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

Фролов Александр Валерьевич

Разработка приложения сбора данных с нескольких устройств интернета вещей, подключенных одновременно, на примере медицинской телеметрии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

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

Москва, 2023

Аннотация

Разработка приложения сбора данных с нескольких устройств интернета вещей, подключенных одновременно, на примере медицинской телеметрии

Фролов Александр Валерьевич

Данная выпускная квалификационная работа нацелена на создание клиентского приложения сервиса сбора и обработки медицинской телеметрии с устройств интернета медицинских вещей.

В ходе работы изучена проблема масштабируемости клиента в части взаимодействия с несколькими датчиками интернета вещей, предложено архитектурное решение, позволяющее считывать данные с датчиков в фоновом режиме, исследованы задержки передачи в зависимости от числа подключенных устройств.

Abstract

Development of data collection application for simultaneous handling of multiple IoT (Internet of Things) devices: medical telemetry case study

Aleksandr Frolov

This graduation thesis aims to develop the client of a service for collecting and processing medical telemetry data from IoT devices.

The study explores the scalability issue of the client-side in the area of simultaneous interaction with multiple Internet of Medical Things sensors. An architectural solution is proposed that allows reading data from sensors in the background, and dekays are investigated depending on the number of connected devices.

Содержание

1 Введение

В современном мире технологии развиваются стремительными темпами, затрагивая все аспекты повседневной жизни. Одним из самых перспективных и актуальных направлений является Интернет вещей (IoT). Интернет вещей - это сеть, состоящая из датчиков и приборов, взаимодействующих друг с другом с целью мониторинга различных показателей системы и управлением этой системой на основе считанных данных.

Умный дом - пример использования интернета вещей для мониторинга показателей дома (таких, как давление, температура, освещенность) и управления приборами (такими как кондиционеры, лампы, розетки).

В последние годы особое внимание уделяется использованию IoT в медицине, где это направление получило название Интернет медицинских вещей (IoMT). IoMT предоставляет возможность мониторинга состояния человеческого организма в реальном времени, улучшая процессы лечения, восстановления пациентов и их мониторинг, а также совершенствуя процесс физических тренировок. Проблема мониторинга состояния человека актуальна не только для медицинских учреждений, но и для других сфер, где важно оперативное выявление патологий и принятие адекватных мер. Сбор и обработка информации о текущем функционировании организма человека в реальном времени позволяют быстро анализировать медицинские показатели и выявлять патологии, что может быть определяющим для здоровья конкретного человека при критическом состоянии пациента.

Однако, на текущий момент существующие платформы мониторинга здоровья человека, такие как Fitbit, Omron, Apple Health и многие другие, а также интеграционные системы типа Validic, Seqster, CareSimple, Capsule Technologies, имеют свои ограничения. Все эти программные продукты являются проприетарными, из них невозможно получить данные или они выдаются уже в обработанном виде. Кроме того, такие решения обычно не являются бесплатными и открытыми проектами, что ограничивает их доступность и применимость для медицинских учреждений и исследователей.

В связи с этим, актуальной является задача разработки приложения сбора данных интернета вещей с датчиков, подключенных одновременно, на примере медицинской телеметрии. Создание такой платформы должно предусматривать открытый исходный код, что обеспечит доступность и возможность его доработки для медицинских учреждений и исследователей. Это позволит сформировать сообщество разработчиков и

экспертов, способствующее обмену знаниями и опытом, а также стимулированию совместной работы над улучшением разработанных решений.

Существующая реализация приложения подразумевает использование в активном режиме - приложение должно быть запущено, что дополнительно нагружает телефон, приводит к повышенному использованию ресурсов и ограничивает использование телефона. Необходимо разработать архитектуру, позволяющую взаимодействовать с несколькими датчиками интернета медицинских вещей в фоновом режиме и реализовать ее в рамках клиентской части сервиса сбора и обработки медицинской телеметрии.

Результатом выполнения данной дипломной работы станет приложение, которое обеспечит постоянный сбор и обработку медицинских данных, а также предоставление их в реальном времени на сервер.

2 Постановка задачи

Заделом для данной работы служит выпускная квалификационная работа студента Чайчица Даниила [1], а также курсовая работа автора [2].

В ходе первой работы было написано приложение, использующееся для подключения к датчикам IoMT по протоколу Bluetooth LE. В ходе второй работы была реализована поддержка разных датчиков, передающих разные данные. Описание параметров, передаваемых датчиком, не программируется в исходном коде, а задаётся в конфигурационном файле. В связи с особенностями реализации предыдущей версии приложения, одновременное подключение нескольких устройств было невозможно.

Цель данной выпускной квалификационной работы - улучшить существующее приложение для смартфона в части считывания данных с нескольких датчиков, подключенных одновременно, и обеспечить надежную отправку данных на сервер для дальнейшей обработки.

Для достижения поставленной цели в работе необходимо решить следующие задачи:

1. Сравнить существующие протоколы обмена данными Интернета вещей с целью выбора протокола, подходящего для эффективной отправки большого числа маленьких сообщений;

2. Разработать и реализовать архитектуру приложения, позволяющую одновременно взаимодействовать как минимум с тремя датчиками интернета медицинских вещей в фоновом режиме;

3. Реализовать сохранение считанных данных в локальное хранилище на смартфоне;

4. Реализовать возможность периодической отправки собранных данных на сервер с периодом 15 минут, а также предусмотреть возможность настройки периода;

5. Провести тестирование разработанного приложения на долговечность функционирования в течение суток;

6. Провести тестирование разработанного приложения в части масштабируемости на несколько одновременно подключенных устройств;

7. Подготовить документацию по разработанному приложению и опубликовать приложение в открытом доступе.

Результат выполнения поставленных задач - приложение сбора данных интернета медицинских вещей в фоновом режиме с нескольких (вообще говоря, разнотипных) датчиков, подключенных одновременно, с открытым задокументированным исходным кодом и возможностью использования в учебных и исследовательских целях в области медицины, а также в области информатики.

3 Обзор протоколов прикладного уровня обмена данных интернета вещей

3.1 Цель обзора

Цель обзора - рассмотреть основные протоколы прикладного уровня для обмена данными Интернета вещей (IoT) и определить их преимущества и недостатки в контексте создания приложений сбора данных интернета вещей, в том числе и для приложений, связанных с медицинской телеметрией. Обзор протоколов IoT поможет выбрать наиболее подходящий протокол для приложения сбора данных интернета вещей, учитывая особенности задачи и требования к энергопотреблению, надежности и скорости передачи данных.

Предпочтительным будет протокол, способный эффективно отправлять большое количество маленьких сообщений и поддерживающий подтверждение получения данных. Такой протокол позволит отправлять данные, считанные с датчика, без привязки к конкретному датчику (множеству характеристик и частоты их отправки).

Данный обзор необходим, так как протокол MQTT [3] использовался в предыдущей реализации, однако выбор его обоснован не был [1]. Требования к приложению расширились по сравнению с предыдущей реализацией, ранее выбранный протокол может не подходить для данной задачи.

3.2 Способы обмена данными

Существует два основных способа передачи данных в сетях IoT:

- "запрос-ответ"(request-reply) [4];
- "издатель-подписчик"(publisher-subscriber) [5].

"Запрос-ответ"(request-reply) – это способ передачи данных, при котором клиент отправляет запрос на сервер. Этот способ передачи данных используется в тех случаях, когда клиенту необходимо получить определенную информацию от сервера, например, данные о состоянии устройства, температуре или другой метрике.

Использование способа "запрос-ответ"(request-reply) в рамках данной задачи также имеет ряд преимуществ:

1. Надежность: Каждый запрос отправляется от устройства на сервер и получает ответ. Это обеспечивает надежность и гарантирует, что данные были успешно получены сервером;

2. Контроль за передачей данных: Способ "запрос-ответ"позволяет точно контролировать, когда данные были отправлены на сервер и были успешно получены;

3. Простота реализации: Реализация способа "запрос-ответ"может быть достаточно простой, особенно в сравнении с более сложными протоколами, такими как MQTT или CoAP.

"Издатель-подписчик"(publisher-subscriber) – это способ передачи данных, при котором устройства подписываются на темы (топики, topics), а устройства-издатели отправляют данные на эти темы. При этом каждое подписавшееся устройство получает данные только по топикам, на которые оно подписалось. Способ передачи данных "издатель-подписчик обеспечивает надежную доставку сообщений в режиме реального времени, что является важным для приложений, связанных с медицинской телеметрией.

Использование способа "издатель-подписчик"(publisher-subscriber) в рамках данной задачи также имеет ряд преимуществ:

- Масштабируемость: в случае добавления новых датчиков, которые могут генерировать данные, необходимо изменять код приложения, если используется способ "запрос-ответ". С другой стороны, способ "издатель-подписчик"может быть легко расширен путем добавления новых топиков для новых датчиков без изменения кода приложения;

- Разделение обязанностей: "издатель-подписчик"позволяет разделить обязанности между клиентским и серверным приложениями. Серверное приложение отвечает за сбор данных и публикацию их в топики, а клиентское приложение подписывается на нужные топики и получает данные по мере их поступления;

- Асинхронность: в случае использования способа "запрос-ответ"клиенты должны отправлять запросы, чтобы получить данные. С другой стороны, способ "издатель-подписчик"позволяет клиентам получать данные асинхронно, как только они появляются в топике. Это может быть особенно полезно в случае, когда данные должны быть получены в режиме реального времени.

Способ "издатель-подписчик"более предпочтителен для приложений, связанных с медицинской телеметрией, поскольку он обеспечивает передачу данных в режиме реального времени, имеет более высокую масштабируемость и позволяет уменьшить нагрузку на устройства и сеть.

3.3 Сравнение протоколов обмена данными IoT

В работе рассматриваются такие протоколы, как MQTT, CoAP, HTTP и AMQP.

