Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Голубкова Мария Сергеевна

Разработка Android приложения, имитирующего датчики интернета вещей

Курсовая работа

Научный руководитель:
к.ф.-м.н.
Бахмуров Анатолий Геннадьевич
Научный консультант:
Любицкий Александр Андреевич

Москва, 2023

Аннотация

Разработка Android приложения, имитирующего датчики интернета вещей

Голубкова Мария Сергеевна

Данная курсовая работа нацелена на создание Android приложения, имитирующего поведение датчиков интернета вещей в части передачи данных посредством Bluetooth Low Energy. В ходе работы разработан пользовательский интерфейс, реализованы работа с Bluetooth, генерация данных, их передача и запись этих данных в файл. В тексте работы описаны общие принципы создания периферийных устройств BLE.

Полученное приложение было успешно протестировано и использовано для исследования масштабируемости клиентской части сервиса сбора и обработки медицинской телеметрии.

Abstract

Development of Android application simulating internet of things sensors

Abstract

Содержание

1 Введение

В последние годы интернет вещей (Internet of Things, IoT) стал неотъемлемой частью технологического прогресса. IoT представляет собой сеть устройств, соединенных между собой, которые могут обмениваться данными и функционировать автономно без человеческого участия. Одной из наиболее перспективных областей применения IoT является динамично развивающийся рынок интернета медицинских вещей (Internet of Medical Things, IoMT).

IoMT — это новый способ реализации медицинских услуг и внедрения инноваций в здравоохранении. Системы мониторинга, дистанционной диагностики, устройства для жизнеобеспечения и управление лекарствами — все это включено в понятие интернета медицинских вещей. С помощью IoMT предоставляются новые возможности для мониторинга здоровья пациентов, снижения затрат на медицинское обслуживание и повышения качества жизни пациентов и медицинского персонала.

Выпускные квалификационные работы Фролова Александра[1] и Князева Егора[2] посвящены развитию сервиса по сбору и обработке медицинской телеметрии. Серверная часть позволит медицинским работникам дистанционно наблюдать за состоянием пациентов и своевременно реагировать на серьёзные изменения в их показателях здоровья, а клиентская часть обеспечит сбор данных с носимых устройств, отображение информации о них пользователю и их передачу на сервер.

Применение IoMT включает разнообразные устройства, такие как умные часы, наушники, браслеты, датчики температуры, сердечного ритма и артериального давления, а также дистанционные приборы и машины для диагностики и лечения пациентов. Количество новых устройств на рынке стремительно растёт, например, в рамках обзора устройств в ходе данной работы были обнаружены такие интересные приборы, как сенсорные стельки для предотвращения диабетических язв стоп, серьга, которая отслеживает уровень сахара в крови и даёт обратную связь в реальном времени, или наушники, регистрирующие электрокардиограмму.

Сервис сбора и обработки медицинской телеметрии должен иметь возможность быстро адаптироваться под изменения рынка. Данная курсовая работа нацелена на разработку удобного инструмента, позволяющего развивать сервис.

Разработка приложения, имитирующего поведение датчика интернета вещей, может значительно помочь в развитии сервиса сбора и обработки медицинской телеметрии и других сервисов Интернета вещей. Такое приложение позволит имитировать работу реального датчика и генерировать тестовые данные, что поможет разработчикам проводить испытания и тестирование новых функций и алгоритмов обработки данных без необходимости использования настоящих устройств.

2 Постановка задачи

Целями данной работы было:

- разработать приложение, имитирующее поведение датчика Интернета вещей в части передачи данных по интерфейсу Bluetooth LE, для исследования работоспособности клиентской части сервиса сбора и обработки медицинской телеметрии;
- создать возможности для выбора типа имитируемого датчика и настройки режима его работы;
- с помощью созданного имитатора датчика продемонстрировать возможность нагрузочного и функционального тестирования существующего приложения.

