МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Комп'ютерних наук,
управління та адміністрування
Кафедра Інформаційних технологій

КОМПЛЕКСНА БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка мобільного застосування для керування смарт-браслетом MGCOOL BAND 4

Склад:

Частина 1 Підсистема управління інтерфейсом зв'язку і базою даних

Виконавець: Тюртюбек Євген Максимович
(П.І.Б.)

Керівник: ст.викл. Рольщиков Вадим Борисович
(П.І.Б.)

Частина 2 Розробка користувальницької частини застосування

Виконавець: Бєлодонов Олександр Сергійович
(П.І.Б.)

Керівник: ст.викл. Рольщиков Вадим Борисович
(П.І.Б.)

Староста роботи: Тюртюбек Євген Максимович
(П.І.Б.)

Провідний керівник проекту: ст.викл. Рольщиков Вадим Борисович
(П.І.Б.)

Рецензент: к.геогр.н., доцент Лужбин Анатолій Михайлович
(П.І.Б.)

ОДЕСА 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Комп'ютерних наук,
управління та адміністрування
Кафедра Інформаційних технологій

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Підсистема управління інтерфейсом зв'язку і базою даних

Виконав студент 4 курсу групи К-25
Спеціальність 122 Комп'ютерні науки
Тюртюбек Євген Максимович

Керівник ст.викл.
Рольщиков Вадим Борисович

Консультант к.геогр.н., доцент
Кузніченко Світлана Дмитрівна

Рецензент к.геогр.н., доцент
Лужбін Анатолій Михайлович

Одеса 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИУ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Комп'ютерних наук, управління та адміністрування
Кафедра Інформаційних технологій
Рівень вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри _____
_____
" 13 " квітня 2020 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Тюртюбеку Євгену Максимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Підсистема управління інтерфейсом зв'язку і базою даних

керівник роботи ст.викл. Рольщиков Вадим Борисович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від " 13 " квітня 2020 р. № 37-С

2. Строк подання студентом роботи 30 травня 2020

3. Вихідні дані до роботи Інструкція до оригінального застосування Wearfit 2.0 і браслету MGCOOL BAND 4, Різноманітна документація: на мову Java; операційну систему Android; базу даних SQLite, керівництво програміста в середовищі розробки IDE Android Studio, дані про аналізатор пакетів Wireshark.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ
Аналіз предметної області з моделюванням процесу розробки проекту
Обґрунтування вибору програмних засобів
Опис процесу і результатів дослідження роботи смарт браслету
Опис реалізації проекту
Висновки

5. Перелік графічного матеріалу
Деякі діаграми моделюванням процесу розробки проекту
Порівняльні таблиці засобів обробки та зберігання даних
Приклади пакетів комунікації і їх аналіз
Демонстрація виконання окремих команд
UML-діаграми деяких класів застосування

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Робота загалом	Кузніченко Світлана Дмитрівна, доц.	13.04	13.03

7. Дата видачі завдання “ 13 ” квітня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Термін виконання етапів роботи	Оцінка виконання етапу	
			у %	за 4-х бальною шкалою
1	Отримання завдання. Вивчення літератури	13.04–16.04	100	Відмінно
2	Моделювання процесу розробки	16.04–18.04	100	Відмінно
3	Вибір програмного забезпечення та проведення			
	аналізу пакетів комунікації.	18.04–22.04	100	Відмінно
4	Моделювання та розробка бази даних.	22.04–26.04	100	Відмінно
5	Розробка класів-адаптерів бази даних.	26.04–01.05	100	Відмінно
6	Розробка командної підсистеми та			
	передача документації другому розробнику.	01.05–11.05	100	Відмінно
7	Рубіжна атестація	11.05–16.05	100	Відмінно
8	Супровід інтеграції класів-адаптерів та			
	командної підсистеми.	16.05–20.05	100	Відмінно
9	Тестування застосування	20.05–24.05	100	Відмінно
10	Оформлення пояснювальної записки.	25.05–30.05	100	Відмінно
	Інтегральна оцінка виконання етапів календарного плану (як середня по етапам)		**100**	**Відмінно**

Студент _____ Тюртюбек Є.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Рольщиков В.Б.
(підпис) (прізвище та ініціали)

ПЕРЕДМОВА

За останній час смарт-браслети отримали визнання у суспільстві споживачів. Такі прилади з невеликим розміром можуть збирати фізіологічні дані, які згодом можна використовувати для аналізу стану користувача або для складання статистики. У даному випадку можна відилити пристрій MGCOOL BAND 4, який є дешевше своїх аналогів та виконує необхідні функції. У зв`язку з обставинами, які будуть розглянуті пізніше даний прилад має недосконале клієнтське застосування та, можливо, недосконале програмне забезпечення у самому пристрої, але технічна частина приладу виконана задовільно.

Проект «Розробка мобільного застосування для керування смарт-браслетом MGCOOL BAND 4» у цілому повинно задовольняти такі потреби: синхронізація пристрою, обробка отриманих даних, дублювання сповіщень, реалізація допоміжних алгоритмів для підвищення досвіду користувача у користувальницькому застосуванні для того щоб відповідати мінімальним вимогам.

На рис. 1 проілюстровано усі необхідні ресурси, вихідні дані та необхідний результат даного проекту.

Рисунок 1 – IDEF0 - діаграма опису проекту у цілому

Після попереднього моделювання проекту можна розділити його у свою чергу на дві незалежні структурні одиниці та виконати їх паралельно при наявності необхідних ресурсів. У даному випадку було промодельовано дві структурні одиниці “Розробка підсистеми зв’язку та бази даних” та “Розробка застосування користувача”, які будуть розроблятися Тюртюбеком Євгеном Максимовичем, старостою даної комплексної роботи та Бєлодоновим Олександром Сергійовичем відповідно.

На рис. 2 проілюстровано діаграму декомпозиції 0 та більш детальніше розглянуто межи відповідальності кожної структурної одиниці, необхідні ресурси та результат кожної роботи у цілому.

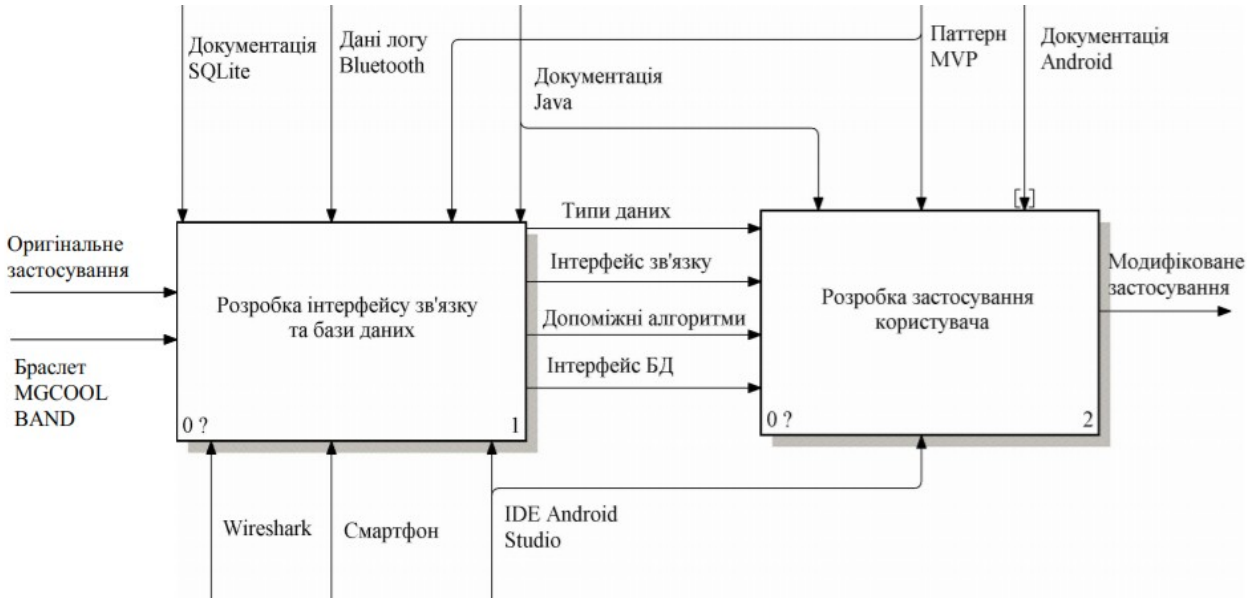


Рисунок 2 – Діаграма декомпозиції 0. Опис структурних елементів

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

СКРБД (система керування реляційними базами даних) – програмне забезпечення для розробки, аналізу та супроводу реляційних баз даних.

Фотоплетізмограмма – метод реєстрації кров'яного потоку з використанням інфрачервоного або світлового джерела вилучення.

BLE (Bluetooth Low Energy) – Bluetooth з пониженим енергоспоживанням.

GATT (Generic Attribute Profile) – профіль загальних атрибутів.

HEX (Hexadecimal) – визначення шістнадцятирічної системи числення.

IDE (Integrated Development Environment) – інтегроване середовище розробки.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, який оснований на JavaScript.

JVM (Java Virtual Machine) – віртуальна машина Java.

MVC (Model View Controller) – Модель Вигляд Керування, патерн командного програмування.

MVP (Model View Presenter) – Модель Вигляд Представлення, патерн командного програмування.

MVVM (Model View ViewModel) – Модель Вигляд ВиглядМодель, патерн командного програмування.

NoSQL (not only SQL) – термін, який визначає ряд підходів для керування документ орієнтованими або графовими базами даних.

PAC (Presentation-Abstraction-Control) – Представлення Абстракція Керування, патерн командного програмування.

Pcap (Packet capture) – технологія захвату пакетів.

UART (Universal Asynchronous Receiver-Transmitter) – інтерфейс для перетворенням даних між пристроями.

ВСТУП

Смарт браслети зараз є зручними та актуальними пристроями, які виконують безліч корисних функцій починаючи від відстеження фізіологічних та біологічних показників користувача до використовування даного пристрою як доповнення до смартфону або навіть замість смартфону.

Для накопичення даних щодо серцевого ритму використовується сенсор серцевого ритму, який використовує технологію фотоплетізмограмми для зняття показнику серцевого ритму.

Для накопичення даних щодо фаз сну необхідно використовувати усі інші дані починаючи від серцевого ритму закінчуючи трьох осевим акселерометром. Також даний сенсор може бути використаний для реагування на переміщення шагів та розрахунку кількості пройденого шляху.

Для синхронізації та передачі команд використовується технологія BLE, тобто звичайний Bluetooth з використовуванням технологій для підвищення енергетичної ефективності.

Смарт браслети накопичують інформацію та синхронізують її з базою даних на смартфоні або з хмарним сервісом. У теорії ці дані можна використовувати для відстеження стану здоров’я та передбачення проблем, відстеження фаз сну.

Також одними з потужних інструментів смарт браслетів є сповіщення, які підвищують ефективність та швидкість реагування користувачем на них у смартфоні.

У даному проекті як образок смарт браслету буде використаний пристрій MGCOOL BAND 4. Для більш масштабного проекту необхідно розробити власний пристрій, щоб забезпечити гнучкість та незалежність при розробці, але це є потребує більших ресурсів. У даному випадку достатньо використати цей прилад як чорний ящик. На аналіз такого пристрою буде витрачено набагато менше ресурсів ніж на розробку аналогічного.

Причиною розробки власного додатку є низька якість, докази цього можуть бути знайде на сторінці оригінального додатку Wearfit 2.0 [1][1)], а найголовніше повний доступ до даних та можливість застосовування власних алгоритмів.

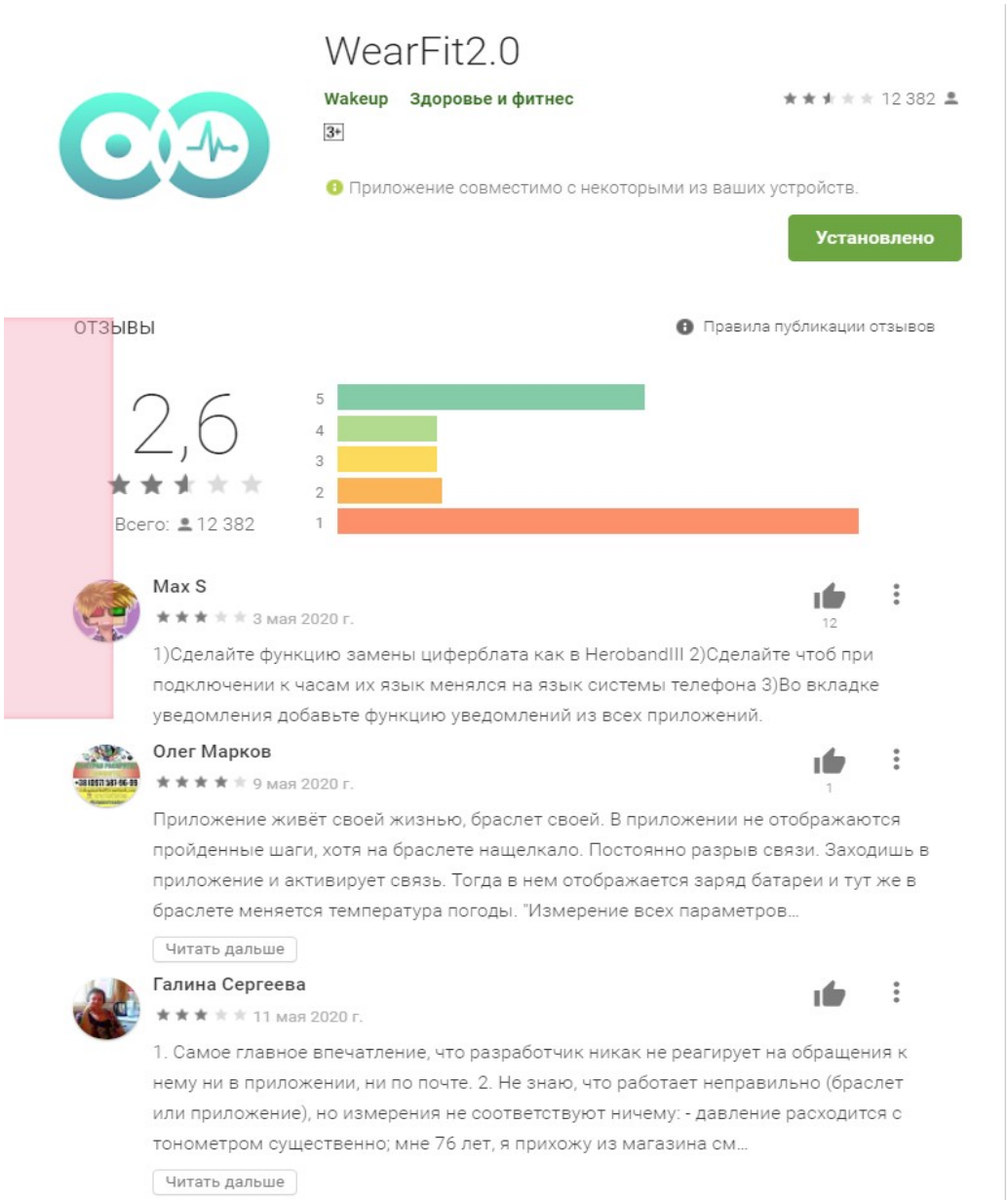


Рисунок 3 – Скріншот офіціальної сторінки Wearfit 2.0 (оригінальне застосування)

[1)] [1] Застосування у Google Play – WearFit2.0. URL: https://play.google.com/store/apps/details?id=com.wakeup.wearfit2 (дата звернення 01.02.2020).

Таким чином загальною метою цієї частини проекту є розробка додатку для керування приладом MGCOOL BAND 4 для котрого необхідно виконати відповідні роботи, а саме проаналізувати пристрій та проаналізувати отримані дані, розробити базу даних для їх систематизації, зберігання та обробки. Також необхідного розробити алгоритми для підвищення позитивного досвіду користувача та підготувати алгоритми для реалізації у другій частині диплому.

Дана пояснювальна записка містить 23 посилання, 60 сторінок, 15 рисунків та 4 таблиці.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У даному розділі будуть детально розглянути проблеми та завдання, які необхідні виконати у даній частині проекту.

Теоретично, даний прилад може використовуватись не тільки для відстеження показників та синхронізації повідомлень зі смартфону, а й для автоматичної ідентифікації користувача у системах захисту, відстеження користувача за допомогою GPS, що робить дану область актуальною для вивчення.

1.1 Опис смарт браслету MGCOOL BAND 4

Фізично, цей пристрій [2][1)] має розмір чорно-білого OLED дисплею 0,96 дюймів, що у свій час (2018 рік), було новинкою у світі смарт браслетів. Довжина – 25.5 см, ширина – 2.2 см, висота – 1.1 см, вага продукту – 23.6 грамів. Матеріал корпусу – полікарбонат. Має захист від навколишнього середовища IP67 (повний захист від пилу, короткочасне занурення у воду на глибину не більше ніж 1 метр).

Пристрій має одну сенсорну кнопку для виконання безпосереднього керування. Кнопка реагує на коротке та на тривале натиснення.

Має літій-іонний акумулятор з ємністю на 90mAh, що є приблизно 15 діб або 360 годин неперервної роботи без підзарядки.

Оперативна пам’ять – 32 кіБ. Внутрішня флеш-пам’ять – 256 кіБ.

Пристрій виконує примітивні функції такі як відображення часу, розрахунок серцевого ритму, розрахунок кількості шагів, розрахунок кількості витрачених калорій та пройденого шляху, відображення повідомлень, пошук телефону, розрахунок фаз сну.

[1)] [2] Mgcool band 4: огляд та технічні характеристики фітнес браслету. URL: http://bazaroved.ru/harakteristiki-i-polnyj-obzor-fitnes-brasleta-mgcool-band-4/ (дата звернення 01.02.2020)

Очевидно, що внутрішня пам’ять використовується для зберігання даних до синхронізації з пристроєм, знаючи що пристрій зберігає дані серцевого ритму кожні п’ять хвилин, то можна допустити що цієї пам’яті досить на як максимум на приблизно 29 000 записів (загальну кількість пам’яті поділити на 8 байтів на дату та час у вигляді YYYYMMDDHHmmSS та 1 байт на значення серцевого ритму), що є 2400 годин, але це без урахування кількості пройдених шагів та без припущення, що пам’ять не розділена програмним застосуванням мікропроцесору, але ці цифри вже говорять про те, що пристрій можна використовувати дуже довго без синхронізації, а строку дії пам’яті буде достатньо на декілька років.

