Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Навчально-науковий інститут фізико-технічних та комп'ютерних наук
(повна назва інституту/факультету)

Кафедра інформаційних технологій та комп'ютерної фізики
(повна назва кафедри)

Інформаційна система для «розумного» будинку «SweeMe» (Клієнтська частина)

Кваліфікаційна робота

Рівень вищої освіти - перший (бакалаврський)

Виконав:
студент *4* курсу, групи *417ск*
спеціальності
126 Інформаційні системи та технології
(назва спеціальності)

Тритенко Юрій Миколайович
(прізвище та ініціали)
Керівник доктор фіз.-мат. наук, доцент.,
Борча Мар`яна Драгошівна
(науковий ступінь, вчене звання, прізвище та ініціали)

До захисту допущено:

Протокол засідання кафедри № 20

від „15" червня 2023 р.

зав. кафедри доц. Борча М. Д.

Чернівці – 2023

РЕЦЕНЗІЯ НА БАКАЛАВРСЬКУ РОБОТУ

Інформаційна система для «розумного» будинку «SweeMe». Клієнтська частина

Студента Тритенка Юрія Миколайовича

Інститут фізико-технічних та комп'ютерних наук

Спеціальність інформаційні системи та технології

Чернівецького національного університету імені Юрія Федьковича

Актуальність теми. «Розумний» будинок – це сучасний інструмент підвищення рівня комфорту і життя людини, а оскільки більшість процесів керується електронікою і відбувається автоматично, то це робить необхідним вивчення і вдосконалення таких технологій. Саме такі завдання вирішувались у дипломній роботі, що свідчить про її актуальність.

Практичне значення. Взаємодія між інформаційною системою та інтерфейсом користувача, зокрема, в інформаційній системі "розумного" будинку полягає у створенні зручного та ефективного способу взаємодії між людиною та системою управління будинком. Це сприяє поліпшенню особистого життя людини та підвищенню ефективності роботи інженерних систем.

Структура та зміст роботи. Робота складається з 3х розділів. У першому розділі описано принципи розробки та функціонування інтерфейсу користувача в інформаційній системі "розумного" будинку. Описано вимоги до системи та очікування від її впровадження. У другому розділі здійснено аналіз та вибір технологій для інформаційної системи "розумного" дому, було визначено оптимальний набір компонентів, що відповідають вимогам та потребам проєкту. Також було розглянуто функціонал для клієнтської частини та продемонстровано функціонал реєстрації користувача. У третьому розділі надана детальна інформація про реалізацію інформаційної системи і її взаємодію з інтерфейсом користувача. Продемонстрована робота клієнтської частини інформаційної системи, де показано основні можливості та спосіб взаємодії з користувачем.

Зауваження. Було б добре додати можливість сповіщення клієнта у критичних чи аварійних випадках роботи пристроїв системи.

Оцінка. В результаті виконання бакалаврської роботи студент Тритенко Ю.М. реалізував інтерфейс користувача для взаємодії з "розумним" будинком в інформаційній системі "SweeMe". Система передбачає здійснення налаштувань параметрів під потреби конкретного клієнта. Вважаю, що завдання бакалаврської роботи виконано повністю, а Тритенко Ю.М. заслуговує оцінки „добре" та присвоєння кваліфікації бакалавра зі спеціальності 126 – інформаційні системи та технології.

Рецензент
Асистент кафедри комп'ютерних систем та мережи
Чернівецького національного університету
Кандидат технічних наук
Одайська Христина Савеліївна

"14" 06 2023 р.

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра інформаційних технологій та комп фізики

ID перевірки:
1015666823

Дата перевірки:
21.06.2023 13:07:47 EEST

Тип перевірки:
Doc vs Internet + Library + DB

Дата звіту:
21.06.2023 13:08:23 EEST

ID користувача:
36471

Назва документа: Тритенко - 1

Кількість сторінок: 9 Кількість слів: 4401 Кількість символів: 36296 Розмір файлу: 59.15 KB ID файлу: 10153113(

18.6%
Схожість

Найбільша схожість: 13.5% з джерелом з Бібліотеки (ID файлу: 1015300663)

Не знайдено джерел з Інтернету

18.9% Джерела з Бібліотеки | 2 ..... Сторінка 11

3.18%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за 12 слів та 2%

0.2% Вилучення з Інтернету | 6 ..... Сторінка 11

3.18% Вилученого тексту з Бібліотеки | 115 ..... Сторінка 11

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 1

Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Навчально-науковий інститут фізико-технічних та комп'ютерних наук
(повна назва інституту/факультету)

Кафедра інформаційних технологій та комп'ютерної фізики
(повна назва кафедри)

Інформаційна система для «розумного» будинку «SweeMe». Клієнтська частина

Кваліфікаційна робота

Рівень вищої освіти - перший (бакалаврський)

Виконав:
студент *4* курсу, групи *417ск*
спеціальності
126 Інформаційні системи та технології
(назва спеціальності)

Тритенко Юрій Миколайович
(прізвище та ініціали)
Керівник доктор фіз.-мат. наук, доцент.,
Борча Мар`яна Драгошівна
(науковий ступінь, вчене звання, прізвище та ініціали)

До захисту допущено:

Протокол засідання кафедри № ____

від ,,____" _____ 2023 р.

зав. кафедри _____доц. Борча М. Д.

Чернівці – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Навчально-науковий інститут фізико-технічних та комп'ютерних наук
Кафедра комп'ютерних систем та мереж

ЗАТВЕРДЖУЮ
Завідувач кафедри
док. фіз.-мат. наук, доц.
_____М. Д. Борча
" ___" _____ 2023 р.

Інформаційна система для «розумного» будинку «SweeMe». Клієнтська частина

ЛИСТ ЗАТВЕРДЖЕННЯ

УЗГОДЖЕНО
Керівник роботи
докт. Фіз-мат. наук, доцент
_____М.Д. Борча
"___" _____ 2023 р.

Виконавець
студент 4-го курсу
_____ Ю. М. Тритенко
"___" _____ 2023 р.

2023

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Тритенку Юрію Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система для «розумного» будинку «SweeMe». Клієнтська частина*
керівник роботи *Борча Мар`яна Драгошівна док. фіз.-мат. наук, ст.н.сп.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від "__"_____202_ року №_____

2. Строк подання студентом проекту (роботи) *2023 р.*

3. Вихідні дані до проекту (роботи) Мета роботи – розробити інформаційну систему «розумного» дому "SweeMe" та специфічної частини цієї системи – розробка клієнтської частини. Система повинна мати такі вікна для зручної роботи з програмою: реєстрації та авторизації користувача, головний екран програми, вікно додавання пристроя до акаунту користувача, мати вікно в якому користувач зможе переглянути статистику та данні пристроя та ще мати вікно для считування Qr-коду для присвоення пристрою конкретному користувачу. Програмне забазпечення розробити на мові Kotlin з використанням серидовища Android Studio

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) дослідити існуючі інформаційні системи-аналоги
2) описати інтерфейс користувача таких сторінок: реєстрації, авторизації, додавання пристрою, перегляд *даних з пристрою та сканування Qr-коду*
3) розробити прототип інтерфейсу
4) розробити інтерфейс користувача з усім необхідним функціоналом
5) дослідити ефективність роботи розробленої програми

5. Перелік графічного матеріалу
1) Інтерфейс користувача

Студент _____ *Тритенко Ю.М*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Борча М.Д*
(підпис) ***(прізвище та ініціали)***
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота виконана студентом групи 417ск Тритенком Юрієм Миколайовичем. Тема «Інформаційна система для «розумного» будинку «SweeMe» (Клієнтська частина)». Робота направлена на здобуття ступеня бакалавр за спеціальністю 126 «Інформаційні системи та технології».

Метою кваліфікаційної роботи є розробка інформаційної системи «розумного» дому "SweeMe" та специфічної частини цієї системи - розробка клієнтської частини. Програмна реалізація системи виконана на мові Kotlin із використанням середовища AndroidStudio.

Бакалаврська робота містить: кількість сторінок – 59, рисунків – 25, додатків – 1, використаних джерел – 12.

Ключові слова: інформаційна система, клієнтська частина, інтерфейс користувача, Retrofit, Kotlin, AndroidStudio.

Робота містить результати власних досліджень. Використання чужих ідей, результатів і текстів мають посилання на відповідне джерело.

ANNOTATION

The qualification work was carried out by student Yuriy Mykolayovych Trytenko from group 417sk. The topic of the work is "Information System for the 'Smart' Home 'SweeMe' (Client-side)". The work is aimed at obtaining a bachelor's degree in the field of specialization 126 "Information Systems and Technologies".

The goal of the qualification work is to develop an information system for the "Smart" home "SweeMe" and the specific part of this system - the development of the client-side. The software implementation of the system is done in Kotlin language using the AndroidStudio environment.

The bachelor's work includes: number of pages - 59, figures - 25, appendices - 1, used sources - 12.

Keywords: information system, client-side, user interface, Retrofit, Kotlin, AndroidStudio.

The work contains the results of the author's own research. The use of other people's ideas, results, and texts is referenced to the respective source.

ЗМІСТ

ВСТУП

Метою даної кваліфікаційної роботи є розробка інформаційної системи «розумного» дому "SweeMe" та специфічної частини цієї системи - розробка інтерфейсу та взаємодії з API.

Мета дослідження полягає в розробці ефективної і надійної інформаційної системи для керування розумним будинком, що відповідає зростаючій популярності інтелектуальних систем у побутовій сфері. Така система має важливе значення в контексті зростання вартості електроенергії, оскільки дозволяє ефективно економити електроенергію та надає повний контроль та можливість віддаленого керування пристроями.

Основні завдання дослідження включають аналіз предметної області, проектування інформаційної системи та вибір відповідного інструментарію та методів для реалізації цієї системи. Аналіз предметної області передбачає вивчення сучасних технологій та рішень у галузі "розумних" будинків, а також виявлення основних потреб і вимог користувачів. На основі цього аналізу буде проведене проектування інформаційної системи "SweeMe", яка включатиме розробку інтерфейсу користувача для зручної взаємодії з розумним будинком.

Основна ідея полягає в тому, що пристрої, які використовуються в розумному будинку, можуть бути розроблені спеціально під вимоги клієнта, а їх взаємодія та параметри можуть бути налаштовані окремо залежно від потреб користувача. Таким чином, інформаційна система "SweeMe" буде забезпечувати індивідуалізований підхід до керування розумним будинком, що відповідає потребам кожного користувача.

Загалом, розробка такої інформаційної системи має великий потенціал для забезпечення зручності, економії електроенергії та вдосконалення контролю над розумним будинком.

Завдання дослідження включають:

- Аналіз предметної області, дослідження потреб користувачів розумного будинку та визначення необхідного функціоналу для інтерфейсу користувача.
- Проектування інформаційної системи, включаючи створення концептуальної, логічної та фізичної моделей бази даних для збереження інформації про користувачів та пристроїв.
- Вибір та використання необхідних інструментів та методів для реалізації інтерфейсу користувача.
- Розробка функціоналу та інтерфейсу для реєстрації та авторизації користувачів, управління пристроями.

Програмна реалізація системи виконана мовою Kotlin за допомогою середовище програмування Android Studio.

У результаті виконання бакалаврської роботи розроблено інформаційну систему та інтерфейс користувача для взаємодії з API, яка призначена для системи «розумного» будинку.

Об'єктом дослідження є інформаційна система «розумного» дому "SweeMe".

Предметом дослідження є технології проектування та розробки інформаційних систем, методики аналізу та оцінки інформаційних систем, а також використання популярних технологій для розробки інтерфейсу користувача.

