Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Засоби Software Development Kit для розумних пристроїв на основі технології Bluetooth Low Energy»

Виконав: студент 2 курсу, групи 1АКІТ-21м, спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології» .

_____ Пакула А.А.

Керівник: к.т.н., проф.. кафедри АІІТ

_____ Паламарчук Є.А.

«____» _____ 2022 р.

Опонент: _____

_____

«____» _____ 2022 р.

Допущено до захисту

Завідувач кафедри АІІТ

_____д.т.н., проф. Бісікало О.В.

«____» _____ 2022 р.

Вінниця ВНТУ - 2022 рік

АНОТАЦІЯ

УДК 004.588

В даній магістерській кваліфікаційній роботі розглянуто питання, пов’язані з розробкою набору програмного забезпечення орієнтованого для роботи зі смартфонами на операційній системі Android для взаємодії з смарт пристроями, використовуючи технологію Bluetooth Low Energy.

В роботі досліджено актуальність смарт пристроїв у сучасному світі з використанням мобільних застосунків для взаємодії та налаштування. Приведено приклади аналогів та проаналізовано способи їх використання.

Проаналізовано існуючі методи, підходи, технології та інструменти для розробки ефективного програмного забезпечення для передачі даних через протокол Bluetooth з низьким енергоспоживанням.

Запропонований підхід дозволив спроектувати та розробити набір програмного забезпечення для взаємодії з різними типами смарт пристроїв, використовуючи архітектурні підходи, методології, алгоритми та способи обробки даних. Було розроблено програмний модуль для визначення специфічних подій з використання моделі машинного навчання. Розроблений набір програмного забезпечення було протестоване та оптимізовано.

Ключові слова: Bluetooth, BLE, набір програмного забезпечення, смарт пристрій.

ABSTRACT

UDC 004.588

This master's degree work deals with issues related to the development of a set of software oriented for working with smartphones on the Android operating system for interaction with smart devices using Bluetooth Low Energy technology.

The work examines the relevance of smart devices in the modern world using mobile applications for interaction and configuration. Examples of analogs are given and methods of their use are analyzed.

Existing methods, approaches, technologies, and tools for developing efficient software for data transfer over Bluetooth Low Energy are analyzed.

The proposed approach made it possible to design and develop SDK for interaction with various types of smart devices using architectural approaches, methodologies, algorithms and data processing methods. A software module was developed to identify specific events using a machine learning model. The developed SDK were tested and optimized.

Keywords: Bluetooth, BLE, software development kit, smart device.

Відгук

Керівника магістерської кваліфікаційної роботи

студента Пакула Антон Артурович група 1АКІТ-21м

на тему: Засоби Software Development Kit для розумних пристроїв на основі технології Bluetooth Low Energy

Представлена магістерська кваліфікаційна робота присвячена розробці набору програмного забезпечення для взаємодії зі смарт пристроями використовуючи технологію Bluetooth Low Energy. Враховуючи стрімкий розвиток сфери розумних пристроїв постає необхідність в проєктуванні відповідних комплектів програмного забезпечення з використанням сучасних підходів та технологій, що забезпечують максимальну ефективність й продуктивність, а також дають можливість зручного оновлення й розширення функціоналу.

В якості новизни роботи варто відзначити запропонований підхід для побудови набору програмного забезпечення на основі багатомодульної архітектури з використанням сучасних інструментів й алгоритмів, що на відміну від традиційних рішень, дозволяє отримати високу універсальність системи, зокрема можливість взаємодіяти з різними типами й моделями смарт пристроїв та значно зменшити витрати на розробку й подальшу підтримку спеціалізованого програмного забезпечення.

Позитивними сторонами дипломної роботи є системність, чітка послідовність викладення матеріалу та детальний аналіз поставленої задачі, що дозволяє ознайомитись та вивчити різні аспекти даної роботи.

Цінність роботи полягає у розробці методичного, алгоритмічного та програмного забезпечення, що дозволяють розробити універсальний та ефективний набір програмного забезпечення для взаємодії з різними типами смарт пристроїв.

Студентом було проведено аналіз та порівняння можливих методів та інструментів для вирішення поставленої задачі й обрано оптимальний варіант. Під час виконання магістерської кваліфікаційної роботи студент Пакула А.А. проявив себе кваліфікованим спеціалістом здатним до самостійної роботи та обґрунтування складних технічних рішень.

Вважаю, що магістерська робота відповідає вимогам магістерської кваліфікаційної роботи, а її автор, Пакула Антон Артурович, заслуговує оцінку «відмінно» та присвоєння кваліфікації: ступінь вищої освіти магістр, спеціальність 151 – «Автоматизація та комп'ютерно інтегровані технології», освітня програма «Інтелектуальні комп'ютерні системи».

Керівник магістерської кваліфікаційної роботи

Д.т.н., професор кафедри АІІТ		Євген ПАЛАМАРЧУК
(посада, науковий ступінь, вчене звання)	(підпис)	(Ім'я ПРІЗВИЩЕ)

Відгук

опонента на магістерську кваліфікаційну роботу

студента Пакула Антон Артурович група 1АКІТ-21м

на тему: Засоби Software Development Kit для розумних пристроїв на основі технології Bluetooth Low Energy

Магістерська кваліфікаційна робота, яку подано на рецензію, виконана у встановлений термін та в повному обсязі згідно актуальних вимог. Робота складається з наступних частин: аналіз предметної області та обґрунтування доцільності розробки, аналіз технічних інструментів та підходів для розробки набору програмного забезпечення, програмна реалізація SDK, тестування програмного забезпечення, економічний розділ, висновки, додатки.

Актуальність розробки обумовлена потребою створення універсального та зручного у використанні набору програмного забезпечення, що дозволить ефективно взаємодіяти з різними типами й моделями смарт пристроїв.

В даній роботі розроблено набір програмного забезпечення, що надає зручні інструменти для взаємодії зі смарт пристроями використовуючи технологію Bluetooth Low Energy. Програма реалізує високий рівень асинхронності виконання коду та використовує алгоритми для обробки отриманих даних, що значно збільшує продуктивність процесу роботи зі смарт пристроями. Розробка виконана в середовищі AndroidStudio з використанням мови програмування Kotlin.

Основні результати досліджень магістерської кваліфікаційної роботи пройшли апробацію на XV міжнародній науково-практичній конференції «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА АВТОМАТИЗАЦІЯ -2022».

До недоліків роботи можна віднести стислий опис адаптації функціоналу набору програмного забезпечення на основі даних користувача. Однак, дане зауваження не є критичним, робота написана на професійному рівні та її якість не викликає сумніву.

Магістерська кваліфікаційна робота виконана у відповідності зі сформованим завданням та з дотриманням всіх вимог. Робота ***заслуговує*** оцінки «**відмінно**», а її автор – присвоєння кваліфікації: ступінь вищої освіти магістр, спеціальність 151 – «Автоматизація та комп'ютерно-інтегровані технології», освітня програма «Інтелектуальні комп’ютерні системи».

Опонент на магістерську кваліфікаційну роботу

Доцент кафедри КН — Олег КОЛЕСНИЦЬКИЙ

(посада, науковий ступінь, вчене звання) (підпис) (Ім’я ПРІЗВИЩЕ)

ЗМІСТ

ВСТУП

Актуальність. В наш час, важко оцінити вплив технологій на наше повсякденне життя. На сьогоднішній день смартфони є практично в кожного, вони повністю домінують в багатьох сферах нашого життя: спілкування, розваги, навчання, здоров'я. Однак, не лише мобільні телефони розвинулись за останні роки, також з'явилось багато й інших розумних пристроїв, які значно спрощують повсякденне життя. Сьогодні багато людей використовують розумні годинники, які можуть аналізувати активність користувача, ритм серцебиття, приймати виклики, показувати мапу тощо. Таких смарт пристроїв існує досить багато та постійно з'являються нові: розумні ваги, розетки, лампи, пилососи, та й навіть розумні будинки. Взаємодія з такими пристроями зазвичай відбувається через мобільні додатки, завдяки їх зручності та підтримці технології Bluetooth й можливості підключення до мережі інтернет, для передачі та отримання даних.

Однією з найважливіших частин сучасних розумних пристроїв – це додатки для взаємодії з ними. В нашу епоху стрімкого технічного прогресу існує безліч різноманітних пристроїв, від розумних розеток до розумних автівок й будинків. Для таких складних систем важливо надати зручний та зрозумілий інтерфейс взаємодії. Ефективність додатку в першу чергу залежить від оптимізації взаємодії зі смарт пристроями, зчитування, надсилання, та швидкість обробки даних. Саме тому, в більшості випадків компанії виробники смарт пристроїв надають розробникам таких додатків оптимізований SDK (Software Development Kit), який дозволяє взаємодіяти з пристроями на програмному рівні, змінювати конфігурації, зчитувати дані тощо.

Часто смарт пристрої можуть не сильно відрізнятись один від одного на рівні задіяних модулів та технічних характеристик, однак виконувати різні дії. Це досягається саме на програмному рівні шляхом обробки даних, які можуть використовуватись для різних цілей. Таким чином, головною проблемою для компаній, що розробляють велику кількість смарт пристроїв різних типів, є

необхідність надання набору програмного забезпечення для кожного розробленого типу смарт пристрою. В подальшому, для інженерів та розробників з боку компанії-розробника, буде дуже важко підтримувати та розширювати функціональні можливості комплектів програмного забезпечення.

Таким чином, розробка та дослідження набору програмного забезпечення для взаємодії зі смарт пристроями з використанням технології Bluetooth Low Energy є актуальним завданням.

Мета і завдання дослідження є реалізація системи для набору програмного забезпечення, що підтримуватиме різні типи смарт пристроїв, незалежно від їх призначення та шляхів використання, покладаючись лише на технічні характеристики.

Об’єктом дослідження є процес розробки універсального та зручного SDK з використанням технології BLE (Bluetooth Low Energy) для передачі даних та підтримкою різних типів смарт пристроїв.

Предметом дослідження є сучасні та актуальні методи, засоби та інструменти для розробки SDK (Software Development Kit).

Для досягнення поставленої мети необхідно розв’язати такі задачі:

- проаналізувати існуючі підходи, методи й інструменти для розробки набору програмного забезпечення на мобільні пристроїв;
- запропонувати власний підхід розробки SDK для взаємодії з різними типами смарт пристроїв, на основі аналізу існуючих методів й інструментів розробки програмного забезпечення та обґрунтувати його ефективність;
- на основі обраного підходу розробити комплект програмного забезпечення;
- протестувати та перевірити ефективність розробленого програмного забезпечення.

Методи дослідження. Для вирішення проблеми було використано: методи зберігання, поширення та компіляції програмного коду, методи моделювання архітектури програмного забезпечення, методи збереження й обробки даних, методи обміну даними з використанням BLE, методи взаємодії з сервером та методи тестування.

Основні науково-технічні результати:

- запропоновано власний підхід до розробки універсального комплекту програмного забезпечення для передавання та обробки даних з використанням технології BLE, який дозволяє працювати з різними типами і моделями смарт пристроїв, використовуючи базові функціональні модулі цих пристроїв;
- розроблено SDK на основі запропонованого підходу, який на відміну від інших, дозволяє взаємодіяти з різними пристроями, які відрізняються специфікою використання, виглядом або функціоналом.

Практична цінність даної роботи полягає в тому, що розроблений комплект програмного забезпечення легко використовувати на практиці та розширювати функціонал в подальшому.

Апробація результатів дослідження: основні результати досліджень даної магістерської кваліфікаційної роботи були представлені на XV міжнародній науково-практичній конференції «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА АВТОМАТИЗАЦІЯ -2022» [1].

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану та тенденцій сфери смарт пристроїв

Розумний або смарт пристрій — це електронний пристрій, зазвичай підключений до інших пристроїв або мереж через різні бездротові протоколи, такі як Bluetooth, Zigbee, NFC, Wi-Fi, LiFi, 5G тощо, який може працювати певною мірою інтерактивно та автономно. Кілька відомих типів розумних пристроїв: смартфони, розумні автомобілі, розумні термостати, розумні дверні дзвінки, розумні замки, розумні холодильники, фаблети та планшети, розумні годинники, розумні ремінці, розумні брелоки, розумні окуляри та багато інших. Термін також може стосуватися пристрою, який демонструє деякі властивості повсюдного обчислення, включаючи (хоча і не обов’язково) машинне навчання.

Оскільки в наш час технології невпинно розвиваються, тому щороку з’являються нові види смарт пристроїв. Сфера розумних пристроїв постійно росте та продовжує залучати ще більшу кількість людей. На сьогодні вже існує велика кількість розповсюджених типів смарт пристроїв, такі як наприклад: смарт годинники, фітнес-трекери, голосові помічники, смарт розетки, смарт іграшки, розумні автомобілі, розумні будинки тощо. Такий стрімкий розвиток сфери смарт пристроїв в першу чергу пов’язаний із розвитком технологій та можливості зменшення їх розмірів. Сьогодні комплексні системи, що містять багато високотехнологічних сенсорів, не більші за розміром ніж долонь людини.

Великобританія, Китай, Італія, Іспанія та Данія перевищують середній світовий відсоток населення, що володіє смарт пристроями того чи іншого типу, і ця тенденція продовжує зростати. У період з січня по червень 2022 року продажі розумних продуктів принесли 13,6 мільярдів доларів США на семи основних європейських ринках (Бельгія, Франція, Німеччина, Велика Британія, Італія, Нідерланди, Іспанія). Це фактично на -7,2% менше, ніж за той самий період 2021 року, або на -3,7%, якщо виключити смарт-телевізори.

Домашні розваги та офіс все ще є найбільшою категорією, на яку припадає 58,2% від загальної вартості смарт-ринку (за рахунок смарт-телевізорів) у цих країнах за перше півріччя. Проте категорії домашньої автоматизації та основної побутової техніки показали найкраще зростання з початку року в першому півріччі 2022 року на +9,9% і +2,1% відповідно [2].

Хоча й цей рік має нижчі продажі, ніж попередній, однак, варто також зазначити, що цьому є кілька важливих причин. Війна Росії проти України призвела до економічних санкцій проти багатьох країн, різкого зростання цін на товари та збоїв у ланцюзі поставок, що вплинуло на багато ринків у всьому світі. Однак, незважаючи на це, за оцінкою інсайдерської компанії TBRC (The business research company) [3], очікується, що світовий ринок розумних пристроїв для дому зросте з 78,44 мільярда доларів США у 2021 році до 92,48 мільярда доларів США у 2022 році при середньорічному темпі зростання (CAGR) 17,9%.

1.2 Принцип взаємодії смарт пристроїв із набором програмного забезпечення

Кожен смарт пристрій містить базове програмне забезпечення, яке координує та поєднує функціонал з доступних на ньому сенсорів. В більшості випадків програмне забезпечення смарт пристроїв не виконує складних процесів, як обробка даних, взаємодія з сервером, тощо. Пристрій може надсилати отримані з сенсорів дані до смартфону та приймати дані для зміни конфігурації роботи цих сенсорів.

Набір програмного забезпечення відіграє важливу роль, оскільки виступає в ролі «мосту» при взаємодії між смартфоном та пристроєм. Отримані дані від розумного пристрою спочатку надходять до розробленого пакету програмного забезпечення та після обробки надсилаються в зручному для подальшого використання форматі до програмного коду додатку. При записі даних або ж

надсиланні команд до смарт пристрою, додаток використовує відповідні методи, описані в наборі програмного забезпечення, та вже на боці SDK формується команда й надсилається запит.

Однією з важливих причин використання набору програмного забезпечення, замість реалізації всього функціоналу на боці додатку, полягає у великий кількості класів та програмного коду необхідного для надсилання й обробки даних. Оскільки додаток використовує готові набору програмного забезпечення в скомпільованому форматі, менше пам’яті використовується та програмний код мобільного додатку легше підтримувати.

Використання SDK також необхідне для збереження приватності специфічних даних та форматів команд й даних, що використовуються для взаємодії зі смарт пристроєм. Таким чином можна бути впевненим, що сторонній розробник буде використовувати смарт пристрій лише за призначенням, тощо.

Таким чином необхідність використання набору програмного забезпечення, як «мосту» між смарт пристроєм та смартфоном, є необхідним, оскільки виконує усі складні процеси та захищає специфічні дані.

1.3 Призначення набору програмного забезпечення

Компаніям, що виробляють сенсори та прототипи смарт пристроїв для подальшого продажу необхідно надавати також й SDK для взаємодії з функціоналом. Частіше за все, для кожного прототипу й моделі сенсорів розробники таких компаній використовують розроблений раніше код та адаптують під специфіку нових пристроїв й сенсорів.

У випадках, коли певний тип смарт пристрою або сенсору є основним продуктом компанії – такий підхід є оптимальним. Оскільки це дозволяє зосередитись на покращені та розширені програмного забезпечення для основного

продукту. Однак, у випадках, якщо компанія виготовляє різні за специфікою та призначенням сенсори й пристрої, такий підхід займає багато часу та створює проблеми з подальшим контролем версій й підтримкою розробленого програмного забезпечення.

Призначення розробленого SDK полягає в універсальності та повній сумісності з основними типами сенсорів, що виготовляються компанією. Усі прототипи смарт пристроїв в загальному складаються з різних типів сенсорів та поєднають процеси їх роботи на програмному рівні прошивки. Набір програмного забезпечення підтримує усі розроблені компанією типи сенсорів та адаптує обробку даних до специфіки кожного з них.

Такий підхід у використанні універсального SDK дозволяє уникнути витрати зайвих ресурсів, оскільки немає необхідності у розробці програмного забезпечення для кожного окремого пристрою.

1.4 Основні типи з’єднань смарт пристроїв

Усі смарт пристрої, як правило, підключені до інших пристроїв або мереж через різні бездротові протоколи (наприклад, Bluetooth, Zigbee, Wi-Fi, LiFi або 5G), які можуть працювати певною мірою інтерактивно та автономно. В переважній більшості використовуються технології бездротового зв’язку та в значній мірі визначається сферою використання та призначенням самого пристрою.

Найпоширенішими серед усіх інших є Bluetooth та Wi-Fi. В залежності від сфер використання кожен протокол має свої переваги та недоліки.

1.4.1 Переваги й недоліки використання протоколу Bluetooth.

Bluetooth — це протокол бездротової технології для полегшення обміну даними між підключеними пристроями на короткій відстані. Він покладається на фізичну близькість і використовує радіохвилі НВЧ (надвисокої частоти) в діапазоні від 2400 до 2485 ГГц.

Bluetooth був розроблений для створення персональних мереж (PAN) і встановлення з'єднань між комп'ютерними пристроями, такими як ноутбуки, смартфони та периферійні пристрої.

Реальні приклади використання технології Bluetooth включають розумні будинки з взаємопов'язаними пристроями, системи управління будівлями (BMS), навушники, колонки, фітнес браслети, розумні годинники тощо.

Головні переваги використання технології Bluetooth:

- стабільна передача даних невеликих об'ємів без втрат. Bluetooth забезпечує пропускну здатність близько 1 Мбіт/с, чого достатньо для передачі числових значень із датчиків та сенсорів;

- низьке споживання електроенергії, що дозволяє значно збільшити автономність смарт пристроїв. Подібно до Wi-Fi, він випромінює сигнали з частотою 2,4 ГГц, але вони поширюються на нижчий діапазон.

Головні недоліки:

- не підходить для передачі великих об'ємів даних на високій швидкості, оскільки значні втрати даних будуть неминучі;

- невеликий діапазон сигналу, який залежить від типу передавача та можливих перешкод на шляху.

Таким чином, використання протоколу Bluetooth доцільно у випадках, коли смарт пристрій надсилає не великі об’єми даних, використовується за межами дому та мережі Wi-Fi, або ж задля збільшення автономності пристрою.

Для подальшої розробки використовується саме технологія BLE, оскільки підходить для формату даних, що надсилаються зі сканерів смарт пристроїв, а також значно збільшує час автономної роботи пристроїв.

1.4.2 Переваги й недоліки використання Wi-Fi.

Wi-Fi — це технологія бездротової мережі, яка базується на стандартах IEEE 802.11, щоб забезпечити обмін інформацією між пристроями. Для передачі інформації між пристроями використовуються різні діапазони радіохвиль, що працюють на частотах 2,4 ГГц або 5 ГГц.

Усі сучасні пристрої, такі як комп’ютери, ноутбуки, смарт-телевізори та смартфони, мають вбудовану функцію Wi-Fi. Технологія базується на протоколі TCP-IP. Згідно з цим, необхідно, щоб кожен пристрій отримав свою IP-адресу, і також автентифікувати себе в мережі.

Wi-Fi широко використовується для потокового відео, додатків віртуальної й доповненої реальності та в усіх інших сферах, що потребують швидку передачу великої кількості даних.

Головні переваги використання технології Wi-Fi:

- висока швидкість передачі даних. Wi-Fi надає різні швидкості. HaLow або інші стандартні версії передають дані зі швидкістю 347 Мбіт/с;
- захищеність даних. WiFi має добре розроблені протоколи безпеки, такі як WEP, WPA, WPA2 і WPA3. З них найбільш розвинений WPA3 й ідеально підходить для передачі важливих даних.

Головні недоліки:

- високе споживання електроенергії. Пристрої Wi-Fi випромінюють потужні сигнали, які поширюються значно далі, ніж BLE. Для цього потрібне постійне джерело живлення, що супроводжується додатковими витратами.

Підсумовуючи, використання технології Wi-Fi доречно при необхідності передавання великих об’ємів даних за короткий інтервал час. Технологію Wi-Fi використовують такі смарт пристрої як: смарт ТВ, голосові асистенти, смарт годинники, сенсори смарт будинку тощо.

1.5 Bluetooth Low energy

Bluetooth Low Energy (BLE) – це цифровий радіо протокол передачі даних від одного пристрою до іншого. Принцип роботи BLE описаний у його назві: Low Energy. Протокол передбачає передачу даних короткими пакетами у разі потреби та вимиканням передавача, якщо таких даних немає. Низьке енергоспоживання частково досягається застосуванням цього принципу. Замість класичного тандему у звичайному Bluetooth, пристрої BLE зв'язуються один з одним лише за необхідності надсилання або отримання інформації.

Один із пристроїв є центральним, а інші периферійними. Центральний пристрій може містити декілька з'єднань з периферійними, але периферійний пристрій може містити лише одне з'єднання. Наприклад, смартфон виступає як центральний пристрій, який може з'єднатися з периферійним: блютуз-колонкою, лампою, розумним годинником і фітнес-трекером. Всі ці пристрої можуть з'єднатися лише з телефоном. Смартфони 5.0 та вище (API 21) можуть виступати і як периферійні пристрої.

У центрального пристрою є два режими: сканування та з'єднання. Периферійний пристрій має інші два режими: оголошення та з'єднання.

Протокол BLE строго структурований за принципом комунікації з іншими пристроями. Спочатку девайси вивчають доступні послуги для відправки/прийняття даних; невід'ємна частина цих сервісів – їх характеристики (characteristics), визначальні тип даних майбутньої передачі. Характеристики з міркувань наочності можуть мати у своєму складі описи-дескриптори (descriptors), які допомагають визначити тип даних. Наприклад, сервіс "Монітор частоти серцебиття" (Heart rate monitor) - серед його характеристик присутні такі, як "вимір пульсу".

Більшість API для Bluetooth LE дозволяють шукати локальні пристрої та визначати доступні в них послуги, характеристики та дескриптори.

ОС Android 4.3 (API 18) представляє вбудовану підтримку Bluetooth Low Energy та API, за допомогою якого програми можуть використовувати пошук пристроїв, запит послуг та читання/запис характеристик. На відміну від класичного Bluetooth, BLE покликаний забезпечити значно менше енергоспоживання. Це дозволяє програмам для Android спілкуватися з BLE-пристроями, які мають низькі вимоги до живлення, таких як датчики, монітори серцевого ритму, фітнес-пристрої тощо.

1.3.1 Ключові терміни та поняття

Generic Attribute Profile (GATT) – профіль GATT є загальною специфікацією для надсилання та отримання коротких фрагментів даних, відомих як "атрибути" через BLE-з'єднання. Всі поточні LE-профілі програм засновані на GATT. Творці BLE визначили багато профілів для низькоенергетичних пристроїв. Профіль є визначення того, як пристрій працює в конкретному додатку. Пристрій може реалізовувати більше одного профілю. Наприклад, пристрій може містити профілі пульсометра та датчика рівня заряду батареї.

Attribute Protocol (ATT) – GATT будується з урахуванням протоколу атрибутів ATT. Це також стосується GATT/ATT. ATT оптимізовано до роботи на BLE-пристроях. Для цього використовується настільки мало байтів, наскільки це можливо. Кожен атрибут ідентифікується унікальним універсальним ідентифікатором (UUID), який є стандартизованим 128-бітовим рядковим ідентифікатором, що використовується для однозначної ідентифікації інформації. Атрибути переносяться за допомогою ATT у вигляді характеристик та послуг.

Характеристика (Characteristic) – містить одне значення і від 0 до N дескрипторів, що описують значення характеристики. Характеристику можна розглядати як тип, аналог класу.

Дескриптор (Descriptor) може містити опис, прийнятний діапазон значень або одиницю вимірювання, конкретні значення характеристики.

Послуга (Service) – це набір параметрів. Наприклад, послуга під назвою "пульсометр" включає таку характеристику, як "Вимір пульсу". Список існуючих на основі GATT (Generic Attribute Profile) профілів та послуг можна знайти на офіційному вебсайті [21].

1.3.2 Ролі та обов'язки при взаємодії Android з BLE-пристроєм

Центральна/периферійна роль. Це стосується самого BLE-з'єднання. Пристрій у центральній ролі сканує, шукає оголошення, а пристрої у периферійній ролі створюють оголошення(advertisements).

GATT-сервер/GATT-клієнт - визначає, яким чином два пристрої спілкуються один з одним, коли вони встановили зв'язок.

Для прикладу візьмемо Android смартфон і фітнес-трекер, який є BLE-пристроєм. Смартфон підтримує центральну роль, а фітнес-трекер підтримує периферійну роль. Щоб встановити BLE-з'єднання, необхідно щоб було по одному

пристрою, який підтримує кожну з ролей, оскільки два периферійні пристрої не можуть спілкуватися один з одним, так само як і два центральні. Як тільки телефон і трекер налагоджують зв'язок, починається передача метаданих GATT.

В залежності від того, які дані передаються між центральним та периферійним пристроєм, обидва пристрої можуть виступати у ролі серверу. Наприклад, якщо смарт пристрій надсилає дані датчика до смартфону, то смарт пристрій виступатиме в ролі серверу. Однак, якщо фітнес-трекер повинен отримувати оновлення з телефону, то в ролі серверу тепер буде смартфон.

1.6 Висновки

В даному розділі було розглянуто актуальність сфери смарт пристроїв, призначення й принцип взаємодії з набором програмного забезпечення, розглянуто методи зв’язку смарт пристроїв та детально описано головні принципи роботи й терміни технології BLE. Було проаналізовано переваги й специфіку використання технології Bluetooth з низьким енергоспоживанням (BLE). Розуміння загальних термінів та процесів взаємодії з технологією Bluetooth дуже важливі для створення якісного та конкурентноспроможного продукту.

Визначено основний принцип роботи з набором програмного забезпечення та проаналізовано його роль при взаємодії зі смарт пристроями.

Також було описано призначення та переваги набору програмного забезпечення. До головних переваг відносяться: економія ресурсів на розробку й підтримку зосередженого на роботу під один тип смарт пристрою набору програмного забезпечення, універсальність та висока якість програмного коду.

2 АНАЛІЗ ТА ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

2.1 Вибір платформи й цільової операційної системи

При початку розробки комплекту ПЗ перед розробниками стоїть вибір: на які платформи та операційні системі буде сфокусоване подальше використання SDK. Найпоширеніші операційні системи – Android та IOS. Оскільки використання мобільних пристроїв для керування та проглядання інформації зі смарт пристроїв є найзручнішим.

Android - операційна система і платформа для мобільних телефонів та планшетних комп'ютерів, створена компанією Google на базі ядра Linux. Підтримується альянсом Open Handset Alliance (OHA).

Хоча Android базується на ядрі Linux, він стоїть дещо осторонь Linux-спільноти та Linux-інфраструктури. Базовим елементом цієї операційної системи є реалізація Dalvik віртуальної машини Java, і все програмне забезпечення і застосування спираються на цю реалізацію Java. У 84 % смартфонів, проданих у 3-му кварталі 2014 року, було встановлено операційну систему Android. У березні 2017 року ОС Android стала найрозповсюдженішою ОС, з якої виходили в інтернет. Так, 37,93 % користувачів заходили в інтернет із Android'a, а з Windows — 37,91 % користувачів. В Азії показники ще вищі — 52,2 % і 29,2 % відповідно.