У ході експлуатації було встановлено, що пристрій дійсно відповідає тим характеристикам, що були заявлені, але очевидно, що розрахунок пройденого шляху є неточним та кількість витрачених калорій має велику погрішність. Причиною цього є використовування середніх значень, котрі є індивідуальними у кожної людини.

Також окремої критики заслуговують алгоритми відстеження фаз сну, які мають неточності. Першою підозрою на це виводять в оригінальному застосуванні параметри «початок сну» та «закінчення сну». Очевидно, що використовуючи показники акселерометру та серцевого ритму можна зрозуміти де є початок сну та закінчення. Зовнішній вигляд пристрою представлений на рис. 4.

Рисунок 4 – Зовнішній вигляд пристрою MGCOOL BAND 4

1.2 Аналіз оригінального застосування Wearfit 2.0

Для розробки алгоритмів для керування смарт-браслетом необхідно мати доступ до технічної документації пристрою, що у даному випадку неможливе. Компанія-виробник не розробила документацію для даного пристрою, а усі вихідні коди закриті тому необхідно використовувати оригінальне застосування для вивчення команд та аналізу вад оригінального пристрою.

Дане застосування має велику кількість вад, що й стали однією з причин для розробки власного застосування.

Оригінальне застосування має професійно виконаний графічний дизайн та очевидно, що реалізує усі функції смарт браслету, має інтеграцію з Google Fit та іншими сторонніми сервісами для вивантаження даних, але алгоритми та гнучкість керування виконана жахливо.

У цілому, оригінальне застосування на мій погляд має такий перелік вад:

- жахлива інтеграція з повідомленнями (підтримуються лише статичні додатки, які передбачені самими розробниками);
- не працює у фоновому режимі;
- локалізація виконана за допомогою машинного переводу на російську;
- розмір додатку значно перевищує ефективний розмір (станом на 01.01.2020 застосування займає 36 Мбайт).

1.3 Моделювання та підготовка проекту

Для успішної розробки даної частини проекту необхідно передбачити його ключові етапи та вивести їх послідовність. Ці етапи та послідовності наведені на рис. 5.

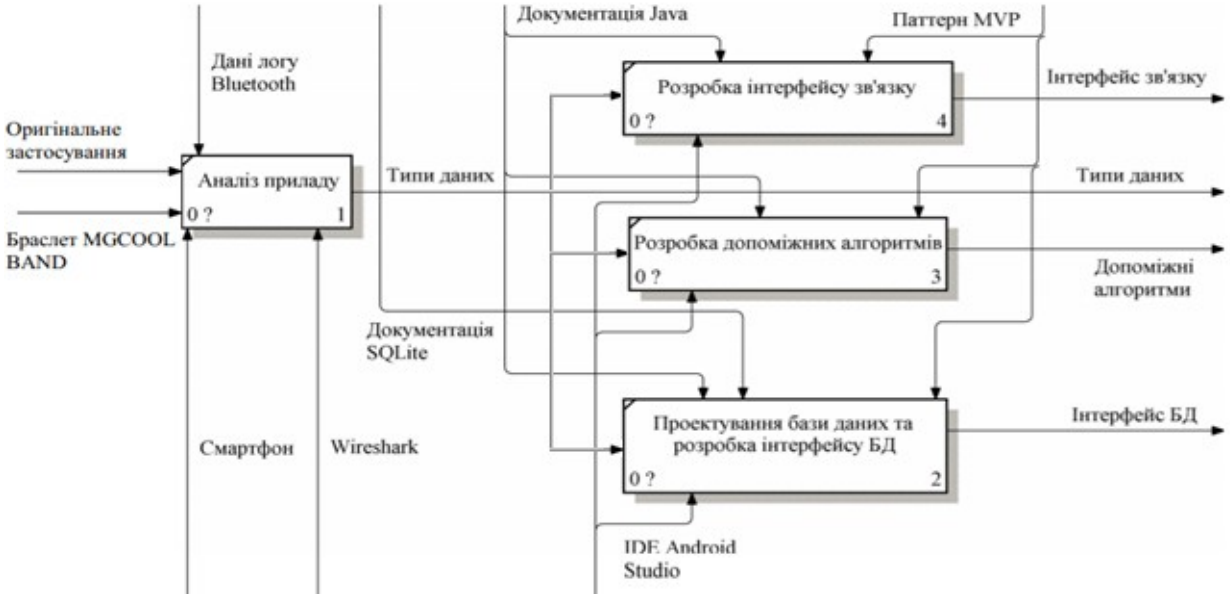


Рисунок 5 – Діаграма декомпозиції 1. Опис структурного елементу «розробка підсистеми управління інтерфейсом зв’язку і базою даних»

Для виконання запланованих робіт необхідно мати лише оригінальне застосування, а саме Wearfit 2.0, смарт-браслет та смартфон з підтримкою технологією BLE. Також необхідно мати навички роботи з необхідною мовою програмування та використовувати командний патерн програмування та інтегроване середовище. Результатами усього масиву робіт є інтерфейс зв’язку, типи даних для обробки, допоміжні алгоритми та інтерфейс БД.

Першим кроком необхідно проаналізувати пристрій за допомогою логів Bluetooth. Після виконання цієї роботи можна буде виділити типи даних та їх структуру. Також команди, які необхідно використовувати для подальших робіт та успішно керувати смарт-браслетом.

Маючи представлення як працює пристрій та які команди він оброблює можна розробити допоміжні алгоритми та інтерфейс зв’язку, також у це поняття входить різні алгоритмі для підвищення досвіду користувача. Маючи типи даних можна розробити базу даних для їх успішного збереження та обробки. Під допоміжними алгоритмами у даній роботі розуміється виключно теоретична розробка.

2 ОБҐРУНТУВАННЯ ВИБОРУ ПРОГРАМНИХ ЗАСОБІВ

2.1 Вибір програмного засобу аналізу пакетів

Для успішного виконання аналізу приладу та подальшої розробки командного модуля необхідно дослідити лог даних зі смартфону. Дані зберігаються у файл btsnoop_hci.log на смартфоні, структура якого аналогічна з pcap файлами.

Pcap-файл [3][1)] – це файл, який вміщує дані, які оброблялися доступною мережевою картою пристрою, та були записані утилітою для прослуховування мережевого каналу. Bluetooth технологія нічим не відрізняться від аналогічних мережевих технологій (наприклад Ethernet, WIFI), тобто має MAC-адреси та пакети, відмінність лише в середовищі передачі даних. У даному випадку Bluetooth з Ethernet розглядається не як окрема одиниця рівню в моделі OSI, а як сукупність технологій для передачі цифрових даних.

Важливо підкреслити, що Bluetooth відноситься до фізичного рівня у моделі OSI. Згідно з моделлю OSI додатки, які використовують Bluetooth також мають прикладний рівень (програмний продукт, який розробляється у рамках цього проекту), рівень представлення (шифрування), сеансовий рівень (UART), транспортний рівень (системне забезпечення пристрою з технологією Bluetooth) та фізичний рівень (середовище передачі даних).

У даному проекті не передбачена необхідність маршрутизувати пакети та виконувати їх перетворення. Загалом даний проект згідно моделі OSI прикладного рівня та необхідність підтримувати інші рівні цієї моделі не має необхідності та можливості. Прикладний рівень компонує команди використовуючи засоби більш високого рівня та передає їх до операційної оболонки наступні дії вже не входять в компетенцію прикладного рівня.

Для читання логу або перехвату пакетів необхідно використовувати програми-аналізатори мережевого трафіку. З усієї множини програм-

---

[1)] [3] Packet capture – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Wireshark (дата звернення 04.02.2020)

аналізаторів трафіку було обрано програму Wireshark. Інші альтернативи (наприклад Intercepter-NG, SmartSniff, MITMEngine, Cain & Abel) аналізують лише HTTP-трафік або необхідні для перехвату паролів чи сертифікатів, тобто для перевірки мережі на безпечність.

Intercepter-NG [4][1)] є потужною утилітою для вивчення мережевого трафіку, але дана утиліта більше призначена для виконування атак на мережеві вузли для перевірки безпеки. Також дана утиліта не підтримується великою командою, що робить її функціонал обмеженим та незадовільним.

SmartSniff [5][2)] аналог спрощений аналог Wireshark. Дана утиліта дозволяє прослуховувати мережеве з'єднання, але не підтримує парсинг пакетів у середовищі Bluetooth. Також дана утиліта має обмежені можливості для фільтрування пакетів на відміну від Wireshark.

Друга група альтернатив (наприклад York, Malcolm) за описом підходять для зазначених завдань, але за певними обставинами не набули популярності та визнання на відміну від Wireshark.

York [6][3)] використовується для моніторингу мережевої активності та також, як і Intercepter-NG підтримується невеликою командою та націлений більше на тестування безпеки, ніж для аналізу трафіку. Також його недоліком стосовно цього проекту є підтримка лише HTTP з'єднань.

Wireshark [7][4)] – це програма-аналізатор трафіку для комп'ютерних мереж. Дане застосування «знає» структуру різноманітних мережевих протоколів, що дозволяє розібрати пакет та прочитати передані дані. Дане застосування має ліцензію GNU GPL 2, а також відкритий вихідний код.
Програма підтримує кросплатформеність та може запускатись навіть безпосередньо на самому смартфоні.

[1)] [4] Юбілейний випуск Intercepter-NG 1.0. URL: https://habr.com/ru/post/309406/ (дата звернення 04.02.2020)

[2)] [5] SmartSniff: Packet Sniffer – Capture TCP/IP packets on your network adapter. URL: https://www.nirsoft.net/utils/smsniff.html (дата звернення 04.02.2020)

[3)] [6] York::Log all network traffic – the sz development. URL: https://www.the-sz.com/products/york/ (дата звернення 04.02.2020)

[4)] [7] Wireshark – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Wireshark (дата звернення 04.02.2020)

Причиною швидкого вибору даного програмного застосування була його велика популярність, наявність досвіду роботи у даному технічному засобі. Також даний програмний засіб має усі необхідні потреби, а саме: підтримка протоколу Bluetooth, ергономічний інтерфейс, надійність.

2.2 Вибір технології зберігання та обробки даних

У даному проекті є необхідність зберігати множину однотипних даних для аналізу та перегляду. Для зберігання таких даних необхідно використовувати технологію баз даних. Таким чином далі необхідно обирати між технологією NoSQL та SQL.

Реляційні бази даних [8][1)] забезпечують максимальну цілісність даних та прекрасно підходить для зберігання однотипних даних, також якщо структура даних заздалегідь відома, то необхідно використовувати реляційну базу даних.

Нереляційні бази [9]2) даних мають більш гнучку структуру, що дозволяє змінювати структури даних у самому процесі розробки додатку, а також спрощує внесення змін. Такі бази даних відмінно підходять для зберігання різнотипних даних або для проекту, який постійно змінює напрямок розробки або пріоритети (наприклад розробка комп’ютерних ігор).

З причини великої кількості однотипних даних, структура яких є заздалегідь очевидною для даного проекту необхідно використовувати реляційну базу даних. Таким чином необхідно робити вибір між збереженням дискового простору та швидкості обробки даних (SQL, статична структура даних) або спрощенням розробки (NoSQL, гнучкість).

Більш детальне порівняння наведено у табл. 1.

[1)] [8] Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель. SQL: повне руководство, 3-е виданя SQL: The Complete Reference, Third Edition. М.: «Вильямс», 2014, 960 с. – ISBN 978-5-8459-1654-9

Таблиця 1 – Детальне порівняння технологій SQL та NoSQL

Технологія/ характеристика	SQL	NoSQL
Масштабування	Ускладнене	Спрощене
Цілісність даних	Відмінна	Ускладнена
Схема даних	Фіксована	Динамічна
Оптимізація під комплексні запити	Так	Ні
Підтримка користувача	Комерційна, професіональна	Відкрита
Вид зберігання даних	Табличне	JSON-подібна структура
Оптимізація під кластеризацію	Ускладнене	Спрощене

Наступним завданням становиться вибір бібліотеки SQL. Існує багато бібліотек та програмних комплексів SQL для керування даними. Вибір необхідно робити між Oracle Database, Microsoft SQL, SQLite, MySQL, Apache Derby.

Microsoft SQL [10][1)] – це СКРБД, яка розроблена компанією Microsoft. Написана на мовах програмування C/C++, C#. Виконується на операційних системах UNIX, OS/2, Windows. Активно підтримується розробниками та комерційна (ліцензія Microsoft EULA). Має дуже потужні інструменти для обслуговування баз даних. У даному проекті не розглядається з причини відсутності необхідності використовувати клієнт-серверну архітектуру та наявності комерційної ліцензії. Також дана СКРБД тісно пов’язана з технологіями Windows.

Oracle Database [11][2)] – це СКРБД, яка розроблена компанією Oracle. Написана на мовах програмування Java, C/C++. Кросплатформене програмне забезпечення. Остання версія була випущена у 2019 році. Комерційна лі-

[1)] [10] Microsoft SQL Server – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Microsoft_SQL_Server (дата звернення 09.02.2020).

[2)] [11] Oracle Database – Вікіпедія. URL:https://ru.wikipedia.org/wiki/Oracle_Database (дата звернення 09.02.2020).

цензія. Як і Microsoft SQL дуже потужний інструмент та має можливість роботи з мережею. Не розглядається у рамках даного проекту з аналогічних причин як і Microsoft SQL.

MySQL [12][1)] – це вільна СКРБД, розробниками якою є компанії: MYSQLAB, Sun Microsystems та Oracle. Написана на мовах програмування C/ C++. Виконується на операційних системах Linux, Microsoft Windows, macOS, FreeBSD, Solaris. Остання версія була випущена у 2019 році, але дана СКРБД вільна та активно доповнюється членами ком'юніті та активістами. Має відкриту ліцензію GNU GPL 2. У даному проекті не розглядається хоч і має відкриту ліцензію, але дана технологія створена для більш крупніших проектів як і Microsoft SQL з Oracle Database.

У висновку можна сказати, що Microsoft SQL, Oracle Database та MySQL використовуються для виконання складних проектів. Дані технології використовують у банківських та торгових сервісах, також вони мають здатність для роботи у мережі, що не є пріоритетом у даному проекті.

Технології SQLite та Apache Derby використовуються для інтегрування у додатки та виконуються локально. Також обидві технології мають можливість взаємодії з бібліотеками Java.

Apache Derby [13][2)] – це СКРБД, яка призначена для інтеграції у Java-додатки або обробки транзакцій у реальному часі. Розроблено компанією Cloudscape Inc, пізніше IBM. Написана на мові програмування Java та відповідно, підтримує кросплатформеність. Остання версія була випущена у 2020 році. Розповсюджується згідно з ліцензією Apache License 2.0. Має можливість виконуватися локально та з розміром ядра 2 мегабайти. Дана технологія повністю відповідає вимогам даного проекту та є одним з кандидатів на інтеграцію у даний проект.

[1)] [12] MySQL – Вікіпедія. URL: https://ru.wikipedia.org/wiki/MySQL (дата звернення 09.02.2020).

[2)] [13] Apache Derby – Вікіпедія. URL:https://ru.wikipedia.org/wiki/Apache_Derby (дата звернення 09.02.2020).

SQLite [14][1)] – це компактна СКРБД, яка розроблена Річардом Хіппом на мові програмування С. Вихідний код було передано у суспільне надбання та у 2005 році отримало нагороду Google-O'Reilly Open Source Awards. Остання версія цього інструменту було випущено у 2020 році. Було обрано саме цю технологію оскільки розробник проекту має досвід роботи з SQLite, а також даний інструмент не має жодної ліцензії, що вирішує потенційні проблеми з бюрократією.

Детальне порівняння технологій наведено у табл. 2.

Таблиця 2 – Детальне порівняння засобів SQL

Технологія / Характеристика	Oracle DB	MySQL	Microsoft SQL	Apache Derby	SQLite
Виконання	Мережеве				Локальне
Транзакції	Підтримуються			Обмежені	
Типи даних	Комплексні			Спрощені	
Кросплатформеність	Підтримується		Немає	Підтримується	
Обмеження	Незначні			Значні	
Ліценція	Комерційна			Apache 2.0	Немає
Кластеризація	Підтримується				Немає

Для виконання даного проекту було обрано технологію SQLite з причини, що дані завжди будуть зберігатися локально та не потребують мережевої синхронізації. Також дані, які будуть оброблюватися не будуть перевищувати обмеження, які зазначені розробником SQLite.

2.3 Вибір мови програмування та інтегрованого середовища розробки

Цільовою платформою для мобільного додатку є смартфон, на якому виконується операційна система IOS або Android. Було прийнято рішення

[1)] [14] SQLite – Вікіпедія. URL: https://ru.wikipedia.org/wiki/SQLite (дата звернення 09.02.2020).

відмовитися від підтримки операційної системи IOS з причини складності компіляції та запуску програм під операційну систему IOS, а саме необхідність мати пристрій з операційною системою IOS. Також розповсюдження програми не передбачається таким чином підтримка операційної системи IOS не є пріоритетом. Таким чином будуть розглянуті інструменти для розробки застосувань для операційної системи Android.

Для створювання додатку на Android можна використовувати технології Xamarin, Eclipse, Android Studio, Qt.

Xamarin [15][1)] – це технологія, яка підтримується компанією Microsoft та інтегрована у середовище Visual Studio. Мова програмування – C#. Велика підтримка ком'юніті. Підтримка операційних систем Windows, Windows Phone, IOS, Android. Середовище розробки Visual Studio має безліч компонентів для розробки та налагодження застосувань та при цьому зручний та не перевантажений інтерфейс. У даному проекті не розглядається з причини відсутності необхідності підтримувати кросплатформеність.

Eclipse [16][2)] – це інтегроване середовище розробки, яке підтримується компанією Oracle. З необхідними плагінами та доповненнями може запускати та розробляти додатки для операційної системи Android або любої іншої, на якій виконується JVM. Мова програмування – Java. Має прості засоби для розробки та відладки застосувань. У даному проекті не розглядається з причини складності налаштування середовища розробки, інші альтернативи мають готові рішення.

