChart?tab=readme-ov-file#examples (дата звернення 20.06.2024)

18. Principles for User Experience: What We Can Learn from Bad Examples / URL https://www.researchgate.net/publication/278033606_Principles_for_User_Experience_What_We_Can_Learn_from_Bad_Examples (дата звернення 21.06.2024)

19. Guide to CRUD APIs / URL https://www.forestadmin.com/blog/an-experts-guide-to-crud-apis-designing-a-robust-one/ (дата звернення 22.06.2024)

20. ViewModel component overview / URL https://developer.android.com/topic/libraries/architecture/viewmodel (дата звернення 16.06.2024)

Додаток А

ЛІСТИНГ ПРОГРАМИ

MainActiviy.kt

```
package com.example.atdvapk

import android.os.Bundle
import android.os.PersistableBundle
import android.util.Log
import android.widget.Button
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.fragment.app.FragmentManager
import androidx.fragment.app.commit
import androidx.fragment.app.replace
import com.example.atdvapk.fragments.SensorListFragment

class MainActivity : AppCompatActivity() {

    // Late initialization of App's ViewModel object
    val viewModel by lazy {SensorViewModel(Dependencies.sensorRepository)}

    override fun onCreate(savedInstanceState: Bundle?) {
        // Initializing Database object and Repositories
        Dependencies.init(applicationContext)

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // Launching main visualization fragment with sensors list
        supportFragmentManager.commit{
            replace<SensorListFragment>(R.id.container)
        }
    }
}
```

SensorListFragment.kt

```
package com.example.atdvapk.fragments

import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.fragment.app.commit
import androidx.fragment.app.replace
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.atdvapk.Dependencies
import com.example.atdvapk.MainActivity
import com.example.atdvapk.R
import com.example.atdvapk.SensorListAdapter
import com.example.atdvapk.SensorViewModel
import com.example.atdvapk.databinding.FragmentSensorListBinding
import com.example.atdvapk.db.entities.SensorDbEntity
import com.google.android.material.floatingactionbutton.FloatingActionButton

class SensorListFragment : Fragment(R.layout.fragment_sensor_list) {
```

```
    private val TAG : String = "SensorListFragment"

    // Declaring variables for visual elements
    private lateinit var fabReg : FloatingActionButton
    private lateinit var mRecyclerView : RecyclerView
    private lateinit var mAdapter : SensorListAdapter

    private lateinit var mDataset : Array<SensorDbEntity>
    private var _binding: FragmentSensorListBinding? = null
    private val binding get() = _binding!!.

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }

    override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?,
        savedInstanceState: Bundle?): View? {

        _binding = FragmentSensorListBinding.inflate(inflater, container,
false)

        // Subscribing recyclerView element to ViewModel, using custom
adapter
        (activity as MainActivity).viewModel.allSensors.observe(activity as
MainActivity) { allSensor ->

            if(_binding != null) {
                val mAdapter =
                    SensorListAdapter((activity as
MainActivity).viewModel.sensorsItemListener)
                mAdapter.dataSet = allSensor.reversed()

                binding.recyclerView.adapter = mAdapter
                binding.recyclerView.layoutManager =
LinearLayoutManager(context)
            }
        }
        // Getting button view element and adding clickListener
        fabReg = binding.floatingActionButton
        fabReg.setOnClickListener{
            Log.i("FRAGMENT_REG","Launching")
// Launching registration fragment with appropriate animation
            activity?.supportFragmentManager?.commit {
                setCustomAnimations(R.anim.enter_right_to_left,
R.anim.exit_right_to_left,R.anim.enter_left_to_right,
R.anim.exit_left_to_right)
                addToBackStack(null)
                replace<RegistrationFragment>(R.id.container)
            }
        }

        val view = binding.root
        return view

    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

RegistrationFragment.kt

```
package com.example.atdvapk.fragments

import android.os.Bundle
import android.util.Log
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.EditText
import androidx.recyclerview.widget.RecyclerView
import com.example.atdvapk.Dependencies
import com.example.atdvapk.MainActivity
import com.example.atdvapk.R
import com.example.atdvapk.SensorViewModel
import com.example.atdvapk.databinding.FragmentRegistrationBinding
import com.example.atdvapk.databinding.FragmentSensorListBinding


class RegistrationFragment : Fragment(R.layout.fragment_registration) {

    private var _binding: FragmentRegistrationBinding? = null
    private val binding get() = _binding!!