Деякі користувачі відзначають, що Android проявляє себе краще одного зі своїх конкурентів, Apple iOS, в ряді особливостей, таких як вебсерфінг, інтеграція з сервісами Google і інших. Також Android, на відміну від iOS, є відкритою платформою, що дозволяє реалізувати на ній більше функцій:

- Попри початкову заборону на установку програм з «неперевірених джерел» (наприклад, з карти пам'яті), це обмеження відключається штатними засобами в налаштуваннях пристрою, що дозволяє встановлювати програми на телефони та планшети без інтернет-

підключення (наприклад, користувачам, які не мають Wi-Fi-точки доступу і не бажають витрачати гроші на мобільний інтернет, який зазвичай коштує дорого), а також дозволяє будь-кому безкоштовно писати програми для Android і тестувати на своєму пристрої.

- Android доступний для різних апаратних платформ, таких як ARM, MIPS, x86.
- Існують альтернативні Google Play магазини додатків: Amazon Appstore, Opera Mobile Store, GetUpps!, F-Droid.

iOS — це власницька мобільна операційна система від Apple. Розроблена спочатку для iPhone, згодом також вдосконалена для використання на iPad (до літа 2019, коли на конференції Apple WWDC було представлено нову OS для iPad — iPadOS), iPod Touch та Apple TV (до 9 вересня 2015, коли на спеціальному заході Apple було представлено tvOS). Apple не дозволяє роботу ОС на мобільних телефонах інших фірм (відома як iPhone OS до червня 2010 року). iOS є похідною від OS X, отже, є за своєю природою Unix-подібною операційною системою.

Станом на 2019 рік інтернет-магазин App Store містить понад 2 мільйони застосунків для iOS, які були завантажені понад 15 мільярдів разів. Станом на травень 2010 року, iOS становив 15,4 % ринку операційних систем для смартфонів, третій після Symbian і Blackberry.

Головні характеристики, що відображають переваги й недоліки кожної платформи:

1) Поширеність та доступність платформи.

Переважна більшість сучасних смартфонів саме на платформі Android. Це пояснюється тим, що операційна система Android на відміну від iOS має відкритий код та тим самим дозволяє використовувати ОС різним виробникам смартфонів. Завдяки доступності платформи існує й багато виробників та моделей смартфонів на платформі Android, що робить їх більш доступними.

2) Продуктивність смартфонів.

В залежності від цінової категорії, флагмани на платформі Android мають потужні процесори та великі об’єми оперативної пам’яті. Однак, в той час, як дорожчі моделі надають велику продуктивність та загальну ефективність, дешевші пристрої мають низьку ефективність. З іншого боку, смартфони на платформі iOS в середньому є високо-продуктивними. Навіть старші покоління телефонів Apple в більшості випадків ефективніші ніж нові смартфони на платформі Android в низькій ціновій категорії.

3) Надійність.

Надійність смартфонів в значній мірі визначається терміном підтримки оновлень від виробників. Так, деякі моделі смартфонів на операційні системі Android можуть отримувати лише незначні оновлення та ігнорувати нові версії ОС. Флагмани від компаній Google та Samsung надають оновлення всім своїм пристроям на протязі 4-5 років, що виділяє їх серед інших компаній-виробників. Пристрої на операційні системі iOS незалежно від моделі пристрою регулярно отримують оновлення, що в значній мірі збільшує їх надійність й продуктивність на протязі років.

Підсумовуючи, для розробки набору програмного забезпечення було обрано платформу Android, оскільки ця операційна система є більш поширена та гнучка в плані розробки. При розробці програмного забезпечення необхідно звернути увагу на оптимізацію програмного коду, оскільки не усі смартфони на платформі Android надають високу продуктивність. Однак, велика кількість інструментів та фреймворків для роботи з технологією Bluetooth, асинхронного виконання коду та обробки й збереження даних надає змогу розробити швидкий й ефективний набір програмного забезпечення.

2.2 Вибір методології розробки програмного забезпечення.

Методологія розробки програмного забезпечення – це набір рекомендацій, принципів, методів, способів, які визначають, яким чином буде розроблятись ПЗ. Методологія допомагає структурувати, зробити його зручним та гнучким.

Для того щоб знизити ступінь ризику, і впорядкувати процес розробки мобільного ПЗ - широко застосовується методологія Agile з її адаптивним (допустимість частих змін), ітеративно-інкрементальним (зворотний зв'язок із замовником на кожній ітерації, і множинні релізи), кооперативним (щільне співробітництво розробників, замовника і кінцевих користувачів) і простим (легко зрозуміти, вносити зміни і розширювати) підходом до розробки.

Agile - це серія підходів до розробки програмних продуктів шляхом безперервної та швидкої поставки цінного робочого функціоналу самоорганізованої команди професіоналів у співпраці з замовником [3].

2.2.1 Методологія розробки Scrum.

Scrum – це одна з agile-методик, суть якої полягає у розробці продукту, шляхом розбиття його на певну серію умовних ітерацій однакової тривалості (Sprint). Методика фокусується на чіткому спостереженні за процесом розробки. Scrum передбачає розробку проекта невеликими ітераціями (зазвичай 2-4 тижні). По завершенню кожної ітерації замовник отримує версію продукту, що містить усі описані у беклозі спринта завдання. Головна ідея полягає в тому, що після закінчення ітерації або ж спринта, замовник отримує робочу версію продукту, до якої можна внести зміни або вказати нові побажання. Після кожної ітерації умовний цикл повторюється до тих пір, поки не буде розроблено програмне забезпечення, що повністю задовольняє вимоги замовника.

У загальному випадку реалізація Scrum зображена на рис. 1.1. Як можна побачити з цього рисунка, в ньому задіяні певні інструменти:

- Беклог продукту (Product Backlog) – сюди додаються усі заплановані задачі та історії користувачів (User Stories);

- Планування спринту – під час планування команда відбирає певну кількість задач, які повинні виконати на протязі спринта;

- Беклог спринта (Sprint Backlog) – містить усі заплановані на спринт задачі;

- Версія продукту готова до релізу (Releasable Software) – версія, яку можна надсилати замовнику або ж впроваджувати в публічний доступ.

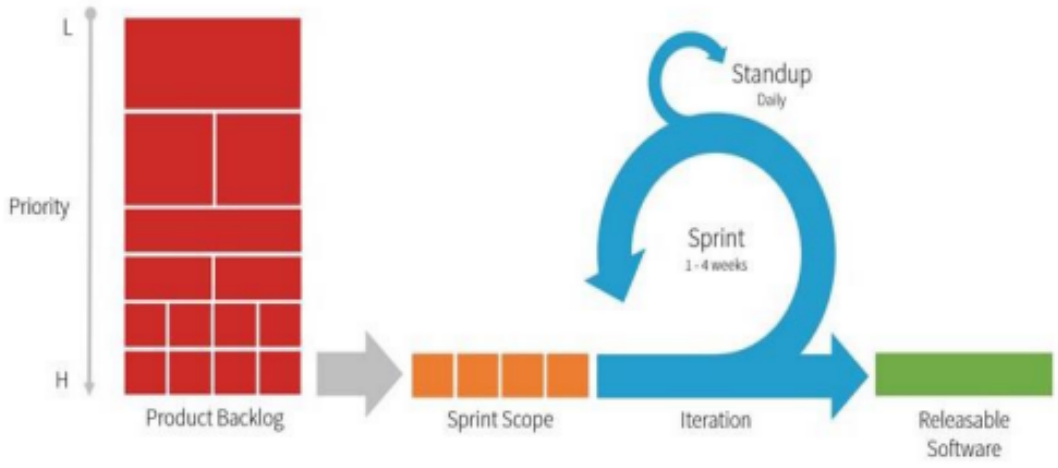


Рисунок 1.1 – загальна схема реалізації методології Scrum

2.2.2 Методологія розробки XP.

Agile-методика XP (Extreme Programming) – це системний підхід до програмування, зосереджений на інженерній стороні розвитку продукту. XP практики, що значно покращують якість продукту:

- Автоматизоване тестування;

- Рефакторинг коду (Code refactor);

- Перегляд коду (Code Review);

- Парне програмування.

На відміну від Scrum, який зосереджений на регулярних оновленнях продукту, XP сфокусована на процесі розробки та полягає в безперервній інтеграції (Continuous Integration (CI)).

Безперервна інтеграція – це інтеграція окремих частин коду програми між собою. Головна суть полягає в тому, щоб часто вносити зміни в проект та тестувати зміни на кожній інтеграції. Процес безперервної інтеграції зображено на рис 1.2.

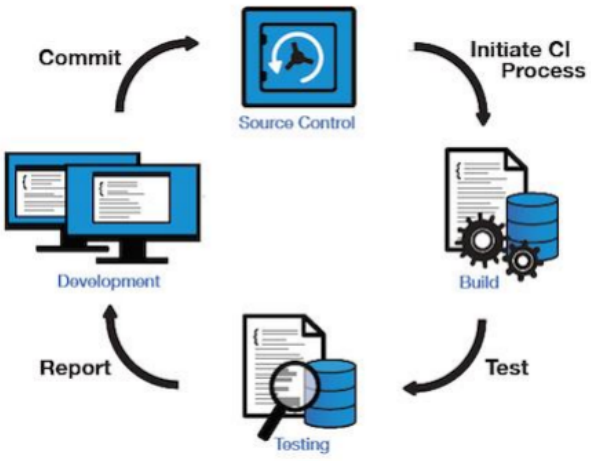


Рисунок 1.2 – схема циклу безперервної інтеграції

Безперервна інтеграція умовно поділяється на декілька етапів:

- Розробники роблять зміни в програмному коді продукту та вносять зміни до системи контролю версій;
- Система контролю версій повідомляє CI про внесені зміни;
- CI компілює програмний код;
- CI тестує скомпільоване програмне забезпечення заданим йому набором тестів;
- CI повідомляє розробників (частіше за все лист на пошту) про результати тестування.

2.2.3 Адаптація методик

Досить часто в мобільній розробці класичні методики необхідно адаптувати з урахуванням особливостей проекту, команди і замовника.

Нижче перелічені деякі варіанти такої адаптації:

- Модифікація тимчасових рамок agile-практик. Наприклад, скорочення часу спринту до 1 тижня - як правило, замовник додатку бажає отримувати нові релізи, а команда отримувати зворотній відгук, як можна частіше. Інший приклад: ретроспектива може бути раз на місяць або кожну ітерацію, в залежності від потреб проекту;
- Модифікація реалізації agile-практик. Наприклад, проведення асинхронних стендапів через Slack / Stride / ін. Якщо команда працює

віддалено - це особливо актуально;

- Модифікація набору agile-практик. Не завжди потрібно включати в роботу відразу всі інструменти обраної методики. Тому спочатку можуть бути обрані лише найнеобхідніші і «працюючі».

Комплект ПЗ може піддаватись внутрішнім або зовнішнім невизначеностям, таким як нечітко визначені характеристики фінального продукту (смарт пристрою) або часта зміна потреб розробників, тому цілком розсудливо буде планувати його життєвий цикл, використовуючи будь-яку agile-методику, або їх поєднання. Розповсюдженим є застосування адаптованих версій Scrum і XP, або ж класичного Scrum.

2.2.4 Обґрунтування вибору методології розробки.

Для розробки даного продукту було обрано методологію Scrum. Невід'ємною перевагою Scrum є ітеративна розробка спринтами, що дозволяє регулярно отримувати відгуки від замовника. Ця методологія добре підходить для розробки SDK оскільки регулярно вносяться певні зміни або нові типи пристроїв, які потрібно підтримувати. Команда зручно може відстежувати завдання, які потрібно виконати в певному спринті, а клієнт по завершенню кожного спринта отримує нову версію SDK з описом всього, що було зроблено. Для створення, керування та відстежування спринтів й задач використовується сервіс Jira.

Jira – це комерційна система відслідковування помилок, призначена для організації взаємодії з користувачами хоча в деяких випадках використовується і для управління проектами. Розроблена компанією Atlassian в 2002 році. Має простий веб-інтерфейс та проста у використанні. Jira надає зручний функціонал для створення та подальшої підтримки Scrum проектів, що значно полегшує роботу команди та дозволяє зосередитись на роботі.

2.3 Вибір мови програмування.

В залежності від обраної сфери для якої буде використовуватись SDK, слід правильно вибрати мову програмування. В даній роботі буде досліджуватись нативний підхід розробки SDK для мобільних пристроїв на ОС Android. Оскільки нативний підхід до розробки є найефективнішим, тому для написання будуть використовуватись відповідні до операційної систем мови програмування. Найпоширеніші нативні мови програмування для ОС Android: Java та Kotlin.

Java – з’явилась в середині 1990-х років й була створена Джеймсом Гослінгом з Sun Microsystems. Згодом Sun Microsystems була куплена Oracle. Java отримала широку популярність у всьому світі, в першу чергу через величезний набір функцій, які вона надає. Девіз Java: «Напиши один раз і запусти де завгодно» був одним з основних факторів успіху Java за останні кілька десятиліть.

Java Mobile Edition був створений для розробки додатків, які можна запускати на мобільних пристроях. Все це сильно вплинуло на стрімке зростання та стало основним фактором, пов'язаним з вирішенням прийняти Java в якості основної мови програмування для розробки мобільних додатків, що працюють на ОС Android. Додатки розроблені на Java безпечні, тому що вони працюють в ізольованому середовищі. Програма компілюються в проміжний код, відомий як байт-код. Потім цей байт-код виконується в контексті віртуальної машини Java.

Довгий час ця мова була основною мовою написання мобільних додатків на Android. Вона досі підтримується компанією Google і багато існуючих додатків розроблені саме на ній. Головна перевага мови програмування Java для розробки мобільних додатків, на сьогоднішній день, полягає в тому, що існує багато бібліотек, готових рішень та готових модулів, які можна використовувати для полегшення розробки.

Kotlin – статично типізована, об'єктно-орієнтована мова програмування, що працює поверх Java Virtual Machine і розробляється компанією JetBrains. В лютому

2016 року вийшла перша стабільна версія та вже в травні 2017 компанія Google повідомила, що мова буде додана, як стандартний інструмент в новій версії Android Studio (головне та найзручніше середовище розробки програмного забезпечення спрямованого на ОС Android). Автори ставили перед собою ціль створити лаконічнішу, безпечнішу та простішу мову, ніж Java. Наслідками спрощення, порівняно з Java стали також швидша компіляція та краща підтримка IDE.

Мова розробляється з 2010 року, публічно представлена в липні 2011. Сирцевий код було відкрито в лютому 2012. В лютому було випущено milestone 1, який містив плагін для IDEA. У червні — milestone 2 з підтримкою Android. У грудні 2012 року вийшов milestone 4 та забезпечив підтримку Java 7. В грудні 2015 року з'явився реліз-кандидат версії 1.0, а 15 лютого 2016 року відбувся реліз версії 1.0. В травні 2019 року Google оголосили пріоритетною мовою програмування саме Kotlin. Він повністю сумісний з Java кодом, що дозволяє, при бажанні, використовувати їх одночасно. Основна відмінність полягає в тому, що Kotlin містить менше «шаблонного» коду, тобто система простіша для читання, та надає більше можливостей.

Порівняння основних мов програмування Java та Kotlin наведено в таблиці 2.1.

Таблиця 2.1 – Характеристики та особливості мов програмування Java й Kotlin.

Характеристики	Kotlin	Java
Ліцензія	Мова з відкритим кодом під ліцензією Apache 2.0	Знаходиться під ліцензією GNU General Public License
Тип	Статично-типізована, об’єктно орієнтована	Статично-типізована, об’єктно орієнтована

Підтримка	Регулярно виходять нові версії з доданням нового функціоналу й інструментів.	Регулярно виходять нові версії з доданням нового функціоналу й інструментів.
Безпечність	Null-Safety, автоматичне керування ресурсами пам’яті.	Автоматичне керування ресурсами пам’яті
Сумісність	Сумісність із попередніми версіями та з мовою Java.	Сумісність із попередніми версіями.
Синтаксис коду	Простий синтаксис, легкий для розуміння.	Складний синтаксис, код важко читається.

Детальний опис характеристик описаних в таблиці 2.1 наведено нижче:

- Інструменти написання асинхронного коду. Для мобільних додатків особливо важливо підтримувати асинхронне виконання операцій, оскільки від цього напряму залежить плавність та швидкість роботи додатку. Довгий час додатки написані на мові Java використовували стандартні інструменти для написання асинхронного коду, які були незручні у використанні та важкі для синхронізації. Згодом з’явилась бібліотека RxJava, яка привнесла новий підхід «реактивного програмування» до написання асинхронного коду та інструменти для цього. Головним недоліком бібліотеки RxJava є важкість освоєння та використання. Мова Kotlin має власну бібліотеку Coroutines, в основі якої закладено парадигму структурованого паралелізму. Конструкції для асинхронного виконання коду мають чіткі точки входу та виходу та гарантують, що всі породжені потоки закінчили свою роботу коректно. Kotlin Coroutines значно простіша для вивчення та зручніша для використання.

- Безпечність. Мова Kotlin внесла можливість використання nullable й not nullable змінних, тобто це ті змінні значення яких може або не може бути null. За рахунок цього, при дотриманні основних принципів написання коду, як SOLID, KISS тощо, розробник не зіштовхнеться з так званою помилкою «на мільйон» та забезпечить надійне виконання програми. Саме довгий досвід використання Java, як основної мови програмування для розробки мобільних додатків, слугував причиною введення null safety до мови Kotlin. В Java розробник може призначити значення null будь-якій змінній і компілятор це проігнорує. Це призведе до неочікуваних збоїв виконання програми або й повного закриття додатку.
- Синтаксис коду. Оскільки мова програмування Kotlin новіша за Java, то має більш сучасний та простіший синтаксис. Це особливо ефективно для нових програмістів, оскільки синтаксис мови Java сформувався багато років тому та з новими версіями не зазнавав значних змін. Також, завдяки новим структурам даних, як наприклад: data class, objects, extensions, та іншим інструментам мови, написання звичних структур займає значно менше коду, а, відповідно й часу. Таким чином класи написані на мові Java більш громіздкі та важчі для читання коду.

Підсумовуючи описані вище характеристики мов програмування, для розробки було обрано мову Kotlin.

2.4 Вибір архітектури

Більшість архітектурних підходів для розробки мобільного програмного забезпечення не підходить для розробки SDK. Головна причина полягає в тому, що майже всі відомі архітектурні підходи, такі як: MVC, MVP, MVVM, MVI тощо, в значній мірі визначають взаємодію в коді враховуючи елемент View, тобто користувацького інтерфейсу.

Оскільки набір програмного забезпечення не містить жодного користувацького інтерфейсу та націлений для використання розробниками, то в основу архітектури було вирішено покласти багатомодульну архітектуру.

Основою багатомодульної архітектури складає модуляризація коду. Модуляризація (з англ. Modulartization) - це практика організації кодової бази на слабко пов’язані та автономні частини. Кожна частина є модулем. Кожен модуль незалежний і служить чіткій меті. Розділивши проблему на менші та легші для вирішення підпроблеми, значно зменшується складність проектування та обслуговування великої системи.

Переваг у багатомодульного підходу багато, хоча кожна з них зосереджена на покращенні зручності обслуговування та загальної якості кодової бази. До основних переваг варто віднести:

- Багаторазове використання. Такий підхід дає можливості для спільного використання коду та створення кількох програм на одній основі. Модулі фактично є будівельними блоками. Програми мають бути сукупністю своїх функцій, де функції організовані як окремі модулі. Функціональність, яку надає певний модуль, може бути або не бути ввімкненою в конкретній програмі;
- Суворий контроль видимості. Модулі дозволяють легко контролювати, до чого надається доступ іншим частинам кодової бази. Таким чином можна позначити все, крім публічного інтерфейсу, як внутрішній або приватний код, щоб запобігти його використанню поза модулем;
- Швидкість компіляції. Оскільки частіше за все зміни будуть вноситись до окремих модулів, а не до всіх одночасно, це значно пришвидшує час компіляції, що є важливим фактором для масштабних проектів. Також певні функції Gradle, такі як: інкрементна збірка, кеш збірка або паралельна збірка, у поєднанні з багатомодульною архітектурою помітно покращують продуктивність збірки всього проекту;

- Можливість подальшого масштабування проекту. У тісно пов’язаній кодовій базі одна зміна може викликати каскад змін у, здавалося б, непов’язаних частинах коду. Багатомодульний підхід охоплює принцип поділу інтересів і, отже, обмежує залежність коду.

Такий підхід є особливо ефективним для розробки обраного продукту, оскільки незалежно від моделі смарт пристрою або специфіки, SDK буде з легкістю працювати з ним використовуючи лише необхідні модулі.

2.4 Аналіз бібліотек для роботи з технологією BLE.

В першу чергу, варто зрозуміти в чому полягає перевага технології Bluetooth Low Energy над класичним Bluetooth. Частина «Low Energy» абревіатури BLE в основному полягає в тому, що у той час як Bluetooth Classic призначений для передачі безперервних потоків даних, таких як відтворення музики, BLE оптимізовано для енергоефективності. Пристрій BLE, як правило, може працювати від невеликої батареї тижнями, якщо не місяцями чи навіть роками, що робить його ідеальним для випадків використання сенсорів або Інтернету речей (IoT).

Ще одна важлива відмінність між Bluetooth Classic і BLE полягає в тому, що BLE набагато зручніший для розробників. BLE відкриває світ нескінченних можливостей, дозволяючи розробникам вказувати різноманітні власні профілі для різних випадків використання, тоді як Bluetooth Classic в основному підтримує профіль послідовного порту (SPP) для надсилання спеціальних даних.

Також варто визначити інструменти для використання BLE. Операційна система Android надає стандартну бібліотеку для взаємодії з Bluetooth low energy, однак є набагато зручніші аналоги. Стандартна бібліотека для роботи з Bluetooth на платформі Android не оновлюється та не підтримується належним чином. Бібліотека Android BLE незручна, складна й негнучка. Тому варто розглянути альтернативні варіанти. Найбільш розповсюджені бібліотеки для роботи з

Bluetooth на платформі Android – це бібліотеки BLESSED та NordicSemi BLE.

Бібліотека BLESSED — це дуже компактна бібліотека Bluetooth Low Energy (BLE) для версії Android 5 і вище, значно спрощує роботу з BLE. Багато аспектів роботи з BLE, які при використанні стандартної бібліотеки розробник повинен враховувати сам, вже реалізовано. Наприклад: постановка команд у чергу, просте сканування, безпека потоків, вищі методи абстракції та підтримка декількох одночасних підключень.

Бібліотека NordicSemi Android BLE – це бібліотека, яка надає весь функціонал, що й BLESSED, однак її перевагою є те, що вона підтримує Kotlin, що значно спрощує код та використання бібліотеки в цілому.

2.5 Вибір способу поширення SDK

Важливо обрати спосіб поширення комплекту ПЗ подальшому користувачу. Частіше за все використовуються фреймворки для автоматизованої збірки проектів на основі опису структури у файлах на мові POM, що є підмножиною XML. Найпоширенішими інструментами для поширення програмного забезпечення є Apache Maven та GitHub.

Apache Maven – це інструмент контролю версій, який надає широкий спектр можливостей для публікації та подальшого поширення готового SDK. Maven дозволяє опублікувати програмний код в окремий репозиторій. Розробникам потрібно лише додати кілька стрічок коду для імпортування комплекту ПЗ розміщеному у публічному репозиторії у власні проекти. У випадках, коли потрібно обмежити можливість використання SDK сторонніми розробниками це досягається шляхом створення приватного репозиторію та керування доступом за допомогою токенів або певних ключів. Maven надає інструменти для компіляції та конфігурації цього процесу. Так, наприклад, багатомодульне ПЗ можна скомпілювати із задіянням лише необхідних модулів й проігнорувати непотрібні.

GitHub – це теж інструмент для контролю версій програмного забезпечення, що зосереджений на роботі з кодом. GitHub надає широкий функціонал для роботи з репозиторієм та кодом загалом. Більшість середовищ розробки містять влаштовану підтримку GitHub-у та дозволяють взаємодіяти з ним шляхом командного рядку або інтерфейсу. GitHub також надає можливість створення приватних репозиторіїв з обмеженням доступу до коду, однак, на відміну від Apache Maven, не містить влаштованого компілятора. Хоча й існують фреймворки для віддаленої компіляції коду, розташованого на GitHub.

В той час, як GitHub орієнтований більше на робочі процеси з кодом та значно спрощує роботу розробникам, Apache Maven більше зосереджений на поширенні програмного забезпечення. GitHub та фреймворки для автоматизованої збірки проектів зручні у використані, однак не надають такої ж гнучкості, як влаштовані інструменти Maven.

Для поширення SDK було обрано фреймворк Apache Maven, оскільки він надає можливість створення та публікації приватних репозиторіїв, а також має гнучку конфігурацію компіляції, що дозволяє підтримувати специфічні типи пристроїв. Оскільки комплект ПЗ не буде знаходитись у відкритому доступі, то для поширення та контролю від неліцензійного передавання також використовується методика перевірки токена доступу (access token) та розроблено веб-сайт.

Задля уникнення можливості перегляду коду іншими розробниками та захисту від декомпіляції будуть використовуватись бібліотеки ProGuard для Android та iXGuard для IOS.

iXGuard – це фреймворк, який захищає мобільні додатки та SDK від зворотного проектування та втручання — двох найпоширеніших загроз для мобільних пристроїв згідно з OWASP — шляхом застосування обфускації iOS, яка вводить кілька рівнів захисту коду та впроваджує перевірки RASP.

ProGuard – це утиліта командного рядка із відкритим програмним кодом, призначена для оптимізації та обфускації Java коду. Обфускація здійснюється

шляхом перейменування імен класів, полів та методів у «беззмістовні» імена, ускладнюючи цим зворотню розробку (reverse-engineer) додатку.

2.6 Висновки

В даному розділі було проаналізовано та обґрунтовано вибір цільової платформи для розробки програмного забезпечення, методологію розробки продукту, а саме планування та постановка задач з використанням сервісу Jira, мову програмування, архітектурний підхід, бібліотеку для роботи з BLE та вибір платформ поширення SDK до кінцевих клієнтів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз функцій системи

Основним призначенням обраного набору для розробки програмного забезпечення (SDK) є взаємодія зі смарт пристроями (надсилання, зчитування та обробка отриманих даних тощо).

Набір програмного забезпечення, повинен містити базові функції для взаємодії зі смарт пристроями, незалежно від їх типу та призначення.

Основний функціонал який повинен бути реалізованим:

1) Сканування пристроїв, використовуючи Bluetooth, поблизу з можливістю фільтрування за назвою пристрою;
2) Можливість встановлення з’єднання з пристроєм та від’єднання. Обов’язковим є підтримка з’єднання та комунікації одночасно з кількома пристроями, в залежності від фізичних можливостей смартфону (в залежності від версії операційної системи Android та моделі пристрою. В загальному це чотири або п’ять одночасно під’єднаних пристроїв);
3) Надсилання та зчитування даних;
4) Ввімкнення та вимкнення сповіщень з сервісів пристрою;
5) Обробка отриманих даних;
6) Синхронізація інформації користувача з веб-сайту для перевірки придбаних пристроїв та доступного функціоналу(використання додаткового функціоналу).

Додаткові функціонал:

1) Використання машинного навчання для аналізу та відстеження певних подій для різних типів пристроїв;
2) Збереження даних на хмарі;

3) Перевірка версії програмного забезпечення смарт пристрою та можливість оновлення за необхідністю;

4) Використання алгоритмів для обробки даних з певних типів смарт пристроїв.

3.2 Розробка мінімального життєздатного продукту

Мінімально життєздатний продукт (MVP) – це продукт із достатньою кількістю функцій, щоб залучити клієнтів, які рано прийняли його, і підтвердити ідею продукту на ранніх етапах циклу розробки продукту. У таких галузях, як програмне забезпечення, MVP може допомогти групі продукту отримати відгуки користувачів якнайшвидше, щоб ітерувати та вдосконалювати продукт.

Оскільки гнучка методологія (Scrum) побудована на перевірці та ітерації продуктів на основі даних користувача, MVP відіграє центральну роль у гнучкій розробці.

Ерік Ріс, який представив концепцію мінімально життєздатного продукту як частину своєї методології “Lean Startup”, описує мету MVP так: це версія нового продукту, яка дозволяє команді зібрати максимальну кількість перевірених знань про клієнтів із найменшими зусиллями.