Qt [17]3) – це кросплатформений фреймворк, який використовує для розробки мову програмування C++. Має відкритий вихідний код та ліцензію GNU GPL. З необхідними завантаженнями може розроблювати на будь-яку платформу. Має безліч компонентів для розробки та налагодження застосувань, але інтерфейс перевантажений та складний для вивчення. У

[1)] [15] Xamarin | Open-source mobile app platform for .NET. URL: https://dotnet.microsoft.com/apps/xamarin (дата звернення 13.02.2020).

[2)] [16] The Platform for Open Innovation and Collaboration | The Eclipse Foundation. URL: https://www.eclipse.org (дата звернення 13.02.2020).

даному проекті не розглядається з причиною аналогічною як у Xamarin, а саме відсутності необхідності підтримувати кросплатформеність.

Android Studio [18][1)] – це інтегроване середовище, яке підтримується компанією Google. Мова програмування – Java/Kotlin. Має відмінні засоби для розробки або налагодження застосувань під операційну систему Android.

Ураховуючи наявні ресурси проекту, а саме спільні мови програмування, які мають розробники даного проекту було прийнято рішення обрати середовище розробки Android Studio. Дане середовище відмінно підходить для розробки мобільного додатку.

Єдиний недолік даного вибору є те, що у разі необхідності розробки додатку під інші мобільні платформи буде необхідно переписати програмний код спочатку.

Більш детальне порівняння наведено у табл. 3.

Таблиця 3 – Детальне порівняння інтегрованих середовищ розробки

Технологія або Характеристика	Android Studio	Xamarin (Visual Studio)	Qt	Eclipse IDE
1	2	3	4	5
Платформи	Android	Android,Windows Phone, IOS, Unix	Більшість	JVM
Підтримка нативних функцій	Максимальна для Android	Висока	Висока	Мінімальна
Інтерфейс	Зручний		Складний	Задовільний
Алгоритми рефактору коду	Відмінні		Задовільні	Незадовільні

[1)] [18] Download Android Studio and SDK tools | Android Developers. URL: https://developer.android.com/studioc (дата звернення 13.02.2020).

Продовження таблиці 3

1	2	3	4	5
Мова програмування	Java, Kotlin	C#	C++	Java
Підтримка	Висока	Задовільна	Задовільна	Висока
Інтеграція з засобами відладки	Максимальна для Android	Висока	Задовільна	Незадовільна

2.4 Вибір шаблону проектування

При розробці командного проекту, де одна частина роботи розділяється між декількома розробниками необхідно використовувати шаблон проектування. Найвідомішими шаблонами проектування є Model-View-Controller (MVC), Model-View-Presenter (MVP), Model-View-ViewModel (MVVM), Presentation-Abstraction-Control (PAC).

PAC – це шаблон проекту, який використовується у ієрархічній структури агентів, котрі у свою чергу складаються з презентації, вигляду та контролеру.

Model-View-ViewModel – це шаблон проектування у якому дані тісно зв’язані с інтерфейсом та проходять через контролер. Таким чином можна пов’язувати дані у обидві сторони.

MVP – це шаблон проектування, який був розроблений для спрощення модульного тестування та підвищування ефективності розділення відповідальності у виконанні одиниць роботи.

Очевидно, що необхідно використовувати шаблон проектування MVP з причини зручного розділення меж відповідальності у даному проекті. Сутність патерну програмування MVP буде представлена на рис. 6.

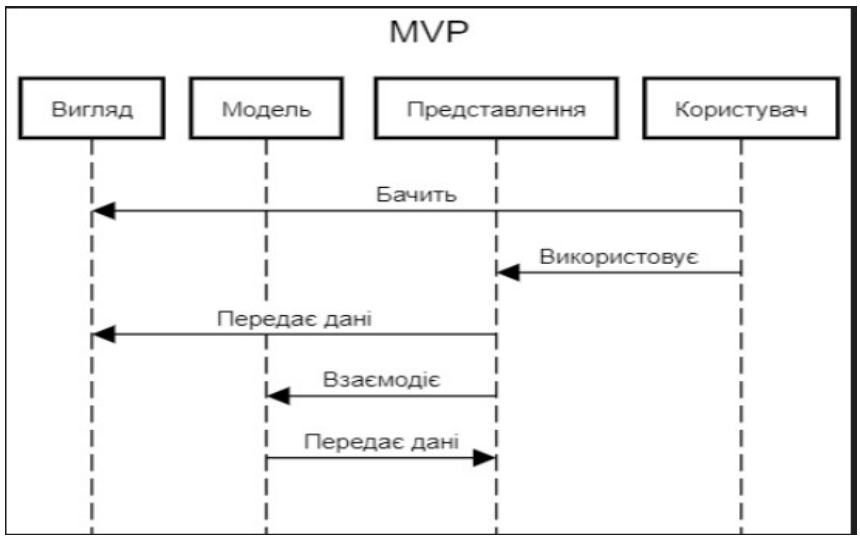


Рисунок 6 – Діаграма послідовності сутності MVP

Таким чином можна зробити висновок, що користувач бачить частину View, котра у свою чергу розроблюється у другій частині проекту. Далі користувач запитує інформацію використовуючи Presenter, котрий у свою чергу розроблюється у першій та другій частині даного проекту. Частина Model зберігає та оброблює дані та передає їх у Presenter. Частина Model повністю виконується у даній частині проекту.

2.5 Вибір програмного засобу для відправки пакетів Bluetooth

Одним з етапів розробки проекту буде аналіз пакетів Bluetooth, таким чином необхідно використовувати засіб для налагоджування та тестування розроблених команд. Таким чином необхідно знайти програмний засіб, який може відправляти пакети до обраного пристрою.

Було прийнято рішення використовувати застосування BLE Tool [19][1)]. Дане застосування відрізняється від своїх аналогів у простоті використання, мінімальному навантаженні, можливості переглядати GATT характеристики пристрою, підтримку BLE, відсутності реклами та найголовніше дане за-

[1)] [19] Застосування в Google Play – BLE Tool. URL: https://play.google.com/store/apps/details?id=com.lapis_semi.bleapp (дата звернення 15.02.2020)

стосування має можливість відправки повідомлень через інтерфейс Bluetooth у шістнадцяти річному коді. З недоліків у даному додатку відсутні додаткові алгоритми та функції, які спрощують роботу з застосуванням (наприклад не має можливості запам’ятати вже відправлені команди або повторити їх).

2.6 Вибір системи контролю версій

Для підвищення ефективності передачі розробленого коду та зберігання копій виконаної роботи, а також аналізу продуктивності у командних проектах необхідно використовувати системи контролю версій. Також рекомендується використовувати локальні системи контролю версій для резервування вже виконаної роботи.

Відомі системи керування та контролю версіями: Microsoft Team Foundation, Azure DevOps, Git, Github.

Git [20][1)] – це локальна система контролю версій, яка підтримує резервування, злиття, розподілення та централізацію. Підтримка даних функцій відмінно підходить для командної розробки. Розроблена компанією Software Freedom Conservancy на мові програмування C/C++, Perl, Python, Tcl, Bash UNIX, що робить її універсальної для будь якої операційної системи. Остання версія була випущена у 2020 році. Має відкриту ліцензію GNU GPL 2. Для підтримки зовнішніх репозитаріїв необхідно підключити плагіни.

Github [21][2)] – це інтернет ресурс, який використовує технології Git з декількома модифікаціями для розробників. Розробниками цієї системи є Кріс Внстрас, Пі Джей Хайеетт, Том Престон-Вернер. Постійно оновлюється відкриваючи більше можливостей для користувачів. Володіє системою на даний момент Microsoft. Має як безоплатну так і комерційну підписку.

[1)] [20] Git – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Git (дата звернення 15.02.2020)

[2)] [21] Github – Вікіпедія. URL: https://ru.wikipedia.org/wiki/GitHub (дата звернення 15.02.2020)

Azure DevOps [22][1)] – це комерційне середовище, яке підтримується корпорацією Microsoft, підтримує безліч інструментів для релізу проектів, зберігання аналітики та інше. Розроблено компанією Microsoft використовуючи мову програмування C++. Недоліками є те, що системи націлені лише на технології Microsoft та комерційність даної платформи. Також дана система дуже перевантажена, що робить її швидкість низькою.

Microsoft Team Foundation [23][2)] – це комплексна середа для координації та планування завдань між великою кількістю розробників. Має комерційну систему контролю версій, яка підтримується компанією Microsoft та розроблена за допомогою мов програмування C++ та C#. Постійно оновлюється. У даному проекті не розглядається з причини очевидних переваг утиліти Git та сервісу Github.

Детальніше порівняння цих сервісів наведено у табл. 4.

Таблиця 4 – Детальне порівняння сервісів командної розробки

Технологія Характеристика	GitHub	Azure DevOps	Microsoft Team Foundation
1	2	3	4
Функції злиття коду	Присутні		
Планування	Обмежене	Відсутнє	Відмінне
Відстеження змін	Присутнє		
Резервування	Присутнє	Відсутнє	Присутнє
Розгортання	Відсутнє	Присутнє	Обмежене
Підтримка	Професійна та ком'юніті, високий рівень	Професійна	
Інтеграція з IDE	Висока	Тільки IDE Visual Studio	

[1)] [22] Azure DevOps Server – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Azure_DevOps_Server (дата звернення 15.02.2020)

[2)] [23] Team Foundation Server – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Team_Foundation_Server (дата звернення 15.02.2020)

3 ДОСЛІДЖЕННЯ СМАРТ БРАСЛЕТУ

3.1 Особливості реалізації та виконування роботи

Перед етапом проектування необхідно дослідити смарт-браслет, який у свою чергу у даному проекті є «чорним ящиком». Як відомо раніше, необхідно дослідити комунікацію між пристроєм та смартфоном через протокол Bluetooth, використовуючи програмний засіб Wireshark та оригінальне застосування для даного пристрою. Після цього стане відомо, чи використовується корпоративне шифрування, типи даних, структуру даних.

3.2 Дослідження пакетів Bluetooth. Дослідження команд керування

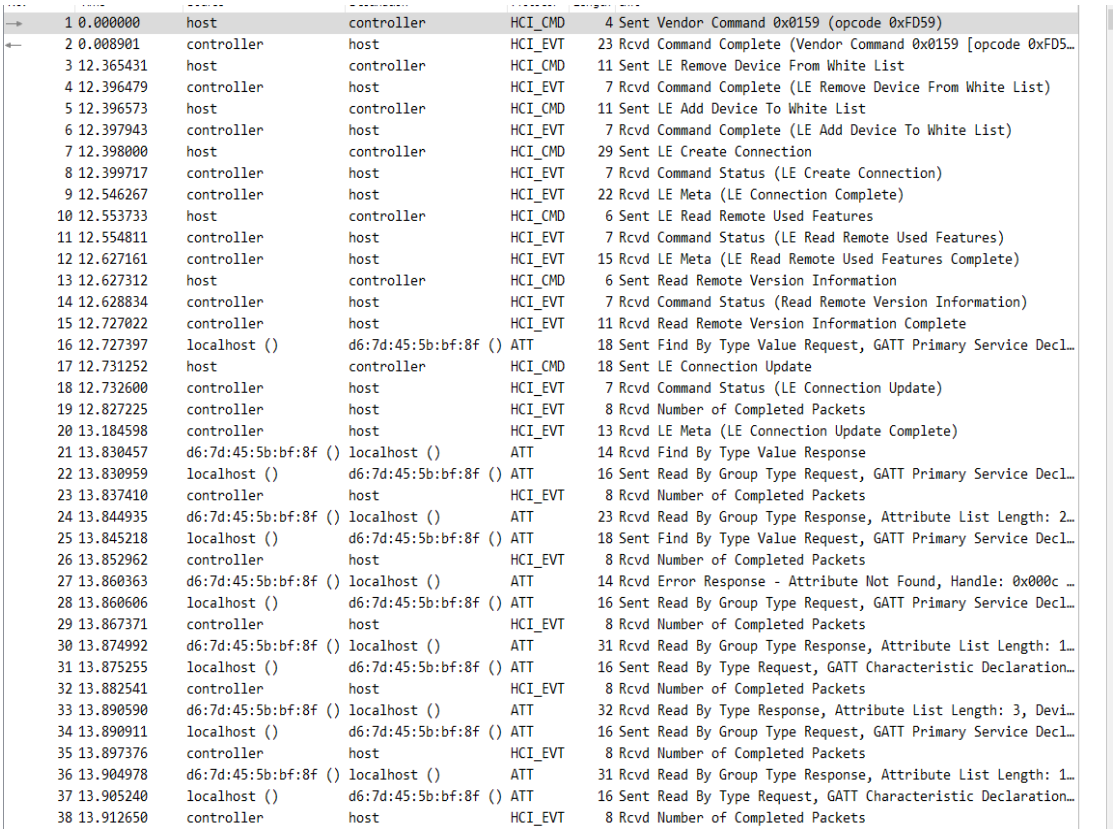
Для того щоб смартфон виконував захват пакетів Bluetooth необхідно у налаштуваннях розробника увімкнути функцію «Enable Bluetooth HCI snoop log» таким чином смартфон буде дампувати усі дані, які проходять через заданий протокол комунікації у файл btsnoop_hci.log. Даний файл можна відкрити за допомогою програми Wireshark та дослідити. Приклад цього файлу представлений на рис. 7.

Дані, які транслювалися у протоколі Bluetooth були успішно записані та успішно інтерпретувалися програмним засобом Wireshark. Було налаштовано фільтри для виконування аналізу даних. Правильним налаштуванням фільтрів для початку є встановлення MAC-адреси приймача та відправника до D6:7D:45:5B:BF:8F. Дана адреса має бути записана на самому смарт-браслеті або її можна дізнатися через журнал з’єднань у самому смартфоні. Дана адреса належить до смарт-пристрою, яку можна побачити при взаємодії з пристроєм. Встановлювання даних фільтрів допоможе виділити інформацію, яка стосується лише даного завдання.

Очевидно, що смартфон відправляє сигнал до смарт-браслету, а смарт-браслет відправляє відповідь, таким чином якщо налаштувати фільтрацію лише до адреси приймаючого то можна буде виділити усі керуючи команди, як-

що встановити адресу лише відправника, то можна буде виділити усі відповіді, таким чином можна зрозуміти структуру даних.

Для прискорення виконування аналізу необхідно використовувати усі доступні джерела, а саме сам фізичний пристрій (відображає значну частину інформації), оригінальне застосування.



→	1	0.000000	host	controller	HCI_CMD	4 Sent Vendor Command 0x0159 (opcode 0xFD59)
←	2	0.008901	controller	host	HCI_EVT	23 Rcvd Command Complete (Vendor Command 0x0159 [opcode 0xFD5…
	3	12.365431	host	controller	HCI_CMD	11 Sent LE Remove Device From White List
	4	12.396479	controller	host	HCI_EVT	7 Rcvd Command Complete (LE Remove Device From White List)
	5	12.396573	host	controller	HCI_CMD	11 Sent LE Add Device To White List
	6	12.397943	controller	host	HCI_EVT	7 Rcvd Command Complete (LE Add Device To White List)
	7	12.398000	host	controller	HCI_CMD	29 Sent LE Create Connection
	8	12.399717	controller	host	HCI_EVT	7 Rcvd Command Status (LE Create Connection)
	9	12.546267	controller	host	HCI_EVT	22 Rcvd LE Meta (LE Connection Complete)
	10	12.553733	host	controller	HCI_CMD	6 Sent LE Read Remote Used Features
	11	12.554811	controller	host	HCI_EVT	7 Rcvd Command Status (LE Read Remote Used Features)
	12	12.627161	controller	host	HCI_EVT	15 Rcvd LE Meta (LE Read Remote Used Features Complete)
	13	12.627312	host	controller	HCI_CMD	6 Sent Read Remote Version Information
	14	12.628834	controller	host	HCI_EVT	7 Rcvd Command Status (Read Remote Version Information)
	15	12.727022	controller	host	HCI_EVT	11 Rcvd Read Remote Version Information Complete
	16	12.727397	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	18 Sent Find By Type Value Request, GATT Primary Service Decl…
	17	12.731252	host	controller	HCI_CMD	18 Sent LE Connection Update
	18	12.732600	controller	host	HCI_EVT	7 Rcvd Command Status (LE Connection Update)
	19	12.827225	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	20	13.184598	controller	host	HCI_EVT	13 Rcvd LE Meta (LE Connection Update Complete)
	21	13.830457	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	14 Rcvd Find By Type Value Response
	22	13.830959	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	16 Sent Read By Group Type Request, GATT Primary Service Decl…
	23	13.837410	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	24	13.844935	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	23 Rcvd Read By Group Type Response, Attribute List Length: 2…
	25	13.845218	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	18 Sent Find By Type Value Request, GATT Primary Service Decl…
	26	13.852962	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	27	13.860363	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	14 Rcvd Error Response - Attribute Not Found, Handle: 0x000c …
	28	13.860606	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	16 Sent Read By Group Type Request, GATT Primary Service Decl…
	29	13.867371	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	30	13.874992	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	31 Rcvd Read By Group Type Response, Attribute List Length: 1…
	31	13.875255	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	16 Sent Read By Type Request, GATT Characteristic Declaration…
	32	13.882541	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	33	13.890590	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	32 Rcvd Read By Type Response, Attribute List Length: 3, Devi…
	34	13.890911	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	16 Sent Read By Group Type Request, GATT Primary Service Decl…
	35	13.897376	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
	36	13.904978	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	31 Rcvd Read By Group Type Response, Attribute List Length: 1…
	37	13.905240	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	16 Sent Read By Type Request, GATT Characteristic Declaration…
	38	13.912650	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets

Рисунок 7 – Практичний приклад файлу btsnoop_hci.log

Досліджуючи файл можна побачити як і звичайний запит-відповідь, що більше схоже на системні чи службові запити самого протоколу Bluetooth (транспортний рівень) так і запити UART-сервісу, що більше схоже на запити прикладного рівня, які необхідно досліджувати. Зазвичай на запити через інтерфейс UART відповідь буде виконана не одним пакетом, а декількома, очевидно, що даний набір пакетів має корисні дані з пристрою. На прикладі очевидно, що усі пакети є службовими, а саме призначені для синхронізації

GATT даних (канали прийому даних, серійний номер, інше). Важливо підкреслити, що наведений приклад немає увімкнених фільтрів.

Приклад UART запитів можна побачити на рис. 8.