// Initialization of input Views variables

    private lateinit var et_name : EditText
    private lateinit var et_chan : EditText
    private lateinit var et_apikey : EditText
    private lateinit var btn_reg : Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentRegistrationBinding.inflate(inflater, container,
false)
// Finding input elements and setting values to variables
        et_name = binding.txtName
        et_chan = binding.txtChan
        et_apikey = binding.txtKey
        btn_reg = binding.btnReg
// OnClickListener that checks the data validity and registrates new sensor
        btn_reg.setOnClickListener {
            if (isEditTextsEmpty()) {
// Calling ViewModel function to insert entry into database
                (activity as MainActivity).viewModel
.insertNewSensorInDatabase(
                    et_name.text.toString(),
                    et_chan.text.toString().toLong(),
et_apikey.text.toString()
                )
// Then return to previous fragment
                parentFragmentManager.popBackStack()
            }
        }
```

```
        val view = binding.root
        return view
    }
// Data validation function. Only checks for empty fields, as APIKeys and ID
//value formats can change with time
    private fun isEditTextsEmpty(): Boolean {
        val isName = if (et_name.text.toString().isNotBlank()) {
            et_name.error = null
            true
        } else {
            et_name.error = "Field can not be empty!"
            false
        }
        val isChan = if (et_chan.text.toString().isNotBlank()) {
            et_chan.error = null
            true
        } else {
            et_chan.error = "Field can not be empty!"
            false
        }
        val isApiKey = if (et_apikey.text.toString().isNotBlank()) {
            et_apikey.error = null
            true
        } else {
            et_apikey.error = "Field can not be empty!"
            false
        }

        return isName && isChan && isApiKey
    }
  }
```

SensorDataFragment.kt

```
package com.example.atdvapk.fragments

import android.os.Bundle
import android.util.Log
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import com.example.atdvapk.MainActivity
import com.example.atdvapk.databinding.FragmentSensorDataBinding
import com.github.mikephil.charting.charts.LineChart

import java.time.format.DateTimeFormatter

import com.android.volley.Request
import com.android.volley.RequestQueue
import com.android.volley.toolbox.JsonObjectRequest
import com.android.volley.toolbox.Volley
import com.github.mikephil.charting.data.Entry
import com.github.mikephil.charting.data.LineData
import com.github.mikephil.charting.data.LineDataSet
import com.github.mikephil.charting.formatter.IndexAxisValueFormatter
import java.time.ZoneId
import java.time.ZonedDateTime

// Initializing passed arguments names
private const val ARG_PARAM1 = "name"
private const val ARG_PARAM2 = "chanId"
```

```
private const val ARG_PARAM3 = "apiKey"

class SensorDataFragment : Fragment() {

// Declaring view elements variables
    private var name: String? = null
    private var chanId: Long? = null
    private var apiKey: String? = null

    private lateinit var textViewData: TextView
    private lateinit var textViewData2: TextView
    private lateinit var textViewData3: TextView

    private lateinit var textViewAnali: TextView
    private lateinit var textViewAnali2: TextView
    private lateinit var textViewAnali3: TextView

    private lateinit var lineChart: LineChart
    private lateinit var lineChart2: LineChart
    private lateinit var lineChart3: LineChart

    private lateinit var queue: RequestQueue

    private var _binding: FragmentSensorDataBinding? = null
    private val binding get() = _binding!!