В попередньому розділі було проаналізовано інструменти, методології розробки та архітектурні підходи для створення SDK, що буде відповідати потребам користувачів. Інструменти, що були обрані для розробки програмного забезпечення: мова програмування Kotlin, фреймворк для написання багатопочного коду Kotlin Coroutines та бібліотека NordicSemi BLE для роботи з протоколом Bluetooth Low Energy. Архітектурою програмного забезпечення було обрано багатомодульний підхід.

MVP проект задає основу для подальшої розробки та містить в собі реалізацію архітектурного рішення, готову структуру, конфігурацію та функціонал для роботи з базовим набором пристроїв.

3.3 Налаштування конфігурацій SDK

Оскільки цільова платформа для розроблюваного набору програмного забезпечення – це Android, то перед початком розробки необхідно створити та налаштувати файл конфігурації. Згодом цей файл буде використовуватись для компіляції програмного коду за вказаними параметрами.

Програмні забезпечення, що націлені на роботу з Android пристроями повинні містити Manifest та gradle файли.

У Manifest файлі описується важлива інформація про розроблене програмне забезпечення, яка використовується для інструментів збірки Android, операційної системи Android і Google Play. Серед обов’язкових речей, які потрібно декларувати у маніфест файлі:

- Компоненти програми, які включають всі екрани, сервіси, приймачі трансляцій даних та постачальників контенту. Кожен компонент повинен визначати основні властивості, такі як ім’я його класу Kotlin або Java. Також можна задекларувати такі можливості, як конфігурації пристроїв, які програма може обробляти, і фільтри намірів, які описують, як можна запустити компонент;
- Дозволи, необхідні програмі для доступу до захищених частин системи або інших програм. А також можна вказати будь-які дозволи, які інші програми повинні мати, щоб отримати доступ до вмісту цієї програми;

- Функції апаратного та програмного забезпечення, необхідні програмі, що впливає на те, на яких пристроях можна встановити програму з Google Play.

Необхідно зазначити дозволи, які необхідно вказати в маніфест-файлі для використання розробленого набору програмного забезпечення. Обов'язковими є такі дозволи: дозвіл на використання Bluetooth, доступ до геолокації, інтернету та пам'яті телефону.

До функцій апаратного та програмного забезпечення варто віднести: ідентифікатор програми та мінімальну й цільову версію Android SDK(версія операційної системи Android).

Gradle — це система збірки, яка відповідає за компіляцію коду, тестування, розгортання та перетворення коду у файли .dex і, отже, запуск програми на пристрої. Оскільки набір програмного забезпечення спрямований на роботу з ОС Android, то обов'язково повинен містити такі gradle файли:

- build.gradle (проект) - це файл на рівні проекту, який визначає конфігурації. Цей файл застосовує конфігурації до всіх модулів у проекті програми;
- build.gradle (модуль) – це файл на рівні модулю й містить назву пакета модулю як ідентифікатор програми, усі додаткові бібліотеки, назву версії, код версії, мінімальна і цільова версія Android SDK для конкретного модуля програми;
- settings.gradle – це файл, що використовується для визначення всіх модулів, які використовуються у програмі. Оскільки для розробки було обрано багатомодульну архітектуру, необхідно вказати усі програмні модулі у settings файлі.

Основними параметрами build.gradle файлу для розробки поставленої задачі є:

- minSdkVersion=21;

- targetSdkVersion=28;

- buildTypes = debug, release;

- signingConfigs = debug, release.

Усі сторонні бібліотеки, що будуть використані у роботі також слід зазначити у файлі build.gradle. Для більшої гнучкості та зручності усі сторонні бібліотеки та їх версії були задекларовані у gradle файлі на рівні проекту.

Приклад організації сторонніх бібліотек в build.gradle файлі проекту зображено на рисунках 3.1 та 3.2.

```
appcompat            : "androidx.appcompat:appcompat:${versions.appcompat}",
junit                : "junit:junit:${versions.junit}",
mockito              : "org.mockito:mockito-core:${versions.mockito}",
testrunner           : "androidx.test:runner:${versions.testrunner}",
espressocore         : "androidx.test.espresso:espresso-core:${versions.espressocore}",
atsl                 : "androidx.test:runner:${versions.atsl}",

nordicscanner        : "no.nordicsemi.android.support.v18:scanner:${versions.nordicscanner}",
nordicdfu            : "no.nordicsemi.android:dfu:${versions.nordicdfu}",
nordicble            : "no.nordicsemi.android:ble:${versions.nordicble}",
nordicblecommon      : "no.nordicsemi.android:ble-common:${versions.nordicble}",

lifecycleExtensions  : "androidx.lifecycle:lifecycle-extensions:${versions.lifecycle}",
lifecycleCommon      : "androidx.lifecycle:lifecycle-common-java8:${versions.lifecycle}",
liveData             : "androidx.lifecycle:lifecycle-livedata-ktx:${versions.lifecycle}",
viewModel            : "androidx.lifecycle:lifecycle-viewmodel-ktx:${versions.lifecycle}",

gson                 : "com.google.code.gson:gson:${versions.gson}",
timber               : "com.jakewharton.timber:timber:${versions.timber}",
androidxlegacysupport: "androidx.legacy:legacy-support-v4:${versions.androidxlegacysupport}",
awsandroidsdkcore    : "com.amazonaws:aws-android-sdk-core:${versions.awsandroid}",
cognitoidentity      : "com.amazonaws:aws-android-sdk-cognitoidentityprovider:${versions.awsandroid}",
localbroadcast       : "androidx.localbroadcastmanager:localbroadcastmanager:${versions.localbroadcast}",
playservices         : "com.google.android.gms:play-services-location:${versions.playservices}",

tensorflowlite       : "org.tensorflow:tensorflow-lite:${versions.tensorflowlite}",
tensorflowliteops    : "org.tensorflow:tensorflow-lite:${versions.tensorflowlite}",
tensorflowlitemeta   : "org.tensorflow:tensorflow-lite:${versions.tensorflowlitemeta}",

corektx              : "androidx.core:core-ktx:${versions.corektx}",
stdlibjdk            : "org.jetbrains.kotlin:kotlin-stdlib-jdk7:${versions.stdlibjdk}",
commonsio            : "org.apache.commons:commons-io:${versions.commonsio}",
kotlinasjava         : "org.jetbrains.dokka:kotlin-as-java-plugin:${versions.kotlinasjava}",
kotlingradleplugin   : "org.jetbrains.kotlin:kotlin-gradle-plugin:${versions.kotlingradleplugin}",
kotlingradle         : "com.android.tools.build:gradle:${versions.kotlingradle}",
coroutines           : "org.jetbrains.kotlinx:kotlinx-coroutines-android:${versions.coroutines}",
coroutinescore       : "org.jetbrains.kotlinx:kotlinx-coroutines-core:${versions.coroutines}",
```

Рисунок 3.1 – Перелік використаних бібліотек.

```
final versions : LinkedHashMap<String, String> = [
        appcompat             : '1.1.0',
        junit                 : '4.12',
        mockito               : '2.19.0',
        testrunner            : '1.2.0',
        espressocore          : '3.2.0',
        atsl                  : '1.2.0-beta01',
        nordicscanner         : '1.0.0',
        nordicdfu             : '1.9.0',
        nordicble             : '2.1.1',
        lifecycle             : '2.2.0',
        gson                  : '2.8.5',
        timber                : '4.7.1',
        androidxlegacysupport: '1.0.0',
        awsandroid            : '2.12.7',
        localbroadcast        : '1.0.0',
        playservices          : '17.0.0',
        tensorflowlite        : '2.3.0',
        tensorflowlitemeta    : '0.1.0-rc1',
        corektx               : '1.3.2',
        stdlibjdk             : '1.4.10',
        commonsio             : '1.3.2',
        kotlinasjava          : '1.4.20',
        kotlingradleplugin    : '1.4.10',
        kotlingradle          : '4.2.0',
        coroutines            : '1.3.9',
        dokkagradleplugin     : '1.4.20'
]
```

Рисунок 3.2 – Перелік версій для використаних бібліотек.

Таким чином кожен окремий модуль може імпортувати необхідну бібліотеку, уникаючи конфлікту версій та повторення коду. Приклад декларування бібліотек у окремому модулі програми зображено на рисунку 3.3.

```
testImplementation deps.junit
testImplementation deps.mockito
androidTestImplementation deps.testrunner
androidTestImplementation deps.espressocore

implementation deps.appcompat
implementation deps.androidxlegacysupport
implementation deps.localbroadcast
implementation deps.corektx
implementation deps.stdlibjdk

implementation deps.nordicble
implementation deps.nordicscanner
implementation deps.nordicdfu

implementation deps.tensorflowlite
implementation deps.tensorflowlitemeta

implementation deps.awsandroidsdkcore
implementation deps.cognitoidentity

implementation deps.commonsio
implementation deps.gson

implementation deps.coroutinescore
implementation deps.coroutines
```

Рисунок 3.3 – Імпортування сторонніх бібліотек в окремому модулі.

3.4 Розділення програмного коду на функціональні модулі.

Оскільки було обрано багатомодульний підхід для розробки програмного коду, то весь функціонал набору програмного забезпечення було розділено на окремі за призначенням модулі.

Виходячи з функцій й можливостей необхідних для досліджуваного проекту отримуємо наступні модулі:

- **bluetooth** – цей модуль відповідає за усі операції пов’язані з використанням технології Bluetooth Low Energy: сканування пристроїв, з’єднання/від’єднання, надсилання та зчитування даних;

- **database** – метою цього модулю є можливість взаємодії із внутрішньою пам’яттю смартфону та запису й зчитування даних;
- **dfu** – цей модуль відповідає за зчитування та оновлення версії програмного забезпечення на смарт пристроях;
- **machinelearning –** цей модуль містить алгоритми для аналізу даних та визначення ключових подій, таких як, наприклад: стрибки, сон, біг, підвищене серцебиття тощо;
- **core –** це основний модуль, в якому використовуються усі вище перелічені модулі. Саме завдяки цьому модулю подальший користувач буде працювати з набором програмного забезпечення. Core модуль надає зручний для використання інтерфейс для взаємодії з SDK.

Таким чином проект містить п’ять основних модулів, які можна використовувати незалежно один від одногою Однак, користувачу буде наданий доступ лише до core модулю, задля захищення приватності коду.

3.4.1 Аналіз модуля Bluetooth.

Даний модуль розділений умовно на дві частини: перша відповідає за сканування а друга за з’єднання та виконання операцій. Головні класи, які реалізують вище вказаний функціонал:

- BleScanner – цей клас дозволяє розпочати сканування доступних Bluetooth пристроїв з можливістю для користувача вказати діапазон та фільтр назв пристроїв;
- BLEService – це клас-сервіс, тобто усі операції в ньому виконуються асинхронно. Цей клас відповідає за виконання усіх Bluetooth операцій, оскільки створює та зберігає в собі екземпляр класу GattManager для кожного під’єднаного пристрою. Також дозволяє відстежувати стан зв’язку, його якість, відстань тощо;

- GattManager – цей клас містить реалізацію надсилання та зчитування даних з пристроїв, ввімкнення та вимкнення сповіщень даних. Для більшої ефективності та спрощення використовується BleManager з бібліотеки NordicSemi Ble;
- BleProvider - цей клас об’єднує функціонал усіх вище перелічених класів та дозволяє користувачу без ускладнень користуватись ним. Також BleProvider слідкує за станом блютуз з’єднання та вразі раптової втрати зв’язку з пристроєм спробує автоматично перепід’єднатись.

Таким чином, усі блютуз операції та функціонал необхідний для підтримки та відстеження з’єднання знаходяться у цьому модулі та можуть використовуватись незалежно.

Повний перелік класів можна побачити на рисунку 3.4.

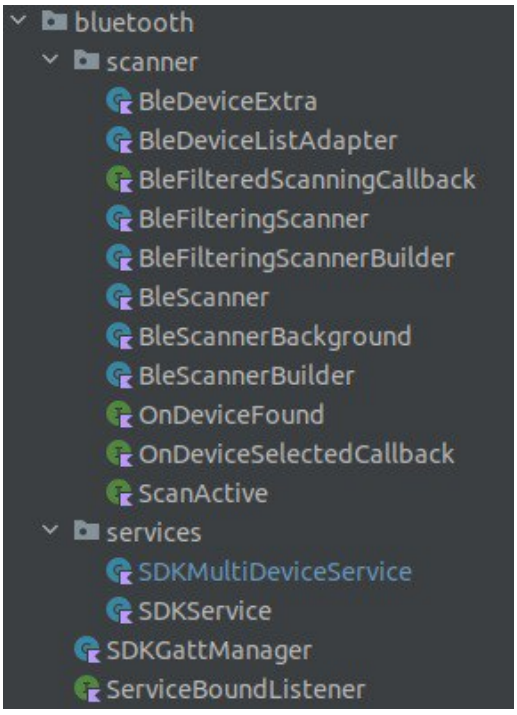


Рисунок 3.4 – Повний перелік класів Bluetooth модуля.

3.4.2 Аналіз модуля Database.

Даний модуль містить у собі класи для роботи з даними користувача. Головною метою є збереження та зчитування даних про користувача та змін у конфігурації набору програмного забезпечення.

Окрім реалізації зчитування та запису даних database модуль також містить класи, що створюють локальну базу даних та класи-моделі для зручного передавання даних. Основні класи:

- SdkConfigDatabase – цей клас дозволяє отримувати конфігураційні дані користувача з веб-сайту;
- UserDatabase – цей клас дозволяє отримувати дані користувача з хмари;
- UserAccountManager – цей клас дозволяє синхронізувати дані про профіль користувача з хмари, а також містить методи для авторизації користувача;
- UserDataManager – дозволяє записувати та зчитувати дані про використання набору програмного забезпечення користувачем. Зокрема зберігається інформація про робочі сесії, модель смарт пристрою, які використовувались, час використання тощо;
- LocalStorageHelper – цей клас містить у собі реалізацію запису та зчитування даних з локального сховища.

Підсумовуючи, даний модуль виконує всі операції пов’язані зі збереженням та оновленням даних. Дані періодично синхронізуються з хмарним сховищем після чого локальний екземпляр оновлюється.

Повний перелік класів зображено на рисунку 3.5.

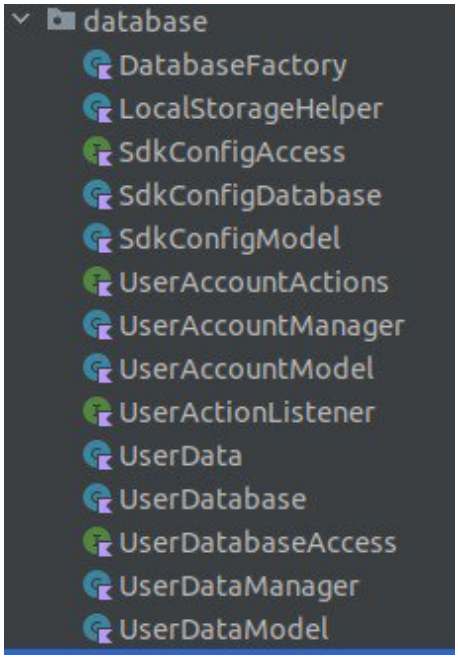


Рисунок 3.5 - Повний перелік класів Database модуля.

3.4.3 Аналіз модуля dfu (Device Firmware Update).

Метою цього модулю є перевірка та, за необхідності, оновлення програмного забезпечення смарт пристроїв. Усі версії ПО для смарт пристроїв знаходяться на приватному хмарному сховищу. Таким чином, користувач зможе оновити програмне забезпечення лише для офіційно придбаних пристроїв.

Процес сканування версії програмного забезпечення смарт пристрою відбувається автоматично при з’єднанні. При скануванні наявної версії програмного забезпечення на смарт пристрої звіряється ідентифікатор пристрою. Якщо ідентифікатор пристрою збігається з ідентифікатором пристрою, що належить користувачу набору програмного забезпечення, то перевіряється наявність нової версії ПО. В разі знайдення новішої версії – користувачу буде запропоновано оновити програмне забезпечення. Після завантаження оновлення SDK надсилає команду до смарт пристрою та оновлює ПО автоматично. Коли оновлення завершиться, пристрій автоматично перепід’єднується.

Головні класи, які відповідають за сканування, завантаження та оновлення ПО смарт пристрою:

- FirmwareUpdateManager – цей клас дозволяє за необхідності оновити програмне забезпечення смарт пристрою;
- FirmwareDownloadManager - надає функціонал для перевірки наявних оновлень ПО на хмарі та завантаження;
- FirmwareUpdateRequest - використовується для отримання версії наявного на смарт пристрої ПО;
- FirmwareUpdate – цей клас поєднує всі вище перелічені та надає зручний для використання інтерфейс.

Розгорнутий перелік класів зображено на рисунку 3.6.

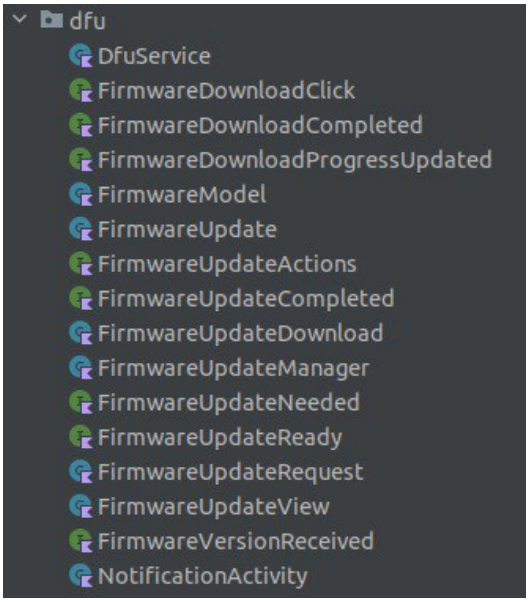


Рисунок 3.6 - Повний перелік класів DFU модуля

3.4.4 Аналіз модуля Machine learning.

Цей модуль призначений для постійної обробки даних з акселерометру, гіроскопу, кватерніону тощо. В залежності від призначення та типу смарт пристрою, тут визначається базовий формат використання пристрою на основі отриманих значень та специфічні події, наприклад стрибок, біг, сон тощо. Проаналізовані дані можуть використовуватись сучасними високоефективними алгоритмами для визначення більш специфічних подій.

Для аналізу даних використовуються класи моделі та метод кластеризації k-середніх (k-means). Дані акселерометру та гіроскопу розбиваються на кластери за методом k-середніх, що дозволяє швидше аналізувати та класифікувати отримані дані. Перелік класів machine learning модулю зображено на рисунку 3.7.

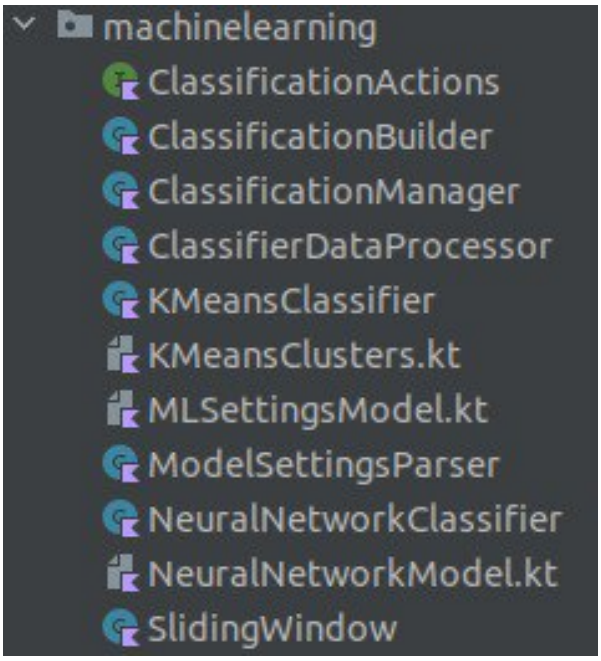


Рисунок 3.7 – Повний перелік класів Machine learning модуля.

3.4.5 Аналіз модуля Core.

Головною метою цього модуля є поєднання функціоналу інших модулів та надання зручного інтерфейсу для взаємодії з SDK. Даний модуль для більшої зручності було розділено на окремі пакети:

- Data – містить класи-моделі для зручного керування даними, інтерфейси зворотнього зв’язку (callback), класи bluetooth команд та результату їх виконання;
- Parsers – тут збережено усі класи, що використовуються для обробки масивів даних, отриманих зі смарт пристроїв. Обробка даних необхідна, оскільки смарт пристрої надсилають дані у форматі масиву байтів;
- Providers – в цьому пакеті знаходяться класи-провайдери, що надають до певного функціоналу, наприклад “UiProvider” дозволяє користувачу взаємодіяти з користувацьким інтерфейсом смарт пристрою: LED підсвітка, звукові сигнали тощо;
- Utils – цей пакет містить класи, що містять набір методів, які виконують загальні, часто повторно використовувані функції. Більшість класів утиліт визначають загальні методи в статичній області.

Основним класом даного модулю є SDKProvider. За допомогою цього класу користувач може авторизуватись та отримати доступ до необхідного функціоналу, отримавши необхідний клас-провайдер.

3.5 Розробка модуля Bluetooth.

Головна мета досліджуваного проекту – це взаємодія зі смарт пристроями з використанням протоколу Bluetooth з низьким енергоспоживанням. Набір програмного забезпечення повинен надавати можливість сканування та з’єднання

з іншими Bluetooth пристроями. Також підтримувати можливість одночасного підключення та взаємодії з кількома пристроями одночасно.

Обмін даними, використовуючи протокол Bluetooth Low Energy, потребує велику кількість обчислювальних ресурсів, тому доречно виконувати усі операції асинхронно. Для цього краще всього використовувати клас сервіс, що дозволить зберігати екземпляр класу SDKGattManager, який необхідний для надсилання команд до пристрою та зчитування даних, а також інформацію про стан з'єднання кожного пристрою. Для цих цілей було розроблено клас SDKService, який дозволяє під'єднатись одночасно до кількох смарт пристроїв та зручно взаємодіяти з ними, використовуючи інтерфейси зворотного зв'язку.

Для зручного формування та надсилання команд на смарт пристрої користувачем, було розроблено специфіковані класи команд. Таким чином користувачу достатньо оголосити зміну необхідної команди та вказати бажані параметри, не знаючи при цьому структури команди. SDKService приймає команди та обгортає передані параметри у байт код та надсилає до смарт пристрою, а після виконання команди сповіщає про це користувача.

Основні методи класу SDKService:

- connect() – цей метод надсилає запит на з'єднання з bluetooth пристроєм;
- disconnect() – цей метод надсилає запит на від'єднання від bluetooth пристрою;
- enableCharacteristicDataNotifications() – цей метод надсилає запит на ввімкнення сповіщень даних з певної характеристики пристрою. Таким чином можливо отримувати дані, що транслюються смарт пристроєм;
- disableCharacteristicDataNotifications() – цей метод надсилає запит на вимкнення сповіщень даних з певної характеристики пристрою;
- writeDataToBleCharacteristic() – цей метод надсилає запит на запис даних до смарт пристрою;

- readDataFromBleCharacteristic() – цей метод надсилає запит на зчитування даних зі смарт пристрою.

Оскільки дані отримані з пристроїв незрозумілі кінцевому користувачу, то при отримані обробляються за допомогою класів-парсерів. Таким чином користувач отримує дані з пристроїв у зрозумілому форматі та може використовувати їх для своїх подальших цілей.

Користувач також має вибір: чи використовувати автоматичне перепід'єднання до пристрою, в разі його неочікуваного відключення. За це відповідає окремий клас ReconnectionHelper, який отримує інформацію про стан підключення з сервісу, та в разі раптового зникнення зв'язку – автоматично буде надсилати запит на підключення, коли пристрій знову з'явиться в доступному діапазоні.

Стандартна Android бібліотека для роботи з Bluetooth є досить складною для використання та неоптимізована, тому в попередньому розділі для роботи з протоколом BLE було обрано сторонню бібліотеку NordicSemi BLE. Ця бібліотека значно спрощує роботу та дозволяє уникнути написання великої кількості коду, який необхідний для коректного використання функціоналу стандартної бібліотеки. Також бібліотека NordicSemi містить реалізацію для сканування пристроїв і визначення їх технічних характеристик, а саме: наявні сервіси, характеристики (це специфічні ідентифікатори, які відповідають за певну функціональну частину пристрою. Наприклад для отримання версії програмного забезпечення пристрою потрібно зробити запит до відповідної характеристики) та їх властивості.

3.6 Розробка модуля database.

Для повноцінної роботи досліджуваного набору програмного забезпечення користувачу необхідно авторизуватись. Це допомагає уникнути поширенню

проекту в інші руки неофіційним шляхом та дозволяє оптимізувати роботу SDK для кінцевого користувача. Це досягається шляхом обов'язкової авторизації. Набір програмного забезпечення отримує та синхронізує дані у вигляді json файлу й зберігає у локальному сховищі мобільного пристрою. Інформація про користувача містить ідентифікатор пристрою та перелік доступного для користувача функціоналу.

Для того, щоб отримати дані з серверу створюється запит на приватну кінцеву адресу, для якої необхідний ідентифікатор користувача. В разі успішного виконання запиту - буде надана інформація про користувача та токен доступу з певним строком дії. Після закінчення сроку дії токену доступу, набір програмного забезпечення при наступному використанні надсилається повторний запит після чого отримується новий токен та синхронізуються дані користувача.

Для надсилання запитів на сервер було використано архітектурний підхід REST, який значно спрощує роботу з сервером, підвищує надійність та дозволяє легко масштабувати програмний код в подальшому. Для надсилання та оформлення запитів не використовувались сторонні бібліотеки.

Хоч використання локального сховища для збереження даних користувача є опціональним, проте збереження токену доступу відбувається у будь-якому випадку. Для запису даних до внутрішньої пам'яті телефону також не використовуються стороні бібліотеки.

Головні класи для роботи з даним модулем – це UserAccountManager та UserDataManager.

Клас UserAccountManager використовується для авторизації та синхронізації даних користувача. Основні методи:

- signIn() – цей метод надсилає запит на авторизацію до бек-енду. В разі успішної авторизації отримується токен доступу;
- signOut() – цей метод виходить з профілю поточного користувача та видаляє закешовані логін та пароль;

- changePassword() – цей метод дозволяє надіслати запит на зміну паролю користувача;
- resetPassword() – цей метод використовується в разі, якщо користувач забув попередній пароль. Також автоматично надсилається повідомлення на пошту з кодом для верифікації операції;
- deleteUserAccount() – цей метод надсилає запит на видалення облікового запиту.

Клас UserDataManager використовується для збереження та оновлення конфігураційних даних користувача та інформації про використання. Основні методи:

- getUserData() – цей метод надсилає запит на зчитування даних користувача з бек-енду;
- sendUserData() – цей метод надсилає запит на запис даних користувача до бек-енду;
- sendUnathenticatedData() – цей метод на відміну від попереднього надсилає запит без використання ідентифікатору користувача, а замість цього використовується ідентифікатор пристрою.

Повний лістинг коду наведено в додатку В.

3.7 Розробка модуля dfu.

Усі придбані смарт пристрої отримують постійну підтримку у вигляді регулярних оновлень програмного забезпечення та зворотного зв’язку з командою інженерів.

При підключенні до пристрою автоматично звіряється наявна на пристрої версія програмного забезпечення та остання випущена версія на хмарному сховищі. Якщо на пристрої встановлена не актуальна версія, буде створено запит

до бек-енду з токеном поточного користувача та завантажено останню версію ПО. Отримана версія програмного забезпечення завантажується у форматі zip та тимчасово зберігається у локальному сховищі. Після завантаження отриманий пакет розпаковується та надсилається спеціальний запит на оновлення програмного забезпечення до смарт пристрою. По завершенні оновлення програмного забезпечення пристрій автоматично перезавантажиться та під'єднається.

Завантаження та оновлення програмного забезпечення виконуються у foreground сервісі. Сервіс основного плану (або з англ. foreground service) – це сервіс, що виконує операції асинхронно та відображає прогрес у рядку сповіщень Android пристрою. Таким чином користувач зможе бачити прогрес завантаження та встановлення оновлення.

Перевірка наявних оновлень, їх завантаження та оновлення програмного забезпечення реалізовані в класі FirmwareUpdate.