132	14.449977	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	26 Sent Write Command, Handle: 0x0011 (Nordic UART Service: Nordic UART Tx)
135	14.527267	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	29 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
136	14.627217	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	20 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
137	14.727484	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	20 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
138	14.827480	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	20 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
139	15.455066	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	29 Sent Write Command, Handle: 0x0011 (Nordic UART Service: Nordic UART Tx)
141	15.627278	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	29 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
142	15.727750	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
143	15.827539	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
144	15.927473	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
145	16.027526	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
146	16.127447	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
147	16.227501	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
148	16.327490	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
149	16.427507	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
150	16.459557	localhost ()	d6:7d:45:5b:bf:8f ()	ATT	29 Sent Write Command, Handle: 0x0011 (Nordic UART Service: Nordic UART Tx)
152	16.527521	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
153	16.627759	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	29 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
154	16.727751	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
155	16.877748	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
156	16.927468	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
157	17.027738	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
158	17.127742	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
159	17.227767	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
160	17.327777	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…
161	17.427833	d6:7d:45:5b:bf:8f ()	localhost ()	ATT	24 Rcvd Handle Value Notification, Handle: 0x000e (Nordic UART Service: Nordic UART…

Рисунок 8 – Приклад UART запитів

На рис. 6 можна побачити як на один запит смарт браслет відповідає серією відповідей. Якщо переглянути серію відповідей то можна побачити статичну та динамічну структуру. Очевидно, що статична структура – це ідентифікатори команд, а динамічна – це дані, які передаються. Зіставивши дані які передаються з даними з оригінального застосування та самого пристрою можна побачити, що спочатку за допомогою команди AB 00 0B FF 93 80 00 07 E4 01 0F 02 33 26 було синхронізовано час та дату. Зробити такий висновок можна виходячи із інформації, що це був один з перших запитів на який не було жодної реакції, а також аналізуючи декілька схожих таких команд можна побачити, що після 6го байту (00) йде рік (кодується двома байтами), місяць, день, часи, хвилини, секунди. На прикладі цієї команди ми бачимо що команда теж має заголовок (статичні дані) та роздільник між параметрами (динамічні дані).

Перевірити результати цього аналізу можна надіславши на пристрій тестові дані. Було відправлено за допомогою засобу BLE Tool тестові дані для перевірки гіпотези. Буде відправлена команда AB 00 0B FF 93 80 00 07 E4 01 28 02 34 26. Перші шість байтів – це заголовок команди, сьомий байт – це роздільник, восьмий та дев’ятий байт – це закодований поточний рік (2020 у десятинній, 07 E4 у HEX), десятий байт – це номер поточного місяця (01 у десятинній та HEX), одинадцятий байт – це номер дня у даному випадку це 40 у десятинній, спеціально обрана величина для доказу роботи гіпотези, дванадцятий байт – це година дня, тринадцятий – це хвилина часу, чотирнадцятий байт – це секунди. Таким чином отримуємо 2020 рік 1 місяць 40 день 2 годину та 52 хвилину. Результат виконання цієї команди можна побачити на рис. 9.

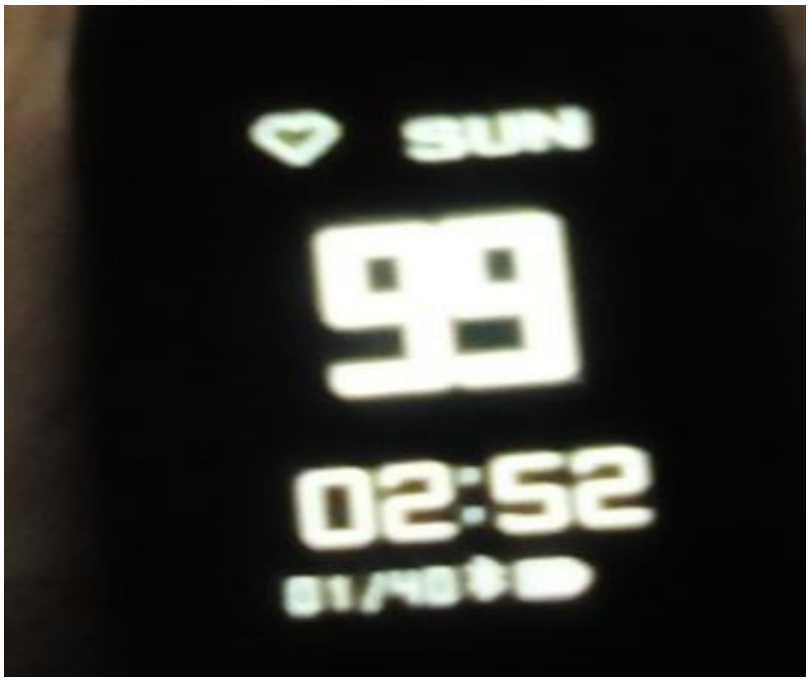


Рисунок 9 – Результат виконання команди
AB 00 0B FF 93 80 00 07 E4 01 28 02 34 26

Гіпотеза підтверджена, отже таким чином необхідно дослідити інші команди керування та приступити до розробки програмного модуля.

3.3 Дослідження пакетів Bluetooth. Дослідження відповідей

Основна відмінність пакетів з відповідями від пакетів з командами керування це те, що дані пакети мають структуру даних, яку необхідно дослідити, а далі оброблювати та зберігати.

Досліджуються запити, які отримані командою керування AB 00 0E FF 51 80 00 12 01 12 01 0F 02 12 01 0F FF FF. Дана команда є командою синхронізації інформації, яка була зібрана смарт-браслетом. Тобто перші дев’ять байт – це заголовок команди з роздільником, з десятого по тринадцятий – це дата останньої синхронізації, чотирнадцятий – це роздільник, а з п’ятнадцятий по дев’ятнадцятий – це дублікат дати, схоже для контролю дати. Результатом виконування даної команди були такі відповіді:

```
AB 00 0E FF 51 08 00 00 00 00 00 00 00 00 00 00
AB 00 0E FF 92 C0 03 03 FA 00 00 00 00 00 00 10 0B
AB 00 09 FF 51 11 13 0B 17 11 05 3E
AB 00 09 FF 51 11 13 0B 17 11 0A 48
AB 00 09 FF 51 11 13 0B 17 11 0F 3D
AB 00 09 FF 51 11 13 0B 17 11 14 44
AB 00 09 FF 51 11 13 0B 17 11 19 43
AB 00 09 FF 51 11 13 0B 17 11 1E 38
AB 00 09 FF 51 11 13 0B 17 11 23 42
AB 00 09 FF 51 11 13 0B 17 11 28 56
```

Було виділено статичну та динамічну частину відповіді, а саме останні два байти у коротких та усі байти після п’ятого байту у довгій відповіді.

Дослідивши дані з оригінального застосування та маючи інформацію, що дані частоти серцебиття зберігаються кожні п’ять хвилин можна зробити висновок, що короткі команди – це історичні дані серцебиття, які були збережені на пристрої, а довгі команди – це історичні або поточні дані, які складаються з кількості пройдених шагів, пройденої довжини, витрачених калорій. Зіставивши дані з оригінальним застосуванням та фізичним пристроєм можна зробити висновок, що коротка команда кодується з заголовку, дати з місяця та дня та часу з хвилинами. Останній байт – це усереднене

значення серцебиття у даний проміжок часу. Таким чином очевидно, що структура даних для зберігання серцебиття – це дата/час та саме значення, а структура даних для зберігання другорядних даних – це дата/час та значення виконаних шагів, пройденого шляху. Очевидно, що база даних та саме застосування може мати додаткові алгоритми для оптимізації даних (наприклад видаляти другорядні дані при отриманні нових для кожної доби).

Таким же алгоритмом необхідно дослідити та підготовити для проектування усі структури даних, які обертаються у предметній області та спроектувати базу даних та примітивні класи для переходу до наступного етапу розробки, а саме моделювання проекту.

4 МОДЕЛЮВАННЯ ТА РОЗРОБКА ПРОЕКТУ

4.1 Моделювання БД та розробка класів-адаптерів БД

Дослідивши відповіді з приладу можна перейти до проектування бази даних та виділити примітивні типи для виконання внутрішньої комунікації. Було створено 5 таблиць для зберігання даних та ще одна по запиту розробника іншої частини проекту:

– таблиця для зберігання годинників;
– таблиця для зберігання даних серцебиття;
– таблиця для зберігання другорядних даних;
– таблиця для зберігання сесій сну;
– таблиця для зберігання даних сесій сну;
– таблиця для зберігання налаштувань фільтрації повідомлень (запит розробника іншої частини проекту).

Таким чином для створювання усіх необхідних таблиць необхідно використовувати такі команди:

```
CREATE TABLE HRRecords (
ID INTEGER PRIMARY KEY AUTOINCREMENT,
DATE INTEGER UNIQUE,
HRValue INTEGER);

CREATE TABLE MainRecords(
ID INTEGER PRIMARY KEY AUTOINCREMENT,
DATE INTEGER UNIQUE,
STEPS INTEGER,
CALORIES INTEGER);

CREATE TABLE Notify(
ID INTEGER PRIMARY KEY AUTOINCREMENT,
PACKAGE VARCHAR UNIQUE,
ENABLED BOOLEAN);

CREATE TABLE Alarms(
ID INTEGER PRIMARY KEY,
HOUR INTEGER, MINUTE INTEGER,
DAYS INTEGER, ENABLED BOOLEAN,
HOURSTART INTEGER, MINSTART INTEGER,
SYNCABLE BOOLEAN);
```

```
CREATE TABLE SleepSessions(
ID INTEGER PRIMARY KEY AUTOINCREMENT,
TIMESTAMP INTEGER UNIQUE,
IDSS INTEGER,
DURATION INTEGER,
TYPE INTEGER,
FOREIGN KEY (IDSS) REFERENCES SleepRecords (ID));

CREATE TABLE SleepRecords(
ID INTEGER PRIMARY KEY AUTOINCREMENT,
DATE INTEGER UNIQUE,
DURABILITY INTEGER DEFAULT -1,
DEEPDURABILITY INTEGER DEFAULT -1);
```

Фізична структура даних буде представлена на рис. 10.

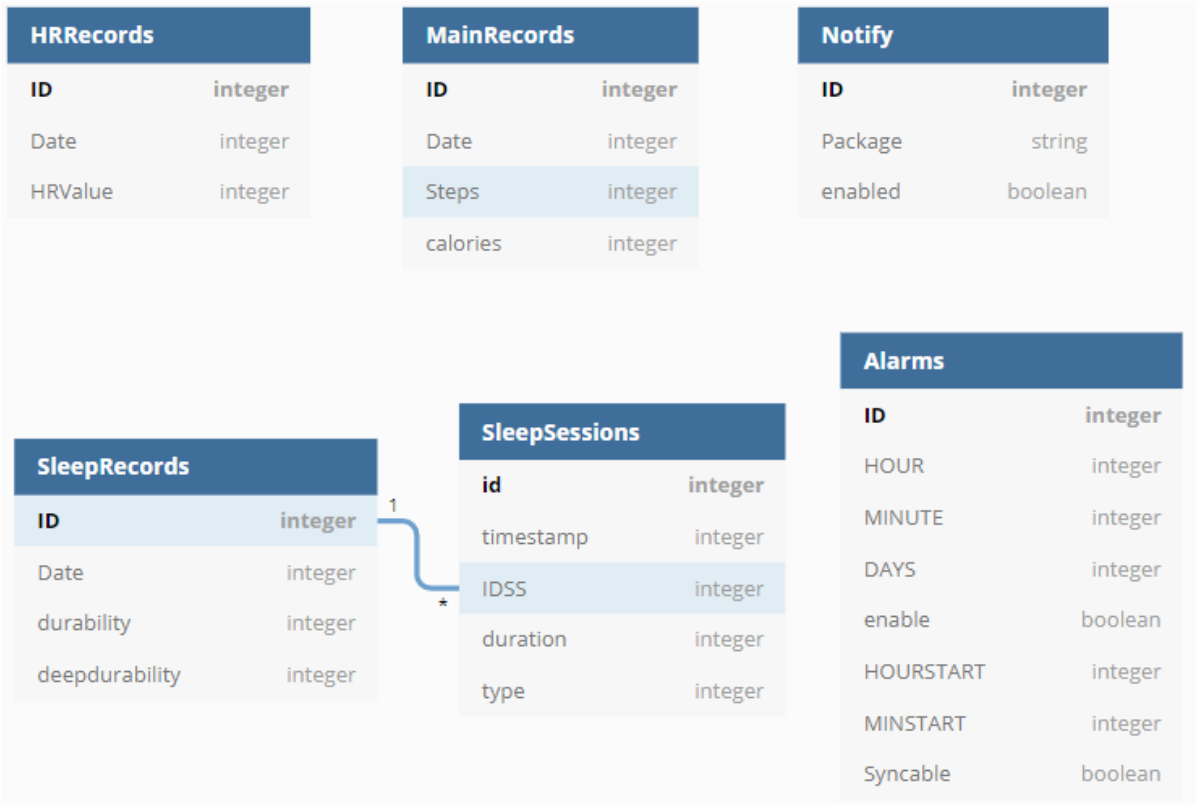


Рисунок 10 – Фізична структура даних

Як зображено на рис. 10 таблиця HRRecords використовується для обробки показників серцебиття та має такі поля:

– ID – ціле число, первинний ключ з авто-інкрементуванням, використовується як ідентифікатор;
– Date – ціле число, закодована дата у форматі YYYYMMDDHHmmss, має обмеження на унікальність;
– HRValue – ціле число, зберігає поточне значення серцебиття у час Date.

Таблиця MainRecords використовується для збереження статистичних даних, а саме кількість кроків, кількість витрачених калорій. Має такі поля:

– ID – ціле число, первинний ключ з авто-інкрементуванням, використовується як ідентифікатор;
– Date – ціле число, закодована дата у форматі YYYYMMDDHHmmss, має обмеження на унікальність;
– Steps – ціле число, зберігає кількість кроків, який передав пристрій у час Date;
– Calories – ціле число, зберігає кількість витрачених калорій у час Date.

Таблиця Notify розроблена для збереження налаштувань фільтрації повідомлень від операційної системи Android. Має такі поля:

– ID – ціле число, первинний ключ з можливістю авто-інкрементуванням, використовується як ідентифікатор;
– Package – рядок змінної довжини, використовується для збереження імені пакету, яке необхідно обробити;
– Enabled – логічна змінна, відповідає за трансляцію повідомлення до смарт-браслету, якщо істина;

Таблиця Alarms зберігає та оброблює список годинників. Має такі поля:

– ID – ціле значення, первинний ключ, використовується як ідентифікатор;
– Hour – ціле число, зберігає значення часу у який обов’язково необхідно запустити годинник;

– Minute – ціле число, зберігає значення хвилини у який обов’язково необхідно запустити годинник;
– HourStart – ціле число, зберігає значення часу у який можна запускати годинник, якщо він відповідає алгоритму запуску годинника;
– MinStart – ціле число, зберігає значення хвилини у який можна запускати годинник, якщо він відповідає алгоритму запуску годинника;
– Days – ціле число, закодована бітова маска, яка відповідає за дні у які працює даний годинник;
– Syncable – логічне значення, відповідає чи необхідно виконувати синхронізацію поточного годинника зі смарт браслетом;
– Enabled – логічне значення, відповідає чи увімкнений поточний годинник.

Таблиця SleepRecords зберігає згруповані дані сну. Групуються вони за одну ітерацію, тобто один сеанс сну. Має такі поля:

– ID – ціле число, первинний ключ з авто-інкрементуванням. Використовується як ідентифікатор;
– Date – ціле число, закодована дата у форматі YYYYMMDD;
– Durability – загальна довжина сну;
– DeepDurability – загальна довжина глибокого сну.
– Таблиця SleepSessions зберігає події сну. Має такі поля:
– ID – ціле число, первинний ключ з авто-інкрементуванням. Використовується як ідентифікатор;
– Timestamp – ціле число, дата початку події, яка закодована у вигляді YYYYMMDDHHmmss;
– IDSS – ціле число, зовнішній ключ, який посилається на поле ID у таблиці SleepRecords;
– Duration – ціле число, тривалість події у хвилинах;
– Type – ціле число, закодований тип події (глибокий сон, швидкий сон, стан пробудження).

Важливо підкреслити, що дані лише вилучаються з пристрою за допомогою команд, але їх достовірність ще досліджується. Мета цього проекту – керування смарт-браслетом, а саме вилучення даних та їх зберігання. Достовірність таких складних даних як статистика сну необхідно дослідити та перевірити.

Для збереження даних типу дата/час використовується ціле численний тип з причини спрощення розробки застосування. Наприклад комбінація рік, місяць, день, година, хвилина, секунда складається вручну та буде набагато легше оперувати цілочисельними типами або іншими примітивними типами, ніж об’єктними. Кожен запис дати/часу має бути унікальним, щоб забезпечити цілісність та унікальність даних. Це єдине обмеження на рівні бази даних, інші перевірки даних відбуваються на програмному рівні.

Наступним етапом було розробка програмного модулю щодо спроектованих даних таблиць. Результатом цієї роботи є класи SleepSessionsTable, SleepRecordsTable, NotifyFilterTable, MainRecordsTable, HRRecordsTable, AlarmsTable, які зберігаються у відповідних файлах їх опис та функціонал буде у відповідних частинах.

Системну функцію виконує клас DatabaseController, який містить інструкції ініціалізації або оновлення фізичної бази у середовищі Android.

4.2 Моделювання алгоритмів

Для успішної розробки проекту необхідно теоретично розробити алгоритми та передати їх розробнику іншої частини. На рис. 11 буде представлена UML-діаграма послідовностей дій.

На даному рисунку усі сутності окрім «програма» вже розроблені або змодельовані. Таким чином розробнику другої частини необхідно розробити дану сутність ураховуючи документацію з частин 4 та 5 даної частини.