Розділ 1 ІНФОРМАЦІЙНА СИСТЕМА ТА ПРИНЦИП ВЗАЄМОДІЇ ВСІХ КОМПОНЕНТІВ З "РОЗУМНИМ" БУДИНКОМ

1.1 Аналіз об’єкту дослідження та предметної області

1.1.1 Теоретичні відомості про об’єкт дослідження та предметну область

Інформаційна система є складним комплексом взаємопов'язаних елементів, включаючи апаратне та програмне забезпечення, бази даних, мережеві зв'язки та людські ресурси. Ці компоненти працюють разом для збирання, зберігання, обробки, передачі та використання інформації з метою підтримки процесів прийняття рішень, виконання завдань та досягнення конкретних цілей. Інформаційні системи виконують функції обробки та передачі даних, забезпечують взаємодію з користувачами і сприяють полегшенню різноманітних завдань, що призводять до покращення ефективності управління в різних галузях діяльності. Застосування інформаційних систем допомагає вдосконалити процеси управління, сприяє оптимізації роботи та забезпечує більш точне та швидке прийняття рішень.

Основними складовими інформаційної системи є:

- Апаратне забезпечення включає фізичні пристрої, такі як комп'ютери, сервери, мережеві пристрої, сенсори і засоби зберігання даних. Ці пристрої використовуються для виконання обчислень, обробки і зберігання інформації.

- Програмне забезпечення: це програми та системи, які встановлюються на апаратному забезпеченні для виконання певних функцій, такі як операційні системи, бази даних, програми обробки даних, додатки для взаємодії з користувачами та інші спеціалізовані програми.

- Бази даних: це структури для організованого зберігання великого обсягу даних. Бази даних забезпечують доступ, пошук, модифікацію та управління даними, які використовуються в інформаційній системі.

- Мережеві зв'язки: це інфраструктура, яка дозволяє передавати дані та забезпечує зв'язок між різними компонентами системи. Мережеві зв'язки можуть включати локальні мережі (LAN), бездротові мережі (Wi-Fi, Bluetooth), глобальну мережу Інтернет та інші комунікаційні канали.

- Людські ресурси: це фахівці, які взаємодіють з інформаційною системою, такі як адміністратори, програмісти, користувачі та інші працівники. Людські ресурси забезпечують конфігурацію, розробку, впровадження, підтримку та використання інформаційної системи.

Ці складові взаємодіють між собою для забезпечення збору, зберігання, обробки, передачі та використання інформації в межах інформаційної системи. Кожен з цих елементів відіграє важливу роль у функціонуванні та ефективності системи в цілому.

Перед початком розробки інтерфейсу користувача необхідно ознайомитися з деякими термінами:

1. UI (User Interface) - це спосіб, яким користувач взаємодіє з інформаційною системою або програмним додатком. Включає в себе всі елементи, з якими користувач може взаємодіяти, такі як кнопки, меню, форми введення даних, списки, зображення та інші візуальні елементи. Інтерфейс користувача повинен бути зрозумілим, легким у використанні та забезпечувати зручну навігацію, щоб користувач міг ефективно взаємодіяти з системою.

2. UX (User Experience) - це враження, яке користувач отримує під час взаємодії з інформаційною системою або програмним додатком. UX охоплює всі аспекти взаємодії користувача, включаючи візуальний дизайн, функціональність, легкість використання, продуктивність та задоволення, яке користувач отримує від використання системи. Головною метою UX є створення задоволення та позитивного враження у користувачів, сприяючи вирішенню їхніх потреб і досягненню поставлених цілей.

Узгоджена робота між UI та UX допомагає створити зручну та задоволену користувачем взаємодію з інформаційною системою або програмним додатком, покращуючи їхню використовуваність та ефективність.

Для моделювання інтерфейсу користувача використовують Figma. Figma - це онлайн-інструмент для дизайну і прототипування, який дозволяє створювати користувальницький інтерфейс (UI) для веб-сайтів, мобільних додатків та інших цифрових продуктів. Він надає можливості спільної роботи в режимі реального часу, що дозволяє дизайнерам, розробникам та іншим зацікавленим сторонам працювати над проектами, коментувати та вносити зміни одночасно. Figma має візуальний редактор, який дозволяє створювати векторні графічні елементи, інтерфейси та макети. Він надає широкий набір інструментів для малювання, редагування, групування та організації різних шарів і компонентів дизайну. Figma також підтримує стилі, компоненти та бібліотеки, що дозволяє створювати і використовувати повторювані елементи і автоматично оновлювати їх у всіх макетах, де вони використовуються.

Для реалізації інтерфейсу користувача використовується середовище розробки Android Studio та мова програмування Kotlin.

Kotlin - це статично типізована мова програмування, яка працює на платформі Java. Вона була розроблена компанією JetBrains з метою створення сучасної, експресивної та безпечної мови програмування для розробки програмного забезпечення на базі JVM (Java Virtual Machine), а також для розширення можливостей розробки на платформі Android. Kotlin поєднує в собі переваги об'єктно-орієнтованого та функціонального програмування, надаючи розробникам потужні інструменти для створення ефективного та зрозумілого коду. Вона має читабельний синтаксис, спрощує роботу з нульовими значеннями (null safety) та надає вбудовану підтримку асинхронного програмування.

Android Studio - це інтегроване середовище розробки (IDE) для платформи Android, створене компанією Google. Воно надає розробникам зручний набір інструментів для створення, тестування, налагодження та розгортання мобільних додатків для операційної системи Android. Android Studio базується на відкритому проекті IntelliJ IDEA і надає розширені можливості, спеціально налаштовані для розробки Android-додатків. Воно включає в себе інструменти, такі як візуальний редактор макетів, редактор коду, компілятор, дебагер, профайлер, емулятори Android-пристроїв та багато іншого.

1.1.2 Принцип взаємодії інформаційної системи та інтерфейсу користувача з «розумним» будинком

Принцип взаємодії між інформаційною системою та інтерфейсом користувача в "розумному" будинку полягає у створенні зручного та ефективного способу взаємодії між людиною та системою управління будинком. Цей принцип охоплює різні аспекти, включаючи комунікаційні протоколи, засоби керування та візуалізацію інформації.

По-перше, інформаційна система розумного будинку повинна мати здатність взаємодіяти з різними компонентами будинку, такими як освітлення, система опалення, побутові пристрої і т.д. Це досягається шляхом підключення до API або інших протоколів взаємодії.

Другий аспект - це інтерфейс користувача, який є інтерактивним інструментом взаємодії між користувачем та системою "розумного" будинку. Інтерфейс користувача може бути реалізований у вигляді мобільного додатку для забезпечення більшої зручності та мобільності.

Основна мета інтерфейсу користувача - зробити взаємодію з "розумним" будинком максимально зручною та доступною для користувача. Це можна досягти шляхом простоти використання, інтуїтивно зрозумілих елементів керування та зручного представлення інформації. Крім того, інформаційна

система може надавати користувачу розширені можливості управління та контролю. Наприклад, вона може надавати статистику температури протягом певного періоду часу.

Загалом, принцип взаємодії між інформаційною системою та інтерфейсом користувача в "розумному" будинку передбачає створення зручної та інтуїтивно зрозумілої системи, яка дозволяє користувачу легко керувати різними аспектами свого будинку та отримувати необхідну інформацію.

1.2. Постановка задачі

1.2.1 Вимоги до системи

Розроблена інформаційна система включає інтерфейс користувача, щоб забезпечити зручну взаємодію з "розумним" будинком. Цей інтерфейс відповідає певним вимогам:

1. Простота використання: Інтерфейс має бути легким у навігації та зрозумілим. Користувач повинен швидко засвоїти, як взаємодіяти з системою та виконувати необхідні дії без заплутування.

2. Інтуїтивність: Інтерфейс використовує знайомі символи, іконки та мову, щоб користувачам було зрозуміло, що робити і як керувати системою. Він враховує загальноприйняті стандарти та конвенції в дизайні інтерфейсу.

3. Швидкість: Інтерфейс повинен працювати швидко та реагувати миттєво. Користувач повинен мати можливість швидко здійснювати дії та отримувати негайний зворотний зв'язок від системи.

4. Інформативність: Інтерфейс надає користувачу достатньо інформації про стан системи, статус пристроїв та режими роботи. Це допомагає користувачу розуміти, що відбувається у будинку та приймати обґрунтовані рішення.

5. Сумісність з різними пристроями: Інтерфейс має бути сумісним з різними пристроями, що використовуються для керування "розумним" будинком.

Він може бути доступний через мобільні додатки, веб-браузери або спеціальні панелі управління.

Інтерфейс користувача був реалізований за допомогою мови програмування Kotlin у середовищі розробки Android Studio.

1.2.2 Очікування від впровадження системи

Впровадження інформаційної системи та інтерфейсу користувача для взаємодії з віддаленими пристроями у розумному будинку спрощує керування всіма його функціями через мобільний пристрій. Цей зручний і простий спосіб керування полегшує повсякденне життя мешканців, надаючи комфорт і зручність. Використання мови програмування Kotlin під час розробки інформаційної системи дозволяє легко додавати новий функціонал до існуючого інтерфейсу користувача. Розробники можуть створювати власні додатки або інтегрувати систему розумного будинку з іншими системами, такими як розумне місто або розумна енергетика. Це створює можливості для інноваційних рішень і дослідження нових способів оптимізації енергоспоживання та ефективного управління ресурсами.

Висновки до розділу 1

В першому розділі були викладені теоретичні знання про інформаційну систему та описано принцип взаємодії всіх компонентів з "розумним" будинком. Також була поставлена задача з описом вимог до системи та очікувань від її впровадження.

Після аналізу наявних аналогів на ринку ми вирішили створити власну інформаційну систему для розумного будинку. Таке рішення було прийняте на основі вартості існуючих аналогів. Розробка власної інформаційної системи допоможе суттєво знизити витрати і дозволить людям з обмеженими фінансовими можливостями отримати доступ до розумного будинку.

Наша система може не мати такого ж рівня якості, як деякі з існуючих аналогів, але вона має потенціал для майбутнього розвитку. Власна система забезпечить гнучку адаптацію до потреб користувачів і надасть можливість розширювати функціональність системи та розробляти власні пристрої, що стає важливою перевагою в умовах швидкого технологічного прогресу.

Для розробки інформаційної системи було обрано мову програмування Kotlin та середовище розробки Android Studio для реалізації інтерфейсу користувача. Kotlin є швидко зростаючою мовою програмування, яка вже стала стандартом для розробки програм під Android. Android Studio надає розробникам зручні та гнучкі інструменти для створення інтерфейсу користувача. Використання Kotlin як мови програмування забезпечить розробникам необхідні інструменти для створення потрібного функціоналу відповідно до потреб користувача.

РОЗДІЛ 2 ПОБУДОВА СИСТЕМИ

2.1 Моделювання системи

2.1.1 Функції та структура системи

Структура інформаційної системи складається із декількох таблиць в базі даних (рис. 2.1).