Метод checkFirmwareVersion() приймає параметром цільовий смарт пристрій(в разі, якщо під'єднано кілька пристроїв одночасно) та перевіряє останні версії ПО на хмарному сховищі. Надсилання запиту до серверу делегується класу FirmwareUpdateManager, а також передається інтерфейс зворотного зв'язку. Якщо було знайдено новішу версію програмного забезпечення, FirmwareUpdateManager повідомить про це шляхом виклику через переданий інтерфейс зворотного зв'язку.

Метод downloadFirmwareUpdate() приймає параметром версію ПО та делегує виконання класу FirmwareUpdateDownload. Прогрес завантаження передається через інтерфейс зворотного виклику(callback).

Метод updatgeFirmwareWithFilePath() приймає параметрами цільовий смарт пристрій та шлях, куди було завантажено програмне забезпечення. Процес оновлення делегується класу FirmwareUpdateManager, а прогрес оновлення передається через інтерфейс зворотного виклику.

Лістинг коду класів наведено в додатку Г.

3.8 Розробка модуля machine learning.

Для певних типів смарт пристроїв, таких як: фітнес-трекер, хокейна клюшка, нашийник для тварин тощо, можна задіяти додатковий аналіз отримуваних даних. Дані, що надходять з пристрою додатково фільтруються класом ClassificationManager з використанням кластеризації методом к-середніх.

Кластерний аналіз (англ. Data clustering) — задача розбиття заданої вибірки об'єктів (ситуацій) на підмножини, які називаються кластерами, так, щоб кожен кластер складався з схожих об'єктів, а об'єкти різних кластерів істотно відрізнялися. Завдання кластеризації відноситься до статистичної обробки.

Кластеризація методом k-середніх (англ. k-means clustering) – це впорядкування множини об'єктів в порівняно однорідні групи. Винайдений в 1950-х роках математиком Гуґо Штайнгаузом і майже одночасно Стюартом Ллойдом.

Мета методу — розділити n спостережень на k кластерів, так щоб кожне спостереження належало до кластера з найближчим до нього середнім значенням. Метод базується на мінімізації суми квадратів відстаней між кожним спостереженням та центром його кластера.

Дані отримані після кластеризації ClassificationManager-ом передаються далі до відповідної моделі tensorflow-lite (tflite). Модель після отримання даних обробляє їх та генерує прогноз, тобто визначає специфічні події (такі як наприклад: сон, біг, стрибки, швидкість та силу замаху тощо) на основі логіки моделі. Детальніше процес роботи моделі tflite зображено на рисунку 3.8. Де тензор(tensor) — це узагальнення векторів і матриць до потенційно вищих розмірностей. TensorFlow представляє тензори як n-вимірні масиви базових типів даних. Кожен елемент у тензорі має однаковий тип даних

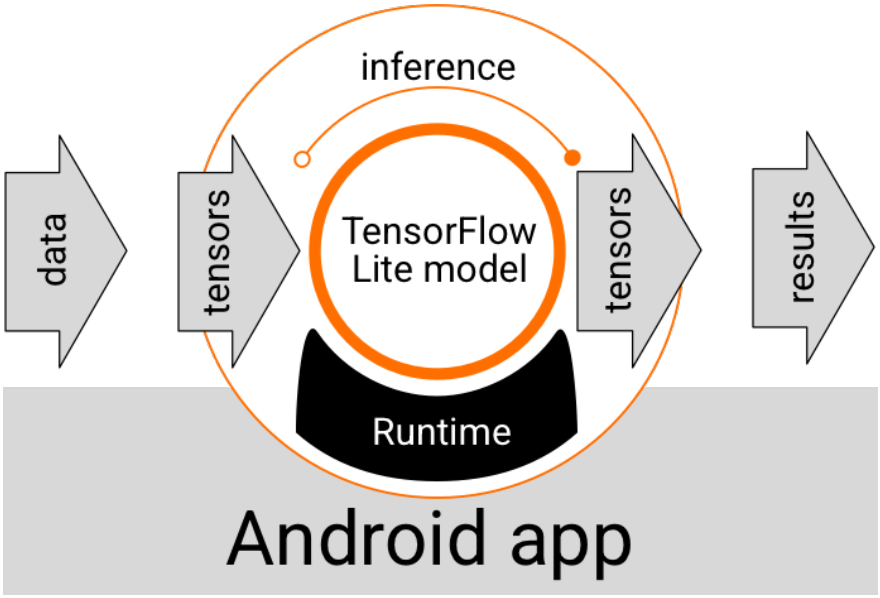


Рисунок 3.8 - Функціональний процес виконання аналізу моделлю tensorflow-lite.

Tensorflow-lite представляє собою набір утиліт, що має дві основні цілі. Перша з яких – зробити з нейромережі модель, прийнятну для мобільного пристрою. Зазвичай під цим мається на увазі зменшення розміру та складності мережі, що, в свою чергу, призводить до невеликого падіння точності роботи. Проте це необхідний компроміс між акуратністю роботи нейромережі та її розмірами на мобільному пристрої. Друга мета - створення середовища виконання для різних мобільних платформ, включаючи android та ios.

Важливою рисою tensorflow-lite є те, що за її допомогою неможливо тренувати модель. Нейромережа повинна бути спочатку навчена за допомогою tensorflow і далі конвертована у формат tensorflow-lite.

Набір програмного забезпечення одразу містить моделі для відповідних пристроїв користувача. В разі, якщо користувач придбав новий смарт пристрій, то моделі потрібно завантажити власноруч з особистого кабінету на веб-сайті, а потім імпортувати до SDK. Кожен тип смарт пристрою має відповідні унікальні моделі.

Клас NeuralNetworkClassifier відповідає за ініціалізацію tensorflow моделі та подальшу класифікацію даних. В ньому містяться два основних методи:

- initializeTensorflowModel() – цей метод приймає кілька параметрів, головним з яких є інтерпретатор(interpreter). Інтерпретатор – це функція, яка дозволяє налаштовувати конфігурацію моделі. Наприклад можна вказати кількість потоків, що будуть використовуватись при аналізі або вказати, чи слід використовувати апаратні прискорювачі Android пристрою тощо. Інтерпретатор використовується при ініціалізації моделі;
- loadModelFile() – цей метод приймає вхідним параметром назву необхідної моделі та завантажує модель з папки активів (assets);
- classifyData() – цей метод класифікує вхідні дані датчика зі смарт пристрою за допомогою структурованих вхідних даних. Основні вхідні параметри даного методу: масив вхідних даних, інтерфейс зворотного виклику для передачі та об’єкт класу NeuralNetworkModel. Цей клас використовується для специфікації та оптимізації роботи моделі. Усі необхідні параметри для ініціалізації об’єкту даного класу передаються разом із відповідним файлом моделі tflite. Приклад зображено на рис. 3.9.

```
  "hardware": {
    "dataRateInHz": 50,
    "accelerometerDataRange": [-16,16],
    "gyroscopeDataRange": [-2000,2000]
  },
  "slidingWindow": {
    "windowInMilliseconds": 800,
    "inputOrder": "AXES_BLOCK",
    "dimensions": 240,
    "featureCount": 6,
    "features": "ACC_GYR"
  },
  "classifiers" : {
    "pipeline": "WRLDS",
    "kMeansModels": [{
      "fileName": "fittness_actions.json",
      "normalizeData": true,
      "normalizationRange": [-1,1],
      "clusterCount": 120,
      "euclidianDistanceThreshold": 0,
      "featureCount": 3,
      "accelerometerFeatures": "NONE",
      "gyroscopeFeatures": "XYZ"
    }],
    "neuralNetworkModel": {
      "type": "TENSORFLOW_LITE",
      "normalizeData": false,
      "normalizationRange": [-1,1],
      "labels": [
        {
          "name": "boxing",
          "ID": 7
        },
        {
          "name": "squatting",
          "ID": 16
        },
        {
          "name": "ski_jumping",
          "ID": 19
        },
        {
          "name": "push_ups",
          "ID": 8
        },
        {
          "name": "jumping",
          "ID": 6
        },
        {
          "name": "running",
          "ID": 20
        },
```


Рисунок 3.9 – Приклад json файлу з налаштуваннями для tflite моделі.

На рисунку 3.10 зображено json файл, що використовується для налаштування роботи моделі та конфігурації попередньої обробки даних. Детальний опис параметрів наведено нижче:

- hardware. Сюди входять три параметри: dataRateInHz, accelerometerDataRange та gyroscopeDataRange. DataRateInHz використовується для визначення періодичності, з якою будуть проводитись підрахунки. accelerometerDataRange та gyroscopeDataRange використовуються для нормалізації даних та являють собою масив з двох чисел, де перше це мінімальне, а друге максимальне значення;
- gyroscopeDataRange kMeansModels. Використовується для ініціалізації класу KMeansClassifier що використовується для попередньої обробки даних та розбиття їх на кластери. Параметр fileName відповідає назві json файлу в якому збережені підготовлені та протестовані кластери для кожної спеціалізованої події. Таким чином, отримані зі смарт пристрою дані після обробки порівнюються з існуючими кластерами. Параметри accelerometerFeatures та gyroscopeFeatures відповідають за формат в якому необхідно обробити дані. Тобто, оскільки на рисунку gyroscopeFeautres дорівнює значенню XYZ, то це означає що дані з усіх осей гіроскопу будуть задіяні;
- neuralNetworkModel. Містить чотири основних параметри. Параметр type вказує тип нейронної моделі, яка буде використовуватись для аналізу. На разі завжди використовується лише модель типу tensorflow-lite. Параметр normalizeData вказує чи потрібно нормалізовувати отримані дані. Параметр labels містить ідентифікатори та назви подій, що підтримуються моделлю.

3.9 Розробка Core модуля.

Задля збереження захищеності коду, усі попередні модулі є внутрішніми (internal), тобто можуть викликатись лише в коді SDK. Користувачу буде надано доступ лише до базового (core) модуля, який поєднуватиме в собі всі інші. Також

це значно полегшить роботу з SDK для кінцевого користувача, оскільки буде надано зручний та зрозумілий інтерфейс для взаємодії.

Для зручності даний модуль було розділено на окремі пакети: data, listeners, parsers, providers та utils. На рисунку 3.11. зображено повний перелік пакетів та класів.

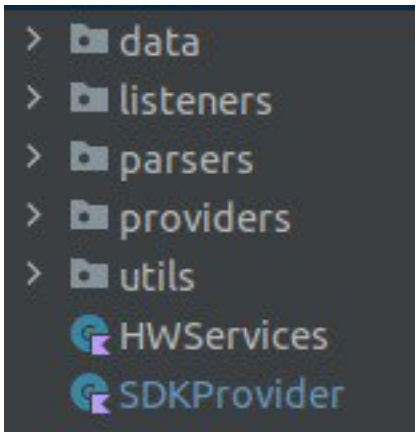


Рисунок 3.11 – Вміст Core модуля.

Клас SDKProvider – це головний клас для взаємодії розробника з SDK. Використовуючи цей клас, користувач може створювати екземпляри інших провайдерів та взаємодіяти з BLE пристроями.

3.9.1 Data.

Data пакет містить класи-моделі для команд, результатів отриманих із сенсорів смарт пристроїв, інтерфейси зворотного виклику(callback) та класи для додаткової обробки отриманих даних. На рисунку 3.12 зображено повний перелік класів.

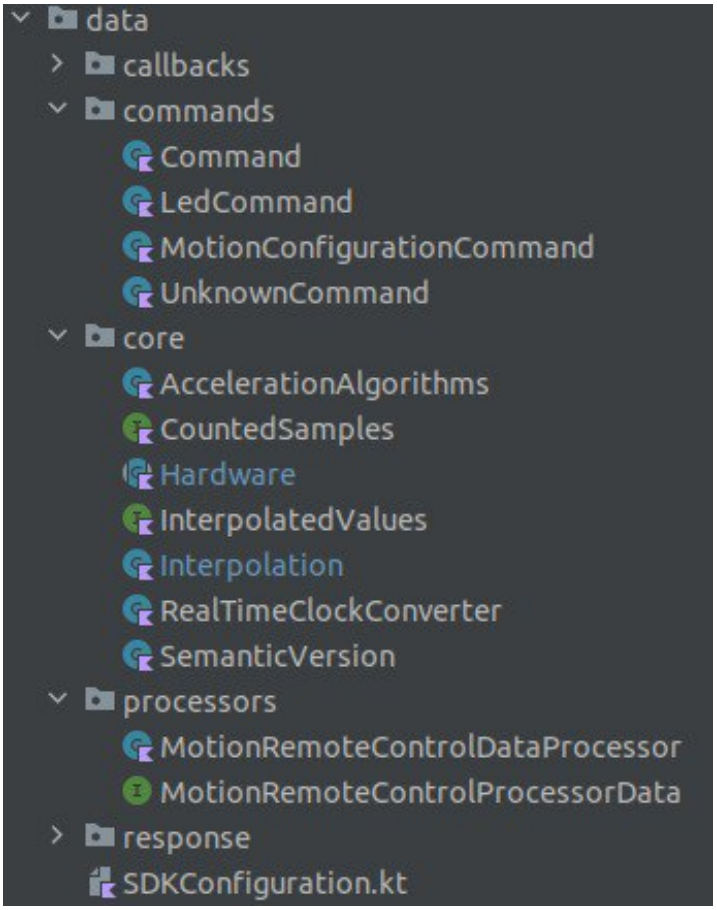


Рисунок 3.12 – Вміст data пакету.

Для зручності класи всередині data пакету було додатково розділено на дочірні пакети за їх призначенням.

Пакет callbacks. Містить усі інтерфейси зворотного виклику, що використовуються для повернення даних після виконання Bluetooth операцій.

Пакет commands. Містить класи моделі для команд, що надсилаються до смарт пристроїв. Детальний опис класів наведено нище:

- Command – базовий клас, що містить набір полів, необхідних для конфігурації сенсорів. Також є батьківським класом для інших класів команд;
- LedCommand – клас, що використовується для формування команд для LED модулів смарт пристроїв;

- MotionConfigurationCommand – клас, що використовується для формування команд, що спрямовані на сенсори акселерометру та гіроскопу;
- UnknownCommand – клас, що повертається в разі вказання некоректного набору параметрів при створені об’єкту Command.

Пакет core. Містить базові класи для додаткової обробки й перетворень отриманих значень зі смарт пристроїв. Детальний опис класів:

- AccelerationAlgorithms – цей клас містить методи з використанням математичних алгоритмів, в разі, якщо користувач хоче побудувати вектор на основі отриманих даних та відобразити його;
- Інтерфейси CountedSamples та InterpolatedValues – використовуються для повернення результатів роботи класу Interpolation;
- Interpolation – цей клас використовується у випадках, коли наявна велика кількість пропущених значень при трансляції даних з сенсору смарт пристрою;
- RealTimeConverter – цей клас використовується для конвертації часу у мілісекундах, отриманого з сенсору смарт пристрою, у стандартні часові формати;
- SemanticVersion – цей клас модель використовується для зручної і коректної репрезентації версії програмного забезпечення;
- Hardware – це абстрактний клас, що використовується для зручного ідентифікування типу сенсорів.

Пакет response. Містить класи моделі для зручного представлення даних, що були отримані після зчитування даних з сенсорів смарт пристроїв.

Клас SDKConfiguration використовується для зручнішого представлення конфігураційних даних та парсингу.

3.9.2 Listeners.

Цей пакет містить в собі інтерфейси слухачі (listeners), що повертають дані щоразу, коли відбувається певна подія. Досліджуваний набір програмного забезпечення також використовується й в Unity проектах, що сфокусовані на платформі Android. Тому усі інтерфейси продубльовано та змінено типи даних, що вони повертають, для підтримки Unity. Повний перелік усіх інтерфейсів зображено на рисунку 3.13.

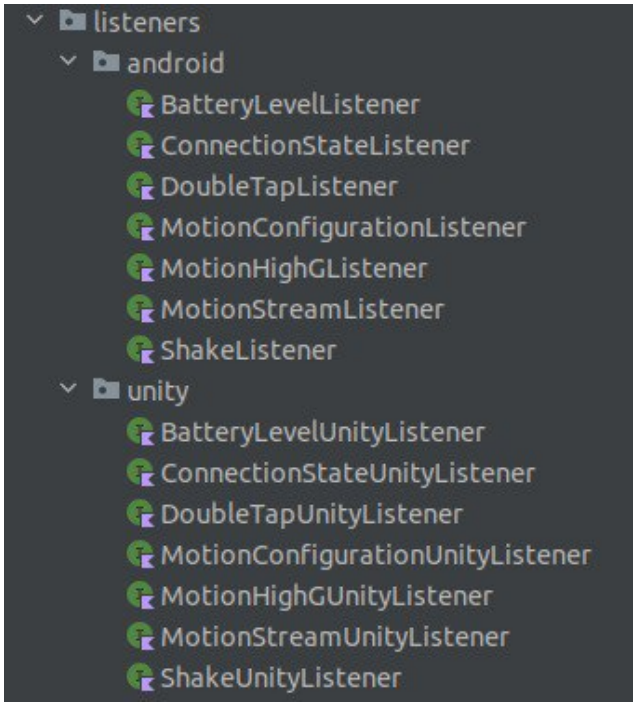


Рисунок 3.13 – Вміст пакету listeners.

3.9.3 Parsers.

Оскільки всі дані отримані із сенсорів смарт пристроїв передаються у форматі масивів байтів, використовуються класи-парсери. Ціллю яких є обробка байтових масивів та формування даних у відповідні класи-моделі. Для кожного

сенсору було розроблено окремі парсери. Це необхідно, оскільки формат та кількість байтів, які надсилає кожен із сенсорів – відрізняються. Перелік класів зображено на рисунку 3.14.

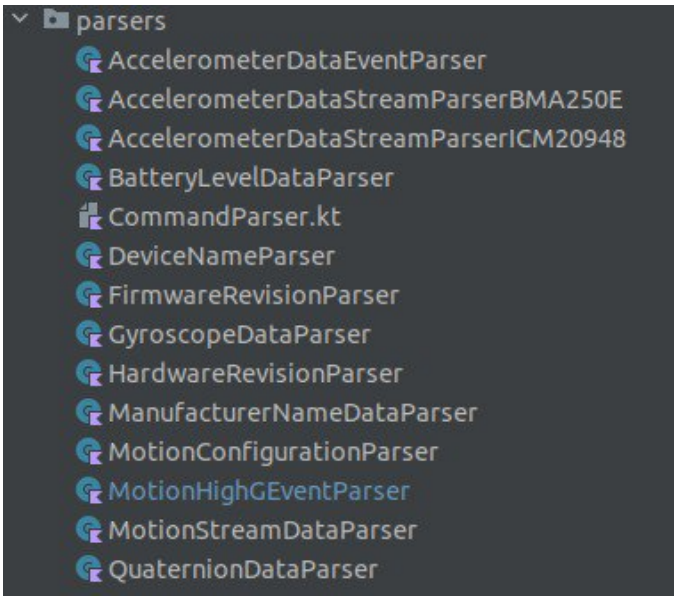


Рисунок 3.14 – Вміст пакету parsers.

3.9.4 Providers.

Для зручності та ефективності функціонал, що надається SDK, було розділено на окремі класи-провайдери. Таким чином, для кожного під’єднаного пристрою використовується окремий екземпляр певного провайдера. Це також дозволяє додатково інкапсулювати код та уникнути ситуацій, коли користувач, тобто інший розробник, за допомогою маніпуляцій в коді зміг би обійти обмеження по типу й ідентифікатору пристрою та використовувати весь функціонал.

Кожен клас провайдер надає доступ до базових Bluetooth операцій, однак, які специфікуються на різних типах та моделях пристроїв. Більш детальний опис провайдерів наведено нище:

- BleMultiDeviceProvider – цей клас містить надає функціонал для сканування, підключення та від’єднання BLE пристроїв та можливість відстеження стану підключення;
- CoreProvider – надає доступ до базових методів, які є однаковими для будь-якого типу й моделі смарт пристрою. До таких методів відносяться: getBatteryLevel(), read/writeDeviceName(), readManufacturerName(), readFirmwareVersion() та readRssiLevel();
- AiProvider – цей клас надає доступ до функціоналу модулю machine learning. Містить методи для початку трансляції даних з сенсорів смарт пристрою та їх аналізу;
- MotionProvider – цей клас містить методи read/writeData, enable/disableDataNotifications для сенсорів пов’язаних з рухом: акселерометр, гіроскоп та кватерніон;
- UiProvider - цей клас містить методи read/writeData, enable/disableDataNotifications для сенсорів пов’язаних з користувацьким інтерфейсом (з англ. User Interface), таких як: LED та динаміку.

Повний вміст пакету зображено на рисунку 3.15.

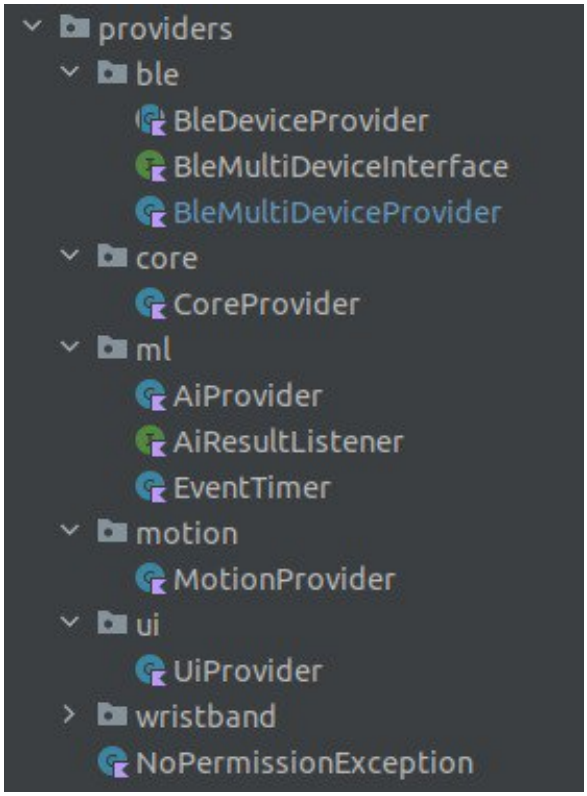


Рисунок 3.15 – Вміст пакету providers.

3.9.5 Utils.

В цьому пакеті зберігаються класи, що містять методи загального значення та використання. Так можна запобігти дублюванню коду та, оскільки всі методи в цих класах статичні, то немає потреби створювати екземпляри цих класів у коді. Головні класи:

- PermissionHelper – цей клас дозволяє зробити запит на усі необхідні для повноцінної роботи SDK дозволи, в разі, якщо якісь з них не були надані до цього;
- ReconnectionHelper – цей клас дозволяє відслідковувати прогрес перепідключення до неочікувано від’єднаних, надає останню інформацію про стан пристроїв. Уся інформація з цього класу виводиться у системні логи, що іноді може бути дуже корисним;

- UnityData – цей клас використовується для переведення даних з типів даних мови Kotlin до примітивів, оскільки Unity підтримує лише їх;
- WifiHelper – цей клас використовується для перевірки наявності інтернет-з’єднання через мережу wi-fi. Якщо смартфон підключено до мережі wi-fi та термін дії токену доступу користувача закінчився – буде автоматично надіслано запит на оновлення токену.

3.10 Висновок

На основі розділу 2 було розроблено набір програмного забезпечення для взаємодії зі смарт пристроями використовуючи технологію Bluetooth Low Energy. Під час розробки проекту, була використана бібліотека, мова програмування та архітектурний підхід, що були описані та вибір яких було обґрунтовані в попередніх розділах.

Для користувача був розроблений мінімалістичний та зрозумілий інтерфейс взаємодії з SDK. Для розробника був створений доступ лише до необхідного функціоналу того типу пристрою, що використовується.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Види тестування

Тестування - це процес виконання програми з метою пошуку помилок. Тестування програмного забезпечення поділяється на автоматизоване та мануальне (ручне).

Мануальне тестування - це примітивний підхід до тестування програмного забезпечення, що виконується тестувальником особисто без використання сторонніх інструментів. Тестувальник взаємодіє з програмним забезпеченням та перевіряє відповідність функціоналу та відсутність помилок в роботі програми.

Перевагами такого підходу тестування є:

- використовується для тестування елементів користувацького інтерфейсу (UI та UX);
- краще підходить для тестування невеликих проектів.

Недоліками мануального тестування є:

- велика затрата часу в залежності від розміру проекту;
- демонструє нижчу точність через більшу ймовірність людських помилок.

Автоматизоване тестування - це підхід тестування програмного забезпечення, що не потребує втручання людини, оскільки весь процес тестування виконується машиною за написаним заздалегідь тестовим сценарієм або тест кейсами.

Переваги автоматизованого тестування:

- висока ефективність при тестуванні великих проектів;
- можливість виконання тестів після будь-яких змін в коді без необхідності редагування тестів;

- швидкість тестування. Оскільки тести виконуються автоматизованими інструментами, весь процес тестування займає мало часу;
- ефективне при тестуванні оптимізації програмного забезпечення.

Недоліки автоматизованого тестування:

- потребує необхідних знань та навичок з боку тестувальника для написання тестів;
- підтримка та модифікація існуючих тестів потребує багато часу;
- низька ефективність при використанні в невеликих проектах або при тестуванні незначного функціоналу.

Для тестування розробленого набору програмного забезпечення було використано обидва підходи: мануальне й автоматизоване. Мануальне тестування використовувалось при тестуванні Bluetooth функціоналу: сканування, з'єднання та від'єднання, надсилання й зчитування даних. Ці процеси важко автоматизувати та участь людини необхідна для взаємодії зі смарт пристроями. Автоматизоване тестування використовувалось для перевірки оптимізації програмного забезпечення.

4.2 Процес тестування програмного забезпечення

Для тестування основного функціоналу розробленого набору програмного забезпечення використовувалось мануальне тестування. Для тестування було розроблено додаток, що використовує більшість функціоналу SDK. Основні тест кейси для перевірки роботи програми представлено у таблиці 4.1.

Автоматизоване тестування використовувалось для перевірки оптимізації програмного забезпечення. Для перевірки навантажень та моніторингу даних було використано інструмент Profiler в середовищі розробки Android Studio.

Таблиця 4.1 – Перелік сценаріїв для тестування розробленого програмного забезпечення.

Тестовий сценарій	Необхідні дії	Очікуваний результат
Перевірка роботи Bluetooth сканера.	Увімкнути два смарт пристрої та увімкнути Bluetooth на смартфоні. Почати сканування.	В списку сканування відобразились обидва смарт пристрої та відповідна інформація: назва й ідентифікатор пристрою
Перевірка встановлення з’єднання одночасно з кількома смарт пристроями.	Після успішного сканування вибрати кілька доступних смарт пристроїв та натиснути почергово на кілька пристроїв зі списку.	У відповідному порядку успішно встановлено з’єднання з кожним обраним пристроєм.
Перевірка автоматичного перепідключення до смарт пристрою.	Ввімкнути функцію автоматичного перепідключення у програмі та після успішного під’єднання вимкнути та увімкнути смарт пристрій.	Програма автоматично під’єдналась до втраченого пристрою після його ввімкнення.
Перевірка надсилання команд до смарт пристроїв.	Після успішного сканування й підключення до смарт пристрою, надіслати команду для перейменування	Команду було успішно надіслано та назва пристрою змінилась й відповідає переданому значенню.

	пристрою й передати тестове значення.	
Перевірка зчитування даних із смарт пристроїв.	Після успішного сканування й підключення до смарт пристрою, надіслати команду для отримання рівня заряду батареї.	Команду було успішно надіслано й отримано актуальне значення заряду батареї.
Перевірка правильності роботи програми з різними типами смарт пристроїв.	Після успішного сканування під'єднатись до прототипу хокейного сенсору й фітнес-трекеру, зчитати дані акселерометра з обох пристроїв.	Набір програмного забезпечення правильно визначив типи смарт пристроїв та зчитав й обробив дані акселерометра відповідно до специфіки кожного з пристроїв.
Перевірка роботи алгоритмів машинного навчання.	Під'єднатись до фітнес-трекеру, надіслати команду на отримання даних із сенсорів акселерометру та гіроскопу, увімкнути аналіз рухів одягнути смарт пристрій на руку. Зімітувати боксування, плескання в долоні, махання рукою.	Програма працює стабільно й без помилок. Результати аналізу даних алгоритмами машинного навчання сповістять про відповідні події: 1) Визначено вправу «боксування» 2) Визначено «плескання в долоні»

		3) Визначено махання рукою.
Перевірка навантаження оперативної пам’яті смартфону.	Під’єднатись до трьох смарт пристроїв, надіслати команди на отримання даних із сенсорів акселерометру та гіроскопу до кожного пристрою.	Програма працює стабільно. Дані надходять з кожного пристрою без затримок.
Перевірка оптимізації SDK.	Повторити кроки попереднього тест кейсу та після надсилання команд для отримання даних з сенсорів, після чого увімкнути аналіз рухів для двох смарт пристроїв.	Програма працює стабільно й без помилок. Дані з сенсорів надходять без затримок, аналіз рухів працює без збоїв.

