МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2022 г.

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ МОНИТОРИНГА СОСТОЯНИЯ ОРГАНИЗМА ВО ВРЕМЯ ПРОВЕДЕНИЯ ЗАНЯТИЙ ПО ФИЗКУЛЬТУРЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-09.03.2022.216 ПЗ ВКР

Руководитель работы,
к.т.н., зав. каф. ЭВМ
_____Д.В. Топольский
«___»_____2022 г.

Автор работы,
студент группы КЭ-405
_____А.Е. Колмаков
«___»_____2022 г.

Нормоконтролёр,
к.п.н., доцент каф. ЭВМ
_____М.А. Алтухова
«___»_____2022 г.

Челябинск-2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2022 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Колмакову Антону Евгеньевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1) **Тема работы:** «Программный комплекс для мониторинга состояния организма во время проведения занятий по физкультуре» утверждена приказом по университету от 12 декабря 2022 г. №308/141

2) **Срок сдачи студентом законченной работы:** 1 июня 2022 г.

3) **Исходные данные к работе:**

– операционная система Android;

– язык программирования Kotlin;

– комплект разработки программного обеспечения Android SDK;

– база данных SQLite;

– библиотека Room для работы с базой данных;

– среда разработки Android Studio;

– технология Bluetooth Low Energy.

4) **Перечень подлежащих разработке вопросов:**

– рассмотрение существующих проектов по считыванию и анализу показаний с устройств для мониторинга состояния организма при проведении занятий по физической культуре;

– анализ современных программных технологий считывания и анализа физических показателей с носимых устройств мониторинга состояния организма при проведении занятий по физической культуре;

– проектирование и разработка собственного программного комплекса по считыванию и анализу показаний с устройств для мониторинга состояния организма при проведении занятий по физической культуре;

– оценка соответствия разработанного программного комплекса предъявленным требованиям.

5) **Дата выдачи задания:** 1 декабря 2021 г.

Руководитель работы _____/*Д.В. Топольский*/

Студент _____/*А.Е. Колмаков*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	10.03.2022	
Разработка модели, проектирование		
Определение функциональных требований	13.03.2022	
Определение нефункциональных требований	15.03.2022	
Проектирование архитектуры	17.03.2022	
Разработка алгоритмов решения задач	19.03.2022	
Описание данных	21.03.2022	
Реализация системы		
Реализация интерфейса	28.03.2022	
Реализация классов	04.04.2022	
Тестирование, отладка, эксперименты		
Проведение тестирования	7.04.2022	
Отладка программы	11.04.2022	
Проведение экспериментов	25.04.2022	

Этап	Срок сдачи	Подпись руководителя
Компоновка текста работы и сдача на нормоконтроль	16.05.2022	
Подготовка презентации и доклада	24.05.2022	

Руководитель работы _____/*Д.В. Топольский*/

Студент _____/*А.Е. Колмаков*/

АННОТАЦИЯ

А. Е. Колмаков. Программный комплекс для мониторинга состояния организма во время проведения занятий по физкультуре. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2022, 119 с., 38 ил., библиогр. список – 36 наим.

В рамках выпускной квалификационной работы проводится анализ программных продуктов для чтения и анализа данных с портативных носимых устройств, считывающих параметры организма во время физических нагрузок. На основании проведенного анализа конкурирующих продуктов предлагаются оптимальные технические решения, требуемые для реализации программного комплекса, выдвигаются функциональные и нефункциональные требования. С учетом предложенных технологических решений, функциональных и нефункциональных требований проектируется и реализуется программный комплекс. Результатом работы является мобильное приложение, отвечающее всем поставленным требованиям, что было доказано на этапе тестирования.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

По данным Всемирной Организации Здравоохранения, на долю сердечно-сосудистых заболеваний приходится 16% от общего числа смертей в мире, что составляет около 9 миллионов смертей ежегодно. Это наиболее распространенное и опасное заболевание человечества [1]. Люди, подверженные высокому риску таких заболеваний, как правило: ведут малоактивный образ жизни, неправильно питаются и имеют лишний вес. Ситуацию также усугубляет эпидемия новой коронавирусной инфекции, поскольку методы, используемые для снижения числа новых заболеваний, предполагают самоизоляцию и снижение социальной активности населения, что приводит к уменьшению физической нагрузки на организм [2].

Актуальность темы, связанной с разработкой программного комплекса для мониторинга состояния организма во время проведения занятий по физкультуре, обусловлена тем, что в последние годы участились случаи плохого самочувствия обучающихся во время занятий по физической культуре [3]. Вышеописанные факторы выступают косвенной причиной возникновения данного явления.

В этой связи возникла потребность в отслеживании состояния обучающихся в процессе физической активности с целью недопущения перегрузок и своевременной коррекции проводимого комплекса упражнений с учетом индивидуальных особенностей обучающихся.

Создание приложения для мониторинга состояния обучающихся на уроках физической культуры технологически многоаспектная задача, требующая глубокого анализа различных технологий для получения ожидаемого результата. Для выявления основных технологических тенденций и выбора наиболее подходящих необходимо рассмотреть аналогичные программные решения.

Целью представленной выпускной квалификационной работы является разработка программного комплекса для операционной системы Android,

обеспечивающего получение и обработку данных с множества устройств для мониторинга состояния организма во время проведения занятий по физкультуре.

Для достижения поставленной цели необходимо решить следующие задачи:

1) рассмотреть существующие аналоги программно-аппаратных комплексов, агрегирующих физиологические показатели группы спортсменов в одно устройство, проанализировать используемые технологические решения и предложить наиболее походящие решения для реализации поставленной цели;

2) определить основные требования, предъявляемые к разрабатываемому программному комплексу;

3) спроектировать архитектуру программного комплекса;

4) реализовать программный комплекс для мониторинга состояния организма во время проведения занятий по физкультуре с соблюдением предложенных архитектурных и технологических решений;

5) определить методики и провести тестирование программного комплекса.

Новизна разрабатываемого программного комплекса заключается в реализации технологии считывания данных с множества устройств (фитнес-трекеров, кардиодатчиков) одновременно, в информативном и удобном визуальном представлении в реальном времени физиологических данных об учениках, в системе оповещений преподавателя о превышении нормативных показаний.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

В настоящее время на рынке существуют аналогичные по предлагаемому автором функционалу программные продукты в составе коммерческих аппаратно-программных комплексов. На основании имеющейся в открытом доступе информации, рассмотрим данные программные решения более подробно с точки зрения предлагаемого функционала, ценовой политики и используемых технологий.

Выделим основные критерии, на которые будем обращать внимание при обзоре аналогов:

– системные требования;

– технологии передачи данных;

– поддерживаемые беспроводные устройства для считывания физиологических показателей;

– количество опрашиваемых устройств;

– отображаемые показатели;

– ценовая политика;

– страна производства.

Polar Club (рисунок 1) – решение для отслеживания частоты сердечных сокращений (далее, ЧСС) во время групповых занятий фитнесом от известной финской компании Polar Electro, разработавшей первый в мире беспроводной пульсометр [4]. Система Polar Club позволяет в режиме реального времени отслеживать и отображать показатели группы пользователей: ЧСС, количество сожженных калорий и зоны ЧСС (цвет и процент от максимальной нагрузки). Приложение имеет пользовательскую оценку 4,9 из 5 (на основании 12 отзывов), требует iPad с iPadOS 12.0 или более поздней версией с подключением к Интернету [5]. Polar Club получает данные об участниках тренировки (до 90 подключений) с сервера Polar. Для этого каждый участник

занятия должен иметь зарегистрированную учетную запись Polar Flow и подключенный датчик Polar к личному Android/iOS устройству с доступом в Интернет. Стоимость лицензии на ПО составляет 185900 рублей в год [6], имеется возможность активации 30 дневного пробного периода.

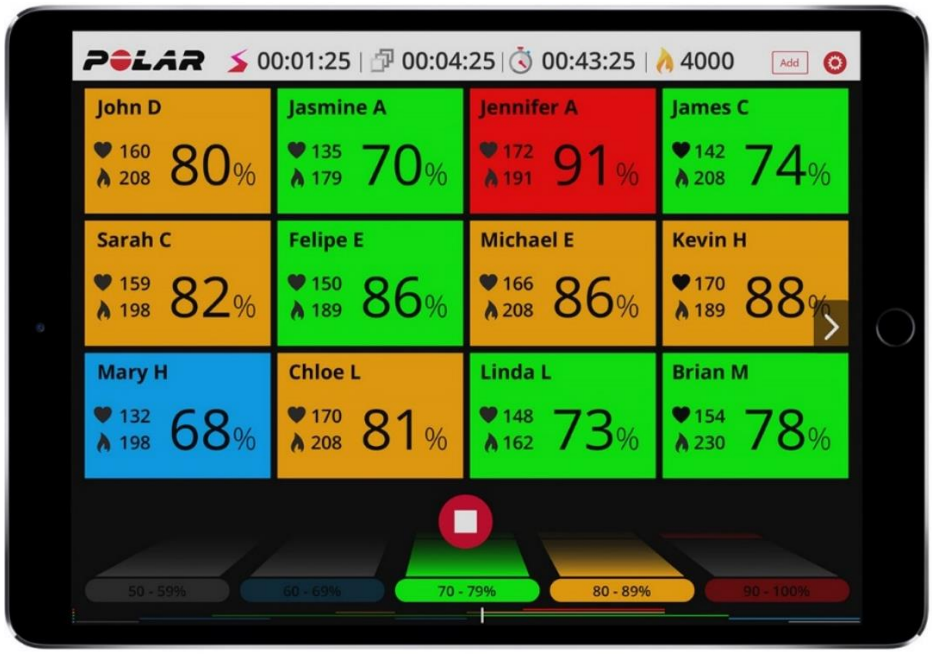

Рисунок 1 – Приложение Polar Club

Myzone In Club [7] (рисунок 2) – готовое решение для фитнес-клубов американской компании Myzone Group, включающее в себя:

– stick PC (Intel Compute Stick с ОС Windows и ПО Myzone), который необходимо подключить к внешнему дисплею с поддержкой сенсорного ввода (не входит в комплект поставки);

– ресивер MZ-R для приема ANT+ сигнала [8] с датчиков MZ-1 или MZ-3;

– комплект датчиков активности (на выбор).

На экране приложение отображает информацию о текущей ЧСС, цветом показывает интенсивность пульса, рассчитывает баллы усилий Myzone Effort Points [9] (количество баллов определяется с учетом времени, проведенного в разных зонах интенсивности), рассчитывает количество сожженных калорий [8] (расчет производится исходя из физических показателей пользователя: пола, возраста, веса и пульса). На сайте компании отсутствует

информация о стоимости лицензии на ПО, для получения конечной цены комплекта и лицензионного сбора необходимо обращаться к менеджеру по продажам фирмы Myzone. В силу недоступности ПО в магазинах приложений пользовательская оценка отсутствует.

Рисунок 2 – Приложение Myzone In Club

OnBeat (рисунок 3) – кардио-приложение американской компании OnBeat LLC для группового занятия фитнесом, отображающее в реальном времени ЧСС, зону сердечного ритма, процент от максимальной ЧСС, средний пульс, количество потеряных калорий [10]. Приложение совместимо с iPhone (iOS 9.3 или новее), iPad (iPadOS 9.3 или новее), iPod touch (iOS 9.3 или новее) с и Mac (macOS 11.0 или новее), имеет пользовательскую оценку 4,7 (на основании 18 отзывов) [11], позволяет подключать до 12 пульсометров, использующих технологию Bluetooth LE, одновременно без использования дополнительного оборудования. Для подключения большего числа устройств необходимо использовать дополнительную внешнюю ANT+ антенну. Приложение совместимо со всеми датчиками сердечного ритма, использующих технологии Bluetooth LE или ANT+, за исключением FitBit и AppleWatch [12]. ПО доступно по ежемесячной подписке, цена подписки зависит от количества одновременно подключаемых устройств, диапазон цен:

от 4000 рублей в месяц за 8 устройств и до 9000 рублей в месяц за 100 устройств [13].

Рисунок 3 – Приложение OnBeat

SelfLoops Group Fitness (рисунок 4) – продукт итальянской компании SelfLoops Snc., позволяющий проводить групповые занятия с отслеживанием сердечного ритма спортсменов. Приложение является кроссплатформенным, имеются версии для Apple TV (tvOS 13.0 и выше), iPad (iPadOS 13.0 и выше), Android (6.0 и выше) [14]. Ни одна из версий приложения не имеет отзывов в магазинах приложений. Поддерживаются мониторы сердечного ритма с технологией ANT+, Apple Watch (для Apple TV и iPad). Для работы приложения с ANT+ устройствами необходимо подключить внешний беспроводной ANT+ передатчик – Selfloops Repeater, который способен обеспечить одновременный прием сигнала от 40 датчиков [15]. Данные о спортсменах в режиме реального времени отображаются в виде таблицы, в которой указаны: пульс, количество сожженных калорий, процент от максимально допустимой нагрузки, рассчитываемый с учетом возраста пользователя. Публично доступных данных о стоимости лицензии нет, для получения цены на ПО необходимо обращаться к менеджеру компании SelfLoops.

Рисунок 4 – Приложение SelfLoops Group Fitness

Squad Heart Rate (рисунок 5) – приложение для групповых занятий фитнесом от австралийской компании Trackteam Pty Ltd. Squad Heart Rate, которое позволяет одновременно считывать данные с 12 пульсометров использующих технологию Bluetooth LE, в том числе и с Apple Watch. Возможно расширение числа устройств до 24 при помощи запуска бесплатного вспомогательного приложения Squad Hub на втором устройстве. Для запуска приложения потребуется iOS/iPadOS 13.0 [16]. Squad Heart Rate выводит на дисплей тренера информацию о частоте сердечного ритма, об очках эффективности (рассчитываются как суммарная продолжительность тренировки в минутах для пяти зон ЧСС, умноженное на коэффициент, относящийся к каждой зоне), о длительности тренировки и о количестве сожженных калорий. Приложение доступно по подписке за 899 рублей в месяц либо за 8999 рублей в год [17].

Рисунок 5 – Приложение Squad Heart Rate

Результат обзора существующих аналогов представлен в таблицах 1 и 2.

Таблица 1 – Характеристики Polar Club, Myzone In Club и OnBeat

Характеристика	**Polar Club**	**Myzone In Club**	**OnBeat**
Системные требования	iPad с iPadOS 12.0 или новее; доступ в Интернет	ПК на Windows уровня Intel Compute Stick или выше; ATN+ ресивер; внешний дисплей	iPhone/iPod touch с iOS 9.3 или новее, iPad с iPadOS 9.3 или новее, Mac с macOS 11.0 или новее; ANT+ приемник
Передача данных	Интернет	ANT+	Bluetooth LE/ ANT+
Кол-во устройств	≤90	Не указано	≤ 12 – Bluetooth LE, 12+ – ANT+
Устройства	Polar датчики	Myzone датчики (MZ-3 или MZ-1)	Устройства без аутентификации
Показатели	ЧСС, калории, зона ЧСС	ЧСС, калории, баллы усилия	ЧСС, калории, зона ЧСС, процент от максимального пульса, средняя ЧСС
Цена в месяц, руб	15500	По договоренности	От 4000
Страна	Финляндия	США	США

Таблица 2 – Характеристики SelfLoops Group Fitness и Squad Heart Rate

Характеристика	SelfLoops Group Fitness	Squad Heart Rate
Системные требования	Apple TV с tvOS 13.0 или новее, iPad с iPadOS 13.0 или новее, ТВ/смартфон/планшет с Android 6.0 или новее; ANT+ приемник	iPhone/iPod touch с iOS 13.0 или новее, iPad с iPadOS 13.0 или новее, поддержка Bluetooth LE
Передача данных	ANT+	Bluetooth LE
Кол-во устройств	≤ 40	≤ 12 – 1 iPhone, ≤ 24 – 2 iPhone
Устройства	Устройства без аутентификации, Apple Watch	Устройства без аутентификации, Apple Watch
Показатели	ЧСС, калории, процент от максимального пульса	ЧСС, калории, очки эффективности, длительность тренировки
Цена в месяц, руб	По договоренности	От 899
Страна	Италия	Австралия

1.2 Анализ основных технологических решений

На основании обзора аналогов проанализируем основные технологические решения конкурентных продуктов и предложим наиболее подходящие варианты решения поставленной задачи.

1.2.1 Операционная система

Большинство рассмотренных аналогов использует в качестве операционной системы iOS. Это третья по популярности ОС с долей мирового рынка в 17,42%, в то время как у Android (самая популярная ОС в мире) этот показатель составляет 42,76% [18]. Так же цена iOS устройств значительно выше смартфонов на Android [19]. Исходя их этих факторов, для большей доступности приложения – будем использовать платформу Android.

1.2.2 Способы передачи данных

Датчики пульса и фитнес-трекеры, в подавляющем большинстве случаев, используют протоколы Bluetooth LE и/или ANT+ для передачи данных. Проведем сравнение рассмотренных протоколов между собой:

– количество каналов: Bluetooth LE – 40 каналов [20], ANT+ – 83 канала;

– дальность приема на открытой местности: Bluetooth LE – ~100 м [21], ANT+ – ~30 м [22];

– поддержка смартфонами: Bluetooth LE – поддерживают все современные смартфоны, ANT+ – для работы необходимо подключать дополнительное оборудование, некоторые смартфоны имеют аппаратную поддержку протокола.

На основании рассмотренных технологий сделан выбор в пользу Bluetooth LE, поскольку использование технологии позволит расширить перечень поддерживаемых устройств, сделает итоговые затраты пользователя по использованию программы минимальным. Так же технология Bluetooth LE более чем в 3 раза превосходит ANT+ по дальности приема сигнала, что позволит проводить занятия на открытой местности. Так, например, дальность связи Bluetooth LE достаточна для занятий на легкоатлетическом стадионе, размеры которого составляют: 177x93 м [23].

1.2.3 Получаемые и анализируемые данные

Все рассмотренные аналоги собирают данные о ЧСС и, в связке с заданными характеристиками спортсмена (возраст, вес, пол), производят расчет критериев для оценки текущего состояния организма. Проведя анализ интерфейсов рассмотренных аналогов, были выявлены следующие отображаемые показатели:

– потерянные калории;

– текущая ЧСС;

– средняя ЧСС;

– зоны интенсивности ЧСС;

– баллы усилия/очки эффективности (оценка приложенных усилий);

– длительность тренировки;

– близость к максимально допустимому пульсу.

Поскольку большое количество одновременно выводимых данных может неблагоприятно сказаться на восприятии информации куратором занятия, для реализации необходимо выбрать ограниченное число наиболее информативных отображаемых показателей. Следовательно, в интерфейсе разрабатываемого приложения представим следующие показатели:

– текущую ЧСС;

– зону интенсивности ЧСС.

Так же следует выводить информацию о уровне заряда и силе сигнала подключенного устройства, это необходимо для своевременной замены или подзарядки устройства.

1.2.4 Поддерживаемые устройства

Поскольку в качестве протокола передачи данных был выбран Bluetooth LE, мы можем использовать обнаружение стандартных GATT сервисов, описанных спецификацией Bluetooth [24]. Например, можно использовать Heart Rate Service для получения информации о ЧСС, Battery Service для получения информации о уровне заряда устройства. Такой подход позволит обеспечить совместимость со всеми поддерживающими данные стандарты устройствами.

1.2.5 Язык программирования

Разрабатываемое приложение будет активно взаимодействовать с Bluetooth LE смартфона, для работоы с которой необходимо использовать Android API, а именно классы, расположенные в пространстве имен android.bluetooth.le [26] (добавлены в версии API Level 21). Эти классы, как и вся библиотека Android API – реализованы на языке Java, поэтому взаимодействовать с ними напрямую можно, используя Kotlin или Java [27]. Документация к библиотеке содержит множество примеров использования на этих двух языках, что ускорит разработку приложения. Исходя из этого, рассмотрим и выберем один из предложенных языков.

Kotlin – современный язык программирования, работающий поверх Java Virtual Machine, что позволяет ему использовать все созданные ранее Java-библиотеки. Kotlin лаконичен, реализует отсутствующий в Java функционал, такой как: классы данных, null-безопасность [28], функции-расширения, иммутабельность и другие. На Google I/O в 2019 году было объявлено, что язык Kotlin стал приоритетным языком программирования в разработке под Android [29]. Все эти факторы делают Kotlin наиболее подходящим для решения поставленной задачи.

1.2.6 Хранение данных

ОС Android предоставляет несколько вариантов хранения данных приложения [30]:

– хранилище для конкретного приложения (app-specific storage): файлы, предназначенные только для использования приложением-владельцем;

– общее хранилище (shared storage): совместно используемые файлы, доступные для других приложений;

– настройки (preferences): частные примитивные данные в парах ключ-значение, используемые для восстановления конфигурации приложения после перезапуска;

– базы данных (databases): хранилище структурированных данных в частной базе данных (на основе SQLite), работа с которой организована при помощи стандартной библиотеки Room [31].

Подберем необходимый тип хранилища для хранения информации о пользователях, группах, устройствах и настройках. Для хранения информации о настройках приложения оптимальным решением будет использование специализированного для данной задачи preferences хранилища, поскольку формат хранения данных о настройках в виде пар ключ-значение отлично для этого подходит. Информацию о пользователях, группах, устройствах, тренировках следует хранить в базе данных, вследствие связности данных между собой. Использование SQL запросов поможет гибко управлять большим объемом данных и генерировать полезные статистические выборки.

1.2.7 Способ монетизации

Все аналогичные приложения предоставляют свой функционал по подписке, минимальная цена которой – 899 рублей в месяц. Эта плата может склонить пользователя в пользу разрабатываемого бесплатного приложения. В качестве замены ежемесячной платы можно использовать ненавязчивую рекламу, которая не будет мешать основному функционалу программы. Баннеры с рекламой следует размещать во всех местах приложения, кроме основного экрана с информацией о спортсменах, поскольку реклама будет мешать восприятию информации куратором во время проведения занятия и уменьшать полезную площадь дисплея, сокращая количество одновременно отображаемых данных.

1.3 Вывод

В результате обзора аналогов и анализа технологических решений конкурирующих продуктов была обоснована целесообразность реализации ПО и АПК в целом, определены основные технологические решения, требуемые для создания приложения для считывания и анализа показаний с устройств для мониторинга состояния организма.

Были выявлены и учтены основные недостатки аналогичных приложений:

– выбор ОС Android позволил расширить обхват аудитории в 2,5 раза относительно приложений на iOS;

– в качестве способа передачи данных сделан выбор в пользу протокола Bluetooth LE. Данное решение позволит не покупать дополнительное оборудование (для приема ANT+ сигнала), что снизит затраты на использование приложения и, как следствие, увеличит охват аудитории;

– использование обнаружения стандартизированных Bluetooth LE GATT-сервисов дает возможность разработать совместимое с различными устройствами приложение;

– ограниченный выбор наиболее важных анализируемых и отображаемых в реальном времени показателей сделает приложение информативным и при этом неперегруженным информацией;

– отсутствие платных подписок позволит большему количеству пользователей использовать приложение.

2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

Определим основные требования, предъявляемые к итоговому приложению для выполнения поставленной цели

2.1 Функциональные требования

К разрабатываемому приложению предъявлены следующие функциональные требования:

1) управление устройствами (Bluetooth-датчиками);

1.1) поиск и добавление нового устройства;

1.2) поиск среди добавленных устройств;

1.3) удаление устройства;

1.4) смена имени устройства;

2) управление пользователями (спортсменами);

2.1) добавление нового пользователя;

2.2) поиск среди добавленных пользователей;

2.3) удаление пользователя;

2.4) изменения информации о пользователе;

2.4.1) изменение имени;

2.4.2) изменение фамилии;

2.4.3) изменение даты рождения;

2.4.4) изменение пола;

2.4.5) изменение группы;

2.4.6) привязка персонального устройства;

3) управление группами пользователей;

3.1) добавление новой группы;

3.2) поиск среди добавленных групп;

3.3) удаление группы;

3.4) изменение группы;

3.5) выбор активной группы спортсменов для отслеживания;

4) управление хранилищем;

4.1) просмотр статистики использования хранилища;

4.2) возможность удаления всех добавленных сущностей;

5) управление системой оповещений;

5.1) установка пороговой частоты сердечных сокращений (процент от максимальной нагрузки от 50% до 100%) для оповещения;

5.2) активация/деактивация оповещений;

6) взаимодействие с главным экраном (мониторинг группы);

6.1) численное отображение пульса пользователя в составе отслеживаемой группы;

6.2) графическое представление зоны ЧСС в виде пятисегментной шкалы, отражающей текущую зону ЧСС пользователя в составе отслеживаемой группы, с возможностью численного отображения при нажатии;

6.3) отображение имени и фамилии пользователя в составе отслеживаемой группы;

6.4) графическое представление уровня заряда и силы сигнала с возможностью численного отображения при необходимости;

6.5) индикация текущего подключённого устройства;

6.6) индикация ошибки подключения к устройству;

6.7) сортировка отображения выводимой информации об отслеживаемой группе по имени/ЧСС/нагрузке;

6.8) возможность временной привязки устройства к пользователю в случае отсутствия персонального устройства;

6.9) остановка/запуск сканирования устройств активной группы.

2.2 Нефункциональные требования

Разрабатываемое приложение должно соответствовать следующим нефункциональным требованиям:

1) поддержка минимальной версии операционной системы Android не ниже 5.0 (API Level 21);

2) приложение должно быть разработано на языке программирования Kotlin;

3) пользовательский интерфейс приложения должен быть удобным интуитивно понятным;

4) приложение не должно весить более 10 МБ (без учета хранимых данных);

5) интерфейс приложения должен быть адаптирован для работы только в портретном режиме;

6) работа с Bluetooth LE совместимыми устройствами с поддержкой стандартных GATT профилей.

3 ПРОЕКТИРОВАНИЕ

Исходя из выдвинутых в предыдущих главах технологических решений и требований, выделим структурные элементы (подсистемы) проектируемого решения, необходимые для реализации функциональных требований. Спроектируем схему локальной базы данных, опишем хранимые данные и представим необходимые для создания приложения алгоритмы.

3.1 Архитектура предлагаемого решения

Архитектурно приложение состоит из шести подсистем, функции и методы реализации которых приведены в таблице 3.

Таблица 3 – Функции и методы реализации подсистем

Название подсистемы	**Функция подсистемы**	**Метод реализации**
Подсистема получения данных	Поочередное подключение и получение данных с устройств	Осуществляется путем взаимодействия с Android SDK, позволяющего считать информацию с Bluetooth LE устройств
Подсистема хранения данных	Взаимодействие с локальной БД для хранения пользовательских данных	Осуществляется путем взаимодействия с БД SQLite средствами Android SDK (Room)
Подсистема анализа данных	Обработка данных с целью выявления отклонений от нормы	Осуществляется путем анализа (сравнение с индивидуальными расчетными нормативными значениями) полученных данных и физиологических параметров
Подсистема визуализации данных	Отображение (численное и визуальное) в реальном времени данных о пользователях	Осуществляется путем отрисовки визуальных шкал и цветового акцентирования в реальном времени, отображающих текущие физиологических показателей

Продолжение таблицы 3

Подсистема оповещения оператора	Оповещение средствами смартфона (вибрация, звук, визуальное сообщение) оператора о превышении заданных нормативных показаний	Осуществляется путем сигнализации средствами смартфона с целью привлечения внимания оператора
Подсистема конфигурации приложения	Настройка и сохранение параметров приложения (подключенные устройства, пользователи, группы и нормативные значения)	Осуществляется путем установки оператором новых параметров приложения при помощи графического интерфейса пользователя

Логическая и компонентная архитектура системы, отражающая взаимодействие подсистем, изображена на рисунке 6.

