МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения»

Специализация: «Эксплуатация вычислительных машин, комплексов, систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

Приложение обработки телеметрической информации

Студент _____ Милованов Дмитрий Вячеславович
(подпись, дата) (ФИО полностью)

Руководитель _____ Федюнин Р.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Кучин А.В.
(подпись, дата) (фамилия, инициалы)

Рецензент доцент, ПГУ _____ Печерский С.В.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 10.06.2020 г. № 13)

Заведующий кафедрой _____ Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой удовлетворительно (протокол заседания ГЭК от 19.06.2020 № 14)

Секретарь ГЭК _____ Гурин Е.И.

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	Вычислительная техника

«Утверждаю»
Заведующий кафедрой
_____ Митрохин М.А.
«30» апреля 2020г.

ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ

1. Милованов Дмитрий Вячеславович гр. 15ВО2 факультета вычислительной техники.

2.Тема работы: «Приложение обработки телеметрической информации».

Тема утверждена приказом ПГУ № 0335а/о от 30.04.2020 г.

3. Руководитель работы доцент каф. ВТ ПГУ Федюнин Р.М.

4. Задание на работу (назначение разработки, исходные данные и т.п.).

Приложение предназначено для обмена необходимой информацией в целях диагностики, лечения и профилактики заболеваний и травм, проведения исследований и оценок.

5. Перечень подлежащих разработке вопросов.

В процессе работы определен список требований программного обеспечения, разработано приложение для Android, которое работает с носимым датчиком, передающим полученные данные клиенту и серверу. Также выполнить задание по БЖД и экономике.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области		12.02.2020-21.02.2020	
2.	Выбор стека технологий и компонентов для реализации приложения		21.02.2020-29.02.2020	
3.	Составление UML схемы приложения		01.03.2020-21.03.2020	
4.	Написание программы		22.03.2020-31.03.2020	
5.	Отладка системы		01.04.2020-11.04.2020	
6.	Моделирование и поиск уязвимостей, исправление ошибок в работе системы		12.04.2020-22.04.2020	
7.	Оформление выпускной работы		25.04.2020-09.06.2020	

Дата выдачи задания 30 «апреля» 2020 года.

Руководитель дипломного проекта _____ Федюнин Р.Н.
(подпись, дата)

Задание к исполнению принял студент _____ Милованов Д.В.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Работа содержит 110 листов, 39 рисунков, 4 таблицы, 8 источников, 3 приложения.

ТЕЛЕМЕТРИЯ, ОС ANDROID, BLUETOOTH, ECLIPSE, БАЗА ДАННЫХ.

Цель работы – приложение, которое обеспечивает обработку телеметрических данных полученных с оконечного устройства.

Объектом разработки является разработка приложения для Android, которое позволяет работать с носимым датчиком, передающим полученные данные клиенту и серверу.

В результате проведенной работы должен быть разработан проект, удовлетворяющий поставленной задаче, а именно, создан программный комплекс и разработаны соответствующие алгоритмы, реализующие возможность хранить, обрабатывать и доставлять до потребителя информацию о запланированных мероприятиях в виде сообщений.

Данное приложение рекомендуется использовать для обмена необходимой информацией в целях диагностики, лечения и профилактики заболеваний и травм, проведения исследований и оценок.

Содержание

Введение…… 7

1 Исследование……..……...……. 9

1.1 Анализ предметной области……... 9

1.1.1 Стандарты персональных медицинских данных (PHD) ISO / IEEE 11073…….. 10

1.1.2 Описание существующих решений мобильной медицины…. 14

1.2 Выбор мобильной операционной системы……... 16

1.3 Обзор устройств телеметрии…….. 23

2 Реализация приложения……. 27

2.1 Режим работы с конечным устройством……... 29

2.2 Режим работы с основным сервером……. 44

2.3 Интерфейсы приложения…… 50

3 Технология разработки приложений ОС Android в IDE Eclipse……... 57

3.1 Установка среды разработки Eclipse……. 57

3.2 Установка Android SDK…….. 59

3.3 Реализация минимального проекта…… 60

4 Технико-экономическое обоснование разработки программного обеспечения дипломного проекта…… 65

4.1 Организация создания программного обеспечения……. 65

4.2 Расчет цены программного обеспечения…….. 65

4.2.1 Определение затрат на разработку программного обеспечения…… 66

4.2.2 Установление цены программного обеспечения…….. 70

4.2.3 Установление нижнего предела цены с учетом тиражирования……..... 71

5 Безопасность жизнедеятельности……. 72

5.1 Организация безопасных условий труда на рабочем месте

оператора ПЭВМ……... 72
5.2 Расчет освещения на рабочем месте…….. 73
5.3 Параметры микроклимата……... 76
5.4 Шум и вибрация……... 78
Заключение……. 81
Список использованных источников……... 82
ПРИЛОЖЕНИЕ А…….. 83
ПРИЛОЖЕНИЕ Б…….. 102
ПРИЛОЖЕНИЕ В…….. 107

Введение

В течение последнего десятилетия произошло активное проникновение в повседневную практику технологий удаленного мониторинга (УМ) жизненно важных показателей, которые позволяют увеличить доступность и качество медицинской помощи.

Так, в США, по данным исследования за 2014–2019 г.г., отмечался более чем двукратный рост рынка систем УМ.

В странах Европы в 2019 году величина совокупных расходов на удаленный мониторинг составила 233 миллиарда евро, что превысило данные 2018 года на 50% [2,3].

Применению систем УМ способствует ряд факторов, среди которых следует отметить старение населения и, как следствие, увеличение распространенности хронических заболеваний в ряде стран, наряду с прогрессирующим ростом стоимости лечебно-диагностических мероприятий.

Использование УМ в условиях повседневной практики способствует реализации индивидуальной модели здравоохранения, которая ориентирована на пациента и обеспечивает раннее выявление хронических заболеваний [4,5].

УМ играет важную роль как элемент комплексного контроля состояния пожилых пациентов. Вне зависимости от возраста больных, применение УМ позволяет оптимизировать ведение пациентов с хроническими заболеваниями за счет уменьшения потребности в количестве госпитализаций, которые сопряжены со значительной финансовой нагрузкой на систему здравоохранения. Несмотря на потенциальные возможности УМ, требуется дальнейшее изучение данной тематики с инженерной и софтверной точек зрения. В данной работе речь идет о технологиях удаленного мониторинга/телемедицины пациентов.

Телемедицина – предоставление услуг здравоохранения в условиях, когда расстояние является критическим фактором, для чего используются информационно-коммуникационные технологии для обмена необходимой информацией в целях диагностики, лечения и профилактики заболеваний и травм, проведения исследований и оценок.

Цель дипломного проекта - разработка приложения для Android, которое работает с носимым датчиком, передающим полученные данные клиенту и серверу.

Получение и анализ предоставленных данных в режиме реального времени производятся в зоне действия беспроводной сети на основе Bluetooth или Wi-Fi сети.

Система способна хранить записанные данные и размещать их на удаленном сервере, что обеспечивает возможность анализа данных на любом этапе ведения пациентов.

1 Исследование

В данной главе будут рассмотрены архитектурные решения теле мониторинга, стандарты работы систем теле мониторинга, а также стек технологий разработки приложений, работающих совместно с системами теле мониторинга.

1.1 Анализ предметной области

В данной главе будут рассмотрены архитектурные решения телемониторинга, стандарты работы систем телемониторинга, а также стек технологий разработки приложений работающих совместно с системами телемониторинга.

Существуют случаи, когда пациентов следует контролировать в режиме реального времени. Пример, если пациент страдает хроническим заболеванием, идеальным вариантом будет его непрерывный контроль в режиме 24/7/365 (что не всегда возможно).

Телемониторинг является решением этой проблемы. Система телемониторинга в основном состоит из медицинских устройств, необходимых для снятия параметров с пациента, шлюза, действующего как посредник между устройствами и сервером данных.

Шлюзом является ПК, КПК, телефон или любое устройство, способное выполнять функцию связного между устройствами пациента и сервером.

В этой системе обязательно, чтобы все устройства были оснащены возможностью беспроводной связи с должным уровнем безопасности информации.

Одной из целей персональной системы телемониторинга является обмен данными между устройствами, образующими сеть, по принципу «включай и работай», чтобы эти устройства могли подключать, настраивать и передавать свои данные без вмешательства пациента.

1.1.1 Стандарты персональных медицинских данных (PHD) ISO / IEEE 11073

Во всех системах передачи данных отправка, обработка и получение сообщений происходит в рамках стандартов. То же самое происходит в среде пользовательских систем дистанционного мониторинга.