1. MQTT (Message Queuing Telemetry Transport) [3] – это протокол для передачи сообщений между устройствами и серверами в формате "издатель-подписчик". Протокол использует минимальный объем данных и предоставляет надежный механизм обработки сообщений в режиме реального времени. Протокол MQTT часто используется в приложениях IoT, связанных с медицинской телеметрией, так как он обеспечивает низкую задержку передачи данных и надежную доставку сообщений;

2. HTTP (Hypertext Transfer Protocol) [6] – это стандартный протокол для передачи данных между клиентом и сервером. Протокол HTTP обеспечивает передачу данных в формате "запрос-ответ а также поддерживает различные методы передачи данных, например, GET, POST, PUT и DELETE. HTTP является одним из самых распространенных протоколов в сети Интернет, и его использование в приложениях IoT, связанных с медицинской телеметрией, может быть полезным для интеграции с другими системами;

3. CoAP (Constrained Application Protocol) [7] – это протокол, который обеспечивает передачу данных между устройствами и серверами в формате "запрос-ответ". Протокол был разработан для использования в сетях с ограниченными ресурсами, таких как устройства IoT. CoAP использует UDP-протокол для передачи данных, что, по сравнению с HTTP или другим протоколом, в основе которого лежит TCP, обеспечивает низкую задержку и уменьшает объем передаваемых данных;

4. AMQP (Advanced Message Queuing Protocol) [8] – это протокол для передачи сообщений между устройствами и серверами в формате "издатель-подписчик". Протокол AMQP предоставляет механизмы обработки сообщений в режиме реального

времени, а также поддерживает многообразие форматов передаваемых данных, что позволяет использовать его в различных приложениях IoT.

Основные критерии для сравнения данных протоколов - размер заголовка в байтах, наличие механизмов подтверждения получения сообщения и тип общения.

Критерий	HTTP	CoAP	MQTT	AMQP
Тип общения	З-О	З-О	И-П	И-П
Основной протокол	TCP	UDP	TCP	TCP
Размер заголовка (байты)	Б (500)	М (4)	М (2)	Б (50)
Механизмы шифрования	SSL/TLS	DTLS	SSL/TLS	SSL/TLS
Подтверждения получения	Да	Нет	Да	Да

Таблица 1: Сравнение протоколов обмена данными

На таблице 1 изображены результаты сравнения способов передачи данных. Размер заголовка указан в байтах, буквы М и Б обозначают маленький и большой соответственно. Тип общения обозначен первыми буквами ("Издатель-Подписчик"и "Запрос-Ответ")

3.4 Результаты обзора

С учетом энергопотребления и ресурсов устройства, а так же размера заголовка, предпочтительно использовать способ передачи данных MQTT в рамках данной работы [9]. MQTT имеет малый размер заголовка и гарантирует получение сообщений подписчиком, что делает его предпочтительным для передачи большого числа записей с датчиков с телефона на сервер.

3.5 Выводы из обзора

В данном разделе был проведен обзор четырех протоколов обмена данными IoT, в результате которого для дальнейшей реализации был выбран протокол MQTT, использующийся в предыдущей реализации. В рамках решения задачи одновременного считывания данных с различных датчиков необходимо для каждого датчика выделить один топик под отправку данных, считанных с этого датчика.

4 Архитектурные решения для разработки приложений интернета вещей

4.1 Виды архитектуры приложения

Рассмотрим два вида архитектуры приложения:

- Монолитная архитектура;
- Сервисная архитектура.

Монолитная архитектура предполагает разработку приложения как единого, самодостаточного блока, включающего в себя все необходимые функции и компоненты. В данном случае, монолитная архитектура могла бы предполагать разработку приложения, которое бы собирало данные с датчиков на устройстве, сохраняло из в локальном буфере, а затем отправляло на сервер при помощи выбранного протокола. Монолитная архитектура предполагает, что все компоненты приложения, такие как пользовательский интерфейс, логика приложения и доступ к данным, хранятся в контексте одного приложения (а значит, имеют время жизни равное времени жизни приложения). В случае приложения сбора и обработки данных с нескольких Bluetooth LE датчиков, в монолитной архитектуре все операции по подключению к датчикам, считыванию и обработке данных, а также их отправке на сервер осуществляются в рамках одного приложения. Данная реализация не позволяет считывать данные в фоновом режиме.

Преимущества монолитной архитектуры включают простоту разработки и управления, поскольку все компоненты приложения находятся в одном месте и могут быть легко отлажены и изменены. Однако, такая архитектура может быть менее масштабируемой и гибкой, так как любые изменения в функциональности приложения могут потребовать больших изменений в коде.

Сервисная архитектура предполагает разделение приложения на независимые компоненты, которые обмениваются данными посредством Android API [10]. В случае приложения сбора и обработки данных с нескольких BLE датчиков, можно выделить отдельный сервис, который будет осуществлять подключение к датчикам, считывание и обработку данных, а также их отправку на сервер. Сервисная архитектура позволяет повысить гибкость и масштабируемость приложения, улучшить производительность и

эффективность использования ресурсов. В данном случае, приложение могло бы включать несколько сервисов, таких как сервис для сбора данных с датчиков и их буферизации и сервис для отправки данных на сервер. Преимущества сервисной архитектуры включают более легкое масштабирование и гибкость, так как каждый сервис может быть изменен или заменен без воздействия на другие компоненты приложения. Однако, такая архитектура может быть более сложной в разработке и управлении, так как необходимо обеспечивать согласованность между различными сервисами.

Монолитная архитектура проста в реализации и понимании, но может иметь проблемы с производительностью и расходом ресурсов, особенно в случае сложных и многофункциональных приложений. Сервисная архитектура, в свою очередь, позволяет улучшить масштабируемость, гибкость и производительность приложения, но требует дополнительных усилий по проектированию и реализации.

Без применения сервисной архитектуры затруднительно написать приложение, которое считывает данные с датчиков Bluetooth LE в фоновом режиме. Приложение должно работать в фоновом режиме и в то же время получать данные от датчиков. В монолитной архитектуре такой сценарий может привести к быстрому разряду батареи и ограничениям на использование приложения в фоновом режиме (пользователю необходимо поддерживать приложение в рабочем состоянии, так как Android может останавливать неиспользуемые приложения), что значительно снизит его эффективность и удобство использования. В свою очередь, сервисная архитектура обеспечивает постоянную работу сервисов в фоновом режиме, позволяет получать данные от датчиков и обрабатывать их без участия пользователя, что позволяет повысить эффективность приложения и удобство его использования.

4.2 Средства Android API

Различные API, такие, как Service, ForegroundService, WorkManager, Companion Service и другие, являются важными средствами для разработки приложений на Android, особенно для приложений, связанных с интернетом вещей и мониторингом состояния человека. Рассмотрим их возможности и применение в рамках приложения сбора и обработки данных с нескольких датчиков BLE.

- Service [11] - это основное средство для выполнения длительных задач в фоно-

вом режиме. Сервис может работать в фоновом режиме даже после того, как пользователь закрыл приложение (однако начиная с Android версии 8.0 введено ограничение на работу при закрытом приложении [12]). Сервис может использоваться для считывания данных с датчиков BLE, сохранения данных в локальной базе данных и передачи данных на удаленный сервер. Запущенный сервис может останавливаться програмно или при помощи пользовательского интерфейса ОС;

- ForegroundService [13] - это сервис, который может работать в фоновом режиме с постоянным уведомлением в статус-баре. Этот тип сервиса может быть использован для выполнения задач, которые требуют постоянного внимания пользователя, например, считывание показаний сердечного ритма. Приложение с ForegroundService не может быть закрыто без уведомления пользователя. Запущенный сервис может останавливаться програмно или при помощи пользовательского интерфейса ОС;

- WorkManager [14] - это часть библиотеки Android Jetpack , разрабатываемой компанией Google, используемый для выполнения асинхронных задач в фоновом режиме. WorkManager используется для выполнения задач, которые могут быть отложены и запущены в более удобное время. Он может быть использован для выполнения задач сбора данных с датчиков в фоновом режиме и отправки их на сервер в определенное время;

- Companion Service [15] - это специальный тип сервиса, который используется для взаимодействия с внешним устройством, например, с датчиком Bluetooth LE. Companion Service позволяет приложению подключаться к устройству в фоновом режиме и получать данные с него. Он может быть использован для считывания данных с датчиков BLE и сохранения их в локальной базе данных.

В целом, все эти средства Android имеют свои преимущества и могут быть использованы в различных сценариях приложений IoT. Service и ForegroundService могут использоваться для считывания и передачи данных в фоновом режиме, WorkManager - для выполнения задач в заданное время, а Companion Service - для взаимодействия с внешним устройством. Дальнейшая комбинация ForegroundService и WorkManager будет использоваться при построении новой архитектуры приложения.

5 Исследование и построение решения задачи

5.1 Исходные данные

В ходе работы Чайчица Даниила [1] было разработано приложение, считывающее данные с умного жилета Hexoskin [16], сохраняющее данные в локальный буфер и отправляющее их на сервер. В ходе курсовой работы автора [2] была реализована поддержка различных типов датчиков, что позволило избавиться от привязки к Hexoskin.