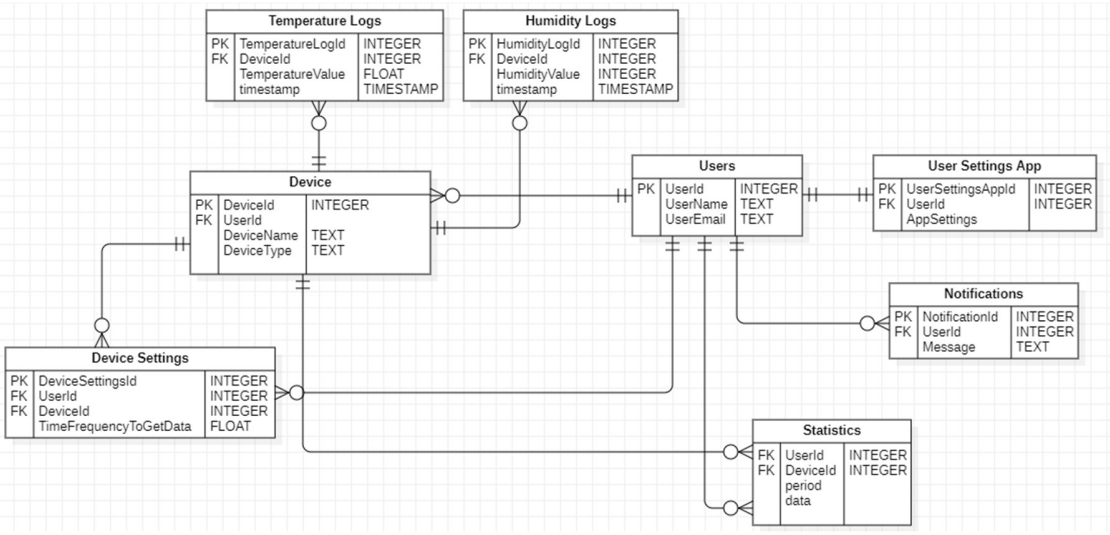


Рисунок 2.1 – Початкова UML-діаграма бази даних для інформаційної системи «розумного» будинку “SweeMe”

Основними функціями інформаційної системи є наступні:

- Реєстрація користувачів: ця можливість дозволяє користувачам створювати свої облікові записи шляхом введення необхідних даних, таких як ім'я, електронна пошта та пароль..

- Автентифікація та авторизація: після реєстрації, користувачам надається можливість автентифікуватися за допомогою спеціального токена. Паролі користувачів також захищені шляхом їх хешування

- Додавання пристроїв: користувачі можуть додавати свої пристрої до облікових записів, використовуючи QR-код або серійний номер.

- Отримання даних від пристроїв: пристрої можуть збирати різні дані, наприклад, температуру, вологість та інші. Ці дані надсилаються користувачам

залежно від типу пристрою. Також користувач може отримати ці дані за певний термін який обере користувач.

2.1.2 Опис роботи системи

Для початку роботи, користувачу потрібно зареєструватися в додатку «розумного» будинку. Для цього потрібно розробити сторінку реєстрації користувача. Але почнемо з розгляду процесу реєстрації який продемонстровано на(рис. 2.2):

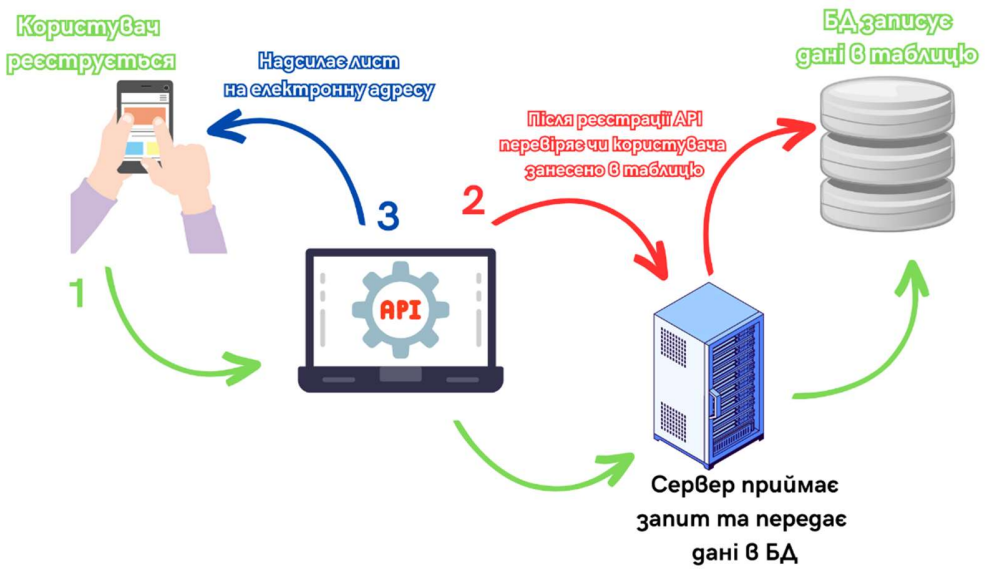


Рисунок 2.2 – Процес реєстрації нового користувача

коли користувач реєструється він заповнює поля з даними та надсилає запит на реєстрацію з своїми даними на сервер він в свою чергу перевіряє ці данні та надсилає результат який отримує користувач та після отриманого результату користувачу або дозволено увійти або заборонено через ввід деяких не правельних даних . Приклад заповнення даних реєстрації(рис. 2.3) – “Ім`я”, “пошту” та “пароль”

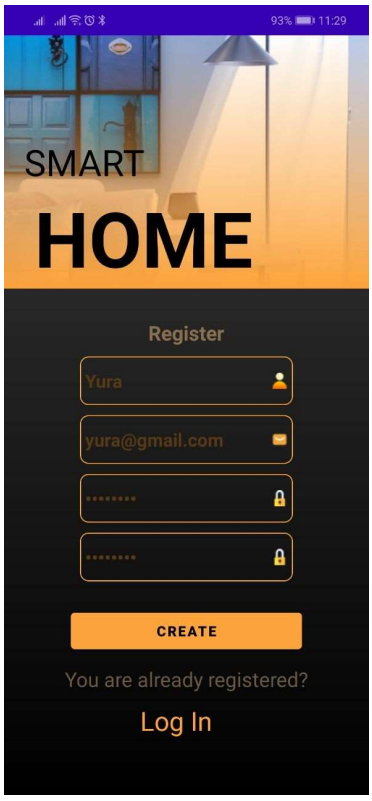


Рисунок 2.3 – Приклад заповнення даних реєстрації

Якщо сервер перевірив правильність цих даних, наприклад, чи відповідає введена електронна адреса є дійсно електронною адресою. Після повернення відповіді, якщо не виникає помилок, користувача перекидає на сторінку підтвердження пошти з якої користувач одразу може перейти на свою пошту та підтвердити електронну адресу. Дані користувача після успішної реєстрації та підтвердження пошти записуються у таблицю “users” (рис. 2.4).

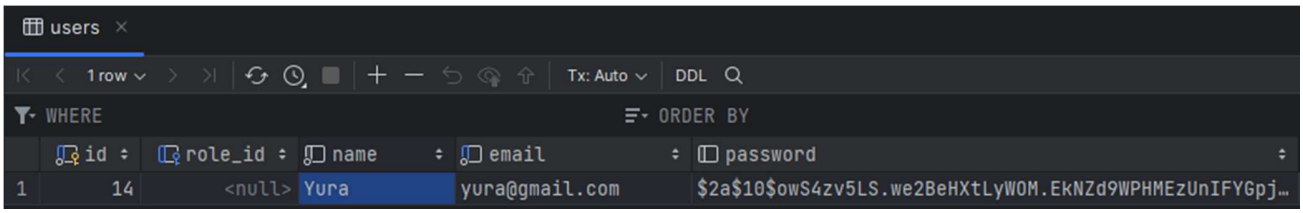


Рисунок 2.4 – Приклад відображення користувача в таблиці

Після успішної реєстрації, користувачу необхідно пройти процедуру авторизації. Для цього використовується сторінка авторизації, де користувач вводить свої дані. При натисканні кнопки "Увійти", дані надсилаються на сервер для перевірки. Якщо всі дані проходять перевірку, сервер повертає JWT (токен авторизації) з інформацією про користувача. Якщо користувач не зміг авторизуватися то йому повертається деякий текст який відображається внизу екрану. Він повідомляє користувача що саме сталося не так та як виправити помилку.

Після успішної авторизації, користувач може перейти на сторінку додавання пристрою до свого облікового запису. Для цього використовується система з серійним номером для кожного типу пристрою. Користувач може відкрити сторінку для додавання серійного номера вручну та після цього зчитати QR-код. Після проведення даних маніпуляцій на головному екрані користувача буде відображатися всі його девайси.

2.2 Аналіз та вибір технологій для інформаційної системи

Було досліджено декілька сучасних засобів розробки та технологій для інформаційної системи "розумного" дому і порівняв їх я обрав такі компоненти і технології: Android Studio, GitHub, JWT, Google OAuth2, BarcodeScanner, Retrofit і GSON.

Android Studio є популярним середовищем розробки для створення частини інформаційної системи "розумного" дому. Він має кілька переваг:

1. Використання декількох мов програмування: Android Studio дозволяє використовувати Kotlin та Java, що вже знайомі багатьом розробникам. Через те що Android Studio може використовувати дві мови тому і в коді програми вони сперечатися не будуть та дозволять легко використовувати плюси одне одної для вирішення різних проблем.

2. Підходить для великих додатків: Android Studio має потужні можливості для побудови великих додатків для мобільних телефонів.

GitHub є веб-платформою, яка надає хостинг для репозиторіїв Git. Використання GitHub має декілька переваг:

1. Управління версіями: GitHub дозволяє зберігати та управляти версіями програмного коду за допомогою системи контролю версій Git. Це дозволяє ефективно керувати змінами в коді та спільно працювати над проектом з іншими розробниками.

2. Колаборація: GitHub сприяє колаборації між розробниками, забезпечуючи зручний веб-інтерфейс для спільної роботи над проектами. Він надає інструменти для огляду та затвердження змін, управління проблемами, відстеження задач та багато іншого.

JWT (JSON Web Token) - це стандарт для безпечної передачі токенів у форматі JSON між двома сторонами. JWT використовується для автентифікації та авторизації в розподілених системах, включаючи веб-додатки та мікросервіси.

JWT дозволяє передавати безпечні токени, які містять інформацію про користувача, між клієнтом і сервером. Це дозволяє здійснювати автентифікацію та авторизацію без збереження стану сервера. JWT є простим у використанні, масштабованим та дозволяє передавати додаткові дані у безпечному форматі.

Додатковою технологією, яку ви обрали, є Google OAuth2. Вона забезпечує безпечну автентифікацію користувачів через облікові записи Google. Це дозволяє використовувати авторизацію Google для доступу до різноманітних функцій та ресурсів у вашій інформаційній системі.

Крім того, обраний BarcodeScanner, Retrofit і GSON як компоненти для розробки "розумного" дому. BarcodeScanner дозволяє сканувати штрих-коди та отримувати інформацію про продукти, Retrofit використовується для взаємодії з веб-сервісами, а GSON - для обробки даних у форматі JSON.

Вибір таких компонентів та технологій дозволяє забезпечити зручне середовище розробки, ефективне управління версіями, безпечну автентифікацію, можливість роботи з штрих-кодами та обробки даних у форматі JSON.

Google OAuth2 (Open Authorization 2.0) - це протокол автентифікації та авторизації, розроблений Google. Він дозволяє користувачам надавати доступ до своїх облікових записів Google іншим додаткам та сервісам без необхідності передавати свої облікові дані. Google OAuth2 забезпечує безпеку та приватність користувачів, оскільки облікові дані Google не передаються додаткам та сервісам, а використовуються лише авторизаційні токени. Крім того, він дозволяє користувачам контролювати доступ до своїх облікових записів Google.

BarcodeScanner (або Barcode Scanner) - це бібліотека, яка дозволяє зчитувати та декодувати штрих-коди з різних джерел, таких як камера смартфона. Вона допомагає зручно та швидко зчитувати інформацію з штрих-кодів. BarcodeScanner підтримує різні типи штрих-кодів, включаючи EAN, UPC, QR-коди, Data Matrix та інші популярні формати.

Retrofit - це бібліотека для роботи з мережевими запитами в мові програмування Kotlin (та інших мовах, таких як Java) для платформи Android. Вона надає простий та зручний спосіб виконання HTTP-запитів до веб-серверів та обробки отриманих відповідей. Retrofit спрощує взаємодію з веб-сервісами шляхом опису API-інтерфейсів та автоматичної серіалізації/десеріалізації даних у форматі JSON.