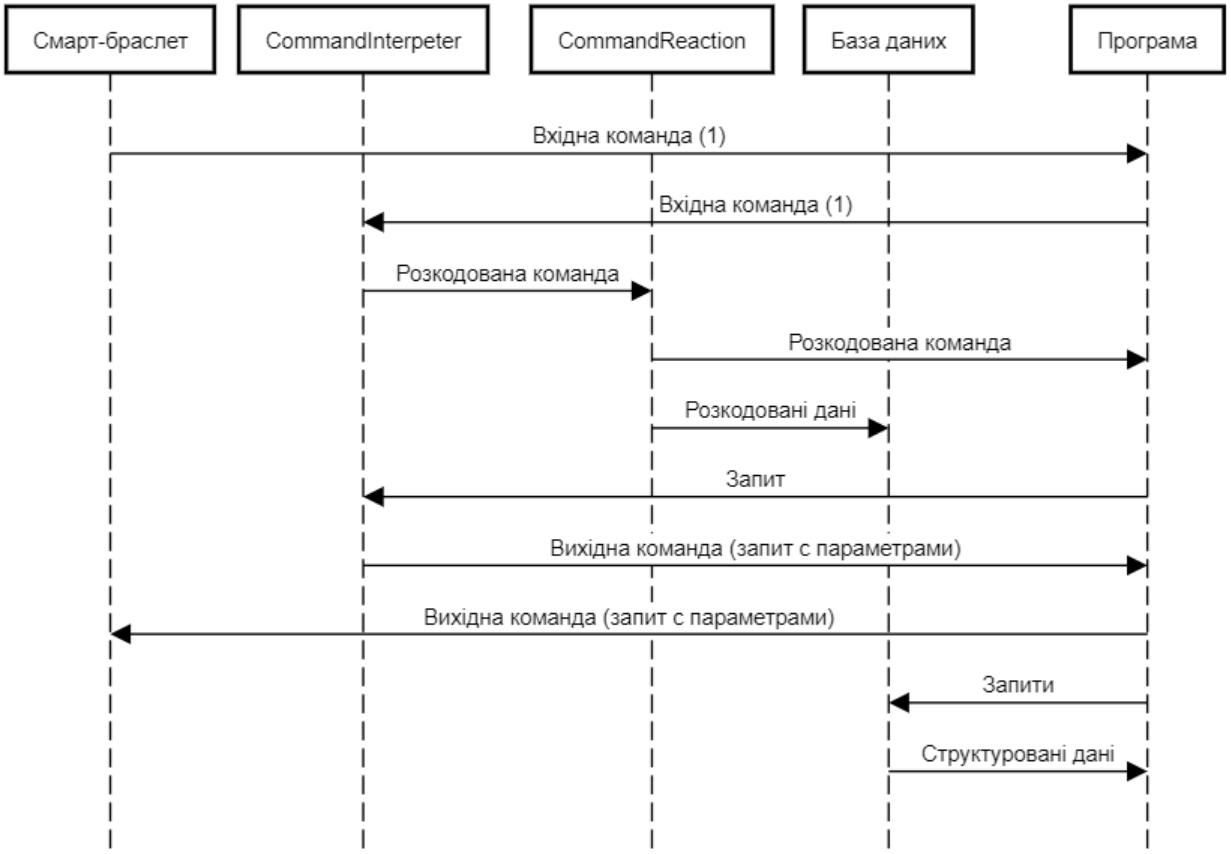


Рисунок 11 – Діаграма послідовності дій у проекті.

Смарт-браслет надсилає команду до програми (технічний прилад смартфон опущено) та програма повинна її обробити за допомогою сутності CommandInterpreter. Далі ця сутність розкодує команду та дані буде передані до інтерфейсу CommandReaction, який буде оброблювати команду згідно з інструкціями сутності «програма». Також реалізація сутності CommandReaction дозволяє додавати розкодовані записи до бази даних безпосередньо без участі сутності «програма».

Програма повинна робити запити до бази даних для того щоб отримувати розкодовані дані або записувати до бази даних налаштування програми.

Програма повинна робити запит до сутності CommandInterpreter, щоб отримати послідовність байт, яку необхідно відправити до смарт-браслету.

Також необхідно урахувати внутрішні повідомлення Android та транслювати їх копію до смарт браслету, а також транслювати дані вхідного дзвінка до смарт-браслету.

Приклад обробки таких сповіщень буде представлено на рис. 12.

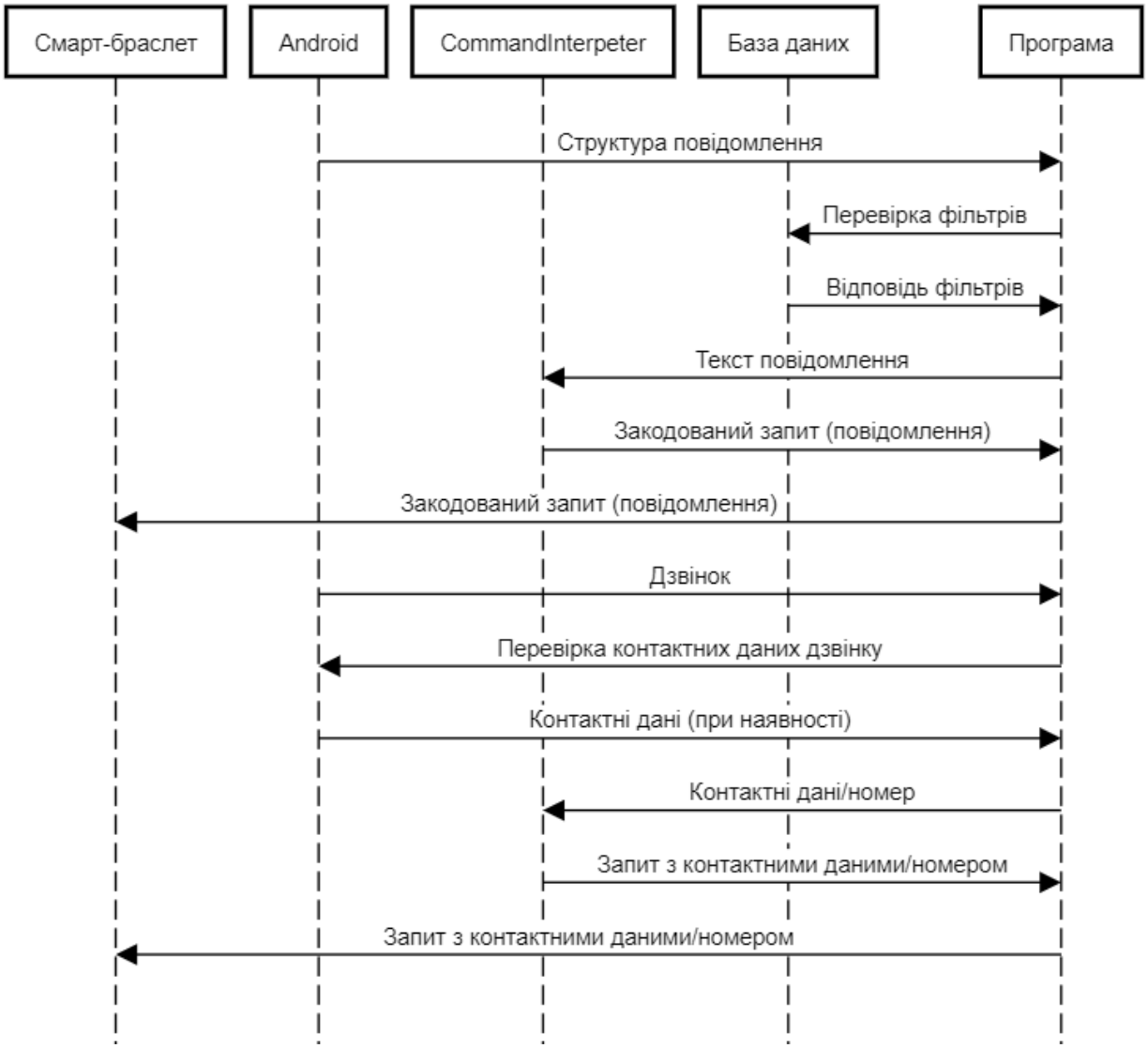


Рисунок 12 – Діаграма послідовності обробки сповіщень.

Операційна система Android має можливість відправляти повідомлення від інших програм, їх можливо оброблювати. Програма повинна використовуючи базу даних перевірити чи необхідно оброблювати повідомлення (таблиця Notify у базі даних, статичний клас NotifyFilterTable, метод

IsEnabled). Перевірка здійснюється за допомогою строки, очевидно що вона повинна бути унікальна та ідентифікувати кожну встановлену програму. При отримані відповіді необхідно використати вже розроблений CommandInterpreter для підготовки команди та після відправити її до смарт-браслету. Для цих повідомлень необхідно використовувати метод BuildNotify класу CommandInterpreter.

Дзвінки теж можуть бути оброблені у середовищі Android. Необхідно з'ясувати усі параметри дзвінку та продублювати їх до смарт-браслету. Для цього необхідно використовувати команду BuildLongNotify з класу CommandInterpreter. Також необхідно використовувати команду StopLongAlarm при скиданні або прийому дзвінку.

5 РЕАЛІЗАЦІЯ ПРОЕКТУ

5.1 Реалізація бази даних

У частиною «Моделювання БД та розробка класів адаптерів БД» було проведено моделювання бази даних. Наступним етапом є розробка інтерфейсу для взаємодії з базою даних. У відповідній частині було розроблено класи SleepSessionsTable, SleepRecordsTable, NotifyFilterTable, MainRecordsTable, HRRecordsTable, AlarmsTable та системний клас DatabaseController, які реалізують увесь необхідний функціонал для взаємодії з фізичною базою даних.

Важливо підкреслити, що на даному етапі розробки описані класи мають обмежену кількість зв’язків та взаємодію з іншими класами. Дані класи повинні бути використані у другій частині цього проекту для надання користувачу можливість взаємодіяти з базою даних. Дані класи розроблені для надання розробнику графічного інтерфейсу можливість отримувати дані з бази. Очевидно, що додавання інформації відбувається безпосередньо до бази при наявності такої можливості.

Також важливо підкреслити, що технічно усі класи, окрім DatabaseController та внутрішніх, спроектовані як статичні, тобто не можуть мати жодного об’єкту. Більшість функцій статичні та виконуються без створення об’єкту класу. DatabaseController виконаний з використанням технології Singleton. Цей об’єкт створюється автоматично при старті програми та має тільки один екземпляр. Даний клас містить посилання на фізичний файл бази даних з якою необхідно оперувати. Інші класи є сутність окремих таблиць та мають функції, які виконують відповідні запити оновлення, видалення, додавання записів.

На рис. 13 зображена UML-діаграма класу DatabaseController та його зв’язків.

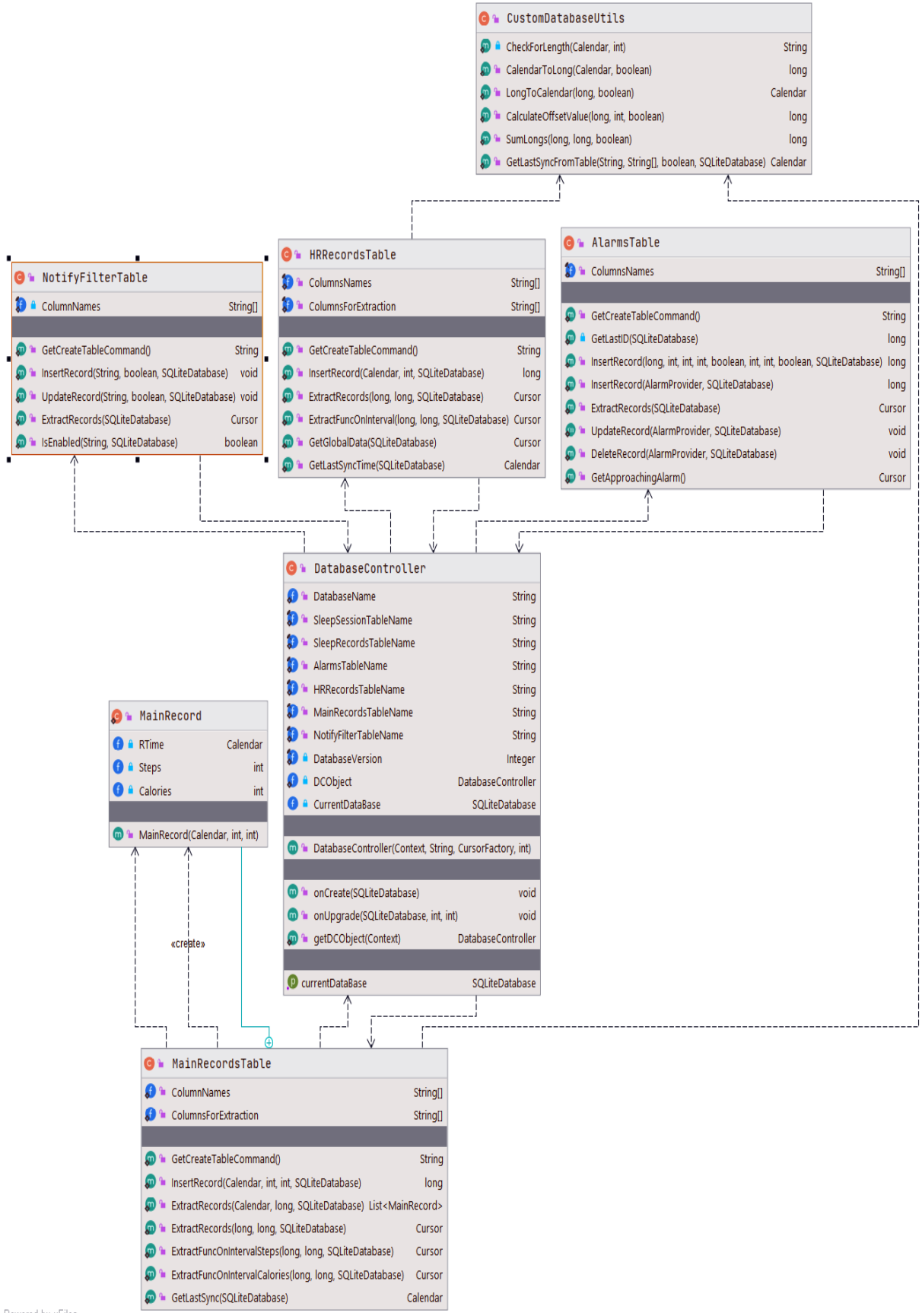


Рисунок 13 – UML-діаграма класу DatabaseController
та його основних зв’язків

Даний клас призначений для надання фізичного доступу до бази даних та її оновлення при необхідності. Даний клас є нащадком класу SQLiteOpenHelper. Даний клас спроектований для другої частини проекту, згідно моделі MVP є Model. Даний клас має такі поля та методи:

- поле DatabaseName є константою та зберігає ім’я бази даних;
- поля SleepSessionTableName, SleepRecordsTableName, AlarmTableName, HRRecordsTableName, NotifyFilterTableName є константами та зберігають імена відповідних таблиць;
- поле CurrentDatabase зберігає посилання на фізичний об’єкт бази даних SQLite;
- поле DatabaseVersion зберігає версію бази даних. Необхідно для оновлення бази даних з методом onUpgrade при необхідності;
- поле DCObject містить посилання на об’єкт класу DatabaseController. Необхідний для виконання технології програмування Singleton;
- конструктор DatabaseController отримує як параметри контекст, назву бази даних, курсор читання та версію БД. Ініціалізує фізичний доступ до бази даних або створює та оновлює її, якщо це необхідно;
- метод onCreate запускається операційною системою після створення класу, як параметр отримує посилання на фізичну базу даних. На практиці створює базу даних;
- метод onUpgrade запускається операційною системою, якщо параметри DatabaseVersion різні. На практиці цей метод не використовується, але повинен бути реалізований згідно з потребою суперкласу;
- метод getDCObject приймає параметром контекст виконання та у результаті повертає посилання на об’єкт поточного класу. Необхідний для надання доступу до запису структурам вищого рівня. На практиці є точкою доступу для запису.

На рис. 14 представлено клас AlarmTable та його основні зв’язків.

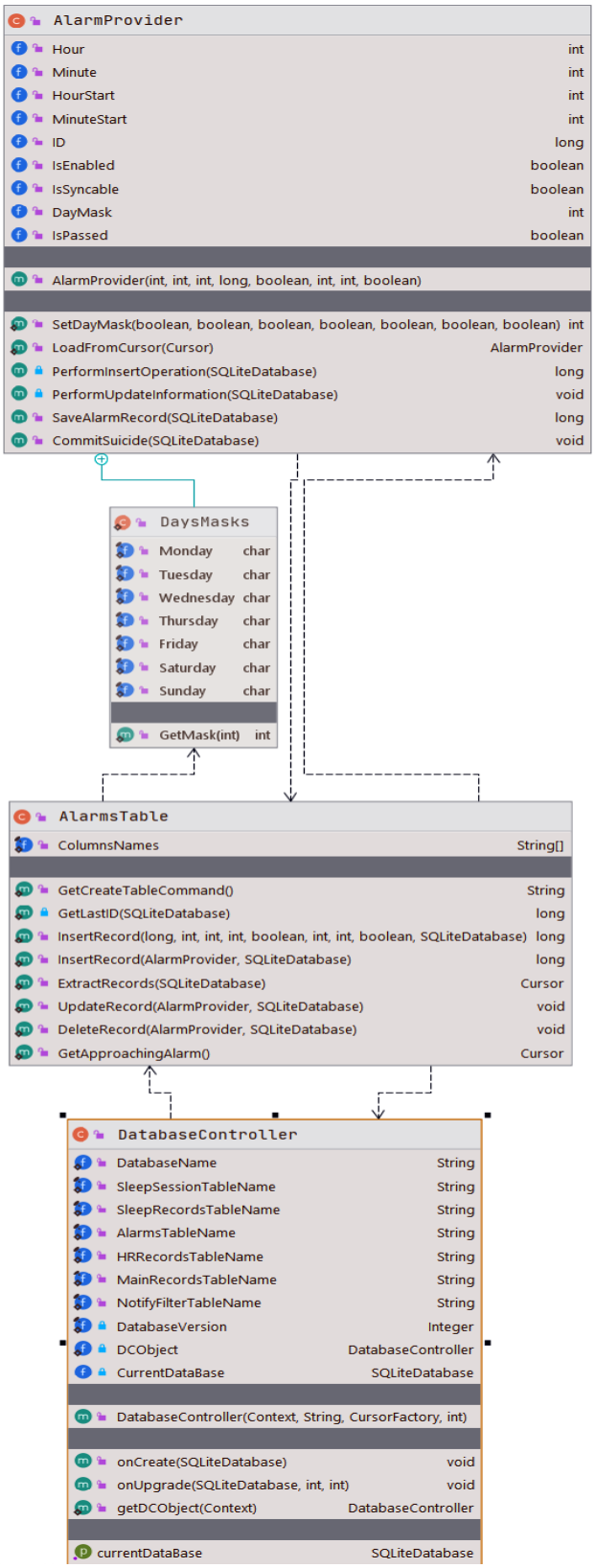


Рисунок 14 – UML-діаграма класу AlarmTable, AlarmProvider та допоміжного класу DatabaseController.