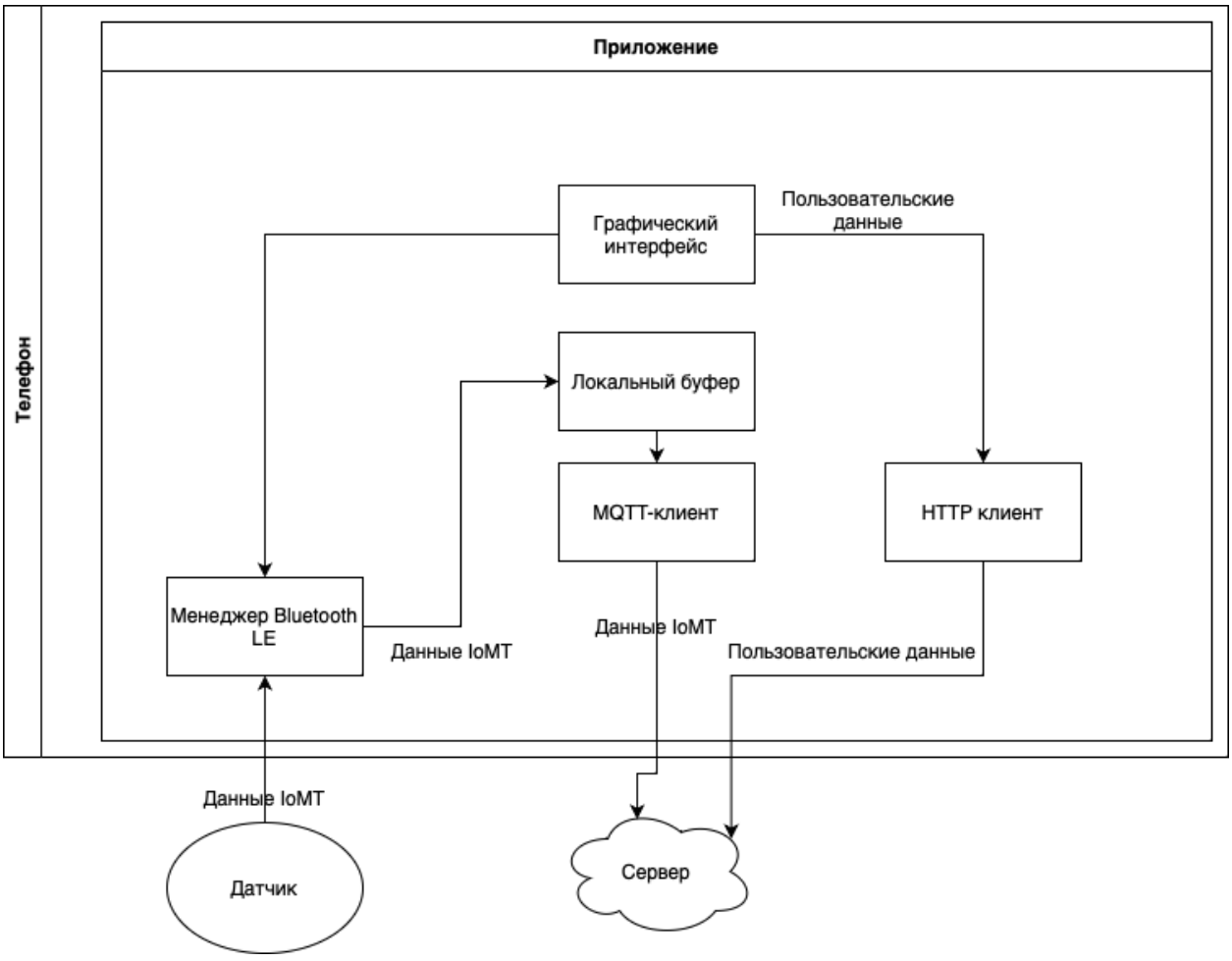


Рис. 1: Схема исходного приложения

На рисунке 1 представлена схема исходного приложения, рассмотрим её подробнее.

Пользователь, взаимодействуя с графическим интерфейсом приложения, подключается к датчику при помощи библиотеки `android.bluetooth` - стандартной библиотеки

Android для взаимодействия при помощи Bluetooth и Bluetooth LE. Считанные с датчика данные отображаются в графическом интерфейсе и сохраняются в локальный буфер.

Локальный буфер в данном случае представляет собой базу данных SQLite, взаимодействие с которой реализовано при помощи `android.database.sqlite.SQLiteDatabase` - стандартным средством Android, предоставляющим возможность писать SQL-запросы для управления базой данных. База данных содержит одну таблицу - данные, считанные с Hexoskin, помеченные временем считывания.

MQTT-клиент отправляет данные из локального буфера на сервер для дальнейшего хранения и обработки. Все пользовательские данные - логин, пароль, информация об устройствах и физиологические параметры пользователя (вес, рос, возраст) - передаются между клиентом и сервером при помощи протокола HTTP.

Данная реализация ведет к ограничению работы приложения в фоновом режиме, так как в новых версиях Android (начиная с версии 8.0) были введены ограничения на работу приложений в фоновом режиме (Background Execution Limits [12]), чтобы улучшить производительность устройств и увеличить время автономной работы. Система Android может ограничивать запуск фоновых служб после того, как пользователь выйдет из приложения. Это означает, что приложение не сможет продолжать работать в фоновом режиме и получать данные от датчиков, если они не связаны с пользовательским действием.

Для работы в фоновом режиме в новых версиях Android рекомендуется использовать специальные сервисы, такие как Foreground Service или WorkManager, которые позволяют запускать задачи в фоновом режиме, даже если приложение закрыто пользователем.

5.2 Архитектура сервисов приложения

5.2.1 BluetoothLeForegroundService

В новой схеме приложения следует учесть необходимость вынести все взаимодействия с `android.bluetooth` в сервис, который будет работать в фоновом режиме даже когда приложение выключено. В разделе 4 были рассмотрены различные типы сервисов, предоставляемых Android. В рамках задачи считывания данных по Bluetooth LE

в фоновом режиме необходимо все взаимодействие с Bluetooth LE инкапсулировать в ForegroundService.

Приложение инициализирует Bluetooth LE сервис, работающий в фоновом режиме, отправляет запрос на подключение к датчику. Менеджер Bluetooth LE отвечает за подключение к датчику, обработку считанных данных и передачу этих данных в локальный буфер при помощи клиента базы данных.

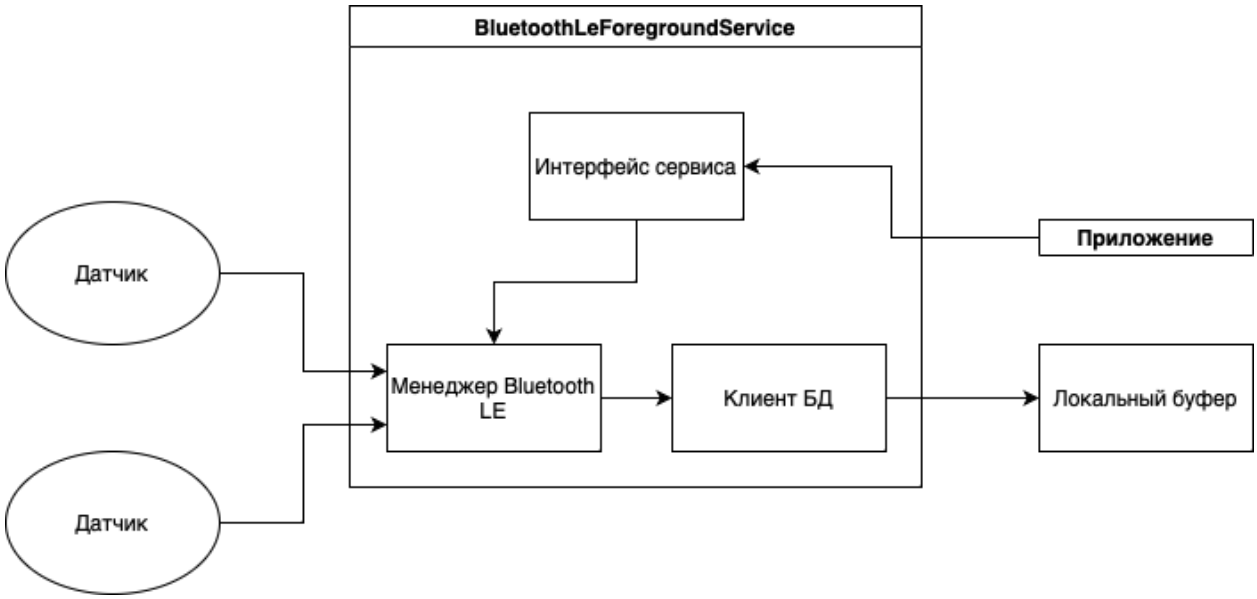


Рис. 2: Схема BluetoothLE-сервиса

5.2.2 MqttWorker

При разработке MQTT-сервиса необходимо учесть, с какой частотой данные должны синхронизироваться с сервером. В данном случае существует два варианта архитектуры:

1. ForegroundService позволит отправлять данные на сервер в момент, когда они появляются в базе данных. Данный способ гарантирует получение актуальных данных, взамен нагружая телефон, что приводит к повышенному расходу заряда батареи.

2. WorkManager в свою очередь служит для запуска фоновых задач, которые могут выполняться в определенное время или в ответ на определенные события, такие как изменение состояния сети или подключение к зарядному устройству, даже

если приложение закрыто или устройство перезагружено, что позволяет надежно отправлять данные на сервер. Также WorkManager может управлять задачами в зависимости от условий, таких как состояние сети, уровень заряда батареи и т.д., что позволяет оптимизировать использование ресурсов устройства. Так как WorkManager не поддерживает подключение к сети в режиме реального времени, возникают задержки при отправке данных на сервер.

Так как небольшие задержки в передаче данных допустимы, было принято решение использовать WorkManager в качестве основы MQTT-сервиса, что должно сэкономить заряд батареи.