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
// Retrieving passed values from list fragment
        arguments?.let {
            name = it.getString((ARG_PARAM1))
            chanId = it.getLong(ARG_PARAM2)
            apiKey = it.getString(ARG_PARAM3)
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentSensorDataBinding.inflate(inflater, container,
false)
// Getting visual elements from layout and setting their links to variables
        textViewData = binding.textViewData
        textViewData2 = binding.textViewData2
        textViewData3 = binding.textViewData3
        textViewAnali = binding.textViewAnal
        textViewAnali2 = binding.textViewAnal2
        textViewAnali3 = binding.textViewAnal3

        lineChart = binding.lineChart
        lineChart2 = binding.lineChart2
        lineChart3 = binding.lineChart3

// Forming a HTTPS request to thingSpeak server, using appropriate ID and
APIKey
        queue = Volley.newRequestQueue(activity)
        val url = "https://api.thingspeak.com/channels/" + chanId +
"/feeds.json?api_key=" + apiKey + "&results=10"
        Log.i("url",url)
        val request = JsonObjectRequest(Request.Method.GET, url, null, {
response ->
            val entriesTemp = mutableListOf<Entry>()
            val entriesHum = mutableListOf<Entry>()
```

```
            val entriesSM = mutableListOf<Entry>()
// Parsing data from response
            val feeds = response.getJSONArray("feeds")
            val indexAxisValForm = arrayListOf<String>()

            val tempVals = arrayListOf<Int>()
            val humVals = arrayListOf<Int>()
            val smVals = arrayListOf<Int>()
// Formatting data and structuring it into arrays to display them later
            for (i in 0 until feeds.length()) {
                val feed = feeds.getJSONObject(i)
                val zonedDateTime =
ZonedDateTime.parse(feed.getString("created_at"))
                //Adding formatted datetime field to entry

indexAxisValForm.add(zonedDateTime.withZoneSameInstant(ZoneId.systemDefault()
).format(DateTimeFormatter.ofPattern("(dd.MM hh:mm)")))
                val entryNo = i

                entriesTemp.add(Entry(i.toFloat(),
feed.getInt("field1").toFloat()))
                entriesHum.add(Entry(i.toFloat(),
feed.getInt("field2").toFloat()))
                entriesSM.add(Entry(i.toFloat(),
feed.getInt("field3").toFloat()))

                tempVals.add(feed.getInt("field1"))
                humVals.add(feed.getInt("field2"))
                smVals.add(feed.getInt("field3"))
            }
// Forming datasets for charts
            val dataSetTemp = LineDataSet(entriesTemp, "Temperature Data")
            val dataSetHum = LineDataSet(entriesHum, "Humidity Data")
            val dataSetSM = LineDataSet(entriesSM, "Soil Moisture Data")
// Setting up graphs
            dataSetTemp.setValueTextSize(12f)
            dataSetHum.setValueTextSize(12f)
            dataSetSM.setValueTextSize(12f)

            lineChart.xAxis.valueFormatter =
IndexAxisValueFormatter(indexAxisValForm)
            lineChart.extraTopOffset = 10f
            lineChart.data = LineData(dataSetSM)
            lineChart.getXAxis().setTextSize(12f);
            lineChart.invalidate()

            lineChart2.xAxis.valueFormatter =
IndexAxisValueFormatter(indexAxisValForm)
            lineChart2.extraTopOffset = 10f
            lineChart2.data = LineData(dataSetTemp)
            lineChart2.getXAxis().setTextSize(12f);
            lineChart2.invalidate()

            lineChart3.xAxis.valueFormatter =
IndexAxisValueFormatter(indexAxisValForm)
            lineChart3.extraTopOffset = 10f
            lineChart3.data = LineData(dataSetHum)
            lineChart3.getXAxis().setTextSize(12f);
            lineChart3.invalidate()
// Calculating analytics data such as average value and menian
            textViewAnali.text = "Average value of Soil Moisture= " +
findAverage(smVals) + " ; Median = " + findMedian(smVals)
            textViewAnali2.text = "Average value of Temperature = " +
findAverage(tempVals) + " ; Median = " + findMedian(tempVals)
```