При перевірці навантаження оперативної пам’яті смартфону програма працювала стабільно, однак, при детальному аналізі з використанням інструменту Profiler було помічено велике навантаження на пам’ять смартфону. Причиною навантаження слугували неоптимізовані формати даних та зберігання даних у змінних після кожного процесу обробки отриманих даних.. Для вирішення цієї проблеми, було оптимізовано типи даних, використовуючи лише примітиви та масиви замість списків, а також знищувались попередні отримані дані. Таким чином, навантаження на оперативну пам’ять пристрою не перевищує 256 Мбайт.

Під час перевірки оптимізації розробленого набору програмного забезпечення дані надходили із затримками в кілька секунд за рахунок великого

навантаження на CPU смартфону. Під час дослідження програмного коду було виявлено, що кластеризація даних виконується в одному ж потоці, що й обробка отриманих даних з сенсорів. Проблема була вирішена шляхом винесення процесу кластеризації даних в окремий потік для асинхронного виконання. Після внесених змін, зайнятість CPU мобільного пристрою в середньому сягає від 2% до 25%.

Приклад навантаження на ресурси смартфону при одночасному отриманні даних з трьох смарт пристроїв зображено на рисунку 4.1

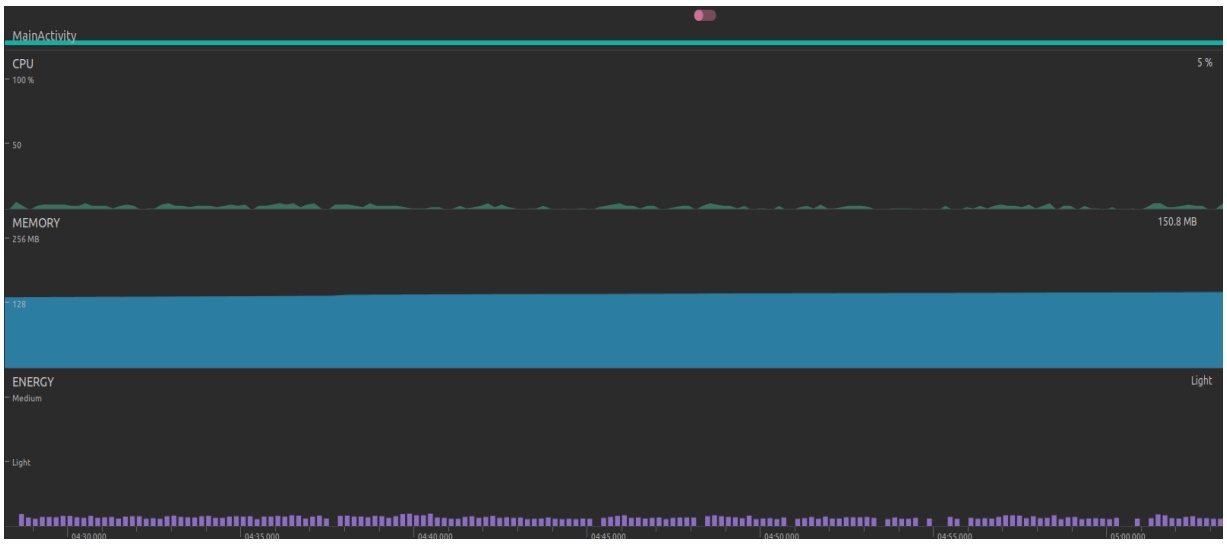


Рисунок 4.1 – Візуалізація навантаження на CPU, оперативну пам’ять та споживання батареї.

4.3 Демонстрація результатів роботи

Результатом даної роботи є готовий до використання набір програмного забезпечення для взаємодії зі смарт пристроями, використовуючи технологію Bluetooth Low Energy.

SDK містить увесь функціонал, описаний в попередньому розділі та повністю відповідає своєму призначенню. Для тестування було розроблено

додаток, що використовує більшість функціоналу SDK. На рисунку 4.2 зображено екран сканування Bluetooth пристроїв.

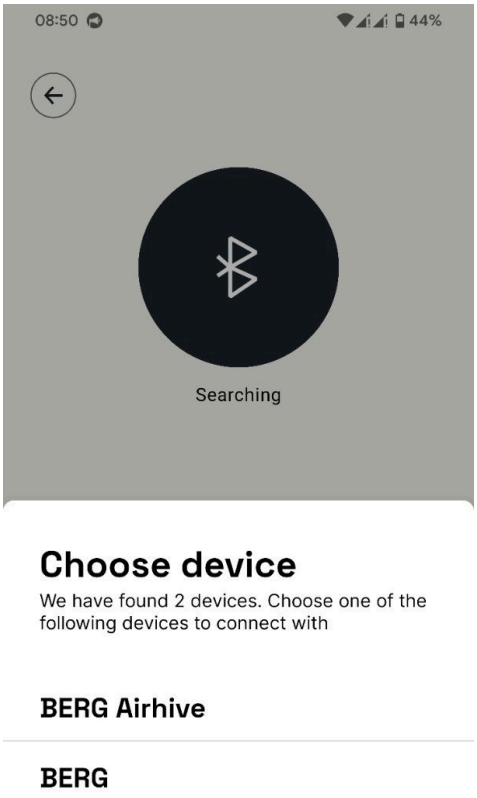


Рисунок 4.2 – Екран сканування доступних Bluetooth пристроїв.

Під’єднавшись до смарт пристрою відбувається перехід на головну сторінку додатку, де користувач може обрати формат тестування: постійна передача даних з пристрою або ж виявлення спеціальних рухів(відрізняються для кожного з пристроїв). Головний екран додатку зображено на рисунку 4.3.

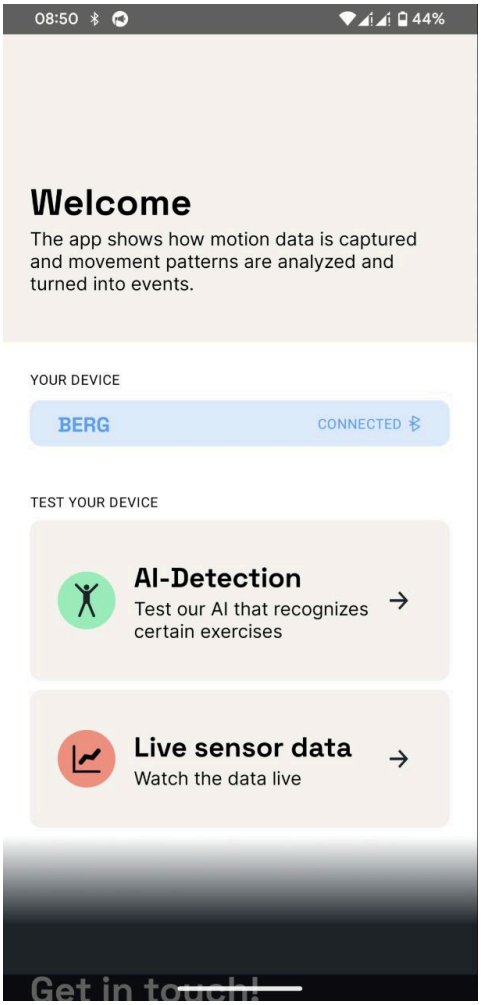


Рисунок 4.3 – Головна сторінка додатку.

В залежності від обраного формату тестування додаток або ж візуалізує отримані зі смарт пристрою дані акселерометру/гіроскопу у вигляді графіку, або ж перейде до екрану визначення спеціальних рухів. На рисунку 4.3 зображено візуалізацію даних з акселерометру та гіроскопу у вигляді графіку під час боксування. Візуалізація, визначеної на основі отриманих даних з сенсорів акселерометру та гіроскопу, вправи – боксування, зображено на рисунку 4.4.

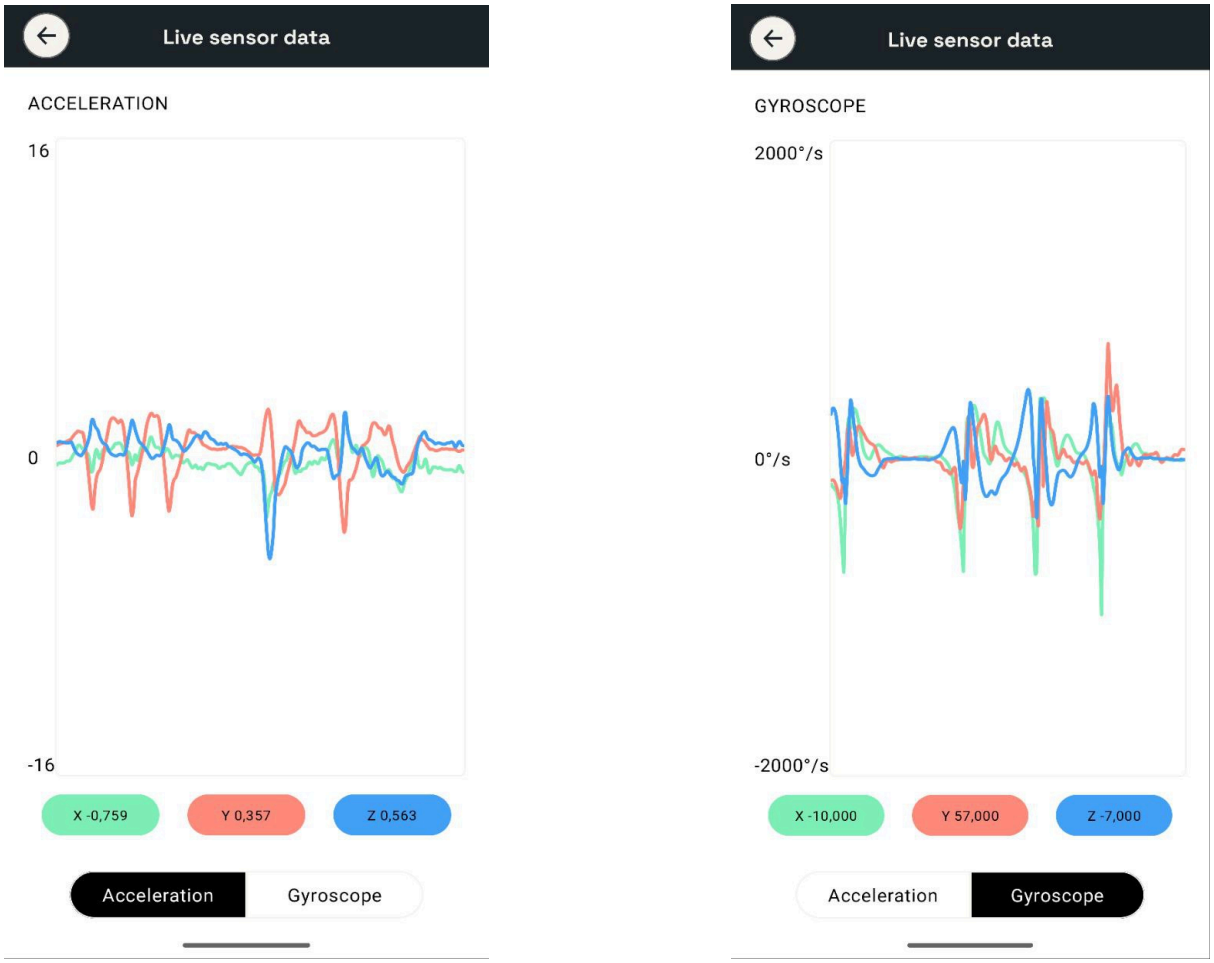


Рисунок 4.4 – Візуалізація даних з акселерометру та гіроскопу при боксуванні.

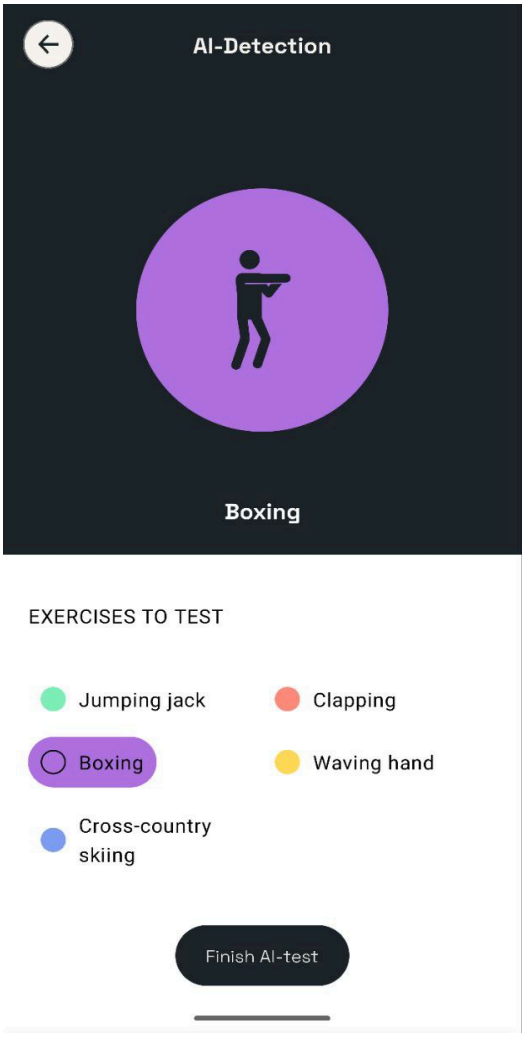


Рисунок 4.5 – Візуалізація визначеної вправи «боксування».

Реальні прототипи смарт пристроїв, які використовувались під час розробки й тестування зображено на рисунку 4.6. Один з пристроїв виконує роль фітнес-трекеру, а інший є тестовим прототипом сенсорів для тренування гри в хокей.

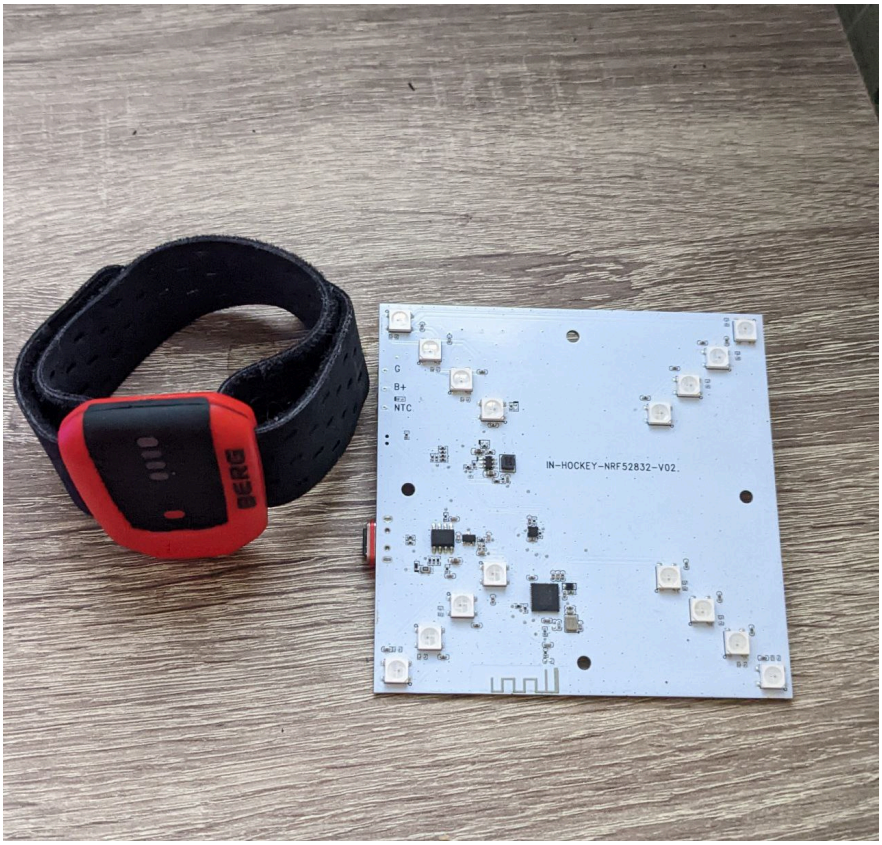


Рисунок 4.6 – Прототипи смарт пристроїв.

4.4 Висновок

Підсумовуючи, було проаналізовано види тестування: мануальне та автоматизоване, й, беручи до уваги переваги й недоліки кожного з підходів, сформовано та обґрунтовано процес тестування розробленого набору програмного забезпечення.

В ході тестування розробленого SDK всі описані тест кейси були виконані успішно й програма працювала без помилок. Використовуючи інструмент Profiler для виконання автоматизованого тестування оптимізації розробленого програмного забезпечення, були отримані середні значення навантаження на ресурси смартфону. Наведений вище рисунок 4.1 демонструє середні значення під час постійного отримання й обробки даних з сенсорів трьох смарт пристроїв:

- середнє значення навантаження на CPU дорівнює 5%, та максимально сягає 25%, якщо отримані з сенсорів дані проходять фільтрацію та надсилаються для подальшого аналізу з використанням нейронної моделі;
- середнє навантаження на пам’ять смартфону 150 Мбайт та максимально досягає 256 Мбайт, що є оптимальним значенням;
- енергоспоживання під час отримання й обробки даних з сенсорів під’єднаних смарт пристроїв знаходиться на низькому рівні.

5. ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Технологічний аудит результатів проведених досліджень технології BLE та особливостей її використання у SDK

Як було зазначено раніше, останнім часом у світі з’явилось багато так званих «розумних» пристроїв, які значно спрощують повсякденне життя людини. Серед них: розумні автомобілі, розумні термостати, дверні дзвінки, замки, холодильники, планшети, годинники, ремінці, окуляри та багато інших.

Розумний або смарт пристрій — це електронний пристрій, зазвичай підключений до інших пристроїв або мереж через різні бездротові протоколи, такі, наприклад, як Bluetooth, Zigbee, NFC, Wi-Fi, LiFi, 5G тощо. Однією з найважливіших частин сучасних розумних пристроїв – є додатки для взаємодії з ними. Саме тому виробники смарт пристроїв надають розробникам таких додатків так званий SDK (Software Development Kit) – набір засобів розробки, який дозволяє фахівцям з програмування створювати додатки для певного пакету програм.

Саме тому у виконаній нами магістерській кваліфікаційній роботі було проведено дослідження технології BLE та особливостей її взаємодії зі смарт пристроями і використання цієї технології у SDK.

В результаті виконаної роботи було запропоновано універсальний комплект програмного забезпечення (універсального та зручного SDK), який дозволяє працювати з різними пристроями і моделями, використовуючи базові функціональні модулі цих пристроїв, а також запропоновано формат SDK, який дозволяє взаємодіяти з різними пристроями, які відрізняються специфікою використання, виглядом або функціоналом. Тобто будь-який пристрій, що використовує за основу базові модулі (акселерометр, кватерніон, термометр, LED індикатори тощо, може вільно взаємодіяти з SDK.

Для встановлення комерційного потенціалу розробленого універсального пакету програмного забезпечення (універсального SDK) було запрошено 3-х

відомих експертів – кандидатів технічних наук, доцентів Кулика Я.А., Овчинникова К.В. та Барабан М.В.

Встановлення комерційного потенціалу розробленого універсального пакету програмного забезпечення здійснено за критеріями, наведеними в таблиці 5.1.

Таблиця 5.1 – Рекомендовані критерії оцінювання технічного рівня та комерційного потенціалу будь-якої розробки і їх бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
Ринкові перспективи					
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший	Термін реалізації ідеї більший	Термін реалізації ідеї	Термін реалізації ідеї менше 3-х років.	Термін реалізації ідеї менше 3-х років.

	за 10 років	за 5 років. Термін окупності інвестицій більше 10-ти років	від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін окупності інвестицій від 3-х до 5-ти років	Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Запрошені експерти оцінили розроблений нами універсальний пакет програмного забезпечення (універсального SDK) таким чином (див. табл. 5.2):

Таблиця 5.2 – Результати технологічного аудиту розробленого універсального пакету програмного забезпечення (універсального SDK) (за шкалою оцінювання 0-1-2-3-4)

Критерії	Прізвище, ініціали експертів		
	Кулик А.Я.	Овчинников К.В.	Барабан М.В.
	Бали, що їх виставили експерти:		
1	4	3	3
2	3	3	4
3	3	3	4
4	4	4	4
5	4	3	4
6	4	4	3
7	3	3	4
8	4	3	3
9	4	3	3
10	3	3	4
11	4	4	4
12	3	3	4
Сума балів	$СБ_1 = 43$	$СБ_2 = 39$	$СБ_3 = 44$

Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{43+39+44}{3} = \frac{126}{3} = 42,00$

Встановлення комерційного потенціалу розробленого нами універсального пакету програмного забезпечення (універсального SDK) будемо здійснювати на основі рекомендацій, наведених в таблиці 5.3 [1].

Таблиця 5.3 – Рівні комерційного потенціалу будь-якої наукової розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Оскільки середньоарифметична сума балів, що їх виставили експерти, складає 42,00 балів, то це свідчить, що розроблений універсальний пакет програмного забезпечення (універсального SDK) має рівень комерційного потенціалу, який вважається «високим».

Це пояснюється тим, що розроблений нами універсальний пакет програмного забезпечення (універсального SDK) є комплектом програмного забезпечення, який може взаємодіяти зі смарт пристроями та смарт картами різних типів, що будуть використовуватись для тестування.

5.2 Розрахунок витрат на розроблення універсального пакету програмного забезпечення (універсального SDK)

При розробленні універсального пакету програмного забезпечення (універсального SDK) були зроблені певні витрати. Зокрема:

А). Основна заробітна плата $З_о$ розробників, яка визначається за формулою:

$$З_о = \frac{М}{Т_р} \cdot t \text{ грн}, \quad (5.1)$$

де М – місячний посадовий оклад розробника, грн; приймемо, що

М = (6700…23000) грн/місяць;

$Т_р$ – число робочих днів в місяці; приймемо $Т_р$ = 20 день;

t – число днів роботи розробників.

Зроблені розрахунки зведемо до таблиці 5.4:

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на оплату праці, грн
1. Науковий керівник магістерської роботи	19500	975	20 годин	≈ 3250
2. Магістрант-студент-виконавець	2000 (беремо 6700)	335	86	≈ 28810
3. Консультант з економічної частини	17750	887,5	1,5 години	≈ 222 (при 6-годинному робочому дні)
Загалом				$З_о$ = 32282 грн

Б). Додаткова заробітна плата $З_д$ розробників розраховується як (10…12)% від величини їх основної заробітної плати, тобто:

$$З_д = \alpha \cdot З_о = (0{,}1…0{,}12) \cdot З_о. \quad (5.2)$$

Приймемо, що α = 0,105. Тоді для нашого випадку отримаємо:

$$З_д = 0{,}105 \times 32282 = 3389{,}61 \approx 3390 \text{ грн}.$$

В). Нарахування на заробітну плату $НЗП_{зп}$ розробників (дослідників) розраховуються за формулою:

$$НЗП_{зп} = (З_о + З_д) \cdot \frac{\beta}{100}, \quad (5.3)$$

де β – ставка обов'язкового єдиного внеску на державне соціальне страхування, %. β = 22%. Тоді:

$$НЗН_{зп} = (32282 + 3390) \times 0,22 = 7847,84 \approx 7848 \text{ грн.}$$

Г). Амортизація основних засобів А, які використовувались під час виконання цієї роботи:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12} \text{ грн,} \qquad (5.4)$$

де Ц – загальна балансова вартість основних засобів, грн;

$Н_а$ – річна норма амортизаційних відрахувань. Для нашого випадку можна прийняти, що $Н_а$ = (2,5...25)%;

Т – термін використання основних засобів, місяці.

Зроблені розрахунки зведено в таблицю 5.5.

Таблиця 5.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень тощо	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
1. Комп'ютерна техніка, обладнання тощо	44000	25	3,1 (при 80% використанні	2273,33 ≈ 2274
2. Приміщення університету, кафедри	16000	3,3	3,1 при 90% використанні	122,76 ≈ 123
Всього				А = 2397 грн

Д). Витрати на матеріали М розраховуються за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \text{ грн.,} \qquad (5.5)$$

де $Н_i$ – витрати матеріалу *i*-го найменування, кг; $Ц_i$ – вартість матеріалу *i*-го най-менування; $К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1...1,15); $В_i$ – маса відходів матеріалу *i*-го найменування; $Ц_в$ – ціна відходів матеріалу *i*-го найменування; n – кількість видів матеріалів.

Е). Витрати на комплектуючі К розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \text{ грн,} \qquad (5.6)$$

де $Н_i$ – кількість комплектуючих *і*-го виду, шт.; $Ц_i$ – ціна комплектуючих *і*-го виду; $К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); n – кількість видів комплектуючих.

Під час виконання роботи загальні витрати на матеріали та комплектуючі склали приблизно 1600 грн.

Ж). Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д}, \qquad (5.7)$$

де В – вартість 1 кВт-год. електроенергії, в 2022 р. В ≈ 3,0 грн/кВт;

П – установлена потужність обладнання, кВт; П = 1,35 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

Приймемо, що Ф = 222 годин;

$К_п$ – коефіцієнт використання потужності; $К_п$ < 1 = 0,8.

$К_д$ – коефіцієнт корисної дії, $К_д$ = 0,7.

Тоді витрати на силову електроенергію будуть дорівнювати:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д} = \frac{3 \cdot 1,35 \cdot 222 \cdot 0,8}{0,7} = 1027,54 \approx 1028 \text{ грн.}$$

И). Інші витрати $В_{інш}$ можна прийняти як (50…300)% від основної заробітної плати розробників, тобто:

$$В_{інш} = (0,5....3) \times З_о. \qquad (5.8)$$

Для нашого випадку отримаємо:

$$В_{інш} = 1,45 \times 32282 = 46808,90 \approx 46809 \text{ грн.}$$

К). Сума всіх попередніх статей витрат складає витрати на виконання нашої роботи (безпосередньо розробником-магістрантом) – В.

В = 32282+ 3390 + 7848+ 2397 + 1600+1028 + 46809 = 95354 грн.

Л). Загальні витрати на розроблення та можливе впровадження розробленого нами універсального пакету програмного забезпечення (універсального SDK) $В_{заг}$ розраховуються за формулою:

$$В_{заг} = \frac{В}{\beta}, \qquad (5.9)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання цієї роботи. Можна прийняти, що, $\beta \approx 0{,}85$ [1], оскільки робота потребує незначного доопрацювання.

Тоді: $В_{заг} = \frac{95354}{0{,}85} = 112181{,}17$ грн або приблизно 113 тисяч грн.

Тобто прогнозовані загальні витрати на розробку та можливе впровадження (комерціалізацію) розробленого нами універсального пакету програмного забезпечення (універсального SDK) становлять приблизно 113 тисяч грн.

5.3 Розрахунок економічного ефекту від можливої комерціалізації нашої розробки

Економічний ефект від впровадження та можливої комерціалізації розробленого універсального пакету програмного забезпечення (універсального SDK) пояснюється його значно кращими функціональними можливостями. Тому нашу розробку можна реалізовувати на ринку дещо дорожче, ніж аналогічні за функціями розробки. Так, якщо подібна за функціями розробка у 2022 році коштувала на ринку в середньому приблизно від $50 до $400 або від 2-х тисяч грн до 16-ти тисяч грн (для розрахунків приймемо середню ціну розробки в 8 тис. грн), то нашу розробку можна буде реалізовувати на ринку в середньому приблизно за 13 тисяч грн або на 5 тисячі грн дорожче.

Оскільки розроблене нами програмне забезпечення може використовуватися в багатьох сферах життєдіяльності людини: спорті,

розвагах тощо, то аналіз місткості ринку показує цього продукту показує, що на сьогодні в Україні кількість подібних користувачів нашої розробки може складати приблизно 500 осіб, і їх кількість буде стрімко зростати . Тому можна очікувати зростання попиту на нашу розробку принаймні протягом 3-х років після її впровадження.

Тобто, якщо наша розробка буде впроваджена з 1 січня 2024 року (оскільки потребує що доопрацювання), то її результати будуть виявлятися протягом 2024-го, 2025-го та 2026-го років.

Прогноз зростання попиту на нашу розробку складає по роках:

а) 2024 р. – приблизно +100 шт. до базового року;

б) 2025 р. – +200 шт. до базового року;

в) 2026 р. – +300 шт. до базового року.