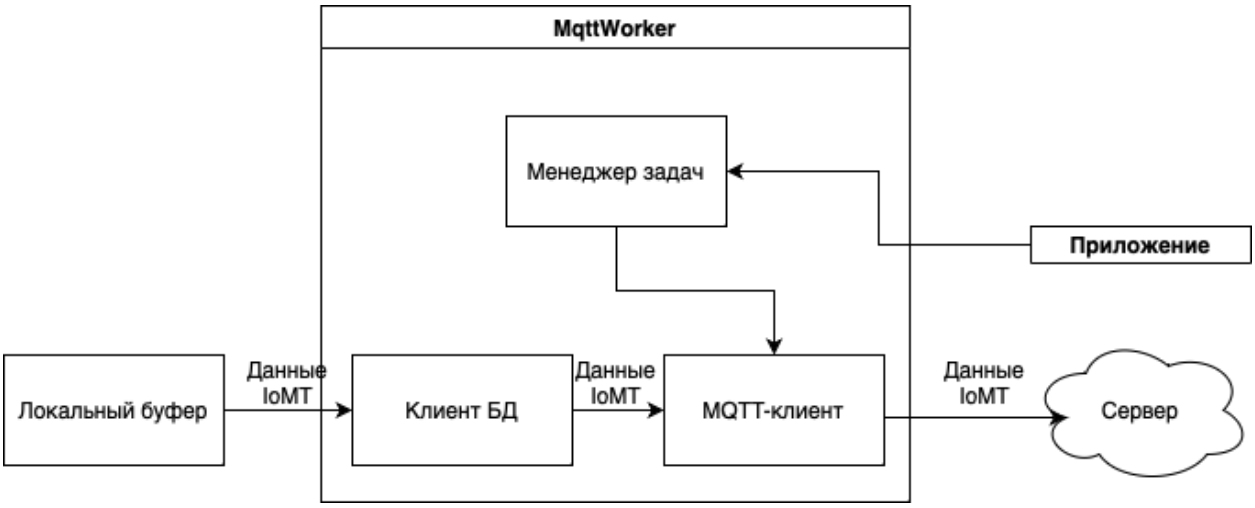


Рис. 3: Схема MQTT-сервиса

5.3 Новая архитектурная схема

В данном разделе была предложена схема устройства приложения (Рис. 4), позволяющая считывать данные с нескольких устройств одновременно в фоновом режиме и отправлять их на сервер с некоторой периодичностью (так же в фоновом режиме).

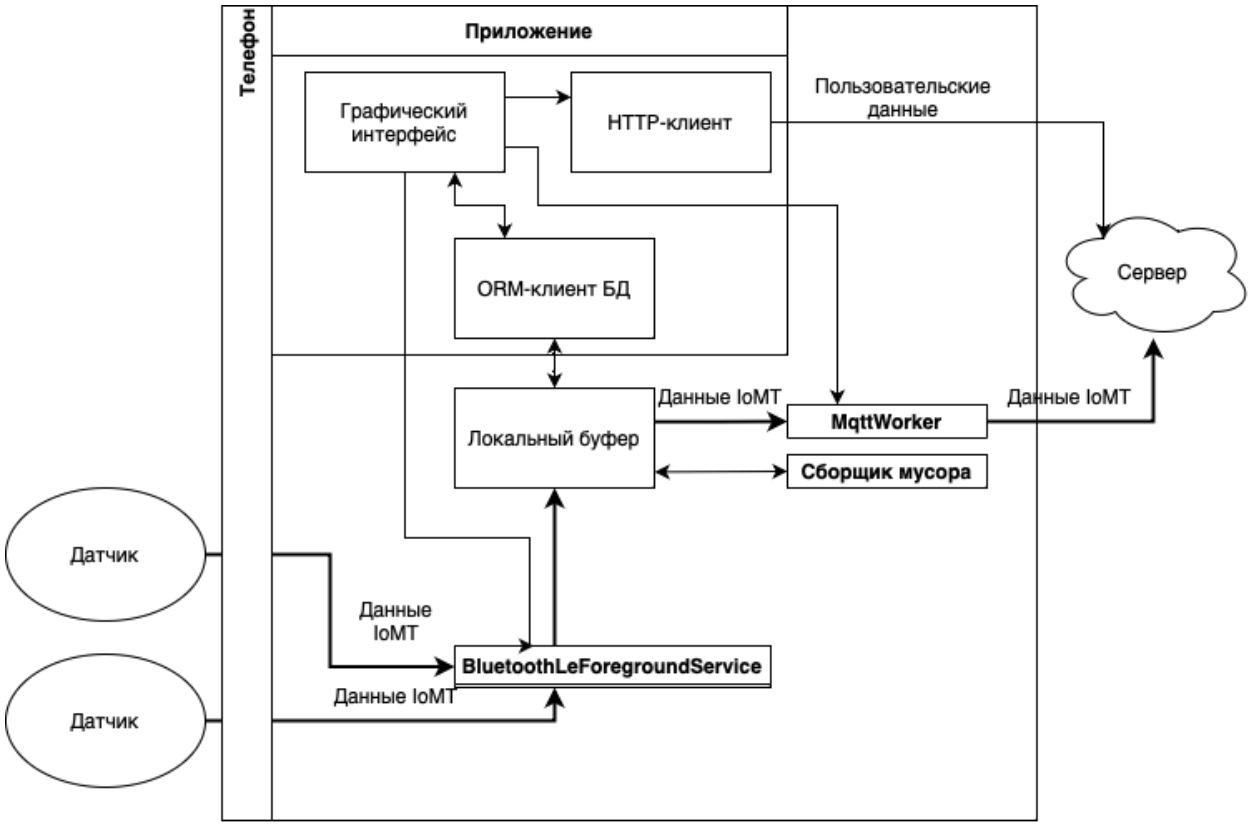


Рис. 4: Предложенная схема приложения

6 Описание реализации

6.1 Пользовательский интерфейс

Пользовательский интерфейс был реализован при помощи библиотеки Jetpack Compose [17] - части Android Jetpack, содержимое всех ранее созданных страниц интерфейса осталось без изменений. Jetpack Compose - это относительно новая UI-библиотека от Google, которая позволяет разрабатывать интерфейсы приложений на Android с помощью декларативного подхода.

Одним из главных преимуществ Jetpack Compose является то, что она позволяет ускорить процесс создания пользовательского интерфейса, уменьшив количество кода и улучшив его читаемость благодаря декларативному подходу. Благодаря использованию Kotlin, Jetpack Compose позволяет более эффективно использовать язык и облегчить работу с Android API.

Так как Jetpack Compose все еще находится на стадии разработки, библиотека может содержать некоторые ошибки и несовершенства. В настоящее время она также не поддерживает некоторые старые устройства и версии Android, поэтому ее использование может быть недоступно для части пользователей.

6.2 HTTP-клиент

6.2.1 Ktor

Для реализации сетевых взаимодействий был использован HTTP-клиент Ktor, разрабатываемый компанией JetBrains, создателем языка Kotlin. Ktor имеет множество дополнительных плагинов и пакетов, которые упрощают работу с HTTP-запросами, авторизацией, сериализацией данных.

6.2.2 Авторизация

В рамках данной работы был переработан процесс авторизации.

При аутентификации в приложении отправляется POST запрос, схема ответа на который приведена на листинге 1.

Листинг 1: Ответ на запрос авторизации

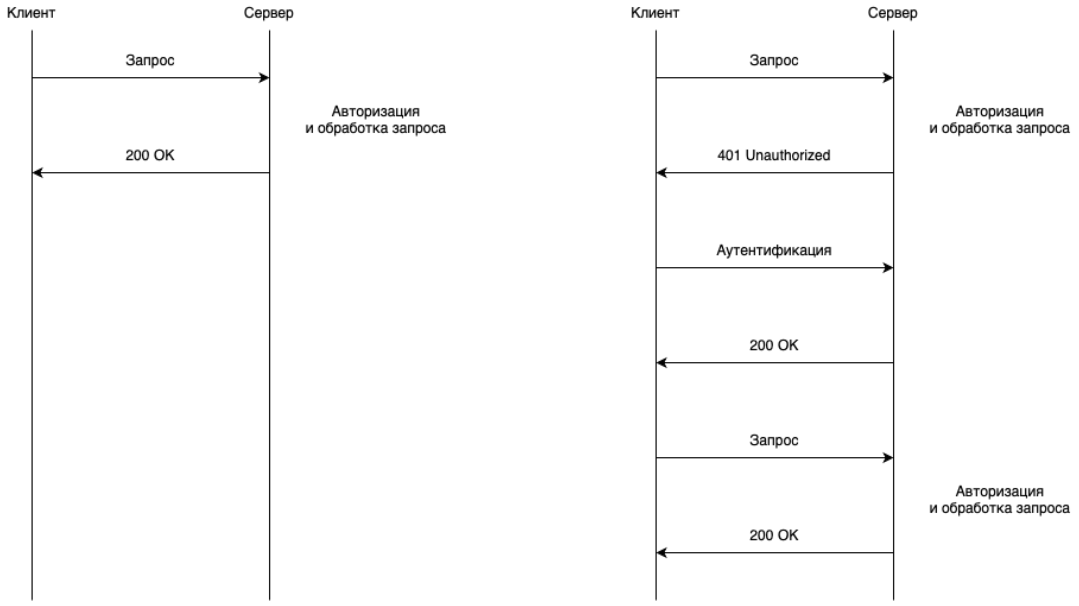
```
{
  "expires": "2023-07-20T20:11:27+00:00",
  "token": "JWT-TOKEN"
}
```

Ключ, полученный в ответ на данный запрос, необходимо добавлять в заголовок каждого последующего запроса в качестве ключа авторизации. Для этого был использован официальное расширение ktor - Auth [18]. Данное расширение позволяет настроить поведение ktor при ошибке `401 Unauthorized`.

Схема авторизации следующая:

1. Пользователь отправляет запрос на сервер;
2. Сервер проверяет наличие в запросе ключа аутентификации;
3. Если ключ присутствует, сервер проверяет его действительность и авторизует пользователя;
4. Если ключ отсутствует или недействителен, сервер отвечает соответствующей ошибкой аутентификации;
5. В случае ошибки аутентификации клиент отправляет запрос на авторизацию с целью обновить ключ авторизации;
6. При успешной авторизации клиент повторяет исходный запрос с обновленным ключом в заголовках;
7. После успешной аутентификации, сервер обрабатывает запрос пользователя и отправляет ответ.