Gson - це бібліотека для роботи з серіалізацією та десеріалізацією об'єктів у формат JSON в мові програмування Kotlin (та інших мовах, таких як Java). Вона надає зручні методи для перетворення об'єктів Kotlin на JSON-рядки та навпаки, дозволяючи ефективно обробляти дані у форматі JSON.

За допомогою цих компонентів і технологій ви зможете розробляти додатки з автентифікацією через Google OAuth2, зчитувати штрих-коди за

допомогою BarcodeScanner, виконувати мережеві запити до сервера з використанням Retrofit та обробляти дані у форматі JSON з використанням Gson

Висновки до розділу 2

Після аналізу та вибору технологій для інформаційної системи "розумного" дому, було визначено оптимальний набір компонентів, що відповідають вимогам та потребам проєкту.

Також було розглянуто деякий функціонал для клієнтської частини та продемонстровано функціонал реєстрації користувача.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Засоби розробки

Для розробки інформаційної системи для «розумного» будинку “SweeMe” та інтерфейсу користувача було обрано мову програмування Kotlin та середовищі програмування Android Studio.

Платформою для зберігання та керування проектами, з використанням системи контролю версій Git обрано GitHub.

Для тестування підключення та надсилання запитів використовувалася API.

У програмний код було імпортовано такі бібліотеки:

1. Retrofit - це бібліотека для роботи з мережевими запитами в мові програмування Kotlin
2. Google OAuth2 – протокол який допомагає легко та зручно здійснити реєстрацію та авторизацію користувача в системі через google account.
3. JWT - використовується для отримання токенів, які допомагають обмінюватися даними з сервером.
4. BarcodeScanner - це бібліотека, яка дозволяє зчитувати та декодувати штрих-коди з різних джерел, таких як камера смартфона.
5. Gson - це бібліотека для роботи з серіалізацією та десеріалізацією об'єктів у формат JSON в мові програмування Kotlin
6. Volley - є одним із рекомендованих способів для роботи з мережевими запитами в Android додатках.
7. AppCompat - надає зручний спосіб підтримувати звичний зовнішній вигляд та функціональність на різних версіях платформи Android.
8. Core-KTX - є набором розширень Kotlin для бібліотеки AndroidX Core.

3.2 Опис реалізації системи

За допомогою сервісу для розробки інтерфейсів Figma було створено дизайн інтерфейсу користувача(рис. 3.1).

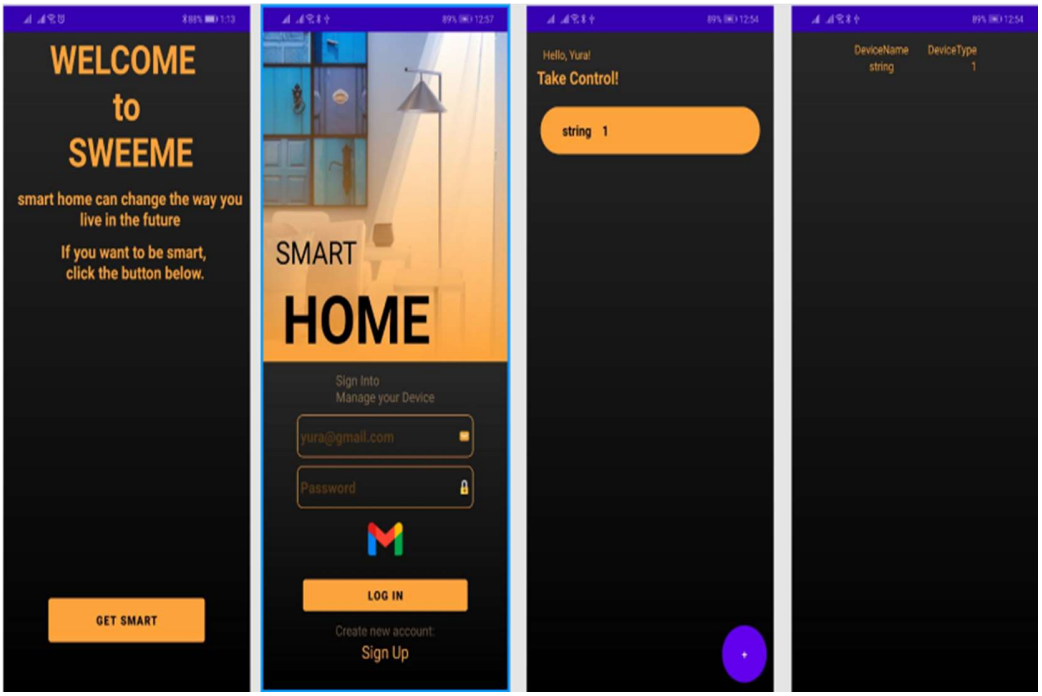


Рисунок 3.1 – Дизайн для інформаційної системи «розумного» будинку “SweeMe”

На рисунку ми бачимо 4 основні сторінки керування з якими взаємодіє користувач сторінки.

Почнемо розгляд сторінок з 1 ми бачимо початкове вікно яке бачить користувач при вході в програму на якій знаходиться з назвою програми та кнопкою при натисканні якої користувач починає знайомитися з програмою.

На наступній сторінці ми бачимо сторінку входу на якій користувач має ввести свої данні та увійти в аккаунт. Якщо у користувача немає аккаунту він може натиснути на кнопку SignUp та перейти на сторінку реєстрації та створити аккаунт.

На 3 сторінці ми бачимо сторінку користувача. На даній сторінці відображається привітання з ім'ям користувача, список всіх девайсів користувача та кнопка яка відповідає за додавання нового девайсу або сканування QR коду.

На останній сторінці в нас зображено сторінку на якій ми бачимо Назву девайсу та тип даного девайсу. На цій сторінці користувач бачить основну інформацію про обраний девайс.

Я би хотів більш детальніше зупинитися на 3 сторінці тобто на сторінці головного екрану користувача на якій користувач буде проводити основу діяльність(рис. 3.2).

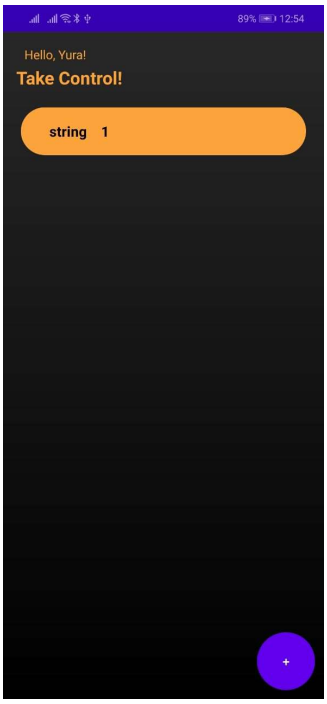


Рисунок 3.2 Зображення сторінки головного екрану

На даній сторінці користувач буде проводити основну роботу в нашій програмі це:

- Перегляд девайсів
- Додавання нових девайсів
- Сканування QR коду

Користувач може переглянути свої девайси одразу як ввійде в аккаунт але при першому вході в користувача девайсів не буде тому ми перейдемо до додавання девайсів користувачу для того щоб це зробити потрібно натиснути на кнопку яка знаходиться зправа знизу екрана телефону. Після цього в нас відкриється меню вибору дії(рис.3.3).

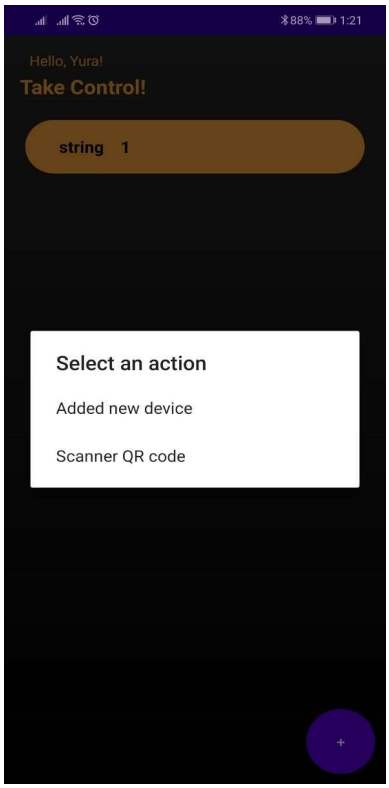


Рисунок 3.3 Меню Вибору дії після натискання кнопки “+” “Додавання нового девайсу”.

В даному меню ми можемо обрати одну з двух дій або створення нового девайсу або сканування QR коду. Для початку оберемо дію додавання нового девайсу. Після натискання даної дії користувачу відкриється сторінка на якій користувач зможе додати девайс(рис 3.4).

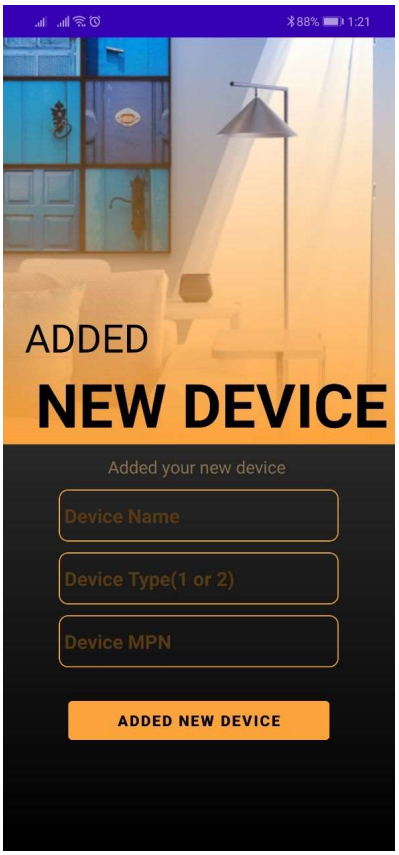


Рисунок 3.4 Сторінка додавання нового девайсу користувача

На даній сторінці ми можемо побачити 3 текстових поля куди користувач має ввести данні девайсу який він хоче додати. Поле “назва девайсу” — це назва девайсу наприклад “Камера”. Далі йде “тип девайсу” поки їх можливо тільки 2. І останнє поле містить MPN(Manufacturer Part Number(номер виробника)) - це унікальний ідентифікатор, який використовується виробниками для ідентифікації конкретного товару або компонента, який вони виробляють. Кожен товар може мати свій власний MPN, який відрізняє його від інших товарів того ж виробника або з тим самим брендом. Після заповнення всіх полів користувач натискає кнопку “Додати новий девайс”. Якщо користувач коректно заповнив поля та запит пройшов успішно користувача перекине на наступне вікно на якому він побачить QR код свого девайсу за допомогою якого він зможе

відобразити свій девайс в списку девайсів на головній сторінці. Вигляд сторінки яку користувач бачить після вдалого додавання девайсу(рис. 3.5)

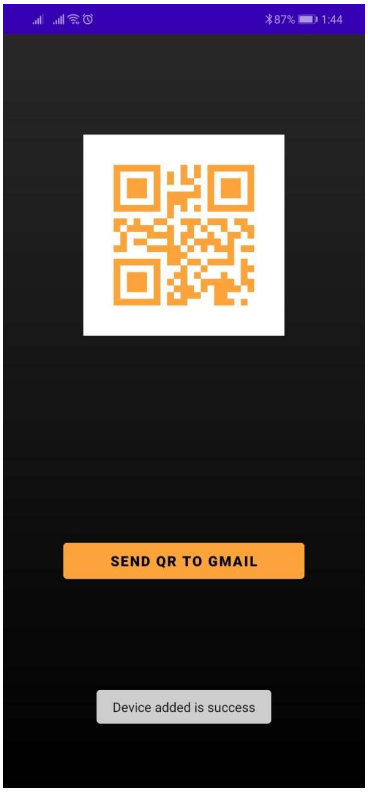


Рисунок 3.5 Сторінка успішно виконаної операції додавання девайсу на аккаунт користувача з QR кодом

На данному етапі користувач може зсканувати QR код та додати девайс до нашого списку. Але це в ситуації коли в нас 2 пристроя на яких здійснено вхід в 1 і той самий аккаунт і ми можемо це зробити але коли в нас є один пристрій то це стає проблемою .Але саме на такий випадок користувач може надіслати QR код собі на пошту або в приватні повідомлення месенджера. Щоб надіслати його на пошту або в месенджер потрібно натиснути на кнопку з назвою “Надіслати QR на пошту” та обрати потрібну нам програму наприклад оберемо пошту(рис 3.6).