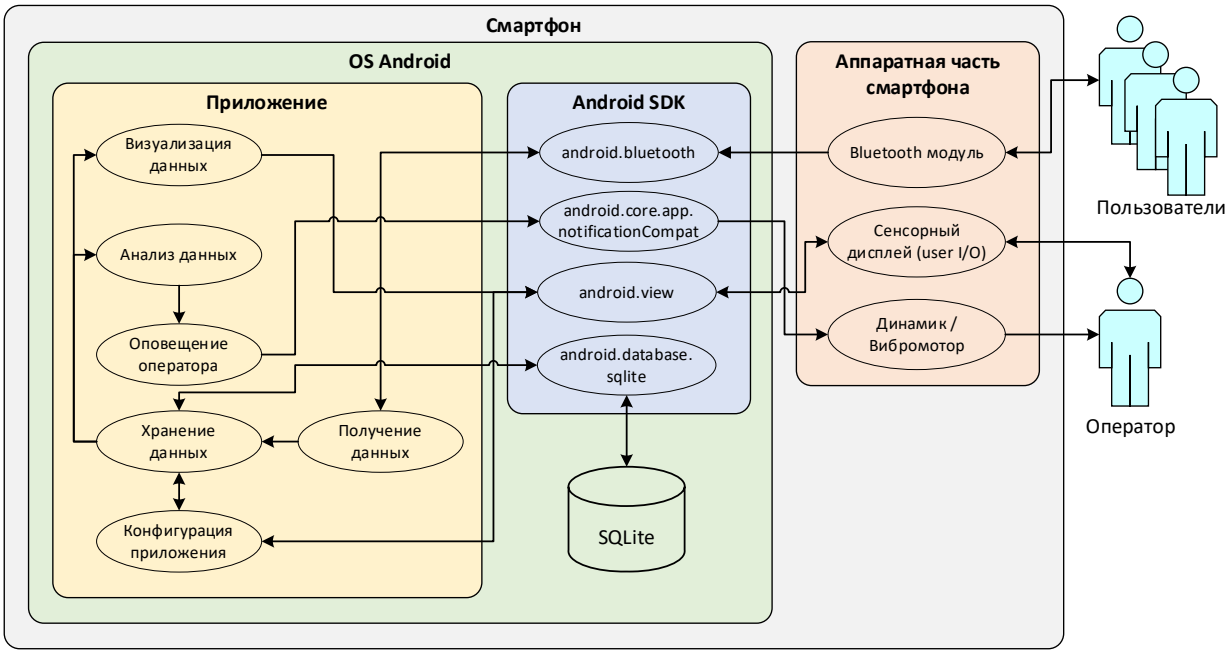

Рисунок 6 – Логическая и компонентная структура системы

3.2 Алгоритмы и формулы для решения задачи

Для визуализации зон пульса необходимо рассчитать максимально допустимое значение пульса. Для расчета персональных показателей воспользуемся формулами (1) и (2) основанными на возрасте и поле человека, предложенными в исследовании Американского колледжа кардиологии [32].

$$HR_{max} = 214 - 0{,}8 \times age, \quad (1)$$

$$HR_{max} = 209 - 0{,}7 \times age, \quad (2)$$

где: HR_{max} – максимальный пульс, уд./мин.;

age – возраст, лет.

Зная максимальный пульс человека, можно узнать, в какой пульсовой зоне находится его ЧСС. Обычно принято выделять пять зон ЧСС, которые находятся в диапазоне между 50 и 100% (таблица 4) [33].

Таблица 4 – Зоны ЧСС

Название зоны	**Интенсивность, % от HR_{max}**
МПК (максимальное потребление кислорода) (зона 5)	90-100
Анаэробная (зона 4)	80-90
Аэробная (зона 3)	70-80
Сжигание жира (зона 2)	60-70
Легкая активность (зона 1)	50-60
Покой	< 50

Тренировка неподготовленного спортсмена в интенсивных пульсовых зонах (зоны 3, 4, 5) может быстро исчерпать внутренние резервы организма, поэтому важно следить за соблюдением границ интенсивности с целью предотвращения перегрузок и повышения эффективности тренировок [34].

Для соблюдения вышеописанных условий требуется сравнивать полученные значение ЧСС, рассчитывать процент текущей нагрузки и, в случае превышения заданного порогового значения нагрузки, предупреждать оператора о превышении заданных параметров (рисунок 7).

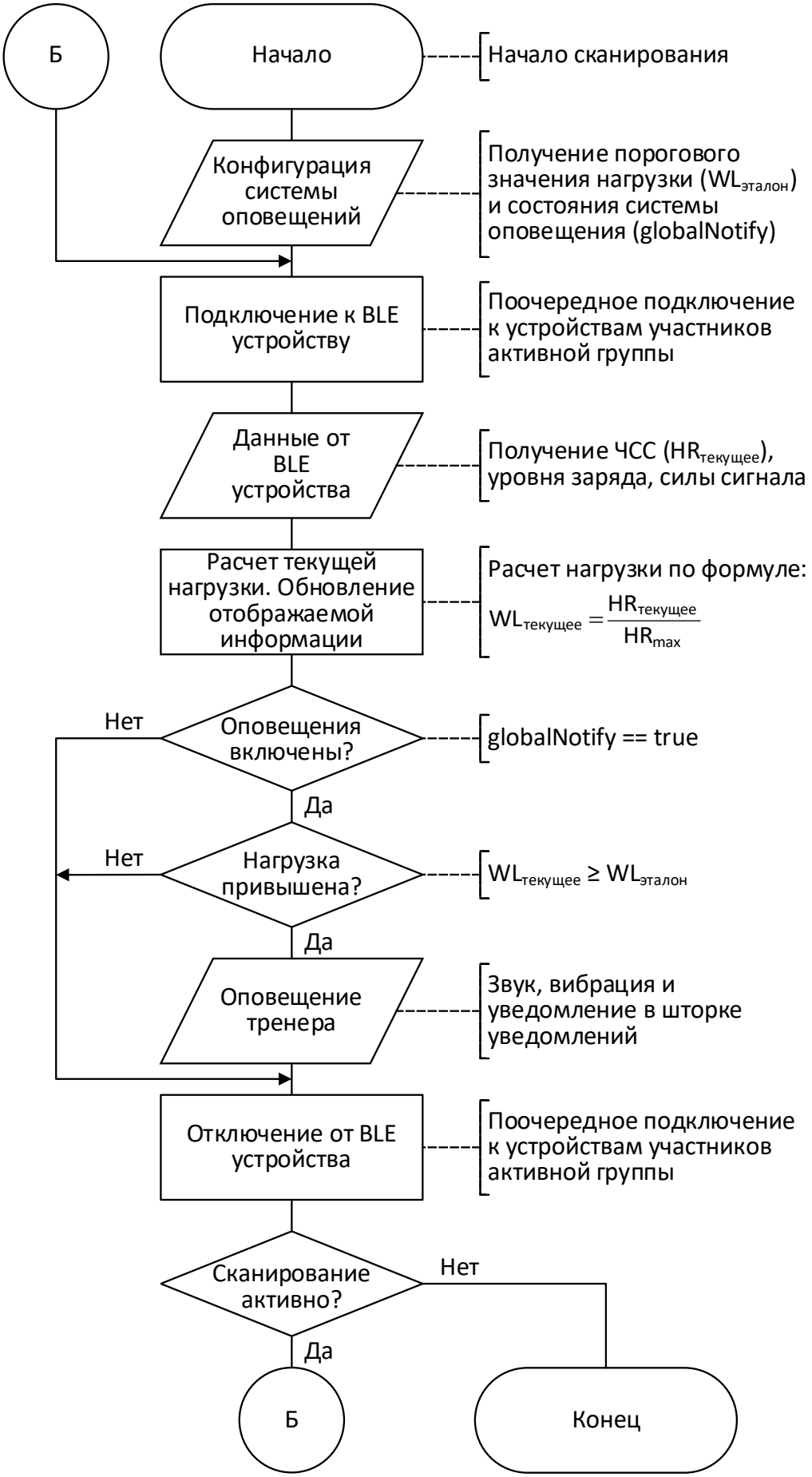

Рисунок 7 – Блок-схема работы системы получения и обработки данных

Программная реализация протокола Bluetooth LE в ОС Android ограничивает одновременное подключение к более чем семи устройствам (константа GATT_MAX_PHY_CHANNEL [35]). Для получения данных с большего количества устройств реализуем поочередный опрос устройств. Для этого необходимо поочередно подключаться к заранее известным Bluetooth LE устройствам, считывать необходимые GATT характеристики, а затем – отключаться от устройств. Данный подход позволит получать данные от 40 устройств (физические ограничения протокола Bluetooth LE) [20].

3.3 Описание данных

Данные о пользователях, группах и устройствах сохраняются в локальную базу данных (рисунок 8).

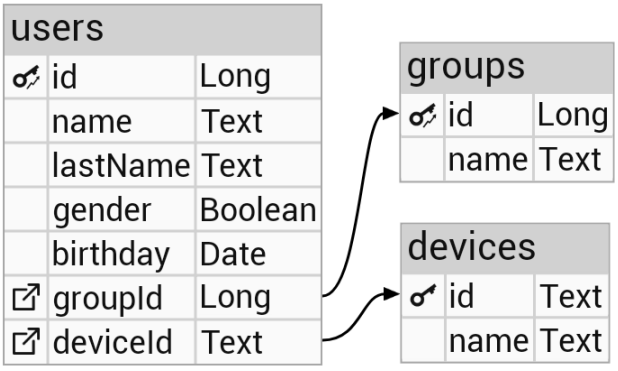

Рисунок 8 – Схема данных

Таблица «Пользователи» содержит:

– поле «id» – автоинкрементный первичный ключ, однозначно идентифицирующий пользователя;

– поле «name» – строковое поле, содержащее имя пользователя;

– поле «lastName» – строковое поле, содержащее фамилию пользователя;

– поле «gender» – логическое поле, обозначающее пол пользователя: true – мужчина, false – женщина;

– поле «brthday» – поле, содержащее дату рождения пользователя;

– поле «groupId» – внешний ключ для связи пользователей и групп (может быть пустым, если пользователь не привязан к группе). Обнуляется при удалении связной записи, каскадно обновляется при обновлении связной записи;

– поле «deviceId» – внешний ключ для связи пользователей и устройств (может быть пустым, если у пользователя нет персонального устройства). Обнуляется при удалении связной записи, каскадно обновляется при обновлении связной записи.

Таблица «groups»:

– поле «id» – автоинкрементный первичный ключ, однозначно идентифицирующий группу;

– поле «название группы» – строковое поле, содержащее название группы. Содержимое поля уникально (unique index).

Таблица «devices»:

– поле «id» – первичный ключ, основанный на уникальном идентификаторе Bluetooth устройства;

– поле «name» – строковое поле, содержащее имя Bluetooth устройства, которое задается пользователем. Содержимое поля уникально (unique index).

Сохранение настроек приложения реализовано при помощи SharedPreferences API [36]. Объект SharedPreferences указывает на файл, содержащий пары ключ-значение. В разрабатываемом приложении будет использоваться два значения для хранения настроек конфигурации системы оповещений:

– globalNotify (boolean) – флаг разрешения оповещений (включение и выключение оповещений);

– procentValue (int) – пороговое значение (процент от нагрузки). Величина нагрузки, при достижении которой сработает система оповещения.

4 РЕАЛИЗАЦИЯ

4.1 Реализация интерфейсов

Структурно приложение состоит из 6 экранов. Навигация между экранами осуществлена при помощи бокового меню. Опишем экраны более подробно.

4.1.1 Главный экран (изначальное состояние)

Главным и начальным экраном приложения является список участников активной группы. Поскольку изначально активная группа не выбрана, данный список пуст. При пустом списке отслеживания пользователю дается рекомендация о том, что ему необходимо выбрать группу для отслеживания (рисунок 9).

По нажатию на левую верхнюю иконку (hamburger button) открывается боковое меню, в котором можно перейти на различные экраны для управления пользователями, группами, устройствами, хранилищем и оповещениями.

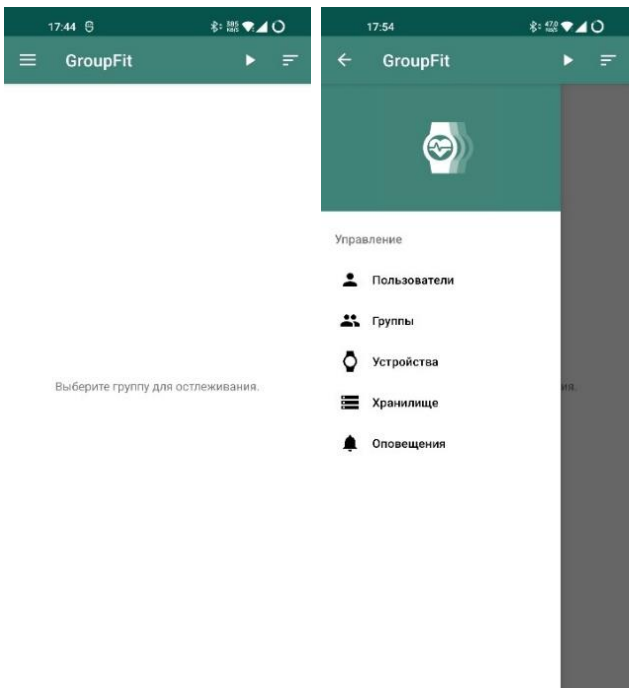

Рисунок 9 – Начальный экран приложения

4.1.2 Экран управления пользователями

Для добавления нового пользователя необходимо нажать на плавающую кнопку (floating action button) в правом нижнем углу (рисунок 10).

Рисунок 10 – Экран управления пользователями

После нажатия на кнопку добавления пользователя открывается форма для заполнения данных нового пользователя (рисунок 11).

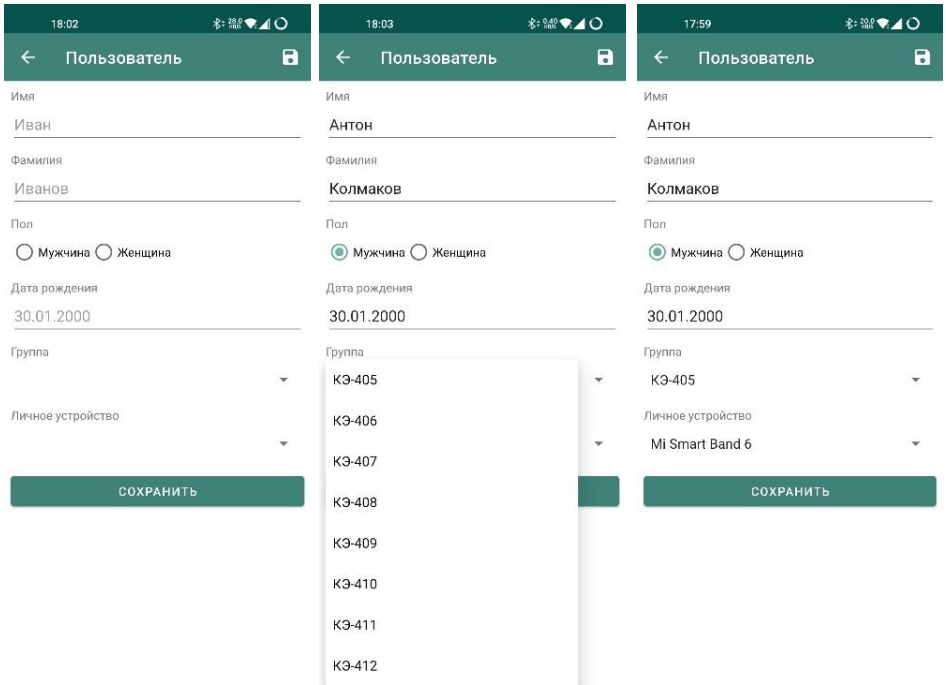

Рисунок 11 – Процесс добавления нового пользователя

При попытке сохранения некорректных значений будет выведено советующее предупреждение в виде всплывающего toast-уведомления (рисунок 12).

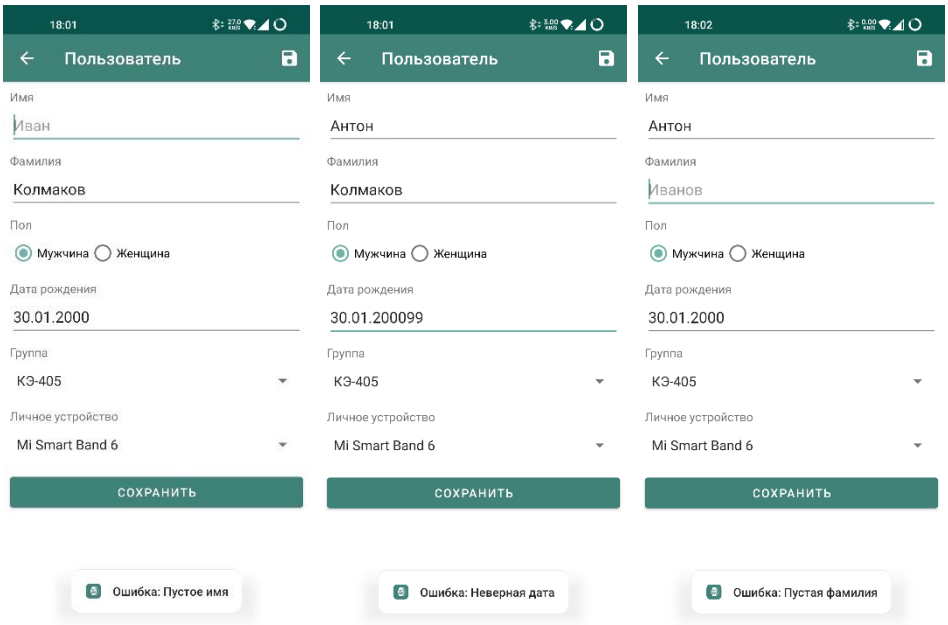

Рисунок 12 – Попытка ввода некорректных данных

После добавления пользователей имеется возможность удалить или отредактировать ранее введенные пользовательские данные (рисунок 13).

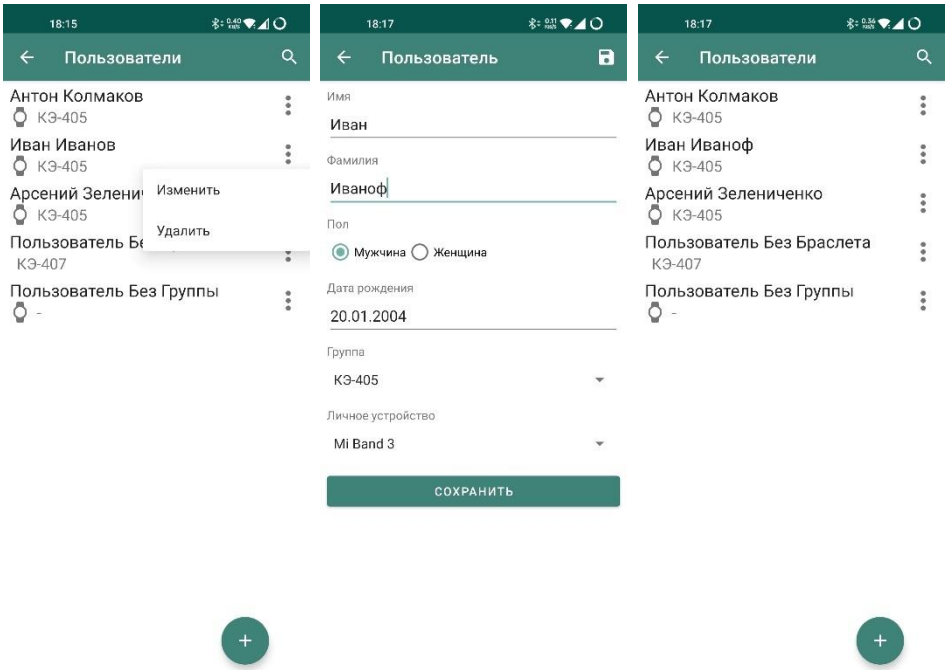

Рисунок 13 – Редактирование данных пользователя

Помимо имени и фамилии пользователя, в списке отображается название группы (рисунок 14), в которой состоит пользователь, а также иконка, обозначающая наличие персонального устройства (рисунок 15).

Рисунок 14 – Группа, к которой принадлежит пользователь

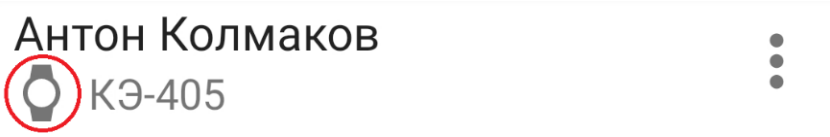

Рисунок 15 – Наличие персонального устройства

Для нахождения пользователя можно воспользоваться поиском. Для этого нужно нажать на иконку лупы в правом верхнем углу и ввести имя искомого пользователя (рисунок 16).

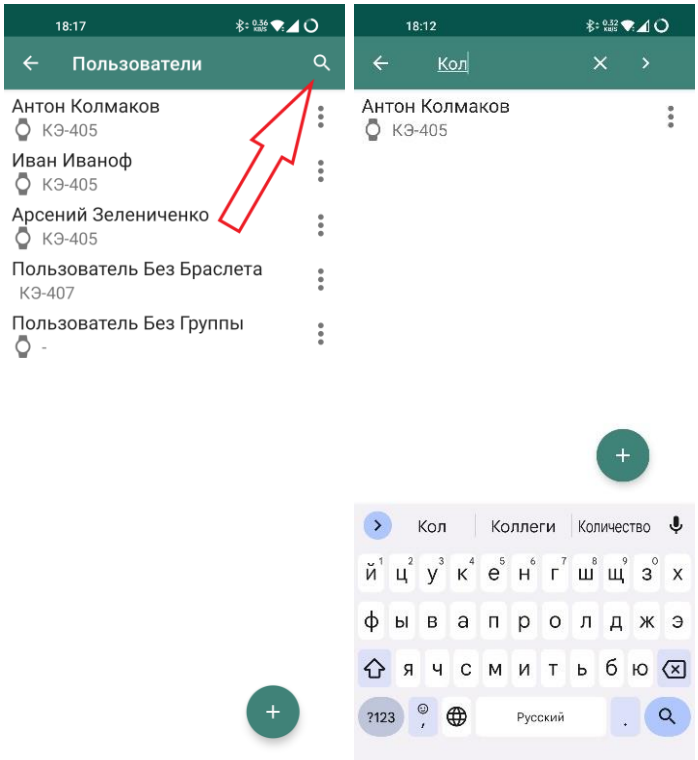

Рисунок 16 – Поиск пользователей

4.1.3 Экран управления группами

Изменить (рисунок 17), добавить (рисунок 18) или удалить группу можно на экране управления группами.

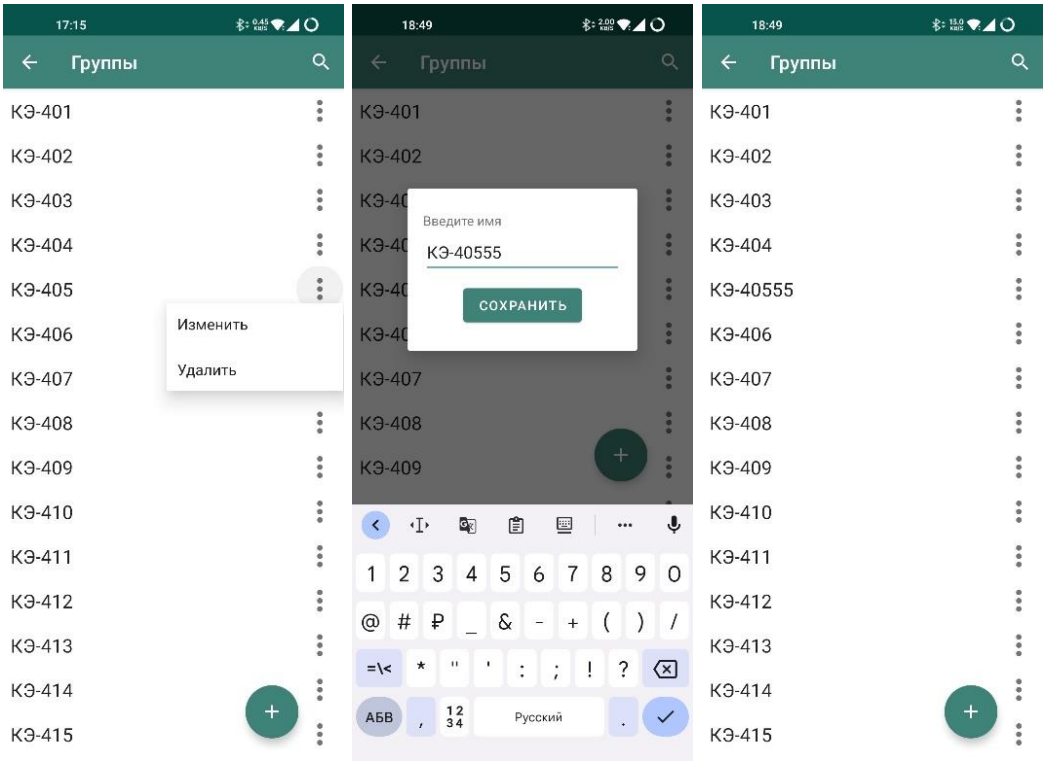

Рисунок 17 – Изменение существующей группы

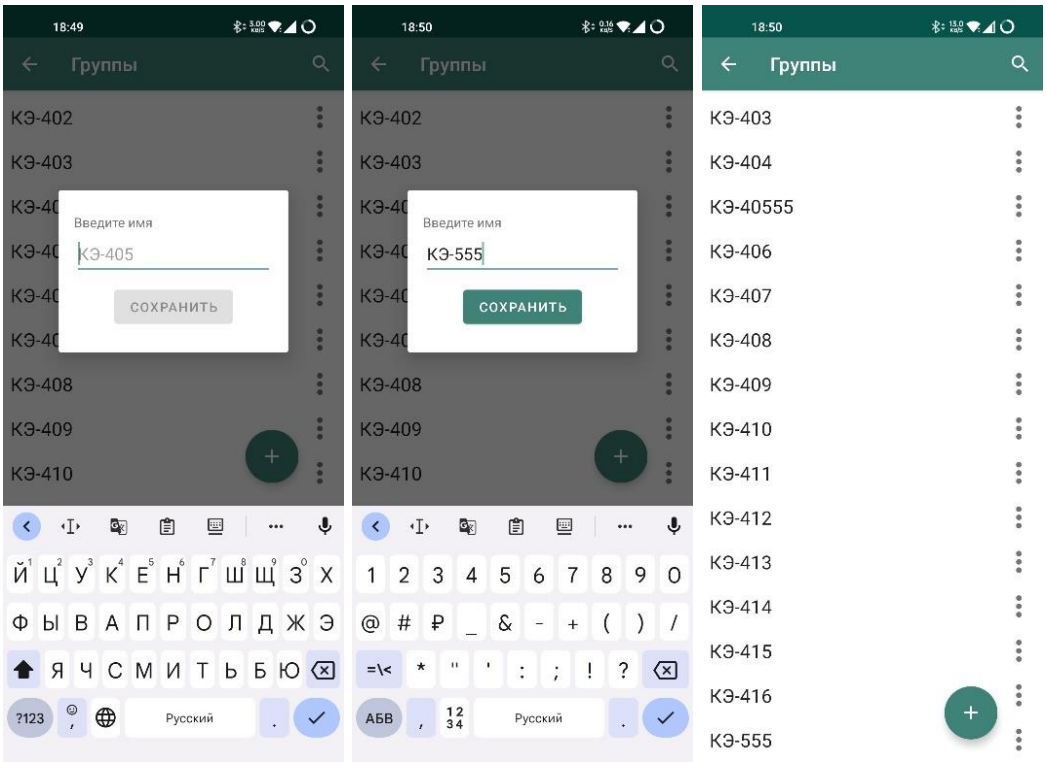

Рисунок 18 – Добавление новой группы

При вводе пустой строки во время добавления группы или редактирования имени группы кнопка «сохранить» становится неактивна (рисунок 18). При попытке изменить имя группы или добавить новую с уже существующем именем пользователь увидит ошибку в виде всплывающего toast-уведомления (рисунок 19).