```
            textViewAnali3.text = "Average value of Soil Moisture= " +
findAverage(humVals) + " ; Median = " + findMedian(humVals)

        }, { error ->
            // Handle error
            Log.e("RequestErr", "Data Not received")
        })

        queue.add(request)

        val view = binding.root
        return view
    }
// Methods for finding average and median values
    fun findAverage(numbers: List<Int>): Double {
        var sum = 0.0
        for (num in numbers) {
            sum += num
        }
        val average = sum / numbers.size
        return average
    }

    fun findMedian(list: List<Int>) = list.sorted().let {
        if (it.size % 2 == 0)
            (it[it.size / 2] + it[(it.size - 1) / 2]) / 2
        else
            it[it.size / 2]
    }
}
```

Dependencies.kt

```
package com.example.atdvapk

import android.content.Context
import androidx.room.Room
import com.example.atdvapk.db.AppDatabase
import com.example.atdvapk.db.MetricsRepository
import com.example.atdvapk.db.SensorRepository

// Static object that contains database instance and repository objects
object Dependencies {

    private lateinit var applicationContext: Context

    fun init(context: Context) {
        applicationContext = context
    }

    private val appDatabase: AppDatabase by lazy {
        Room.databaseBuilder(applicationContext, AppDatabase::class.java,
"database.db")
            .build()
    }

    val sensorRepository: SensorRepository by
lazy{SensorRepository(appDatabase.getSensorDao())}

    val metricsRepository: MetricsRepository by
lazy{MetricsRepository(appDatabase.getMetricsDao())}

}
```

SensorListAdapter.kt

```
package com.example.atdvapk

import android.app.AlertDialog
import android.app.Dialog
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.fragment.app.DialogFragment
import androidx.fragment.app.commit
import androidx.fragment.app.replace
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import com.example.atdvapk.databinding.SensorListItemBinding
import com.example.atdvapk.db.entities.SensorDbEntity
import com.example.atdvapk.fragments.RegistrationFragment
import com.example.atdvapk.fragments.SensorDataFragment

// Class that checks lists and updates LiveData if different
class SensorDiffUtil(
    private val oldList: List<SensorDbEntity>,
    private val newList: List<SensorDbEntity>
) : DiffUtil.Callback() {

    override fun getOldListSize(): Int = oldList.size
    override fun getNewListSize(): Int = newList.size
// Comparing list item's id
    override fun areItemsTheSame(oldItemPosition: Int, newItemPosition: Int):
Boolean {
        val oldItem = oldList[oldItemPosition]
        val newItem = newList[newItemPosition]

        return oldItem.id == newItem.id
    }
// Comparing list elements
    override fun areContentsTheSame(oldItemPosition: Int, newItemPosition:
Int): Boolean {
        val oldItem = oldList[oldItemPosition]
        val newItem = newList[newItemPosition]

        return oldItem == newItem
    }
}

interface SensorItemListener{
    fun getInfoAboutSensor(id: Long)
    fun removeSensor(id: Long)
}

class SensorListAdapter(private val sensorItemListener: SensorItemListener) :
    RecyclerView.Adapter<SensorListAdapter.SensorViewHolder>(),
View.OnClickListener {
// Initializing dataset list variable and its setter function
        var dataSet : List<SensorDbEntity> = emptyList()
            set(newValue) {
                val diffUtil = SensorDiffUtil(field, newValue)
                val diffUtilResult = DiffUtil.calculateDiff(diffUtil)
                field = newValue
```