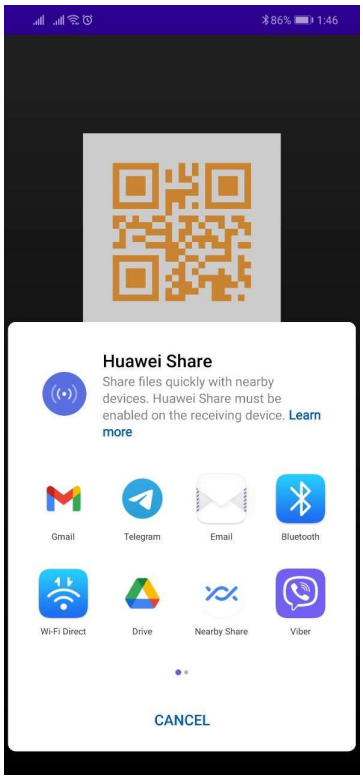


Рисунок 3.6 Меню вибору програми в яку нам потрібно надіслати QR код.

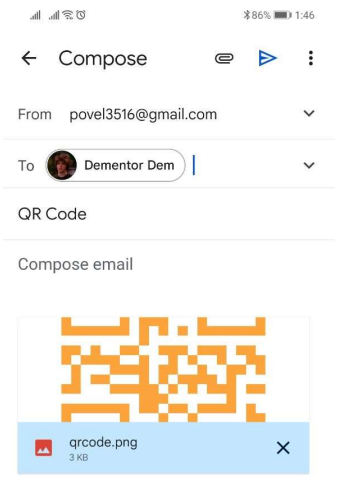


Рисунок 3.7 Вікно надсилання QR коду на пошту.

Після завершення всіх цих кроків користувач додав собі на аккаунт новий пристрій але він ще не буде відображатися на головному екрані тому що для того щоб девайс відобразився на головному екрані потрібно щоб користувач відсканував створений QR код. Щоб відсканувати QR код потрібно вернутися на головний екран знову натиснути кнопку позначену “+”. Та обрати дію “Сканувати QR код” (рис 3.8).

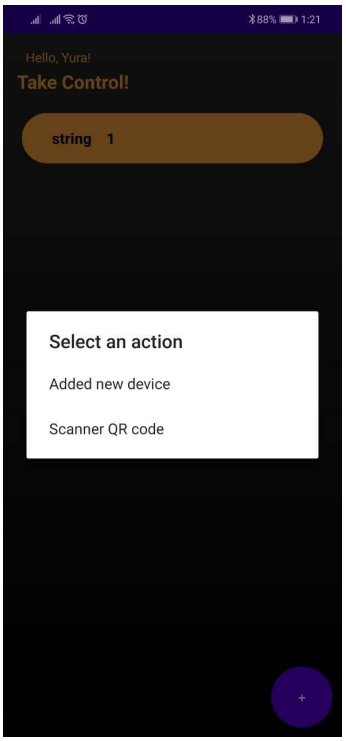


Рисунок 3.8 Меню вибору дії після натискання кнопки “+” “Сканувати QR код”

Далі потрібно обрати дію “Сканер QR коду”. Відкриється вікно сканеру камери з сканером. Якщо користувач відкриває сканер в перше то потрібно буде надати дозвіл на використання камери якщо користувач не надасть дозвіл на використання камери то при наступному вході в користувача знову запитається дозвіл на використання камери. Якщо користувач надав дозвіл на використання камери то він побачить наступне вікно(рис 3.9).

Рисунок 3.9 Вікно сканеру QR коду з успішним додавання девайсу до списку.

Якщо після відскануваня QR коду данні будуть надісланні успішно то вікно сканеру закриється та відкриється головна сторінка на якій ми побачимо що в нас в списку з'явився новий девайс(рис 3.10).

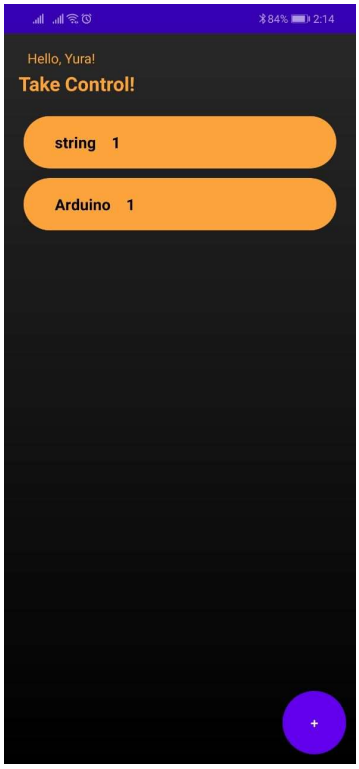
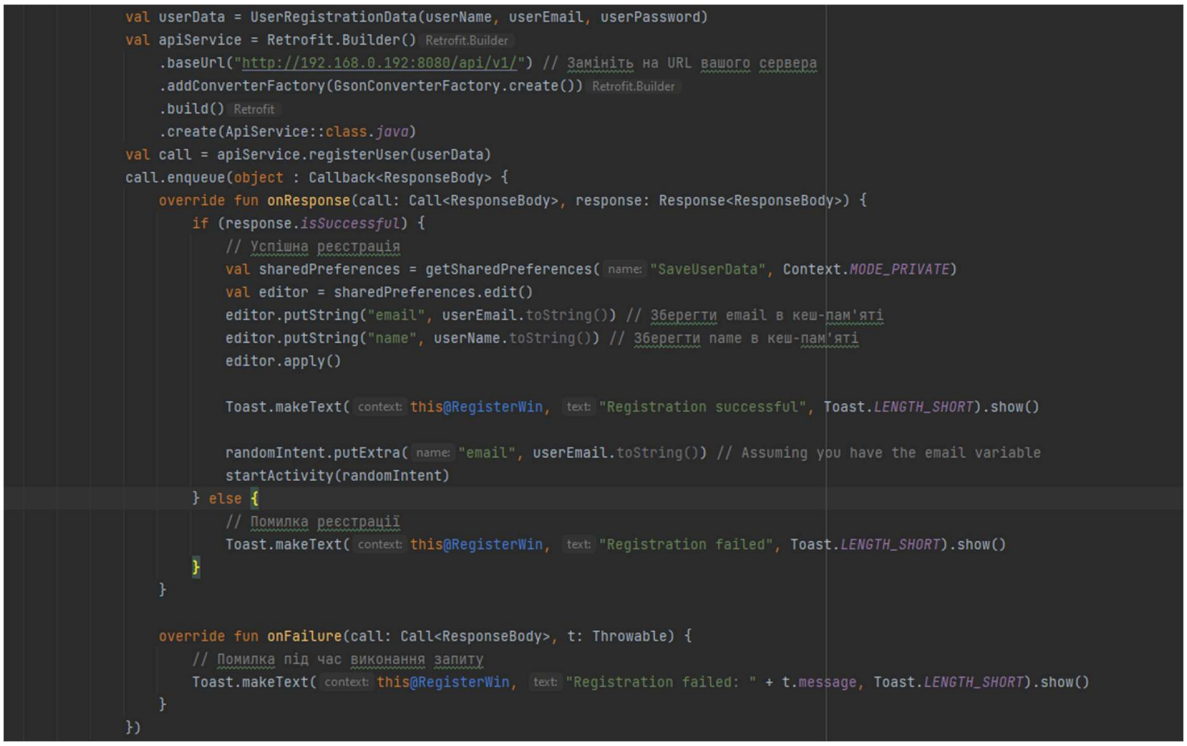


Рисунок 3.10 Зображення головного екрану після всіх проведених маніпуляцій

3.3 Аналіз отриманих результаті

Розглянемо процес реєстрації, авторизації, додавання пристрою до акаунту користувача детальніше.

Користувач вводить дані в поля та надсилає запит. Сервер відповідає йому, що користувача було зареєстровано та лист для верифікації електронної пошти надіслано (рис. 3.11).



```
val userData = UserRegistrationData(userName, userEmail, userPassword)
val apiService = Retrofit.Builder() Retrofit.Builder
    .baseUrl("http://192.168.0.192:8080/api/v1/") // Замініть на URL вашого сервера
    .addConverterFactory(GsonConverterFactory.create()) Retrofit.Builder
    .build() Retrofit
    .create(ApiService::class.java)
val call = apiService.registerUser(userData)
call.enqueue(object : Callback<ResponseBody> {
    override fun onResponse(call: Call<ResponseBody>, response: Response<ResponseBody>) {
        if (response.isSuccessful) {
            // Успішна реєстрація
            val sharedPreferences = getSharedPreferences( name: "SaveUserData", Context.MODE_PRIVATE)
            val editor = sharedPreferences.edit()
            editor.putString("email", userEmail.toString()) // Зберегти email в кеш-пам'яті
            editor.putString("name", userName.toString()) // Зберегти name в кеш-пам'яті
            editor.apply()

            Toast.makeText( context: this@RegisterWin, text: "Registration successful", Toast.LENGTH_SHORT).show()

            randomIntent.putExtra( name: "email", userEmail.toString()) // Assuming you have the email variable
            startActivity(randomIntent)
        } else {
            // Помилка реєстрації
            Toast.makeText( context: this@RegisterWin, text: "Registration failed", Toast.LENGTH_SHORT).show()
        }
    }

    override fun onFailure(call: Call<ResponseBody>, t: Throwable) {
        // Помилка під час виконання запиту
        Toast.makeText( context: this@RegisterWin, text: "Registration failed: " + t.message, Toast.LENGTH_SHORT).show()
    }
})
```

Рисунок 3.11 Надсилання запиту на реєстрацію

Тут ми бачимо створення запиту на надсилання даних користувача на API та обробку відповіді від сервера що якщо сервер відповів що запит виконано успішно то виведеться повідомлення про те що реєстрація успішна і перекине користувача на сторінку з підтвердженням пошти (рис. 3.12).

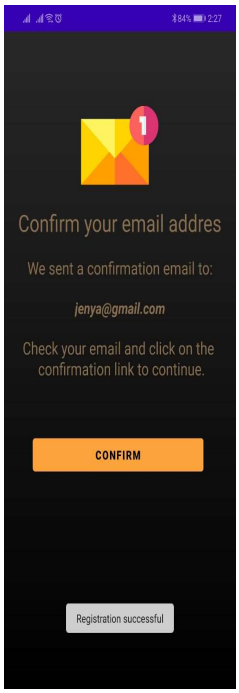


Рисунок 3.12 Сторінка підтвердження пошти

Для того, щоб авторизуватися потрібно підтвердити пошту. Якщо цього не зробити, ми побачимо те саме вікно яке зображено на рисунку(3.12).

Щоб підтвердити пошту потрібно перейти до пошти та натиснути на посилання яке надійшло(рис. 3.13).