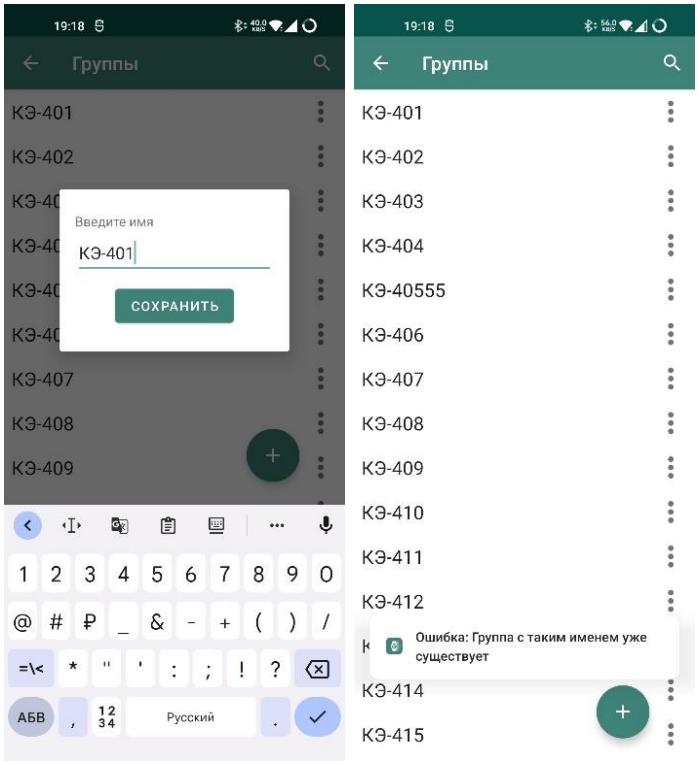

Рисунок 19 – Ошибка при добавлении уже существующей группы

Для нахождения нужной группы можно воспользоваться поиском. Для этого нужно нажать на иконку лупы в правом верхнем углу и ввести номер искомой группы (рисунок 20).

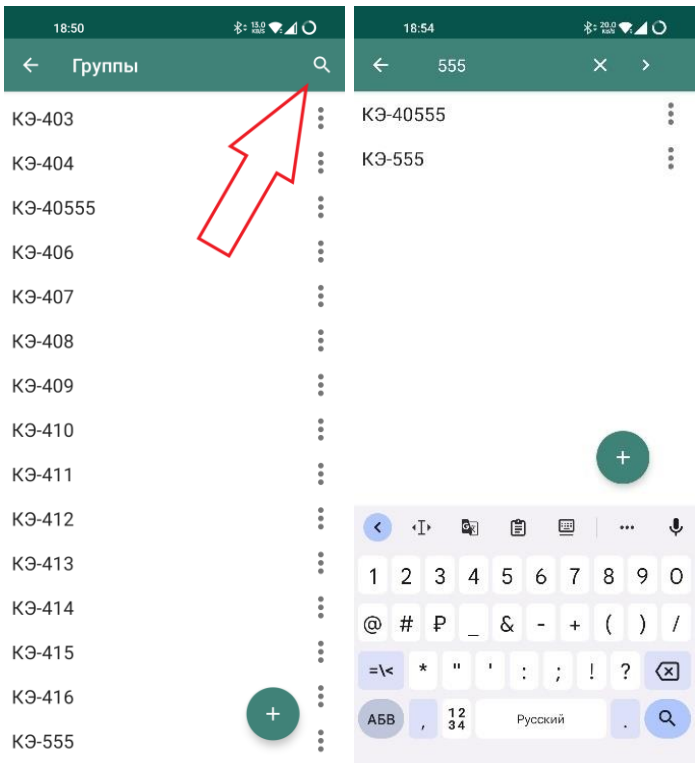

Рисунок 20 – Поиск среди групп

4.1.4 Экран управления устройствами

Экран управления устройствами аналогичен по функционалу (рисунок 21) экрану управления группами, за исключением того, что в списке устройств, в дополнение к имени устройства, ниже отображается его уникальный адрес Bluetooth.

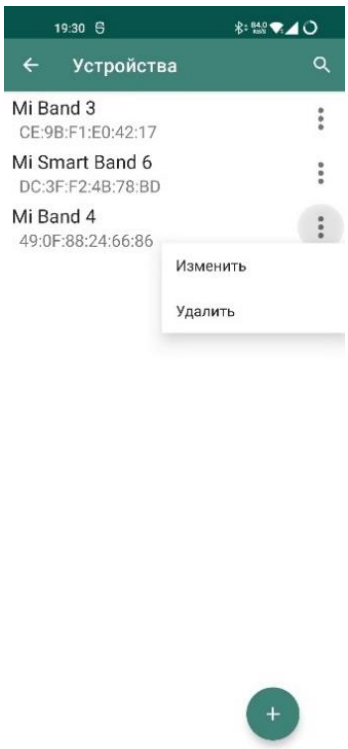

Рисунок 21 – Опции для взаимодействия с устройством

Добавление нового устройства происходит при помощи сканирования Bluetooth LE устройств поблизости (рисунок 22).

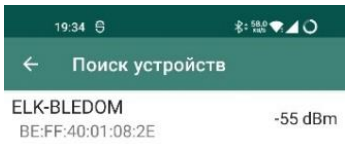

Рисунок 22 – Обнаруженные устройства

Для добавления найденного устройства необходимо выбрать новое устройство из списка доступных и, при желании, изменить его имя (рисунок 23).

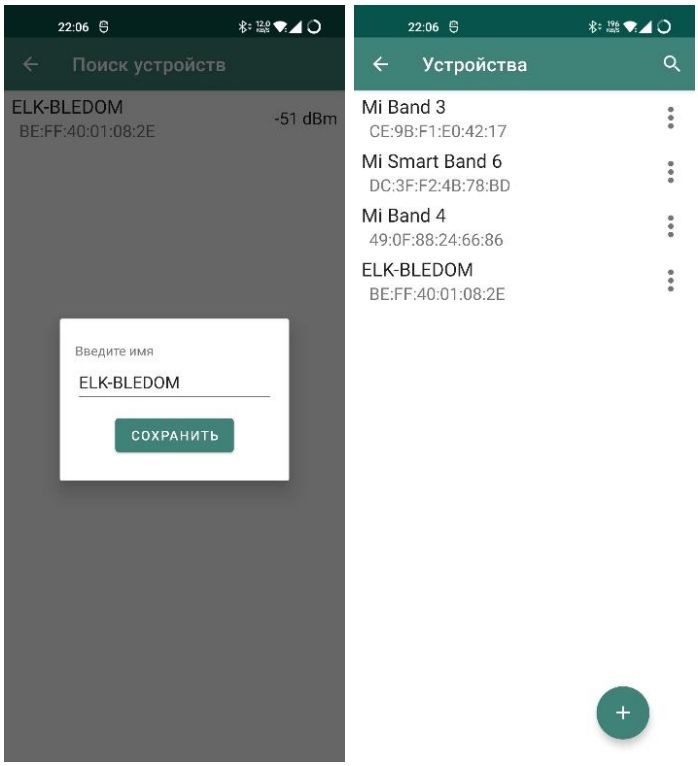

Рисунок 23 – Добавление нового устройства

Во избежание дублирования добавленных устройств, ранее добавленные устройства не отображаются в списке обнаруженных устройств. Список найденных устройств автоматически сортируется по уровню сигнала (наиболее близкие к смартфону устройства – вверху списка), что позволяет найти из множества устройств ближайшее к смартфону.

4.1.5 Экран управления хранилищем

На экране управления хранилищем отображается статистика использования хранилища пользователями, группами и устройствами (рисунок 24).

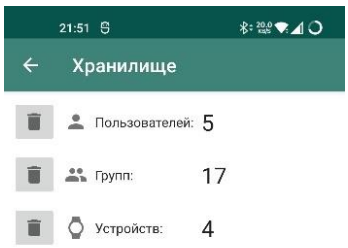

Рисунок 24 – Управление хранилищем

Имеется возможность быстрого удаления всех пользователей, групп или устройств, для этого необходимо нажать на кнопку с изображением мусорного бака. В качестве примера удалим все добавленные ранее устройства (рисунок 25).

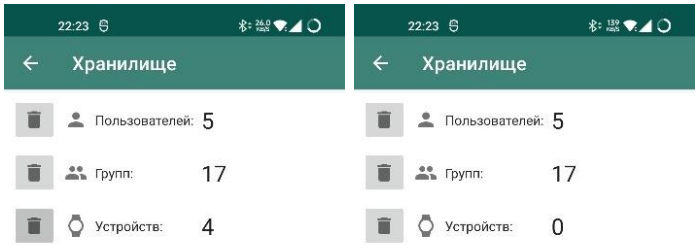

Рисунок 25 – Удаление устройств

В результате удаления всех устройств они были удалены, в том числе, и у всех пользователей (рисунок 26).

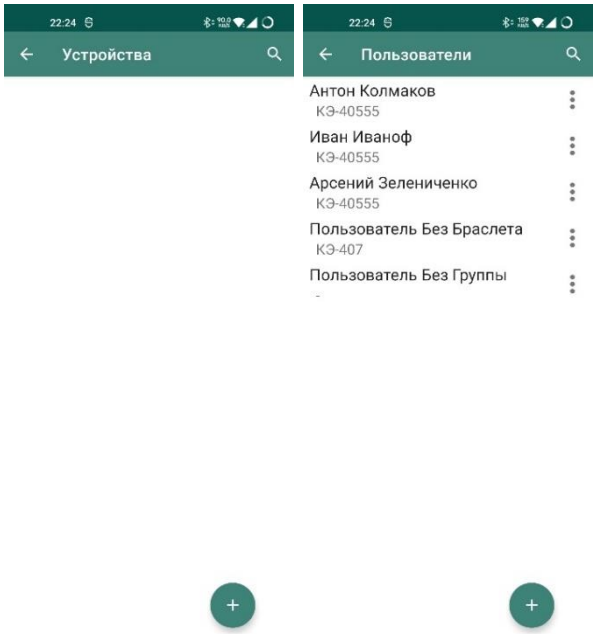

Рисунок 26 – Результат удаления устройств

4.1.6 Экран управления оповещениями

Экран управления оповещениями позволяет активировать оповещения о превышении заданной нагрузки и выставить требуемое пороговое значение для срабатывания оповещения (рисунок 27).

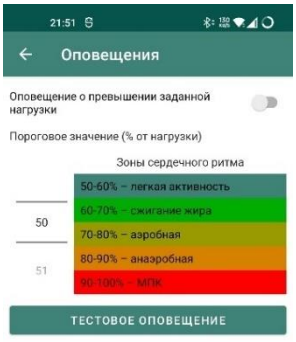

Рисунок 27 – Изначальная настройка системы оповещений

Присутствует возможность проверки оповещения, для этого необходимо установить желаемые параметры и нажать кнопку «тестовое оповещение» (рисунок 28).

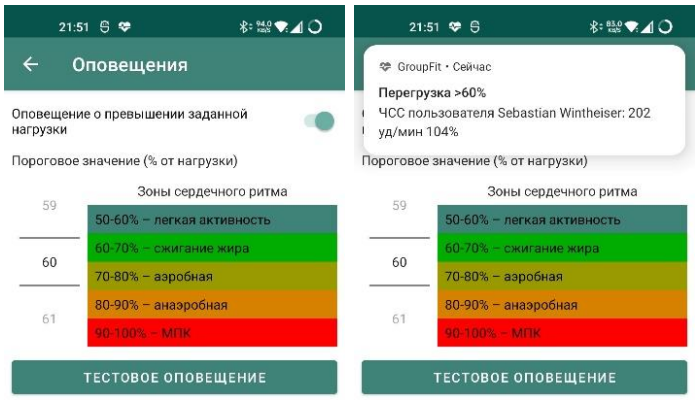

Рисунок 28 – Тестовое оповещение

4.1.7 Главный экран (режим мониторинга)

Для активации мониторинга ЧСС необходимо выбрать активную группу для тренировки. Для этого нужно перейти на экран управления группам и нажать на нужную группу (рисунок 29).

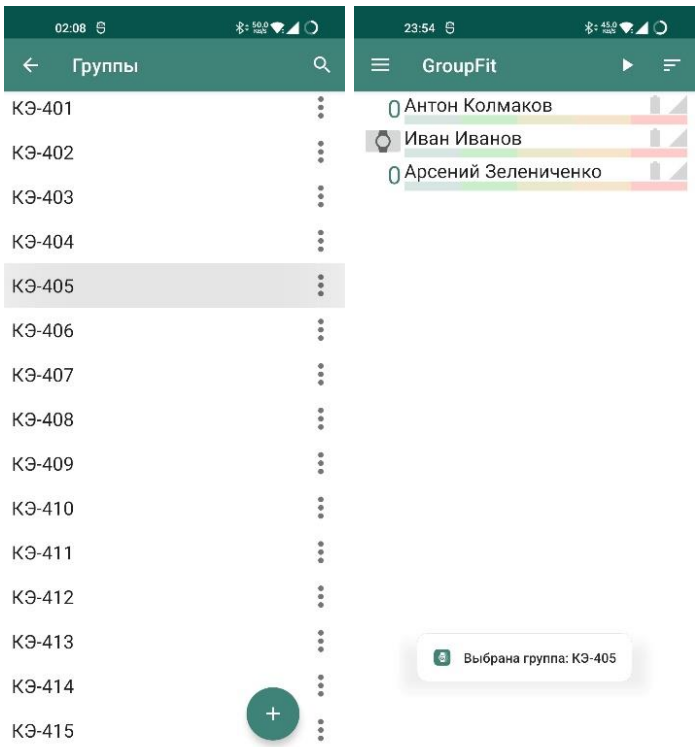

Рисунок 29 – Выбранная группа

В случае, если у участника группы нет собственного устройства, имеется возможность временно привязать (выдать) пользователю устройство. Для этого необходимо нажать на кнопку на месте пульса участника и выбрать устройство из списка непривязанных (общих) устройств (рисунок 30).

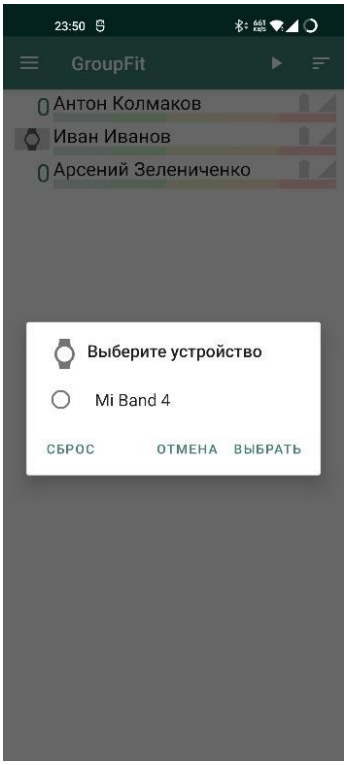

Рисунок 30 – Выбор устройства

Имеется возможность временно изменить устройство участника тренировки, для этого необходимо выполнить «долгое нажатие» по строчке с информацией о пользователе, после чего появится аналогичное диалоговое окно со списком доступных устройств.

После того, как все участники получили свои устройства, нужно запустить сканирование нажатием кнопки «плей» в верхней части приложения. После нажатия кнопки данной кнопки ее изображение сменится на «стоп» и начнется поочередное сканирование устройств (рисунок 31).

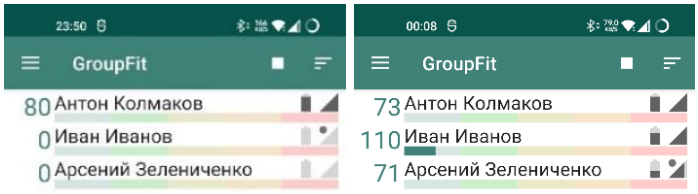

Рисунок 31 – Процесс поочередного подключения к устройствам

Рассмотрим более подробно элемент списка, содержащий информацию о пользователе (рисунок 32).

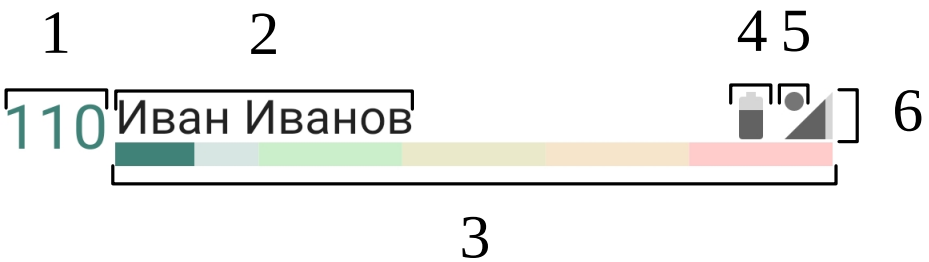

Рисунок 32 – Расположение элементов

Описание элементов:

1) текущее значение ЧСС;

2) имя и фамилия участника тренировки;

3) полоса-индикатор, показывающая текущую зону ЧСС (в диапазоне от 50 до 100%);

4) уровень заряда устройства;

5) обозначение текущего соединения. Если соединение прошло с ошибкой, на месте данного индикатора останется красный круг (рисунок 33);

6) уровень сигнала.

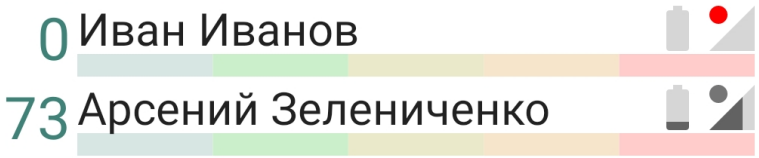

Рисунок 33 – Пример ошибки подключения к устройству

Реализована возможность просмотра численных значений визуальных индикаторов (нагрузка, уровень заряда, сила сигнала) (рисунок 34). Для этого необходимо нажать на интересующий показатель (иконка уровня заряда / силы сигнала / полосы нагрузки). Информация будет отображена в виде всплывающего toast-уведомления.

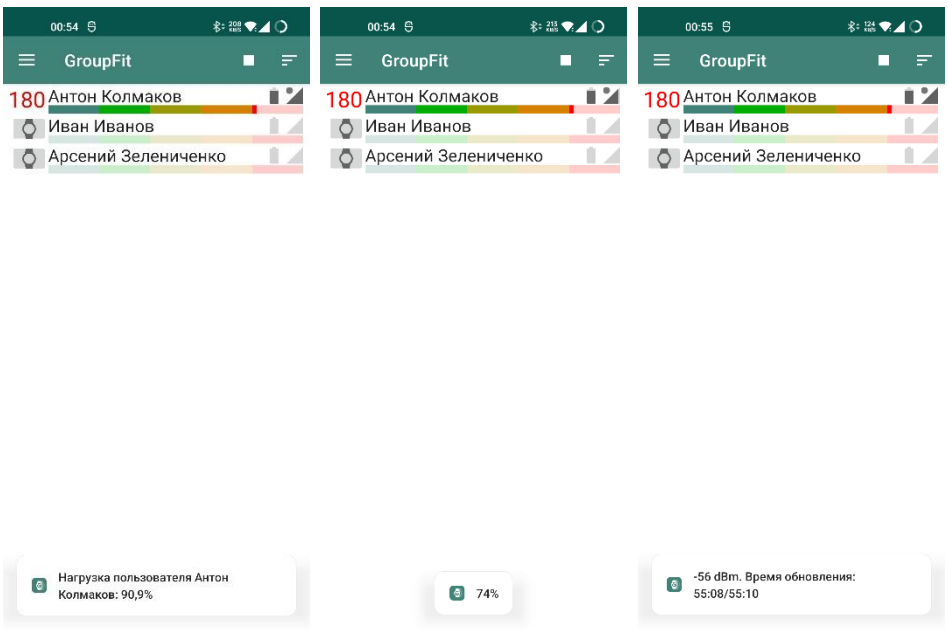

Рисунок 34 – Отображение численных значений графической информации

Для удобства восприятия информации на главном экране реализована возможность сортировки по имени, пульсу, зонам пульса (нагрузка). Для выбора режима сортировки необходимо нажать на иконку сортировки в правом верхнем углу и выбрать нужный режим (рисунок 35).

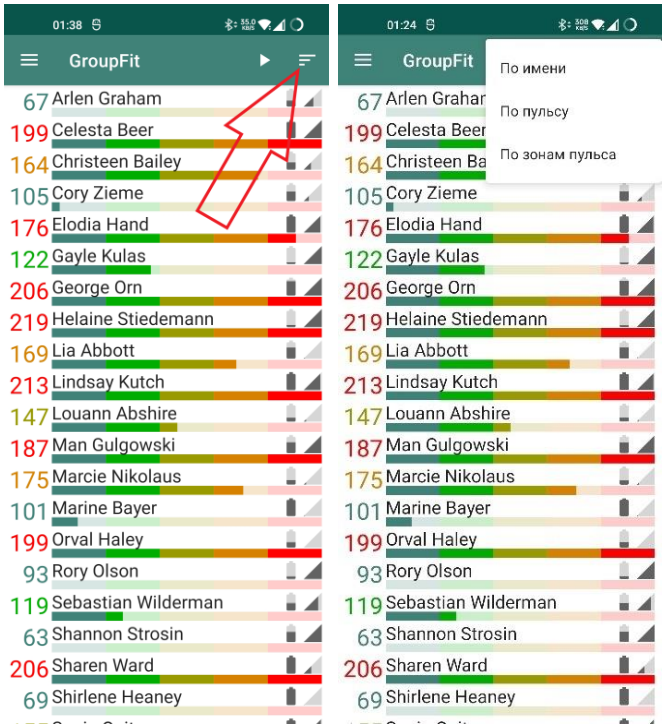

Рисунок 35 – Выбор режима сортировки

Применение сортировок представлено на рисунке 36.

Рисунок 36 – Возможные варианты сортировок

4.2 Реализация классов

Итоговая реализация приложения включает в себя (рисунок 37):

– 7 классов данных;

– 7 интерфейсов;

– 1 статичный класс;

– 30 классов.

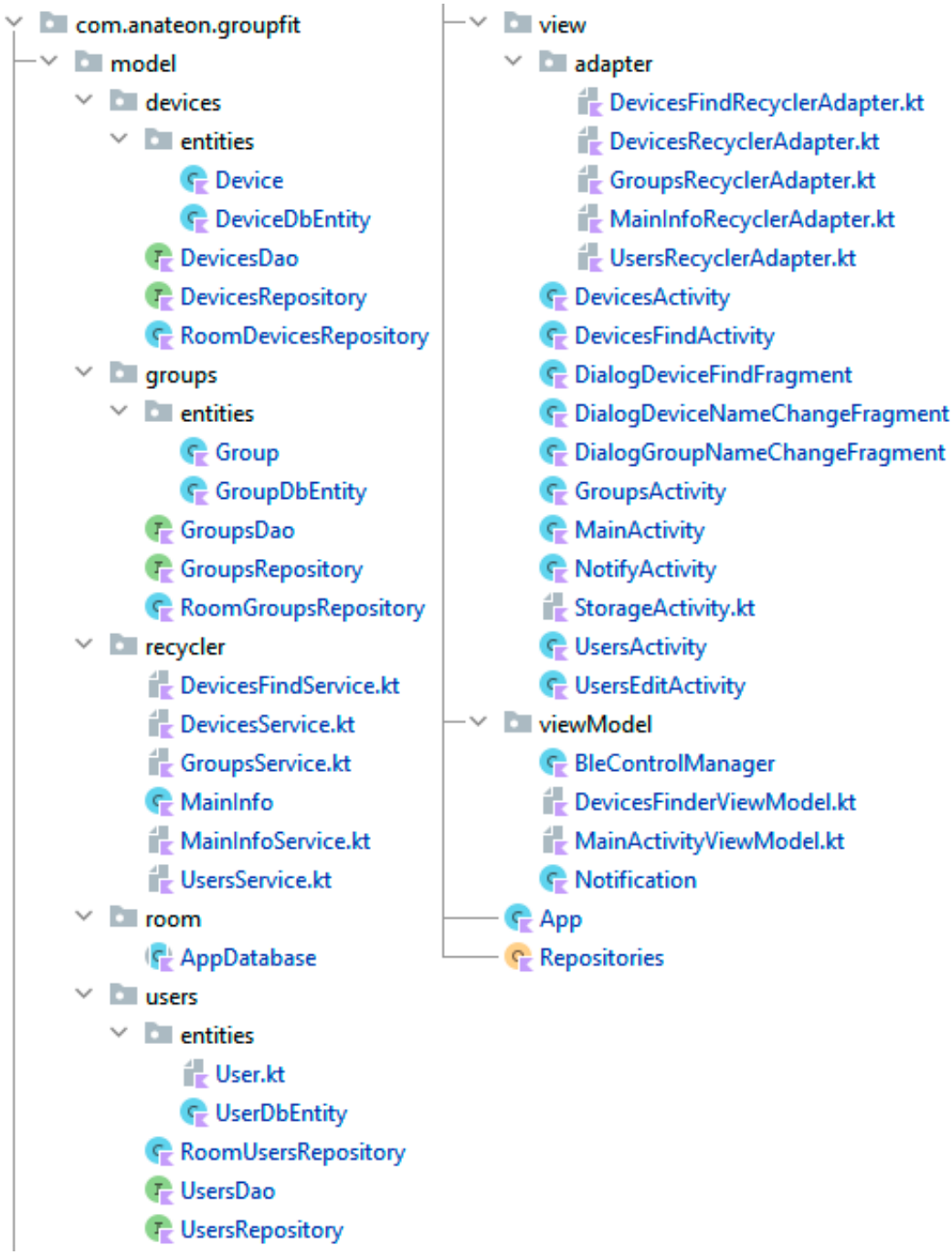

Рисунок 37 – Структура проекта

Дадим краткую характеристику каждой программной структуре. Полный исходный код описываемых классов и интерфейсов приведен в приложении А к пояснительной записке.