Можливе збільшення чистого прибутку $\Delta П_i$, що його може отримати потенційний інвестор від комерціалізації, тобто виведення нашої розробки на ринок, становитиме:

$$\Delta П_i = \sum_{1}^{n} (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\upsilon}{100}), \qquad (5.10)$$

де $\Delta Ц_o$ – покращення основного якісного показника від впровадження результатів нашої розробки у цьому році. Для нашого випадку це є збільшення ціни реалізації нашої розробки $\Delta Ц_o$ = 13 - 8 = + 5 тисячі грн;

N – основний кількісний показник, який визначає обсяг діяльності у році до впровадження результатів розробки; N = 500 шт.;

ΔN – покращення основного кількісного показника від впровадження результатів розробки. Таке покращення становитиме по роках, відповідно: у 2024 році – + 100 шт., у 2025 році +200 шт, та у 2026 році + 300 шт. (відносно базового 2022 року);

$Ц_o$ – основний якісний показник (тобто ціна), який визначає обсяг діяльності у році після впровадження результатів розробки, грн; $Ц_o$= 10 тисяч грн;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки; для нашого випадку n = 3;

λ – коефіцієнт, який враховує сплату податку на додану вартість; $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати $\rho = (0{,}2...0{,}5)$; візьмемо $\rho = 0{,}5$;

υ – ставка податку на прибуток. У 2022-23 роках υ = 18%. У 2024 році також очікуємо на 18%.

Тоді можливе зростання чистого прибутку $\Delta П_1$ для потенційного інвестора протягом першого року від можливого впровадження нашої розробки (2024 р.) складе:

$$\Delta П_1 = [5 \cdot 500 + 13 \cdot 100] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) \approx 1298 \text{ тис. грн.}$$

Можливе зростання чистого прибутку $\Delta П_2$ для потенційного інвестора від можливого впровадження нашої розробки протягом другого (2025 р.) року складе:

$$\Delta П_2 = [5 \cdot 500 + 13 \cdot 200] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) \approx 1742 \text{ тис. грн.}$$

Можливе зростання чистого прибутку $\Delta П_3$ для потенційного інвестора від можливого впровадження нашої розробки протягом третього (2026 р.) року складе:

$$\Delta П_3 = [5 \cdot 500 + 13 \cdot 300] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) \approx 2187 \text{ тис. грн.}$$

Приведена вартість зростання всіх чистих прибутків від можливого впровадження нашої розробки становитиме:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої роботи, грн;

т – період часу, протягом якого виявляються результати впровадженої роботи, роки. Для нашого випадку т = 3 роки;

τ – ставка дисконтування. Приймемо τ = 0,10 (10%);

t – період часу від моменту початку розроблення автоматизованої системи обліку та аналізу споживання електроенергії в домогосподарствах до моменту отримання можливих чистих прибутків потенційним інвестором.

Тоді приведена вартість зростання всіх можливих чистих прибутків ПП, що їх може отримати потенційний інвестор від комерціалізації нашої розробки, складе:

$$ПП = \frac{1298}{(1+0,1)^2} + \frac{1742}{(1+0,1)^3} + \frac{2187}{(1+0,1)^4} \approx 1073 + 1309 + 1494 = 3876 \text{ тисяч грн.}$$

Теперішня вартість інвестицій PV, що повинні бути вкладені для реалізації нашої розробки: $PV = (1,0…5) \times В_{заг}$.

Для нашого випадку $PV = (1,0…5) \times 115 = 5 \times 113 = 565$ тисяч грн.

Розраховуємо абсолютний ефект від можливих вкладених інвестицій $Е_{абс}$.

$$Е_{абс} = ПП - PV, \qquad (5.12)$$

де ПП – приведена вартість збільшення всіх чистих прибутків для інвестора від можливого впровадження нашої розробки, грн;

PV – теперішня вартість інвестицій PV = 565 тисяч грн.

Абсолютний ефект від можливого впровадження нашої розробки (при прогнозованому ринку збуту) за три роки складе:

$$Е_{абс} = 3876 - 565 = 3311 \text{ тисяч грн.}$$

Оскільки $Е_{абс} > 0$, то комерціалізація нашої розробки може бути доцільною. Далі розрахуємо внутрішню дохідність $Е_в$ вкладених інвестицій:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1, \qquad (5.13)$$

де $Е_{абс}$ – абсолютний ефект вкладених інвестицій; $Е_{абс}$ = 3311 тис. грн;

PV –теперішня вартість початкових інвестицій PV = 565 тис. грн;

$Т_ж$ – життєвий цикл розробки, роки.

$T_ж$ = 4 років (2023-й, 2024-й, 2025-й, 2026-й роки)

Для нашого випадку отримаємо:

$$E_в = \sqrt[4]{1+\frac{3311}{565}} - 1 = \sqrt[4]{1+5,8602} - 1 = \sqrt[4]{6,8602} - 1 = 1,618 - 1 = 0,618 = 61,8\%.$$

Далі визначимо ту мінімальну дохідність, нижче за яку потенційному інвестору не вигідно буде займатися комерціалізацією нашої розробки.

Мінімальна дохідність або мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f, \qquad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,10...0,12);

f – показник, що характеризує ризикованість вкладень; f = (0,05...0,30).

Для нашого випадку отримаємо:

$$\tau_{мін} = 0,12 + 0,30 = 0,42 \text{ або } \tau_{мін} = 42\%.$$

Оскільки величина $E_в = 61,8\% > \tau_{мін} = 42\%$, то потенційний інвестор у принципі може бути зацікавлений у фінансуванні та комерціалізації нашої розробки.

Далі розраховуємо термін окупності коштів, вкладених у можливу комерціалізацію розробленого універсального пакету програмного забезпечення (універсального SDK).

Термін окупності $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в}. \qquad (5.15)$$

Для нашого випадку термін окупності $T_{ок}$ коштів становитиме:

$$T_{ок} = \frac{1}{0,618} = 1,62 \text{ років} < 3 \text{ років},$$

що свідчить про потенційну доцільність комерціалізації розробленого універсального пакету програмного забезпечення (універсального SDK) .

Далі проведено моделювання залежності величини внутрішньої дохідності вкладених потенційних інвестицій від рівня інфляції в країні. Як відомо, на наступні роки, на жаль, прогнозується високий рівень інфляції (в межах 30% і вище), що обумовлюється військовою агресію росії проти України.

Прийнявши рівень інфляції у 30% отримаємо:

$$ПП = \frac{1298}{(1+0,3)^2} + \frac{1742}{(1+0,3)^3} + \frac{2187}{(1+0,3)^4} \approx 768 + 793 + 766 = 2327 \text{ тисяч грн.}$$

Тоді абсолютний ефект від можливого впровадження нашої розробки за три роки складе:

$$Е_{абс} = 2327 – 565 = 1762 \text{ тисяч грн.}$$

Внутрішня дохідність $Е_в$ вкладених інвестицій становитиме:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1,$$

де $Е_{абс}$ – абсолютний ефект вкладених інвестицій; $Е_{абс}$ = 1762 тисяч грн;

PV –теперішня вартість початкових інвестицій PV = 565 тисяч грн.

Для нашого випадку отримаємо:

$$Е_в = \sqrt[4]{1 + \frac{1762}{565}} - 1 = \sqrt[4]{1 + 3,1186} - 1 = \sqrt[4]{4,1186} - 1 = 1,424 - 1 = 0,424 = 42,4\%.$$

Прийнявши рівень інфляції у 50% отримаємо:

$$ПП = \frac{1298}{(1+0,5)^2} + \frac{1742}{(1+0,5)^3} + \frac{2187}{(1+0,5)^4} \approx 577 + 516 + 432 = 1525 \text{ тисяч грн.}$$

Тоді абсолютний ефект від можливого впровадження нашої розробки за три роки складе:

$$Е_{абс} = 1525 – 565 = 960 \text{ тисяч грн.}$$

Внутрішня дохідність $Е_в$ вкладених інвестицій становитиме:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1,$$

де $Е_{абс}$ – абсолютний ефект вкладених інвестицій; $Е_{абс}$ = 960 тисяч грн;

PV –теперішня вартість початкових інвестицій PV = 565 тисяч грн.

Для нашого випадку отримаємо:

$$E_в = \sqrt[4]{1+\frac{960}{565}} - 1 = \sqrt[4]{1+1,6991} - 1 = \sqrt[4]{2,6991} - 1 = 1,28 - 1 = 0,28 = 28\%.$$

Зроблені розрахунки у вигляді графіків наведено на рис. 5.1.

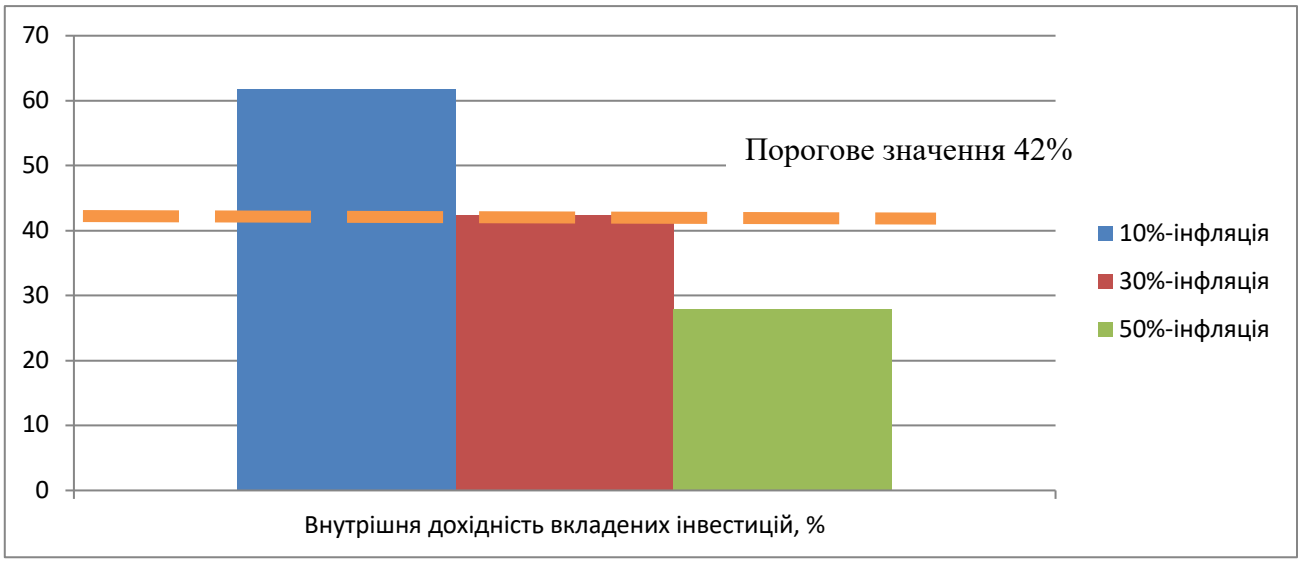


Рисунок 5.1 – Моделювання залежності величини внутрішньої дохідності

потенційних інвестицій від рівня інфляції в країні

Аналіз графіка на рис 5.1 показує, що при рівні інфляції в 10% величина внутрішньої дохідності інвестицій становить $E_в$ = 61,8%, що значно більше порогового значення $\tau_{мін} = 42\%$ і тому комерціалізація нашої розробки може бути доцільною. При рівні інфляції в 30% величина внутрішньої дохідності інвестицій, вкладених в комерціалізацію нашої розробки, становить $E_в$ = 42,4%, тобто практично дорівнює пороговому значенню $\tau_{мін} = 42\%$, і тому комерціалізація нашої розробки також може бути для інвестора доцільною. І тільки при інфляції у 50% величина внутрішньої дохідності інвестицій, вкладених в комерціалізацію нашої розробки, становить $E_в = 28,0\%$, що значно менше порогового значення $\tau_{мін} = 42\%$. і комерціалізація нашої розробки є проблематичною.

Результати виконаної економічної частини магістерської кваліфікаційної роботи зведено у таблицю:

Показники	Задані у ТЗ	Досягнуті у	Висновок

		магістерській кваліфікаційній роботі	
1. Витрати на розробку	Не більше 120 тис. грн	113 тис. грн.	Досягнуто
2. Абсолютний ефект від впровадження розробки (в майбутній вартості грошей), тисяч грн	Не менше 3000 тисяч грн (за три роки)	3311 тисяч грн	Практично виконано
3. Внутрішня дохідність інвестицій, %	не менше 42%	61,8%	Досягнуто
4. Термін окупності інвестицій, роки	до 3-ти років	1,62 років	Виконано

Таким чином, основні техніко-економічні показники розробленого нами універсального пакету програмного забезпечення (універсального SDK), визначені у технічному завданні, виконані.

ВИСНОВКИ

У даній магістерській кваліфікаційній дипломній роботі було розглянуто та проаналізовано переваги використання технології Bluetooth Low Energy для взаємодії з смарт пристроями. Розглянуто головні проблеми існуючих наборів програмного забезпечення для взаємодії зі смарт пристроями.

Було проаналізовано актуальність розробленого програмного забезпечення та проведено аналіз сучасного стану предметної області. На основі проведеного аналізу були визначені вимоги для розробки якісного набору програмного забезпечення: методології розробки, бібліотеки для роботи з протоколом BLE, мови програмування, архітектурний підхід. Було розглянуто головні проблеми при розробці універсального SDK та шляхи їх вирішення: досліджено специфіку роботи з технологією Bluetooth Low Energy, розглянуто сучасні бібліотеки для роботи з BLE та обрано найефективніший варіант, обґрунтовано вибір мови програмування та обрано оптимальний архітектурний підхід.

На основі знайдених рішень було розроблено власне програмне забезпечення для ефективної роботи з різними смарт пристроями. Описано функціонал розробленого набору програмного забезпечення, внутрішні зв’язки та загальну логіку. Розглянуто та детально описано процес синхронізації інформації користувача та процес обробки даних при використанні програмного забезпечення.

Після завершення розробки було проведено тестування для перевірки функціональних можливостей набору програмного забезпечення, ефективності та оптимізації. Також розроблено тестовий додаток для візуалізації отриманих даних зі смарт пристрою та продемонстровано прототипи смарт пристроїв. Проведено

економічні розрахунки, а результати роботи доповідались на науковій конференції [1].

Розроблений набір програмного забезпечення вказаним вимогам, потребам користувача та надає зручний для використання інтерфейс, який не залежить від специфіки, типу та сфери використання смарт пристрою. Набір програмного забезпечення є універсальним та дозволяє зекономити ресурси на розробку й подальшу підтримку персоналізованих ПЗ для окремих смарт пристроїв, оскільки надає необхідний функціонал для роботи з будь-яким виготовленим компанією пристроєм.

В подальшому до SDK буде додаватись новий функціонал та розширення підтримки більшого спектру смарт пристроїв.

ПЕРЕЛІК ПОСИЛАНЬ

1. Пакула А.А. Використання технології Bluetooth Low Energy для розумних пристроїв в мобільній розробці / А.А. Пакула., Є.А. Паламарчук // XV міжнародна науково-практична конференція “ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА АВТОМАТИЗАЦІЯ – 2022”(жовтень 2022) Одеського національного технічного університету, - Одеса: ОНТУ, 20-21 жовтня 2022 р.
2. Kotlin Language [Електронний ресурс]:[Веб-сайт] – Електронні дані. Режим доступу: https://kotlinlang.org/docs/home.html.
3. Global smart home market: dynamics, opportunities & challenges in 2022 [Електронний ресурс]:[Веб-сайт] – Електронні дані. Режим доступу: https://www.gfk.com/blog/global-smart-home-market-opportunities-challenges-in-2022-gfk-blog
4. Kotlin Coroutines Tutorial [Електронний ресурс]:[Веб-сайт] – Електронні дані. — Режим доступу: https://kotlinlang.org/docs/async-programming.html.
5. Smart Home Devices Global Market Report 2022 [Електронний ресурс]:[Веб-сайт] – Електронні дані. — Режим доступу: https://www.thebusinessresearchcompany.com/report/smart-home-devices-global-market-report
6. RxJava2 Course [Електронний ресурс]:[Веб-сайт] – Електронні дані. Режим доступу: https://startandroid.ru/ru/19-course/rxjava/435-urok-1.html.
7. NordicSemi Android BLE library [Електронний ресурс]:[Веб-сайт] – Електронні дані. Режим доступу: https://github.com/NordicSemiconductor/Android-BLE-Library
8. Modular app architecture [Електронний ресурс]:[Відео] – Електронні дані. Режим доступу: https://youtu.be/PZBg5DIzNww
9. Bluetooth Low Energy [Електронний ресурс]:[Веб-сайт] – Електронні дані. Режим доступу: https://developer.android.com/guide/topics/connectivity/bluetooth/ble-overview

10. Griffiths D. Head First Android Development: A Brain-Friendly Guide — 1st Edition / D. Griffiths. — O'Reilly Media, 2015 — 734 p.
11. Murphy M. The Busy Coder's Guide to Advanced Android Development M. Murphy. — CommonsWare, LLC, 2011 — 630 p.
12. Teacher Training and Professional Development: Concepts, Methodologies, Tools and Aplication / Management Association, Information Resources, P. 915.
13. iPhone SDK Development (The Pragmatic Programmers) 1st Edition – Pragmatic Bookshelf 2009 – 576p.
14. Simulation models of Bluetooth Low Energy BLE: Multidevice Environment — AV Akademikerverlag, 2019 — 76p.
15. The 2019-2024 World Outlook for Bluetooth Low Energy — ICON Group International, Inc 2018 — 272p.
16. Essential Scrum: A Practical Guide to the Most Popular Agile Process — Addison-Wesley Professional, 1st edition 2012 — 496p.
17. Introduction to Algorithms — Thomas H. Cormen, Charles E. Leiserson, 2nd Edition — The MIT Press 2009 — 1292p.
18. TensorFlow Lite for Android [Електронний ресурс]:[Веб-сайт] – Електронні дані. Режим доступу: https://www.tensorflow.org/lite/android
19. Programming Android with Kotlin 1s Edition — Pierre-Olivier Laurence, Amanda Hinchman-Dominguez — O'Reilly Media, 2021 — 356p.
20. Clean Architecture: A Craftsman's Guide to Software Structure and Design – Pearson, 1st edition 2017 – 432p.
21. Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People / A. Bharghava – Manning, 2016 – 256p.
22. Clean Code: A Handbook of Agile Software Craftsmanship – Pearson, 2008 – 464p.
23. Bluetooth specifications [Електронний ресурс]:[Веб-сайт] – Електронні дані. Режим доступу: https://www.bluetooth.com/specifications/specs/

24. Розробка програмного забезпечення – [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Розробка_програмного_забезпечення
25. Infostretch – [Електронний ресурс] – Режим доступу: https://www.infostretch.com/blog/a-primer-to-mobile-operating-systems/
26. Clarion Technologies – [Електронний ресурс] – Режим доступу: https://www.clariontech.com/blog/top-mobile-app-development- frameworks-in-2019
27. Офіційний сайт Google Play Market – [Електронний ресурс] – Режим доступу: https://play.google.com/store/apps/details?id=com.fivesysdev.mathy&hl= en_US&gl=US
28. Офіційний сайт Apple App Store – [Електронний ресурс] – Режим доступу: https://www.apple.com/ua/app-store/
29. Jeff McWherter. Professional Mobile Application Development / Scott Gowell, Jeff McWherter, 2012р. – 432 с.
30. Chris Adamson. iOS 10 SDK Development: Creating iPhone and iPad Apps with Swift / Chris Adamson, Janie Clayton, 2017р – 264 с.
31. Баженов В.А., Лізунов Інформатика. Комп"ютерна техніка. Комп"ютерні технології: підручник для студ. вищ. навч. закл. / В.А. Баженов та ін.; [наук. ред. : Г.А. Шинкаренко, О.В. Шишов] ; ЛНУ ім. І. Франка ; Київський нац. унт будівництва і архітектури ; Нац. техн. ун-т України "Київський політ. ін-т" - Київ: Каравела, 2011. - 592 с.
32. Murphy M. The Busy Coder's Guide to Advanced Android Development M. Murphy. — CommonsWare, LLC, 2011 — 630 p.
33. Darwin F. Android Cookbook: Problems and Solutions for Android Developers 2nd Edition / F. Darwin. — O'Reilly Media, 2017 — 772 p. Schildt H. Java: A Beginner's Guide, Eighth Edition, 8th Edition / H.Schildt. — McGraw-Hill Education, 2018 — 720 p.

34. Quickly Creating, Designing and Utilizing Mobile Apps for Your Business, 2nd Edition / J.McCalister. — CreateSpace Independent Publishing Platform, 2014 — 170 p.
35. Lee V. Mobile Applications: Architecture, Design, and Development / V.Lee. — Prentice Hall, 2004 — 368 p.
36. Iversen J. Learning Mobile App Development: A Hands-on Guide to Building Apps with iOS and Android, 1st Edition / J.Iversen, M.Eierman. — Addison-Wesley Professional, 2013 — 464 p.
37. The world of Android to create robust applications with Kotlin / M.Vasic. — Packt Publishing, 2017 — 378 p.
38. Leiva A. Kotlin for Android Developers: Learn Kotlin the easy way while developing an Android App, 1st Edition / A.Leiva. — CreateSpace Independent Publishing Platform, 2016 — 240 p.
39. KamathU.Deep Learning for NLP and Speech Recognition/ U.Kamath, W.Whitaker. — Springer, 2018 — 621 p.
40. Grenadiuk A. Mobile App Marketing And Monetization / A.Grenadiuk.— Semantic Valley LLC, 2014 — 151 p.
41. Berney P. Mobile Marketing: Lessons from Global Brand Leaders on How to Make a Success of the Mobile Channel / P.Berney. — Kogan Page, 2019 — 224 p.
42. Greene R. App Marketing: Top Mobile App Monetization and Promotion Strategies / R.Greene. — Tru Nobilis Pubilishing, 2017 —104p.
43. Google Play: number of available apps 2009-2019 [Електронний ресурс]: [Website] Точка доступу : https://www.statista.com/statistics/266210/number-of-available- applications-inthe-google-play-store.
44. Paid app availability [Електронний ресурс]: [Website] Точка доступу: https://support.google.com/googleplay/answer/143779.
45. Supportedlocations for developer & merchant registration [Електронний ресурс]: [Website] Точка доступу:

https://support.google.com/googleplay/androiddeveloper/answer/9306917?visit_id=637110980992615646-2850568539&rd=1.

46. How to use the Google Play Developer Console [Електронний ресурс]: [Website] Точка доступу: https://support.google.com/googleplay/android-developer/answer/6112435.
47. Kotlin and Android Development featuring Jetpack. Build Better, Safer Android Apps / M. Fazio. — Pragmatic Bookshelf, 2021 — 446p.
48. Kotlin Coroutines by Tutorials (Second Edition): Mastering Coroutines in Kotlin and Android / F. Babic, N.Srivastava. — Razeware LLC, 2019 — 352p.
49. Learning Algorithms: A Programmer's Guide to Writing Better Code. 1st Ed./ G. Heineman. — O'Reilly Media, 2016 — 484p.
50. Mobilized: An Insider's Guide to the Business and Future of Connected Technology / N. Eyal. — Berrett-Koehler Publishers, 2016 — 193p.

ДОДАТКИ

Додаток А

(обов’язковий)

ВНТУ

ЗАТВЕРДЖЕНО

Зав. Кафедри КСУ ВНТУ,

д.т.н., професор

_____Євген ПАЛАМАРЧУК

«___»_____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Засоби Software Development Kit для розумних пристроїв на основі технології Bluetooth Low Energy»

08-02.МКР.000.05.000 ТЗ

Студент групи 1АКІТ-21м

Антон ПАКУЛА

Підпис *Ім’я ПРІЗВИЩЕ*

Керівник к.т.н., професор кафедри АІІТ

Євген ПАЛАМАРЧУК

Підпис *Ім’я ПРІЗВИЩЕ*

Вінниця 2022

1. Назва та галузь застосування

Набір програмного забезпечення для взаємодії з смарт пристроями з використанням технології Bluetooth Low Energy. Розроблена система є універсальною та високоефективною, для обробки отриманих даних використовуються алгоритми, а для аналізу даних використовується машинне навчання на основі моделі нейронної мережі.

2. Підстава для розробки

Сфера розумних пристроїв постійно росте та продовжує залучати ще більшу кількість людей. Компаніям, що виробляють сенсори та прототипи смарт пристроїв для подальшого продажу необхідно надавати також й SDK для взаємодії з функціоналом. Частіше за все, для кожного прототипу й моделі сенсорів розробники таких компаній використовують розроблений раніше код та адаптують під специфіку нових пристроїв й сенсорів. У випадках, якщо компанія виготовляє різні за специфікою та призначенням сенсори й пристрої, такий підхід займає багато часу та створює проблеми з подальшим контролем версій й підтримкою розробленого програмного забезпечення.

Саме тому, виникає необхідність розробки універсального набору програмного забезпечення, що дозволяє уникнути витрати зайвих ресурсів, необхідних при розробці й підтримці програмного забезпечення для кожного окремого пристрою.

3. Мета та призначення розробки

Головною метою є побудова ідеології, алгоритму та програмно-апаратної реалізації системи, що дозволяє взаємодіяти з різними типами й моделями смарт пристроїв, ефективно обробляти й класифікувати отримані з сенсорів дані для подальшого аналізу з використанням машинного навчання та синхронізувати дані про використання з серверною частиною.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Пакула А.А. Використання технології Bluetooth Low Energy для розумних пристроїв в мобільній розробці / А.А. Пакула., Є.А. Паламарчук // XV міжнародна науково-практична конференція “ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА АВТОМАТИЗАЦІЯ – 2022”(жовтень 2022) Одеського національного технічного університету, - Одеса: ОНТУ, 20-21 жовтня 2022 р.
2. NordicSemi Android BLE library [Електронний ресурс]:[Веб-сайт] – Електронні дані. Режим доступу: https://github.com/NordicSemiconductor/Android-BLE-Library
3. The world of Android to create robust applications with Kotlin / M.Vasic. — Packt Publishing, 2017 — 378 p.
4. The 2019-2024 World Outlook for Bluetooth Low Energy — ICON Group International, Inc 2018 — 272p.
5. Simulation models of Bluetooth Low Energy BLE: Multidevice Environment — AV Akademikerverlag, 2019 — 76p.

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- сканування пристроїв;

- підтримка одночасного Bluetooth з’єднання з кількома пристроями;

- можливість запису та зчитування даних;

- синхронізація даних з сервером;

- обробка й аналіз даних.

5.2. Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:

- Android API 21+;

- Bluetooth;

- Android Studio.

5.2.2. Умови експлуатації системи:

- робота зі смарт пристроями;

- обробка даних, класифікація подій.

6. Стадії та етапи розробки.

6.1 Розділ 1 «Аналіз предметної області» має бути виконаний до 04.10.2022.

6.2 Розділ 2 «Вибір технологій для розробки набору програмного забезпечення» має бути виконаний до 23.10.2022.

6.3 Розділ 3 «Розробка програмного» має бути виконаний до 14.11.2022.

6.4 Розділ 4 «Тестування розробленого набору програмного забезпечення» має бути виконаний до 20.11.2022.

6.5 Розділ 5 «Економічний розділ» має бути виконаний до 02.12.2022.

6.5 Оформлення пояснювальної записки має бути виконаний до 12.12.2022.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «28»__11_2022 р.

7.2. Атестація МКР здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «16»__12_ 2022 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист магістерської кваліфікаційної роботи провести до «23»__12_ 2022р.