```
                Log.e("FRAGMENT","IM CHANGEDDDD!" + dataSet.size)

                diffUtilResult.dispatchUpdatesTo(this@SensorListAdapter)
            }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
SensorViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val binding = SensorListItemBinding.inflate(inflater, parent, false)

        return SensorViewHolder(binding)
    }
// Fires for every created ViewHolder
    override fun onBindViewHolder(viewHolder: SensorViewHolder, position:
Int) {
// Setting visual data for elements
        viewHolder.binding.textView.text = dataSet[position].name
        viewHolder.binding.textViewID.text = "ID:" +
dataSet[position].channelId.toString()
        viewHolder.binding.textViewKey.text = "API Key:" +
dataSet[position].apiKey
// Adding clickListener that shows sensor delete dialog
        viewHolder.binding.root.setOnLongClickListener {
            DeleteSensorDialogFragment(viewHolder, dataSet[position].id).
            show((viewHolder.itemView.context as
MainActivity).supportFragmentManager, "DELETE_DIALOG")
            true
        }
// Adding clickListener that transfers user to data fragment
        viewHolder.binding.root.setOnClickListener{
            val fragment = SensorDataFragment()
            val bundle = Bundle()
// Passing parameters as Bundle
            bundle.putString("name", dataSet[position].name)
            bundle.putLong("chanId", dataSet[position].channelId)
            bundle.putString("apiKey", dataSet[position].apiKey)
            fragment.arguments = bundle
            (viewHolder.itemView.context as
MainActivity).supportFragmentManager?.commit {
                setCustomAnimations(R.anim.enter_right_to_left,
R.anim.exit_right_to_left,R.anim.enter_left_to_right,
R.anim.exit_left_to_right)
                addToBackStack(null)
                replace(R.id.container, fragment)
            }
        }
    }
    class DeleteSensorDialogFragment(viewHolder: SensorViewHolder, sid: Long)
: DialogFragment() {
        val viewHolder : SensorViewHolder = viewHolder
        val sid: Long = sid
        override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
            return activity?.let {
                val builder = AlertDialog.Builder(it)
                builder.setMessage("Delete Sensor: " +
viewHolder.binding.textView.text + "?")
                    .setPositiveButton("Delete") { dialog, id ->
// ViewModel request to delete entry from DB
                        (viewHolder.itemView.context as
MainActivity).viewModel.sensorsItemListener
                            .removeSensor(sid)
                    }
                    .setNegativeButton("Cancel") { dialog, id ->
                    }
```

```
                builder.create()
            } ?: throw IllegalStateException("Activity cannot be null")
        }
    }

    override fun getItemCount() : Int = dataSet.size

    class SensorViewHolder(val binding: SensorListItemBinding) :
RecyclerView.ViewHolder(binding.root)

}
```

SensorViewModel.kt

```
package com.example.atdvapk

import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.atdvapk.db.SensorRepository
import com.example.atdvapk.db.entities.Sensor
import com.example.atdvapk.db.entities.SensorDbEntity
import kotlinx.coroutines.launch

class SensorViewModel (private val sensorRepository: SensorRepository) :
ViewModel(){

    private val _allSensors = MutableLiveData<List<SensorDbEntity>>()
    val allSensors: LiveData<List<SensorDbEntity>> = _allSensors

    init{
        getAllSensors()
    }

    val sensorsItemListener = object : SensorItemListener{
        override fun getInfoAboutSensor(id: Long){
            val sensorId = _allSensors.value?.indexOfFirst { it.id == id }
            if(sensorId == -1) return

            val sensorInfo = _allSensors.value?.get(sensorId!!)
            println(sensorInfo)
        }
        override fun removeSensor(id: Long){
            viewModelScope.launch {
                sensorRepository.removeSensorDataById(id)
                getAllSensors()
            }
        }
    }
// Method that fetch metrics data from DB
    private fun getAllSensors(){
        viewModelScope.launch {
            _allSensors.value = sensorRepository.getAllSensorData()
            Log.e("DATABASE", "GETTING VALS")
        }
    }
// Method that insert Sensor entry to DB
    fun insertNewSensorInDatabase(name: String, chan: Long, apiKey: String) {
        viewModelScope.launch {
            val newSensor = Sensor(name, chan, apiKey)
```