Для достижения поставленных целей необходимо выполнить следующие задачи:

1. Изучить существующие на рынке устройства Интернета медицинских вещей, входящие в них компоненты (датчики) и принципы их работы.
2. Изучить принципы разработки мобильных приложений под Android OS, освоить инструменты и фреймворки.
3. Ознакомиться с протоколом BLE, его работой и особенностями разработки BLE-приложений для Android.
4. Реализовать интерфейс приложения, логику его работы.

3 Актуальность задачи

3.1 Основные цели тестирования приложения сбора телеметрии

Для формирования списка функциональных возможностей разрабатываемого приложения, необходимо понять, что именно мы хотим тестировать.

Нагрузочное тестирование:

- Стресс-тестирование: тестирование работоспособности приложения при высоких частотах обновления данных
- Стабильность: проверка целостности и сохранности данных и работоспособности приложения при долговременной работе.
- Параллельное тестирование: определение количества датчиков, которые могут одновременно работать с приложением.

Функциональное тестирование

- Проверить отправку уведомлений (звонок, смс) на носимое устройство (например, часы)
- Получение недостоверных данных (заявленный максимум - 220, а приходит 400)
- Получение не изменяющихся данных

3.2 Потребности тестирования

Использование реальных датчиков может быть недостаточным при тестировании работоспособности приложений, взаимодействующих с устройствами IoT.

Разработка приложения, имитирующего поведение датчика, с возможностью контроля пользователем этого поведения позволит:

- имитировать датчики, приобрести которые в настоящий момент нет возможности
- тестировать приложение при работе с несколькими датчиками, нужного количества которых может даже не быть в наличии

- отследить возникновение ошибок на уровне приложений
- убедиться в передаче всех пакетов для устройства в разных режимах работы, например в спящем режиме
- имитировать некорректную работу датчика или же его новые возможности

3.3 Обзор существующих решений

Учитывая требования к приложению, были рассмотрены следующие существующие решения:

HCI sniffer - это инструмент, позволяющий перехватывать пакеты данных между уровнями контроллера и хоста. Снифер обеспечивает получение данных до их попадания на уровень приложения, но не предоставляет никаких возможностей для нагрузочного и функционального тестирования. Подобный инструмент существует в каждом смартфоне Android[3], а также на рынке есть множество аппаратных наблюдателей такого типа, их цена варьируется от 50 до 40 000 долларов[4].

BlueZ - это официальная реализация стека протоколов Bluetooth для Linux. Позволяет устанавливать соединение между ноутбуком и смартфоном и управлять им с помощью командной строки. С его помощью можно, например, писать Python скрипты[5], управляющие соединением, но использование компьютерных программ менее удобно для рядового пользователя, а главное - делает затруднительным параллельное тестирование, требующее несколько BLE устройств.

Nordic nRF Connect for Android[6] - это мобильное приложение с широкими возможностями. Оно позволяет настраивать собственную конфигурацию BLE сервера, рассылать широковещательные пакеты, обеспечивает одновременное сканирование, адвертайзинг и поддержку нескольких подключений. Однако, несмотря на длинный список возможностей, приложение всё же не предоставляет возможностей для автоматизированного функционального и более гибкого нагрузочного тестирования.

Решения, удовлетворяющие описанным выше потребностям тестирования, на данный момент неизвестны, их нет в открытом доступе.

4 Шаги решения задачи

Для создания приложения необходимо разбить исходную задачу на более простые подзадачи. Далее будут приведены результаты решения каждой из них.

1. Разработать графический интерфейс, позволяющий настраивать набор генерируемых данных, их величины и частоту передачи
2. Реализовать работу приложения с технологией Bluetooth и передачу данных
3. Создать возможности для записи трассы передаваемых значений и её последующей выгрузки
4. Протестировать корректность работы приложения с помощью сторонних инструментов
5. Убедиться в совместимости разработанного приложения и клиентской части сервиса сбора и обработки медицинской телеметрии, продемонстрировать возможности тестирования

5 Графический интерфейс

5.1 Выбор устройства

Первый экран приложения позволяет пользователю выбрать тип имитируемого устройства. Список был составлен на основе обзора существующих на рынке устройств интернета медицинских вещей. На выбор предложены:

- Hexoskin - смарт-рубашка с открытыми данными для мониторинга различных показателей здоровья человека (частота сердечных сокращений, частоты дыхания и т.д.) и других измерений активности, таких как подсчет шагов и частота вращения педалей.
- Ritmer - смарт-браслет для мониторинга сердечной активности.
- Dexcom - глюкометр с открытым исходным кодом.
- Smart весы - имитация поведения таких весов, как, например, Picooc или Mi Body Composition Scale.
- Smart часы - имитация поведения таких часов, как Xiaomi Mi Smart Band.
- Универсальное устройство - возможность для пользователя составлять свой список передаваемых данных.

При выборе конкретного устройства (любого, кроме универсального) также выбирается и соответствующий конфигурационный файл, задающий список предоставляемых устройством данных.

5.2 Универсальное устройство

При переходе на этот экран приложение устанавливает http-соединение с серверной частью сервиса сбора и обработки медицинской телеметрии и получает от сервера хранящиеся на нём конфигурационные файлы[7]. В соответствии с полученным списком на экране динамически[8] генерируются кнопки для выбора имитируемого устройства.

Таким образом для имитации устройства по индивидуальным запросам необходимо прежде всего написать конфигурационный файл, задающий список передаваемых

данных. В настоящий момент приложение может получать только загруженные на сервер авторизированными пользователями файлы, однако в будущем можно будет предоставить возможность для загрузки конфигурационного файла, например, из памяти смартфона.

5.3 Настройка передаваемых данных

На новом экране в соответствии с конфигурационным файлом генерируется динамический список[8] всех предоставляемых данных (например, частота сердцебиения и/или частота дыхания), и для каждого показателя предоставляются поля для установки минимального и максимального значений и частоты их обновления.