Додаток Б

(Лістинг програми)

```
class UserAccountManager(context: Context) : UserAccountActions {
    private val TAG = UserAccountManager::class.java.simpleName

    private var applicationContext: Context? = null
    private var userAccountActions: UserAccountActions? = null
    private var userDataModel: UserAccountModel? = null
    private var userDataManager: UserDataManager? = null

    private val userName: String? = null
    private var sessionPassword: String? = null
    private var resetPasswordContinuation: ForgotPasswordContinuation? = null

    private var USER_POOL_ID: String? = ""
    private var CLIENT_ID: String? = ""
    private var CLIENT_SECRET: String? = ""
    private var IDENTITY_POOL_ID: String? = ""

    private val REGION = Regions.fromName("us-east-1")

    private var userPool: CognitoUserPool? = null
    private var userAttributes: CognitoUserAttributes? = null
    private val userConfirmed = false
    private var session: CognitoUserSession? = null
    private val cognitoUserCodeDeliveryDetails: CognitoUserCodeDeliveryDetails? = null

    // This will cause confirmation to fail if the user attribute has been verified for another user in the same pool
    var forcedAliasCreation = false
```

```
//Continuations
private val forgotPasswordContinuation: ForgotPasswordContinuation? = null
private val newPasswordContinuation: NewPasswordContinuation? = null

var userActionListener: UserActionListener? = null

@JvmName("setUserActionListener1")
fun setUserActionListener(listener: UserActionListener?) {
    userActionListener = listener
}

init {
    applicationContext = context
    userDataModel = UserAccountModel(context)
    userDataManager = UserDataManager(context, this)
}

constructor(context: Context, userAccountActions: UserAccountActions) : this(context) {
    this.userAccountActions = userAccountActions
}

fun setRemoteConfig(configuration: SdkConfigModel) {
    USER_POOL_ID = configuration.getUserPoolID()
    CLIENT_ID = configuration.getClientID()
    CLIENT_SECRET = configuration.getClientSecret()
    IDENTITY_POOL_ID = configuration.getIdentityPoolID()
    initializeUserPool()
}

private fun initializeUserPool() {
```

```
        userPool =
            CognitoUserPool(applicationContext, USER_POOL_ID, CLIENT_ID, CLIENT_SECRET,
REGION)
        userPool!!.setPersistenceEnabled(true)
        userAttributes = CognitoUserAttributes()
        Log.i("userPool", userPool!!.user.toString())
        // Check if user has given permission for data gathering upon initialization
        userPermission()
    }

    private fun addAttribute(key: String, value: String) {
        userAttributes!!.addAttribute(key, value)
    }

    /**
     * Function that sends a request for user signup using the AWS Cognito SDK
     * If mail or password input is empty an error message will be returned through
onGenericReceivedListener
     *
     * @param password the password with which the user has signed up.
     * @param mail     the mail with which the user has signed up.
     */
    fun signUp(mail: String, password: String) {
        if (mail.isEmpty() || password.isEmpty()) {
            if (userActionListener != null) userActionListener!!.onUserAction(
                "signUp", false,
                "Mail or Password is invalid. Mail: $mail Password: $password"
            )
            return
        }

        // Create a CognitoUserAttributes object and add user attributes
```

```
    addAttribute("email", mail)
    if (WifiHelper.connectedToNetwork(applicationContext)) {
        userPool!!.signUpInBackground(mail, password, userAttributes, null, signupCallback)
    } else {
        Log.e("Wifi", "No wifi connection")
    }
}

//AWS Signup Handler
private val signupCallback: SignUpHandler = object : SignUpHandler {
    override fun onSuccess(
        cognitoUser: CognitoUser,
        userConfirmed: Boolean,
        cognitoUserCodeDeliveryDetails: CognitoUserCodeDeliveryDetails
    ) {
        // Sign-up was successful
        // cognitoUserCodeDeliveryDetails will indicate where the confirmation code was sent
        val message = "Verification mail sent to: " + cognitoUserCodeDeliveryDetails.destination
        Toast.makeText(applicationContext, message, Toast.LENGTH_LONG).show()
        if (!userConfirmed) {
            if (userActionListener != null) userActionListener!!.onUserAction(
                "signUp",
                true,
                message
            )
        } else {
            // The user has already been confirmed
            Log.i("User already confirmed", "")
        }
    }
```

```
    override fun onFailure(exception: Exception) {
        // Sign-up failed, check exception for the cause
        Log.e("Failed to sign up user", "exception: $exception")
        if (userActionListener != null) userActionListener!!.onUserAction(
            "signUp",
            false,
            exception.toString()
        )
    }
}

/**
 * Function to resend the confirmation link to verify that the user has signed up for a user account.
 * Resending the confirmation link can be used in scenarios in which the original link has expired.
 *
 * @param mail The mail of the user who needs another confirmation link
 */
fun resendConfirmationLink(mail: String) {
    val user = userPool!!.getUser(mail)
    if (WifiHelper.connectedToNetwork(applicationContext)) {
        user.resendConfirmationCodeInBackground(resendConfirmationLinkHandler)
    }
}

private val resendConfirmationLinkHandler: VerificationHandler = object : VerificationHandler {
    override fun onSuccess(verificationCodeDeliveryMedium: CognitoUserCodeDeliveryDetails) {
        val message =
            "Another verification mail sent to: " + verificationCodeDeliveryMedium.destination
        Toast.makeText(applicationContext, message, Toast.LENGTH_LONG).show()
        if (userActionListener != null) userActionListener!!.onUserAction(
            "resendConfirmationLink",
```

```
                true,
                message
            )
        }

        override fun onFailure(exception: Exception) {
            // Sign-up failed, check exception for the cause
            Log.e("Failed to sign up user", "exception: $exception")
            if (userActionListener != null) userActionListener!!.onUserAction(
                "resendConfirmationLink",
                false,
                exception.toString()
            )
        }
    }

    /**
     * Use userPermission function to verify that the user has signed up to Web and therefore has given permission to gather data
     */
    private fun userPermission() {
        val user: CognitoUser?
        val cachedUser = UserAccountModel.getCachedUser()
        if (cachedUser != null) {
            if (cachedUser == "unauthenticated") {
                if (userAccountActions != null) userAccountActions!!.onUnauthenticatedSession()
            } else {
                user = userPool!!.getUser(cachedUser)
                if (user != null) {
                    if (WifiHelper.connectedToNetwork(applicationContext)) {
                        user.getSessionInBackground(authenticationHandler)
                    } else {
```

```
                Log.e("Wifi", "No wifi connection")
            }
        } else {
            if (userActionListener != null) userActionListener!!.onUserAction(
                "signIn",
                false,
                "Tokens expired for user: " + UserAccountModel.getCachedUser()
            )
        }
    }
} else {
    Log.e("getCachedUser", "Could not get cached user or unauthenticated user")
}
}

/**
 * Function that sends a request for user signin using the AWS Cognito SDK
 * If mail or password input is empty an error message will be returned through
onGenericReceivedListener in the "signIn" action
 *
 * @param mail     the mail address of the user in Cognito user account management
 * @param password the password with which the user has signed up.
 */
fun signIn(mail: String, password: String) {
    if (mail.isEmpty() || password.isEmpty()) {
        if (userActionListener != null) userActionListener!!.onUserAction(
            "signIn", false,
            "Mail or Password is invalid. Mail: $mail Password: $password"
        )
    }
    if (session != null) {
        if (session!!.username == mail && session!!.isValid) {
```

```
            if (userActionListener != null) userActionListener!!.onUserAction(
                "signIn",
                false,
                session!!.username + " is already logged in"
            )
            return
        }
    }
    // Sign in the user
    sessionPassword = password
    val cognitoUser = userPool!!.getUser(mail)
    if (WifiHelper.connectedToNetwork(applicationContext)) {
        cognitoUser.getSessionInBackground(authenticationHandler)
    } else {
        Log.e("Wifi", "No wifi connection")
    }
}

// Callback handler for the sign-in process
private val authenticationHandler: AuthenticationHandler = object : AuthenticationHandler {
    override fun onSuccess(
        cognitoUserSession: CognitoUserSession,
        cognitoDevice: CognitoDevice
    ) {
        session = cognitoUserSession

        // Sign-in was successful, cognitoUserSession will contain tokens for the user
        val accessToken = cognitoUserSession.accessToken.jwtToken
        val IDToken = cognitoUserSession.idToken.jwtToken
        UserAccountModel.setCachedUser(applicationContext!!, cognitoUserSession.username)
        Log.i("currentUser", userPool!!.getUser(UserAccountModel.getCachedUser()).userId)
```

```
        // After we have set the new session we can add initial data to it and request previous sessions to be sent
        UserAccountModel.setTokens(accessToken, IDToken)
        if (userAccountActions != null) userAccountActions!!.onUserLogin(accessToken, IDToken)
        if (userActionListener != null) userActionListener!!.onUserAction(
            "signIn",
            true,
            "Signed in: " + UserAccountModel.getCachedUser()
        )
    }

    override fun getAuthenticationDetails(
        authenticationContinuation: AuthenticationContinuation,
        userNameContinuation: String
    ) {
        Log.e("AuthContinuation", userNameContinuation)
        if (sessionPassword != null) {
            // The API needs user sign-in credentials to continue
            val authenticationDetails =
                AuthenticationDetails(userNameContinuation, sessionPassword, null)

            // Pass the user sign-in credentials to the continuation
            authenticationContinuation.setAuthenticationDetails(authenticationDetails)

            // Allow the sign-in to continue
            authenticationContinuation.continueTask()
        } else {
            val message =
                "Cognito needs the password for user: $userNameContinuation to continue"
            if (userActionListener != null) userActionListener!!.onUserAction(
                "signIn",
```

```
                false,
                message
            )
            Log.e("Authentication", message)
        }
    }

    override fun getMFACode(multiFactorAuthenticationContinuation:
MultiFactorAuthenticationContinuation) {
        // Multi-factor authentication is required; get the verification code from user
        // multiFactorAuthenticationContinuation.setMfaCode(mfaVerificationCode);

        // Allow the sign-in process to continue
        // multiFactorAuthenticationContinuation.continueTask();
    }

    override fun authenticationChallenge(continuation: ChallengeContinuation) {
        // This challenge is invoked for MFA_SETUP Challenge
        val regMFAContinuation = continuation as RegisterMfaContinuation
    }

    override fun onFailure(exception: Exception) {
        // Sign-in failed, check exception for the cause
        Log.e("onFailure", "exception: $exception")
        if (userActionListener != null) userActionListener!!.onUserAction(
            "signIn",
            false,
            exception.toString()
        )
        if (exception.message!!.contains("UserNotFoundException")) {
            Log.e(
                "UserNotFoundException",
```

```
                "Sign in failed because user is not present in the Cognito user pool"
            )
        }
    }
}

fun requestNewTokens() {
    if (WifiHelper.connectedToNetwork(applicationContext)) {
        userPool!!.getUser(UserAccountModel.getCachedUser())
            .getSessionInBackground(authenticationHandler)
    }
}

/**
 * Checks if a user is logged into the AWS Cognito service
 * onGenericReceivedListener will communicate if the user is logged in or not through the action "loggedIn"
 */
fun checkLoginState(): Boolean {
    return if (session != null) {
        session!!.isValid
    } else false
}

fun getCurrentUser(): String {
    if (session != null) {
        if (session!!.isValid) {
            return session!!.username + " is logged in"
        }
    }
    val attributes = userAttributes!!.attributes
    return "There is no user logged in"
```

```
    }

    /**
     * Function that removes locally cached cognito credentials for the user who is signed in
     * If there is no user to sign out an error message will be returned through
onGenericReceivedListener
     */
    fun signOut() {
        if (userPool != null) {
            val user = userPool!!.getUser(UserAccountModel.getCachedUser())
            if (user != null) {
                user.signOut()
                if (userActionListener != null) userActionListener!!.onUserAction(
                    "signOut",
                    true,
                    user.userId + " is successfully signed out"
                )
                Log.i("Cognito", "User: " + user.userId + " is now logged out")
                UserAccountModel.removeTokens()
                session = null
            } else {
                if (userActionListener != null) userActionListener!!.onUserAction(
                    "signOut",
                    false,
                    "There is no user logged in"
                )
            }
        } else {
            if (userActionListener != null) userActionListener!!.onUserAction(
                "signOut",
                false,
                "Cannot sign out user"
```

```
            )
            Log.e("Signout", "Cannot sign out user")
        }
    }

    /**
     * Function to request a change in a user attribute
     *
     * @param attributeKey   The attribute to change. In this link is a list of standard attributes you can change for the user
     * https://docs.aws.amazon.com/cognito/latest/developerguide/user-pool-settings-attributes.html
     * @param attributeValue The value of the attribute to change
     *
     * preferred_username" is a cognito alias which has to be unique
     * For us it's more important to have a unique device_ID associated with a user
     * therefore "preferred_username" is repurposed to contain the user's device_ID
     */
    fun changeAttribute(attributeKey: String, attributeValue: String) {
        var key = attributeKey
        if (attributeKey == "link_ball") {
            key = "preferred_username"
        }
        val updateAttribute = CognitoUserAttributes()
        updateAttribute.addAttribute(key, attributeValue)
        if (userPool != null) {
            if (WifiHelper.connectedToNetwork(applicationContext)) {
                userPool!!.getUser(UserAccountModel.getCachedUser())
                    .updateAttributesInBackground(updateAttribute, updateAttributesHandler)
            } else {
                Log.e("Wifi", "No wifi connection")
            }
        }
```

```
}

private val updateAttributesHandler: UpdateAttributesHandler =
    object : UpdateAttributesHandler {
        override fun onSuccess(details: List<CognitoUserCodeDeliveryDetails>) {
            for (detail in details) {
                // Verify the attribute in case it's a change of mail.
                // Without verified contact information we can't be sure that we can reach our users
                if (detail.attributeName == "email") {
                    if (WifiHelper.connectedToNetwork(applicationContext)) {
                        userPool!!.getUser(UserAccountModel.getCachedUser())
                            .getAttributeVerificationCodeInBackground(
                                "email",
                                verifyAttributeHandler
                            )
                        return
                    } else {
                        Log.e("Wifi", "No wifi connection")
                    }
                }
            }

            // Attribute verification was successful!
            if (userActionListener != null) userActionListener!!.onUserAction(
                "changeAttributes",
                true,
                "Attribute is changed (unverified)"
            )
        }

        override fun onFailure(exception: Exception) {
```

```
            // Attribute verification failed, probe exception for details
            if (userActionListener != null) userActionListener!!.onUserAction(
                "changeAttributes",
                false,
                exception.toString()
            )
        }
    }

private val verifyAttributeHandler: VerificationHandler = object : VerificationHandler {
    override fun onSuccess(details: CognitoUserCodeDeliveryDetails) {
        // Attribute verification code was successfully sent!
        if (userActionListener != null) userActionListener!!.onUserAction(
            "changeAttributes",
            true,
            "Verification code has been sent to: " + details.destination
        )
    }

    override fun onFailure(exception: Exception) {
        // Attribute verification code request failed, probe exception for details
        if (userActionListener != null) userActionListener!!.onUserAction(
            "changeAttributes",
            false,
            exception.toString()
        )
    }
}

/**
 * Function to confirm the changes in an attribute that has to be verified (such as email)
```

```
     *
     * @param code          The code that was sent to the email linked to the user's account. If the mail is the attribute to verify, then the code is sent to the new mail that has been submitted.
     */
    fun confirmAttribute(code: String) {
        if (WifiHelper.connectedToNetwork(applicationContext)) {
            userPool!!.getUser(UserAccountModel.getCachedUser())
                .verifyAttributeInBackground("email", code, confirmAttributeHandler)
        } else {
            Log.e("Wifi", "No wifi connection")
        }
    }

    private val confirmAttributeHandler: GenericHandler = object : GenericHandler {
        override fun onSuccess() {
            // Attribute verification was successful!
            if (userActionListener != null) userActionListener!!.onUserAction(
                "confirmAttribute",
                true,
                "Attribute is verified"
            )
        }

        override fun onFailure(exception: Exception) {
            // Attribute verification failed, probe exception for details
            if (userActionListener != null) userActionListener!!.onUserAction(
                "confirmAttribute",
                false,
                exception.toString()
            )
        }
    }
```

```
    /**
     * Function to change the user's account password.
     * If the old or new password input is empty an error message will be returned through onGenericReceivedListener in the "changePassword" action
     *
     * @param oldPassword The old password value given by the user
     * @param newPassword The new password value given by the user
     */
    fun changePassword(oldPassword: String, newPassword: String) {
        if (oldPassword.isEmpty() || newPassword.isEmpty()) {
            if (userActionListener != null) userActionListener!!.onUserAction(
                "changePassword",
                false,
                "Old password or new password is invalid. Old password: " + oldPassword + "New password: " + newPassword
            )
            return
        }
        if (userPool != null) {
            if (WifiHelper.connectedToNetwork(applicationContext)) {
                userPool!!.getUser(UserAccountModel.getCachedUser())
                    .changePasswordInBackground(oldPassword, newPassword, changePasswordHandler)
            } else {
                Log.e("Wifi", "No wifi connection")
            }
        }
    }

    private val changePasswordHandler: GenericHandler = object : GenericHandler {
        override fun onSuccess() {
            // Password change was successful!
```

```
            if (userActionListener != null) userActionListener!!.onUserAction(
                "changePassword",
                true,
                "Password is succesfully changed"
            )
        }

        override fun onFailure(exception: Exception) {
            // Password change failed, probe exception for details
            Log.e("changePassword ", "failed: $exception")
            if (userActionListener != null) userActionListener!!.onUserAction(
                "changePassword",
                false,
                exception.toString()
            )
        }
    }



    /**
     * Function to reset the user's account password.
     * This function triggers a mail with a verification code to be sent to the user's verified mail account, which is submitted with the new password
     */
    fun requestNewPassword(mail: String?) {
        val user: CognitoUser?
        if (userPool != null) {
            //Check if the mail is provided, otherwise request new password for user that is currently logged in
            user = if (mail != null) userPool!!.getUser(mail) else {
                userPool!!.getUser(UserAccountModel.getCachedUser())
            }
```

```
        if (user != null) {
            if (WifiHelper.connectedToNetwork(applicationContext)) {
                user.forgotPasswordInBackground(forgotPasswordHandler)
            } else {
                Log.e("Wifi", "No wifi connection")
            }
        } else {
            if (userActionListener != null) userActionListener!!.onUserAction(
                "requestNewPassword", false,
                "Could not request new password for user: $mail . Please submit a valid mail to sent the password to"
            )
        }
    }

    private val forgotPasswordHandler: ForgotPasswordHandler = object : ForgotPasswordHandler {
        override fun getResetCode(continuation: ForgotPasswordContinuation) {
            // A code will be sent, use the "continuation" object to continue with the forgot password process
            val codeSentHere = continuation.parameters.destination
            if (userActionListener != null) userActionListener!!.onUserAction(
                "requestNewPassword", true,
                "Password reset code is sent to: $codeSentHere"
            )
            resetPasswordContinuation = continuation
        }

        override fun onSuccess() {
            // Forgot password process completed successfully, new password has been successfully set
            if (userActionListener != null) userActionListener!!.onUserAction(
                "resetPassword",
```

```
                true,
                "Password successfully reset for: " + resetPasswordContinuation!!.parameters.destination
            )
        }

        /**
         * This is called for all fatal errors encountered during the password reset process
         * @param exception: Forgot password processing failed, look at the exception for the cause
         */
        override fun onFailure(exception: Exception) {
            if (userActionListener != null) userActionListener!!.onUserAction(
                "resetPassword",
                false,
                exception.toString()
            )
            // Forgot password processing failed, probe the exception for cause
        }
    }

    /**
     * Function to reset the user's account password.
     * This function triggers a mail with a verification code to be sent to the user's verified mail account, which is submitted with the new password
     *
     * @param newPassword The user's new password
     * @param code        The code that has been sent to the mail that is linked to the user's account
     */
    fun resetPassword(newPassword: String, code: String) {
        if (resetPasswordContinuation != null) {
            resetPasswordContinuation!!.setPassword(newPassword)
            resetPasswordContinuation!!.setVerificationCode(code)
            if (WifiHelper.connectedToNetwork(applicationContext)) {
```

```
                resetPasswordContinuation!!.continueTask()
            } else {
                Log.e("Wifi", "No wifi connection")
            }
        } else {
            if (userActionListener != null) userActionListener!!.onUserAction(
                "resetPassword",
                false,
                "Could not find the Cognito continuation object for resetting the password"
            )
        }
    }

    /**
     * Function for deleting the user's account
     * The user has to be logged in to delete an account. If the userClaim does not match the logged in user then the account will not be deleted.
     *
     * @param userClaim The user who's account will be deleted
     */
    fun deleteUserAccount(userClaim: String?) {
        if (userClaim != null) {
            // To verify the user means to delete the account we double check if it's the same as the logged in user
            if (userClaim != UserAccountModel.getCachedUser()) if (userActionListener != null) userActionListener!!.onUserAction(
                "deleteUserAccount",
                false,
                "User claim to delete does not match logged in user"
            )
        } else {
            if (userActionListener != null) userActionListener!!.onUserAction(
```

```
            "deleteUserAccount",
            false,
            "No user found"
        )
    }
    val user = userPool!!.getUser(UserAccountModel.getCachedUser())
    if (user != null) {
        if (session!!.isValid) {
            if (WifiHelper.connectedToNetwork(applicationContext)) {
                user.deleteUserInBackground(deleteUserHandler)
                session = null
            } else {
                Log.e("Wifi", "No wifi connection")
            }
        } else {
            if (userActionListener != null) userActionListener!!.onUserAction(
                "deleteUserAccount",
                false,
                "User is not logged in"
            )
        }
    } else {
        if (userActionListener != null) userActionListener!!.onUserAction(
            "deleteUserAccount",
            false,
            "User does not exist or session tokens have expired"
        )
    }
}

private val deleteUserHandler: GenericHandler = object : GenericHandler {
```

```
        override fun onSuccess() {
            // Delete was successful!
            if (userActionListener != null) userActionListener!!.onUserAction(
                "deleteUserAccount",
                true,
                "User account has been deleted"
            )

            // Remove the locally cached user and session tokens after deleting the account
            val deletedUser = userPool!!.getUser(UserAccountModel.getCachedUser())
            deletedUser?.signOut()
            UserAccountModel.removeCachedUser(applicationContext!!)
            createNextSession(applicationContext!!.filesDir.toString() + "/data" + "/" +
UserAccountModel.getCachedUser())
            UserAccountModel.removeTokens()
        }

        override fun onFailure(exception: Exception) {
            // Delete failed, probe exception for details
            if (userActionListener != null) userActionListener!!.onUserAction(
                "deleteUserAccount",
                false,
                exception.toString()
            )
        }
    }

    fun onDestroy() {
        userPool = null
        userAttributes = null
        applicationContext = null
    }
```

```
    fun getIdentityPoolID(): String? {
        return IDENTITY_POOL_ID
    }


    override fun onUnauthenticatedSession() {
        userDataManager!!.setSessionData()
    }

    override fun onUserLogin(accessToken: String, idToken: String) {
        userDataManager!!.setSessionData()
    }

    override fun sdkConfigurationReceived(sdkConfigModel: SdkConfigModel) {
        setRemoteConfig(sdkConfigModel)
    }

    override fun requestNewSessionTokens() {}
}
```

Додаток В

(Лістинг програми)

```
class FirmwareUpdateManager(val context: Context, val bluetoothDevice: BluetoothDevice) :
    FirmwareVersionReceived,
    FirmwareDownloadClick,
    FirmwareDownloadProgressUpdated,
    FirmwareDownloadCompleted,
    DfuProgressListener {
    private val TAG = FirmwareUpdateManager::class.java.simpleName

    var firmwareUpdateDownload: FirmwareUpdateDownload? = null
    private var firmwareUpdateRequest: FirmwareUpdateRequest? = null
    private var firmwareUpdateView: FirmwareUpdateView? = null
    private var dfuServiceController: DfuServiceController? = null

    private var firmwareModel: FirmwareModel? = null
    private var fetchedFirmwareFilename: String? = null
    private var currentFirmwareFilename: String? = null
    private var firstDFURequest = true

    /** Listeners  */
    var firmwareUpdateNeeded: FirmwareUpdateNeeded? = null
    var firmwareUpdateReady: FirmwareUpdateReady? = null
    var firmwareUpdateCompleted: FirmwareUpdateCompleted? = null

    fun setFirmwareUpdateNeededListener(listener: FirmwareUpdateNeeded?) {
        firmwareUpdateNeeded = listener
    }

    fun setFirmwareUpdateReadyListener(listener: FirmwareUpdateReady?) {
```

```
        firmwareUpdateReady = listener
    }

    fun setFirmwareUpdateCompletedListener(listener: FirmwareUpdateCompleted?) {
        firmwareUpdateCompleted = listener
    }

    init {
        DfuServiceListenerHelper.registerProgressListener(context, this)
    }

    fun destroy() {
        DfuServiceListenerHelper.unregisterProgressListener(context, this)
        if (firmwareUpdateRequest != null) firmwareUpdateRequest!!.destroy()
        if (firmwareUpdateView != null) firmwareUpdateView!!.destroy()
        if (firmwareUpdateDownload != null) firmwareUpdateDownload!!.destroy()
        firmwareUpdateRequest = null
        firmwareUpdateView = null
        firmwareUpdateDownload = null
    }

    fun getDeviceAddress() = bluetoothDevice.address


    fun getFirmwareModel() = firmwareModel

    fun checkLatestAvailableFirmwareVersion(device: BluetoothDevice, firmwareFilename: String) {

        //We set the firmware version first before deciding whether to prompt the user in
firstDFURequest
        currentFirmwareFilename = firmwareFilename
```

```
        //firstDFURequest: Only prompt the user to perform DFU once when the app runs (from onCreate to onDestroy)
        if (!firstDFURequest) return
        firstDFURequest = false
        firmwareUpdateRequest = FirmwareUpdateRequest(device)
        firmwareUpdateRequest!!.setFirmwareVersionReceivedListener(this)
        if (connectedToNetwork(context)) firmwareUpdateRequest!!.checkFWversion(
            context.getString(R.string.firmware_update_url),
            firmwareFilename
        )
    }

    fun requestFirmwareVersion(requestedFirmwareRevision: String): String {
        if (currentFirmwareFilename == null) return "Could not compare the requested version with the current version. First connect with a device to request a new firmware version"
        if (currentFirmwareFilename!!.contains(requestedFirmwareRevision)) {
            return "Requested firmware version is already in use"
        }
        if (firmwareUpdateRequest == null) {
            firmwareUpdateRequest = FirmwareUpdateRequest(bluetoothDevice)
            firmwareUpdateRequest!!.setFirmwareVersionReceivedListener(this)
        }
        if (connectedToNetwork(context)) {
            firmwareUpdateRequest!!.getFirmwareVersion(
                context.getString(R.string.firmware_update_url),
                requestedFirmwareRevision
            )
            return "Requesting firmware version: $requestedFirmwareRevision"
        }
        return "Disconnected from network. Could not retrieve requested firmware version"
    }
```

```
override fun onFirmwareVersionReceived(JSONResult: String?) {
    val gson = Gson()
    firmwareModel = gson.fromJson(JSONResult, FirmwareModel::class.java)
    Log.i(TAG, "JSONString $JSONResult")
    //If the version is unavailable we do not continue
    if (firmwareModel == null) {
        Log.e(
            TAG,
            "Could not retrieve firmware version. JSON String received is: $JSONResult"
        )
        return
    }
    fetchedFirmwareFilename = firmwareModel!!.filename
    Log.i("JSON", firmwareModel!!.version)
    if (!firmwareModel!!.filename.equals(currentFirmwareFilename)) {
        if (firmwareUpdateNeeded != null) firmwareUpdateNeeded!!.onFirmwareUpdateNeeded(
            bluetoothDevice
        )
    }
}

fun createFirmwareUpdateView(applicationActivity: Activity) {
    if (firmwareModel != null) {
        firmwareUpdateView = FirmwareUpdateView(applicationActivity)
        firmwareUpdateView!!.setFirmwareDownloadClickListener(this)
        firmwareUpdateView!!.startFirmwareUpdateView(firmwareModel!!)
    } else Log.e(TAG, "Could not create firmware update view, because firmware model is null.")
}

override fun onFirmwareDownloadClick() {
    downloadFirmwareUpdate(this)
```

```
    }

    fun downloadFirmwareUpdate(firmwareDownloadProgressCallback:
FirmwareDownloadProgressUpdated?) {
        firmwareUpdateDownload = FirmwareUpdateDownload(context)
        if (firmwareDownloadProgressCallback != null)
firmwareUpdateDownload!!.setFirmwareDownloadProgressListener(
            firmwareDownloadProgressCallback
        )
        firmwareUpdateDownload!!.setFirmwareDownloadCompletedListener(this)
        firmwareUpdateDownload!!.downloadFirmwareUpdate(firmwareModel!!)
    }

    override fun onDownloadProgressUpdated(progressPercentage: Int) {
        Log.i(TAG, "DownloadProcess: $progressPercentage")
        if (firmwareUpdateView != null) firmwareUpdateView!!.updateDownloadProgressBar(
            progressPercentage
        )
    }

    override fun onFirmwareDownloadCompleted() {
        if (firmwareUpdateReady != null) firmwareUpdateReady!!.onFirmwareUpdateReady(
            bluetoothDevice
        )
    }

    /**
     * Start the DFU OTA process (Device Firmware Upgrade, Over The Air using BLE)
     * If you want to have experimental buttonless DFU feature supported call {@see
setUnsafeExperimentalButtonlessServiceInSecureDfuEnabled}:
     * but be aware of this: https://devzone.nordicsemi.com/question/100609/sdk-12-bootloader-erased-
after-programming/
     * and other issues related to this experimental service.
```

```
     *
     * @param context: Context needed in case a notification channel needs to be created
     * @param device: The com.sdk.bluetooth device with firmware to upgrade
     */
    fun startDFUupdate(context: Context, device: BluetoothDevice) {
        val path =

Objects.requireNonNull(this.context.getExternalFilesDir(Environment.DIRECTORY_DOWNLOADS
))
                .toString()
        val mFilePath = "$path/$fetchedFirmwareFilename"
        startDFUupdateWithFilePath(context, device, mFilePath)
    }


    fun startDFUupdateWithFilePath(
        context: Context,
        device: BluetoothDevice,
        updateZipPath: String
    ) {
        val starter = DfuServiceInitiator(device.address)
            .setForeground(true)
            .setDeviceName(device.name)
            .setKeepBond(true)
        starter.setUnsafeExperimentalButtonlessServiceInSecureDfuEnabled(true)
        starter.setZip(null, updateZipPath)
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            DfuServiceInitiator.createDfuNotificationChannel(context)
        }
        Log.i("dfu", "starting")

        // You may use the controller to pause, resume or abort the DFU process.
        dfuServiceController = starter.start(this.context, DfuService::class.java)
```

```
    }

    @SuppressLint("SetTextI18n")
    override fun onDeviceConnecting(deviceAddress: String) {
        if (firmwareUpdateView != null) {
            firmwareUpdateView!!.firmwareDownloadMessage?.text = "Connecting to Device."
            firmwareUpdateView!!.firmwareDownloadProgressBar?.isIndeterminate = true
        }
    }

    override fun onDeviceConnected(deviceAddress: String) {}

    @SuppressLint("SetTextI18n")
    override fun onDfuProcessStarting(deviceAddress: String) {
        if (firmwareUpdateView != null) {
            firmwareUpdateView!!.firmwareDownloadMessage?.text = "Starting firmware update."
            firmwareUpdateView!!.firmwareDownloadProgressBar?.isIndeterminate = true

firmwareUpdateView!!.firmwareDownloadDialog?.getButton(AlertDialog.BUTTON_NEUTRAL)?.is
Enabled =
                false
        }
    }

    override fun onDfuProcessStarted(deviceAddress: String) {}

    override fun onEnablingDfuMode(deviceAddress: String) {}

    override fun onFirmwareValidating(deviceAddress: String) {}

    override fun onDeviceDisconnecting(deviceAddress: String?) {}
```

```
    override fun onDeviceDisconnected(deviceAddress: String) {}

    @SuppressLint("SetTextI18n")
    override fun onDfuCompleted(deviceAddress: String) {
        Log.i(TAG, "DFU completed")
        if (firmwareUpdateView != null) firmwareUpdateView!!.firmwareUpdateCompleted()
        if (firmwareUpdateCompleted != null)
firmwareUpdateCompleted!!.onFirmwareUpdateCompleted(
            bluetoothDevice
        )
        if (firmwareUpdateRequest != null) firmwareUpdateRequest!!.destroy()
        if (firmwareUpdateView != null) firmwareUpdateView!!.destroy()
        if (firmwareUpdateDownload != null) firmwareUpdateDownload!!.destroy()
        firmwareUpdateRequest = null
        firmwareUpdateView = null
        firmwareUpdateDownload = null
    }


    override fun onDfuAborted(deviceAddress: String) {}

    @SuppressLint("SetTextI18n")
    override fun onError(deviceAddress: String, error: Int, errorType: Int, message: String) {
        if (firmwareUpdateView != null) {
            firmwareUpdateView!!.firmwareDownloadMessage!!.text = """
            Error:
            ${message.toLowerCase()}
            """.trimIndent()

firmwareUpdateView!!.firmwareDownloadDialog!!.getButton(AlertDialog.BUTTON_NEUTRAL).is
Enabled =
                true
```

```
        }

        //todo: upon an error the user should be asked to restart the DFU update
        Log.e(TAG, message + "errorType: " + errorType)
    }

    @SuppressLint("SetTextI18n")
    override fun onProgressChanged(
        deviceAddress: String,
        percent: Int,
        speed: Float,
        avgSpeed: Float,
        currentPart: Int,
        partsTotal: Int
    ) {
        if (firmwareUpdateView != null) {
            firmwareUpdateView!!.firmwareDownloadProgressBar?.isIndeterminate = false
            firmwareUpdateView!!.firmwareDownloadMessage?.text = "Updating firmware."
            firmwareUpdateView!!.firmwareDownloadProgressBar?.progress = percent
        }
    }
}
```