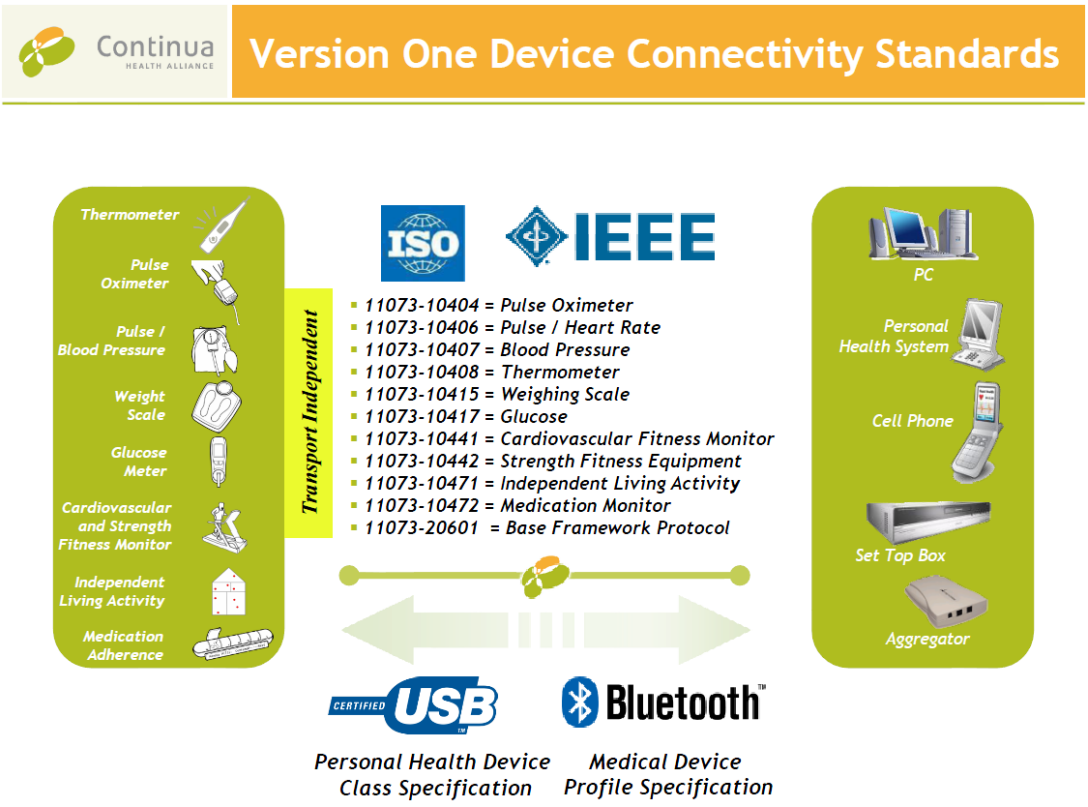


Рисунок 1.1 - Серия стандартов передачи данных телеметрии

Как правило, медицинские приборы разных производителей существуют и работают со своими собственными форматами обработки данных. Они используют разные способы записи и передачи данных с

использованием разных протоколов. Это показывает большую потребность в стандартизации этой области.

Стандартизация в области системы пользовательского теле мониторинга позволяет снижать затраты благодаря возможности выбора продуктов разных компаний. Благодаря чему система теле мониторинга становится масштабируемой и гибкой.

Функциональная совместимость аппаратуры и программных продуктов позволяет автоматически определять устройства при подключении, настраивать их и использовать без вмешательства пользователя.

Все эти причины привели к созданию стандарта взаимодействия медицинских устройств.

Текущий стандарт связи с медицинскими устройствами - ISO/IEEE 11073. Это семейство стандартов охватывает семь уровней стека протоколов OSI и обеспечивает универсальность для преобразования информации в формат, который может передавать данные между персональными устройствами и сервером. Данные могут храниться на сервере и/или быть отправлены в хранилище (например, облако) для контроля или хранения.

Стандарты связи между медицинскими устройствами обеспечивают связь между устройствами и внешними компьютерными системами. Они обеспечивают автоматический сбор информации о клиентских данных и показателях жизнедеятельности, а также рабочих данных устройства.

Стандарты IEEE 11073 PHD основаны на понятиях «агенты» и «менеджеры».

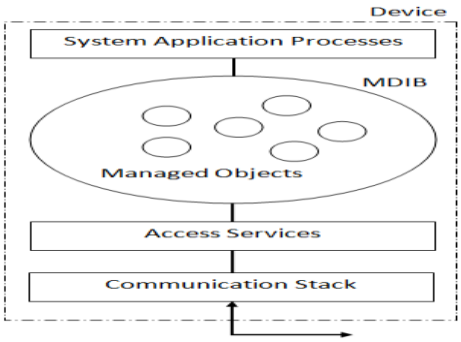


Рисунок 1.2 - Модель устройства телеметрии

Агенты - это персональные устройства, и, как правило, это небольшие недорогие устройства с батарейным питанием, которым не хватает дисплеев и других пользовательских интерфейсов.

Менеджеры - это, как правило, небольшие компьютеры или смартфоны с большими вычислительными ресурсами и необходимыми возможностями маршрутизации для автономной передачи информации от источника к указанному классу приемников.

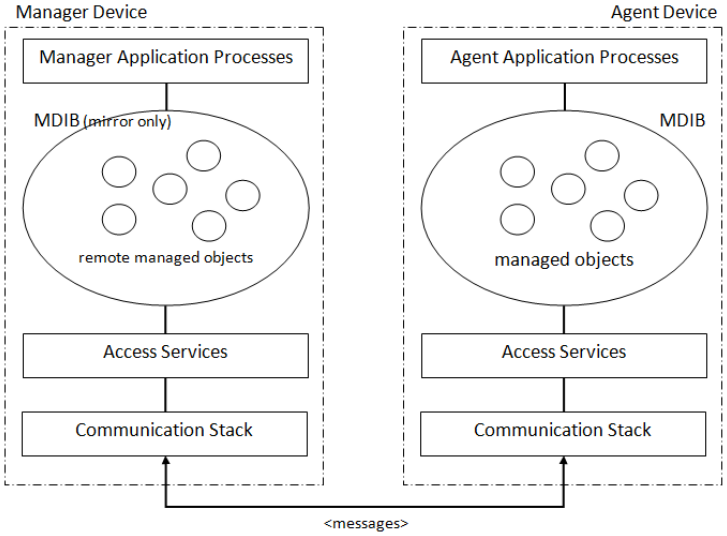


Рисунок 1.3 - Модель передачи данных между устройствами

Все коммуникации между агентами и менеджерами предпочтительно являются мобильными, так как пациент сам является мобильным субъектом.

Когда агенты передают свои данные менеджерам, они могут обрабатываться и отображаться на менеджерах, а затем, возможно, передаваться через сеть третьим лицам.

Стандарты предполагают, что каждый агент одновременно общается с одним менеджером. Менеджер может общаться с более чем одним агентом.

Оперируя понятиями и определениями ISO / IEEE 11073, отметим Health Level 7 –формат данных передачи, управления и интеграции электронной медицинской информации между различными устройствами. HL7 является надстройкой над **ISO / IEEE 11073**). В данной работе он будет использован в работе в качестве библиотеки SDK производителя

оборудования телемедицины для унификации пакетов отправки сообщений между устройствами.

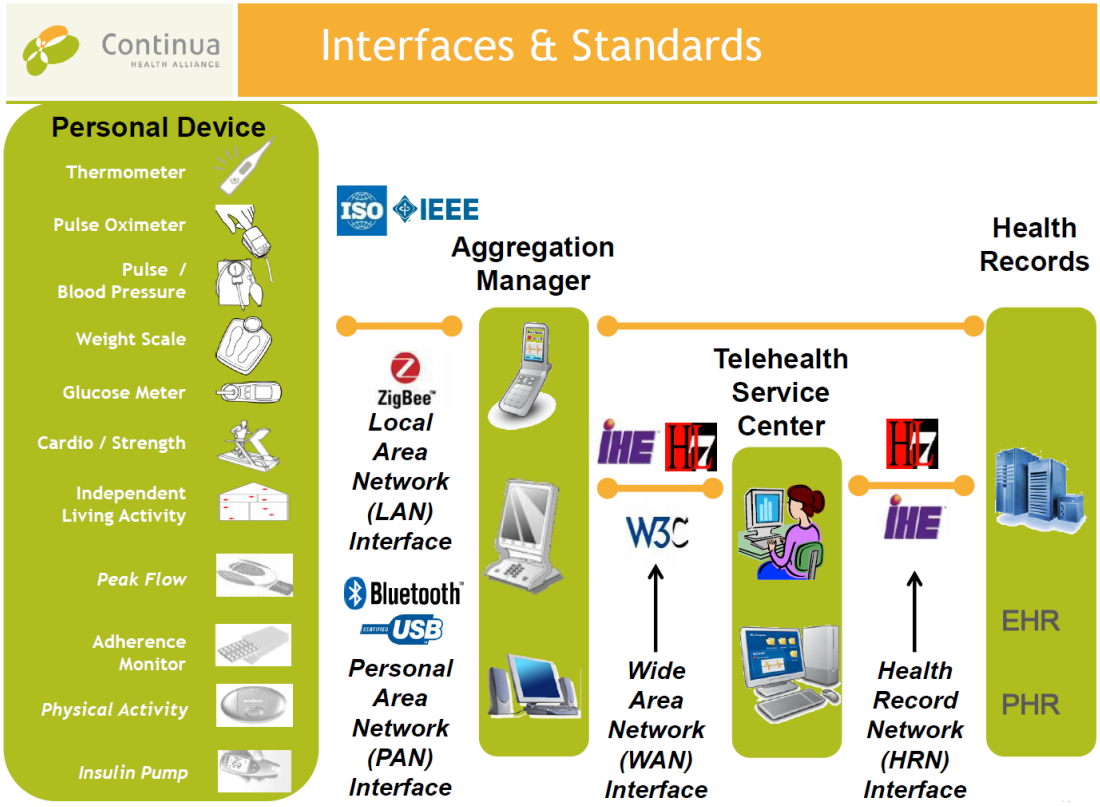


Рисунок 1.4 - Связь ISO / IEEE 11073 и HL7.

По аналогии с высшим уровнем коммуникационной модели открытых систем (OSI) HL7 поддерживает выполнение таких задач как:

- структурирование передаваемых данных;
- возможности проектирования систем;
- достижение согласованности передач;
- безопасность;
- идентификация участников;
- доступность.

Таким образом, делаем вывод:

С помощью ISO / IEEE 11073 определяется физическая среда передачи данных между устройствами: Bluetooth Device Profile, Wi-Fi Device Class, ZigBee Profile.

С помощью HL7 определяются форматы пакетов и параметры пакетов данных прикладного уровня модели OSI (уровня приложения) для удобства программирования и обработки сообщений между устройствами.