Допоміжний клас DatabaseController передає посилання на фізичну базу даних, яке необхідно у всіх операціях з маніпулюванням даних. Клас AlarmsTable виконує обробку усіх даних, які стосуються годинників. Даний клас розроблений для другої частини проекту, згідно з моделлю MVP є Model.

Клас AlarmProvider є реалізацією окремого запису у таблиці годинників та реалізує необхідні функції годинника, згідно з моделлю MVP даний клас є Presentor`ом. Обидва класи були розроблені для передачі та реалізації у другу частину проекту. Поля класу AlarmProvider описані у пункті 4.2 у відповідному списку. Вбудований клас DayMasks представляє собою сутність бітовою маски, тобто зберігає відповідне цілочисельне значення, яке відноситься для кожного дню тижня. Клас AlarmsTable має такі поля та методи:

– поле ColumnNames є константою, та зберігає масив назв стовпців відповідної таблиці;
– метод GetLastID приймає параметром фізичну базу даних та повертає значення ідентифікатору останнього створеного годинника;
– метод GetCreateTableCommand повертає строку, яка необхідна для створення відповідної таблиці;
– методи InsertRecord приймають параметри окремо або самою сутністю AlarmProvider та виконує додавання годиннику або його оновлення, якщо такий вже існує;
– метод ExtractRecord вилучає усі записи годинників с таблиці;
– метод UpdateRecord виконує оновлення запису годинника;
– метод DeleteRecord виконує видалення запису годинника;
– метод GetApproachingAlarm повертає запис годинника, який ближче усього до запуску.

Клас AlarmProvider має такі поля та методи:

– поле DayMasks містить значення для побудови бітової маски, які зберігають дні неділі у яких необхідно запускати годинник;

- поле ID, StartHours, Hours, StartMinutes, Minutes IsEnabled, IsSyncable зберігають дані, які описані у пункті 4.2 у відповідному списку;
- поле IsPassed зберігає значення чи було спрацювання годинника;
- метод PerformInsertOperation виконує зберігання нового годинника у базу даних за допомогою методів InsertRecord у класі AlarmsTable;
- метод PerformUpdateInformation виконує збереження змін внесених до годинника за допомогою методів UpdateRecord у класі AlarmsTable;
- метод SaveAlarmRecord виконує збереження годинника, а саме обирає необхідно створити новий запис чи існуючий. В залежності від алгоритму буде обрано методи PerformUpdateInformation або PerformInsertOperation;
- метод CommitSuicide виконує видалення годинника з бази даних за допомогою методу DeleteRecord у класі AlarmsTable;
- метод IsAlarmsEqual виконує порівняння годинників. Необхідний для перевірок перед плануванням;
- метод SetDayMask повертає значення бітової маски відповідно до обраних днів;
- метод LoadFromCursor надає взаємодію для ініціалізації об’єкту класу AlarmProvider, який отримується за допомогою методу ExtractRecords у класі AlarmsTable. Необхідний для взаємодії між частинами Model та Presenter даного проекту.

Для збереження налаштувань повідомлень та сповіщень, необхідно створити таблицю, яка буде містити перелік пакетів, повідомлення яких необхідно транслювати на смарт браслет. UML-діаграма цього класу представлена на рис. 15.

Дана таблиця зберігає лише назву пакета, яка є унікальною у системі Android та не може повторюватись. Інші дані, наприклад назву застосування або його іконку можна отримати за допомогою запиту до операційної системи з назвою пакета відповідного застосування як параметр запиту.

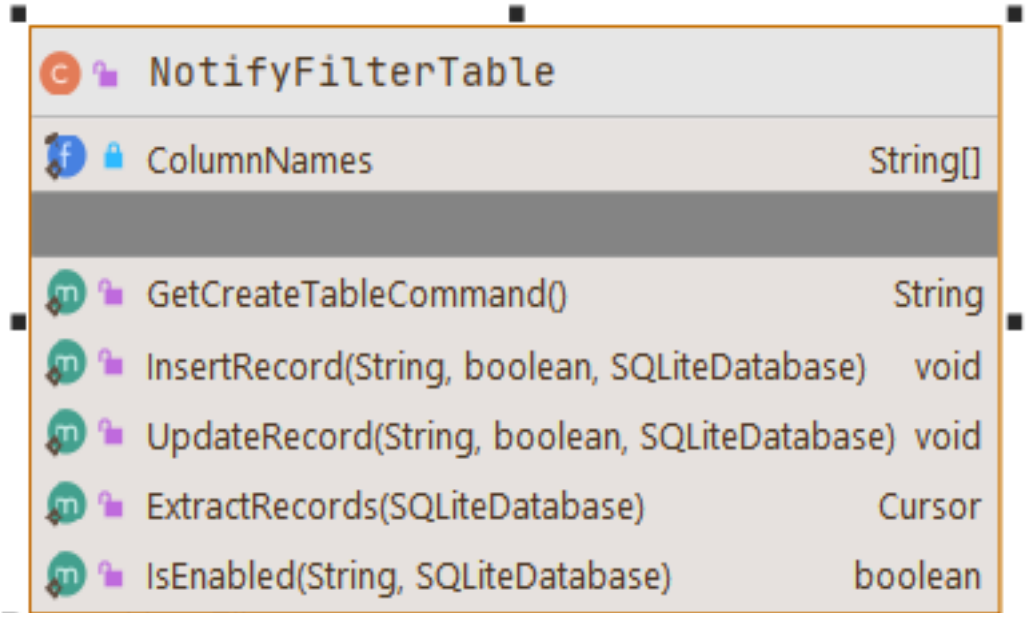


Рисунок 15 – UML-діаграма таблиці з фільтрами

Даний клас має такі поля та методи:

– метод GetCreateTableCommand виконує аналогічну функцію як у попередніх класах;
– методи InsertRecord та UpdateRecord виконують додавання або оновлення налаштування у таблицю відповідно;
– метод ExtractRecords вилучає увесь список та передає структуру даних;
– метод IsEnabled перевіряє, чи необхідно оброблювати пакет, який було передано у параметрі.

Наступним кроком було спроектовано класи HRRecordsTable, MainRecordsTable та їх основні зв’язки. Відмінність від попередніх структур та типів даних є те, що ці дані можуть бути записані безпосередньо до бази даних без жодних посередників.

Клас HRRecordsTable має внутрішній клас HRRecord, який відображає сутність одного запису показників серцебиття, а саме дату та саме значення. Клас MainRecordsTable має внутрішній клас MainRecord, який відображає

сутність одного запису статистичного запису, а саме дату, кількість кроків та витрачених калорій.

Клас HRRecordsTable містить такі поля та методи:

- поле ColumnsForExtraction – це константа, яка містить масив імен стовпців таблиці, які необхідні для вилучення даних;
- поле ColumnsNames – це константа, яка містить масив імен стовпців відповідної таблиці;
- метод IsAlreadyExists приймає як параметр дату та перевіряє чи існує запис з такою датою;
- метод GetCreateTableCommand повертає строку, яка необхідна для створення відповідної таблиці;
- метод InsertRecord приймає як параметр дату та значення серцебиття та додає запис;
- метод ExtractRecords приймає як параметр інтервал дат та повертає структуру з цими записами;
- метод ExtractFuncOnInterval повертає мінімальне, середнє та максимальне значення показнику серцебиття на вказаному інтервалі;
- метод GetGlobalData повертає мінімальне, середнє та максимальні значення за увесь період;
- метод GetLastSyncTime повертає дату останнього запису.

Клас MainRecordsTable має аналогічні поля як і у таблиці HRRecordsTable, але має відмінності у методах:

- методи IsAlreadyExists, GetCreateTableCommand, InsertRecord, GetLastSync, виконує аналогічні функції як у попередніх класах;
- методи ExtractRecords приймають значення дати та повертає структуру з даними або згенерований список;
- методи ExtractFuncOnIntervalCalories та ExtractFuncOnIntervalSteps повертає мінімальне, середнє та максимальне значення витрачених калорій чи кроків на інтервалі відповідно.

UML-діаграма цих класів та всіх їх зв’язків булла наведена на рис. 11.

Наступною комбінацією даних, які можна безпосередньо додавати до бази даних – це статистика сну. Важливо наголосити, що ці дані лише зчитуються з браслету, але достовірність ще досліджується, тому у проекті статистика сну розглядається обмежено.

Класи SleepSessionsTable та SleepRecordsTable мають внутрішні класи SleepRecord та SleepSession, які відображають окремий запис у цих таблицях відповідно. Об’єкт класу SleepRecord зберігає довжину події сну, час початку події та тип події. Об’єкт класу SleepRecordSession зберігає дані тривалості усієї сесії сну та триваліть довжини глибокої фази сну, також існує метод, який автоматично після створення об’єкту класу розраховує довжину фази легкого сну.

Класи SleepRecordsTable та SleepSessionsTable мають такі поля та методи:

– поле ColumnNames у цих класах мають аналогічну функцію, як у попередніх;
– методи InsertRecord та GetCreateTableCommand виконують аналогічні функції як у попередніх класах;
– методи ExtractRecord та ExtractValues повертає масив об’єктів класів SleepRecord або SleepSession відповідно у заданому інтервалі;
– метод GetIDSleepSession у класі SleepRecordTable приймає параметром дату та повертає ідентифікатор сесії сну, який пов’язаний з цією датою;
– метод GetLastSyncTime у класі SleepSessionsTable повертає дату останнього запису у таблиці.

UML-діаграма цих класів та їх основних зв’язків представлена на рис. 16.

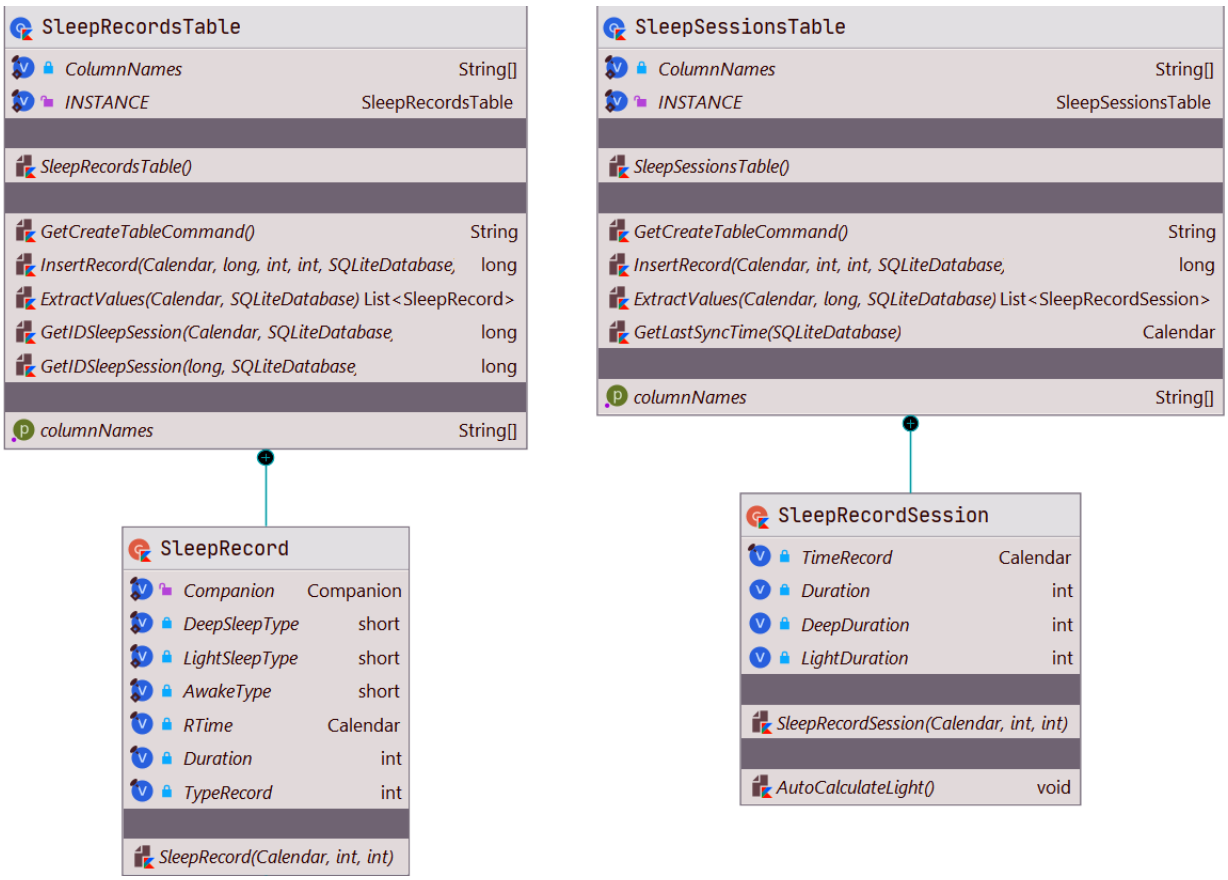


Рисунок 16 – UML-діаграма класів бази даних, які обслуговують статистику сну та їх основні зв’язки.

5.2 Реалізація інтерфейсів зв’язку та алгоритмів

У даному розділі будуть описані класи та програмні елементи, які відповідають за програмну логіку та за інтерфейс зв’язку програми. Було розроблено клас CommandInterpreter та інтерфейс CommandReaction. Важливо наголосити, що клас CommandInterpreter спроектований статичним та усі його методи теж статичні. Інтерфейс CommandReaction створений для інших розробників, щоб вони мали можливість вручну прописувати реакцію на події.

Таким чином інтерфейс CommandReaction має такі методи, які необхідно реалізувати:

– метод MainInfo повертає поточне значення витрачених калорій та кількість кроків;
– метод BatteryInfo повертає поточний рівень заряду акумулятора смарт-браслету;
– метод HRIncome повертає поточне розшифроване значення серцебиття;
– метод HRHistoryRecord повертає значення даних серцебиття з прив’язкою до часу у минулому, які були збережені локально на браслеті;
– метод MainHistoryRecord повертає значення даних витрачених калорій чи пройдених кроків з прив’язкою до часу у минулому, які були збережені локально на браслеті;
– метод SleepHistoryRecord повертає значення подій сну, які були збережені локально на браслеті.

Кожен розробник може реалізувати свій інтерфейс та оброблювати події як потребує логіка програмного забезпечення. Через цей інтерфейс записи потрапляють у базу даних безпосередньо.

Клас CommandInterpreter має такі поля:

– об’єкт WeekIDs зберігає константи для спрощення побудови бітової маски;
– поле Callback зберігає посилання на об’єкт класу з реалізованим інтерфейсом CommandRection;
– поле GetMainInfo зберігає закодовану HEX строку для виконування запиту інформації щодо стану акумулятора, кількості пройдених кроків на даний момент та кількість витрачених калорій;
– поле HRRealTimeHeader зберігає закодовану HEX строку увімкнення чи вимкнення передачу показників серцебиття у режимі реального часу;

– поле HRHistoryHeader зберігає закодовану HEX строку для побудови запиту даних показників серцебиття з локального бази даних смарт-браслету;
– поле TimeSyncHeader зберігає закодовану HEX строку для побудови команди, яка встановить нове значення часу та дати на смарт-браслеті;
– поле SleepHistoryHeader зберігає закодовану HEX строку для побудови запиту даних історії сну з локальної бази даних смарт-браслету;
– поле RestoreCommandHeader зберігає закодовану HEX строку команди, яка виконує скидання налаштувань браслету;
– поле EraseDataHeader зберігає закодовану HEX строку команди, яке повністю видаляє дані з локальної бази даних смарт-браслету;
– поле FindCommand зберігає закодовану HEX строку розпізнавання команди пошуку смартфону;
– поле GyroActionCommandHeader зберігає закодовану HEX строку для побудови команди, яке вимикає чи вмикає налаштування, яке реагує на повертання смарт-браслету у площині. Наприклад браслет буде автоматично вмикати дисплей, якщо його положення було різко змінено;
– поле AlarmHeader зберігає закодовану HEX строку для програмування годинників на самому браслеті;
– поле LongMessageHeaderPartOne зберігає закодовану HEX строку для побудови повідомлення, яке надсилається до смарт-браслету. Необхідно передавати у парі з другою частини цієї команди;
– поле LongMessageHeaderPartTwo зберігає закодовану HEX строку для побудови повідомлення, яке надсилається до смарт-браслету;
– поле ShortMessageHeader зберігає закодовану HEX строку для побудови короткого повідомлення, яке надсилається до смарт-браслету;

– поле StopLongAlarmHeader зберігає закодовану HEX строку яке містить команду для зупинки годиннику на смарт-браслеті або для зупинки довгого повідомлення.

Майже усі поля є константами та зберігають строки. Важливо наголосити, що усі методи не виконують безпосередньої відправки команди до смарт браслету. Даний клас виконує перетворення між типами даними та повертає масив байтів, які необхідно відправити у сесію Bluetooth з'єднання.