Device – дата класс для хранения данных об устройстве.

DeviceDbEntity – дата класс для хранения данных об устройствах в базе данных (аннотирован @Entity).

DevicesDao – интерфейс, описывающий запросы к базе данных для таблицы с устройствами (аннотирован @Dao).

DevicesRepository – интерфейс, описывающий методы для получения данных из таблицы устройств в базе данных.

RoomDevicesRepository – класс, реализующий методы интерфейса DevicesRepository.

Group – дата класс для хранения данных об устройстве.

GroupDbEntity – дата класс для хранения данных о группах в базе данных (аннотирован @Entity).

GroupsDao – интерфейс, описывающий запросы к базе данных для таблицы с группами (аннотирован @Dao).

GroupsRepository – интерфейс, описывающий методы для получения данных из таблицы групп в базе данных.

RoomGroupsRepository – класс, реализующий методы интерфейса GroupsRepository.

DevicesFindService – класс для управления элементами списка найденных устройств.

DevicesService – класс для управления элементами списка устройств.

GroupsService – класс для управления элементами списка групп.

MainInfo – дата класс, содержащий информацию об элементе списка главного экрана.

MainInfoService – класс для управления элементами списка главного экрана.

UsersService – класс для управления элементами списка пользователей.

AppDatabase – абстрактный класс (аннотирован @Database), описывающий методы получения Dao-классов.

User – дата класс для хранения данных о пользователе.

UserDbEntity – дата класс для хранения данных о пользователях в базе данных (аннотирован @Entity).

RoomUsersRepository – класс, реализующий методы интерфейса UsersRepository.

UsersDao – интерфейс, описывающий запросы к базе данных для таблицы с пользователями (аннотирован @Dao).

UsersRepository – интерфейс, описывающий методы для получения данных из таблицы пользователей из базы данных.

DevicesFindRecyclerAdapter – класс для переиспользования элементов списка найденных устройств в RecyclerView.

DevicesRecyclerAdapter – класс для переиспользования элементов списка устройств в RecyclerView.

GroupsInfoRecyclerAdapter – класс для переиспользования элементов списка групп в RecyclerView.

MainInfoRecyclerAdapter – класс для переиспользования элементов списка на главном экране в RecyclerView.

UsersInfoRecyclerAdapter – класс для переиспользования элементов списка пользователей в RecyclerView.

DevicesActivity – класс, реализующий активность (экран) управления устройствами.

DevicesFindActivity – класс, реализующий активность (экран) поиска и добавления найденных устройств.

DialogDeviceFindFragment – класс для диалогового окна добавления найденного устройства.

DialogDeviceNameChangeFragment – класс для диалогового окна изменения имени устройства.

GroupsActivity – класс, реализующий активность (экран) с группами.

MainActivity – класс, реализующий активность (экран) с отслеживанием участников тренировки.

NotifyActivity – класс, реализующий активность (экран) управления оповещениями.

StorageActivity – класс, реализующий активность (экран) управления хранилищем.

UsersActivity – класс, реализующий активность (экран) управления пользователями.

UsersEditActivity – класс, реализующий активность (экран) редактирования/добавления пользователей.

BleControlManager – класс для взаимодействия с Bluetooth LE устройствами.

DevicesFinderViewModel – класс для поиска Bluetooth LE устройств.

MainActivityViewModel – класс для подключения к устройствам участников активной группы.

Notification – класс, реализующий систему оповещений.

App – класс для реализации синглтон классов.

Repositories – статичный класс для инициализации базы данных и предоставления доступа к содержимому базы данных.

5 ТЕСТИРОВАНИЕ

5.1 Методологии тестирования

Для сдачи в эксплуатацию программного обеспечения необходимо убедиться в том, что все предъявленные требования к ПО выполняются. Для этого проведем функциональное и нефункциональное тестирование приложения на физических Android устройствах (эмуляция устройств не подходит, поскольку необходима проверка корректной работы Bluetooth). Для тестирования были использованы следующие устройства:

– OnePlus 8T (Android 12);

– Meizu MX4 Pro (Android 5.0).

Выбор данных устройств обусловлен разницей в версии Android (одно из нефункциональных требований).

В качестве подключаемых устройств были использованы Bluetooth LE устройства с поддержкой GATT профилей:

– Xiaomi Mi Band 3;

– Xiaomi Mi Band 4;

– Xiaomi Mi Band 6;

– аппаратная часть.

5.2 Проведение процедуры тестирования

Результаты прохождения тестирования представлены в виде таблицы 5.

Таблица 5 – Результаты тестирования

Требование	Описание теста	Ожидаемый результат	Итог
Поиск и добавление нового устройства	Открыть экран управления устройствами. Нажать на плавающую кнопку «+». Откроется экран добавления нового устройства. Ожидать появления новых устройств в радиусе действия Bluetooth. Добавить обнаруженное устройство, изменив имя на уже существующее имя, затем добавить устройство с указанием уникального имени.	Должны быть обнаружены устройства отсутствующие в списке добавленных ранее устройств. Попытка добавить устройство с существующим именем должна привести к появлению соответствующего уведомления. Устройство с уникальным именем должно корректно добавиться.	Успех
Поиск среди добавленных устройств	Открыть экран управления устройствами. Нажать на иконку «лупы», ввести поисковый запрос.	Искомое устройство должно быть найдено.	Успех
Удаление устройства	Открыть экран управления устройствами. Нажать на иконку с тремя точками. Выбрать «Удалить» из выпадающего списка. Проверить, что удаленное устройство было отвязано от всех пользователей.	После нажатия кнопки «Удалить» устройство должно пропасть из списка. У всех пользователей с данным устройством значение «Персональное устройство» должно быть обнулено.	Успех
Изменение имени устройства	Открыть экран управления устройствами. Нажать на иконку с тремя точками. Выбрать «Изменить» из выпадающего списка. Попытаться ввести пустую строку. Ввести уже существующее имя. Ввести новое имя.	При нажатии на кнопку «Изменить» должно отобразиться модальное окно с формой изменения имени устройства. При попытке ввести устройство с пустым именем кнопка «Сохранить» становится неактивной. При попытке ввести уже существующее имя должно выводиться уведомление об ошибке операции. При вводе нового имени устройство в списке должно быть успешно изменено.	Успех

Продолжение таблицы 5

Добавление нового пользователя	Открыть экран управления пользователями. Нажать на плавающую кнопку «+». Откроется экран с формой для добавления нового пользователя. Попытаться ввести некорректные данные и сохранить (пустая строка имени, фамилии, некорректная дата, не выбран пол). Ввести корректные данные и сохранить пользователя	При попытке ввести некорректные данные должно выводиться предупреждение о неверных данных. После ввода корректных данных и нажатия кнопки «Сохранить» новый пользователь должен появиться в списке пользователей.	Успех
Поиск среди добавленных пользователей	Открыть экран управления пользователями. Нажать на иконку «лупы», ввести поисковый запрос.	Искомый пользователь должен быть найден.	Успех
Удаление пользователя	Открыть экран управления пользователями. Нажать на иконку с тремя точками. Выбрать «Удалить» из выпадающего списка.	После нажатия кнопки «Удалить» пользователь должен пропасть из списка.	Успех
Изменения информации о пользователе	Открыть экран управления пользователями. Нажать на иконку с тремя точками. Выбрать «Изменить» из выпадающего списка. Попытаться ввести некорректные данные и сохранить. Ввести корректные данные и сохранить.	При попытке ввести некорректные данные должно выводиться предупреждение о неверных данных. После ввода корректных данных и нажатия кнопки «Сохранить» пользователь в списке пользователей должен быть отображен с учетом обновлённых данных.	Успех
Добавление новой группы	Открыть экран управления группами. Нажать на плавающую кнопку «+». Откроется модальное окно с формой ввода имени группы. Ввести пустое имя группы. Ввести существующее имя группы. Ввести уникальное имя группы.	При попытке ввести пустую группу кнопка «Сохранить» должна быть неактивна. При попытке сохранить группу с существующем именем, должно отобразиться сообщение об ошибке добавления. После успешного добавления группы с уникальным именем, она должна быть добавлена в списке групп .	Успех
Поиск среди добавленных групп	Открыть экран управления группами. Нажать на иконку «лупы», ввести поисковый запрос.	Искомая группа должна быть найдена.	Успех

Продолжение таблицы 5

Удаление группы	Открыть экран управления группами. Нажать на иконку с тремя точками. Выбрать «Удалить» из выпадающего списка.	После нажатия кнопки «Удалить» группа должна пропасть из списка.	Успех
Изменение имени группы	Открыть экран управления группами. Нажать на иконку с тремя точками. Выбрать «Изменить» из выпадающего списка. Попытаться ввести пустую строку. Ввести уже существующее имя. Ввести новое имя.	При нажатии на кнопку «Изменить» должно отобразиться модальное окно с формой изменения имени группы. При попытке ввести устройство с пустым именем кнопка «Сохранить» становится неактивной. При попытке ввести уже существующее имя должно выводиться уведомление об ошибке операции. При вводе нового имени группа в списке должна быть успешно изменена.	Успех
Выбор активной группы спортсменов для отслеживания	Открыть экран управления группами. Нажать на группу.	После нажатия на группу должен быть осуществлен переход на главный экран. Пользователи в группе должны отображаться в списке на главном экране.	Успех
Просмотр статистики использования хранилища	Посчитать количество устройств, пользователей и групп. Открыть экран управления хранилищем. Сравнить посчитанное количество с выводимой информацией.	Информация о количестве пользователей, групп и устройств должно совпадать с фактическим количеством.	Успех
Возможность удаления всех добавленных сущностей	Открыть экран управления хранилищем. Нажать на кнопку с изображением мусорного бака напротив пользователей / групп / браслетов. Проверить успешность удаления пользователей / групп / браслетов просмотром соответствующих экранов.	После нажатия кнопки с изображением мусорного бака число пользователей / групп / браслетов должно обнулиться. Экраны с пользователями / группами / браслетами должны содержать пустые списки.	Успех

Продолжение таблицы 5

Управление системой оповещений	Открыть экран управления оповещениями. Активировать / деактивировать оповещения. Нажать кнопку «Тестовое оповещение» .	При активной системе оповещений после нажатия кнопки «Тестовое оповещение» должно прийти оповещение. При выключенной системе оповещения после нажатия кнопки «Тестовое оповещение» не должно ничего произойти.	Успех
Численное отображение пульса пользователя в составе отслеживаемой группы	Выбрать активную группу для отслеживания. Запустить тренировку нажатием кнопки «Плей». Сравнить показания на физическом устройстве и на главном экране.	Показания выводимые устройством и показания в списке главного экрана должны соответствовать.	Успех
Графическое представление зоны ЧСС и проверка численного отображения	Заранее рассчитать по формуле 1 или 2 максимальную ЧСС. Выбрать активную группу для отслеживания. Запустить тренировку нажатием кнопки «Плей». Повысить пульс у проверяемого пользователя.	При достижении определенной зоны ЧСС графический индикатор зоны ЧСС должен показывать корректные значения. При нажатии на пользователя должно быть выведено численное значение уровня нагрузки. Необходимо сравнить выводимые показатели с реальными.	Успех
Отображение имени и фамилии пользователя	Выбрать активную группу для отслеживания. Поименно сравнить состав группы и выводимых участников тренировки на главном экране.	Имена и фамилии участников тренировки должны соответствовать составу выбранной группы.	Успех

Окончание таблицы 5

Графическое представление уровня заряда и силы сигнала и проверка численного отображения	Выбрать активную группу для отслеживания. Посмотреть уровень заряда подключенного устройства и сравнить с выводимыми значениями. По нажатию на индикатор батареи должно быть выедено численное значение уровня заряда. Постепенно удаляться от устройства и наблюдать за уровнем сигнала. По нажатию на индикатор уровня сигнала должно быть выедено численное значение уровня сигнала.	Значение уровня заряда должно соответствовать фактическому уровню заряда, отображаемого на устройстве. При удалении от устройства уровень сигнала должен уменьшаться.	Успех
Сортировка отображения выводимой информации об отслеживаемой группе	Выбрать активную группу для отслеживания. Запустить тренировку нажатием кнопки «Плей». Включать различные режимы сортировки.	Элементы списка должны быть корректно отсортированы согласно типу сортировки.	Успех
Привязка временного устройства к пользователю	Выбрать активную группу для отслеживания. Если у участника нет персонального устройства, то должна появиться возможность по нажатию на кнопку с изображением фитнес-трекера выбрать из списка доступных устройств временное устройство.	После привязки устройства кнопка привязки устройства должна пропасть.	Успех

Все функциональные требования были выполнены. Нефункциональные требования были также подтверждены в ходе выполнения тестирования функциональных требований:

– тестирование подводилось на ОС Android 5.0 (минимальная версия согласно нефункциональным требованиям);

– приложение полностью написано на языке программирования Kotlin;

– интерфейс приложения адаптирован для работы в портретном режиме;

– все тестовые Bluetooth LE устройства корректно передали данные о ЧСС;

– Итоговый размер приложения составил 8 МБ (рисунок 38).

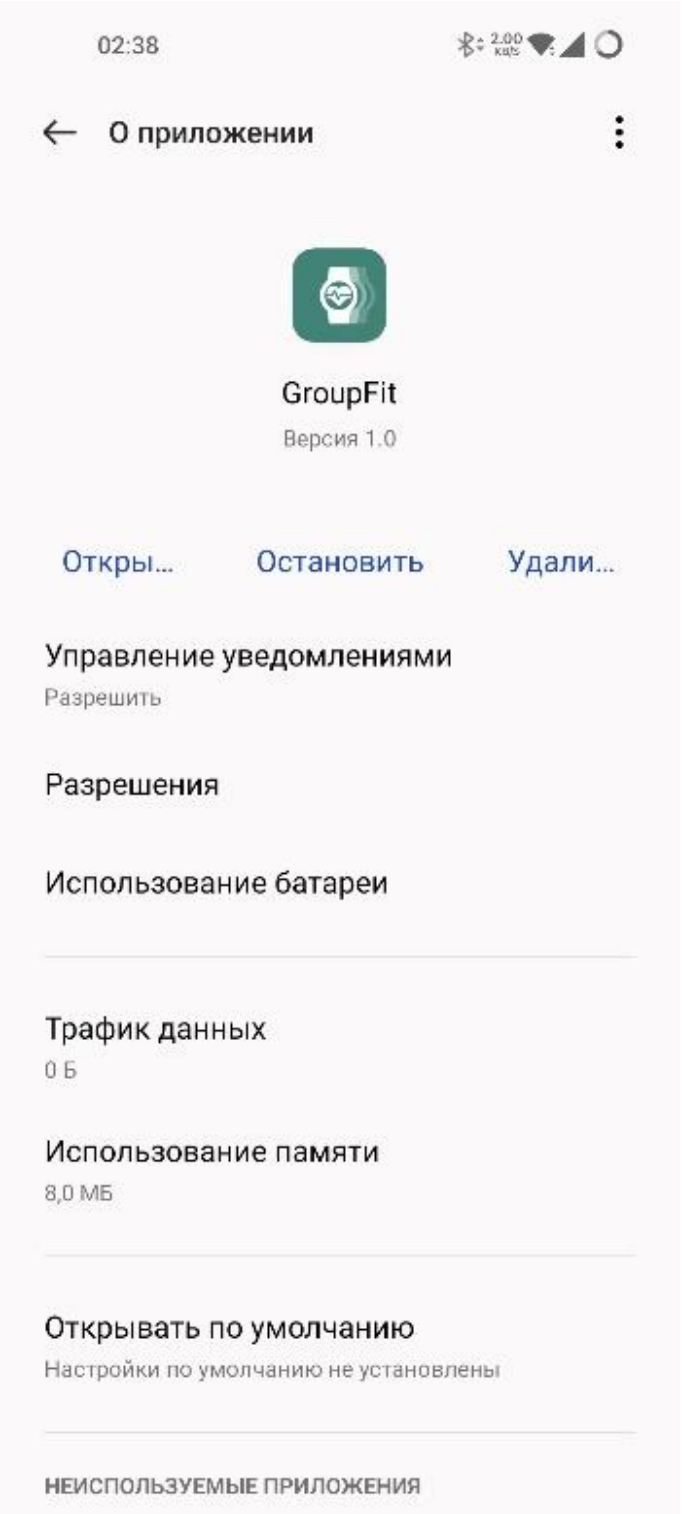

Рисунок 38 – Сведения о приложении

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы проведен анализ программных продуктов для чтения и обработки данных с портативных носимых устройств, считывающих параметры организма во время физических нагрузок. На основании проведенного анализа конкурирующих продуктов предложены оптимальные технические решения, требуемые для реализации программного комплекса. Выдвинуты функциональные и нефункциональные требования. С учетом предложенных решений и требований было спроектировано и реализовано требуемое программное обеспечение. Результатом работы стало мобильное приложение, отвечающее всем поставленным требованиям, что было доказано на этапе тестирования.

В дальнейшем возможна реализация интеграции с системой «Универис» для синхронизации данных о пользователях и группах, сохранения истории измерения ЧСС и локализация интерфейса приложения на другие языки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Статистика о ведущих причинах смертности и инвалидности во всем мире за 2000–2019 гг. – Текст : электронный // Всемирная организация здравоохранения : [официальный сайт]. – URL: https://www.who.int/ru/news/item/09-12-2020-who-reveals-leading-causes-of-death-and-disability-worldwide-2000-2019 (дата обращения: 14.04.2022).

2. Митрохин, О. В. Теоретическое обоснование оценки факторов риска здоровью в условиях самоизоляции / О. В. Митрохин, Н. А. Ермакова, Е. В. Белова. // Анализ риска здоровью. – 2021. – № 1. – С. 143-150.

3. Несчастные случаи на уроках физкультуры. – Текст : электронный // Российская газета : [сайт]. – URL: https://rg.ru/sujet/2292/ (дата обращения: 14.04.2022).

4. Fitzgerald, M. Runner's World The Cutting-Edge Runner: How to Use the Latest Science and Technology to Run Longer, Stronger, and Faster / M. Fitzgerald. – 1. – New York : Rodale Books, 2005. – 256 с. – Текст : непосредственный.

5. Polar Club. – Текст : электронный // App Store : [сайт]. – URL: https://apps.apple.com/ru/app/polar-club/id840071561 (дата обращения: 25.05.2022).

6. Polar Club приложение для отслеживания ЧСС во время групповых занятий фитнесом. – Текст : электронный // Polar Club : [официальный сайт]. – URL: https://www.polar.com/ru/club (дата обращения: 25.05.2022).

7. Myzone In Club Solution. – Текст : электронный // Myzone : [официальный сайт]. – URL: https://www.myzone.org/in-club-system (дата обращения: 25.05.2022).

8. Инструкция к Myzone In Club . – Текст : электронный // Myzone : [сайт]. – URL: https://gymshop.pro/MyZone_instruction.pdf (дата обращения: 25.05.2022).

9. Вопросы-ответы о Myzone In Club . – Текст : электронный // Myzonerussia : [сайт]. – URL: https://myzonerussia.ru/faq#!/tab/135343765-3 (дата обращения: 25.05.2022).

10. OnBeat app. – Текст : электронный // OnBeat : [официальный сайт]. – URL: https://www.onbeat.fit/interface (дата обращения: 25.05.2022).

11. OnBeat - group heart rate app. – Текст : электронный // App Store : [сайт]. – URL: https://apps.apple.com/us/app/onbeat/id1106772740 (дата обращения: 25.05.2022).

12. OnBeat compatible heart rate monitors. – Текст : электронный // OnBeat : [официальный сайт]. – URL: https://www.onbeat.fit/compatible-heart-rate-monitors (дата обращения: 25.05.2022).

13. OnBeat monthly subscription. – Текст : электронный // OnBeat : [официальный сайт]. – URL: https://www.onbeat.fit/monthly-subscription (дата обращения: 25.05.2022).

14. Selfloops Group Fitness. – Текст : электронный // SelfLoops : [официальный сайт]. – URL: https://www.selfloops.com/products/groupfitness.html (дата обращения: 25.05.2022).

15. Selfloops инструкция по подключению. – Текст : электронный // SelfLoops : [сайт]. – URL: https://support.selfloops.com/wp-content/uploads/2020/10/Selfloops-Group-Fitness-Plus-Android.pdf (дата обращения: 25.05.2022).

16. Squad Heart Rate. – Текст : электронный // App Store : [сайт]. – URL: https://apps.apple.com/ru/app/squad-heart-rate/id1483898877?uo=4&mt=8 (дата обращения: 25.05.2022).

17. Squad Heart Rate. – Текст : электронный // Squad Heart Rate : [официальный сайт]. – URL: https://trackteam.com.au/squad-heart-rate/ (дата обращения: 25.05.2022).

18. Operating System Market Share Worldwide. – Текст : электронный // Statcounter : [сайт]. – URL: https://gs.statcounter.com/os-market-share (дата обращения: 25.05.2022).

19. Android – платформа для всех. – Текст : электронный // Android : [официальный сайт]. – URL: https://www.android.com/intl/ru_ru/everyone/ (дата обращения: 25.05.2022).

20. Bluetooth Technology Overview. – Текст : электронный // Bluetooth : [официальный сайт]. – URL: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ (дата обращения: 25.05.2022).

21. Understanding Bluetooth Range. – Текст : электронный // Bluetooth : [официальный сайт]. – URL: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/range/ (дата обращения: 25.05.2022).

22. Teah FAQ. – Текст : электронный // Thisisant : [сайт]. – URL: https://www.thisisant.com/developer/resources/tech-faq/what-is-the-maximum-range/ (дата обращения: 25.05.2022).

23. Уилсон, Д. Руководство по проектированию и строительству легкоатлетических спортивных сооружений / Д. Уилсон, Р. Бриборн, Э. Гай. – Монако : Мультипринт, 2008. – 288 с. – Текст : непосредственный.

24. Assigned Numbers. – Текст : электронный // Bluetooth : [сайт]. – URL: https://www.bluetooth.com/specifications/assigned-numbers/ (дата обращения: 25.05.2022).

25. Heart rate service. – Текст : электронный // Bluetooth : [сайт]. – URL: https://www.bluetooth.com/specifications/specs/heart-rate-service-1-0/ (дата обращения: 25.05.2022).

26. Android Developers Docs. – Текст : электронный // Android documentation : [сайт]. – URL: https://developer.android.com/reference/android/bluetooth/le/package-summary (дата обращения: 25.05.2022).

27. Android API reference. – Текст : электронный // Android documentation : [сайт]. – URL: https://developer.android.com/reference (дата обращения: 25.05.2022).

28. Kotlin for Android. – Текст : электронный // Kotlin Lang : [сайт]. – URL: https://kotlinlang.org/docs/android-overview.html (дата обращения: 25.05.2022).

29. Kotlin is now Google’s preferred language for Android app development. – Текст : электронный // TechCrunch : [сайт]. – URL: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/ (дата обращения: 25.05.2022).

30. Data and file storage overview. – Текст : электронный // Android documentation : [сайт]. – URL: https://developer.android.com/training/data-storage (дата обращения: 25.05.2022).

31. Save data in a local database using Room. – Текст : электронный // Android documentation : [сайт]. – URL: https://developer.android.com/training/data-storage/room (дата обращения: 25.05.2022).

32. Tanaka, H. Age-predicted maximal heart rate revisited / H. Tanaka, K. D. Monahan, D. R. Seals. – Текст : непосредственный // Journal of the American College of Cardiology. – 2001. – № 1. – С. 153-156.

33. Что такое зоны ЧСС? – Текст : электронный // Polar : [сайт]. – URL: https://www.polar.com/ru/smart-coaching/what-are-heart-rate-zones (дата обращения: 25.05.2022).

34. Почему бег на высоком пульсе опасен для здоровья. – Текст : электронный // Российская газета : [сайт]. – URL: https://rg.ru/2015/06/30/puls-site.html (дата обращения: 25.05.2022).

35. Исходный код Android. – Текст : электронный // Google Git : [сайт]. – URL: https://android.googlesource.com/platform/external/bluetooth/bluedroid/+/master/include/bt_target.h#1428 (дата обращения: 25.05.2022).

36. Save key-value data. – Текст : электронный // Android documentation : [сайт]. – URL: https://developer.android.com/training/data-storage/shared-preferences (дата обращения: 25.05.2022).