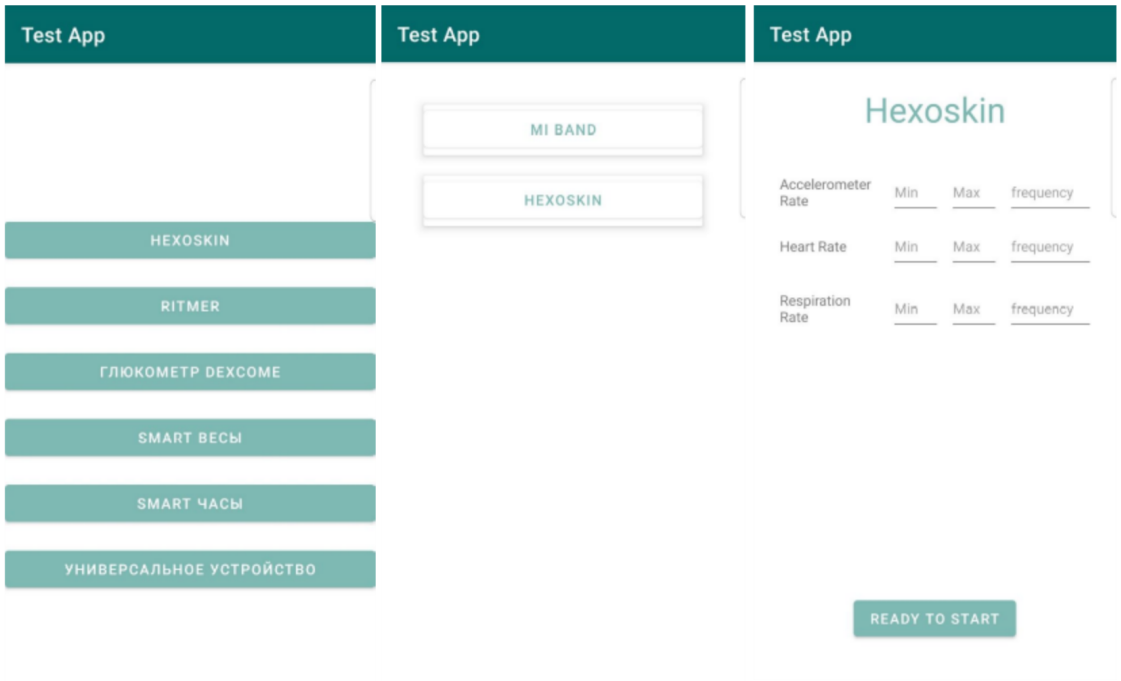

Рис. 1: Графический интерфейс

6 Основные сведения о Bluetooth

6.1 Bluetooth Classic vs BLE

В настоящее время существует два типа устройств с поддержкой Bluetooth - Bluetooth Classic и Bluetooth Low Energy (BLE) — это два отличающихся протокола беспроводной связи:

- Bluetooth Classic, используется в беспроводных громкоговорителях, автомобильных информационно-развлекательных системах и наушниках;
- Bluetooth Low Energy (BLE), т.е. Bluetooth с низким энергопотреблением. Он чаще всего применяется в приложениях, чувствительных к энергопотреблению или в устройствах, передающих небольшие объемы данных с большими перерывами между передачами (например, разнообразные сенсоры параметров окружающей среды или управляющие устройства, такие как беспроводные выключатели).

6.2 Описание стека протоколов BLE

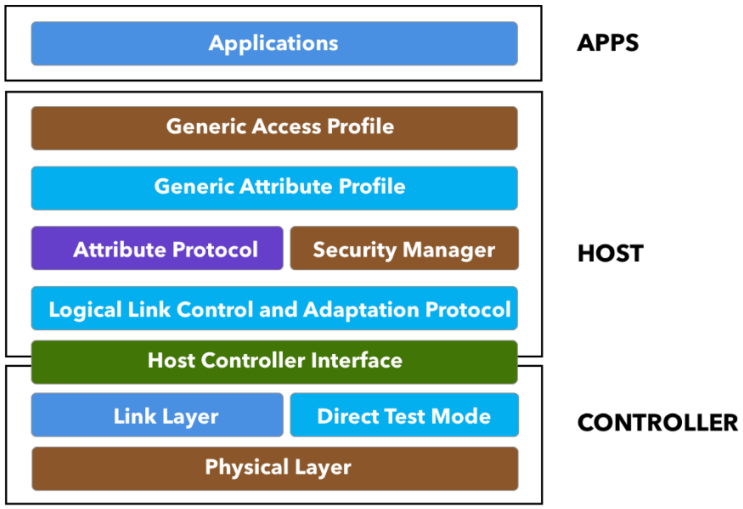

Рис. 2: Архитектура BLE

На Рис. 2 представлена архитектура BLE[9]. Я подробнее остановлюсь на трёх уровнях: канальный (Link Layer), уровень протокола атрибутов (ATT) и общий профиль атрибутов (GATT).

6.3 Канальный уровень

Канальный уровень управляет различными состояниями устройства BLE:

- **Standby**: когда устройство не передает и не принимает никаких данных.
- **Advertising**: устройство посылает широковещательные пакеты для обнаружения другими устройствами.
- **Scanning**: ищутся устройства, посылающие широковещательные пакеты.
- **Initiating**: начинается процесс установки соединения с устройством, находящимся в состоянии advertising.
- **Connected**: одно устройство установило соединение с другим и регулярно обменивается с ним информацией.

Определив понятия сканирования и адвертайзинга, можно определить роли устройств:

Периферийное устройство – устройство, которое объявляет о своем присутствии путем адвертайзинга и принимает запросы на соединение от центральных устройств.

Центральное устройство – устройство, которое обнаруживает периферийные устройства и считывает передаваемую ими информацию. Оно также может устанавливать соединение с одним или несколькими устройствами одновременно.

6.4 Протокол атрибутов (ATT)

Протокол ATT определяет, в каком виде сервер представит свои данные клиенту и как эти данные будут структурированы.

Сервер - устройство, которое предоставляет данные, которые содержит или которыми управляет, а также получает входящие команды от связанного устройства и отправляет ответы и уведомления.