Даний клас має такі методи:

– метод hexStringToByteArray переводить закодовану HEX строку у масив байтів, який необхідно відправити до смарт-браслету;
– метод GetCalendarValueInHex приймає параметром об'єкт календарю та індекс частини дати, яке необхідно вилучити та перевести HEX строку;
– метод DayStringToHex переводить строку з переліком днів тижня до HEX закодованої строки;
– метод CommandAction є вхідною точкою для кожної команди. Даний метод перевіряє у який метод необхідно подальше передати управління. Перевірка відбувається за допомогою перевірки необхідного тегу у вхідному масиві;
– метод CommandID14 перевіряє усі команди с тегом 14 та якщо вони відповідають перевіркам, то ділить вхідні дані даної команди на кількість поточних кроків та витрачених калорій. Після усіх маніпуляцій передає дані у відповідний метод інтерфейсу CommandReaction, а саме MainInfo;
– метод HRRTHandler перевіряє та оброблює усі команди, які стосується даних серцебиття у реальному часі. Викликає метод HRIncome у інтерфейсі CommandReaction;
– метод HRHistoryHandler оброблює усі повідомлення історичними статистичними дані, які пов'язані з серцебиттям;

– метод MainHistoryHandler оброблює усі повідомлення з історичними статистичними даними, які пов’язані з основними даними (кількість пройдених кроків, кількість витрачених калорій);
– метод BatteryCommandHandler оброблює повідомлення з даними поточного статусу акумулятора смарт-браслету;
– метод BuildNotify приймає параметром строку, яку необхідно передати до смарт-браслету коротким повідомленням. Максимальна довжина повідомлення – 34 байти (17 символів). Використовує константу ShortMessageHeader для відправки команди;
– метод BuildLongNotify приймає параметром строку, яку необхідно відправити до смарт-браслету. Підтримує повідомлення розміром до 56 байтів (28 символів). Використовує константи LongMessageHeaderPartOne та LongMessageHeaderPartTwo;
– метод SetGyroAction приймає параметром чи увімкнути дисплей при подіях акселерометру смарт-браслету. Використовує константи GyroActionCommandHeader;
– метод StopLongAlarm підготовлює команду, яка зупиняє довге повідомлення або годинник. Використовує константу StopLongAlarmHeader;
– метод SetAlarm підготовлює команду для програмування годиннику на смарт-браслеті. Використовує константу AlarmHeader;
– метод RestoreToDefaults підготовлює команду скидання пристрою до стандартний налаштувань. Використовує константу RestoreCommandHeader;
– метод EraseDatabase підготовлює команду для видалення локальної бази даних на смарт-браслеті. Використовує константу EraseDataHeader;
– метод SyncTime підготовлює команду, яка встановлює налаштування дати та часу на смарт-браслеті. Використовує константу TimeSyncHeader;

– метод RequestHRHistory приймає параметром дату та час починаючи від якої необхідно виконати вивантаження з смарт-браслету. Використовує константу HRHistoryHeader;
– метод GetMainInfoRequest підготовлює команду для вивантаження значень кількості виконаних кроків та витрачених калорій. Використовує константу GetMainInfo;
– метод HRRealTimeControl підготовлює команду, яка вмикає або вимикає передачу значень серцебиття у режимі реального часу. Використовує константу HRRealTimeHeader;
– метод SubIntegerConversionCheck є допоміжною функцією. Використовується при побудові HEX строк.

UML-діаграма цих класів та їх зв’язків представлена на рис. 17.

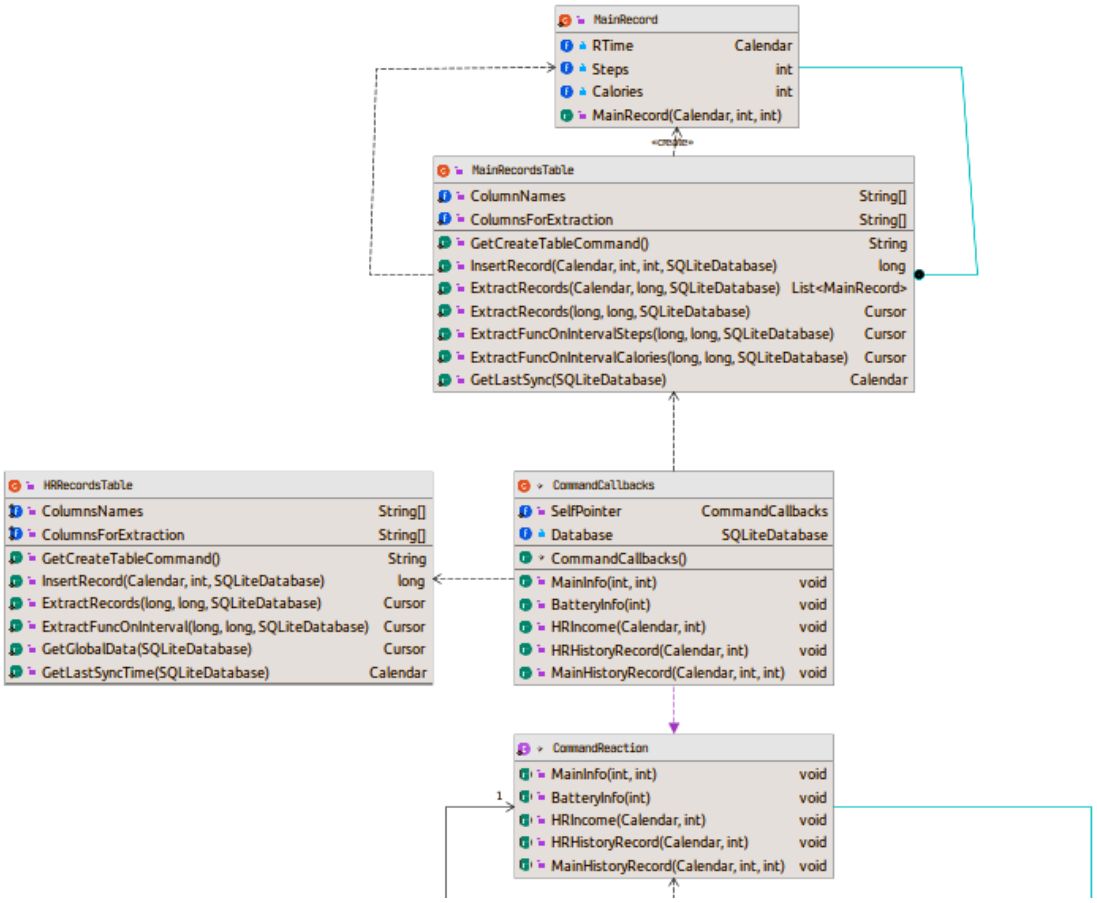


Рисунок 17 – UML-діаграма класів MainRecordsTable, HRRecordsTable та їх основних зв’язків.

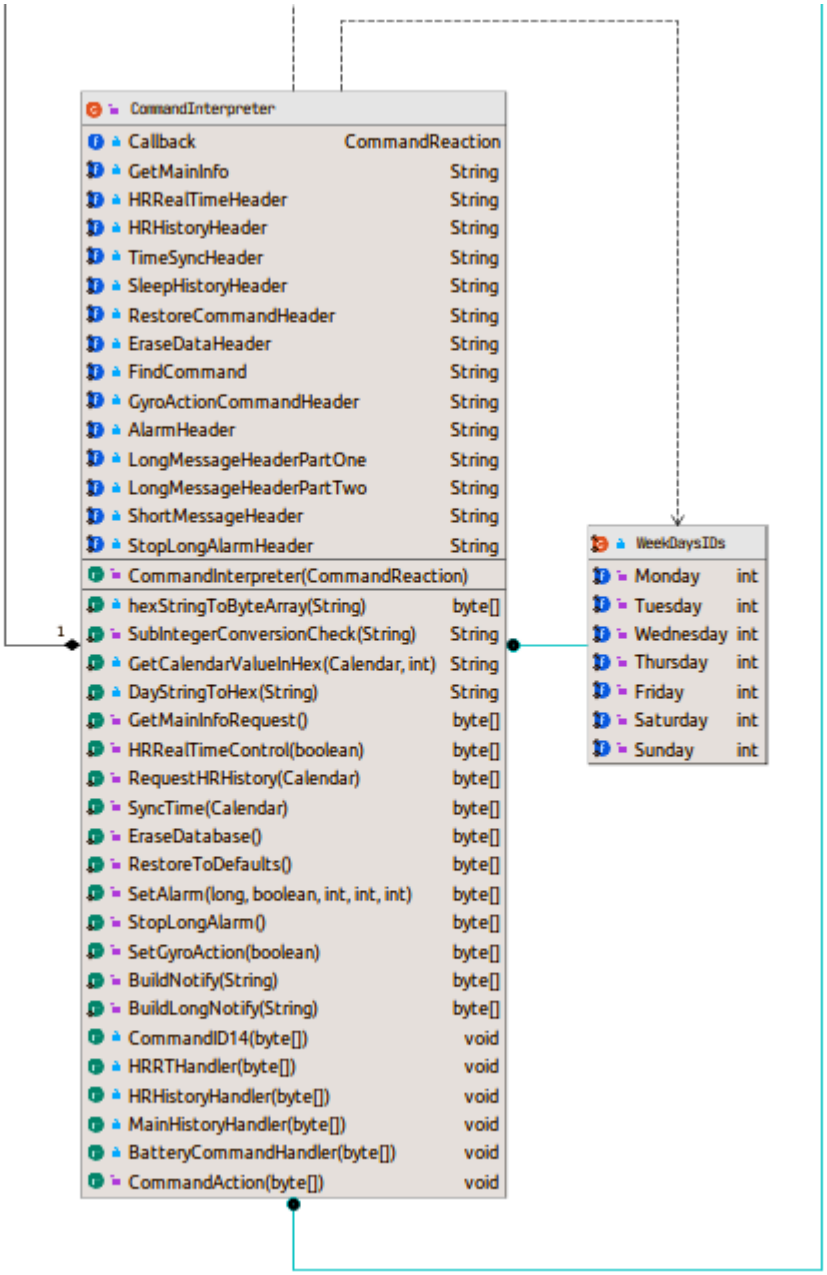


Рисунок 17, аркуш 2

ВИСНОВКИ

В ході роботи було проведено аналіз предметної області, сплановано та розділено проект на дві незалежні частини. Було обрано актуальні технології та інструменти для розробки та реалізації програмного забезпечення.

Було досліджено та вивчено протоколи передачі даних Bluetooth, проаналізовано та розкодовано ключові послідовності структур даних, які передаються між смарт-браслетом та смартфоном.

Було спроектовано усі необхідні роботи та первинні алгоритми, які мають бути реалізовані для подальшого розвитку проекту.

Існують шляхи розвитку даної частини так і усього проекту у цілому. Наприклад, існує можливість розробити алгоритми, які базуючись на великому обсязі даних серцебиття зможуть аналізувати аномалії. Також існує можливість проаналізувати достовірність даних щодо фаз сну та розробити алгоритми для годинників, які будуть ураховувати фазу сну користувача.

Даний проект сплановано модулями, що робить його гнучким. Таким чином можна використовувати інші джерела даних починаючи від других смарт браслетів так і спеціально сконструйованими приладами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Застосування у Google Play – WearFit2.0. URL: https://play.google.com/store/apps/details?id=com.wakeup.wearfit2 (дата звернення 01.02.2020).
2. Mgcool band 4: огляд та технічні характеристики фітнес браслету. URL: http://bazaroved.ru/harakteristiki-i-polnyj-obzor-fitnes-brasleta-mgcool-band-4/ (дата звернення 04.02.2020)
3. Packet capture – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Wireshark (дата звернення 04.02.2020)
4. Юбілейний випуск Intercepter-NG 1.0. URL: https://habr.com/ru/post/309406/ (дата звернення 04.02.2020)
5. SmartSniff: Packet Sniffer - Capture TCP/IP packets on your network adapter. URL: https://www.nirsoft.net/utils/smsniff.html (дата звернення 04.02.2020)
6. York::Log all network traffic - the sz development. URL: https://www.the-sz.com/products/york/ (дата звернення 04.02.2020)
7. Wireshark – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Wireshark (дата звернення 04.02.2020)
8. Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель. SQL: повне-руководство, 3-е виданя SQL: The Complete Reference, Third Edition. М.: «Вильямс», 2014,960 с. ISBN 978-5-8459-1654-9
9. Мартин Фаулер, Прамодкумар Дж. Садаладж. NoSQL: нова методологія розробки нереляційних баз даних NoSQL Distilled. М.: «Вильямс», 2013. 192 с. ISBN 978-5-8459-1829-1
10. Microsoft SQL Server – Вікіпедія. URL:https://ru.wikipedia.org/wiki/Microsoft_SQL_Server (дата звернення 09.02.2020).
11. Oracle Database – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Oracle_Database (дата звернення 09.02.2020).
12. MySQL – Вікіпедія. URL: https://ru.wikipedia.org/wiki/MySQL (дата звернення 09.02.2020).

13. Apache Derby – Вікіпедія. URL:https://ru.wikipedia.org/wiki/Apache_Derby (дата звернення 09.02.2020).
14. SQLite – Вікіпедія. URL: https://ru.wikipedia.org/wiki/SQLite (дата звернення 09.02.2020).
15. Xamarin | Open-source mobile app platform for .NET. URL: https://dotnet.microsoft.com/apps/xamarin (дата звернення 13.02.2020).
16. The Platform for Open Innovation and Collaboration | The Eclipse Foundation. URL: https://www.eclipse.org (дата звернення 13.02.2020).
17. Qt | Cross-platform software development for embedded & desktop. URL: https://www.qt.io (дата звернення 13.02.2020).
18. Download Android Studio and SDK tools | Android Developers. URL: https://developer.android.com/studioc (дата звернення 13.02.2020).
19. Застосування в Google Play – BLE Tool. URL: https://play.google.com/store/apps/details?id=com.lapis_semi.bleapp (дата звернення 15.02.2020)
20. Git – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Git (дата звернення 15.02.2020)
21. Github – Вікіпедія. URL: https://ru.wikipedia.org/wiki/GitHub (дата звернення 15.02.2020)
22. Azure DevOps Server – Вікіпедія. URL: https://ru.wikipedia.org/wiki/Azure_DevOps_Server (дата звернення 15.02.2020)
23. Team Foundation Server – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Team_Foundation_Server (дата звернення 15.02.2020)

Додаток А

UML-діаграми основних класів та їх зв'язків

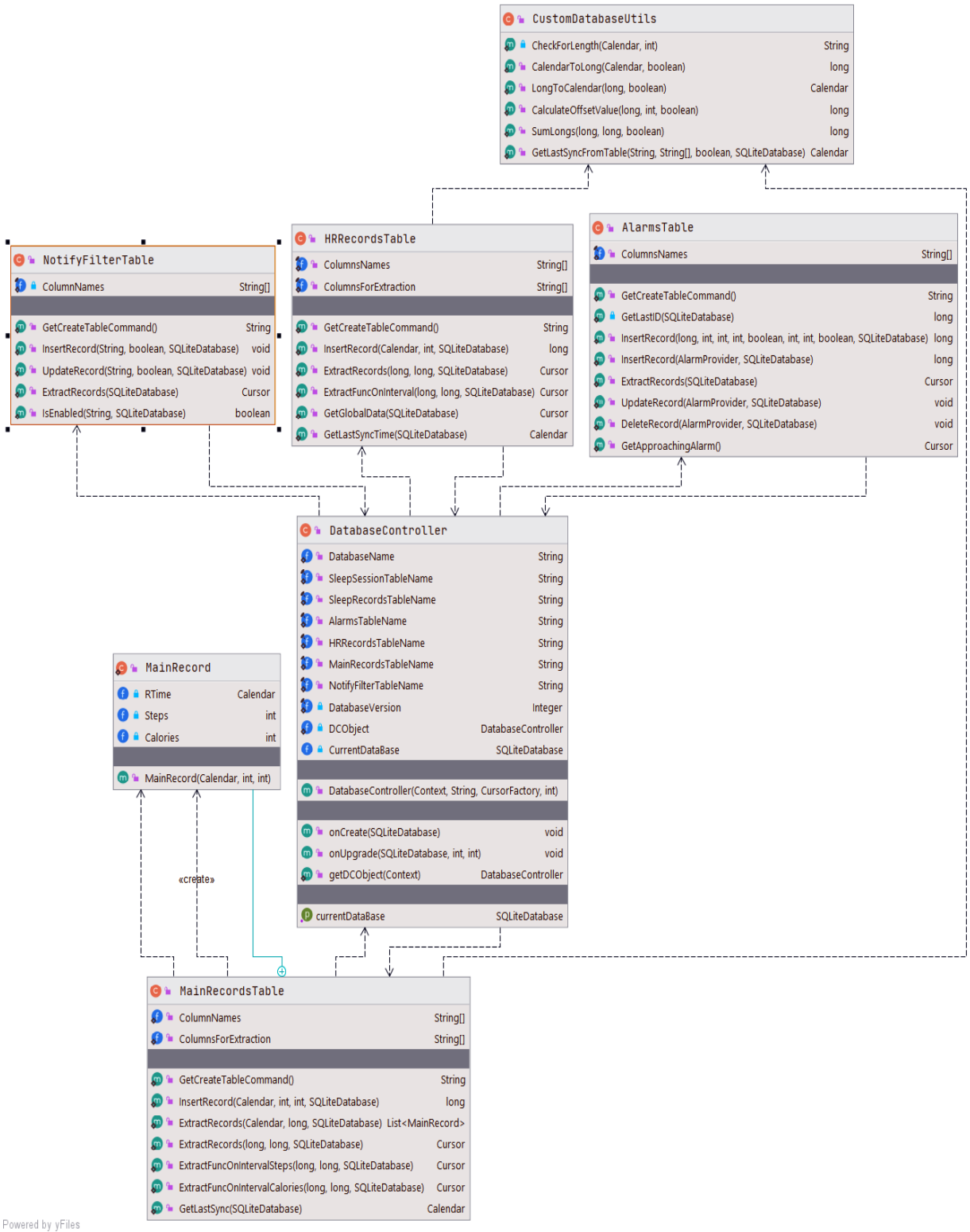


Рисунок А.1 – UML-діаграма класу DatabaseController

та його основних зв’язків

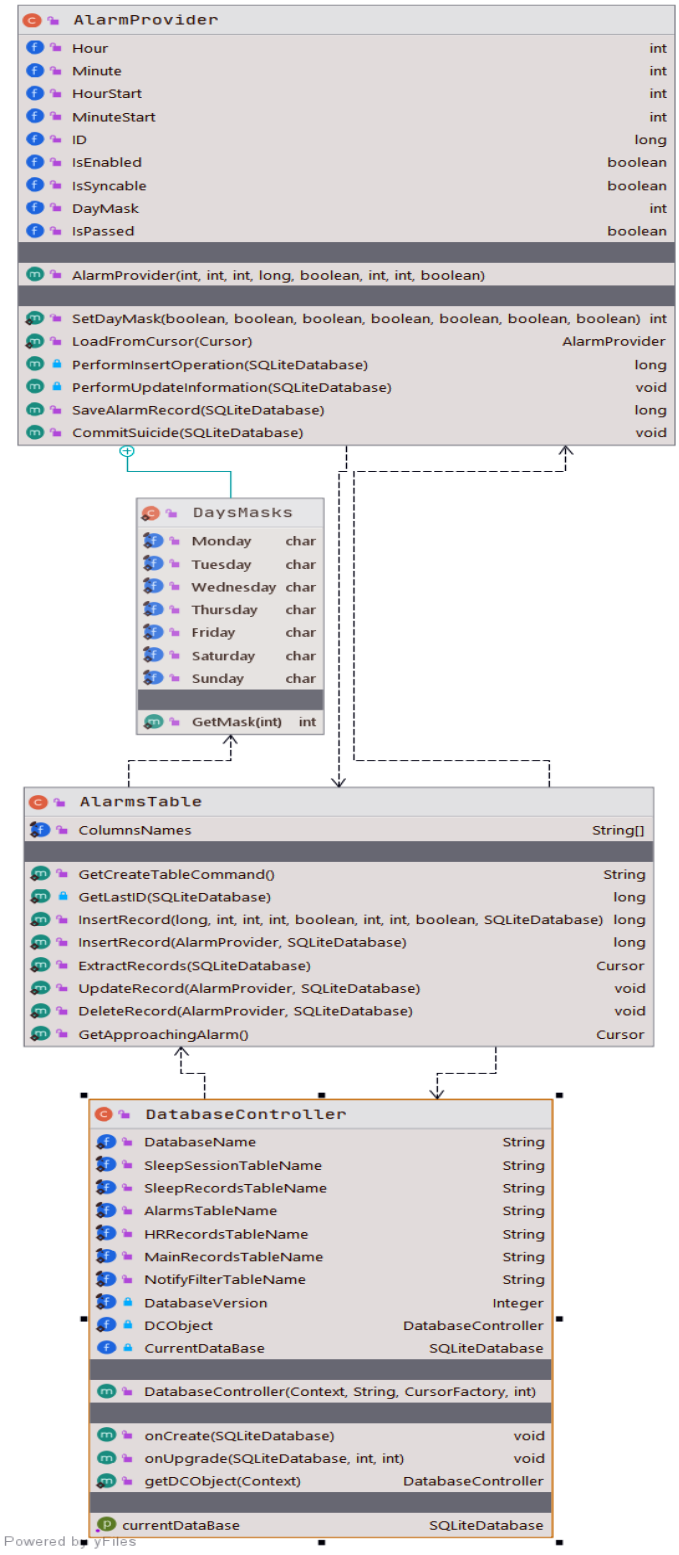


Рисунок А.2 – UML-діаграма класу AlarmTable, AlarmProvider та допоміжного класу DatabaseController.