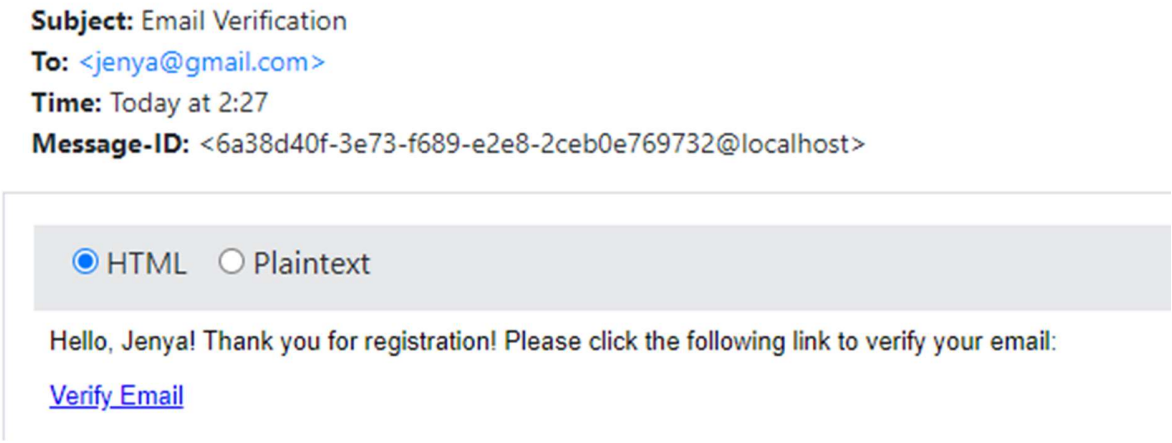


Рисунок 3.13 – Електронний лист із згенерованим посиланням верифікації

Після того, як користувач переходить по посиланню, він бачить повідомлення, що його пошту верифіковано (рис. 3.15), (рис. 3.16).

{"message":"Email verified successfully"}

Рисунок 3.15 – Повідомлення про успішну верифікацію електронної пошти

Коли акаунт було верифіковано, можна здійснити авторизацію. Для цього потрібно заповнити поля та натиснути кнопку логін в цей момент дані які ввів користувач будуть надіслані на сервер та перевірені. Якщо користувач правильно ввів дані та підтвердив пошту то він успішно авторизується в програму та побачить наступний головний екран(рис. 3.16).

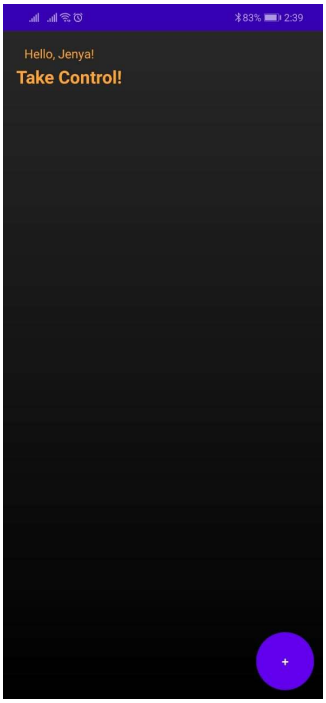
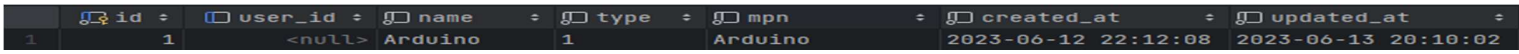


Рисунок 3.16 – Приклад успішної авторизації користувача

Тепер авторизований користувач може додати пристрій. Оскільки він має токен, серверна частина яка відповідає за додавання пристроїв зможе дізнатися ідентифікатор користувача і присвоїти йому пристрій. В таблиці було попередньо створено пристрій (рис. 3.17).



	id	user_id	name	type	mpn	created_at	updated_at
1	1	<null>	Arduino	1	Arduino	2023-06-12 22:12:08	2023-06-13 20:10:02

Рисунок 3.17 – Пристрій в таблиці «devices»

Щоб користувачу додати пристрій, потрібно надіслати запит на сервер далі сервер перевірить, чи існує такий пристрій. Якщо так, то в цей пристрій буде додано до аккаунту користувача та користувач зможе користуватися цим пристроєм.

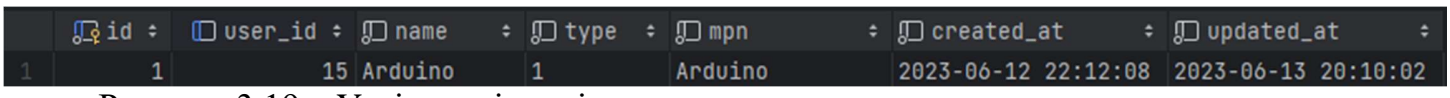
Приклад оформлення запиту на додавання девайсу: в тілі запиту потрібно вказати серійний номер це рядок “post(requestBody)” — в цьому рядку передається MPN пристроя

а в рядку “header()” передається заголовок запиту з токеном користувача.(рис. 3.18)

```
val url = "http://192.168.0.192:8080/api/v1/devices/add-device-by-mpn"
val request = Request.Builder()
    .url(url)
    .header( name: "Authorization",  value: "Bearer $token")
    .post(requestBody)
    .build()
```

Рисунок 3.18 – Тіло запиту для додав девайсу

Якщо всі дані введено правильно сервер поверне пристрій у JSON-форматі де вже пристрій буде додано до користувача приклад з таблиці див на (рис. 3.19).



	id	user_id	name	type	mpn	created_at	updated_at
1	1	15	Arduino	1	Arduino	2023-06-12 22:12:08	2023-06-13 20:10:02

Рисунок 3.19 – Успішна відповідь на запит користувача про додавання пристрою

Якщо той самий користувач знову відправить запит на додавання того ж самого пристрою, то отримає помилку, що такий пристрій користувач вже додав до свого аккаунту та запропонує додати інший(рис. 3.20).

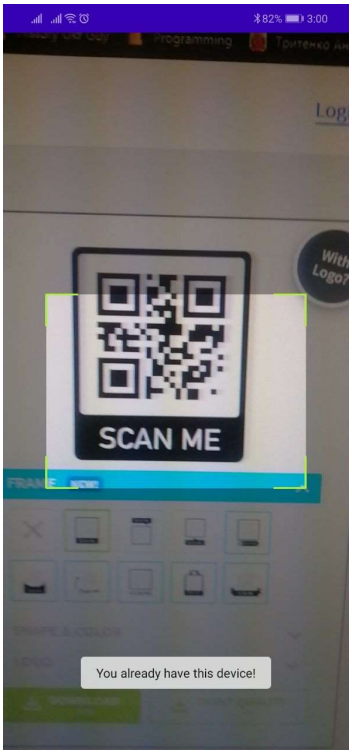


Рисунок 3.20 – Приклад помилки, коли користувач намагається знову додати пристрій

Висновки до розділу 3

У цьому розділі було розглянуто й вибрано різні засоби розробки, а також надано огляд їх переваг і обґрунтувань для використання. Була надана детальна інформація про реалізацію інформаційної системи і її взаємодію з інтерфейсом користувача.

В кінцевій частині розділу була продемонстрована робота інформаційної системи, де показано основні можливості та спосіб взаємодії з користувачем.

Інформаційна система та її інтерфейс працюють без помилок і здатні взаємодіяти з API. Система володіє гнучкістю та потенціалом для модифікацій та покращень.

ВИСНОВКИ

Під час проектування та розробки інформаційної системи та інтерфейсу користувача для "розумного" будинку "SweeMe" було успішно досягнуто мети проекту та виконано всі поставлені задачі.

Завдання дослідження включали аналіз предметної області, проектування інформаційної системи, вибір необхідних інструментів та методів для її реалізації. Аналіз предметної області включав вивчення сучасних технологій та рішень у галузі "розумних" будинків, а також встановлення основних потреб і вимог користувачів. На основі цього аналізу було розроблено інформаційну систему "SweeMe" та реалізовано інтерфейс користувача для взаємодії з "розумним" будинком.

Процес розробки системи дозволив набути цінного досвіду та навичок, які полегшать подальшу роботу з подібними інформаційними системами і сприятимуть подальшому розвитку вже існуючої системи.

Розробка даної системи виявилася актуальною, оскільки, порівнявши ринок та існуючі системи, було встановлено, що розробка власної системи дозволить зекономити кошти та матиме потенціал для подальшого росту і вдосконалення. Ця система може конкурувати з іншими, оскільки передбачає розробку та впровадження пристроїв, які можуть бути настроєні під потреби конкретного клієнта. Більше того, ця система може бути застосована не лише в приватних будинках і квартирах, але й на підприємствах та в різних бізнес-сферах.

Використання цієї інформаційної системи дозволить зменшити споживання електроенергії, забезпечить зручне керування пристроями, розробку власних систем та просту інтеграцію з ними.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Kotlin Programming: The Big Nerd Ranch Guide" авторів Josh Skeen та David Greenhalgh
2. "Android Programming: The Big Nerd Ranch Guide" авторів Bill Phillips, Chris Stewart та Kristin Marsicano
3. "Head First Android Development" авторів Dawn Griffiths та David Griffiths
4. "Mastering Android Studio 3.0" автора Kyle Mew
5. "Android Studio Development Essentials" автора Neil Smyth
6. "Material Design for Android Developers" автора Ian G. Clifton
7. "The UX Book: Process and Guidelines for Ensuring a Quality User Experience" авторів Rex Hartson та Pardha S. Pyla
8. "Don't Make Me Think, Revisited: A Common Sense Approach to Web and Mobile Usability" автора Steve Krug
9. "Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days" авторів Jake Knapp, John Zeratsky та Braden Kowitz
10. Офіційна документація Kotlin (https://kotlinlang.org/docs/home.html)
11. Android Development with Kotlin (https://developer.android.com/courses/android-development-with-kotlin/course?utm_source=dac&utm_medium=website&utm_campaign=edu)
12. Офіційна документація Retrofit (https://square.github.io/retrofit/)

Додаток А. Код програми

MainActivity.kt

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
}

fun openRegister(view: View)
{
    val randomIntent = Intent(this, RegisterWin::class.java)
    val randomIntentLogIn = Intent(this, Log_in::class.java)

    val sharedPreferences = getSharedPreferences("SaveUserData", Context.MODE_PRIVATE)
    val email = sharedPreferences.getString("email", "")

    val fieldEmail = findViewById<TextView>(R.id.textEmailLogIn)

    if(email.toString().isNotEmpty()) startActivity(randomIntentLogIn)
    else startActivity(randomIntent)
}
```

Log_in.kt

```
class Log_in : AppCompatActivity(),
    GoogleApiClient.ConnectionCallbacks,
    GoogleApiClient.OnConnectionFailedListener {

    private lateinit var googleApiClient: GoogleApiClient

    private val client = OkHttpClient()
    private var token: String? = null

    companion object{
        const val REQUEST_CODE_GOOGLE_SIGN_IN = 9001
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_log_in)

        val gso = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestEmail()
            .build()

        googleApiClient = GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
```

```
            .addOnConnectionFailedListener(this)
            .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
            .build()

        val sharedPreferences = getSharedPreferences("SaveUserData", Context.MODE_PRIVATE)
        val email = sharedPreferences.getString("email", "")

        val fieldEmail = findViewById<TextView>(R.id.textEmailLogIn)
        val imageViewClick = findViewById<ImageView>(R.id.signInWithGmail)

        imageViewClick.setOnClickListener {  signInWithGoogle() }

        if(email.toString().isNotEmpty())
        {
            fieldEmail.text = email.toString()
        }
    }

    private fun signInWithGoogle() {
        val signInOptions = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestEmail()
            .build()

        val googleSignInClient = GoogleSignIn.getClient(this, signInOptions)
        googleSignInClient.signOut().addOnCompleteListener {
            startActivityForResult(googleSignInClient.signInIntent,
REQUEST_CODE_GOOGLE_SIGN_IN)
        }
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == REQUEST_CODE_GOOGLE_SIGN_IN) {
            val result = data?.let { Auth.GoogleSignInApi.getSignInResultFromIntent(it) }
            if (result != null) {
                handleGoogleSignInResult(result)
            }
        }
    }

    private fun handleGoogleSignInResult(result: GoogleSignInResult) {
        if (result.isSuccess) {
            val account = result.signInAccount

            val randomIntent = Intent(this, HomePage::class.java)
            startActivity(randomIntent)
```