Рисунок 5 иллюстрирует схему авторизации описанную ранее.



(a) Успешная авторизация

(b) Обновление ключа авторизации

Рис. 5: Схема авторизации

6.3 База данных

6.3.1 Библиотека Room

Room [19] - это библиотека для работы с базами данных SQLite на платформе Android, которая предоставляет удобный API для выполнения CRUD-операций (create, read, update, delete) и других запросов к базе данных. Особенность Room заключается в том, что она предоставляет абстракцию над SQLite, которая позволяет разработчикам писать более чистый и понятный код, упрощает управление базами данных, а также предоставляет механизмы для обработки конфликтов синхронизации данных и миграции схемы базы данных. Room использует аннотации для создания сущностей и запросов к базе данных, реализуя подход ORM.

ORM (Object-Relational Mapping) - это подход в программировании, который позволяет связывать объектно-ориентированный код с реляционной базой данных. С помощью ORM-библиотек можно работать с базой данных на языке программирования, используя объекты и методы, а не писать SQL-запросы вручную.

Основная задача ORM - облегчить процесс работы с базой данных и уменьшить

количество кода, необходимого для выполнения операций. С помощью ORM-библиотек можно создавать таблицы, изменять и удалять записи, выполнять поиск и фильтрацию данных, а также использовать связи между таблицами.

Основными достоинствами использования ORM являются:

- Упрощение процесса работы с базой данных и уменьшение объёма исходного кода, необходимого для выполнения операций;
- Повышение уровня абстракции при работе с данными;
- Ускорение разработки благодаря использованию готовых решений вместо написания своего собственного кода для работы с базой данных.
- Улучшение читаемости кода и его поддерживаемости, так как SQL-запросы заменяются на более понятный объектно-ориентированный код.

6.3.2 Схема базы данных

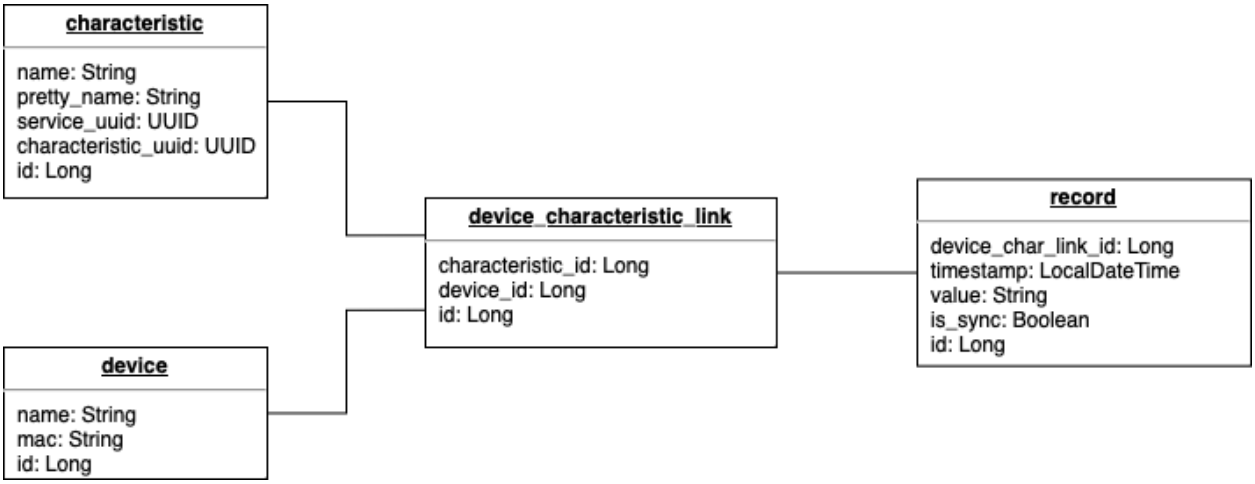


Рис. 6: Схема базы данных

Схема базы данных показана на рисунке 6. В базе данных созданы 2 независимые таблицы: `characteristic` и `device`.

В таблице `characteristic` хранится информация о характеристиках, которые когда-либо могли быть считаны. Данная таблица заполняется на основе конфигурационного

файла, полученного с сервера. Так как производители датчиков имеют право использовать свои UUID для стандартных характеристик, игнорируя предложения Bluetooth Special Interest Group, будем считать, что признак уникальности характеристики - имя и UUID характеристики.

В таблице `device` хранится информация о датчиках, с которых когда-либо могли быть считаны данные. Данная таблица заполняется при подключении устройства к приложению. В качестве уникального идентификатора датчика принят MAC-адрес.

Таблица `device_characteristic_link` связывает датчик и характеристику, которая может быть с него считана (она заполняется при выборе конфигурационного файла, который необходимо использовать для подключения к устройству). ID датчика и характеристики помечены как ForeignKey.

Таблица `record` содержит в себе информацию о считанных показателях с данного устройства по данной характеристике. Каждая запись помечается строкой содержащей дату и время считывания значения характеристики в формате ISO-8601.

6.4 Менеджер Bluetooth LE

Взаимодействие с устройствами при помощи Bluetooth LE реализовано при помощи класса BluetoothLeManager. Данный класс хранит подключенные устройства в виде изменяемого словаря, где ключ - MAC-адрес датчика, а значение - BluetoothGatt - класс стандартной библиотеки Android, хранящий в себе всю информацию связанную с подключением и взаимодействием с устройством при помощи BLuetooth Low Energy.

Дополнительная информация хранится в изменяемом словаре, где ключ - MAC-адрес, а значение - структура, поля которой - дополнительная информация о датчике, такая как конфигурационный файл, список сущностей "характеристика"и сущность "устройство".

Объект типа BluetoothLeManager хранится на уровне BluetoothLeForegroundService, гарантируя, работу с Bluetooth LE даже в случае закрытого приложения.

6.5 MQTT-клиент

Обмен медицинскими данными с сервером осуществляется при помощи класса MqttWorker. MqttWorker представляет собой задачу, которую можно запланировать при

помощи WorkManager, рассмотренного в разделе 4. Класс MqttWorkManager планирует периодическую задачу типа MqttWorker с частотой выполнения 15 минут (параметр конфигурируется), после чего каждые 15 минут из локального буфера извлекается список не синхронизированных данных, который будет отправлен на сервер после успешного подключения к брокеру MQTT. В случае успешной отправки одной записи, она помечается как синхронизированная. В случае ошибки при отправке данных, исполнение MqttWorker завершается с ошибкой. В случае успешной отправки всех записей, исполнение MqttWorker завершается успешно.

Каждая запись публикуется в топик с именем

```
c/{user_id}/{mac_address}/{characteristic}
```

где `user_id` - уникальный идентификатор пользователя, полученный от сервера, `characteristic` - название считанной характеристики, а `mac_address` - MAC адрес датчика, с которого были считаны данные.

Каждое сообщение имеет вид

```
{ "value": "Double", "timestamp": "LocalDateTime" }
```

где `value` - считанное значение, `timestamp` - дата и время записи в формате ISO-8601.

6.6 Документация

Программный код задокументирован в формате KDoc [20] - формате документации в Kotlin, основанном на Javadoc. KDoc позволяет описывать классы, функции и переменные, а также добавлять параметры, возвращаемые значения и примеры использования.

Библиотека dokka [21] позволяет автоматически генерировать документацию в формате HTML на основе KDoc-комментариев в коде Kotlin. При публикации кода в публичном репозитории на GitHub, запускается автоматическая генерация документации с последующим разворачиванием сайта на основе сгенерированной документации на базе GitHub [22].

7 Экспериментальное исследование

7.1 Тестирование на долговечность

Тестирование на долговечность позволяет проверить приложение на отсутствие сбоев и прерываний в течение продолжительного времени.

Приложение должно в течение суток считывать данные и отправлять их на сервер без сбоев. При этом испытуемый ведет привычный образ жизни. Приложение было запущено на телефоне Google Pixel 6 (версия Android - UpsideDownCake), телефон использовался обычным образом.

Для проверки отсутствия сбоев в течение продолжительного времени была реализована периодическая задача с периодом 30 минут. Данная задача считывает общее число записей в таблице `record`, а также число отправленных на сервер записей (is_sync помечен true) и сохраняет эти числа в специальную таблицу `statistics`.

Данные - частота сердечных сокращений - в течение 25 часов считывались с умного браслета Mi Band 6, произведенного компанией Xiaomi. Интервал подсчета количества записей - 30 минут. Графики, соответствующие количеству записей в базе и количеству добавленных записей (с прошедшего подсчета) представлены на рисунке 7. Синим цветом помечены синхронизированные данные, красным - все данные.