.

Додаток Г (обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ЗАСОБИ SOFTWARE DEVELOPMENT KIT ДЛЯ РОЗУМНИХ ПРИСТРОЇВ НА ОСНОВІ ТЕХНОЛОГІЇ BLUETOOTH LOW ENERGY

Студент групи 1АКІТ-21м

Антон ПАКУЛА

Підпис *Ім’я ПРІЗВИЩЕ*

Керівник к.т.н., професор кафедри АІІТ

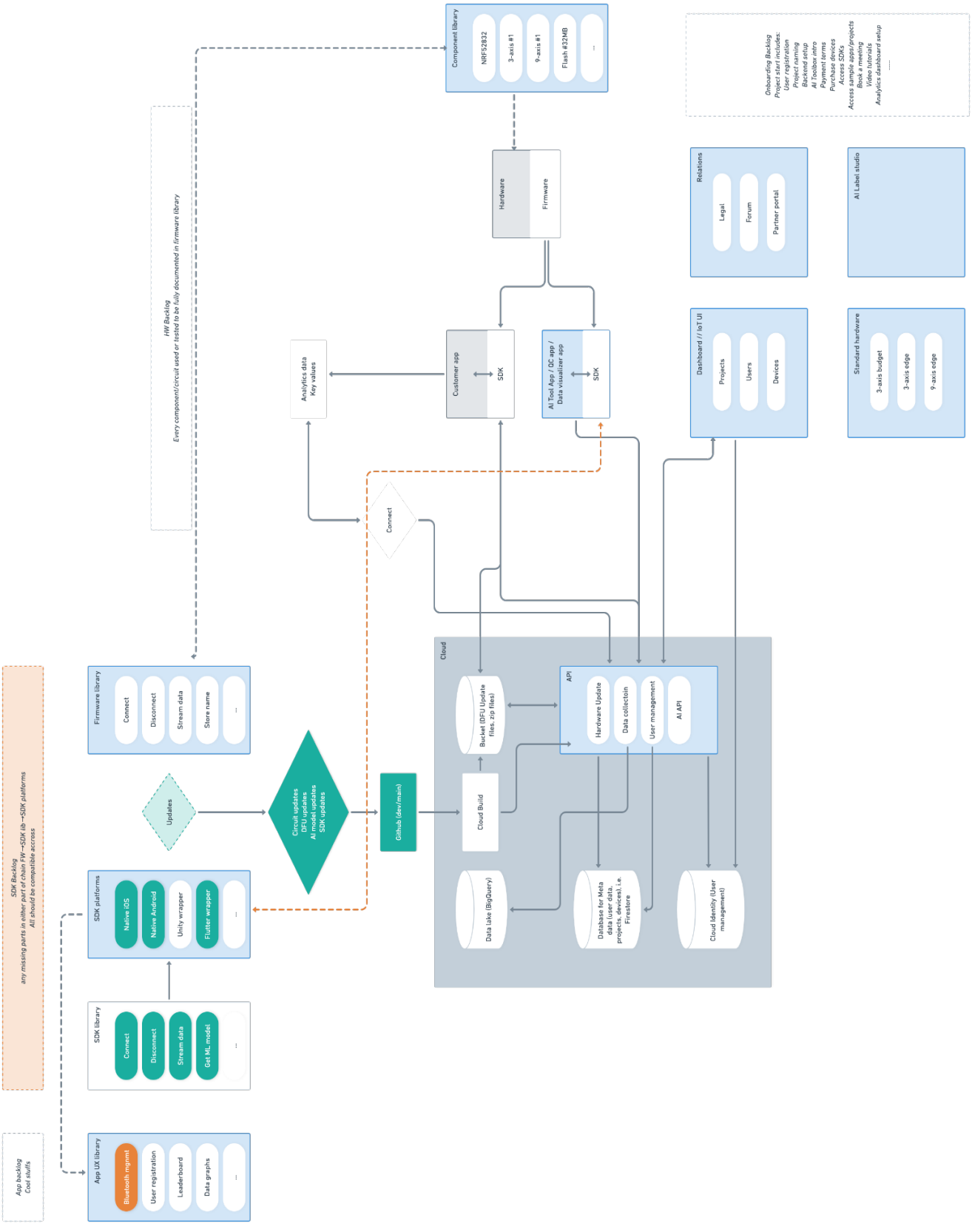
Євген ПАЛАМАРЧУК

Підпис *Ім’я ПРІЗВИЩЕ*

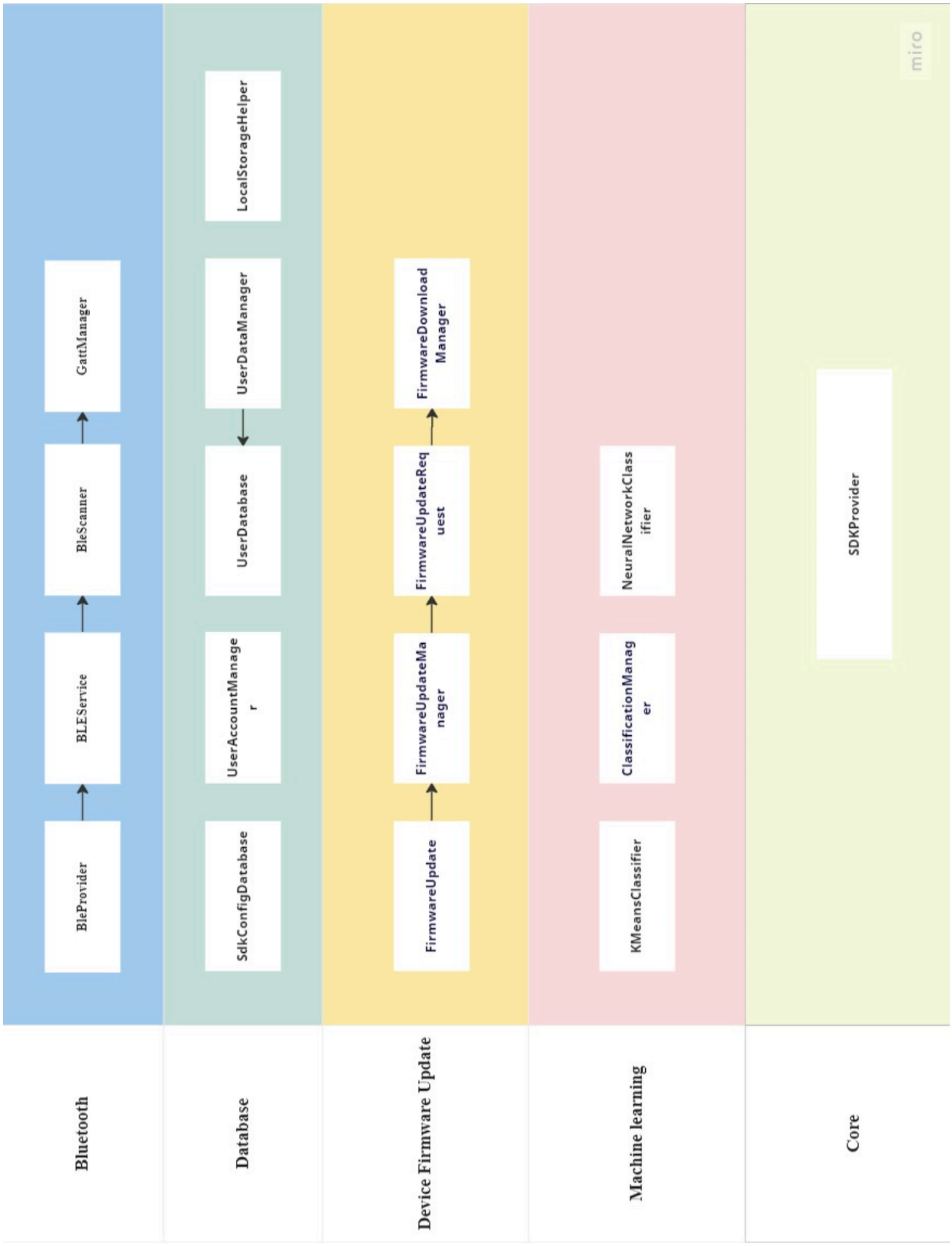
Опонент к.т.н., доцент, кафедри КН

Олег Колесницький

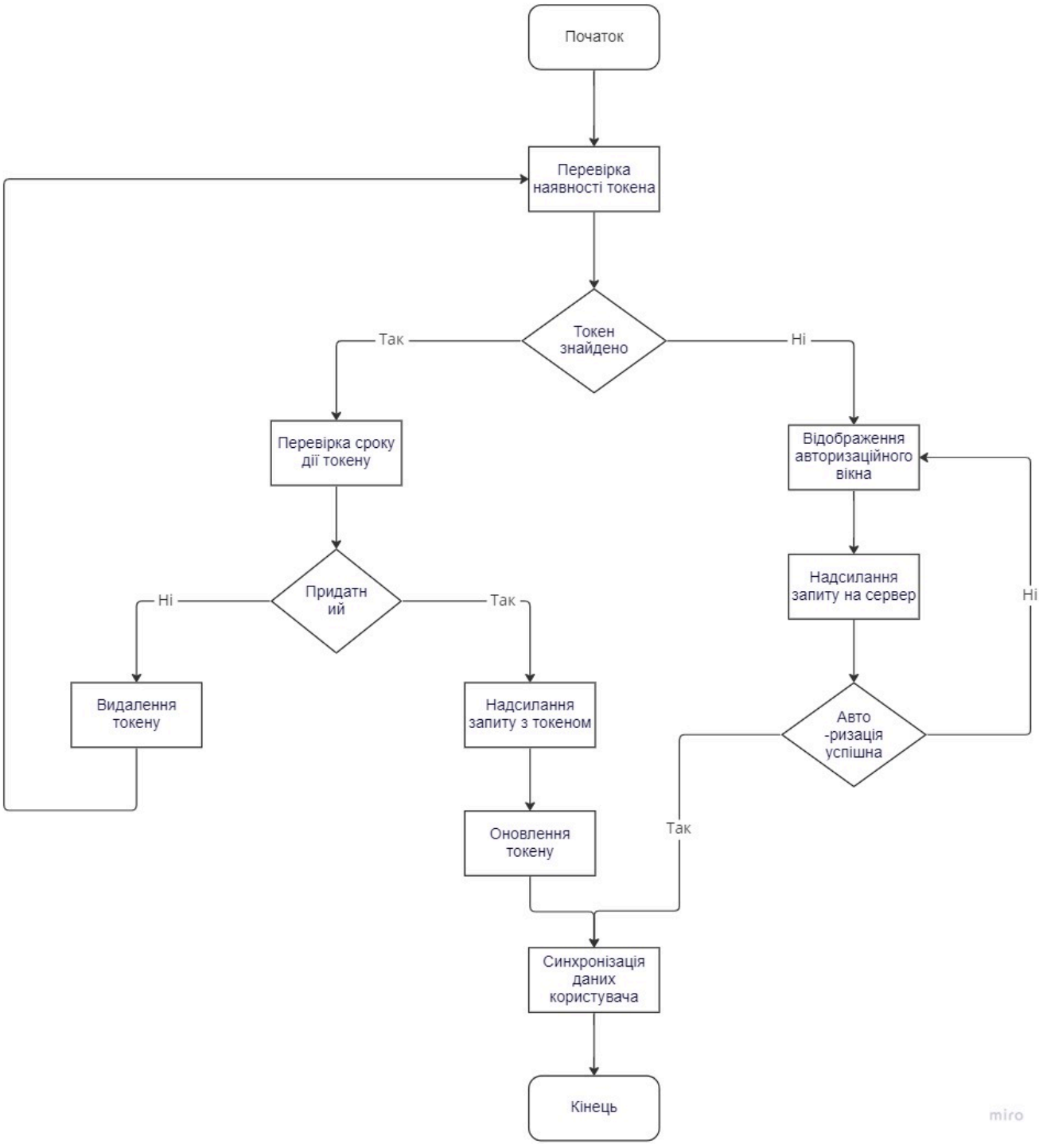
Підпис. *Ім’я ПРІЗВИЩЕ*



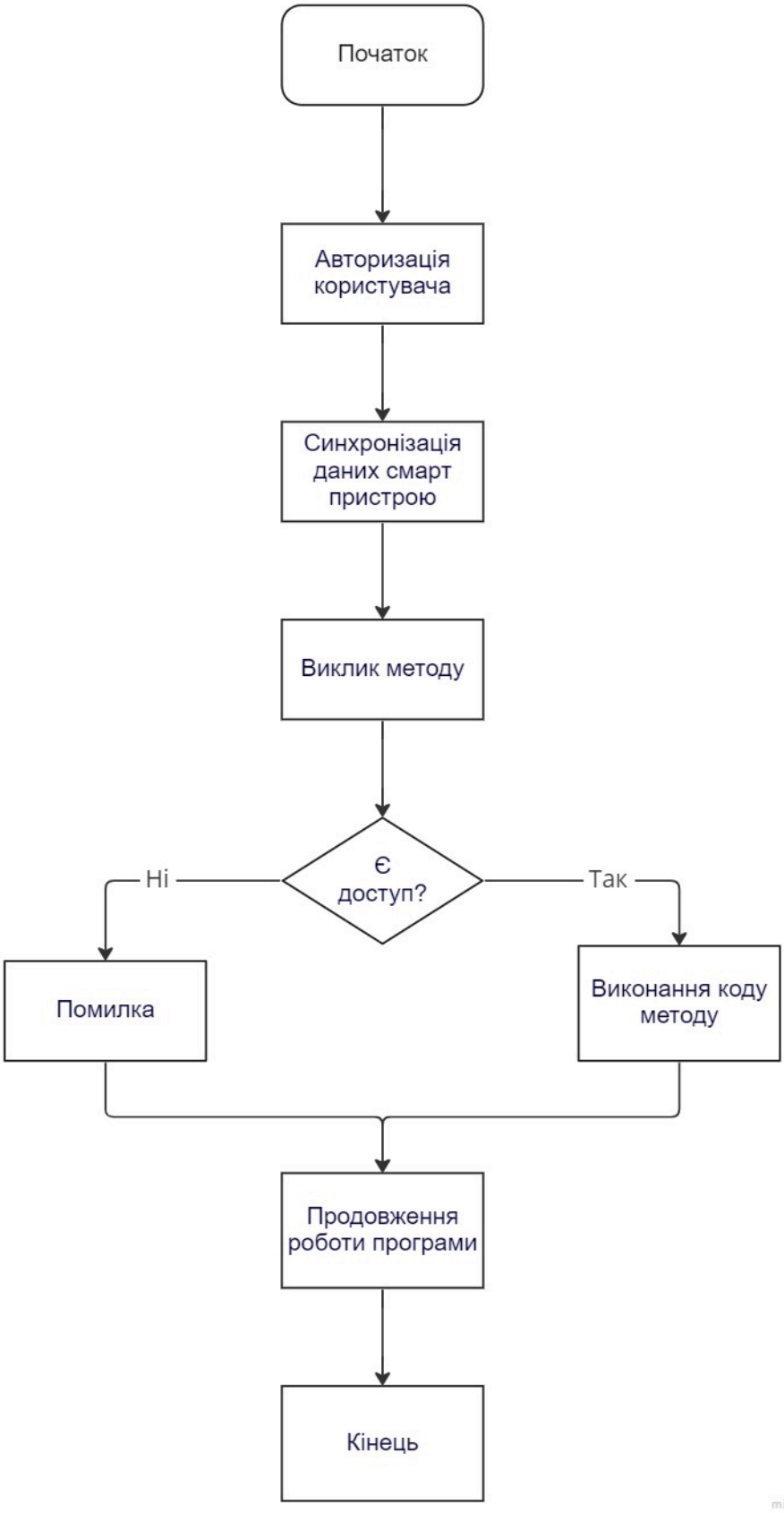
Загальна технічна архітектура набору програмного забезпечення



Архітектура модулів програмного



Алгоритм авторизації користувача



Алгоритм адаптації функціоналу

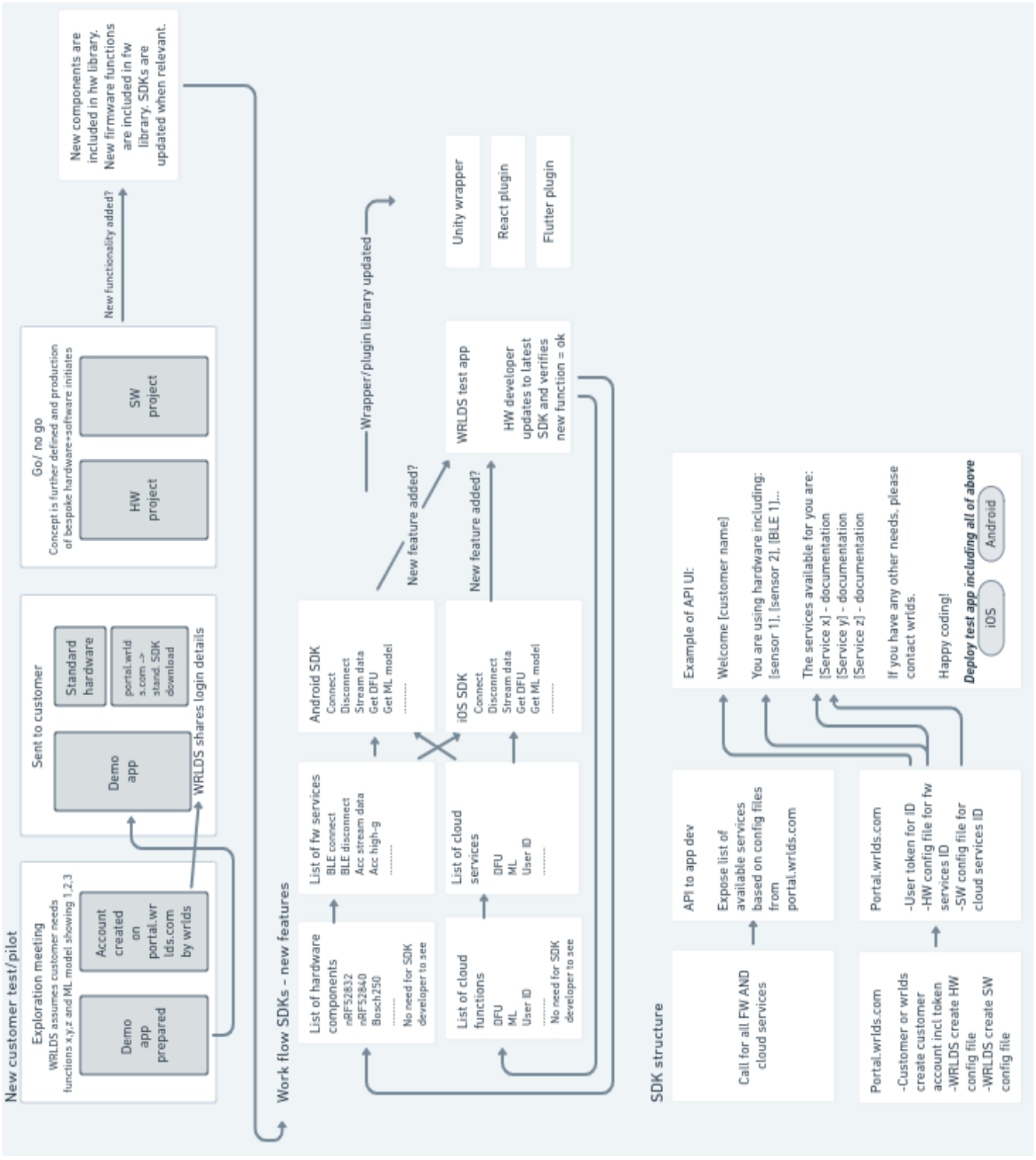


Схема взаємодії з набором програмного забезпечення

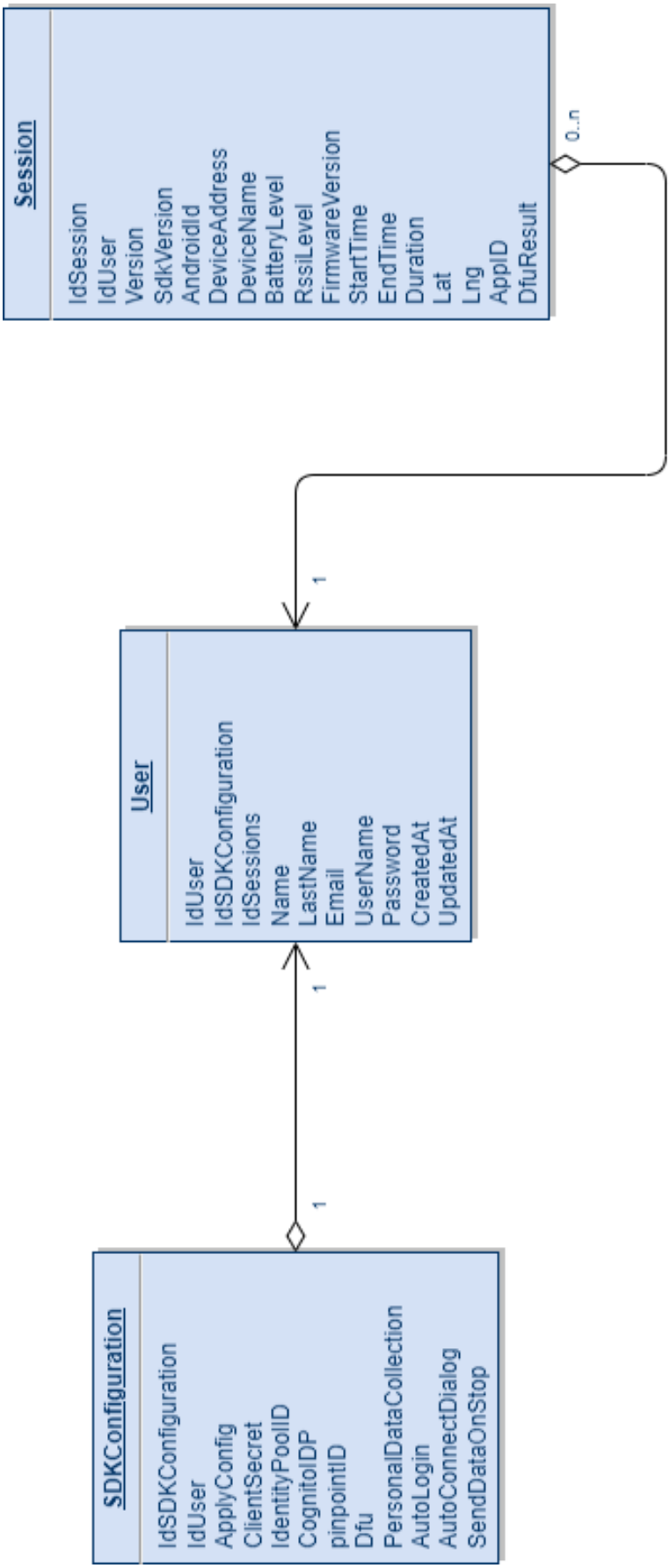


Схема локальної бази даних

Додаток Д(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: *Комплексна магістерська кваліфікаційна робота «Комп'ютеризований навчальний засіб для практичного освоєння процесу проектування HMI для системи управління виробництвом. Частина 2. Технічне проектування*

Тип роботи: кваліфікаційна робота
(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше – зазначити)

Підрозділ: кафедра АІІТ, ФКСА, 1АКІТ-21м
(кафедра, факультет, навчальна група)

Науковий керівник: Паламарчук Є.А., проф. каф. АІІТ
(прізвище, ініціали, посада)

Показники звіту подібності

Plagiat.pl (StrikePlagiarism)		*Unicheck*	
КП1	-	Оригінальність	95.3%
КП2	-		
Тривога/Білі знаки	/	Схожість	4.7%

Аналіз звіту подібності (відмітити потрібне)

X Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Пакула А.А.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення:

Допустити до захисту

Особа, відповідальна за перевірку _____ Маслій Р.В.
(підпис) (прізвище, ініціали)