Клиент - устройство, которое взаимодействует с сервером с целью прочитать предоставляемые сервером данные и/или для того, чтобы контролировать его поведение. Это устройство, которое посылает команды и запросы и получает входящие уведомления.

Данные, предоставляемые сервером, сгруппированы в атрибуты. Атрибут это общий термин для любых типов данных, предоставляемых сервером. К атрибутам относятся сервисы, характеристики и дескрипторы.

6.5 Общий профиль атрибутов (GATT)

Профиль GATT определяет формат сервисов и их характеристик, а также процедуры, используемые для взаимодействия с этими атрибутами, такие как обнаружение сервисов, чтение и запись характеристик, уведомления.

Сервисы это группа из одного или большего числа атрибутов, часть (или все) из которых являются характеристиками. Он предназначен для группировки связанных атрибутов, удовлетворяющих специфической функциональности сервера.

Характеристика всегда является частью сервиса и предоставляет часть тех данных, которые сервер хочет предоставить клиенту. Характеристика содержит набор атрибутов, которые облегчают работу с содержащимися в характеристике данными:

- Свойства: представлены набором бит, определяющих то, каким образом значение характеристики может использоваться (например, чтение, запись, запись без ответа, уведомление).
- Дескрипторы: используются для хранения информации, связанной со значением характеристики. Примеры использования: расширенные свойства, пользовательское описание, поля, используемые для подписки на уведомления и индикации.

Сервисы, характеристики и дескрипторы, являясь атрибутами, имеют тип атрибута (универсальный уникальный идентификатор, UUID) - это 16-битное (в случае стандартных атрибутов) или 128-битное число (в случае атрибутов, определенных разработчиком устройства). Для того чтобы клиент мог получить информацию о каком-либо атрибуте, он должен знать его UUID, поэтому для согласования набора атрибутов у имитируемого устройства с клиентским приложением, все необходимые UUID хранятся в конфигурационном файле.

6.6 Промежуточный итог

На основании описанных выше особенностей архитектуры BLE можно заключить, что разрабатываемое приложение должно быть, с точки зрения протокола BLE, периферийным устройством: оно должно рассылать широковещательные пакеты, пока не будет обнаружено центральным устройством, которое установит с ним соединение для считывания данных. Приложение также должно быть сервером: оно должно создавать

GATT-таблицу (иерархия и отношения между различными атрибутами) для предоставляемых данных, а так же создавать эти данные и передавать клиенту. GATT-таблица создаётся на основании конфигурационного файла, задающего набор сервисов, характеристик в них и их UUID.

6.7 Программная реализация GATT-таблицы

Далее приведены функции, создающие GATT-сервер. Реализация адвертайзинга осуществляется библиотечными функциями и не представляет интереса.[1]

```
1 private fun startServer() {
2   if (!hasRequiredRuntimePermissions()) {
     requestRelevantRuntimePermissions() }
3   else {
4     val bluetoothManager = getSystemService(Context.BLUETOOTH_SERVICE) as
     BluetoothManager
5     gattServer = bluetoothManager.openGattServer(this, gattServerCallback)
6     gattServer.addService(newService(0))
7   }
8 }
```

Прежде всего нужно создать GATT-сервер. При открытии сервера на строке 5 листинга в аргументах функции указывается класс обратного вызова[10]. Функции этого класса автоматически вызываются при событиях на сервере (например, чтение и запись атрибутов, соединение и т.д.). Так же в этом классе есть функция, уведомляющая о статусе добавления в GATT-таблицу нового сервиса.

Очень важно не проводить никаких операций с сервером до тех пор, пока успешно не выполнятся предыдущие, поэтому в стартовой функции мы добавляем только первый сервис, последующие же добавляются внутри функции обратного вызова.