ПРИЛОЖЕНИЕ А
Исходный код программы

Листинг 1 – Device

```
package com.anateon.groupfit.model.devices.entities

import java.io.Serializable

// Дата класс для хранения данных об устройстве
data class Device(
    // Уникальный идентификатор устройства
    val id: String,
    // Уникальное имя устройства
    val name: String
) : Serializable
```

Листинг 2 – DeviceDbEntity

```
package com.anateon.groupfit.model.devices.entities

import androidx.room.Entity
import androidx.room.PrimaryKey

// аннотация Entity (будет создана таблица в БД)
@Entity(
    // имя таблицы
    tableName = "devices",
    // уникальное иднесированное поле "name"
    indices = [
        androidx.room.Index("name", unique = true)
    ],
)

data class DeviceDbEntity(
    // первичный ключ ID устройства
    @PrimaryKey val id: String,
    // имя устройства
    val name: String
) {
    // преобразование в класс Device
    fun toDevice(): Device = Device(
        id = id,
        name = name
    )

    companion object {
        // преобразование из класса Device в класс DeviceDbEntity
        fun fromDevice(Device: Device): DeviceDbEntity = DeviceDbEntity(
            id = Device.id,
            name = Device.name
        )
    }
}
```

Листинг 3 – DevicesDao

```
package com.anateon.groupfit.model.devices

import androidx.room.*
import com.anateon.groupfit.model.devices.entities.DeviceDbEntity

// аннотация DAO (Room автоматически генерирует класс, реализующий интерфейс)
@Dao
interface DevicesDao {
    // получить устройство по ID
    @Query("SELECT * FROM devices WHERE id = :deviceId")
    fun getDeviceById(deviceId: Long): DeviceDbEntity?
```

```
    // получить все устройства
    @Query("SELECT * FROM devices")
    fun getDevices(): List<DeviceDbEntity>

    // добавить устройство
    @Insert
    fun addDevice(deviceDbEntity: DeviceDbEntity)

    // обновить устройство
    @Update(entity = DeviceDbEntity::class)
    fun updateDevice(device: DeviceDbEntity)

    // удалить устройство
    @Delete(entity = DeviceDbEntity::class)
    fun deleteDevice(device: DeviceDbEntity)

    // удалить все устройства
    @Query("DELETE FROM devices")
    fun deleteDevices()
}
```

Листинг 4 – DevicesRepository

```
package com.anateon.groupfit.model.devices

import com.anateon.groupfit.model.devices.entities.Device

// интерфейс описывающий методы для получения данных из таблицы устройств из базы
данных
interface DevicesRepository {
    fun getDeviceById(groupId: Long): Device?
    fun getDevices(): List<Device>
    fun addDevice(device: Device)
    fun updateDevice(device: Device)
    fun deleteDevice(device: Device)
    fun deleteDevices()
}
```

Листинг 5 – RoomDevicesRepository

```
package com.anateon.groupfit.model.devices

import com.anateon.groupfit.model.devices.entities.Device
import com.anateon.groupfit.model.devices.entities.DeviceDbEntity

// класс, реализующий методы интерфейса DevicesRepository
class RoomDevicesRepository(
    private val devicesDao: DevicesDao,
) : DevicesRepository {
    override fun getDeviceById(groupId: Long): Device? {
        return devicesDao.getDeviceById(groupId)?.toDevice()
    }

    override fun getDevices(): List<Device> {
        return devicesDao.getDevices().map { it.toDevice() }
    }

    override fun addDevice(device: Device) {
        devicesDao.addDevice(DeviceDbEntity.fromDevice(device))
    }

    override fun updateDevice(device: Device) {
        devicesDao.updateDevice(DeviceDbEntity.fromDevice(device))
    }

    override fun deleteDevice(device: Device) {
        devicesDao.deleteDevice(DeviceDbEntity.fromDevice(device))
    }
```

```
    override fun deleteDevices() {
        devicesDao.deleteDevices()
    }

}
```

Листинг 6 – Group

```
package com.anateon.groupfit.model.groups.entities

import java.io.Serializable

// дата класс для хранения данных об устройстве
data class Group(
    // уникальный ID группы
    val id: Long,
    // уникальное имя группы
    val name: String
) : Serializable
```

Листинг 7 – GroupDbEntity

```
package com.anateon.groupfit.model.groups.entities

import androidx.room.Entity
import androidx.room.PrimaryKey


// аннотация Entity (будет создана таблица в БД)
@Entity(
    // имя таблицы
    tableName = "groups",
    // уникальное иднесированное поле "name"
    indices = [
        androidx.room.Index("name", unique = true)
    ],
)
data class GroupDbEntity(
    // первичный ключ ID устройства (автоинкрементный)
    @PrimaryKey(autoGenerate = true) val id: Long,
    // имя устройства
    val name: String
) {
    // преобразование в класс Group
    fun toGroup(): Group = Group(
        id = id,
        name = name
    )

    companion object {
        // преобразование из класса Group в класс GroupDbEntity
        fun fromGroup(Group: Group): GroupDbEntity = GroupDbEntity(
            id = Group.id,
            name = Group.name
        )
    }
}
```

Листинг 8 – GroupsDao

```
package com.anateon.groupfit.model.groups

import androidx.room.*
import com.anateon.groupfit.model.groups.entities.GroupDbEntity

// аннотация DAO (Room автоматически генерирует класс, реализующий интерфейс)
@Dao
interface GroupsDao {
```

```
    // получить группу по ID
    @Query("SELECT * FROM groups WHERE id = :groupId")
    fun getGroupById(groupId: Long): GroupDbEntity?

    // получить все группы
    @Query("SELECT * FROM groups")
    fun getGroups(): List<GroupDbEntity>

    // добавить группу
    @Insert
    fun createGroup(group: GroupDbEntity)

    // обновить данные группы
    @Update(entity = GroupDbEntity::class)
    fun updateGroup(group: GroupDbEntity)

    // удалить группу
    @Delete(entity = GroupDbEntity::class)
    fun deleteGroup(group: GroupDbEntity)

    // удалить все группы
    @Query("DELETE FROM groups")
    fun deleteGroups()
}
```

Листинг 9 – GroupsRepository

```
package com.anateon.groupfit.model.groups

import com.anateon.groupfit.model.groups.entities.Group

// интерфейс описывающий методы для получения данных из таблицы групп из базы данных
interface GroupsRepository {
    fun getGroupById(groupId: Long): Group?
    fun getGroups(): List<Group>
    fun addGroup(group: Group)
    fun updateGroup(group: Group)
    fun deleteGroup(group: Group)
    fun deleteGroups()
}
```

Листинг 10 – RoomGroupsRepository

```
package com.anateon.groupfit.model.groups

import com.anateon.groupfit.model.groups.entities.Group
import com.anateon.groupfit.model.groups.entities.GroupDbEntity

// класс, реализующий методы интерфейса GroupsRepository
class RoomGroupsRepository(
    private val groupsDao: GroupsDao
) : GroupsRepository {
    override fun getGroupById(groupId: Long): Group? {
        return groupsDao.getGroupById(groupId)?.toGroup()
    }

    override fun getGroups(): List<Group> {
        return groupsDao.getGroups().map { it.toGroup() }
    }

    override fun addGroup(group: Group) {
        groupsDao.createGroup(GroupDbEntity.fromGroup(group))
    }

    override fun updateGroup(group: Group) {
        groupsDao.updateGroup(GroupDbEntity.fromGroup(group))
    }
```

```
    override fun deleteGroup(group: Group) {
        groupsDao.deleteGroup(GroupDbEntity.fromGroup(group))
    }

    override fun deleteGroups() {
        groupsDao.deleteGroups()
    }
}
```

Листинг 11 – DevicesFindService

```
package com.anateon.groupfit.model.recycler

import android.bluetooth.le.ScanResult

typealias DevicesFindListener = (devicesFind: List<ScanResult>) -> Unit

// класс для управления элементами списка найденных устройств
class DevicesFindService {
    // список найденных устройств
    private var devicesFind = mutableListOf<ScanResult>()
    // список слушателей
    private val listeners = mutableSetOf<DevicesFindListener>()

    // добавление слушателей
    fun addListener(listener: DevicesFindListener) {
        listeners.add(listener)
        listener.invoke(devicesFind)
    }

    // удаление слушателя
    fun removeListener(listener: DevicesFindListener) {
        listeners.remove(listener)
    }

    // оповестить слушателй об изменении
    private fun notifyChanges() {
        sortDevicesFind()
        listeners.forEach { it.invoke(devicesFind) }
    }

    // получить все устройства
    fun getDevices(): List<ScanResult> {
        return devicesFind
    }

    // отсортировать все устройства по rssi
    private fun sortDevicesFind() {
        devicesFind.sortByDescending { it.rssi }
    }

    // удалить устройства по ID
    fun removeDeviceById(id: String) {
        val indexToDelete = devicesFind.indexOfFirst { it.device.address == id }
        if (indexToDelete != -1) {
            devicesFind.removeAt(indexToDelete)
            notifyChanges()
        }
    }

    // назначить лист устройств
    fun setList(newDevices: List<ScanResult>) {
        devicesFind = newDevices.toMutableList()
        notifyChanges()
    }
}
```

Листинг 12 – DevicesService

```
package com.anateon.groupfit.model.recycler
import com.anateon.groupfit.model.devices.entities.Device
typealias DevicesListener = (devices: List<Device>) -> Unit
// класс для управления элементами списка устройств
class DevicesService {
    // список устройств
    private var devices = mutableListOf<Device>()
    // временный список устройств (для сортировки)
    private var tmpDevices = mutableListOf<Device>()
    // список слушатели
    private val listeners = mutableSetOf<DevicesListener>()

    // добавить слушателя
    fun addListener(listener: DevicesListener) {
        listeners.add(listener)
        listener.invoke(devices)
    }

    // удалить слушателя
    fun removeListener(listener: DevicesListener) {
        listeners.remove(listener)
    }

    // оповестить слушателей об изменениях
    private fun notifyChanges() {
        listeners.forEach { it.invoke(devices) }
    }

    // получить список всех устройств
    fun getDevices(): List<Device> {
        return devices
    }

    // удалить устройство
    fun deleteDevice(device: Device) {
        val indexToDelete =
            devices.indexOfFirst { it.id == device.id }
        if (indexToDelete != -1) {
            devices.removeAt(indexToDelete)
            notifyChanges()
        }
        if (tmpDevices.isNotEmpty()) {
            tmpDevices.remove(device)
        }
    }

    // найти устройство
    fun search(searchString: String) {
        if (searchString == "") {
            if (tmpDevices.isNotEmpty()) {
                devices = tmpDevices.toMutableList()
                tmpDevices.clear()
                notifyChanges()
            }
        } else {
            if (tmpDevices.isEmpty()) {
                tmpDevices = devices.toMutableList()
            }
            val searchModString: String = searchString.lowercase()
            devices.clear()
            tmpDevices.forEach {
                if (it.name.lowercase().contains(searchModString)) {
                    devices.add(it)
                }
            }
```

```
            notifyChanges()
        }
    }

    // назначить новый лист устройств
    fun setList(newDevices: List<Device>) {
        devices = newDevices.toMutableList()
        notifyChanges()
    }
}
```

Листинг 13 – GroupsService

```
package com.anateon.groupfit.model.recycler

import com.anateon.groupfit.model.groups.entities.Group

typealias GroupsListener = (users: List<Group>) -> Unit

// класс для управления элементами списка групп
class GroupsService {
    // список групп
    private var groups = mutableListOf<Group>()
    // временный список групп (для сортировки)
    private var tmpGroups = mutableListOf<Group>()
    // список слушателей
    private val listeners = mutableSetOf<GroupsListener>()

    // добавить слушателя
    fun addListener(listener: GroupsListener) {
        listeners.add(listener)
        listener.invoke(groups)
    }

    // удалить слушателя
    fun removeListener(listener: GroupsListener) {
        listeners.remove(listener)
    }

    // оповестить слушателя об изменениях
    private fun notifyChanges() {
        listeners.forEach { it.invoke(groups) }
    }

    // удалить группу
    fun deleteGroup(group: Group) {
        val indexToDelete =
            groups.indexOfFirst { it.id == group.id }
        if (indexToDelete != -1) {
            groups.removeAt(indexToDelete)
            notifyChanges()
        }
        if (tmpGroups.isNotEmpty()) {
            tmpGroups.remove(group)
        }
    }

    // найти группу
    fun search(searchString: String) {
        if (searchString == "") {
            if (tmpGroups.isNotEmpty()) {
                groups = tmpGroups.toMutableList()
                tmpGroups.clear()
                notifyChanges()
            }
        } else {
            if (tmpGroups.isEmpty()) {
                tmpGroups = groups.toMutableList()
```

```
            }
            val searchModString: String = searchString.lowercase()
            groups.clear()
            tmpGroups.forEach {
                if (it.name.lowercase().contains(searchModString)) {
                    groups.add(it)
                }
            }
            notifyChanges()
        }
    }

    // установить новый список групп
    fun setList(newGroups: List<Group>) {
        groups = newGroups.toMutableList()
        notifyChanges()
    }
}
```

Листинг 14 – MainInfo

```
package com.anateon.groupfit.model.recycler

import java.util.*

// дата класс, содержащий информацию об элементе списка главного экрана
data class MainInfo(
    // ID пользователя
    val userId: Long,
    // ID устройства
    var deviceId: String?,
    // имя пользователя
    val name: String,
    // фамилия ползователя
    val lastName: String,
    // ЧСС пользователя
    var heartRate: Int,
    // максимально допустимый ЧСС пользователя
    val maxHeartRate: Int,
    // уровень сигнала
    var signaldBm: Int,
    // уровень заряда
    var chargeLevel: Int,
    // статус подключения устройства (для обновления данных)
    var updateStatus: Boolean,
    // время последнего удачного обновления данных
    var updateTime: GregorianCalendar?,
    // если последнее данных было неудачным
    var errorStatus: Boolean
)
```

Листинг 15 – MainInfoService

```
package com.anateon.groupfit.model.recycler

import com.anateon.groupfit.model.users.entities.Gender
import com.anateon.groupfit.model.users.entities.User
import java.util.*

typealias MainInfoListener = (mainInfo: List<MainInfo>) -> Unit

// возможные варианты сортировки
enum class SortState {
    None,
    Name,
    HeartRate,
    MaxHeartRate
}
```

```
class MainInfoService {
    // список активных пользователей
    private var mainInfo = mutableListOf<MainInfo>()
    // режим сортировки
    private var sortMode: SortState = SortState.MaxHeartRate
    // список слушателей
    private val listeners = mutableSetOf<MainInfoListener>()

    // сортировка пользователей
    private fun sortUsers(sortType: SortState = SortState.None) {
        when (sortType) {
            SortState.Name -> {
                if (sortMode == sortType)
                    mainInfo.sortBy { it.name + it.lastName }
            }
            SortState.HeartRate -> {
                if (sortMode == sortType)
                    mainInfo.sortByDescending { it.heartRate }
            }
            SortState.MaxHeartRate -> {
                if (sortMode == sortType)
                    mainInfo.sortBy { it.maxHeartRate.toDouble() / it.heartRate }
            }
            else -> {
                when (sortMode) {
                    SortState.Name -> mainInfo.sortBy { it.name + it.lastName }
                    SortState.HeartRate -> mainInfo.sortByDescending { it.heartRate }
                    SortState.MaxHeartRate -> mainInfo.sortBy {
it.maxHeartRate.toDouble() / it.heartRate }
                    else -> return
                }
            }
        }
    }

    // добавление слушателя
    fun addListener(listener: MainInfoListener) {
        listeners.add(listener)
        listener.invoke(mainInfo)
    }

    // удаление слушателя
    fun removeListener(listener: MainInfoListener) {
        listeners.remove(listener)
    }

    // оповещение слушателей об изменении
    private fun notifyChanges() {
        listeners.forEach { it.invoke(mainInfo) }
    }

    // получение списка с активными пользователями
    fun getMainInfo(): List<MainInfo> {
        return mainInfo
    }

    // получение кол-ва активных пользователей
    fun getCount(): Int {
        return mainInfo.count()
    }
    // преобразование листа пользователей в лист активных пользователей
    fun setMainInfoUsersFromUser(newUsers: List<User>) {
        this.mainInfo.clear()
        this.mainInfo = newUsers.mapTo(mainInfo) {
            // расчет возраста человека
            val now = GregorianCalendar.getInstance()
            var res: Int = now.get(Calendar.YEAR) - it.birthday.get(Calendar.YEAR)
            if (it.birthday.get(Calendar.MONTH) > now[Calendar.MONTH]
```

```
                || it.birthday.get(Calendar.MONTH) == now[Calendar.MONTH]
                && it.birthday.get(Calendar.DAY_OF_MONTH) > now[Calendar.DAY_OF_MONTH]
            ) {
                res--
            }
            // расчет максимального пусльа взависимости от пола
            val maxHR = if (it.gender == Gender.MALE){
                (209 - 0.7 * res).toInt()
            } else{
                (214 - 0.8 * res).toInt()
            }
            MainInfo(
                userId = it.id,
                deviceId = it.deviceId,
                name = it.name,
                lastName = it.lastName,
                heartRate = 0,
                maxHeartRate = maxHR,
                signaldBm = 0,
                chargeLevel = 0,
                updateStatus = false,
                updateTime = null,
                errorStatus = false
            )
        }.toMutableList()
        notifyChanges()
    }

    // установка статуса обновления
    fun setUpdateStatus(
        mainInfo: MainInfo,
        isUpdate: Boolean,
        updeteTime: GregorianCalendar? = null
    ) {
        val indexToUpdate = this.mainInfo.indexOfFirst { it.userId == mainInfo.userId
}
        if (indexToUpdate != -1) {
            this.mainInfo[indexToUpdate].updateStatus = isUpdate
            this.mainInfo[indexToUpdate].errorStatus = false
            if (updeteTime != null) {
                this.mainInfo[indexToUpdate].updateTime = updeteTime
            } else {
                if (!isUpdate) {
                    this.mainInfo[indexToUpdate].errorStatus = true
                }
            }
            notifyChanges()
        }
    }

    // установка новой ЧСС
    fun setUserHeartRate(mainInfo: MainInfo, newHeartRate: Int) {
        val indexToUpdate = this.mainInfo.indexOfFirst { it.userId == mainInfo.userId
}
        if (indexToUpdate != -1) {
            this.mainInfo[indexToUpdate].heartRate = newHeartRate
            sortUsers(SortState.HeartRate)
            notifyChanges()
        }
    }

    // установка уровня заряда
    fun setUserBattery(mainInfo: MainInfo, newBattery: Int) {
        val indexToUpdate = this.mainInfo.indexOfFirst { it.userId == mainInfo.userId
}
        if (indexToUpdate != -1) {
            this.mainInfo[indexToUpdate].chargeLevel = newBattery
            notifyChanges()
```

```
        }
    }

    // установка нового значения RSSI (сила сигнала)
    fun setUserRssi(mainInfo: MainInfo, newRssi: Int) {
        val indexToUpdate = this.mainInfo.indexOfFirst { it.userId == mainInfo.userId
}
        if (indexToUpdate != -1) {
            this.mainInfo[indexToUpdate].signaldBm = newRssi
            notifyChanges()
        }
    }

    // привязка нового временного устройства к пользователю
    fun setDevice(mainInfo: MainInfo, newDeviceId: String?) {
        val indexToUpdate = this.mainInfo.indexOfFirst { it.userId == mainInfo.userId
}
        if (indexToUpdate != -1) {
            this.mainInfo[indexToUpdate].deviceId = newDeviceId
            notifyChanges()
        }
    }

    // установка нового режима сортировки
    fun setSortMode(newSortMode: SortState) {
        sortMode = newSortMode
        sortUsers()
        notifyChanges()
    }
}
```

Листинг 16 – UsersService

```
package com.anateon.groupfit.model.recycler

import com.anateon.groupfit.model.users.entities.User

typealias UsersListener = (users: List<User>) -> Unit

// класс для управления элементами списка пользователей
class UsersService {
    // список пользователей
    private var users = mutableListOf<User>()
    // временный список пользователей
    private var tmpUsers = mutableListOf<User>()
    // список слушателей
    private val listeners = mutableSetOf<UsersListener>()

    // добавление слушателя
    fun addListener(listener: UsersListener) {
        listeners.add(listener)
        listener.invoke(users)
    }

    // удаление слушателя
    fun removeListener(listener: UsersListener) {
        listeners.remove(listener)
    }

    // оповестить слушателей об изменениях
    private fun notifyChanges() {
        listeners.forEach { it.invoke(users) }
    }

    // удалить пользователя
    fun deleteUser(user: User) {
        val indexToDelete =
            users.indexOfFirst { it.id == user.id }
```

```
        if (indexToDelete != -1) {
            users.removeAt(indexToDelete)
            notifyChanges()
        }
        if (tmpUsers.isNotEmpty()) {
            tmpUsers.remove(user)
        }
    }

    // поиск среди пользователей
    fun search(searchString: String) {
        if (searchString == "") {
            if (tmpUsers.isNotEmpty()) {
                users = tmpUsers.toMutableList()
                tmpUsers.clear()
                notifyChanges()
            }
        } else {
            if (tmpUsers.isEmpty()) {
                tmpUsers = users.toMutableList()
            }
            val searchModString: String = searchString.lowercase()
            users.clear()
            tmpUsers.forEach {
                if ("${it.name} ${it.lastName}".lowercase().contains(searchModString))
{
                    users.add(it)
                }
            }
            notifyChanges()
        }
    }

    // установить список пользователей
    fun setList(newUsers: List<User>) {
        users = newUsers.toMutableList()
        notifyChanges()
    }

}
```

Листинг 17 – AppDatabase

```
package com.anateon.groupfit.model.room

import androidx.room.Database
import androidx.room.RoomDatabase
import com.anateon.groupfit.model.devices.DevicesDao
import com.anateon.groupfit.model.groups.GroupsDao
import com.anateon.groupfit.model.devices.entities.DeviceDbEntity
import com.anateon.groupfit.model.groups.entities.GroupDbEntity
import com.anateon.groupfit.model.users.UsersDao
import com.anateon.groupfit.model.users.entities.UserDbEntity

// параметры БД
@Database(
    // версия БД
    version = 1,
    // сущности БД
    entities = [
        UserDbEntity::class,
        GroupDbEntity::class,
        DeviceDbEntity::class
    ]
)
abstract class AppDatabase : RoomDatabase() {
    abstract fun getUsersDao(): UsersDao
    abstract fun getGroupsDao(): GroupsDao
```

```
    abstract fun getDevicesDao(): DevicesDao
}
```

Листинг 18 – User

```
package com.anateon.groupfit.model.users.entities

import java.io.Serializable
import java.util.*

enum class Gender {
    MALE,
    FEMALE
}

// дата класс для хранения данных о пользователе
data class User(
    // ID пользователия
    val id: Long,
    // имя пользователя
    val name: String,
    // фамилия
    val lastName: String,
    // пол
    val gender: Gender,
    // дата рождения
    val birthday: GregorianCalendar,
    // ID группы
    val groupId: Long?,
    // ID устройства
    val deviceId: String?
) : Serializable
```

Листинг 19 – UserDbEntity

```
package com.anateon.groupfit.model.users.entities

import androidx.room.Entity
import androidx.room.ForeignKey
import androidx.room.Index
import androidx.room.PrimaryKey
import com.anateon.groupfit.model.devices.entities.DeviceDbEntity
import com.anateon.groupfit.model.groups.entities.GroupDbEntity
import java.util.*

// аннотация Entity (будет создана таблица в БД)
@Entity(
    // имя таблицы
    tableName = "users",
    // уникальное иднесированное поле deviceId
    indices = [
        Index("deviceId", unique = true)
    ],
    // внешние ключи
    foreignKeys = [
        ForeignKey(
            entity = GroupDbEntity::class,
            parentColumns = ["id"],
            childColumns = ["groupId"],
            onDelete = ForeignKey.SET_NULL,
            onUpdate = ForeignKey.CASCADE,
        ),
        ForeignKey(
            entity = DeviceDbEntity::class,
            parentColumns = ["id"],
            childColumns = ["deviceId"],
            onDelete = ForeignKey.SET_NULL,
            onUpdate = ForeignKey.CASCADE
```

```
        ),
    ]
)
data class UserDbEntity(
    // первичный ключ ID пользователя (автоинкрементный)
    @PrimaryKey(autoGenerate = true) val id: Long,
    // имя пользователя
    val name: String,
    // фамилия пользователя
    val lastName: String,
    // пол пользователя
    val gender: Boolean,
    // дата рождения
    val birthday: Long,
    // ID группы пользователя
    val groupId: Long?,
    // ID личного устройства
    val deviceId: String?,
) {
    // преобразование в класс User
    fun toUser(): User {
        val date = GregorianCalendar()
        date.timeInMillis = birthday
        return User(
            id = id,
            name = name,
            lastName = lastName,
            gender = if (gender) Gender.MALE else Gender.FEMALE,
            birthday = date,
            groupId = groupId,
            deviceId = deviceId
        )
    }

    companion object {
        // преобразование из класса User в класс UserDbEntity
        fun fromUser(User: User): UserDbEntity = UserDbEntity(
            id = User.id,
            name = User.name,
            lastName = User.lastName,
            gender = User.gender == Gender.MALE,
            birthday = User.birthday.timeInMillis,
            groupId = User.groupId,
            deviceId = User.deviceId
        )
    }
}
```

Листинг 20 – RoomUsersRepository

```
package com.anateon.groupfit.model.users

import com.anateon.groupfit.model.users.entities.User
import com.anateon.groupfit.model.users.entities.UserDbEntity

// класс, реализующий методы интерфейса UsersRepository
class RoomUsersRepository(
    private val usersDao: UsersDao
) : UsersRepository {
    override fun findByGroupId(groupId: Long): List<User> {
        return usersDao.findByGroupId(groupId).map { it.toUser() }
    }

    override fun findByDeviceId(deviceId: String): User? {
        return usersDao.findByDeviceId(deviceId)?.toUser()
    }
```

```
    override fun updateUserInfo(user: User) {
        usersDao.updateUser(UserDbEntity.fromUser(user))
    }

    override fun createUser(user: User) {
        usersDao.createUser(UserDbEntity.fromUser(user))
    }

    override fun getUserById(userId: Long): User? {
        return usersDao.getById(userId)?.toUser()
    }

    override fun getUsers(): List<User> {
        return usersDao.getAll().map { it.toUser() }
    }

    override fun deleteUser(user: User) {
        usersDao.deleteUser(UserDbEntity.fromUser(user))
    }

    override fun deleteUsers() {
        usersDao.deleteUsers()
    }
}
```

Листинг 21 – UsersDao

```
package com.anateon.groupfit.model.users

import androidx.room.*
import com.anateon.groupfit.model.users.entities.UserDbEntity

// аннотация DAO (Room автоматически генерирует класс, реализующий интерфейс)
@Dao
interface UsersDao {
    // получить пользователей по ID группы
    @Query("SELECT * FROM users WHERE groupID = :groupId")
    fun findByGroupId(groupId: Long): List<UserDbEntity>

    // получить пользователя по ID устройства
    @Query("SELECT * FROM users WHERE deviceId = :deviceId")
    fun findByDeviceId(deviceId: String): UserDbEntity?

    // обновить пользователя
    @Update
    fun updateUser(user: UserDbEntity)

    // добавить пользователя
    @Insert
    fun createUser(userDbEntity: UserDbEntity)

    // получить всех пользователей
    @Query("SELECT * FROM users")
    fun getAll(): List<UserDbEntity>

    // получить пользователя по ID
    @Query("SELECT * FROM users WHERE id = :accountId")
    fun getById(accountId: Long): UserDbEntity?

    // удалить пользователя
    @Delete(entity = UserDbEntity::class)
    fun deleteUser(user: UserDbEntity)

    // удалить всех пользователей
    @Query("DELETE FROM users")
    fun deleteUsers()
}
```

Листинг 22 – UsersRepository

```
package com.anateon.groupfit.model.users

import com.anateon.groupfit.model.users.entities.User

// интерфейс описывающий методы для получения данных из таблицы пользователей из базы данных
interface UsersRepository {
    fun findByGroupId(groupId: Long): List<User>
    fun findByDeviceId(deviceId: String): User?
    fun updateUserInfo(user: User)
    fun createUser(user: User)
    fun getUserById(userId: Long): User?
    fun getUsers(): List<User>
    fun deleteUser(user: User)
    fun deleteUsers()
}
```