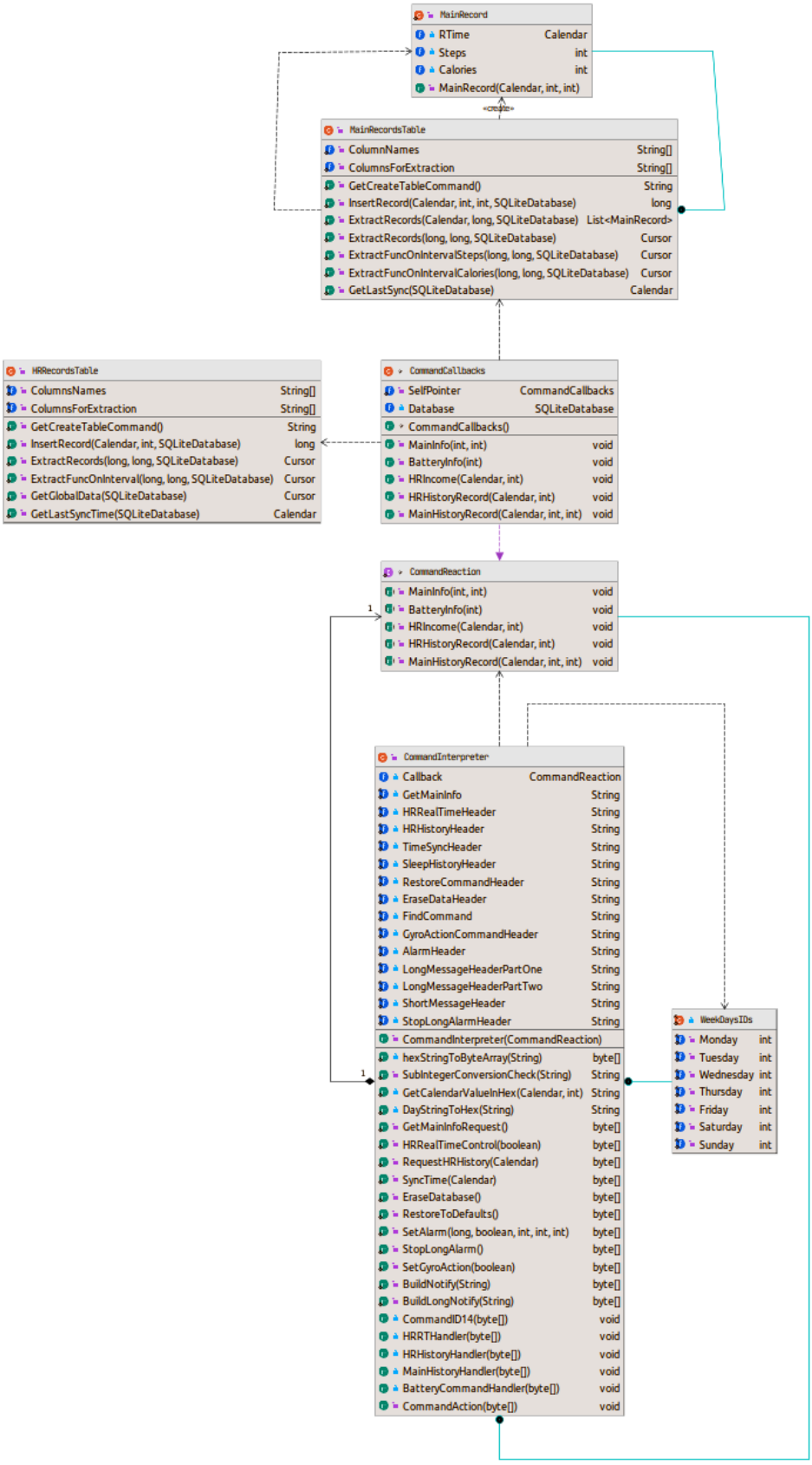


Рисунок А.3 – UML-діаграма класів MainRecordsTable, HRRecordsTable та їх основних зв’язків.

Додаток Б

Лістинг коду командного модуля

```
package anonymouls.dev.MGCEX.App

import java.nio.ByteBuffer
import java.util.Calendar
import kotlin.experimental.and

class CommandInterpreter() {

    private object WeekDaysIDs {
        val Monday = 1
        val Tuesday = 2
        val Wednesday = 4
        val Thursday = 8
        val Friday = 16
        val Saturday = 32
        val Sunday = 64
    }

    interface CommandReaction {
        fun MainInfo(Steps: Int, Calories: Int)
        fun BatteryInfo(Charge: Int)
        fun HRIncome(Time: Calendar, HRValue: Int)
        fun HRHistoryRecord(Time: Calendar, HRValue: Int)
        fun MainHistoryRecord(Time: Calendar, Steps: Int,
Calories: Int)
        //void SleepHistoryRecord(Calendar Time, int Duration,
int Type);
    }

    companion object {

        var Callback: CommandReaction? = null

        private const val GetMainInfo = "AB000EFF51800012"
        private const val HRRealTimeHeader = "AB0004FF8480"
        private const val HRHistoryHeader = "AB000EFF518000"
        private const val TimeSyncHeader = "AB000BFF938000"
        private const val SleepHistoryHeader =
"AB0009FF52800012"
        private const val RestoreCommandHeader = "AB0003FFFF80"
        private const val EraseDataHeader = "AB0004FF238000"
        private const val FindCommand = "AB0003FF7180"
        private const val GyroActionCommandHeader =
"AB0004FF7780"
        private const val AlarmHeader = "AB0008FF7380"
        private const val LongMessageHeaderPartOne = "AB00"
        private const val LongMessageHeaderPartTwo =
```

```
"FF72800102"
        private const val ShortMessageHeader =
"AB0029FF72800302"
        private const val StopLongAlarmHeader =
"AB0005FF72800202"

        private fun hexStringToByteArray(s: String): ByteArray {
            var s = s
            s = s.toUpperCase()
            val len = s.length
            val data = ByteArray(len / 2)
            var i = 0
            while (i < len) {
                data[i / 2] = ((Character.digit(s[i], 16) shl 4)
+ Character.digit(s[i + 1], 16)).toByte()
                i += 2
            }
            return data
        }
        fun SubIntegerConversionCheck(CheckIn: String): String {
            return if (CheckIn.length != 2) {
                "0" + CheckIn.toUpperCase()
            } else
                CheckIn
        }
        private fun GetCalendarValueInHex(CUtil: Calendar,
CField: Int): String {
            var Value: Int? = CUtil.get(CField)
            if (CField == Calendar.MONTH) Value =
Value?.plus(1);
            return if (Value!! < 10) {
                "0" + Value!!
            } else {

SubIntegerConversionCheck(Integer.toHexString(Value!!))
            }
        }
        private fun DayStringToHex(Input: String): String {
            var Input = Input
            Input = Input.toUpperCase()
            val InputStr = StringBuilder(Input)
            InputStr.deleteCharAt(Input.length - 1)
            val ChoosedDays =
Input.split("|".toRegex()).dropLastWhile({ it.isEmpty() }).toTyp
edArray()
            var Output = 0
            for (i in ChoosedDays.indices) {
                when (ChoosedDays[i]) {
                    "MONDAY" -> Output += WeekDaysIDs.Monday
                    "TUESDAY" -> Output += WeekDaysIDs.Tuesday
                    "WEDNESDAY" -> Output +=
WeekDaysIDs.Wednesday
```

```
                    "THURSDAY" -> Output += WeekDaysIDs.Thursday
                    "FRIDAY" -> Output += WeekDaysIDs.Friday
                    "SATURDAY" -> Output += WeekDaysIDs.Saturday
                    "SUNDAY" -> Output += WeekDaysIDs.Sunday
                }
            }
            return Integer.toHexString(Output)
        }
        fun GetMainInfoRequest(): ByteArray {
            val CUtil = Calendar.getInstance()
            // 00 00        // + 12 + Month + Day + FF FF
            val Request = (GetMainInfo +
                    GetCalendarValueInHex(CUtil, Calendar.MONTH)
+
                    GetCalendarValueInHex(CUtil,
Calendar.DAY_OF_MONTH) +
                    GetCalendarValueInHex(CUtil,
Calendar.HOUR_OF_DAY) + "00"
                    + "12" +
                    GetCalendarValueInHex(CUtil, Calendar.MONTH)
+
                    GetCalendarValueInHex(CUtil,
Calendar.DAY_OF_MONTH) + "FFFF")
            return hexStringToByteArray(Request)
        }

        fun HRRealTimeControl(Enable: Boolean): ByteArray {
            val Request: String
            if (Enable) {
                Request = HRRealTimeHeader + "01"
            } else {
                Request = HRRealTimeHeader + "00"
            }
            return hexStringToByteArray(Request)
        }
        fun RequestHRHistory(FromDate: Calendar?): ByteArray {
            var FromDate = FromDate
            if (FromDate == null) {
                FromDate = Calendar.getInstance()
                FromDate!!.add(Calendar.DAY_OF_MONTH, -3)
            }
            val Request = HRHistoryHeader + "12" +
GetCalendarValueInHex(FromDate, Calendar.MONTH) +
                    GetCalendarValueInHex(FromDate,
Calendar.DAY_OF_MONTH) +
                    GetCalendarValueInHex(FromDate,
Calendar.HOUR_OF_DAY) + "0012" +
                    GetCalendarValueInHex(FromDate,
Calendar.MONTH) +
                    GetCalendarValueInHex(FromDate,
Calendar.DAY_OF_MONTH) +
                    GetCalendarValueInHex(FromDate,
```

```
Calendar.HOUR_OF_DAY) +
                    GetCalendarValueInHex(FromDate,
Calendar.MINUTE)
            return hexStringToByteArray(Request)
        }
        fun SyncTime(SyncTime: Calendar?): ByteArray {
            var SyncTime = SyncTime
            if (SyncTime == null) SyncTime =
Calendar.getInstance()
            val Request = TimeSyncHeader +
GetCalendarValueInHex(SyncTime!!, Calendar.YEAR) +
                    GetCalendarValueInHex(SyncTime,
Calendar.MONTH) +
                    GetCalendarValueInHex(SyncTime,
Calendar.DAY_OF_MONTH) +
                    GetCalendarValueInHex(SyncTime,
Calendar.HOUR_OF_DAY) +
                    GetCalendarValueInHex(SyncTime,
Calendar.MINUTE) +
                    GetCalendarValueInHex(SyncTime,
Calendar.SECOND)
            return hexStringToByteArray(Request)
        }
        /**
         * public static byte[] RequestSleepHistory(Calendar
FromDate){
         * String Request = SleepHistoryHeader+
         * GetCalendarValueInHex(FromDate, Calendar.MONTH)+
         * GetCalendarValueInHex(FromDate,
Calendar.DAY_OF_MONTH)+"0000";
         * return  hexStringToByteArray(Request);
         * }
         */
        fun EraseDatabase(): ByteArray {
            return hexStringToByteArray(EraseDataHeader)
        }
        fun RestoreToDefaults(): ByteArray {
            return hexStringToByteArray(RestoreCommandHeader)
        }
        fun SetAlarm(AlarmID: Long, IsEnabled: Boolean, Hour:
Int, Minute: Int, Days: Int): ByteArray {
            var Command = AlarmHeader
            Command +=
SubIntegerConversionCheck(java.lang.Long.toHexString(AlarmID))
            if (IsEnabled) Command += "01" else Command += "00"
            Command +=
SubIntegerConversionCheck(Integer.toHexString(Hour))
            Command +=
SubIntegerConversionCheck(Integer.toHexString(Minute))
            Command +=
SubIntegerConversionCheck(Integer.toHexString(Days))
            return hexStringToByteArray(Command)
```

```
        }
        fun StopLongAlarm(): ByteArray {
            return hexStringToByteArray(StopLongAlarmHeader)
        }
        fun SetGyroAction(IsEnabled: Boolean): ByteArray {
            var Command = GyroActionCommandHeader
            if (IsEnabled) Command += "01" else Command = "00"
            return hexStringToByteArray(Command)
        }
        fun BuildNotify(Message: String): ByteArray {
            var Request = ShortMessageHeader
            val Msg = Message.toByteArray()
            var Offset = 0
            for (i in 0..34) {
                if (i == 12) {
                    Request += "00"
                    Offset++
                    continue
                }
                if (i == 32) {
                    Request += "01"
                    Offset++
                    continue
                }
                if (i < Msg.size && !(i>=Msg.size)) {
                    if (Msg[i - Offset] > 0)
                        Request +=
SubIntegerConversionCheck(Integer.toHexString(Msg[i -
Offset].toInt()))
                    else
                        Request +=
SubIntegerConversionCheck(Integer.toHexString((Msg[i -
Offset]).toInt() and 0xFF))
                } else
                    Request += "00"
            }
            return hexStringToByteArray(Request + "2E2E2E")
        }
        fun BuildLongNotify(Message: String): ByteArray {
            val MessageBytes = Message.toByteArray()
            var Lenght = 5 + MessageBytes.size//MAX 12 bytes
            if (Lenght > 17) Lenght = 17
            var Request = (LongMessageHeaderPartOne +
SubIntegerConversionCheck(Integer.toHexString(Lenght))
                    + LongMessageHeaderPartTwo)
            for (i in 0..11) {
                if (i < MessageBytes.size)
                    Request +=
SubIntegerConversionCheck(Integer.toHexString(MessageBytes[i].to
Int()))
                else
                    break
```

```
            }
            return hexStringToByteArray(Request)
        }
        private fun CommandID14(Input: ByteArray) {
            if (Input[4].toInt() != 81 && Input[5].toInt() != 8)
return
            var Buff: ByteBuffer
            Buff = ByteBuffer.wrap(Input, 7, 2)
            val Steps = Buff.short
            Buff = ByteBuffer.wrap(Input, 10, 2)
            val Calories = Buff.short
            Callback?.MainInfo(Steps.toInt(), Calories.toInt())
        }

        private fun HRRTHandler(Input: ByteArray) {
            if (Input[4].toInt() != -124 || Input[5].toInt() !=
-128) return
            Callback?.HRIncome(Calendar.getInstance(),
Input[Input.size - 1].toInt())
        }
        private fun HRHistoryHandler(Input: ByteArray) {
            try {
                val CRecord = Calendar.getInstance()
                CRecord.set(CRecord.get(Calendar.YEAR), Input[7]
- 1, Input[8].toInt(), Input[9].toInt(), Input[10].toInt())
                Callback?.HRHistoryRecord(CRecord,
Input[11].toInt())
            } catch (Ex: Exception) {
                // todo analyse it ??? need info
            }

        }
        private fun MainHistoryHandler(Input: ByteArray) {
            if (Input[4].toInt() != 81 && Input[5].toInt() !=
32) return
            val RecordTime = Calendar.getInstance()
            RecordTime.set(RecordTime.get(Calendar.YEAR),
Input[7].toInt(), Input[8].toInt(), Input[9].toInt(), 0)
            var Buff = ByteBuffer.wrap(Input, 11, 2)
            val Steps = Buff.short
            Buff = ByteBuffer.wrap(Input, 14, 2)
            val Calories = Buff.short
            Callback?.MainHistoryRecord(RecordTime,
Steps.toInt(), Calories.toInt())
        }

        private fun BatteryCommandHandler(Input: ByteArray) {
            Callback?.BatteryInfo(Input[Input.size - 1].toInt())
        }

        /**
         * private void SleepHistoryHandler(byte[] Input){
```

```
         * Calendar RecordTime = Calendar.getInstance();
         * RecordTime.set(RecordTime.get(Calendar.YEAR),
Input[7]-1, Input[8], Input[9], Input[10], 0);
         * int Type = Input[11];    int Duration = Input[13];
         * if (Callback!
=null)Callback.SleepHistoryRecord(RecordTime, Duration, Type);
         * }
           */
        fun CommandAction(Input: ByteArray) {
            val CommandID = Input[2]
            when (CommandID) {
                (14).toByte()// OE, Main Info
                -> CommandID14(Input)
                (9).toByte()//HR history
                -> HRHistoryHandler(Input)
                (5).toByte() -> BatteryCommandHandler(Input)
                (4).toByte() -> HRRTHandler(Input)
                (22).toByte() -> MainHistoryHandler(Input)
                (11).toByte() -> {
                }
            }//SleepHistoryHandler(Input);
        }
    }
}
```