```
sensorRepository.insertNewSensorData(newSensor.toSensorDbEntity())
            getAllSensors()
        }
    }

}
```

JsonDataTypes.kt

```
package com.example.atdvapk.JSON

import com.google.gson.annotations.SerializedName
import com.google.gson.internal.bind.DefaultDateTypeAdapter.DateType

class JsonDataTypes {

    data class Response(
        val channel : Channel,
        val feeds: ArrayList<Feeds>
    )
    data class Channel(
        val id : Long,
        val name : String,
        val description: String,
        val latitude: Float,
        val longitude: Float,
        val field1: String,
        val field2: String,
        val field3: String,
        val field4: String,
        val field5: String,
        val createdAt: String, // DATE!
        val updatedAt: String, // DATE!
        val lastEntryId: Long,
    )
    data class Feeds(
        val createdAt: String,
        val entryId: Long,
        val field1: Int,
        val field2: Int,
        val field3: Int,
        val field4: Float,
        val field5: Float
    )
}
```

AppDatabase.kt

```
package com.example.atdvapk.db

import androidx.room.Database
import androidx.room.RoomDatabase
import com.example.atdvapk.db.entities.MetricsDbEntity
import com.example.atdvapk.db.entities.SensorDbEntity

@Database(
    version = 1,
    entities = [
        SensorDbEntity::class,
        MetricsDbEntity::class
    ]
)
abstract class AppDatabase : RoomDatabase() {
```

```
    abstract fun getSensorDao(): SensorDao

    abstract fun getMetricsDao(): MetricsDao

}
```

Metrics.kt

```
package com.example.atdvapk.db.entities

data class Metrics(
    val sensorId: Long,
    val temperature: Float,
    val humidity: Float,
    val latitude: Float,
    val longitude: Float,
    val soilMoisture: Float,
    val brightness: Float,
    val datetime: Long
) {

    fun toMetricsDbEntity(): MetricsDbEntity = MetricsDbEntity(
        id = 0,
        sensorId = sensorId,
        temperature = temperature,
        humidity = humidity,
        latitude = latitude,
        longitude = longitude,
        soilMoisture = soilMoisture,
        brightness = brightness,
        datetime = datetime
    )
}
```

MetricsDbEntity.kt

```
package com.example.atdvapk.db.entities

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.ForeignKey
import androidx.room.Index
import androidx.room.PrimaryKey

@Entity(tableName = "metrics",
    indices = [Index("id")],
    foreignKeys = [
        ForeignKey(
            entity = SensorDbEntity::class,
            parentColumns = ["id"],
            childColumns = ["sensor_id"]
        )
    ])
data class MetricsDbEntity (
    @PrimaryKey(autoGenerate = true) val id: Long,
    @ColumnInfo(name = "sensor_id") val sensorId: Long,
    val temperature: Float,
    val humidity: Float,
    val latitude: Float,
    val longitude: Float,
    @ColumnInfo(name = "soil_moisture") val soilMoisture: Float,
    val brightness: Float,
```

```
    val datetime: Long
)
```

MetricsDao.kt

```
// DAO object that contains SQL querries to work with metrics table
package com.example.atdvapk.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.Query
import com.example.atdvapk.db.entities.MetricsDbEntity

@Dao
interface MetricsDao {

    @Insert(entity = MetricsDbEntity::class)
    fun insertNewMetricsData(metrics: MetricsDbEntity)

    @Query("SELECT * FROM metrics")
    fun getAllMetricsData(): List<MetricsDbEntity>

    @Query("DELETE FROM metrics WHERE id = :metricsId")
    fun deleteMetricsDataById(metricsId: Long)