Листинг 23 – DevicesFindRecyclerAdapter

```
package com.anateon.groupfit.view.adapter

import android.annotation.SuppressLint
import android.bluetooth.le.ScanResult
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.anateon.groupfit.databinding.ItemDeviceFindBinding

// интерфейс для функций-обработчиков нажатий
interface DevicesFindRecyclerAdapterActionListener {
    fun onUserItemClick(deviceFind: ScanResult)
}

// класс для переиспользования элементов списка найденных устройств в RecyclerView
class DevicesFindRecyclerAdapter(
    private val actionListener: DevicesFindRecyclerAdapterActionListener
) : RecyclerView.Adapter<DevicesFindRecyclerAdapter.DevicesFindViewHolder>(),
View.OnClickListener {
    // список найденных устройств
    var devicesFind: List<ScanResult> = emptyList()
        set(newValue) {
            field = newValue
            notifyDataSetChanged()
        }

    // получить кол-во найденных устройств
    override fun getItemCount(): Int = devicesFind.size

    // создает ViewHolder, когда RecyclerView нуждается в этом
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
DevicesFindViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val binding = ItemDeviceFindBinding.inflate(inflater, parent, false)
        binding.root.setOnClickListener(this)
        return DevicesFindViewHolder(binding)
    }

    // метод для обновления ViewHolder
    @SuppressLint("MissingPermission")
    override fun onBindViewHolder(holder: DevicesFindViewHolder, position: Int) {
        val deviceFind = devicesFind[position]
        with(holder.binding) {
            holder.itemView.tag = deviceFind
            deviceName.text = deviceFind.device.name ?: "Без имени"
```

```
            deviceMac.text = deviceFind.device.address
            deviceSignal.text = "${deviceFind.rssi} dBm"
        }
    }

    // переопределение ViewHolder для добавления поля binding
    class DevicesFindViewHolder(
        val binding: ItemDeviceFindBinding
    ) : RecyclerView.ViewHolder(binding.root)

    // переопределение метода срабатывающего при нажатии ViewHolder
    override fun onClick(v: View) {
        val deviceFind = v.tag as ScanResult
        actionListener.onUserItemClick(deviceFind)
    }
}
```

Листинг 24 – DevicesRecyclerAdapter

```
package com.anateon.groupfit.view.adapter

import android.view.LayoutInflater
import android.view.Menu
import android.view.View
import android.view.ViewGroup
import android.widget.PopupMenu
import androidx.recyclerview.widget.RecyclerView
import com.anateon.groupfit.R
import com.anateon.groupfit.databinding.ItemDeviceBinding
import com.anateon.groupfit.model.devices.entities.Device

// интерфейс для функций-обработчиков нажатий
interface DevicesRecyclerAdapterActionListener {
    fun onDeviceDelete(device: Device)
    fun onDeviceEdit(device: Device)
}

// класс для переиспользования элементов списка устройств в RecyclerView
class DevicesRecyclerAdapter(
    private val actionListener: DevicesRecyclerAdapterActionListener
) : RecyclerView.Adapter<DevicesRecyclerAdapter.DevicesViewHolder>(),
View.OnClickListener {
    // список устройств
    var devices: List<Device> = emptyList()
        set(newValue) {
            field = newValue
            notifyDataSetChanged()
        }

    // получить кол-во устройств
    override fun getItemCount(): Int = devices.size

    // создает ViewHolder, когда RecyclerView нуждается в этом
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
DevicesViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val binding = ItemDeviceBinding.inflate(inflater, parent, false)
        binding.root.setOnClickListener(this)
        binding.deviceMore.setOnClickListener(this)
        return DevicesViewHolder(binding)
    }

    // метод для обновления ViewHolder
    override fun onBindViewHolder(holder: DevicesViewHolder, position: Int) {
        val device = devices[position]
        with(holder.binding) {
            holder.itemView.tag = device
            deviceMore.tag = device
```

```
            deviceMac.text = device.id
            deviceName.text = device.name
        }
    }

    // переопределение ViewHolder для добавления поля binding
    class DevicesViewHolder(
        val binding: ItemDeviceBinding
    ) : RecyclerView.ViewHolder(binding.root)

    // переопределение метода срабатывающего при нажатии ViewHolder
    override fun onClick(v: View) {
        when (v.id) {
            R.id.deviceMore -> {
                showPopupMenu(v)
            }
        }
    }

    // показ всплывающего меню при нажатии за иконку с тремя точками
    private fun showPopupMenu(view: View) {
        val popupMenu = PopupMenu(view.context, view)
        val device = view.tag as Device
        popupMenu.menu.add(0, ID_EDIT, Menu.NONE, "Изменить")
        popupMenu.menu.add(0, ID_DELETE, Menu.NONE, "Удалить")
        popupMenu.setOnMenuItemClickListener {
            when (it.itemId) {
                ID_EDIT -> {
                    actionListener.onDeviceEdit(device)
                }
                ID_DELETE -> {
                    actionListener.onDeviceDelete(device)
                }
            }
            return@setOnMenuItemClickListener true
        }
        popupMenu.show()
    }

    // константы itemId
    companion object {
        private const val ID_EDIT = 1
        private const val ID_DELETE = 2
    }
}
```

Листинг 25 – GroupsInfoRecyclerAdapter

```
package com.anateon.groupfit.view.adapter

import android.view.LayoutInflater
import android.view.Menu
import android.view.View
import android.view.ViewGroup
import android.widget.PopupMenu
import androidx.recyclerview.widget.RecyclerView
import com.anateon.groupfit.R
import com.anateon.groupfit.databinding.ItemGroupBinding
import com.anateon.groupfit.model.groups.entities.Group

// интерфейс для функций-обработчиков нажатий
interface GroupsRecyclerAdapterActionListener {
    fun onGroupDelete(group: Group)
    fun onGroupEdit(group: Group)
    fun onGroupItemClick(group: Group)
}

// класс для переиспользования элементов списка групп в RecyclerView
```

```
class GroupsInfoRecyclerAdapter(
    private val actionListener: GroupsRecyclerAdapterActionListener
) : RecyclerView.Adapter<GroupsInfoRecyclerAdapter.GroupsViewHolder>(),
View.OnClickListener {
    // список групп
    var groups: List<Group> = emptyList()
        set(newValue) {
            field = newValue
            notifyDataSetChanged()
        }

    // получить кол-во групп
    override fun getItemCount(): Int = groups.size

    // создает ViewHolder, когда RecyclerView нуждается в этом
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
GroupsViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val binding = ItemGroupBinding.inflate(inflater, parent, false)
        binding.root.setOnClickListener(this)
        binding.groupMore.setOnClickListener(this)
        return GroupsViewHolder(binding)
    }

    // метод для обновления ViewHolder
    override fun onBindViewHolder(holder: GroupsViewHolder, position: Int) {
        val group = groups[position]
        with(holder.binding) {
            holder.itemView.tag = group
            groupMore.tag = group
            name.text = group.name
        }
    }

    // переопределение ViewHolder для добавления поля binding
    class GroupsViewHolder(
        val binding: ItemGroupBinding
    ) : RecyclerView.ViewHolder(binding.root)

    // переопределение метода срабатывающего при нажатии ViewHolder
    override fun onClick(v: View) {
        val group = v.tag as Group
        when (v.id) {
            R.id.groupMore -> {
                showPopupMenu(v)
            }
            else -> {
                actionListener.onGroupItemClick(group)
            }
        }
    }

    // показ всплывающего меню при нажатии за иконку с тремя точками
    private fun showPopupMenu(view: View) {
        val popupMenu = PopupMenu(view.context, view)
        val group = view.tag as Group
        popupMenu.menu.add(0, ID_EDIT, Menu.NONE, "Изменить")
        popupMenu.menu.add(0, ID_DELETE, Menu.NONE, "Удалить")
        popupMenu.setOnMenuItemClickListener {
            when (it.itemId) {
                ID_EDIT -> {
                    actionListener.onGroupEdit(group)
                }
                ID_DELETE -> {
                    actionListener.onGroupDelete(group)
                }
            }
            return@setOnMenuItemClickListener true
```

```
        }
        popupMenu.show()
    }

    // константы itemId
    companion object {
        private const val ID_EDIT = 1
        private const val ID_DELETE = 2
    }
}
```

Листинг 26 – MainInfoRecyclerAdapter

```
package com.anateon.groupfit.view.adapter

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.View.INVISIBLE
import android.view.View.VISIBLE
import android.view.ViewGroup
import androidx.core.content.ContextCompat
import androidx.recyclerview.widget.RecyclerView
import com.anateon.groupfit.R
import com.anateon.groupfit.databinding.ItemMainInfoBinding
import com.anateon.groupfit.model.recycler.MainInfo

// интерфейс для функций-обработчиков нажатий
interface MainInfoRecyclerAdapterActionListener {
    fun onMainInfoMoreData(user: MainInfo)
    fun getChargeLvl(user: MainInfo)
    fun getSignalLvl(user: MainInfo)
    fun onButtonDeviceClick(user: MainInfo)
    fun onMainInfoMoreDataLong(user: MainInfo)
}

// класс для переиспользования элементов списка информации о участниках активной
группы в RecyclerView
class MainInfoRecyclerAdapter(
    private val context: Context,
    private val actionListener: MainInfoRecyclerAdapterActionListener
) : RecyclerView.Adapter<MainInfoRecyclerAdapter.MainInfoViewHolder>(),
View.OnClickListener,
    View.OnLongClickListener {
    // список участников активной группы
    var mainInfo: List<MainInfo> = emptyList()
        set(newValue) {
            field = newValue
            notifyDataSetChanged()
        }

    // получить кол-во участников активной группы
    override fun getItemCount(): Int = mainInfo.size

    // создает ViewHolder, когда RecyclerView нуждается в этом
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
MainInfoViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val binding = ItemMainInfoBinding.inflate(inflater, parent, false)
        binding.root.setOnClickListener(this)
        binding.backgroundBattery.setOnClickListener(this)
        binding.backgroundSignal.setOnClickListener(this)
        binding.addDeviceButton.setOnClickListener(this)
        binding.root.setOnLongClickListener(this)
        return MainInfoViewHolder(binding)
    }

    // метод для обновления ViewHolder
```

```
    override fun onBindViewHolder(holder: MainInfoViewHolder, position: Int) {
        val mainInfo1 = mainInfo[position]
        with(holder.binding) {
            holder.itemView.tag = mainInfo1
            backgroundBattery.tag = mainInfo1
            backgroundSignal.tag = mainInfo1
            addDeviceButton.tag = mainInfo1

            // если у пользователя нет своего девайса
            if (mainInfo1.deviceId == null) {
                // скрыть пульс, показать кнопку добавления устройства
                addDeviceButton.visibility = VISIBLE
                heartRate.visibility = INVISIBLE
            } else {
                // показать пульс, скрыть кнопку добавления устройства
                addDeviceButton.visibility = INVISIBLE
                heartRate.visibility = VISIBLE
            }

            heartRate.text = mainInfo1.heartRate.toString()
            // установка цветов цифры в зависимости от нагрузки пользователя
            when (mainInfo1.heartRate.toDouble() / mainInfo1.maxHeartRate.toDouble())
{
                in 0.6..0.7 -> heartRate.setTextColor(
                    ContextCompat.getColor(
                        context,
                        R.color.zone2
                    )
                )
                in 0.7..0.8 -> heartRate.setTextColor(
                    ContextCompat.getColor(
                        context,
                        R.color.zone3
                    )
                )
                in 0.8..0.9 -> heartRate.setTextColor(
                    ContextCompat.getColor(
                        context,
                        R.color.zone4
                    )
                )
                in 0.9..999.0 -> heartRate.setTextColor(
                    ContextCompat.getColor(
                        context,
                        R.color.zone5
                    )
                )
                else -> heartRate.setTextColor(ContextCompat.getColor(context,
R.color.zone1))
            }
            progressBar.max = mainInfo1.maxHeartRate - mainInfo1.maxHeartRate / 2
            progressBar.progress =
                progressBar.max - (mainInfo1.heartRate - mainInfo1.maxHeartRate / 2)
            fullName.text = "${mainInfo1.name} ${mainInfo1.lastName}"
            val factor = holder.itemView.context.resources.displayMetrics.density
            var params: ViewGroup.LayoutParams = batteryLayout.layoutParams
            params.height = (mainInfo1.chargeLevel.toDouble() * 0.24 * factor).toInt()
            batteryLayout.layoutParams = params
            params = signalLayout.layoutParams

            // графическое отображение уровня сигнала
            with(mainInfo1) {
                if (signaldBm <= -100)
                    params.width = 0
                if (signaldBm == 0)
                    params.width = 0
                else if (signaldBm >= -50)
                    params.width = (24 * factor).toInt()
```

```
                else
                    params.width = (2 * (signaldBm + 100) * 0.24 * factor).toInt()
            }
            // индикация подключения / ошибки подключения
            signalLayout.layoutParams = params
            if (mainInfo1.updateStatus) {
                statusConnectImage.visibility = VISIBLE
            } else {
                statusConnectImage.visibility = INVISIBLE
            }
            if (mainInfo1.errorStatus) {
                statusErrorImage.visibility = VISIBLE
            } else {
                statusErrorImage.visibility = INVISIBLE
            }
        }
    }

    // переопределение ViewHolder для добавления поля binding
    class MainInfoViewHolder(
        val binding: ItemMainInfoBinding
    ) : RecyclerView.ViewHolder(binding.root)

    // переопределение метода срабатывающего при нажатии ViewHolder
    override fun onClick(v: View) {
        val mainInfo1 = v.tag as MainInfo
        when (v.id) {
            R.id.backgroundBattery -> {
                actionListener.getChargeLvl(mainInfo1)
            }
            R.id.backgroundSignal -> {
                actionListener.getSignalLvl(mainInfo1)
            }
            R.id.addDeviceButton -> {
                actionListener.onButtonDeviceClick(mainInfo1)
            }
            else -> {
                actionListener.onMainInfoMoreData(mainInfo1)
            }
        }
    }

    // переопределение метода срабатывающего при долгом нажатии ViewHolder
    override fun onLongClick(v: View): Boolean {
        val mainInfo1 = v.tag as MainInfo
        actionListener.onMainInfoMoreDataLong(mainInfo1)
        return true
    }
}
```

Листинг 27 – UsersInfoRecyclerAdapter

```
package com.anateon.groupfit.view.adapter

import android.content.Context
import android.view.LayoutInflater
import android.view.Menu
import android.view.View
import android.view.ViewGroup
import android.widget.PopupMenu
import androidx.recyclerview.widget.RecyclerView
import com.anateon.groupfit.App
import com.anateon.groupfit.R
import com.anateon.groupfit.Repositories
import com.anateon.groupfit.databinding.ItemUserBinding
import com.anateon.groupfit.model.users.entities.User
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
```

```
import kotlinx.coroutines.withContext

// интерфейс для функций-обработчиков нажатий
interface UsersRecyclerAdapterActionListener {
    fun onUserItemClick(user: User)
    fun onUserDelete(user: User)
    fun onUserEdit(user: User)
}

// класс для переиспользования элементов списка пользователей в RecyclerView
class UsersInfoRecyclerAdapter(
    private val context: Context,
    private val actionListener: UsersRecyclerAdapterActionListener
) : RecyclerView.Adapter<UsersInfoRecyclerAdapter.UsersViewHolder>(),
View.OnClickListener {
    // корунтина для IO (ввода-вывода)
    private val ioScope
        get() = (context.applicationContext as App).ioScope


    // список пользователей
    var users: List<User> = emptyList()
        set(newValue) {
            field = newValue
            notifyDataSetChanged()
        }

    // получить кол-во пользователей
    override fun getItemCount(): Int = users.size

    // создает ViewHolder, когда RecyclerView нуждается в этом
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UsersViewHolder
{
        val inflater = LayoutInflater.from(parent.context)
        val binding = ItemUserBinding.inflate(inflater, parent, false)
        binding.root.setOnClickListener(this)
        binding.userMore.setOnClickListener(this)
        return UsersViewHolder(binding)
    }

    // метод для обновления ViewHolder
    override fun onBindViewHolder(holder: UsersViewHolder, position: Int) {
        val user = users[position]
        with(holder.binding) {
            holder.itemView.tag = user
            userMore.tag = user
            fullName.text = "${user.name} ${user.lastName}"
            if (user.groupId == null) {
                groupName.text = "-"
            } else {
                // ассинхронное получение групп (для отображеия имен)
                ioScope.launch {
                    val group =
Repositories.groupsRepository.getGroupById(user.groupId)
                    if (group == null) {
                        withContext(Dispatchers.Main) { groupName.text = "-" }
                    } else {
                        withContext(Dispatchers.Main) { groupName.text = group.name }
                    }
                }
            }
            // скрытие/отображение логотипа персонального устройства
            if (user.deviceId == null) {
                deviceLogo.visibility = View.GONE
            } else {
                deviceLogo.visibility = View.VISIBLE
            }
        }
```

```
    }

    // переопределение ViewHolder для добавления поля binding
    class UsersViewHolder(
        val binding: ItemUserBinding
    ) : RecyclerView.ViewHolder(binding.root)

    // переопределение метода срабатывающего при нажатии ViewHolder
    override fun onClick(v: View) {
        val user = v.tag as User
        when (v.id) {
            R.id.userMore -> {
                showPopupMenu(v)
            }
            else -> {
                actionListener.onUserItemClick(user)
            }
        }

    }
    // показ всплывающего меню при нажатии за иконку с тремя точками
    private fun showPopupMenu(view: View) {
        val popupMenu = PopupMenu(view.context, view)
        val user = view.tag as User
        popupMenu.menu.add(0, ID_EDIT, Menu.NONE, "Изменить")
        popupMenu.menu.add(0, ID_DELETE, Menu.NONE, "Удалить")
        popupMenu.setOnMenuItemClickListener {
            when (it.itemId) {
                ID_EDIT -> {
                    actionListener.onUserEdit(user)
                }
                ID_DELETE -> {
                    actionListener.onUserDelete(user)
                }
            }
            return@setOnMenuItemClickListener true
        }
        popupMenu.show()
    }

    // константы itemId
    companion object {
        private const val ID_EDIT = 1
        private const val ID_DELETE = 2
    }
}
```

Листинг 28 – DevicesActivity

```
package com.anateon.groupfit.view

import android.content.Intent
import android.os.Bundle
import android.view.Menu
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.SearchView
import androidx.recyclerview.widget.LinearLayoutManager
import com.anateon.groupfit.*
import com.anateon.groupfit.databinding.ActivityUsersBinding
import com.anateon.groupfit.model.devices.entities.Device
import com.anateon.groupfit.model.recycler.DevicesListener
import com.anateon.groupfit.model.recycler.DevicesService
import com.anateon.groupfit.view.adapter.DevicesRecyclerAdapter
import com.anateon.groupfit.view.adapter.DevicesRecyclerAdapterActionListener
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
```

```
// класс, реализующий активность (экран) управления устройствами
class DevicesActivity : AppCompatActivity(), SearchView.OnQueryTextListener {

    private lateinit var binding: ActivityUsersBinding
    private lateinit var adapter: DevicesRecyclerAdapter
    private lateinit var devices: List<Device>
    private val deviceListener: DevicesListener = {
        adapter.devices = it
    }

    private val ioScope
        get() = (applicationContext as App).ioScope
    private val devicesService: DevicesService
        get() = (applicationContext as App).devicesService

    // создание Activity
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityUsersBinding.inflate(layoutInflater)
        setContentView(binding.root)
        // коммуникация между активити. Назначение слушателя результата
        supportFragmentManager.setFragmentResultListener(
            DialogDeviceNameChangeFragment.REQUEST_KEY, this
        ) { _, _ ->
            ioScope.launch {
                devices = Repositories.devicesRepository.getDevices()
                withContext(Dispatchers.Main) { devicesService.setList(devices) }
            }
        }
        // инициализация RecyclerView
        adapter = DevicesRecyclerAdapter(object : DevicesRecyclerAdapterActionListener
{
            override fun onDeviceEdit(device: Device) {
                val dialog = DialogDeviceNameChangeFragment()
                val args = Bundle()
                args.putString("id", device.id)
                args.putString("name", device.name)
                dialog.arguments = args
                dialog.show(supportFragmentManager,
DialogDeviceNameChangeFragment.TAG)
            }
            override fun onDeviceDelete(device: Device) {
                ioScope.launch {
                    Repositories.devicesRepository.deleteDevice(device)
                }
                devicesService.deleteDevice(device)
            }
        })
        binding.recyclerView.layoutManager = LinearLayoutManager(this)
        binding.recyclerView.adapter = adapter
        devicesService.addListener(deviceListener)
        // получение устройств из БД
        ioScope.launch {
            devices = Repositories.devicesRepository.getDevices()
            withContext(Dispatchers.Main) { devicesService.setList(devices) }
        }
        // назначение обработчика события OnClick для плавающей кнопки
        binding.floatingActionButton.setOnClickListener {
            startActivity(Intent(this, DevicesFindActivity::class.java))
        }
    }

    // инициализация меню
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.search_menu, menu)
        val search = menu?.findItem(R.id.search_action)
        val searchView = search?.actionView as? SearchView
        searchView?.isSubmitButtonEnabled = true
```

```
        searchView?.setOnQueryTextListener(this)
        return true
    }

    // уничтножение активити
    override fun onDestroy() {
        super.onDestroy()
        devicesService.removeListener(deviceListener)
    }

    // при нажатии кнопки назад эмулируется нажатие onBackPressed
    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return true
    }

    // вызывается, когда пользователь отправляет поисковый запрос
    override fun onQueryTextSubmit(query: String?): Boolean {
        devicesService.search(query ?: "")
        return true
    }

    // после перезапуска активити обнавляется список устройств
    override fun onRestart() {
        ioScope.launch {
            devices = Repositories.devicesRepository.getDevices()
            withContext(Dispatchers.Main) { devicesService.setList(devices) }
        }
        super.onRestart()
    }

    // вызывается, когда пользователь изменяет строку поиска
    override fun onQueryTextChange(newText: String?): Boolean {
        devicesService.search(newText ?: "")
        return true
    }
}
```

Листинг 29 – DevicesFindActivity

```
package com.anateon.groupfit.view

import android.Manifest
import android.annotation.SuppressLint
import android.bluetooth.le.ScanResult
import android.os.Build
import android.os.Bundle
import android.util.Log
import androidx.activity.result.contract.ActivityResultContracts
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import com.anateon.groupfit.App
import com.anateon.groupfit.view.adapter.DevicesFindRecyclerAdapter
import com.anateon.groupfit.view.adapter.DevicesFindRecyclerAdapterActionListener
import com.anateon.groupfit.Repositories
import com.anateon.groupfit.databinding.ActivityDevicesFindBinding
import com.anateon.groupfit.model.recycler.DevicesFindListener
import com.anateon.groupfit.model.recycler.DevicesFindService
import com.anateon.groupfit.viewModel.DeviceViewModelFactory
import com.anateon.groupfit.viewModel.DevicesFinderViewModel
import kotlinx.coroutines.launch

// класс, реализующий активность (экран) поиска и добавления найденных устройств
class DevicesFindActivity : AppCompatActivity() {

    private lateinit var binding: ActivityDevicesFindBinding
    private lateinit var adapter: DevicesFindRecyclerAdapter
```

```
    private val deviceListener: DevicesFindListener = {
        adapter.devicesFind = it
    }

    private val viewModel: DevicesFinderViewModel by viewModels {
        DeviceViewModelFactory((applicationContext as App).adapterProvider)
    }

    private val ioScope
        get() = (applicationContext as App).ioScope
    private val devicesService: DevicesFindService
        get() = (applicationContext as App).devicesFindService

    // создание Activity
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // получение списка добавленных устройсв
        ioScope.launch {
            viewModel.pairedDevices = Repositories.devicesRepository.getDevices().map
{
                it.id
            }.toMutableList()
        }
        binding = ActivityDevicesFindBinding.inflate(layoutInflater)
        setContentView(binding.root)
        // получение разрешений, если они не получены
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            requestPermissionLauncher.launch(Manifest.permission.BLUETOOTH_SCAN)
            requestPermissionLauncher.launch(Manifest.permission.BLUETOOTH_CONNECT)
        }
        requestPermissionLauncher.launch(Manifest.permission.BLUETOOTH)
        requestPermissionLauncher.launch(Manifest.permission.ACCESS_FINE_LOCATION)
        // запуск сканирования устройств
        viewModel.startScan()
        // коммуникация между активити и диалоговым окном. Назначение слушателя
результата
        supportFragmentManager.setFragmentResultListener(
            DialogDeviceFindFragment.REQUEST_KEY, this
        ) { _, bundle ->
            val newDeviceId = bundle.getString("id")
            if (newDeviceId != null)
                devicesService.removeDeviceById(newDeviceId)
            finish()
        }
        // инициализация RecyclerView
        adapter =
            DevicesFindRecyclerAdapter(object :
DevicesFindRecyclerAdapterActionListener {
                @SuppressLint("MissingPermission")
                override fun onUserItemClick(deviceFind: ScanResult) {
                    val dialog = DialogDeviceFindFragment()
                    val args = Bundle()
                    args.putString("id", deviceFind.device.address)
                    args.putString("name", deviceFind.device.name)
                    dialog.arguments = args
                    dialog.show(supportFragmentManager, DialogDeviceFindFragment.TAG)
                }
            })
        binding.recyclerView.layoutManager = LinearLayoutManager(this)
        binding.recyclerView.adapter = adapter
        devicesService.addListener(deviceListener)
        // подписаться на обнаружение устройств
        subscribeOnViewModel()
    }

    // запрос разрешений
    private val requestPermissionLauncher =
        registerForActivityResult(
```

```
            ActivityResultContracts.RequestPermission()
        ) { granted ->
            if (granted) {
                Log.d("MyDebugLog", "Permission: Granted")
            } else {
                Log.d("MyDebugLog", "Permission: Denied")
            }
        }

    // уничтножение активити
    override fun onDestroy() {
        super.onDestroy()
        devicesService.removeListener(deviceListener)
    }

    // при нажатии кнопки назад эмулируется нажатие onBackPressed
    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return true
    }

    // остановка сканирования после остановки активити
    override fun onStop() {
        super.onStop()
        viewModel.stopScan()
    }

    // подписка на обновления листа найденых устройств
    private fun subscribeOnViewModel() {
        viewModel.devices.observe(this) { devices ->
            devicesService.setList(devices)
        }
    }
}
```