Далее рассмотрим создание очередного сервиса.

```
1 private fun newService(id: Int): BluetoothGattService {
2   val charConfig = gattTableSettings.characteristics[charKeys[id]]
3   val service = BluetoothGattService(
4     UUID.fromString(charConfig!!.serviceUuid),
5     BluetoothGattService.SERVICE_TYPE_PRIMARY,
6   )
```

[1]Полный текст программы: https://github.com/IoMT-LVK/TestApp

```
7   val characteristic = BluetoothGattCharacteristic(
8     UUID.fromString(charConfig.characteristicUuid),
9     BluetoothGattCharacteristic.PROPERTY_READ or
      BluetoothGattCharacteristic.PROPERTY_NOTIFY,
10    BluetoothGattCharacteristic.PERMISSION_READ,
11  )
12  val descriptor = BluetoothGattDescriptor(
13    UUID.fromString(CLIENT_CONFIGURATION_DESCRIPTOR_UUID),
14    BluetoothGattDescriptor.PERMISSION_READ or BluetoothGattDescriptor.
      PERMISSION_WRITE
15  )
16  characteristic.addDescriptor(descriptor)
17  service.addCharacteristic(characteristic)
18  newBytesByUUID[UUID.fromString(charConfig.characteristicUuid)] =
      byteArrayOf()
19  scope.launch { updateData(characteristic, charKeys[id]) }
20  return service
21 }
```

Прежде всего для каждого сервиса мы получаем его параметры из конфигурационного файла. Для сервиса и характеристики UUID указан в этом файле. Также из фрагмента кода видна иерархия используемых атрибутов: сервер включает множество сервисов, каждый сервис включает в себя одну характеристику, характеристика включает в себя дескриптор. Но если у сервисов и характеристик UUID различные и берутся из конфигурационного файла, то в каждую характеристику добавляется дескриптор с одним и тем же UUID.

CLIENT_CONFIGURATION_DESCRIPTOR_UUID - это дескриптор специального вида (дескриптор конфигурации), нужный для создания возможности подписки на обновления значений определённой характеристики. Используя этот дескриптор, сервер позволяет клиенту при желании подписываться на обновления, и тогда сервер сможет просто отправлять эти данные, а клиенту не придётся постоянно спрашивать, есть ли на сервере какие-то изменения.

7 Генерация данных

В функции newService() после создания каждого сервиса в сопрограммах[11] для каждой характеристики запускаются функции updateData, в которых асинхронно, с задержкой, определённой пользователем при настройке параметров, из промежутка значений, определённого пользователем, выбираются случайные числа. Число присваивается соответствующей характеристике, после чего подключенному устройству отправляется уведомление об обновлении этой характеристики.

7.1 Формат данных

Данные, отправляемые сервером имеют определённую структуру, она определяется либо стандартами Bluetooth SIG[12], либо производителем конкретного устройства IoT. В данный момент в приложении реализована корректная генерация значений только для стандартного сервиса частоты сердцебиения.

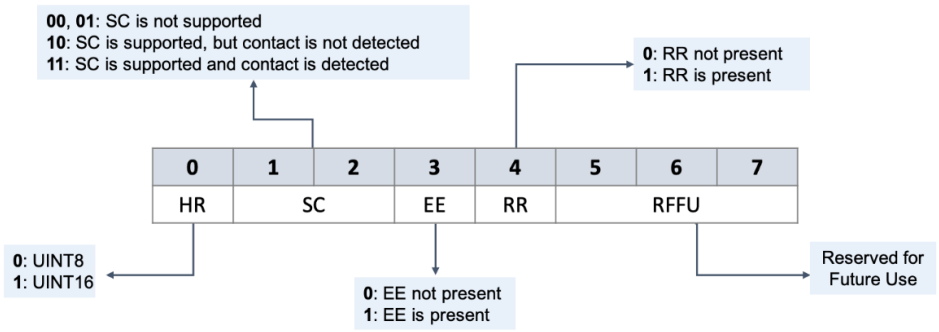

Рис. 3: Структура служебного байта

На Рис. 2 [13] изображена структура служебного байта сервиса частоты сердцебиения. Здесь SC (Sensor Contact) - состояние контакта сенсора с кожей, EE (Energy Expended) - наличие в сервисе характеристики, отображающей расход электроэнергии, RR (RR-Intervals) - поддержка вариабельности сердечного ритма. На данный момент вся эта информация не представляет для нас интереса. В текущей реализации единице равен только самый первый бит, чтобы не ограничивать пользователя в размере передаваемых данных.