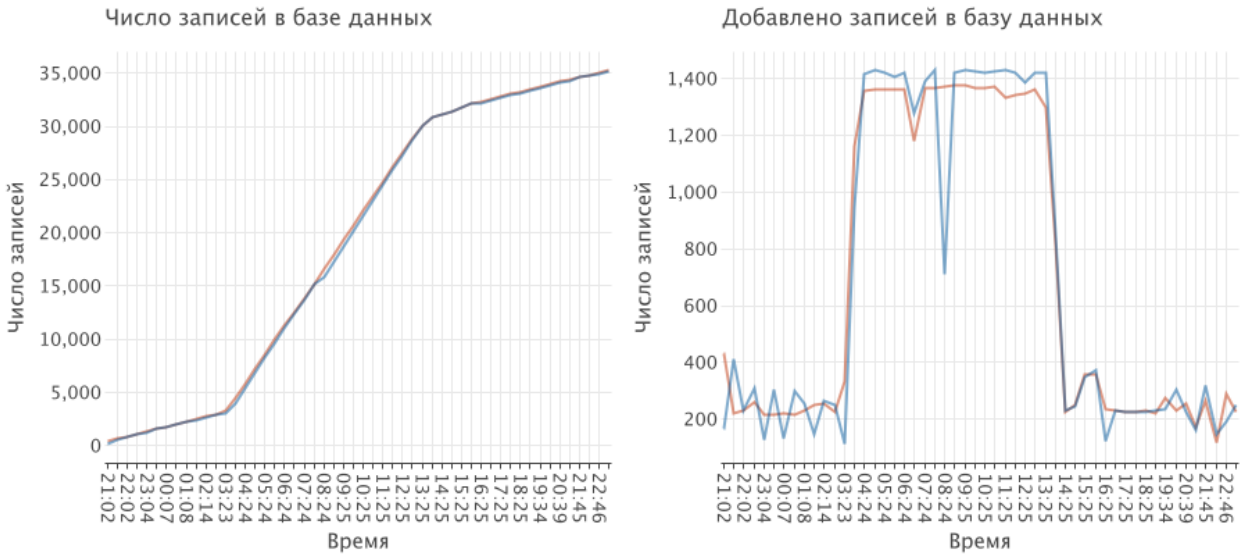


Рис. 7: Тестирование на долговечность

Таким образом, в течение 24 часов все считанные данные были успешно синхрони-

зированы. Стоит отметить, что на браслете была включена функция анализа качества сна, в связи с чем во время сна (с 3:25 до 13:55) сердцебиение считывалось в 7 раз чаще.

Синий график имеет резкий скачок в 08:24, связанный с с тем, что задача сбора статистики началась во время синхронизации.

Использование приложением заряда батареи, в соответствии со сведениями из настроек Android, на протяжении суток составило менее одного процента от полного заряда.

7.2 Тестирование на масштабируемость

Тестирование на масштабируемость поможет определить, насколько приложение способно к работе с большим количеством датчиков.

В ходе считывания данных приложением с датчиков, время считывания характеристики принимается равным времени начала обработки этой характеристики на стороне приложения. В связи с этим, необходимо оценить существующие задержки между отправкой датчиком данных по BLE и обработкой этих данных приложением (задержки считывания). Для проведения исследования использовалось приложение `TestApp` [23] (далее - тестовое приложение), реализованное в рамках курсовой работы студентки 3 курса Голубковой Марии.

Тестовое приложение позволяет генерировать данные из определенного числового интервала и отправлять их тестируемому приложению при помощи Bluetooth LE. Это позволяет эмулировать поведение реального датчика. Кроме того, тестовое приложение фиксирует время отправки данных.

В рамках данного исследования необходимо изучить изменение задержек считывания в зависимости от числа подключенных устройств. Для этого в течение 15 минут проводилось считывание данные с одного или нескольких тестовых приложений (частота отправки данных - 1 запись в секунду) и фиксировалась разность между временем отправки и временем начала обработки.

Исследование проводится в три этапа:

1. Считывание данных с одного тестового приложения, запущенного на телефоне Pixel 5;

2. Считывание данных с двух тестовых приложений: запущенного на телефоне Pixel

5 и запущенного на телефоне Pixel 3;

3. Считывание данных с трех тестовых приложений: запущенного на телефоне Pixel 5, запущенного на телефоне Pixel 3 и и запущенного на телефоне BQ-5046 Choice.

Ожидается, что возможность подключения 3 устройств покрывает большинство потребностей со стороны медицинских учреждений, так как при подключении большего числа устройств проведение медицинских экспериментов и замеров становится затруднительным.

7.2.1 Одно устройство

На рисунке 8 изображена диаграмма размаха для задержки считывания данных с тестового приложения, запущенного на телефоне Pixel 5. Медианное значение задержки считывания равно 722,3 миллисекундам. На диаграмме присутствуют выбросы, объясняемые другими запущенными приложениями как на телефоне с тестовым приложением, так и на стороне с тестируемым.

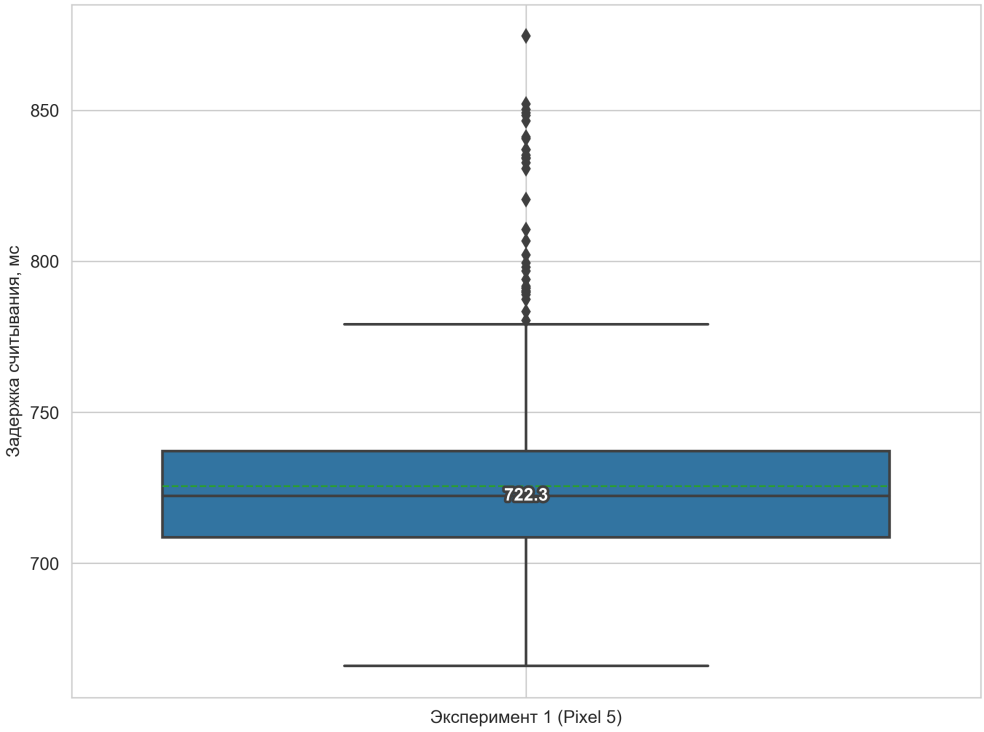


Рис. 8: Задержка считывания данных с одного устройства

7.2.2 Два устройства

На рисунке 9 изображена диаграмма размаха для задержки считывания данных с тестового приложения, запущенного на телефоне Pixel 5 и Pixel 3. Медианное значение задержки считывания с Pixel 5 (при параллельном считывании с Pixel 3) равно 740,9 миллисекундам, что свидетельствует о росте задержки по сравнению с задержками при считывании только лишь с Pixel 5. Рост задержки для Pixel 5 при подключении другого устройства составляет 18,6 миллисекунд.

Медианное значение задержки считывания с Pixel 3 составляет 983,8 мс, что больше, чем медианное значение задержки считывания с Pixel 5. Причины такого явления не исследовались, но предположительно это связано с тем, что Pixel 5 является более новым устройством (с более новой версией Android), в связи с чем он может исполнять тестовое приложение быстрее, чем Pixel 3, а значит время отправки является более точным.

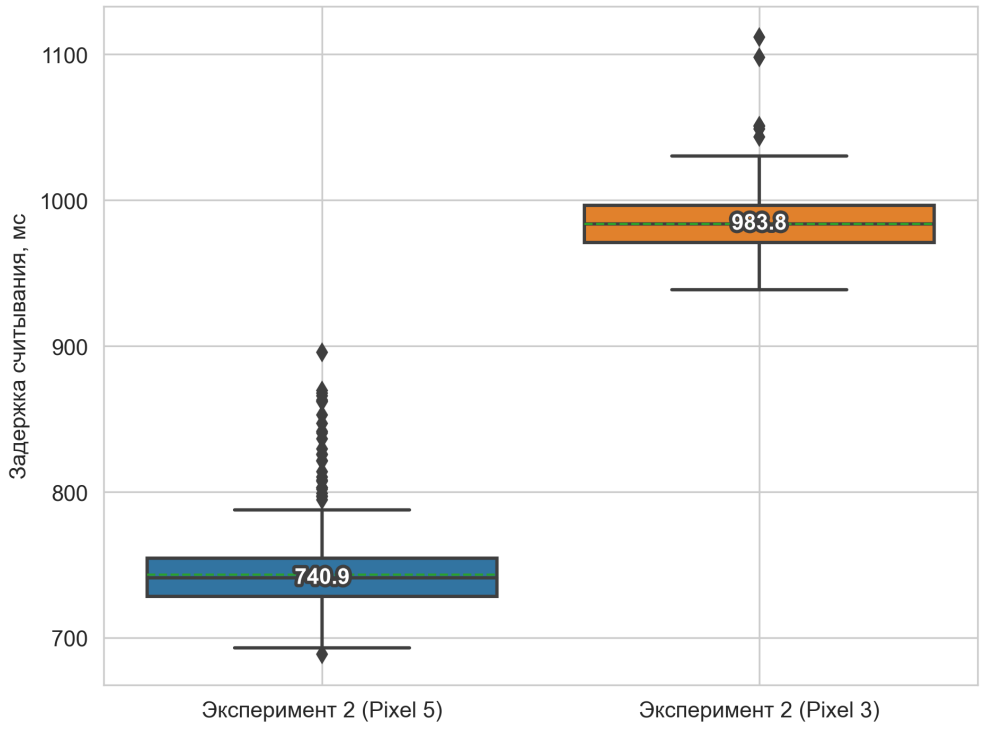


Рис. 9: Задержка считывания данных с двух устройств

7.2.3 Три устройства

На рисунке 10 изображена диаграмма размаха для задержки считывания данных с тестового приложения, запущенного на телефоне Pixel 5, Pixel 3 и BQ Choice. Медианное значение задержки считывания с Pixel 5 на данном этапе равно 816,8 миллисекундам, что свидетельствует о росте задержки по сравнению с предыдущими этапами. Рост задержки для Pixel 5 на данном этапе составляет примерно 75,9 миллисекунд.