Листинг 30 – DialogDeviceFindFragment

```
package com.anateon.groupfit.view

import android.database.sqlite.SQLiteConstraintException
import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.core.os.bundleOf
import androidx.fragment.app.DialogFragment
import com.anateon.groupfit.R
import com.anateon.groupfit.Repositories
import com.anateon.groupfit.model.devices.entities.Device
import kotlinx.android.synthetic.main.fragment_dialog_name_change.view.*
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

// класс для диалогового окна добавления найденного устройства
class DialogDeviceFindFragment : DialogFragment() {
    var id: String? = null
    var name: String? = null
    // создание диалогового окна
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
```

```
        // получение агрументов от родительской view
        id = arguments?.getString("id")
        name = arguments?.getString("name")
        val rootView: View =
            inflater.inflate(R.layout.fragment_dialog_name_change, container, false)
        // заполнение полей полученнеми данными
        if (name != null) {
            rootView.editTextName.setText(name)
            rootView.buttonSubmitFragment.isEnabled = true
        }
        // деактивация кнопки при пустой строке
        rootView.editTextName.addTextChangedListener(object : TextWatcher {
            override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int,
after: Int) {}
            override fun onTextChanged(s: CharSequence?, start: Int, before: Int,
count: Int) {}
            override fun afterTextChanged(s: Editable?) {
                rootView.buttonSubmitFragment.isEnabled = s.toString() != ""
            }
        })
        // слушатель нажатия на кнопку
        rootView.buttonSubmitFragment.setOnClickListener {
            GlobalScope.launch {
                withContext(Dispatchers.IO) {
                    try {
                        if (id != null) {
                            Repositories.devicesRepository.addDevice(
                                Device(
                                    id!!,
                                    rootView.editTextName.text.toString()
                                )
                            )
                            withContext(Dispatchers.Main) {
                                parentFragmentManager.setFragmentResult(
                                    REQUEST_KEY,
                                    bundleOf(Pair("id", id))
                                )
                            }
                        } else
                            throw Exception("Null id")

                    } catch (e: Exception) {
                        if (e is SQLiteConstraintException) {
                            withContext(Dispatchers.Main) {
                                Toast.makeText(
                                    inflater.context,
                                    "Ошибка: Устройство с таким именем уже
существует",
                                    Toast.LENGTH_SHORT
                                ).show()
                            }
                        } else {
                            withContext(Dispatchers.Main) {
                                Toast.makeText(
                                    inflater.context,
                                    "Ошибка: ${e.message}",
                                    Toast.LENGTH_SHORT
                                ).show()
                            }
                        }
                    }
                }
            }
            dismiss()
        }
        return rootView
    }
```

```
    companion object {
        const val TAG = "DialogDeviceFind"
        const val REQUEST_KEY = "$TAG:defaultRequestKey"
    }
}
```

Листинг 31 – DialogDeviceNameChangeFragment

```
package com.anateon.groupfit.view

import android.database.sqlite.SQLiteConstraintException
import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.core.os.bundleOf
import androidx.fragment.app.DialogFragment
import com.anateon.groupfit.R
import com.anateon.groupfit.Repositories
import com.anateon.groupfit.model.devices.entities.Device
import kotlinx.android.synthetic.main.fragment_dialog_name_change.view.*
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

// класс для диалогового окна изменения имени устройства
class DialogDeviceNameChangeFragment : DialogFragment() {
    var id: String? = null
    var name: String? = null
    // создание диалогового окна
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        // получение агрументов от родительской view
        id = arguments?.getString("id")
        name = arguments?.getString("name")
        val rootView: View =
            inflater.inflate(R.layout.fragment_dialog_name_change, container, false)
        // заполнение полей полученнеми данными
        if (name != null) {
            rootView.editTextName.setText(name)
            rootView.buttonSubmitFragment.isEnabled = true
        }
        // деактивация кнопки при пустой строке
        rootView.editTextName.addTextChangedListener(object : TextWatcher {
            override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int,
after: Int) {}
            override fun onTextChanged(s: CharSequence?, start: Int, before: Int,
count: Int) {}
            override fun afterTextChanged(s: Editable?) {
                rootView.buttonSubmitFragment.isEnabled = s.toString() != ""
            }
        })
        // слушатель нажатия на кнопку
        rootView.buttonSubmitFragment.setOnClickListener {
            GlobalScope.launch {
                withContext(Dispatchers.IO) {
                    try {
                        if (id != null)
                            Repositories.devicesRepository.updateDevice(
                                Device(
                                    id!!,
```

```
                                    rootView.editTextName.text.toString()
                                )
                            )
                        else
                            throw Exception("Null id")
                        withContext(Dispatchers.Main) {
                            parentFragmentManager.setFragmentResult(
                                REQUEST_KEY,
                                bundleOf()
                            )
                        }
                    } catch (e: Exception) {
                        if (e is SQLiteConstraintException) {
                            withContext(Dispatchers.Main) {
                                Toast.makeText(
                                    inflater.context,
                                    "Ошибка: Устройство с таким именем уже
существует",
                                    Toast.LENGTH_SHORT
                                ).show()
                            }
                        } else {
                            withContext(Dispatchers.Main) {
                                Toast.makeText(
                                    inflater.context,
                                    "Ошибка: ${e.message}",
                                    Toast.LENGTH_SHORT
                                ).show()
                            }
                        }
                    }
                }
            }
            dismiss()
        }
        return rootView
    }

    companion object {
        const val TAG = "DialogDeviceNameChange"
        const val REQUEST_KEY = "$TAG:defaultRequestKey"
    }
}
```

Листинг 32 – DialogGroupNameChangeFragment

```
package com.anateon.groupfit.view

import android.database.sqlite.SQLiteConstraintException
import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.core.os.bundleOf
import androidx.fragment.app.DialogFragment
import com.anateon.groupfit.R
import com.anateon.groupfit.Repositories
import com.anateon.groupfit.model.groups.entities.Group
import kotlinx.android.synthetic.main.fragment_dialog_name_change.view.*
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

// класс для диалогового окна изменения имени группы
class DialogGroupNameChangeFragment : DialogFragment() {
```

```
    var id: Long? = null
    var name: String? = null
    // создание диалогового окна
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        // получение агрументов от родительской view
        id = arguments?.getLong("id")
        name = arguments?.getString("name")
        val rootView: View =
            inflater.inflate(R.layout.fragment_dialog_name_change, container, false)
        rootView.editTextName.hint = "КЭ-405"
        // заполнение полей полученнеми данными
        if (name != null) {
            rootView.editTextName.setText(name)
            rootView.buttonSubmitFragment.isEnabled = true
        }
        // деактивация кнопки при пустой строке
        rootView.editTextName.addTextChangedListener(object : TextWatcher {
            override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int,
after: Int) {}
            override fun onTextChanged(s: CharSequence?, start: Int, before: Int,
count: Int) {}
            override fun afterTextChanged(s: Editable?) {
                rootView.buttonSubmitFragment.isEnabled = s.toString() != ""
            }
        })
        // слушатель нажатия на кнопку
        rootView.buttonSubmitFragment.setOnClickListener {
            GlobalScope.launch {
                withContext(Dispatchers.IO) {
                    try {
                        if (id != null)
                            Repositories.groupsRepository.updateGroup(
                                Group(
                                    id!!,
                                    rootView.editTextName.text.toString()
                                )
                            )
                        else
                            Repositories.groupsRepository.addGroup(
                                Group(
                                    0,
                                    rootView.editTextName.text.toString()
                                )
                            )
                        withContext(Dispatchers.Main) {
                            parentFragmentManager.setFragmentResult(
                                REQUEST_KEY,
                                bundleOf()
                            )
                        }
                    } catch (e: Exception) {
                        if (e is SQLiteConstraintException) {
                            withContext(Dispatchers.Main) {
                                Toast.makeText(
                                    inflater.context,
                                    "Ошибка: Группа с таким именем уже существует",
                                    Toast.LENGTH_SHORT
                                ).show()
                            }
                        } else {
                            withContext(Dispatchers.Main) {
                                Toast.makeText(
                                    inflater.context,
                                    "Ошибка: ${e.message}",
```

```
                                Toast.LENGTH_SHORT
                            ).show()
                        }
                    }
                }
            }
            dismiss()
        }
        return rootView
    }

    companion object {
        const val TAG = "DialogNameChange"
        const val REQUEST_KEY = "$TAG:defaultRequestKey"
    }
}
```

Листинг 33 – GroupsActivity

```
package com.anateon.groupfit.view

import android.content.Intent
import android.os.Bundle
import android.view.Menu
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.SearchView
import androidx.recyclerview.widget.LinearLayoutManager
import com.anateon.groupfit.*
import com.anateon.groupfit.databinding.ActivityUsersBinding
import com.anateon.groupfit.model.groups.entities.Group
import com.anateon.groupfit.model.recycler.GroupsListener
import com.anateon.groupfit.model.recycler.GroupsService
import com.anateon.groupfit.view.adapter.GroupsInfoRecyclerAdapter
import com.anateon.groupfit.view.adapter.GroupsRecyclerAdapterActionListener
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

// класс, реализующий активность (экран) с группами
class GroupsActivity : AppCompatActivity(), SearchView.OnQueryTextListener {

    private lateinit var binding: ActivityUsersBinding
    private lateinit var adapter: GroupsInfoRecyclerAdapter
    private lateinit var groups: List<Group>
    private val groupListener: GroupsListener = {
        adapter.groups = it
    }

    private val ioScope
        get() = (applicationContext as App).ioScope
    private val groupsService: GroupsService
        get() = (applicationContext as App).groupsService

    // создание Activity
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityUsersBinding.inflate(layoutInflater)
        setContentView(binding.root)
        // коммуникация между активити и фрагментом. Назначение слушателя результата
        supportFragmentManager.setFragmentResultListener(
            DialogGroupNameChangeFragment.REQUEST_KEY, this
        ) { _, _ ->
            ioScope.launch {
                groups = Repositories.groupsRepository.getGroups()
                withContext(Dispatchers.Main) { groupsService.setList(groups) }
            }
```

```
        }
        // инициализация RecyclerView
        adapter = GroupsInfoRecyclerAdapter(object :
GroupsRecyclerAdapterActionListener {
            override fun onGroupEdit(group: Group) {
                val dialog = DialogGroupNameChangeFragment()
                val args = Bundle()
                args.putLong("id", group.id)
                args.putString("name", group.name)
                dialog.arguments = args
                dialog.show(supportFragmentManager, DialogGroupNameChangeFragment.TAG)
            }
            override fun onGroupItemClick(group: Group) {
                val i = Intent()
                i.putExtra("group", group.id)
                setResult(RESULT_OK, i)
                Toast.makeText(
                    this@GroupsActivity,
                    "Выбрана группа: ${group.name}",
                    Toast.LENGTH_SHORT
                ).show()
                finish()
            }
            override fun onGroupDelete(group: Group) {
                ioScope.launch {
                    Repositories.groupsRepository.deleteGroup(group)
                }
                groupsService.deleteGroup(group)
            }
        })
        binding.recyclerView.layoutManager = LinearLayoutManager(this)
        binding.recyclerView.adapter = adapter
        groupsService.addListener(groupListener)
        // получение групп из БД
        ioScope.launch {
            groups = Repositories.groupsRepository.getGroups()
            withContext(Dispatchers.Main) { groupsService.setList(groups) }
        }
        // назначение обработчика события OnClick для плавающей кнопки
        binding.floatingActionButton.setOnClickListener {
            val dialog = DialogGroupNameChangeFragment()
            dialog.show(supportFragmentManager, DialogDeviceNameChangeFragment.TAG)
        }
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.search_menu, menu)
        val search = menu?.findItem(R.id.search_action)
        val searchView = search?.actionView as? SearchView
        searchView?.isSubmitButtonEnabled = true
        searchView?.setOnQueryTextListener(this)
        return true
    }

    override fun onDestroy() {
        super.onDestroy()
        groupsService.removeListener(groupListener)
    }

    // при нажатии кнопки назад эмулируется нажатие onBackPressed
    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return true
    }
```

```
    // вызывается, когда пользователь отправляет поисковый запрос override fun
    onQueryTextSubmit(query: String?): Boolean {
        groupsService.search(query ?: "")
        return true
    }
    // после перезапуска активити обнавляется список групп
    override fun onRestart() {
        ioScope.launch {
            groups = Repositories.groupsRepository.getGroups()
            withContext(Dispatchers.Main) { groupsService.setList(groups) }
        }
        super.onRestart()
    }
    // вызывается, когда пользователь изменяет строку поиска
    override fun onQueryTextChange(newText: String?): Boolean {
        groupsService.search(newText ?: "")
        return true
    }
}
```

Листинг 34 – MainActivity

```
package com.anateon.groupfit.view
import android.bluetooth.BluetoothAdapter
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.view.Menu
import android.view.MenuItem
import android.view.View.GONE
import android.widget.Toast
import androidx.activity.result.ActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.activity.viewModels
import androidx.appcompat.app.ActionBarDrawerToggle
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.content.res.AppCompatResources
import androidx.recyclerview.widget.LinearLayoutManager
import com.anateon.groupfit.*
import com.anateon.groupfit.databinding.ActivityMainBinding
import com.anateon.groupfit.model.devices.entities.Device
import com.anateon.groupfit.model.recycler.MainInfo
import com.anateon.groupfit.model.recycler.MainInfoListener
import com.anateon.groupfit.model.recycler.MainInfoService
import com.anateon.groupfit.model.recycler.SortState
import com.anateon.groupfit.view.adapter.MainInfoRecyclerAdapter
import com.anateon.groupfit.view.adapter.MainInfoRecyclerAdapterActionListener
import com.anateon.groupfit.viewModel.MainActivityViewModel
import com.anateon.groupfit.viewModel.MainActivityViewModelFactory
import com.anateon.groupfit.viewModel.Notification
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import java.text.SimpleDateFormat
import java.util.*

// класс, реализующий активность (экран) с отслеживанием участников тренировки
class MainActivity : AppCompatActivity() {
    private var channelId = "CHANNEL_MAIN"
    private lateinit var notification: Notification
    private lateinit var binding: ActivityMainBinding
    private lateinit var adapter: MainInfoRecyclerAdapter
    private lateinit var toggle: ActionBarDrawerToggle
    private lateinit var availableDevices: MutableList<Device>
```

```
    private val mainInfoListener: MainInfoListener = {
        adapter.mainInfo = it
    }
    private var activeGroupId: Long? = null
    private var menu: Menu? = null
    private var scanIsActive: Boolean = false


    private val mainInfoService: MainInfoService
        get() = (applicationContext as App).mainInfoService

    private val ioScope
        get() = (applicationContext as App).ioScope
    private val uiScope
        get() = (applicationContext as App).uiScope

    private val viewModel: MainActivityViewModel by viewModels {
        MainActivityViewModelFactory((applicationContext as App).adapterProvider)
    }

    // создание Activity
    override fun onCreate(savedInstanceState: Bundle?) {
        // получение свободных устройств для присвоения пользователям
        ioScope.launch {
            Repositories.init(applicationContext)
            availableDevices =
Repositories.devicesRepository.getDevices().toMutableList()
            val allUsers = Repositories.usersRepository.getUsers()
            val badDevices: MutableList<Device> = mutableListOf()
            availableDevices.forEach { device ->
                if (allUsers.any { it.deviceId == device.id }) {
                    badDevices.add(device)
                }
            }
            availableDevices.removeAll(badDevices)
        }

        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        // коммуникация между активити и активити с групами (выбор активной группы).
        val activityResultLauncher =

registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result:
ActivityResult ->
                val resultNum = result.data?.getLongExtra("group", -1)
                if (resultNum != null) {
                    activeGroupId = resultNum
                    // если группа найдена, получаем список участников группы
                    if (activeGroupId != null) {
                        ioScope.launch {
                            val users =
Repositories.usersRepository.findByGroupId(activeGroupId!!)
                            withContext(Dispatchers.Main) {
                                mainInfoService.setMainInfoUsersFromUser(users)
                                binding.textViewAdvice.visibility = GONE
                            }
                        }
                    }
                }
            }
        // инициализация системы оповещений
        notification = Notification(this, channelId)
        notification.createNotificationChannel(
            "Heart rate overlimit notification",
            "Notification about exceeded heart rate"
        )
```

```
        // боковое меню
        toggle = ActionBarDrawerToggle(this, binding.drawerLayout, R.string.open,
R.string.close)
        binding.drawerLayout.addDrawerListener(toggle)
        toggle.syncState()
        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        // открытие активити при нажатии на элементы бокового меню
        binding.navView.setNavigationItemSelectedListener {
            when (it.itemId) {
                R.id.nav_users -> {
                    val intent = Intent(this, UsersActivity::class.java)
                    startActivity(intent)
                }
                R.id.nav_groups -> {
                    activityResultLauncher.launch(Intent(this,
GroupsActivity::class.java))
                }
                R.id.nav_devices -> {
                    val intent = Intent(this, DevicesActivity::class.java)
                    startActivity(intent)
                }
                R.id.nav_storage -> {
                    val intent = Intent(this, StorageActivity::class.java)
                    startActivity(intent)
                }
                R.id.nav_notify -> {
                    val intent = Intent(this, NotifyActivity::class.java)
                    startActivity(intent)
                }
            }
            true
        }

        // инициализация RecyclerView
        adapter = MainInfoRecyclerAdapter(this, object :
MainInfoRecyclerAdapterActionListener {
            override fun onMainInfoMoreData(user: MainInfo) {
                Toast.makeText(
                    this@MainActivity,
                    "Нагрузка пользователя ${user.name} ${user.lastName}: ${
                        String.format(
                            "%.1f",
                            user.heartRate.toDouble() / user.maxHeartRate.toDouble() *
100
                        )
                    }%",
                    Toast.LENGTH_SHORT
                ).show()
            }

            override fun getChargeLvl(user: MainInfo) {
                Toast.makeText(this@MainActivity, "${user.chargeLevel}%",
Toast.LENGTH_SHORT).show()
            }

            override fun getSignalLvl(user: MainInfo) {
                var infoUpdateTime = ""
                if (user.updateTime != null) {
                    infoUpdateTime =

SimpleDateFormat("mm:ss").format(user.updateTime?.time).toString()
                }
                Toast.makeText(
                    this@MainActivity,
                    "${user.signaldBm} dBm. Время обновления: $infoUpdateTime/${
                        SimpleDateFormat("mm:ss").format(GregorianCalendar().time)
                    }",
```

```
                    Toast.LENGTH_SHORT
                ).show()
            }

            override fun onButtonDeviceClick(user: MainInfo) {
                pickDevice(user)
                stopScan()
            }

            override fun onMainInfoMoreDataLong(user: MainInfo) {
                pickDevice(user)
                stopScan()
            }
        })
        val layoutManager = LinearLayoutManager(this)
        binding.recyclerView.layoutManager = layoutManager
        binding.recyclerView.adapter = adapter
        mainInfoService.addListener(mainInfoListener)
    }

    // начало сканирования
    private fun startScan() {
        // если активная группа пустая или сканер еще работает
        if (mainInfoService.getCount() == 0 || viewModel.scanWorking) {
            if (viewModel.scanWorking)
                Toast.makeText(this, "Не так быстро...", Toast.LENGTH_SHORT).show()
            else {
                Toast.makeText(this, "Выберите группу для тренировки",
Toast.LENGTH_SHORT).show()
            }
            return
        }
        scanIsActive = true
        menu?.getItem(0)?.icon = AppCompatResources.getDrawable(
            this,
            R.drawable.ic_baseline_stop_24
        )
        // выполняется в IO потоке
        ioScope.launch {
            // получение списка активных пользователей
            // и удаление пользователей без устройств
            viewModel.scanWorking = true
            val users = mainInfoService.getMainInfo().toMutableList()
            users.toMutableList().forEach {
                if (it.deviceId == null)
                    users.remove(it)
            }
            // пока сканирование активно
            while (scanIsActive) {
                users.forEach {
                    uiScope.launch { mainInfoService.setUpdateStatus(it, true) }
                    Thread.sleep(500)
                    while (viewModel.needWaitDisconnect) {
                        Log.d("MyDebugLog", "device not disconnected. Wait 0.5 sec")
                        Thread.sleep(500)
                    }
                    // подключение к устройству
                    viewModel.connect(it.deviceId!!)
                    for (i in 1..20) {
                        Log.d("MyDebugLog", "wait connection ${it.deviceId} $i")
                        Thread.sleep(500)
                        if (viewModel.isReady)
                            break
                        if (i == 20 || viewModel.errorConnection) {
                            uiScope.launch { mainInfoService.setUpdateStatus(it,
false) }
                            viewModel.disconnect()
                            return@forEach
```

```
                        }
                    }
                    var boolHR = false
                    var rssi = false
                    // чтение пульса
                    viewModel.readHeartRate { _, data ->
                        val heartRate = Integer.valueOf(data.toString().split('-')[1],
16)
                        Log.d("MyDebugLog", "HeartRate: $heartRate")
                        uiScope.launch { mainInfoService.setUserHeartRate(it,
heartRate) }
                        boolHR = true
                    }
                    // чтение уровня заряда
                    viewModel.readBattery { _, data ->
                        val battery = Integer.valueOf(data.toString().split(' ')[1],
16)
                        Log.d("MyDebugLog", "Battery: $battery")
                        uiScope.launch { mainInfoService.setUserBattery(it, battery) }
                    }
                    // чтение уровня сигнала
                    viewModel.readRssi { _, signal ->
                        rssi = true
                        Log.d("MyDebugLog", "rssi: $signal")
                        uiScope.launch { mainInfoService.setUserRssi(it, signal) }
                    }
                    // ожидание получения данных
                    for (i in 1..30) {
                        Log.d("MyDebugLog", "wait data ${it.deviceId} $i $boolHR
$rssi")
                        Thread.sleep(500)
                        if (boolHR && rssi) {
                            notification.sendNotification(it)
                            break
                        }
                        if (i == 30) {
                            uiScope.launch { mainInfoService.setUpdateStatus(it,
false) }
                            viewModel.disconnect()
                            return@forEach
                        }
                    }
                    // обновлеие UI
                    uiScope.launch {
                        mainInfoService.setUpdateStatus(
                            it,
                            false,
                            GregorianCalendar()
                        )
                    }
                    // отключение от устройства
                    viewModel.disconnect()
                    if (!scanIsActive) {
                        viewModel.scanWorking = false
                        return@launch
                    }
                }
            }
            viewModel.scanWorking = false
        }
    }

    // остановка сканирования
    fun stopScan() {
        scanIsActive = false
        menu?.getItem(0)?.icon = AppCompatResources.getDrawable(
            this,
            R.drawable.ic_baseline_play_arrow_24
```

```
    )
}

// выбор элементов меню
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    if (toggle.onOptionsItemSelected(item)) {
        return true
    }
    when (item.itemId) {
        R.id.startStopItem -> {
            if (scanIsActive) {
                stopScan()
            } else {
                startScan()
            }
            return true
        }
        R.id.headerItemName -> {
            mainInfoService.setSortMode(SortState.Name)
            return true
        }
        R.id.headerItemHeartRate -> {
            mainInfoService.setSortMode(SortState.HeartRate)
            return true
        }
        R.id.headerItemHeartRateZone -> {
            mainInfoService.setSortMode(SortState.MaxHeartRate)
            return true
        }
    }
    return super.onOptionsItemSelected(item)
}

// уничтожение активити
override fun onDestroy() {
    super.onDestroy()
    stopScan()
    mainInfoService.removeListener(mainInfoListener)
}

// создание меню
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    this.menu = menu
    menuInflater.inflate(R.menu.header_menu, menu)
    return super.onCreateOptionsMenu(menu)
}

// включение блютуз
private fun enableBluetooth() {
    requestEnableBluetooth.launch(Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE))
}

// запрос на включение блютуз
private val requestEnableBluetooth = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult()
) { result ->
    if (result.resultCode == RESULT_CANCELED) {
        showEnableBluetoothMessage()
    }
}

// отображение сообщения о том, что для работы приложения необходим Bluetooth
private fun showEnableBluetoothMessage() {
    AlertDialog.Builder(this)
        .setTitle("Включить Bluetooth")
        .setMessage("Для работы приложения необходим Bluetooth")
        .setPositiveButton("Включить") { _, _ ->
            enableBluetooth()
```

```
            }
            .setNegativeButton("Отмена") { dialog, _ ->
                dialog.dismiss()
                finish()
            }
            .create()
            .show()
    }

    // выбор временного устройства для пользователя без устрйоства
    private fun pickDevice(user: MainInfo) {
        // удаление уже назначеных устройств из списка доступных устройств
        val tmpDevice = availableDevices.toMutableList()
        mainInfoService.getMainInfo().forEach { mainInfo ->
            availableDevices.forEach {
                if (it.id == mainInfo.deviceId && it.id != user.deviceId) {
                    tmpDevice.remove(it)
                }
            }
        }
        var strings: ArrayList<String> = arrayListOf()
        strings = tmpDevice.mapTo(strings) {
            it.name
        }
        // создание AlertDialog со списком досутпных устройств
        val dialogBuilder = AlertDialog.Builder(this@MainActivity)
        dialogBuilder.setTitle("Выберите устройство")
        dialogBuilder.setIcon(R.drawable.ic_baseline_watch_24)
        dialogBuilder.setCancelable(true)
        var pickIndex = -1
        if (user.deviceId != null) {
            pickIndex = tmpDevice.indexOfFirst { it.id == user.deviceId }
        }
        dialogBuilder.setSingleChoiceItems(strings.toTypedArray(), pickIndex) { _,
which ->
            pickIndex = which
        }
        dialogBuilder.setPositiveButton("Выбрать") { _, _ ->
            mainInfoService.setDevice(user, tmpDevice[pickIndex].id)
            Log.d("MyDebugLog", pickIndex.toString())
        }
        dialogBuilder.setNegativeButton("Отмена") { _, _ -> }
        dialogBuilder.setNeutralButton("Сброс") { _, _ ->
            mainInfoService.setDevice(user, null)
        }
        dialogBuilder.create()
        val alertDialog = dialogBuilder.create()
        alertDialog.show()
    }
}
```

Листинг 35 – NotifyActivity

```
package com.anateon.groupfit.view

import android.content.Context
import android.content.SharedPreferences
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.anateon.groupfit.databinding.ActivityNotifyBinding
import com.anateon.groupfit.model.recycler.MainInfo
import com.anateon.groupfit.viewModel.Notification
import com.github.javafaker.Faker

// класс, реализующий активность (экран) управления оповещениями
class NotifyActivity : AppCompatActivity() {
    private var channelId = "CHANNEL_TEST"
```

```
    private lateinit var notification: Notification
    private lateinit var prefs: SharedPreferences
    private lateinit var binding: ActivityNotifyBinding

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

        binding = ActivityNotifyBinding.inflate(layoutInflater)
        setContentView(binding.root)
        prefs = getSharedPreferences("settings", Context.MODE_PRIVATE)
        val editor = prefs.edit()
        // инициализая системы оповещений
        notification = Notification(this, channelId)
        // настройка numberPickerNotify элемента
        binding.numberPickerNotify.minValue = 50
        binding.numberPickerNotify.maxValue = 100
        binding.numberPickerNotify.wrapSelectorWheel = false
        restorVal()
        // кнопка включения/выключения оповещений
        binding.enableSwitchNotify.setOnCheckedChangeListener { _, isChecked ->
            binding.numberPickerNotify.isEnabled = isChecked
            editor.putBoolean("globalNotify", isChecked).apply()
        }
        // слушатель изменений значения numberPickerNotify
        binding.numberPickerNotify.setOnValueChangedListener { _, _, newVal ->
            editor.putInt("procentValue", newVal).apply()
        }
        // по нажатию на элемент таблицы с нагрузкой выставляется нужное значение
        binding.textViewZone1.setOnClickListener {
            binding.numberPickerNotify.value = 60; editor.putInt("procentValue",
60).apply()
        }
        binding.textViewZone2.setOnClickListener {
            binding.numberPickerNotify.value = 70; editor.putInt("procentValue",
70).apply()
        }
        binding.textViewZone3.setOnClickListener {
            binding.numberPickerNotify.value = 80; editor.putInt("procentValue",
80).apply()
        }
        binding.textViewZone4.setOnClickListener {
            binding.numberPickerNotify.value = 90; editor.putInt("procentValue",
90).apply()
        }
        binding.textViewZone5.setOnClickListener {
            binding.numberPickerNotify.value = 100; editor.putInt("procentValue",
100).apply()
        }
        // тестовое оповещение
        binding.buttonTestNotify.setOnClickListener {
            val faker = Faker()
            notification.sendNotification(
                MainInfo(
                    userId = (1..1000).random().toLong(),
                    deviceId = "",
                    name = faker.name().firstName(),
                    lastName = faker.name().lastName(),
                    heartRate = (200..230).random(),
                    maxHeartRate = (170..200).random(),
                    signaldBm = -1,
                    chargeLevel = 100,
                    updateStatus = false,
                    updateTime = null,
                    errorStatus = false
                )
            )
        }
```

```
        // создание канала оповещений
        notification.createNotificationChannel(
            "Test notification",
            "Notifications for testing the notification system"
        )
        restorVal()
    }

    // при нажатии кнопки назад эмулируется нажатие onBackPressed
    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return true
    }

    // восстановление сохраненных значений
    fun restorVal() {
        if (prefs.contains("globalNotify")) {
            binding.enableSwitchNotify.isChecked = prefs.getBoolean("globalNotify",
false)
        }
        if (prefs.contains("procentValue")) {
            binding.numberPickerNotify.value = prefs.getInt("procentValue", 50)
        }
    }
}
```