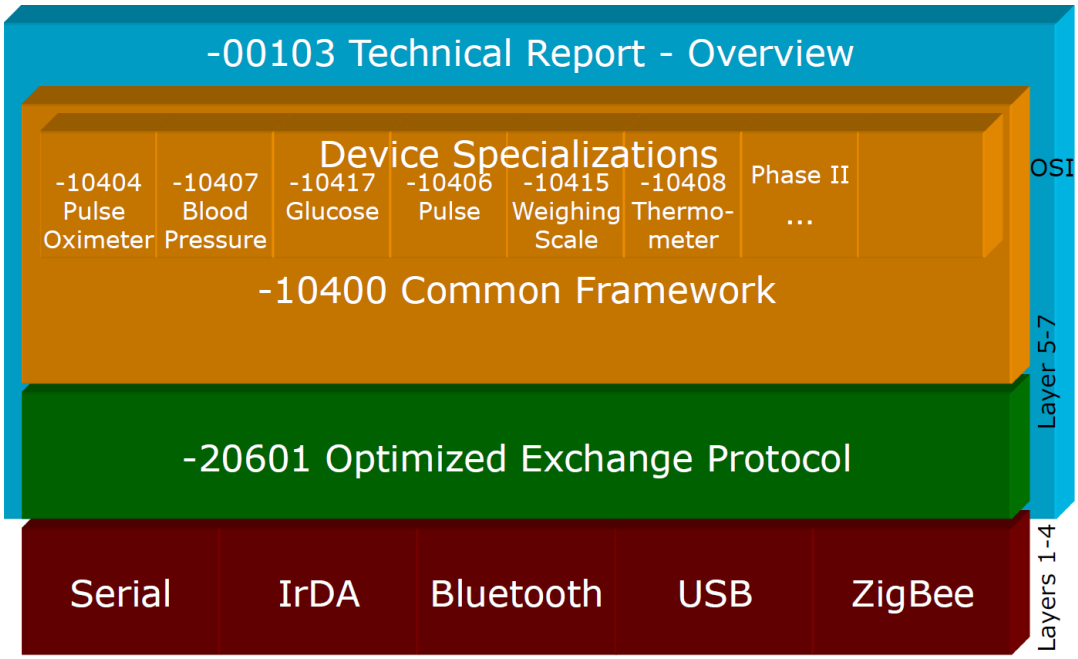


Рисунок 1.5 - Взаимосвязь стандартов телеметрии и модели OSI

1.1.2 Описание существующих решений мобильной медицины

Сервисы по телемедицине в последнее время распространены для:

- путешественников, т.к. это помогает сокращать время на получение консультации и не требует страховок;
- бизнесменов и работников офисов, у которых просто физически не хватает времени на посещение врачей;
- людей, которые хотят получить второе мнение врача по своему заболеванию, жителям регионов, у которых, бывает, нет доступа к врачу с высокой категорией;
- людей с хроническими заболеваниями.

Рассмотрим основные сервисы, распространенные в рамках Росссийской Федерации.

Телемедицина от МТС

Приложение мобильный оператор создал в соавторстве с группой клиник Медси. Их платформа называется SmartMed. Достаточно установить на смартфон приложение, чтобы можно было начать диалоги с врачами. Допускается общение с докторами посредством видеосвязи и чатов. Медицинская карта, результаты анализов и предлагаемые назначения хранятся в защищенном месте, но при этом доступны для просмотра прямо со смартфона в любое время.

Приложение позволяет пройти консультации и у узких специалистов.

Онлайн можно получить консультацию врача, сделать расшифровку анализов, обеспечить советы и вызвать экстренную помощь при необходимости. Врачи нередко оценивают данные, полученные с умных гаджетов — смарт-часов, пульсометров и т.д. Работает приложение по подписке.

Телемедицина от Tele2

Tele2 работает в партнерстве с компанией DOC+. Распространяется приложение по подписке, которая дает право на неограниченное число консультаций. Доступ к врачам обеспечивается посредством чатов, аудио- или видеозвонков. И тут нет никакого ограничения по времени приема. Педиатры и терапевты на связи круглосуточно.

Также приложение можно использовать для расшифровки анализов, получения назначений. Создается виртуальная медкарта, в которой содержатся все основные сведения о пациенте.

Получение данных с умных гаджетов отсутствует.

Телемедицина от Билайн

Билайн предлагает своим абонентам специальное медицинское приложение «Мой доктор». Создал он его в партнерстве с ООО «Мобильные Медицинские Технологии». Абонентам оператора доступны 150 врачей, которые проводят онлайн-консультации. Специалисты работают по 42 направлениям. В списке доступных услуг консультации, получение «второго

мнения» врача, расшифровка результатов. Сервис доступен в любое удобное время. В приложении определен базовый пакет услуг, который имеет определенную цену за месяц.

Телемедицина от Мегафон

МегаФон стал первым из операторов, кто запустил свой телемедицинский сервис — «МегаФон Здоровье». Это целый сервис медицинских онлайн-консультаций. Получить онлайн-консультацию врачей через мобильное приложение «МегаФона» смогут абоненты всех операторов связи, работающих в зоне +7. Воспользоваться услугой можно по всей России и за ее пределами.

Связь с врачом в приложении осуществляется по трем разным каналам: видео звонок, чат, аудио звонок.

Получение данных с умных гаджетов отсутствует.

Списывать оплату за консультацию можно как с банковской карты, так и со счета телефона.

Из вышесказанного можно сделать вывод, что на данный момент создание бесплатного мобильного приложения телеметрии, для связи с устройствами пациента является актуальной темой, т.к. известные решения являются в основном платными, закрытыми для добавления новых функций и не все из них поддерживают связь с персональными устройствами напрямую, требуя наличия доступа к облачным сервисам оператора связи, что не всегда возможно.

1.2 Выбор мобильной операционной системы

По аналитическим данным всемирного агентства мобильных разработок, Android лидирует на рынке мобильных устройств, как с точки зрения пользователей, так и разработчиков.

По последним статистическим данным, операционной системой от Google пользуются более 80% пользователей во всем мире. Господство Google объясняется доступностью и широчайшим ассортиментом мобильных гаджетов.

ОС iOS от Apple долго занимала лидирующие позиции благодаря удобству, стабильности и превосходному дизайну. Однако со временем ситуация изменилась: Google, последовательно модернизируя ОС Android, привнесла важные обновления и адаптировала ее под 64-разрядную систему.

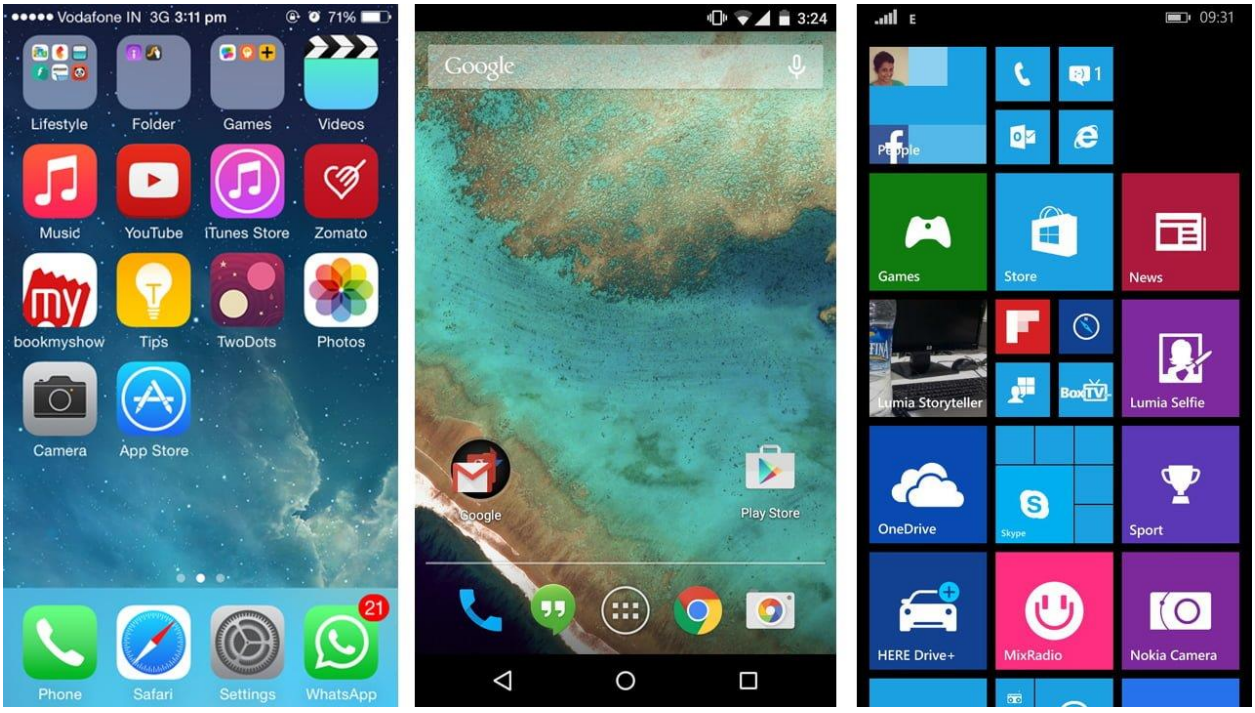


Рисунок 1.6 - Скриншоты тройки мобильных операционных систем.

Ниже в таблице 1.1 представлены основные характеристики сравнения трех лидеров рынка мобильных ОС: Android, IOS, Windows Mobile.

Таблица 1.1 - Сравнительный анализ мобильных операционных систем.