```
        val googleId = account?.id ?: ""
        Log.i("Log_in", "Google id = $googleId")
        val googleFirstName = account?.givenName ?: ""
        Log.i("Log_in", "Google FirstName = $googleFirstName")
        val googleLastName = account?.familyName ?: ""
        Log.i("Log_in", googleLastName)
        val googleEmail = account?.email ?: ""
        Log.i("Log_in", googleEmail)
        val googleProfilePicURL = account?.photoUrl.toString()
        Log.i("Log_in", googleProfilePicURL)
        val googleIdToken = account?.idToken ?: ""
        Log.i("Log_in", googleIdToken)

        // Виконання запиту на сервер для створення аккаунта
        val url = "http://192.168.0.192:8080/api/v1/google?email=$googleEmail"
        val request = Request.Builder()
            .url(url)
            .get()
            .build()

        Log.i("Log_in", "request$request")
        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                // Обробка помилок під час виконання запиту
                runOnUiThread {
                    Toast.makeText(this@Log_in, "Failed to create account",
Toast.LENGTH_SHORT).show()
                }
            }


            override fun onResponse(call: Call, response: Response) {
                // Обробка відповіді від сервера
                val responseBody = response.body?.string()
                if (response.isSuccessful) {
                    // Аккаунт успішно створений
                    runOnUiThread {
                        Log.i("Log_in", "$responseBody")
                        val randomIntent = Intent(this@Log_in, HomePage::class.java)
                        startActivity(randomIntent)
                    }
                } else {
                    // Помилка створення аккаунта
                    runOnUiThread {
                        Toast.makeText(this@Log_in, "Failed to create account",
Toast.LENGTH_SHORT).show()
```

```
                }
            }
        }
    })
} else {
    // Sign-in failed, handle accordingly
}
}

//Відкритя форми реєстрації
fun openRegister(view: View)
{
    val randomIntent = Intent(this, RegisterWin::class.java)

    startActivity(randomIntent)
}

fun readData(view: View)
{
    val randomIntent = Intent(this, HomePage::class.java)
    val randomIntentVerif = Intent(this, Confirmation_Form::class.java)

    val fieldEmail = findViewById<TextView>(R.id.textEmailLogIn)
    val fieldPassword = findViewById<TextView>(R.id.textPasswordLogIn)

    val userEmail = fieldEmail.text
    val userPassword = fieldPassword.text
    if(userEmail.toString().isNotEmpty() && userPassword.toString().isNotEmpty())
    {
        Log.i("LogIn", "StartReadData")
        val requestBody = FormBody.Builder()
            .add("email", userEmail.toString())
            .add("password", userPassword.toString())
            .build()
        Log.i("LogIn", "StartFeelQueryInfo")
        Log.i("LogIn", "${userEmail.toString()} ${userPassword.toString()}")
        val request = Request.Builder()
            .url("http://192.168.0.192:8080/api/v1/login")
            .post(requestBody)
            .build()

        Log.i("LogIn", "StartQuery")
        Log.i("LogIn", "$request")
        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                Toast.makeText(this@Log_in, "Registration successful",
Toast.LENGTH_SHORT).show()
```

```
            }

            override fun onResponse(call: Call, response: Response) {
                Log.i("LogIn", "StartonResponse")
                val body = response.body?.string()
                Log.i("LogIn", "Body- $body")
                val json = JSONObject(body)

                if (json.has("token")) {
                    token = json.getString("token")
                    Log.i("LogIn", "$token")
                    // Збереження токену на пристрої
                    Log.i("LogIn", "SaveToken")

                    // Відкриття іншої активності або виконання іншого коду, що потребує
авторизації
                        randomIntent.putExtra("token", token)
                        randomIntent.putExtra("email", userEmail.toString()) // Assuming you have the
email variable
                    startActivity(randomIntent)
                } else {
                    // Обробка помилок
                    val errorMessage = json.getString("message")
                    Log.i("LogIn", "Error message: $errorMessage")
                    if(errorMessage == "Please verify your email before logging in")
                    {
                            randomIntentVerif.putExtra("email", userEmail.toString()) // Assuming you
have the email variable
                        startActivity(randomIntentVerif)
                    }
                }
            }
        })
    } else Toast.makeText(this@Log_in, "Data was incorrect", Toast.LENGTH_SHORT).show()
    }

  override fun onConnected(p0: Bundle?) {
    TODO("Not yet implemented")
  }

  override fun onConnectionSuspended(p0: Int) {
    TODO("Not yet implemented")
  }

  override fun onConnectionFailed(p0: ConnectionResult) {
    TODO("Not yet implemented")
```

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

        setContentView(R.layout.activity_register_win)
    }


    //Відкритя форми входу в акаунт
    fun openLogIn(view: View) {
        val randomIntent = Intent(this, Log_in::class.java)

        startActivity(randomIntent)
    }

    // Считування даних з полей форми
    fun readData(view: View) {
        val randomIntent = Intent(this, Confirmation_Form::class.java)

        val fieldName = findViewById<EditText>(R.id.textName)
        val fieldPassword = findViewById<EditText>(R.id.textPassword)
        val fieldCheckPassword = findViewById<EditText>(R.id.textPasswordConfirm)
        val fieldEmail = findViewById<EditText>(R.id.textEmail)

        val userName = fieldName.text.toString()
        val userPassword = fieldPassword.text.toString()
        val userCheckPassword = fieldCheckPassword.text.toString()
        val userEmail = fieldEmail.text.toString()

        if (userName.isNotEmpty() && userPassword.isNotEmpty() &&
userCheckPassword.isNotEmpty() && userEmail.isNotEmpty()) {
            if (userPassword != userCheckPassword) {
                Toast.makeText(this, "Your password does not match", Toast.LENGTH_SHORT).show()
            } else if (userEmail.isEmpty()
|| !android.util.Patterns.EMAIL_ADDRESS.matcher(userEmail).matches()) {
                Toast.makeText(this, "Your email is wrong!", Toast.LENGTH_SHORT).show()
            } else {
                val userData = UserRegistrationData(userName, userEmail, userPassword)
                val apiService = Retrofit.Builder()
                    .baseUrl("http://192.168.0.192:8080/api/v1/") // Замініть на URL вашого сервера
                    .addConverterFactory(GsonConverterFactory.create())
                    .build()
                    .create(ApiService::class.java)
                val call = apiService.registerUser(userData)
                call.enqueue(object : Callback<ResponseBody> {
```

```
                override fun onResponse(call: Call<ResponseBody>, response:
Response<ResponseBody>) {
                    if (response.isSuccessful) {
                        // Успішна реєстрація
                        val sharedPreferences = getSharedPreferences("SaveUserData",
Context.MODE_PRIVATE)
                        val editor = sharedPreferences.edit()
                        editor.putString("email", userEmail.toString()) // Зберегти email в кеш-пам'яті
                        editor.putString("name", userName.toString()) // Зберегти name в кеш-пам'яті
                        editor.apply()

                        Toast.makeText(this@RegisterWin, "Registration successful",
Toast.LENGTH_SHORT).show()

                        randomIntent.putExtra("email", userEmail.toString()) // Assuming you have the
email variable
                        startActivity(randomIntent)
                    } else {
                        // Помилка реєстрації
                        Toast.makeText(this@RegisterWin, "Registration failed",
Toast.LENGTH_SHORT).show()
                    }
                }

                override fun onFailure(call: Call<ResponseBody>, t: Throwable) {
                    // Помилка під час виконання запиту
                    Toast.makeText(this@RegisterWin, "Registration failed: " + t.message,
Toast.LENGTH_SHORT).show()
                }
            })
        }
    } else {
        Toast.makeText(this, "Fill in all the fields", Toast.LENGTH_SHORT).show()
    }
  }
}

interface ApiService {
  @POST("register")
  fun registerUser(@Body userData: UserRegistrationData): Call<ResponseBody>
}
HomePage.kt
 private var user_id: Int? = null
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_home_page)
```

```
        val recyclerView = findViewById<RecyclerView>(R.id.recyclerViewDevices)
        val layoutManager = LinearLayoutManager(this)
        recyclerView.layoutManager = layoutManager
        recyclerView.setHasFixedSize(true)

        val token = intent.getStringExtra("token")

        val buttonAddDevice = findViewById<Button>(R.id.button)
        buttonAddDevice.setOnClickListener { showPopupDialog(token.toString()) }

        val myDatasource = Datasource()

        myDatasource.loadUserDevice(token.toString(), object: LoadUserDeviceCallback {
            override fun onUserDeviceLoaded(deviceList: List<Device>) {
                // Оновити адаптер з отриманим списком пристроїв
                Log.i("HomePage", " device List $deviceList")

                var adapter = DeviceAdapter(deviceList)
                recyclerView.adapter = adapter
                adapter.setOnItemClickListener(object :  DeviceAdapter.onItemClickListener {

                    override fun onItemClick(position: Int) {
                        //Toast.makeText(this@HomePage, "Your Clicked on item no. $position",
Toast.LENGTH_SHORT).show()

                        val intent = Intent(this@HomePage, DeviceInformation::class.java)
                        intent.putExtra("deviceName", deviceList[position].name)
                        intent.putExtra("deviceType", deviceList[position].type)
                        startActivity(intent)
                    }
                })
            }

            override fun onUserDeviceLoadError(error: String) {
                // Обробити помилку завантаження даних
                Log.i("HomePage", "Error loading user devices: $error")
            }
        })

        sendRequestToServer(token.toString())
    }

    private fun showPopupDialog(token: String) {
        val actions = arrayOf("Added new device", "Scanner QR code")
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Select an action")
            .setItems(actions) { _, which ->
```

```
            when (which) {
                0 -> openAddNewDevice()
                1 -> openQrScanner(token)
            }
        }
        .show()
}

private fun openAddNewDevice() {
    val intent = Intent(this, NewDevice::class.java)
    startActivity(intent)
}

private fun openQrScanner(token:String){
    val randomIntent = Intent(this, QRScannerActivity::class.java)
    randomIntent.putExtra("token", token)
    startActivity(randomIntent)
}

private fun sendRequestToServer(jwt: String) {
    val client = OkHttpClient()
    val url = "http://192.168.0.192:8080/api/v1/decode-jwt" // Replace with your server URL
    val userName = findViewById<TextView>(R.id.textHelloToUser)

    Log.i("HomePage", "Jwt:$jwt")
    Log.i("HomePage", "Data:$url")
    val request = Request.Builder()
        .url(url)
        .header("Authorization", "Bearer $jwt")
        .build()
    Log.i("HomePage", "Create request")
    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            // Handle network failure or server error
            Log.i("HomePage", "Suck")
            e.printStackTrace()
        }

        override fun onResponse(call: Call, response: Response) {
            if (response.isSuccessful) {
                val responseData = response.body?.string()
                val jsonObject = JSONObject(responseData)
                Log.i("HomePage", "Json ${jsonObject}")
                // Process the received data here
                val userObject = jsonObject.getJSONObject("user")
                if (userObject.has("id")) {
                    user_id = userObject.getInt("id")
```

```
                    userName.text = "Hello, ${userObject.getString("name")}!"
                    Log.i("HomePage", "User Id $user_id")
                } else {
                    Log.i("HomePage", "Id is not create")
                }
            } else {
                // Handle unsuccessful response (e.g., unauthorized access)
                // You can check response.code() for the specific HTTP status code
            }
            response.close()
        }
    })
  }
}
```