8 Запись трассы

Для тестирования работы клиентского приложения необходимо сравнение полученных от датчика данных с данными, в последствии поступившими на сервер, также интерес представляет временная задержка при передаче данных. Поэтому при подключении стороннего устройства к разрабатываемому приложению, все данные, отправляемые ему, записываются в текстовый файл вместе с названием характеристики и временем отправки. После окончания работы становится доступной кнопка SHARE LOGS, с помощью которой созданный файл можно отправить по почте или в любые мессенджеры.

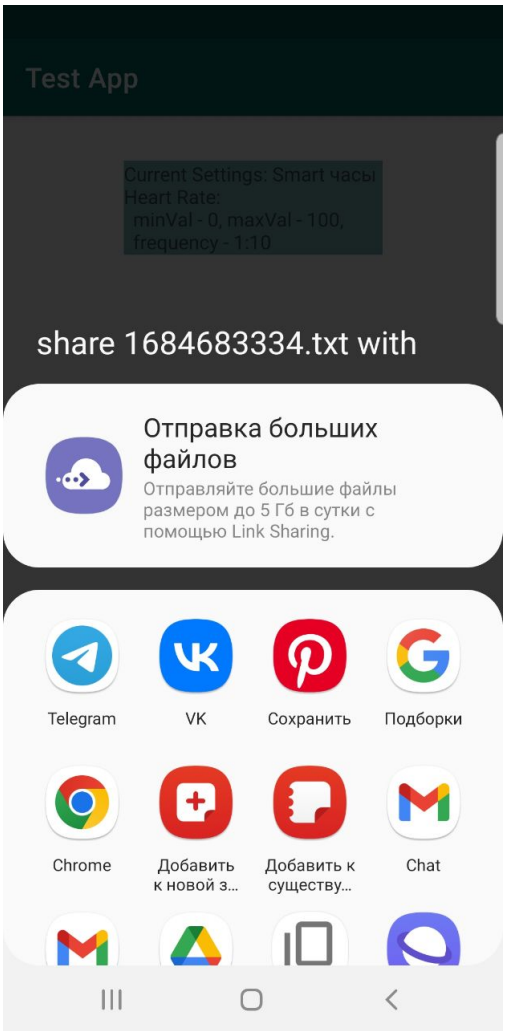

Рис. 4

9 Результаты работы и тестирование

В ходе курсовой работы удалось реализовать большинство планируемых функциональных возможностей приложения. В будущем необходимо добавить служебные байты для большего числа различных сервисов, реализовать возможности для тестирования отображения на смарт-часах входящих на смартфон уведомлений, а также дать возможность пользователю загружать собственные конфигурационные файлы.

9.1 Тестирование

На первых этапах разрабатываемое приложение тестировалось с помощью приложения nRF Connect, рассмотренного в главе 3.2. С его помощью получилось устранить несколько незначительных ошибок в логике работы программы.

Когда все основные функциональные возможности приложения были реализованы, а их корректная работа подтверждена, была обнаружена серьёзная ошибка при попытках подключения разработанного приложения к клиентской части сервиса сбора и обработки медицинской телеметрии.

Ошибка звучала так: GATT ERROR Connection state changed with status: 22.

9.2 Исправление ошибок

Когда клиент подписывается на обновление характеристики, он записывает в дескриптор соответствующее этому состоянию значение. В библиотеках Kotlin для работы с Android многое реализовано 'под капотом', большинство необходимых действий и проверок выполняются автоматически, но проверка значения дескриптора при попытке отправить уведомление автоматически не производилась, более того, оказалось необходимым хранить значение, установленное клиентом, и каждый раз при отправке уведомления проверять, действительно ли клиент хочет его получить.