}
```

MetricsRepository.kt

```
package com.example.atdvapk.db

import com.example.atdvapk.db.entities.MetricsDbEntity
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext

class MetricsRepository(private val metricsDao: MetricsDao) {

    suspend fun insertNewMetricsData(metricsDbEntity: MetricsDbEntity) {
        withContext(Dispatchers.IO) {
            metricsDao.insertNewMetricsData(metricsDbEntity)
        }
    }

    suspend fun getAllMetricsData(): List<MetricsDbEntity> {
        return withContext(Dispatchers.IO) {
            return@withContext metricsDao.getAllMetricsData()
        }
    }

    suspend fun removeMetricsDataById(id: Long) {
        withContext(Dispatchers.IO) {
            metricsDao.deleteMetricsDataById(id)
        }
    }

}
```

Sensor.kt

```
package com.example.atdvapk.db.entities

data class Sensor(
```

```
    val name: String,
    val channelId: Long,
    val apiKey: String
) {

    fun toSensorDbEntity(): SensorDbEntity = SensorDbEntity(
        id = 0,
        name = name,
        channelId = channelId,
        apiKey = apiKey
    )
}
```

SensorDbEntity.kt

```
package com.example.atdvapk.db.entities

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "sensors")
data class SensorDbEntity (
    @PrimaryKey(autoGenerate = true) val id : Long,
    val name: String,
    @ColumnInfo(name="channel_id") val channelId: Long,
    @ColumnInfo(name="api_key") val apiKey: String

)
```

SensorDao.kt

```
// DAO object that contains SQL querries to work with metrics table
package com.example.atdvapk.db

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.Query
import com.example.atdvapk.db.entities.SensorDbEntity

@Dao
interface SensorDao {

    @Insert(entity = SensorDbEntity::class)
    fun insertNewSensorData(sensor: SensorDbEntity)

    @Query("SELECT * FROM sensors")
    fun getAllSensorData(): List<SensorDbEntity>

    @Query("DELETE FROM sensors WHERE id = :sensorId")
    fun deleteSensorDataById(sensorId: Long)

}
```

SensorRepository.kt

```
package com.example.atdvapk.db

import com.example.atdvapk.db.entities.SensorDbEntity
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
```

```
class SensorRepository (private val sensorDao: SensorDao) {

    suspend fun insertNewSensorData(sensorDbEntity: SensorDbEntity) {
        withContext(Dispatchers.IO) {
            sensorDao.insertNewSensorData(sensorDbEntity)
        }
    }

    suspend fun getAllSensorData(): List<SensorDbEntity> {
        return withContext(Dispatchers.IO) {
            return@withContext sensorDao.getAllSensorData()
        }
    }