Место	1	2	3
Производитель	Google	Apple	Microsoft
Общая оценка, баллов	87,7	85,6	72,9

Продолжение таблицы 1.1

Удобство использования (40%)	88	87	78
Функциональность (40%)	84	79	73
Магазин приложений (20%)	96	97	62
удобство использования			
Настройка Рабочих столов	продвинутая	простая	простая
Настройка приложений	продвинутая	продвинутая	простая
Быстрый доступ к настройкам	есть	есть	есть
Единый интерфейс в ОС и приложениях	есть	есть	есть
Использование жестов	есть	есть	есть
Поддержка виджетов	продвинутая	простая	простая
Геолокационные сервисы	есть	есть	есть
Доступ с заблокированного экрана	фотокамера	фотокамера	фотокамера
Подключение к компьютеру	доступ через Проводник	(iTunes)	доступ через Проводник

Продолжение таблицы 1.1

функциональность			
Онлайн-бэкап	приложения, данные, контакты	приложения, данные, контакты	приложения, данные, контакты
Бесплатное сетевое хранилище	15 Гбайт	5 Гбайт	15 Гбайт
Поддержка многозадачности	есть	есть	есть
Автоматическое энергосбережение	есть	нет	есть
Шифрование данных	есть	есть	нет
разработка приложений			
Время разработки	Освоение языка: быстрое. Время на разработку: высокое.	Освоение языка: Swift – быстрое; Objective-C – медленное. Время на разработку: среднее.	Освоение языка: быстрое. Время на разработку: среднее.
Наличие специалистов	В основе используется язык Java, поэтому программировать могут большое количество специалистов.	Язык Objective-C: узкий круг специалистов. Язык Swift: большое количество специалистов.	В основе используется язык C#, поэтому программировать могут большое количество специалистов

Продолжение таблицы 1.1

Удобство разработки и отладки	Инструменты разработки развиты в полной мере.	Инструменты разработки развиты в полной мере.	Инструменты разработки развиты в полной мере.
Скорость работы ОС	Время загрузки всех действий пользователя и систем: быстрое. Отклик системы на действия пользователя: быстрое. Возможность и количество времени обработки сложных запросов: быстрое, возможны задержки.	Время загрузки всех действий пользователя и систем: быстрое. Отклик системы на действия пользователя: быстрое. Возможность и количество времени обработки сложных запросов: быстрое, возможны задержки.	Время загрузки всех действий пользователя и систем: быстрое. Отклик системы на действия пользователя: быстрое. Возможность и количество времени обработки сложных запросов: быстрое, возможны задержки.
Эргономичность	Наличие ограничений отсутствует. Пользовательский интерфейс интуитивно понятный.	Наличие ограничений. Пользовательский интерфейс интуитивно понятный.	Наличие ограничений отсутствует. Пользовательский интерфейс интуитивно понятный.

Продолжение таблицы 1.1

Интеграция со смежными системами	Интеграция возможна со смежными системами.	Интеграция возможна со смежными системами.	Интеграция возможна со смежными системами.
Открытость исходного кода ОС	Открыт	Закрыт	Закрыт

Для большей наглядности данных в таблице 1.1 продемонстрируем архитектурные решения мобильных ОС на рисунках 1.7 – 1.9.

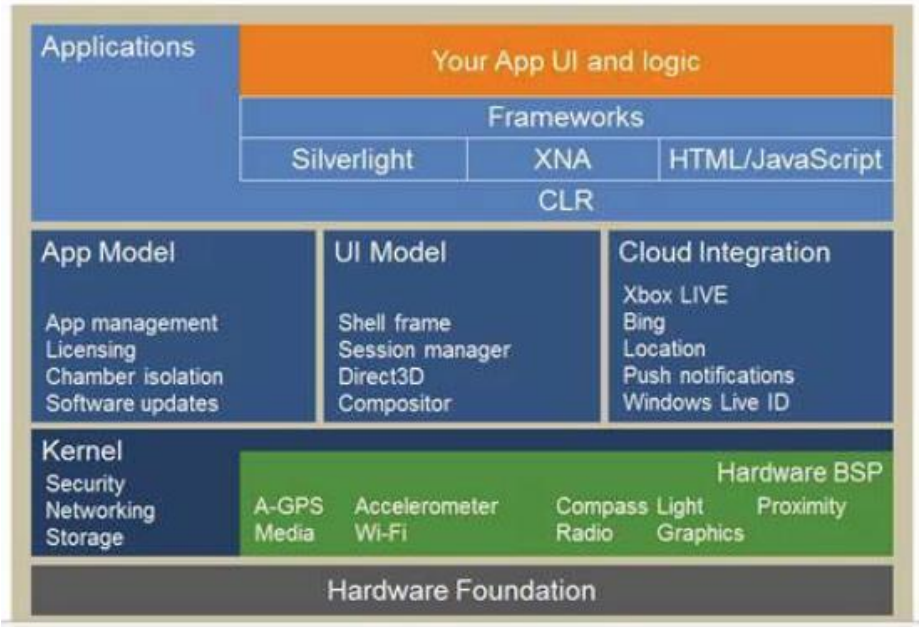


Рисунок 1.7 - Архитектурные решения Microsoft Windows Phone

Рисунок 1.8 - Архитектурные решения IOS

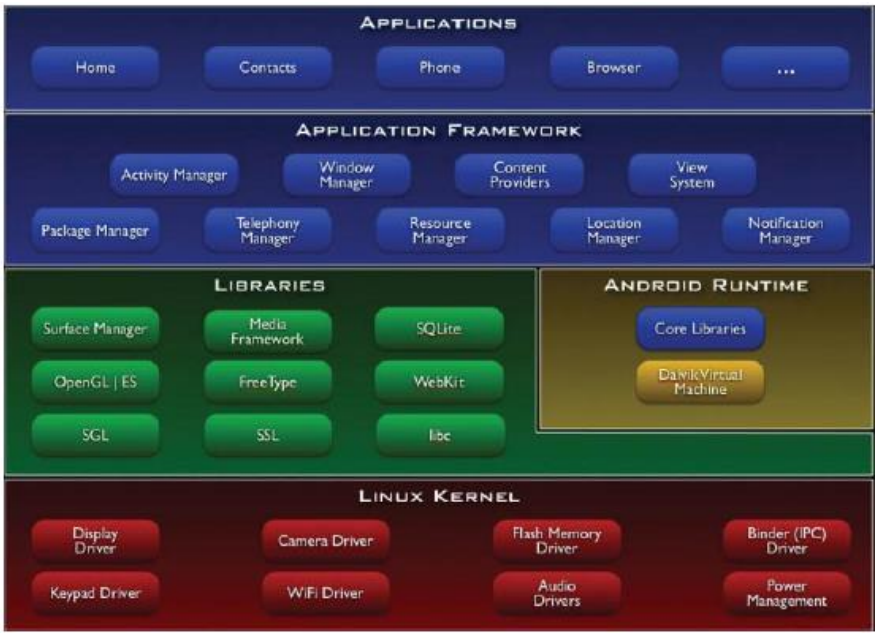


Рисунок 1.9 - Архитектурные решения Android

Проведенный анализ позволяет определить целесообразность использования и разработки приложений по мониторингу на мобильных телефонах. Вышеприведенное исследование приводит к выводу, что

выделенные операционные системы обладают всеми качествами, необходимыми для решения поставленной цели: способность решать трудоемкие задачи, интеграция с внешними системами и удобство использования.

В итоге ОС Android является наиболее приемлемой платформой для разработки приложений. Это подтверждается наличием простого освоения стека разработки и открытостью ОС.

1.3 Обзор устройств телеметрии

Система ViSiMobile

Система ViSi Mobile позволяет соединяться с большинством известных приборов для мониторинга, пока устройство находится в пределах действия беспроводной сети, мобильный сервер может контролировать жизненно важные показатели пациента.

При переключении с одной беспроводной сети на другую, в работе устройства может образоваться перерыв от 10 до 30 секунд. Данные о жизненных функциях отображаются на экране мобильного телефона. Эти данные также передаются в режиме реального времени по беспроводной связи через сети 802.11b.

Система предназначена для связи со стационарными ПК, планшетами и другими мобильными устройствами. Также имеется возможность открытого доступа API через устройства VISI Mobile. Это позволяет программировать доступ к данным на сервере и клиенте, различным приложениям или СУБД.

Рисунок 1.10 - Вид оконечного устройства ViSi Mobile

Система LifiMonitor

Для разработчиков доступны программные модули и библиотеки включающие модули разработки сторонних производителей.

Имеет интеграцию с системами управления и контроля для управления показателями жизнедеятельности одного или нескольких пациентов.

Чтобы приспособить / получить больше данных, в LifiMonitor могут быть интегрированы датчики от разных вендоров (проводные или беспроводные). Это расширяет его возможности для измерения разных параметров вплоть до GPS координат. Когда LifiMonitor приводится в действие без дополнительных датчиков и в режиме полного раскрытия информации, его срок службы батареи достигнет 52 часов. Несколько аккумуляторных блоков можно поместить в стопку для зарядки и затем заменить их в EQ02, чтобы быстро продлить сеансы мониторинга. Внутренняя память объемом 8 ГБ обеспечит непрерывную регистрацию данных до 50 дней [89].

Рисунок 1.11 - Вид оконечного устройства LifiMonitor

Система Zephyr Bioharness

Данные от этого типа устройств поступают в реальном режиме времени. Соединение от Zephyr Bioharness с другими удаленными устройствами осуществляется с помощью wi-fi или мобильных сетей (Bluetooth) для передачи данных.

Устройство обладает расширенным инструментарием разработчика:

- HxM Smart Монитор сердечного ритма SDK;
- HxM BT Монитор сердечного ритма SDK;
- BioModule (BioHarness) Bluetooth с низким энергопотреблением (BTLE) SDK.

HxM Smart Монитор сердечного ритма SDK:

- HxM Smart использует стандартные профили мониторинга сердечного ритма Bluetooth Low Energy (BTLE). К совместимым устройствам относятся iPhone 4S (Bluetooth 4.0) или более поздние версии и некоторые Android-устройства;
- SDK поставляется с документацией для поддержки пользовательских пакетов, которая включает в себя активность дыхания (RR), максимальное ускорение, возможность обновления прошивки и многое другое.

HxM BT Монитор сердечного ритма SDK:

– Используйте этот комплект инструментальных средств для реализации API, чтобы обеспечить связь с устройством Zephyr™ Bluetooth для измерения частоты сердечных сокращений / скорости и расстояния. Доступны как ПК, так и Android ресурсы для подключения к устройству получения онлайн данных.