Листинг 36 – StorageActivity

```
package com.anateon.groupfit.view

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.anateon.groupfit.App
import com.anateon.groupfit.Repositories
import com.anateon.groupfit.databinding.ActivityStorageBinding
import com.anateon.groupfit.model.recycler.UsersService
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

private lateinit var binding: ActivityStorageBinding

// класс, реализующий активность (экран) управления хранилищем
class StorageActivity : AppCompatActivity() {
    private val ioScope
        get() = (applicationContext as App).ioScope

    // создание Activity
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityStorageBinding.inflate(layoutInflater)
        setContentView(binding.root)
        // получение кол-ва пользвателей, групп и устройств
        ioScope.launch {
            val numUsers = Repositories.usersRepository.getUsers().count().toString()
            val numGroups =
Repositories.groupsRepository.getGroups().count().toString()
            val numDevices =
Repositories.devicesRepository.getDevices().count().toString()
            withContext(Dispatchers.Main) {
                binding.textViewNumberUsers.text = numUsers
                binding.textViewNumberGroups.text = numGroups
                binding.textViewNumberDevices.text = numDevices
            }
        }
        // кнопка отчистки пользователей
        binding.imageButtonDeleteUsers.setOnClickListener {
            binding.textViewNumberUsers.text = "0"
```

```
            ioScope.launch {
                Repositories.usersRepository.deleteUsers()
            }
        }
        // кнопка отчистки групп
        binding.imageButtonDeleteGroups.setOnClickListener {
            binding.textViewNumberGroups.text = "0"
            ioScope.launch {
                Repositories.groupsRepository.deleteGroups()
            }
        }
        // кнопка отчистки устройств
        binding.imageButtonDeleteDevices.setOnClickListener {
            binding.textViewNumberDevices.text = "0"
            ioScope.launch {
                Repositories.devicesRepository.deleteDevices()

            }
        }
    }

    // при нажатии кнопки назад эмулируется нажатие onBackPressed
    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return true
    }
}
```

Листинг 37 – UsersActivity

```
package com.anateon.groupfit.view

import android.content.Intent
import android.os.Bundle
import android.view.Menu
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.SearchView
import androidx.recyclerview.widget.LinearLayoutManager
import com.anateon.groupfit.*
import com.anateon.groupfit.databinding.ActivityUsersBinding
import com.anateon.groupfit.model.devices.entities.Device
import com.anateon.groupfit.model.groups.entities.Group
import com.anateon.groupfit.model.recycler.UsersListener
import com.anateon.groupfit.model.recycler.UsersService
import com.anateon.groupfit.model.users.entities.Gender
import com.anateon.groupfit.model.users.entities.User
import com.anateon.groupfit.view.adapter.UsersInfoRecyclerAdapter
import com.anateon.groupfit.view.adapter.UsersRecyclerAdapterActionListener
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import java.text.SimpleDateFormat

// класс, реализующий активность (экран) управления пользователями
class UsersActivity : AppCompatActivity(), SearchView.OnQueryTextListener {

    private lateinit var binding: ActivityUsersBinding
    private lateinit var adapter: UsersInfoRecyclerAdapter
    private val userListener: UsersListener = {
        adapter.users = it
    }

    private lateinit var users: List<User>
    private lateinit var groups: List<Group>
    private lateinit var devices: List<Device>

    private val ioScope
```

```
        get() = (applicationContext as App).ioScope
    private val usersService: UsersService
        get() = (applicationContext as App).usersService

    // создание Activity
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityUsersBinding.inflate(layoutInflater)
        setContentView(binding.root)
        // инициализация RecyclerView
        adapter = UsersInfoRecyclerAdapter(this, object :
UsersRecyclerAdapterActionListener {
            override fun onUserItemClick(user: User) {
                Toast.makeText(
                    this@UsersActivity, """
                    ${user.name} ${user.lastName}
                    ДР: ${SimpleDateFormat("dd.MM.yyyy").format(user.birthday.time)}
                    Пол: ${if (user.gender == Gender.MALE) "Мужчина" else "Женщина"}
                    Устройство: ${(devices.find { it.id == user.deviceId })?.name ?:
"-"}
                    Группа: ${(groups.find { it.id == user.groupId })?.name ?: "-"}
                """.trimIndent(), Toast.LENGTH_LONG
                ).show()
            }
            override fun onUserEdit(user: User) {
                startActivity(Intent(this@UsersActivity,
UsersEditActivity::class.java).apply {
                    putExtra("user", user)
                })
            }
            override fun onUserDelete(user: User) {
                ioScope.launch {
                    Repositories.usersRepository.deleteUser(user)
                }
                usersService.deleteUser(user)

            }
        })
        binding.recyclerView.layoutManager = LinearLayoutManager(this)
        binding.recyclerView.adapter = adapter
        usersService.addListener(userListener)
        // получение данных из БД
        ioScope.launch {
            users = Repositories.usersRepository.getUsers()
            groups = Repositories.groupsRepository.getGroups()
            devices = Repositories.devicesRepository.getDevices()
            withContext(Dispatchers.Main) { usersService.setList(users) }
        }
        // назначение обработчика события OnClick для плавающей кнопки
        binding.floatingActionButton.setOnClickListener {
            startActivity(Intent(this, UsersEditActivity::class.java))
        }

    }

    // инициализация меню
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.search_menu, menu)
        val search = menu?.findItem(R.id.search_action)
        val searchView = search?.actionView as? SearchView
        searchView?.isSubmitButtonEnabled = true
        searchView?.setOnQueryTextListener(this)
        return true
    }

    // уничтножение активити
    override fun onDestroy() {
        super.onDestroy()
```

```
        usersService.removeListener(userListener)
    }

    // при нажатии кнопки назад эмулируется нажатие onBackPressed
    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return true
    }

    // вызывается, когда пользователь отправляет поисковый запрос
    override fun onQueryTextSubmit(query: String?): Boolean {
        usersService.search(query ?: "")
        return true
    }

    // после перезапуска активити обнавляется список устройств
    override fun onRestart() {
        ioScope.launch {
            users = Repositories.usersRepository.getUsers()
            withContext(Dispatchers.Main) { usersService.setList(users) }
        }
        super.onRestart()
    }

    // вызывается, когда пользователь изменяет строку поиска
    override fun onQueryTextChange(newText: String?): Boolean {
        usersService.search(newText ?: "")
        return true
    }
}
```

Листинг 38 – UsersEditActivity

```
package com.anateon.groupfit.view

import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import android.widget.ArrayAdapter
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.anateon.groupfit.App
import com.anateon.groupfit.R
import com.anateon.groupfit.Repositories
import com.anateon.groupfit.databinding.ActivityUsersEditBinding
import com.anateon.groupfit.model.devices.entities.Device
import com.anateon.groupfit.model.groups.entities.Group
import com.anateon.groupfit.model.recycler.UsersService
import com.anateon.groupfit.model.users.entities.Gender
import com.anateon.groupfit.model.users.entities.User
import kotlinx.android.synthetic.main.activity_users_edit.*
import kotlinx.android.synthetic.main.fragment_dialog_name_change.editTextName
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import java.text.SimpleDateFormat
import java.util.*

// класс, реализующий активность (экран) редактирования/добавления пользователей
class UsersEditActivity : AppCompatActivity() {
    private lateinit var binding: ActivityUsersEditBinding

    private val ioScope
        get() = (applicationContext as App).ioScope
    private val usersService: UsersService
        get() = (applicationContext as App).usersService

    private lateinit var users: List<User>
```

```
    private lateinit var groups: List<Group>
    private lateinit var devices: MutableList<Device>

    private var user: User? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        // получение данных из БД
        ioScope.launch {
            users = Repositories.usersRepository.getUsers()
            groups = Repositories.groupsRepository.getGroups()
            devices = Repositories.devicesRepository.getDevices().toMutableList()
            withContext(Dispatchers.Main) { usersService.setList(users) }
        }
        super.onCreate(savedInstanceState)
        binding = ActivityUsersEditBinding.inflate(layoutInflater)
        setContentView(binding.root)
        // получение данных материнской активити
        try {
            user = intent.getSerializableExtra("user") as? User
        } catch (e: Exception) {
        }
        // список групп для выбора
        val groupsList: MutableList<String> = groups.map { it.name }.toMutableList()
        groupsList.add(0, "")
        val adapterGroup = ArrayAdapter(this, android.R.layout.simple_spinner_item,
groupsList)

adapterGroup.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        binding.spinnerGroup.adapter = adapterGroup

        // список устройств для выбора
        val devicesList: MutableList<String> = devices.map { it.name }.toMutableList()
        devicesList.add(0, "")
        val adapterDevice = ArrayAdapter(this, android.R.layout.simple_spinner_item,
devicesList)

adapterDevice.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        binding.spinnerDevice.adapter = adapterDevice

        // заполнить пользовательские данные, если активити была создана для изменения
данных пользоватлеля
        if (user != null) {
            binding.editTextName.setText(user?.name)
            binding.editTextLastName.setText(user?.lastName)
            if (user?.gender == Gender.MALE) {
                binding.radioButtonMale.isChecked = true
            } else {
                binding.radioButtonFemale.isChecked = true
            }

            binding.editTextDate.setText(
                SimpleDateFormat("dd.MM.yyyy").format(user?.birthday?.time).toString()
            )
            if (user?.groupId != null)
                binding.spinnerGroup.setSelection(groups.indexOf(groups.find { it.id
== user?.groupId }) + 1)
            if (user?.deviceId != null)
                binding.spinnerDevice.setSelection(devices.indexOf(devices.find {
it.id == user?.deviceId }) + 1)
        }

        // кнопка подверждения изменения
        binding.buttonSubmit.setOnClickListener {
            try {
                // парсинг даты
                val dateString =
                    binding.editTextDate.text.toString().split(Regex("[^0-9]")).map {
it.toInt() }
```

```
                var dbError: Boolean? = null
                // проверка на ошибки входных данных
                if (dateString.count() != 3) {
                    throw Exception("Неверный формат даты")
                }
                if (editTextName.text.toString() == "") {
                    throw Exception("Пустое имя")
                }
                if (editTextLastName.text.toString() == "") {
                    throw Exception("Пустая фамилия")
                }
                // создание пользователя
                val newUser = User(
                    id = user?.id ?: 0,
                    name = binding.editTextName.text.toString(),
                    lastName = binding.editTextLastName.text.toString(),
                    gender = if (binding.radioButtonMale.isChecked) Gender.MALE
                    else if (binding.radioButtonFemale.isChecked) Gender.FEMALE
                    else throw Exception("Не выбран пол"),
                    birthday = GregorianCalendar(dateString[2], dateString[1] - 1,
dateString[0]),
                    groupId = if (binding.spinnerGroup.selectedItemPosition == 0) null
else groups[binding.spinnerGroup.selectedItemPosition - 1].id,
                    deviceId = if (binding.spinnerDevice.selectedItemPosition == 0)
null else devices[binding.spinnerDevice.selectedItemPosition - 1].id
                )

                // ошибка неверной даты
                if ((newUser.birthday < GregorianCalendar(
                        1900,
                        0,
                        1
                    )) || (newUser.birthday > GregorianCalendar())
                )
                    throw Exception("Неверная дата")
                ioScope.launch {
                    try {
                        if (user != null)
                            // обновить пользователя
                            Repositories.usersRepository.updateUserInfo(newUser)
                        else
                            // создать пользователя
                            Repositories.usersRepository.createUser(newUser)
                        dbError = false
                        withContext(Dispatchers.Main) {
                            finish()
                        }
                    } catch (e: Exception) {
                        dbError = true
                    }
                }
                // ожидание записи в бд
                while (dbError == null) {
                    Thread.sleep(10)
                }
                // если запись с ошибкой
                if (dbError == true) {
                    Toast.makeText(this, "Ошбика БД", Toast.LENGTH_SHORT).show()
                }
                dbError = null
            } catch (e: Exception) {
                // вывод toast уведомления в случае ошибки
                Toast.makeText(this, "Ошибка: ${e.message}",
Toast.LENGTH_SHORT).show()
            }
        }
    }
```

```
    // инициализация меню
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.save_menu, menu)
        return super.onCreateOptionsMenu(menu)
    }

    // выбор элементов меню
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.save_action -> binding.buttonSubmit.callOnClick()
        }
        return super.onOptionsItemSelected(item)
    }

    // при нажатии кнопки назад эмулируется нажатие onBackPressed
    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return true
    }
}
```

Листинг 39 – BleControlManager

```
package com.anateon.groupfit.viewModel

import android.bluetooth.BluetoothGatt
import android.bluetooth.BluetoothGattCharacteristic
import android.content.Context
import no.nordicsemi.android.ble.BleManager
import no.nordicsemi.android.ble.callback.DataReceivedCallback
import no.nordicsemi.android.ble.callback.RssiCallback
import java.util.*

// класс для взаимодействия с Bluetooth LE устройствами
class BleControlManager(context: Context) : BleManager(context) {
    private var controlHeartRate: BluetoothGattCharacteristic? = null
    private var controlBattery: BluetoothGattCharacteristic? = null
    override fun getGattCallback(): BleManagerGattCallback =
BleControlManagerGattCallback()

    // чтение уровня заряда
    fun readBattery(dataReceivedCallback: DataReceivedCallback) {
        readCharacteristic(controlBattery)
            .with(dataReceivedCallback)
            .enqueue()
    }

    // чтение ЧСС
    fun readHeartRate(dataReceivedCallback: DataReceivedCallback) {
        setNotificationCallback(controlHeartRate)
            .with(dataReceivedCallback)
        enableNotifications(controlHeartRate)
            .enqueue()
    }

    // чтения уровня сигнала
    fun readRssi(rssiCallback: RssiCallback) {
        readRssi().with(rssiCallback).enqueue()
    }

    // Callback считывающий значения ЧСС и уровня заряда
    private inner class BleControlManagerGattCallback : BleManagerGattCallback() {
        override fun isRequiredServiceSupported(gatt: BluetoothGatt): Boolean {
            val heartRateService = gatt.getService(SERVICE_HEART_RATE)
            val batteryService = gatt.getService(SERVICE_BATTERY)
            if (heartRateService != null) {
                controlHeartRate =
```

```
heartRateService.getCharacteristic(CHARACTER_HEART_RATE_MEASUREMENT)
            }
            if (batteryService != null)
                controlBattery =
batteryService.getCharacteristic(CHARACTER_BATTERY_LEVEL)
            return controlHeartRate != null
        }

        override fun onServicesInvalidated() {
            controlHeartRate = null
            controlBattery = null
        }
    }

    companion object {
        val SERVICE_HEART_RATE = UUID.fromString("0000180d-0000-1000-8000-
00805f9b34fb")
        val CHARACTER_HEART_RATE_MEASUREMENT =
            UUID.fromString("00002a37-0000-1000-8000-00805f9b34fb")
        val SERVICE_BATTERY = UUID.fromString("0000180f-0000-1000-8000-00805f9b34fb")
        val CHARACTER_BATTERY_LEVEL = UUID.fromString("00002a19-0000-1000-8000-
00805f9b34fb")
    }
}
```

Листинг 40 – BluetoothAdapterProvider

```
package com.anateon.groupfit.viewModel

import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothManager
import android.content.Context

// интерфейс для связи различных Bluetooth адаптеров
interface BluetoothAdapterProvider {

    fun getAdapter(): BluetoothAdapter

    fun getContext(): Context

    class Base(private val context: Context) : BluetoothAdapterProvider {
        override fun getAdapter(): BluetoothAdapter {
            val manager = context.getSystemService(Context.BLUETOOTH_SERVICE) as
BluetoothManager
            return manager.adapter
        }
        override fun getContext(): Context {
            return context
        }
    }
}
```

Листинг 41 – DevicesFinderViewModel

```
package com.anateon.groupfit.viewModel

import android.annotation.SuppressLint
import android.bluetooth.le.*
import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider

// класс для поиска Bluetooth LE устройств
class DevicesFinderViewModel(adapterProvider: BluetoothAdapterProvider) : ViewModel()
{
```

```
var pairedDevices = mutableListOf<String>()
private val _devices: MutableLiveData<List<ScanResult>> = MutableLiveData()
val devices: LiveData<List<ScanResult>> get() = _devices

private val adapter = adapterProvider.getAdapter()

private var scanner: BluetoothLeScanner? = null
private var callback: BleScanCallback? = null


private val settings: ScanSettings
private val filters: List<ScanFilter>

private val foundDevices = HashMap<String, ScanResult>()

init {
    settings = buildSettings()
    filters = buildFilter()
}

// настройка сканера
private fun buildSettings() =
    ScanSettings.Builder()
        .setScanMode(ScanSettings.SCAN_MODE_LOW_POWER)
        .build()

// настройка фильтров сканирования
private fun buildFilter() =
    listOf(
        ScanFilter.Builder()
            .build()
    )

// старт скариновария
@SuppressLint("MissingPermission")
fun startScan() {
    if (callback == null) {
        callback = BleScanCallback()
        scanner = adapter.bluetoothLeScanner
        scanner?.startScan(filters, settings, callback)
    }
}

// остановка сканирования
@SuppressLint("MissingPermission")
fun stopScan() {
    if (callback != null) {
        scanner?.stopScan(callback)
        scanner = null
        callback = null
    }
}

// вызывается по завершению ViewModel
override fun onCleared() {
    super.onCleared()
    stopScan()
}

// callback для добавления найденных устройв в коллекцию найденых устройств
inner class BleScanCallback : ScanCallback() {
    @SuppressLint("MissingPermission")
    override fun onScanResult(callbackType: Int, result: ScanResult) {
        if (!pairedDevices.contains(result.device.address)) {
            foundDevices[result.device.address] = result
            _devices.postValue(foundDevices.values.toList())
        }
    }
```

```
        override fun onBatchScanResults(results: MutableList<ScanResult>) {
            for (result in results) {
                if (!pairedDevices.contains(result.device.address))
                    foundDevices[result.device.address] = result
            }
        }

        override fun onScanFailed(errorCode: Int) {
            Log.d("MyDebugLog", "onScanFailed: $errorCode")
        }
    }
}

// фабрика для создания View model
class DeviceViewModelFactory(
    private val adapterProvider: BluetoothAdapterProvider
) : ViewModelProvider.Factory {
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(DevicesFinderViewModel::class.java)) {
            return DevicesFinderViewModel(adapterProvider) as T
        }
        throw IllegalArgumentException("View model not found")
    }
}
```

Листинг 42 – MainActivityViewModel

```
package com.anateon.groupfit.viewModel

import android.bluetooth.BluetoothDevice
import android.util.Log
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import no.nordicsemi.android.ble.callback.DataReceivedCallback
import no.nordicsemi.android.ble.callback.RssiCallback
import no.nordicsemi.android.ble.observer.ConnectionObserver

// класс для подключения к устройствам участников активной группы
class MainActivityViewModel(private val adapterProvider: BluetoothAdapterProvider) :
ViewModel() {
    private val controlManager = BleControlManager(adapterProvider.getContext())
    val isReady: Boolean
        get() {
            return controlManager.isReady
        }
    var scanWorking = false
    var errorConnection = false
    var needWaitDisconnect = false

    // подключение к устройству
    fun connect(deviceAddress: String) {
        val device = adapterProvider.getAdapter().getRemoteDevice(deviceAddress)
        errorConnection = false
        controlManager.connect(device)
            .retry(7, 550)
            .timeout(7600)
            .useAutoConnect(true)
            .done {
                Log.d("MyDebugLog", "connection success")
            }
            .fail { _, status ->
                Log.d("MyDebugLog", "connection bad, $status")
                errorConnection = true
            }
            .enqueue()
        controlManager.connectionObserver = connectionObserver
    }
```

```
    // отключение от устройства
    fun disconnect() {
        controlManager.disconnect().enqueue()
    }

    // чтение уровня заряда
    fun readBattery(dataReceivedCallback: DataReceivedCallback) {
        if (controlManager.isReady) {
            return controlManager.readBattery(dataReceivedCallback)
        }
    }

    // чтение ЧСС
    fun readHeartRate(dataReceivedCallback: DataReceivedCallback) {
        if (controlManager.isReady) {
            return controlManager.readHeartRate(dataReceivedCallback)
        }
    }

    // чтение уровня сигнала
    fun readRssi(rssiCallback: RssiCallback) {
        if (controlManager.isReady) {
            return controlManager.readRssi(rssiCallback)
        }
    }

    // события происходящие на различных этапах подключения/отключения и при ошибках
    private val connectionObserver = object : ConnectionObserver {
        override fun onDeviceConnecting(device: BluetoothDevice) {
            Log.d("MyDebugLog", "onDeviceConnecting")
        }

        override fun onDeviceConnected(device: BluetoothDevice) {
            Log.d("MyDebugLog", "onDeviceConnected")
        }

        override fun onDeviceFailedToConnect(device: BluetoothDevice, reason: Int) {
            Log.d("MyDebugLog", "onDeviceFailedToConnect")
        }

        override fun onDeviceReady(device: BluetoothDevice) {
            Log.d("MyDebugLog", "onDeviceReady")
        }

        override fun onDeviceDisconnecting(device: BluetoothDevice) {
            needWaitDisconnect = true
            Log.d("MyDebugLog", "onDeviceDisconnecting")
        }

        override fun onDeviceDisconnected(device: BluetoothDevice, reason: Int) {
            needWaitDisconnect = false
            Log.d("MyDebugLog", "onDeviceDisconnected")
        }
    }
}

// фабрика для создания View model
class MainActivityViewModelFactory(
    private val adapterProvider: BluetoothAdapterProvider
) : ViewModelProvider.Factory {
    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(MainActivityViewModel::class.java)) {
            return MainActivityViewModel(adapterProvider) as T
        }
        throw IllegalArgumentException("View model not found")
    }
}
```

Листинг 43 – Notification

```
package com.anateon.groupfit.viewModel

import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.Context
import android.media.RingtoneManager
import android.os.Build
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import com.anateon.groupfit.R
import com.anateon.groupfit.model.recycler.MainInfo

// класс, реализующий систему оповещений
class Notification(val context: Context, val CHANNEL_ID: String) {
    // создание канала оповещений
    fun createNotificationChannel(name: String, descriptionText: String) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val importance = NotificationManager.IMPORTANCE_HIGH
            val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
                description = descriptionText
                enableVibration(true)
            }
            val notificationManager: NotificationManager =
                context.getSystemService(AppCompatActivity.NOTIFICATION_SERVICE) as
NotificationManager
            notificationManager.createNotificationChannel(channel)
        }
    }

    // отправить оповещение
    fun sendNotification(mainInfo: MainInfo) {
        // получение настроек оповещения
        val set = context.getSharedPreferences("settings", Context.MODE_PRIVATE)
        val globalNotify: Boolean = set.getBoolean("globalNotify", false)
        val procentValue: Int = set.getInt("procentValue", 50)
        // если усовия оповещения выполняются
        if (globalNotify && mainInfo.heartRate / mainInfo.maxHeartRate.toDouble() >=
procentValue.toDouble() / 100) {
            val builder = NotificationCompat.Builder(context, CHANNEL_ID)
                .setSmallIcon(R.drawable.ic_heart_pulse)
                .setContentTitle("Перегрузка >$procentValue%")
                .setContentText("ЧСС пользователя ${mainInfo.name}
${mainInfo.lastName}: ${mainInfo.heartRate} уд/мин ${(mainInfo.heartRate /
(mainInfo.maxHeartRate.toDouble()) * 100).toInt()}%")
                .setPriority(NotificationCompat.PRIORITY_MIN)

.setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION))
                .setVibrate(longArrayOf(1000, 1000, 1000))

                .setAutoCancel(true)
            with(NotificationManagerCompat.from(context)) {
                notify(mainInfo.userId.toInt(), builder.build())
            }
        }
    }
}
```

Листинг 44 – App

```
package com.anateon.groupfit

import android.app.Application
import com.anateon.groupfit.model.recycler.*
import com.anateon.groupfit.viewModel.BluetoothAdapterProvider
```

```
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers

// класс для реализации синглтон классов
class App : Application() {
    val adapterProvider: BluetoothAdapterProvider by lazy {
        BluetoothAdapterProvider.Base(applicationContext)
    }
    val mainInfoService: MainInfoService =
        MainInfoService()
    val usersService: UsersService =
        UsersService()
    val groupsService: GroupsService =
        GroupsService()
    val devicesService: DevicesService =
        DevicesService()
    val devicesFindService: DevicesFindService =
        DevicesFindService()
    val uiScope = CoroutineScope(Dispatchers.Main)
    val ioScope = CoroutineScope(Dispatchers.IO)
}
```

Листинг 45 – Repositories

```
package com.anateon.groupfit

import android.app.Application
import com.anateon.groupfit.model.recycler.*
import com.anateon.groupfit.viewModel.BluetoothAdapterProvider
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers

// класс для реализации синглтон классов
class App : Application() {
    val adapterProvider: BluetoothAdapterProvider by lazy {
        BluetoothAdapterProvider.Base(applicationContext)
    }
    val mainInfoService: MainInfoService =
        MainInfoService()
    val usersService: UsersService =
        UsersService()
    val groupsService: GroupsService =
        GroupsService()
    val devicesService: DevicesService =
        DevicesService()
    val devicesFindService: DevicesFindService =
        DevicesFindService()
    val uiScope = CoroutineScope(Dispatchers.Main)
    val ioScope = CoroutineScope(Dispatchers.IO)
}
```