CobfirmationForm.kt

```
class Confirmation_Form : AppCompatActivity() {
  @SuppressLint("MissingInflatedId")
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_confirmation_form)

    val email = intent.getStringExtra("email")
    val userEmail = findViewById<TextView>(R.id.userEmailTextView)

    userEmail.text = email.toString()
    val openGmailButton: Button = findViewById(R.id.openGmailButton)
    openGmailButton.setOnClickListener {
      val intent = packageManager.getLaunchIntentForPackage("com.google.android.gm")
      startActivity(intent)
    }
  }
}
```

DeviceInformation.kt

```
class DeviceInformation : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_device_information)

    val deviceName : TextView = findViewById(R.id.textViewDeviceName)
    val deviceType : TextView = findViewById(R.id.textViewDeviceType)


    deviceName.text = intent.getStringExtra("deviceName")
    deviceType.text = intent.getStringExtra("deviceType")
  }
}
```

MyViewHolder.kt

```
class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    val deviceName: TextView = itemView.findViewById(R.id.recyclerViewDevices)
    val deviceStatus: TextView = itemView.findViewById(R.id.recyclerViewDevices)
    val deviceImage: ImageView = itemView.findViewById(R.id.recyclerViewDevices)

}
```

qrCode.kt

```
class qrCode : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_qr_code)
        val qrCodeImageView: ImageView = findViewById(R.id.qrCodeImageView)
        val sendQrToGmail: Button = findViewById(R.id.sendQrToGmail)

        val text = intent.getStringExtra("deviceMPN")

        val multiFormatWriter = MultiFormatWriter()
        try {
            val bitMatrix: BitMatrix = multiFormatWriter.encode(text, BarcodeFormat.QR_CODE, 600,
600)

            val width = bitMatrix.width
            val height = bitMatrix.height
            val pixels = IntArray(width * height)

            // Змінити колір фону QR-коду на білий
            val backgroundColor = Color.WHITE

            // Змінити колір пікселів QR-коду на червоний
            val foregroundColor = Color.parseColor("#FDA43C")

            for (y in 0 until height) {
                val offset = y * width
                for (x in 0 until width) {
                    pixels[offset + x] = if (bitMatrix.get(x, y)) foregroundColor else backgroundColor
                }
            }

            val bitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888)
            bitmap.setPixels(pixels, 0, width, 0, 0, width, height)

            qrCodeImageView.setImageBitmap(bitmap)

            sendQrToGmail.setOnClickListener {
                val qrCodeFile = saveBitmapToFile(bitmap)
```

```
            if (qrCodeFile != null) {
                val intent = Intent(Intent.ACTION_SEND)
                intent.type = "application/image"
                intent.putExtra(Intent.EXTRA_SUBJECT, "QR Code")
                intent.putExtra(Intent.EXTRA_STREAM, FileProvider.getUriForFile(this,
"${packageName}.fileprovider", qrCodeFile))
                startActivity(Intent.createChooser(intent, "Надіслати за допомогою"))
            }
        }

    } catch (e: Exception) {
        e.printStackTrace()
    }
  }

  private fun saveBitmapToFile(bitmap: Bitmap): File? {
    val cachePath = File(cacheDir, "images")
    cachePath.mkdirs()
    val qrCodeFile = File(cachePath, "qrcode.png")

    try {
        val outputStream = FileOutputStream(qrCodeFile)
        bitmap.compress(Bitmap.CompressFormat.PNG, 100, outputStream)
        outputStream.flush()
        outputStream.close()
        return qrCodeFile
    } catch (e: IOException) {
        e.printStackTrace()
    }

    return null
  }
}
```

NewDevice.kt

```
class NewDevice : AppCompatActivity() {

  private val client = OkHttpClient()

  @SuppressLint("MissingInflatedId")
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_add_new_device)

    val buttonAddedNewDevice : Button = findViewById(R.id.buttonAddedNewDevice)
    val deviceEditTextName : EditText = findViewById(R.id.editTextDeviceName)
    val deviceEditTextType : EditText = findViewById(R.id.editTextDeviceType)
```

```
        val deviceEditTextMPN : EditText = findViewById(R.id.editTextDeviceMPN)

        buttonAddedNewDevice.setOnClickListener
{ sendDeviceToServer(deviceEditTextName.text.toString(),deviceEditTextType.text.toString(),
deviceEditTextMPN.text.toString() ) }
    }

    fun sendDeviceToServer(deviceName : String, deviceType : String, deviceMPN : String){

        if(deviceName != null || deviceType != null || deviceMPN != null)
        {
            val url = "http://192.168.0.192:8080/api/v1/devices/add"

            val requestBody = FormBody.Builder()
                .add("name" , deviceName)
                .add("type", deviceType)
                .add("mpn", deviceMPN)
                .build()

            val request = Request.Builder()
                .url(url)
                .post(requestBody)
                .build()

            client.newCall(request).enqueue(object : Callback{
                override fun onFailure(call: Call, e: IOException) {
                    runOnUiThread {
                        Toast.makeText(this@NewDevice, "Failed to added device",
Toast.LENGTH_SHORT).show()
                    }
                }

                override fun onResponse(call: Call, response: Response) {
                    runOnUiThread {
                        Toast.makeText(this@NewDevice, "Device added is success",
Toast.LENGTH_SHORT).show()
                        val intent = Intent(this@NewDevice, qrCode::class.java)
                        intent.putExtra("deviceMPN", deviceMPN)
                        finish()

                        startActivity(intent)
                    }
                }

            })
        }else {
            runOnUiThread {
```

```
            Toast.makeText(this@NewDevice, "Pls type data in all fields!s",
Toast.LENGTH_SHORT).show()
        }
    }
  }

}
```

QRScannerActivity.kt

```
class QRScannerActivity : AppCompatActivity(), ZXingScannerView.ResultHandler {

    private lateinit var scannerView: ZXingScannerView

    private val CAMERA_PERMISSION_REQUEST_CODE = 101
    private val client = OkHttpClient()

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

        scannerView = ZXingScannerView(this)
        setContentView(scannerView)

        if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
            != PackageManager.PERMISSION_GRANTED
        ) {
            // Якщо дозвіл на використання камери ще не надано, запросити його
            ActivityCompat.requestPermissions(
                this,
                arrayOf(Manifest.permission.CAMERA),
                CAMERA_PERMISSION_REQUEST_CODE
            )
        } else {
            // Якщо дозвіл вже надано, запустити сканер
            startScanner()
        }
    }

    private fun startScanner() {
        scannerView.setResultHandler(this)
        scannerView.startCamera()
    }

    override fun handleResult(result: Result?) {
        // Отримано результат сканування
        val scannedResult: String? = result?.text
        val token = intent.getStringExtra("token")
        // Обробити отриманий результат
        // Наприклад, вивести в Toast
```

```
//Toast.makeText(this, scannedResult, Toast.LENGTH_SHORT).show()

val requestBody = FormBody.Builder()
    .add("mpn", scannedResult.toString())
    .build()

val url = "http://192.168.0.192:8080/api/v1/devices/add-device-by-mpn"
val request = Request.Builder()
    .url(url)
    .header("Authorization", "Bearer $token")
    .post(requestBody)
    .build()


Log.i("qrScanner", "request $request")

Log.i("qrScanner", "token $token")

client.newCall(request).enqueue(object : Callback{
    override fun onFailure(call: Call, e: IOException) {
        Log.i("qrScanner", "FuckIs")
    }

    override fun onResponse(call: Call, response: Response) {
        if(response.isSuccessful)
        {
            Log.i("qrScanner", "Start Secsess")
            runOnUiThread {
                Toast.makeText(
                    this@QRScannerActivity,
                    "данні успушно надіслані",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }else{
            runOnUiThread {
                Toast.makeText(
                    this@QRScannerActivity,
                    "You already have this device!",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
    }

})
// Перезапустити сканер для подальшого сканування
```

```
        scannerView.resumeCameraPreview(this)
    }

    override fun onPause() {
        super.onPause()
        scannerView.stopCamera() // Зупинити камеру при паузі Activity
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)

        if (requestCode == CAMERA_PERMISSION_REQUEST_CODE) {
            if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                // Дозвіл надано, запустити сканер
                startScanner()
            } else {
                // Дозвіл не надано, показати повідомлення або вжити відповідні дії
            }
        }
    }
}
Device.kt
data class Device(val name: String, val type: String)
SignInBody.kt
data class SignInBody(val name: String, val password: String)
UserRegistrationData.kt
data class UserRegistrationData(
    val name: String,
    val email: String,
    val password: String
)
DataSource.kt
interface LoadUserDeviceCallback {
    fun onUserDeviceLoaded(deviceList: List<Device>)
    fun onUserDeviceLoadError(error: String)
}

class Datasource {

    fun loadDevice(): List<Device>{
        return listOf<Device>(
            Device("string", "1"),
```

```
            Device("camEra","2")
        )
    }
    fun loadUserDevice(token: String, callback: LoadUserDeviceCallback) {
        val client = OkHttpClient()
        val url = "http://192.168.0.192:8080/api/v1/devices/get-devices-by-user-id"

        val request = Request.Builder()
            .url(url)
            .header("Authorization","Bearer $token")
            .build()

        client.newCall(request).enqueue(object : Callback {
            override fun onFailure(call: Call, e: IOException) {
                e.printStackTrace()
                callback.onUserDeviceLoadError("Failed to load user devices: ${e.message}")
            }

            override fun onResponse(call: Call, response: Response) {
                if(response.isSuccessful) {
                    val responseData = response.body?.string()
                    val jsonObject = JSONObject(responseData)
                    val deviceArray = jsonObject.getJSONArray("device")
                    val deviceList = mutableListOf<Device>()

                    for(i in 0 until deviceArray.length()) {
                        val deviceObject = deviceArray.getJSONObject(i)
                        val name = deviceObject.getString("name")
                        val type = deviceObject.getString("type")
                        val device = Device(name, type)
                        deviceList.add(device)
                    }

                    callback.onUserDeviceLoaded(deviceList)
                } else {
                    callback.onUserDeviceLoadError("Failed to load user devices: ${response.code}")
                }

                response.close()
            }
        })
    }
}
```

DeviceAdapter.kt

```
class DeviceAdapter(private val dataset: List<Device>):
RecyclerView.Adapter<DeviceAdapter.DeviceViewHolder>() {
```

```
    private lateinit var mListener: onItemClickListener

    interface onItemClickListener{

        fun onItemClick(position : Int)

    }

    fun setOnItemClickListener(listener: onItemClickListener){

        mListener = listener

    }

        inner class DeviceViewHolder(private val view: View, listener: onItemClickListener) :
RecyclerView.ViewHolder(view){

            val nameTextView: TextView = view.findViewById(R.id.device_name_textView)
            val typeTextView: TextView = view.findViewById(R.id.device_type_textView)

            init {

                view.setOnClickListener {

                    listener.onItemClick(adapterPosition)

                }

            }
        }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): DeviceViewHolder {
      val adapterLayout = LayoutInflater.from(parent.context)
          .inflate(R.layout.device_item, parent, false)

        Log.i("DeviceAdapter", "Start")
        return DeviceViewHolder(adapterLayout, mListener)
    }

    override fun getItemCount() = dataset.size

    override fun onBindViewHolder(holder: DeviceViewHolder, position: Int) {
        val item = dataset[position]

        holder.nameTextView.text = item.name
        Log.i("DeviceAdapter", "${item.name}")
        holder.typeTextView.text = item.type
```

```
    }
}
```