Рисунок 1.12 - Вид оконечного устройства Zephyr Bioharness

BioModule ™ (BioHarness) Bluetooth с низким энергопотреблением (BTLE) SDK:

– спецификация связи Bluetooth BioModule;
– руководство по BioModule Bluetooth API;
– интерфейс системы ведения журнала BioModule Bluetooth (включая исходный код утилиты загрузки пробного журнала на Java);
– утилита обновления прошивки BioModule;
– Zephyr USB драйвер;
– приложение для тестирования Zephyr Bluetooth;
– пример исходного кода Android.

Исследование в данном разделе показало, что самый широкий инструментарий для разработчика предоставляют телемониторинговые системы Zephyr, которые к тому же позволяют взаимодействовать с различными устройствами следующими способами: Bluetooth, Wi-Fi, SMS, E-mail.

2 Реализация приложения

В приложении связь между периферийным устройством и мобильным приложением реализована на базе технологии BLE. Разработанное мобильное приложение может работать в следующих режимах:

- режим клиента с конечным устройством. Клиент работает с конечным устройством с помощью способов BLE технологии. Клиент принимает информацию от сервера;
- режим клиента с основным сервером (СУБД для сбора и визуализации данных). Сервер может работать с мобильным устройством с помощью Wi-Fi технологии;
- режим визуализации данных о пользователе.

В первом разделе опишем способ взаимодействия между периферийным устройством и мобильным приложением.

Телефон с Android приложением в данной системе может выступать в качестве клиента, ожидающего входящие подключения от различных периферийных устройств; а с другой стороны может выступать в роли клиента, подключенного к серверу приложения для передачи статистику в центр хранения данных (формат передачи сообщений HL7).

Ответственный за соединение периферийных устройств и Android приложения с помощью технологии Bluetooth является пакет AndroidClient.Bluetooth. В этом пакете хранятся классы позволяющие приложению осуществить соединение с периферийным устройством, обзор доступных сервисов, обзор доступных характеристик сервисов, чтение/запись в режиме клиент/сервер.

Обобщенная архитектура системы представлена на рисунке 2.1.

Система состоит из трех основных частей:

- мобильного приложения, в основе которого лежат библиотеки взаимодействия с BLE и библиотека преобразования данных с

периферийного устройства в данные JSON формата HL7 (стандарт обмена, управления и интеграции электронной медицинской информации);

– web сервера для хранения и визуализации полученных данных;
– периферийного устройства генерации данных.

Рисунок 2.1 – Аппаратно-программная система

Периферийное устройство генерации данных работает по протоколу BLE. Протокол BLE строго структурирован по принципу своей коммуникации с другими устройствами. Вначале устройства изучают доступные сервисы для отправки/принятия данных; неотъемлемая часть этих сервисов – их характеристики (characteristics), определяющие тип данных для будущей передачи. Характеристики, из соображений наглядности, могут иметь в своём составе описания-дескрипторы (descriptors), которые помогают определить тип данных.

Характеристика содержит однозначный параметр, а также дескрипторы. Количество дескрипторов может быть равно нулю, то есть это не обязательная часть характеристики. Дескрипторы описывают значение характеристики.

Дескрипторы представляют собой определённые атрибуты, которые описывают значение характеристики. Дескрипторы могут быть в виде понятных описаний на человеческом языке, определять единицы измерений,

а также задавать ряд допустимых значений. Сервис это совокупность характеристик. Сервис является набором данных и сопоставленных поведений, чтобы выполнить конкретную функцию или функцию. В данном приложении будут использованы сервисы сердечного ритма, артериального давления и сервисы сбора информации о техническом состоянии устройства.

Мобильное приложение с помощью библиотека BLE API может взаимодействовать с периферийным устройством на основании знаний о сервисах устройства. Полученные от периферийного устройства данные с помощью библиотеки HAPI преобразуются в JSON формат и передаются серверу хранения данных по HTTPS каналу.

Сервер хранения данных представляет собой классический набор WAMP сервера и FTP хранилища для данных с мобильных устройств.

2.1 Режим работы с конечным устройством

Работа мобильного приложения с периферийным устройством в режиме клиента реализуется по схеме, представленной на рисунке 2.2.

Клиентское приложение реализуется следующим образом. Во все программные модули подключаем библиотеки: import java.BLEDevice; import java.Wire; import java.util.ble.Zephir; import java.util.UUID.

Основу приложения составляют три класса: BluetoothChatService, BluetoothViewer, DeviceListActivity.

Класс BluetoothChatService используется для создания и управления Bluetooth-соединениями. В данной реализации управления соединениями есть два дополнительных подкласса ConnecThread и Connected Thread. Первый используется для установления соединения с устройством, а второй - для управления работой с устройством. Принципиальное значение имеет

также метод write (), используемый для отправки команды в конечное устройство.

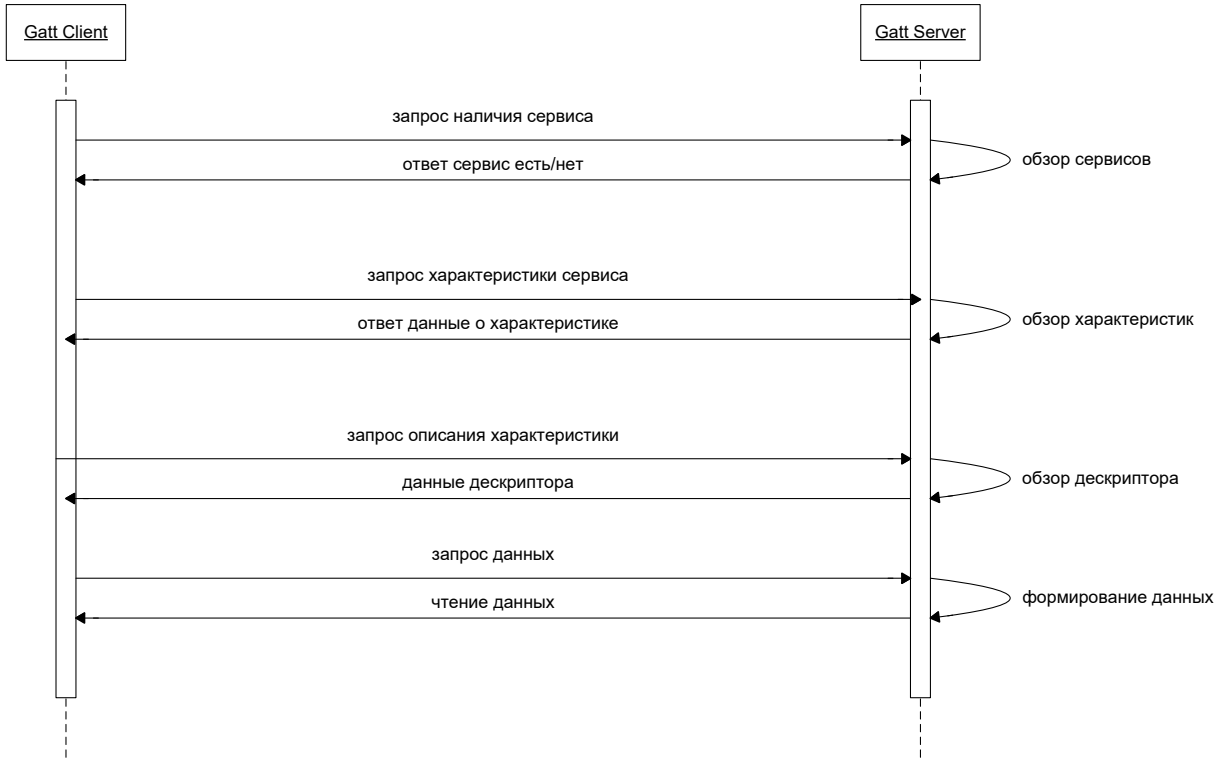


Рисунок 2.2 - Протокол взаимодействия клиент/сервер.

Создание соединения Bluetooth происходит путем создания объекта BluetoothChatService, распознавание устройств активируется нажатием кнопки, а вставленная команда управляется методом whatsend (строка), который выбирает команду для отправки через Bluetooth. Отправка сообщения в устройство происходит путем активации метода sendmessage (string), который использует метод write () объекта BluetoothChatService.

Класс ActivityListActivity - позволяет пользователю визуализировать список сопряженных устройств и выполнить сопряжение.

Первым шагом во взаимодействии с BLE-устройством станет подключение к нему (подключение к GATT-серверу) на устройстве. Для подключения к GATT-серверу на BLE-устройстве нужно использовать метод connectGatt(). Этот метод принимает три параметра: объект контекста,

соединение (логическое значение, указывающее, следует ли автоматически подключиться к BLE-устройству, как только оно станет доступным), и ссылку на BluetoothGattCallback:

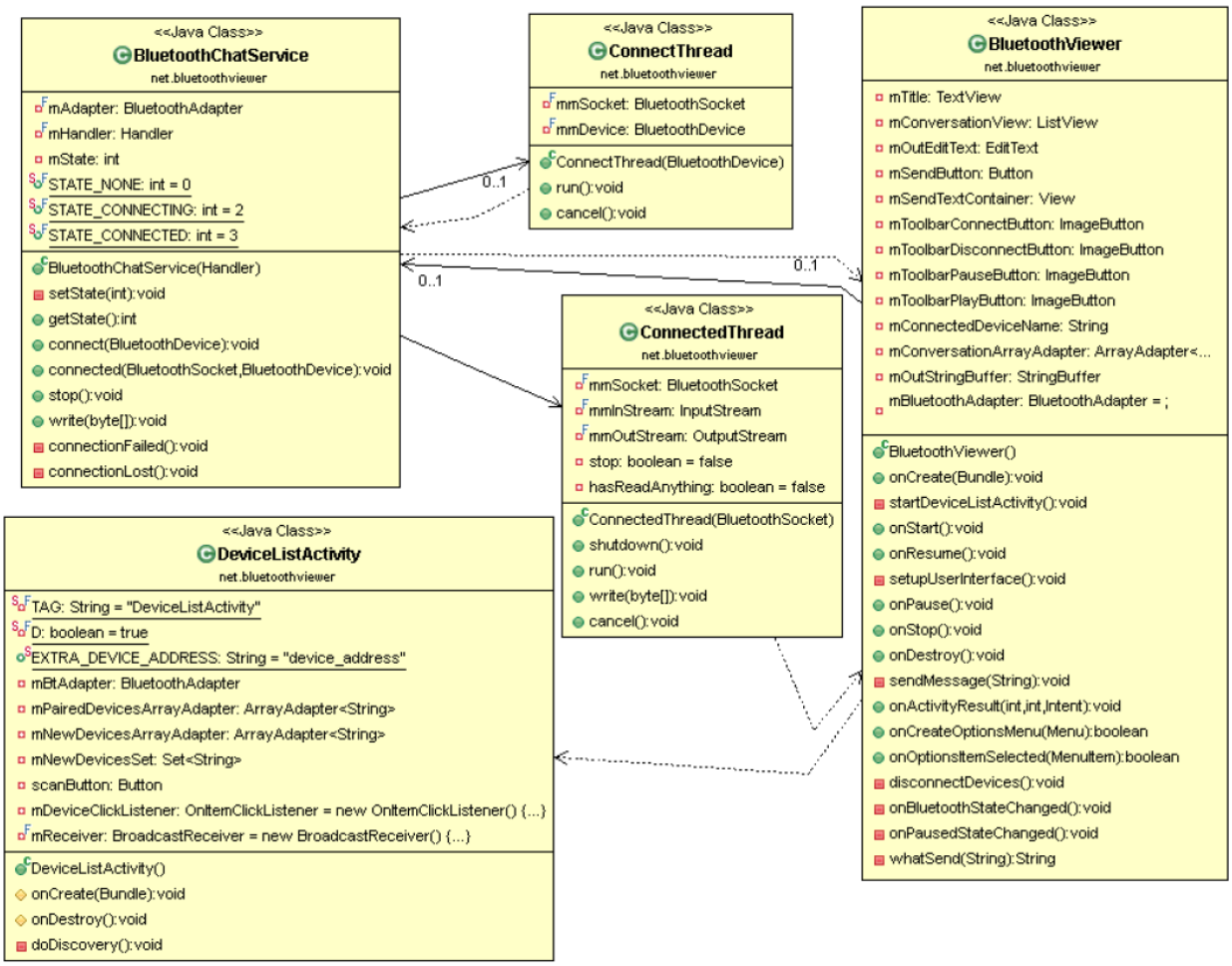


Рисунок 2.3 - Диаграмма классов мобильного приложения.

Мобильное приложение подключается к GATT-серверу, находящемуся на BLE-устройстве, и возвращает экземпляр BluetoothGatt, который затем можно использовать для проведения клиентских операций GATT. Приложение для Android является GATT-клиентом. BluetoothGattCallback используется для получения результатов клиентом, таких как статус подключения, а также любые дополнительные клиентские операции GATT.

BLE-приложение предоставляет активности (DeviceControlActivity) отображение данных о подключении, GATT-услугах и характеристиках,

поддерживаемых устройством. На основе ввода пользователя, эта активность связывается со службой под названием BluetoothLeService, который взаимодействует с BLE-устройством через Android BLE API-интерфейс: UUID.fromString(SampleGattAttributes.HEART_RATE_MEASUREMENT);
private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback().

При обнаружении нового сервиса мобильное приложение вызывает метод public void onServicesDiscovered(BluetoothGatt gatt, int status), для изучения характеристик сервиса и в конечном счете считывания информации с устройства.

Результат чтения характеристик сервиса метод public void onCharacteristicRead(BluetoothGatt gatt, BluetoothGattCharacteristic characteristic, int status) возвращает в объекте characteristic.

Когда срабатывает конкретный обратный вызов, он вызывает соответствующий вспомогательный метод broadcastUpdate() и передает ему действие. Извлечение данных в этот раздел выполняется в соответствии с профилем пульсометра.

Общие для BLE-приложений запросы для получения уведомлений об изменении отдельных характеристик на устройстве позволяют настроить уведомление для какой-либо характеристики с помощью метода setCharacteristicNotification() и получать поток информации о сердцебиении с устройства в автоматическом режиме.

Исходя из выше описанной архитектуры приложения, реализуется следующий интерфейс взаимодействия мобильного приложения и конечного устройства.

Пользователь запускает приложение и выбирает кнопку HDH Server на панели AndroidServerHome, после чего он попадает на панель ServerW3GorBT, где расположены две кнопки, связанные с двумя серверами: Bluetooth Server и Wi-Fi Server.

Функцию сервера можно реализовать программно тремя вариантами: Service, Activity, Thread.

Функция сервера в приложении реализована в виде сервиса (Service). В отличие от оставшихся вариантов сервис позволяет приложению работать с WiFi соединенем, обрабатывать сообщения в формате HL7.

Если приложение было разработано как Activity, прием сообщений HL7 происходил бы в жизненном цикле основного процесса, не позволяя приложению использовать ресурсы приложения и телефона для других целей во время выполнения программы, а данный момент является критичным для сопряжения мобильного приложения и конечного устройства.

Если сервис разработать в виде потока (Thread), мобильное устройство не запускает уведомления, что является критичным при получении сообщений с предупреждениями. Уведомления запускаются при работе с классами Service или Activity.

Обобщенная схема работы приложения в качестве сервера сбора информации от конечного устройства показана на рисунке 2.5

При нажатии на кнопку запуска Сервиса (Service On), сервер начинает ждать входящих соединений от активированного клиента. Пока система ждет клиентского отклика на панеле BluetoothManager, включается индикатор ожидания и уведомление, информирующее пользователя о том, что служба запущена. Каждый раз, когда приложение обращается к панеле BlueoothManager, чтобы увидеть состояние подключения клиентов, вызывается следующий метод stateService. Он проверяет, подключен хотя бы один клиент к серверу или нет. Если клиент подключен и уже работает, индикатор подключения будет во включенном состоянии. Каждый раз, когда пользователь пытается включить или выключить сервер, приложение отображает сообщение, предупреждающее пользователя о его действии. Ниже код функции stateService:

```
private boolean stateService(){
    // Check if the service is running
    ActivityManager activityManager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
    List<ActivityManager.RunningServiceInfo> runningServices = activityManager.getRunningServices(50);

    for (int i = 0; i < runningServices.size(); i++) {
        ActivityManager.RunningServiceInfo runningServiceInfo = runningServices.get(i);
        if (runningServiceInfo.service.getClassName().equals(getString(R.string.hdhmanagerservice))){
            isRunning=true;
        }
        Log.i("BluetoothManager", "Process " + runningServiceInfo.process + " with component " +
                runningServiceInfo.service.getClassName());
    }
    runningServices.clear();
    return isRunning;
}
```

При нажатии на кнопку «Service On», происходит запуск объекта класса HDHManagerService, в котором расположена точка старта приложения, которая запускает жизненный цикл службы.

В течение жизненного цикла сначала проверяется, поддерживает ли устройство, на котором установлено программное обеспечение, технологию Bluetooth. Если это так, создается уведомление, показывающее, что сервер прослушивает порты Bluetooth в ожидании подключения клиентов, а также создается объект класса BluetoothHDHService (который создает поток, когда начинается соединение между конечным устройством и мобильным устройством).

К объекту BluetoothHDHService добавлены обработчики для связи между ListenThread и службой Bluetooth.

После того, как объект BluetoothHDHService создан, ListenThread начинает работать в ожидании входящих соединений.

Прослушивая сокет входящего соединения. Как только соединение установлено объект класса HDMClient становится не равным null и происходит добавление нового клиента в список клиентов присоединившихся к мобильному приложению. Для формирования списка клиентов и их хранения используется объект класса ArrayList.

```
public void run()  //Android as Server
{
    setName("ListenThread");
    try
    {
        while(!this.finish)
        {
            BluetoothSocket socket = mmServerSocket.accept();
            if(socket != null)
            {   // Pass the socket to the "Talking" thread and add to the queue
                HDHClient client = new HDHClient(socket, mHandlerService);
                clients.addClient(client);
            }
        }
    // Close the server
    mmServerSocket.close();
    }catch(IOException e){
        Log.e(TAG, "IO error: " + e.getMessage());
    }catch (Exception e) {
        Log.e(TAG, "Unexpected error: " + e.getMessage());
    }
}
```

В функции run цикл while разрешает соединения с большим количеством клиентов. Объект класса BluetoothServerSocket является блокировщиком, это означает, что цикл while будет ждать в этой строке кода, пока не будет выполнено соединение извне, а затем вернет сокет Bluetooth, который идентифицирует соединение. Объект класса BluetoothServerSocket предупреждает приложение от постоянной 100% нагрузки на процессор мобильного устройства и его зависания.

При срабатывании socket, проверяется, правильно создан socket, и затем создается объект класса HDHClient.

Этот объект будет отвечать за связь с конечным устройством с помощью функции Thread, и устройство будет добавлено в список, в котором перечислены все другие подключенные в данный момент клиенты.

После добавления клиента объект BluetoothServerSocket вернется в состоянии ожидания следующего подключения.

Список пользователей контролируется классом HDHManagedClients.