Медианное значение задержки считывания с Pixel 3 на данном этапе равно 1062,0 миллисекундам, что свидетельствует о росте задержки по сравнению с предыдущим этапом. Рост задержки для Pixel 3 на данном этапе составляет примерно 78,2 миллисекунд, что свидетельствует о росте задержки по сравнению с предыдущим этапом.

Медианное значение задержки считывания с BQ Choice на данном этапе равно 1225,1 миллисекундам, что больше, чем медианное значение задержки считывания с других устройств (объясняется аналогично).

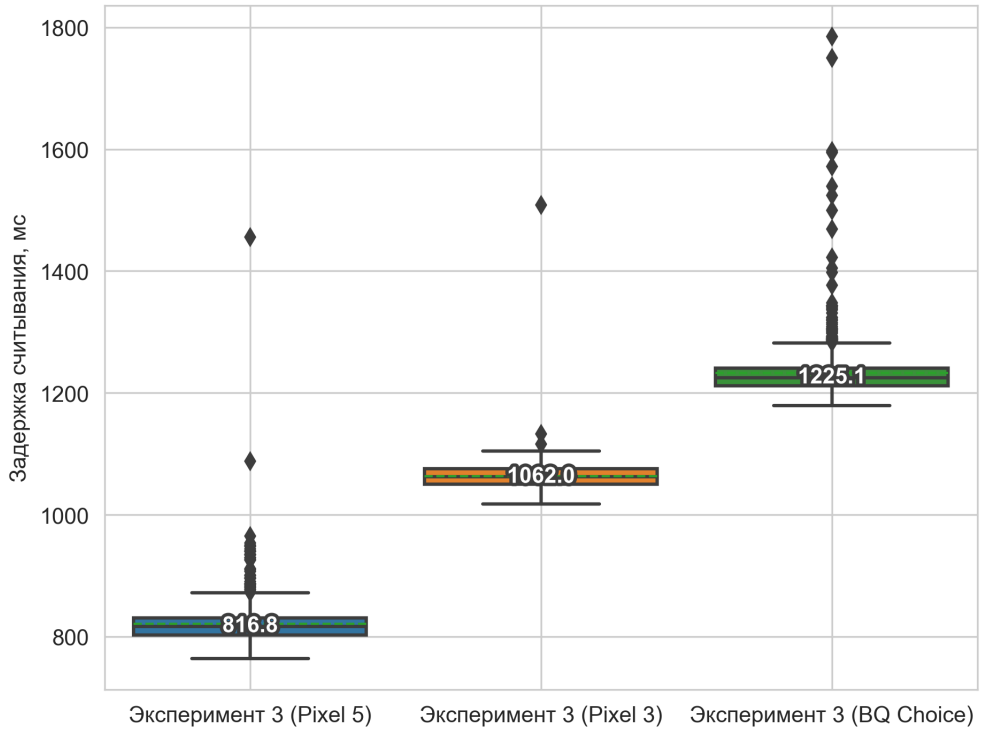


Рис. 10: Задержка считывания данных с трех устройств

7.2.4 Вывод

При подключении большего числа устройств задержки ожидаемо растут, причем рост задержек составляет менее 10% медианного значения. Таким образом, рост задержек при подключении большего числа устройств является достаточно малым при подключении ожидаемого числа датчиков - 3.

Стоит отметить, что на каждом из приведенный графиков среднее значение близко к медиане, что свидетельствует о нормальном распределении (при отбрасывании выбросов, связанных с работой сторонних приложений на тестовом устройстве). Это может свидетельствовать о стабильности системы, обработка пакетов происходит предсказуемо, без значительных искажений в одну или другую сторону.

8 Заключение

В результате данной выпускной квалификационной работы была улучшена реализованная автором ранее клиентская часть открытой платформы для сбора и обработки медицинской телеметрии. Поставленные задачи выполнены в полном объеме.

В ходе работы были изучены существующие протоколы обмена данными, а также архитектурные решения для проектирования приложений для операционной системы Android, была предложена архитектура, позволяющая одновременно считывать данные с 3 различных датчиков интернета вещей в фоновом режиме с буферизацией данных локально и последующей отправкой этих данных на сервер, проведено экспериментальное исследование работоспособности приложения в части задержек считывания данных с датчика и долгосрочного использования приложения.

Исходный код задокументирован, документация опубликована, улучшенная версия приложения выложена в открытый доступ.

Список литературы

[1] *Чайчиц, Д. А.* Разработка и реализация клиентской части платформы для сбора и обработки медицинской телеметрии. — 2021.

[2] *Фролов, А. В.* Развитие клиентской части сервиса сбора и обработки медицинской телеметрии. — 2022.

[3] *International Business Machines Corporation (IBM), Eurotech.* MQTT V3.1 Protocol Specification. — 2022. — 14.04.2023. https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.

[4] *Microsoft.* Описание способа передачи данных "подписчик-издатель". — 2023. — Последнее посещение 03.04.2023. https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber.

[5] *Microsoft.* Описание способа передачи данных "запрос-ответ". — 2023. — Последнее посещение 03.04.2023. https://learn.microsoft.com/en-us/azure/architecture/patterns/async-request-reply.

[6] *Nielsen, Henrik.* Hypertext Transfer Protocol – HTTP/1.1. — RFC 2616. — 1999. — . https://www.rfc-editor.org/info/rfc2616.

[7] *Shelby, Zach.* The Constrained Application Protocol (CoAP). — RFC 7252. — 2014. — . https://www.rfc-editor.org/info/rfc7252.

[8] *Standard, OASIS.* OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0 Part 0: Overview. — 29 October 2012. — Последнее посещение 14.04.2023. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html.

[9] *Nicholas, Stephen.* Сравнение энергопотребления MQTT и HTTPS. — Последнее посещение 15.04.2023. http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https.

[10] *Google.* Services overview. — 2022. — Последнее посещение 23.04.2023. https://developer.android.com/guide/components/services.

[11] *Google*. Service reference. — 2023. — Последнее посещение 23.04.2023. https://developer.android.com/reference/android/app/Service.

[12] *Google*. Background Execution Limits. — 2021. — Последнее посещение 23.04.2023. https://developer.android.com/about/versions/oreo/background.

[13] *Google*. Foreground services. — 2023. — Последнее посещение 23.04.2023. https://developer.android.com/guide/components/foreground-services.

[14] *Google*. Service reference. — 2023. — Последнее посещение 23.04.2023. https://developer.android.com/topic/libraries/architecture/workmanager.

[15] *Google*. CompanionDeviceService reference. — 2023. — Последнее посещение 23.04.2023. https://developer.android.com/reference/android/companion/CompanionDeviceService.

[16] *INC, CARRE TECHNOLOGIES*. Hexoskin. — 2022. — Последнее посещение 20.12.2022. https://www.hexoskin.com.

[17] *Google*. Jetpack Compose. — Последнее посещение 25.02.2023. https://developer.android.com/jetpack/compose.

[18] *JetBrains*. Ktor Authorization Plugin. — Последнее посещение 25.02.2023. https://ktor.io/docs/bearer-client.html.

[19] *Google*. Room. — Последнее посещение 25.03.2023. https://developer.android.com/jetpack/androidx/releases/room.

[20] *JetBrains*. Kotlin Docs. — 2022. — Последнее посещение 20.05.2022. https://kotlinlang.org/docs/home.html.

[21] *JetBrains*. Dokka. — Последнее посещение 25.03.2023. https://kotlinlang.org/docs/dokka-introduction.html.

[22] *А.В., Фролов*. Исходный код IoMT Android. — 2022-2023. https://github.com/IoMT-LVK/iomt-android.

[23] *Голубкова, М. С.* Разработка Android приложения, имитирующего датчики интернета вещей. — 2023. — В процессе публикации.

[24] *Arslan, Reis Burak*. A Wearable System Implementation for the Internet of Medical Things (IoMT) / Reis Burak Arslan, Çağri Candan // 2022 44th Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC). — 2022. — Pp. 2447–2450.