Для устранения этой ошибки был создан словарь charConfigurations, позволяющий по характеристике получить значение дескриптора конфигурации, установленное для этой характеристики. Затем были написаны функции, проверяющие сначала существование такого дескриптора у характеристики, затем существование информации о нём в словаре и наконец его значение.

```
private fun clientEnabledNotifications(characteristic:
    BluetoothGattCharacteristic): Boolean {
  val descriptorList = characteristic.descriptors
  val descriptor = descriptorList.find { isClientConfigurationDescriptor(
    it) }
     ?: // There is no client configuration descriptor, treat as true
     return true
  val charConfiguration = charConfigurations[characteristic]
     ?: // Descriptor has not been set
     return false
  return Arrays.equals(charConfiguration, BluetoothGattDescriptor.
    ENABLE_NOTIFICATION_VALUE)
}

private fun isClientConfigurationDescriptor(descriptor:
    BluetoothGattDescriptor?) =
  descriptor?.let {
     it.uuid.toString() == CLIENT_CONFIGURATION_DESCRIPTOR_UUID
} ?: false
```

После успешного исправления ошибок разработанное приложение было использовано в дипломной работе Александра Фролова[1].

10 Заключение

В рамках данной курсовой работы было разработано Android-приложение, имитирующее поведение датчика интернета вещей в части передачи данных посредством BLE. Приложение позволяет выбрать имитируемое устройство с заранее заданным набором характеристик или создать собственный список характеристик, затем настроить параметры передачи данных (интервал генерируемых значений и частоту их обновления). Отправляемые данные записываются в файл, который впоследствии может быть выгружен для дальнейшего использования.

Было проведено тестирование корректности работы приложения, обнаруженные ошибки были успешно устранены.

Список литературы

[1] *В., Фролов А.* Разработка приложения сбора данных с нескольких устройств интернета вещей, подключенных одновременно, на примере медицинской телеметрии / Фролов А. В. — 2023. — https://github.com/IoMT-LVK/papers/blob/main/client/4.%20Frolov_Diploma_Paper.pdf.

[2] *И., Князев Е.* Разработка серверной части приложения интернета вещей на основе принципов передачи репрезентативного состояния / Князев Е. И. — 2023. — https://github.com/IoMT-LVK/papers/blob/main/server/4.%20Knyazev_Diploma_Paper.pdf.

[3] *Medium.* Bluetooth LE packet capture on Android. — 2020. https://medium.com/propeller-health-tech-blog/bluetooth-le-packet-capture-on-android-a2109439b2a1.

[4] *Bits, Novel.* BLE Sniffer Basics + Comparison Guide. — 2023. https://novelbits.io/bluetooth-low-energy-ble-sniffer-tutorial/.

[5] *Website, Bluetooth® Technology.* Bluetooth® Technology for Linux Developers. — 2023. https://www.bluetooth.com/bluetooth-resources/bluetooth-for-linux/.

[6] *Semiconductor, Nordic.* nRF Connect for Mobile. https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-mobile.

[7] *В., Фролов А.* Развитие клиентской части сервиса сбора и обработки медицинской телеметрии / Фролов А. В. — 2022. — https://github.com/IoMT-LVK/papers/blob/main/client/3.%20Frolov_Couse_Paper.pdf.

[8] *Developers, Android.* Create dynamic lists with RecyclerView. https://developer.android.com/develop/ui/views/layout/recyclerview.

[9] *Afaneh, Mohammad.* Intro to Bluetooth low energy / Mohammad Afaneh. — Novel Bits, 2018.

[10] *Developers, Android.* BluetoothGattCallback. https://developer.android.com/reference/android/bluetooth/BluetoothGattCallback.

[11] *Help, Kotlin.* Coroutines guide: Kotlin. https://kotlinlang.org/docs/coroutines-guide.html.

[12] *Website, Bluetooth® Technology.* Specifications. https://www.bluetooth.com/specifications/specs/.

[13] *Bahameish, Mariam.* Extracting Heart Rate Measurements from Bluetooth LE Packets. — 2019. https://mariam.qa/post/hr-ble/.