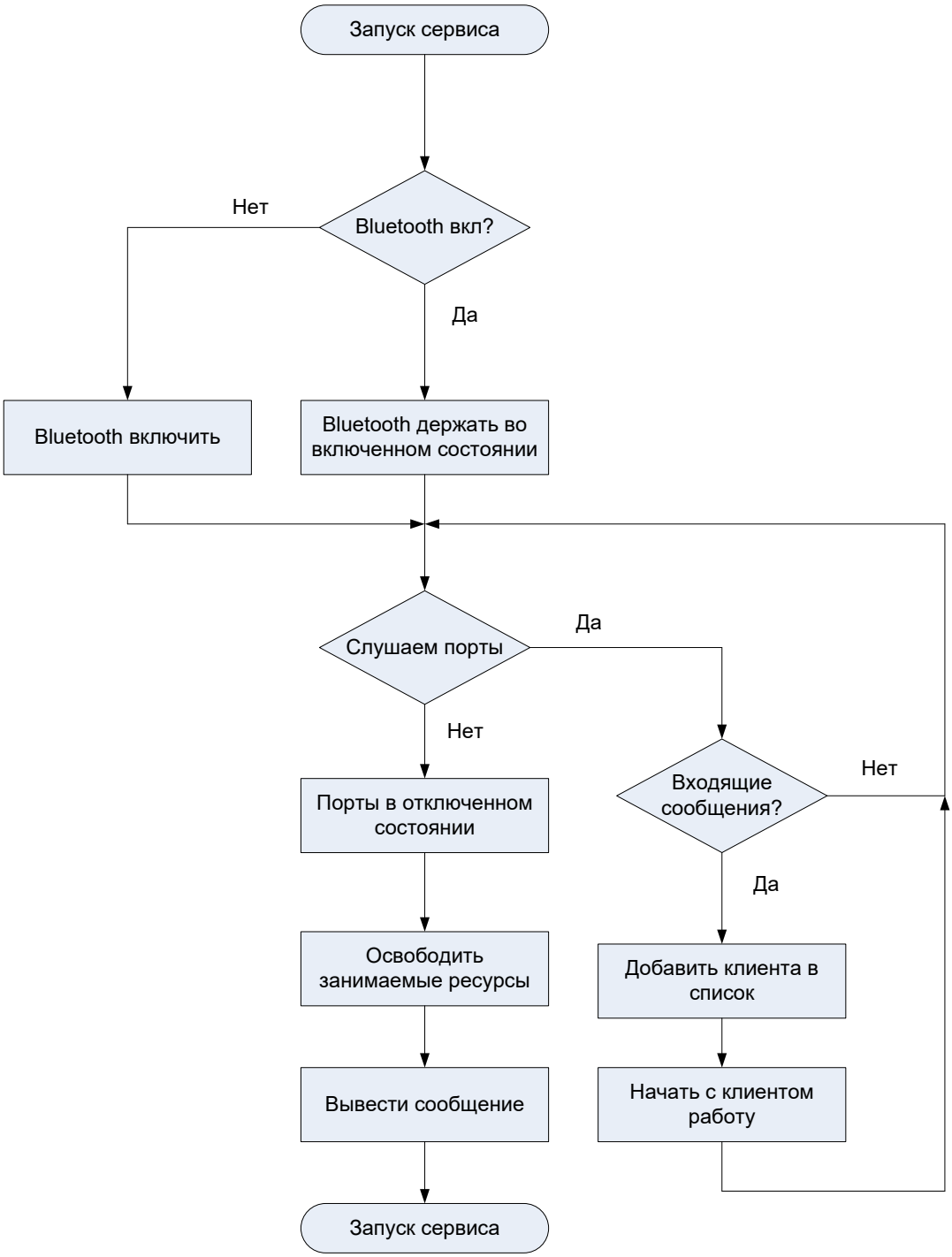


Рисунок 2.4 - Обобщенная работа приложения в режиме ожидания.

Этот класс отвечает за добавление новых клиентов или удаление уже существующих пользователей. С помощью сокетов и функции Thread, приложение реализуется в рамках многопоточного приложения. Ниже приведем ГСА работы потока обработки данных от подключенного клиента серверу.

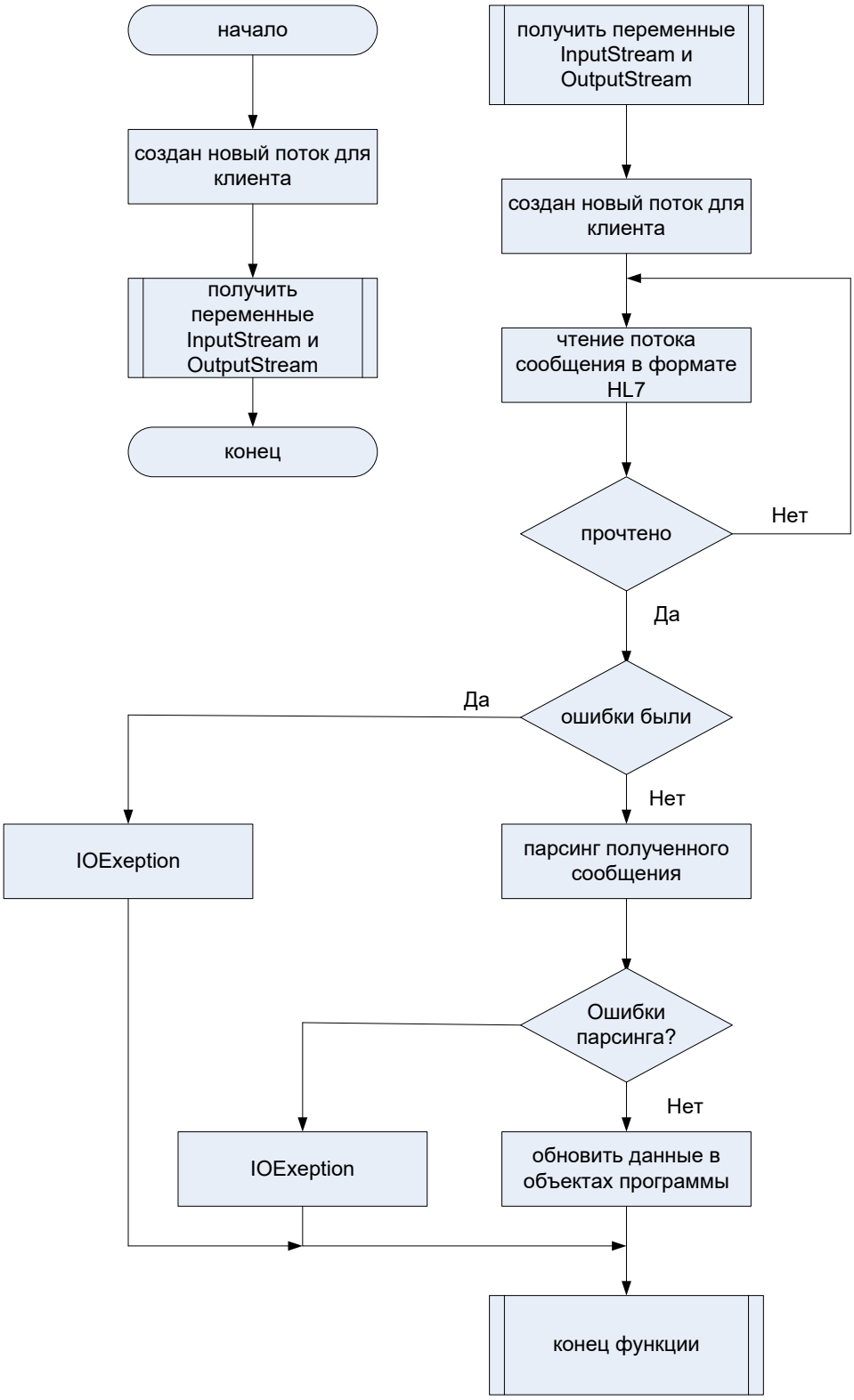


Рисунок 2.5 - Поток чтения данных от клиента сервером.

Поток ReceiverThread отвечает за прием сообщений от клиента в формате HL7. В листинге фиолетовым цветом выделена часть, которая работает с чтением буфера.

```
public void run() {
    setName("ReceiverThread");
    while(!this.finish){
        try {
            int bytesAvailable=1;
            String received="";
            while(bytesAvailable>0){
                byte[] buffer = new byte[1024];
                int readed = mSocket.getInputStream().read(buffer, 0, buffer.length);
                Log.d(TAG, "bytes readed "+String.valueOf(readed));
                Log.d(TAG, "Take it from the buffer: "+new String(buffer));
                Log.d(TAG, "ZZzzZZZZ");
                Thread.sleep(1000);
                Log.d(TAG, "Wake Up");
                bytesAvailable=mSocket.getInputStream().available();
                Log.d(TAG, "bytesAvailable without reading "+String.valueOf(bytesAvailable));

                String received1=new String(buffer).substring(0, readed);
                String received2=ParcheSubstring(received1);
                received=received+received2;
            }
            Log.d(TAG, "Received final: "+received);
            // Remove the null pointer
            received= received.substring(0, received.length()-1);
            HL7Manager hl7=new HL7Manager(received);
            hl7.start();
            Log.d(TAG, "Launching HL7Manager Thread");

        }catch (NullPointerException e) {
        }catch (IOException e) {
            // Disconnect Event
            Log.d(TAG, "EOF readed because channel is closed");
            break;
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    Log.d(TAG, "Receiver thread exiting.");
}
```

Процесс чтения выполняется следующим образом. Класс InputStream слушает поток до тех пор, пока в буфере не появится новое сообщение. При поступлении новых данных, объект InputStream читают из буфера байты в формате HL7.

После прочтения всего сообщения поток ReceiverThread переходит в состоянии ожидания нового сообщения. Таким образом, ReceiverThread проверяет, есть ли что-то еще в буфере, и если это так, ReceiverThread читает и добавляет часть сообщения к уже полученному ранее сообщению. Объект класса ParcheSubstring форматирует полученные сообщения, удаляя пробелы и нечитаемые символы, оставляя только рабочую информацию.

При возникновении ошибки в процессе чтения объект класса IOException обрабатывает все исключительные ситуации, возникающие в данном процессе.

После приема сообщения, сообщение анализируется в отдельном потоке HL7Manager, объектом класса HDHClient.