Приложение

А Итоговая версия схемы архитектуры приложения

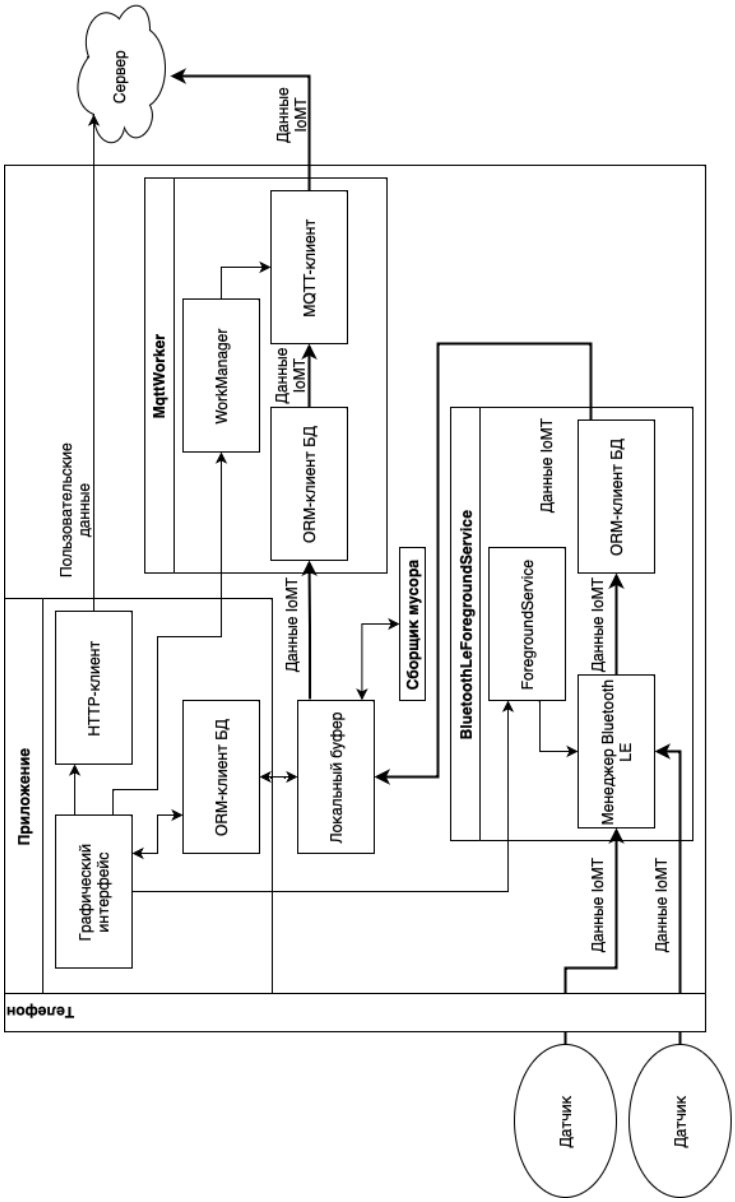
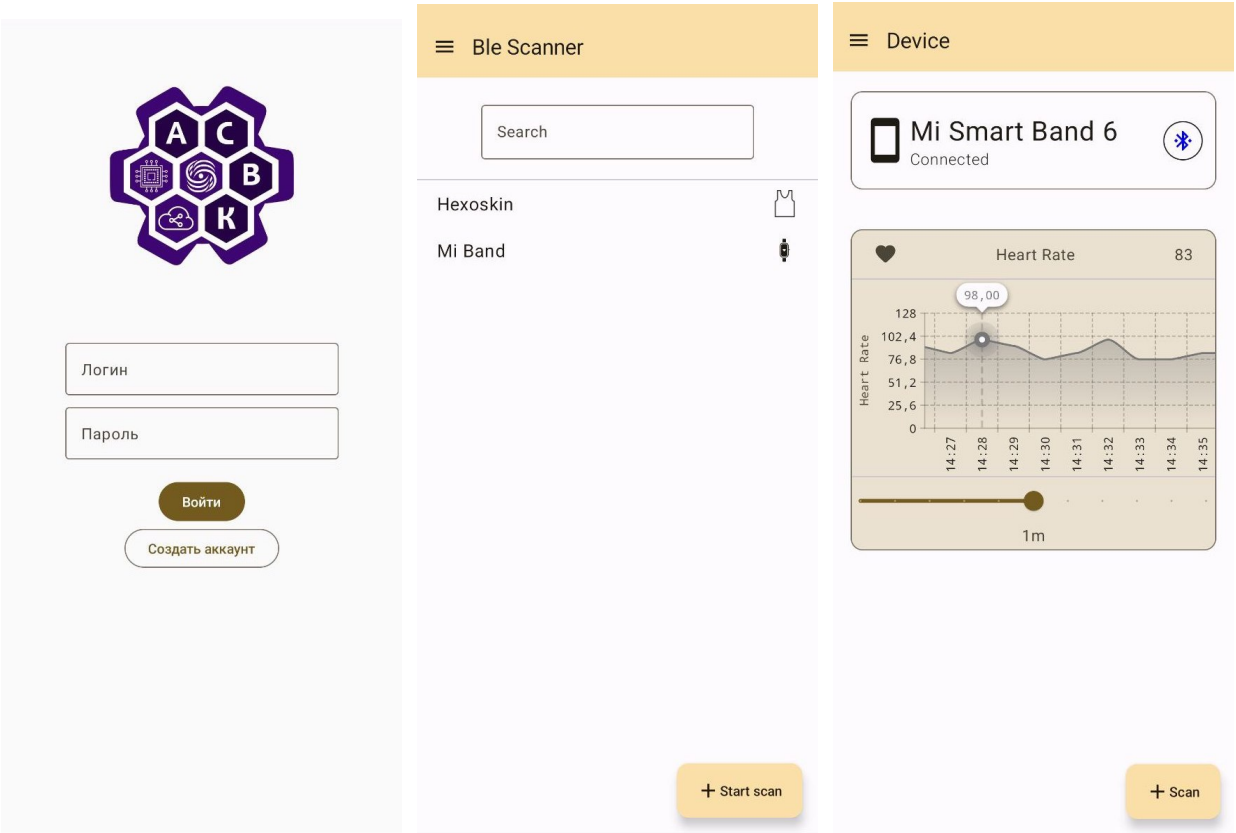


Рис. 11: Схема MQTT-сервиса

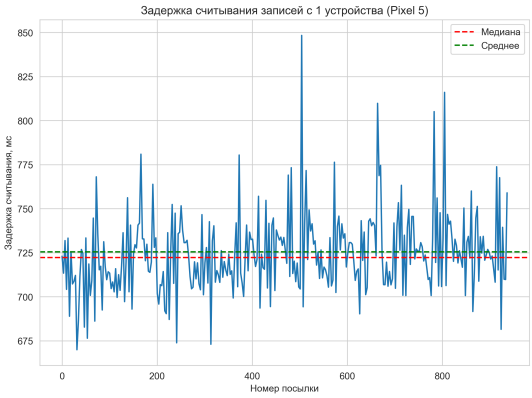
Б Некоторые страницы графического интерфейса приложения



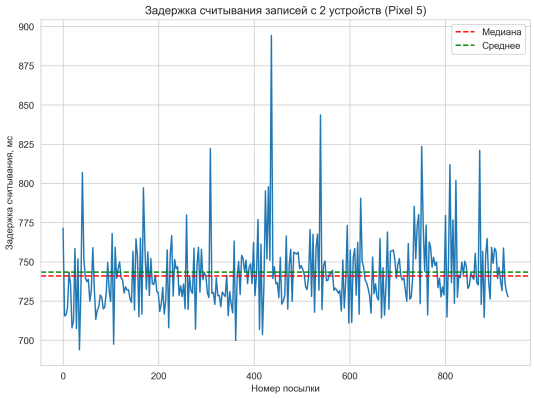
(a) Авторизация (b) Выбор конфигурационного файла (c) Устройство

Рис. 12: Некоторые страницы графического интерфейса приложения

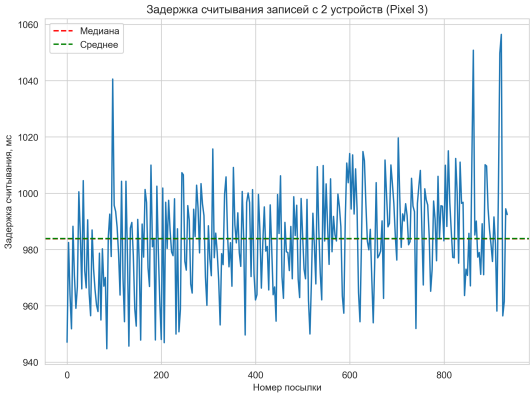
B Подробные графики задержек



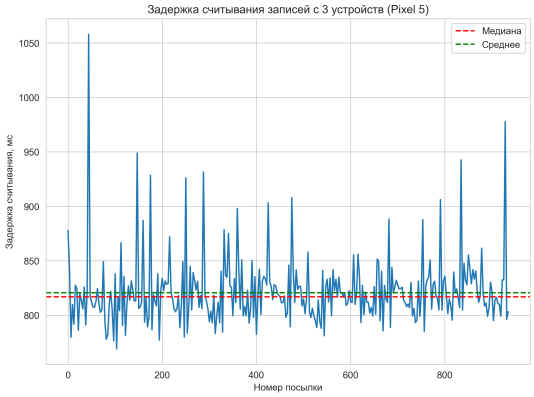
(a) Этап 1 - Pixel 5



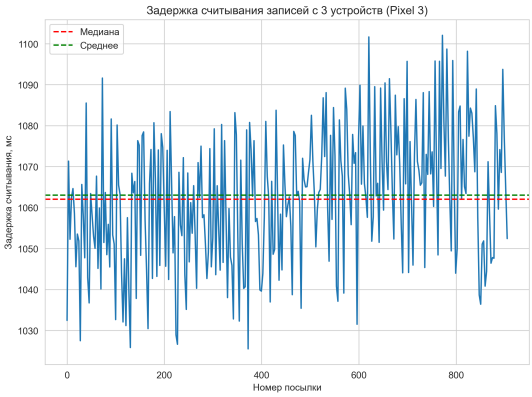
(b) Этап 2 - Pixel 5



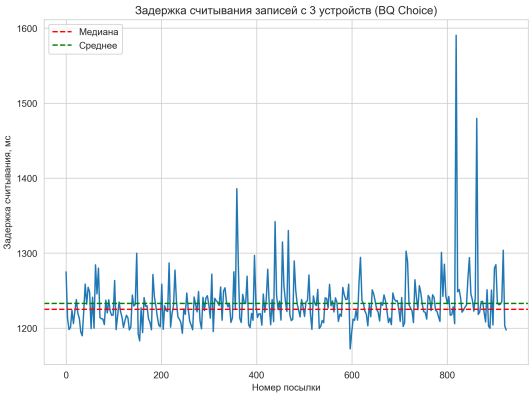
(c) Этап 2 - Pixel 3



(d) Этап 3 - Pixel 5



(e) Этап 3 - Pixel 3



(f) Этап 3 - BQ Choice

Рис. 13: Подробные графики задержек считывания