    suspend fun removeSensorDataById(id: Long) {
        withContext(Dispatchers.IO) {
            sensorDao.deleteSensorDataById(id)
        }
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/container"/>
```

activity_registration.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/loginscrn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="20dp"
        android:text="@string/text_register_sensor"
        android:textSize="25dp"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/inputHolder"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />
    <LinearLayout
        android:id="@+id/inputHolder"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginEnd="30dp"
        android:orientation="vertical"
        app:layout_constraintBottom_toTopOf="@+id/guideline"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/guideline">
        <TextView
```

```
            android:id="@+id/fstTxt"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:text="@string/text_register_name" />
        <EditText
            android:id="@+id/txtName"
            android:layout_width="wrap_content"
            android:layout_height="50dp"
            android:ems="10"
            android:labelFor="@string/label_add_fab"
            />
        <TextView
            android:id="@+id/secTxt"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:text="@string/text_register_chan" />
        <EditText
            android:id="@+id/txtChan"
            android:layout_width="wrap_content"
            android:layout_height="50dp"
            android:ems="10"
            android:inputType="number"
            android:labelFor="@string/label_input_channel" />
        <TextView
            android:id="@+id/trdTxt"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginBottom="5dp"
            android:text="@string/text_register_key" />
        <EditText
            android:id="@+id/txtKey"
            android:layout_width="wrap_content"
            android:layout_height="50dp"
            android:ems="10"
            android:labelFor="@string/label_input_channel" />
    </LinearLayout>
    <Button
        android:id="@+id/btnReg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="30dp"
        android:layout_marginTop="20dp"
        android:layout_marginEnd="30dp"
        android:text="@string/btn_register"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/inputHolder" />
    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_begin="366dp" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

fragment_sensor_data.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/frameLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
tools:context=".fragments.SensorDataFragment">
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <TextView
            android:id="@+id/textViewData"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="@string/text_sm" />
        <androidx.cardview.widget.CardView
            android:id="@+id/cardViewData"
            android:layout_width="match_parent"
            android:layout_height="300dp"
            android:layout_marginStart="8dp"
            android:layout_marginTop="10dp"
            android:layout_marginEnd="8dp">
            <com.github.mikephil.charting.charts.LineChart
                android:id="@+id/lineChart"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />
        </androidx.cardview.widget.CardView>
        <TextView
            android:id="@+id/textViewAnal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:text="TextView" />
        <TextView
            android:id="@+id/textViewData2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:text="@string/text_temp" />

        <androidx.cardview.widget.CardView
            android:id="@+id/cardViewData2"
            android:layout_width="match_parent"
            android:layout_height="300dp"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            android:layout_marginEnd="8dp">
            <com.github.mikephil.charting.charts.LineChart
                android:id="@+id/lineChart2"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />
        </androidx.cardview.widget.CardView>
        <TextView
            android:id="@+id/textViewAnal2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
```

```
                android:text="TextView" />
            <TextView
                android:id="@+id/textViewData3"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="8dp"
                android:text="@string/text_hum" />
            <androidx.cardview.widget.CardView
                android:id="@+id/cardViewData3"
                android:layout_width="match_parent"
                android:layout_height="300dp"
                android:layout_marginStart="8dp"
                android:layout_marginTop="8dp"
                android:layout_marginEnd="8dp">
                <com.github.mikephil.charting.charts.LineChart
                    android:id="@+id/lineChart3"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent" />
            </androidx.cardview.widget.CardView>
            <TextView
                android:id="@+id/textViewAnal3"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="8dp"
                android:text="TextView" />
        </LinearLayout>
    </ScrollView>
</androidx.constraintlayout.widget.ConstraintLayout>
```

fragment_sensor_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/floatingActionButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="20dp"
        android:layout_marginBottom="20dp"
        android:clickable="true"
        android:contentDescription="@string/label_add_fab"
        android:labelFor="@string/label_add_fab"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:srcCompat="@drawable/baseline_add_24" />
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="5dp"
        android:layout_marginTop="5dp"
        android:layout_marginEnd="5dp"
        android:text="@string/text_sensors_list"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <androidx.recyclerview.widget.RecyclerView
```

```
        android:id="@+id/recyclerView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="5dp"
        android:layout_marginTop="5dp"
        android:layout_marginEnd="5dp"
        android:layout_marginBottom="5dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView">
    </androidx.recyclerview.widget.RecyclerView>
</androidx.constraintlayout.widget.ConstraintLayout>
```

sensor_list_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:layout_marginLeft="5dp"
    android:layout_marginRight="5dp"
    android:gravity="center_vertical">
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="4dp"
        android:layout_marginTop="4dp"
        android:fontFamily="sans-serif-black"
        android:text="text"
        android:textSize="16sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/textViewID"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:layout_marginEnd="4dp"
        android:text="text"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/textViewKey"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="4dp"
        android:text="text"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Додаток Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

Додаток В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_Парастатов.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_Парастатов.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і встановчий архів
Презентація	
Презентація_Парастатов.pptx	Презентація кваліфікаційної роботи