Прием сообщений происходит одно за другим и каждый раз, когда принимается новое сообщение, мобильное приложение отправляет сообщение подтверждения конечному устройству. Если произошла исключительная ситуация и сообщение не получено, конечное устройство повторно отправляет сообщение.

Длина сообщений фиксирована и равна 1024 байта. Таймер отправки повторных сообщений равен 2 секунды (рекомендованная задержка по официальной документации Zephyr Bioharness).

```
public void run() {
    setName("HL7Manager");
    try{
        HL7object hl7=new HL7object(HL7Message);
        Log.d(TAG, "HL7 parse susccessful");
        ACK ack=ACK.create ACK(hl7);

        if (ack.getAck_msa().get_acknowlegment_Code()=="AR"){
            this.writeBT(ack);
            Log.d(TAG, "Writting ACK AR");
        }
        if(ack.getAck_msa().get_acknowlegment_Code()=="AA"){
            SDCard.writeMessage(HL7Message, hl7);
            this.writeBT(ack);
            Log.d(TAG, "Writting ACK AA");
            if (hl7.getMsh().get messageControlID().substring(0, 2).equals("PI")){
                UpdateXml upxml=new UpdateXml(hl7);
                upxml.refreshDataInXml();

            }if (hl7.getMsh().get messageControlID().substring(0, 2).equals("AL")){
                Alert al=new Alert(hl7, mHandlerService);
                al.start();
                Log.d(TAG, "Launch Alert Thread");
            }
        }
    }catch (Exception e){
        Log.e(TAG, "Can't Parse HL7 message");
        return;
    }
}
```

Когда мобильное приложение получило сообщение от конечного устройства, следующим шагом является его анализ. В случае успешного парсинга полученных данных создается виртуальный файл метаданных, которые сохраняется на SDCard. Объект класса UpdateXml отвечает за формирование файла метаданных.

```
public Handler mHandlerService=new Handler(){

    @Override
    public void handleMessage(Message msg) {
        // TODO Auto-generated method stub
        super.handleMessage(msg);
           switch (msg.what) {
            case 0:
                HL7object hl7=(HL7object) msg.obj;
                launchNotification(hl7);
            break;
            }
    }
};
```

Метод launchNotification в классе HDHManagerService отвечает за уведомления:

Каждое уведомление имеет уникальный идентификационный номер.

Для обновления информации о пользователе в файле XML-файле находящемся на SDCard используется метод refreshDataInXml в классе UPdateXML, который использует свободно распространяемую библиотеку simplexml.

На этом описание процессов получения и обновления информации с конечного устройства на мобильный телефон закончено.

Далее опишем процесс работы приложения при возникновении исключительной ситуации.

При возникновении исключительной ситуации выполнение потока прерывается, и приложение переходит в ожидание следующего сообщения, которое будет отправлено по истечении таймера.

При нажатии кнопки «Service off», чтобы отключить службу BluetoothServer, приложение проверяет, что объект BluetoothHDHService

существует и не равен null, т.к. данный объект использует поток ListenThread для прослушки сокетов. Если объект BluetoothHDHService доступен, его метод stop вызывается для завершения функции ListenThread и закрытия функции ServerSocket.

```
public synchronized void stop() {  //Android as Server
    // Stop the listening thread and free all the resources used by the communication
    if (mListenThread != null) {mListenThread.cancel(); mListenThread = null;}
    clients.freeAllResources();
}
```

После остановки потока необходимо освободить все ресурсы, используемые при обработке ресурсов.

Для этой цели будет вызван объект класса HDHManagedClients, метод freeAllResources.

```
public synchronized void freeAllResources () {
    Iterator<HDHClient> i = clients.iterator();
    while (i.hasNext()) {
        i.next().freeResources();
    }
    clients.clear();
}

public void freeResources() {
    if(receiver.isAlive())
        receiver.cancel();
}

public void cancel()
{
    this.finish = true;
    try {
        mSocket.close();
    } catch (IOException e) {
        Log.d(TAG, "close() failed");
    }
}
```

Метод freeAllResources будет освобождать по одному каждый ранее открытый поток ReceiverThread. Для этого вызывается метод freeResouces класса HDHClient, который проверяет, работает ли поток. Если это так, поток останавливается, а сокет закрывается, если он еще не закрыт.

Режим обнаружения конечного устройства необходим для сопряжения с конечным устройством.

Вначале оба устройства, наручное устройство и мобильный телефон находятся в равных условиях и друг о друге ничего не знают.

Для совместной работы должны быть сопряжены, чтобы они знали друг друга и могли обмениваться данными. Все соединения, которые будут использовать Bluetooth, будут использовать профиль последовательного порта и формат данных HL7.

После сопряжения пользователь может выбрать одно из этих возможных действий в меню действия BluetoothManager.

Выбрав «Подключить устройство», мобильное устройство может подключиться к Zephyr Bioharness и начать передачу сообщений, выбрав пункт меню «Сделать доступным», пользователь может сделать мобильный телефон доступным для обнаружения и, наконец, нажав кнопку «Отключить», пользователь может завершить соединение с устройством.

Параметр «Подключить устройство» показывает вспомогательный диалог, в котором отображаются все сопряженные объекты, чтобы пользователь мог сканировать устройства Bluetooth. В данном диалоге пользователь может выбрать устройство, к которому он хочет подключиться, для передачи сообщений в формате HL7.

Код поиска устройств приложением показан ниже:

```
private void ensureDiscoverable() {
    if (mBluetoothAdapter.getScanMode() !=
        BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
        //if the device is transparent for other devices
        Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        // Make it discoverable for 300 seconds
        discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
        startActivity(discoverableIntent);
    }
}
```

Управление обнаруженным устройством приложением показано на рисунках ниже:

Сканирование устройств выполняется в объекте класса DeviceListActivity.

Когда устройство выбрано из списка объектов DeviceListActivity, MAC-адрес устройства передается классу BluetoothManager с помощью метода onActivityResult.

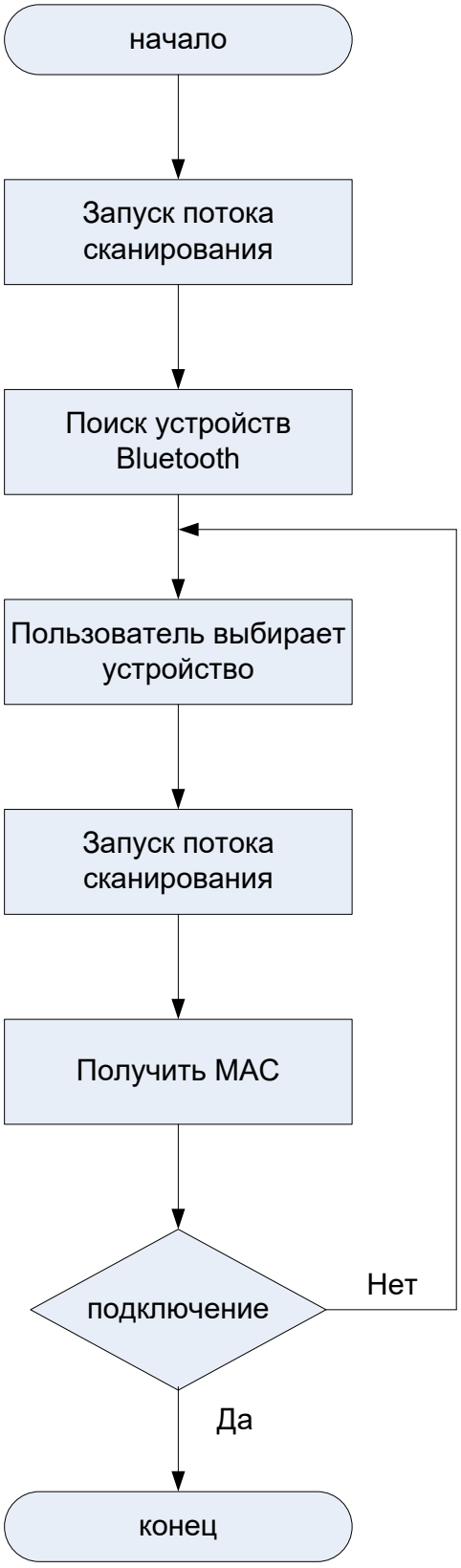


Рисунок 2.6 - Сканирование пространства BLE.

После получения MAC-адреса классом HDHClientService. HDHClientService отвечает за связь приложения с выбранным устройством.

Объект типа BluetoothHDHService создает объект ConnectThread, который создает сокет для связи и запускает поток для получения данных.

```
public ConnectThread(BluetoothDevice device) {
    mmDevice = device;
    BluetoothSocket tmp = null;
    try {
        // Create a socket for the connection
        tmp = device.createRfcommSocketToServiceRecord(MY_UUID_Serial_Port);
    } catch (IOException e) {
        Log.e(TAG, "create() failed", e);
    }
    mmSocket = tmp;
}
```

Объект BluetoothSocket создает сокеты различающиеся UUID_Serial_Port.

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
    case REQUEST_CONNECT_DEVICE:
        // When DeviceListActivity returns with a device to connect
        if (resultCode == Activity.RESULT_OK) {
            // Get the device MAC address
            String address = data.getExtras().getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
            Intent intent=new Intent(this, HDHClientService.class);
            Bundle b= new Bundle();
            b.putString("Address", address);
            intent.putExtras(b);
            startService(intent);
        }
        break;
    }
}
```

Как только сокет создан с UID для подключения, запускается поток ConnectThread.

Соединение класса BluetoothSocket если произошло исключение, создается и добавляется в очередь клиентов объект HDHClient. Он вызывает IOException, и происходит несколько попыток нового подключения.

2.2 Режим работы с основным сервером

Следующим режимом работы приложения является способность отправлять сохраненные сообщения на сервер.

Связь между мобильным телефоном и сервером осуществляется по Wi-Fi. Для правильного и полноценного обмена сообщениями сервер должен иметь IP-адрес и находиться в состоянии «прослушивания».

Соединение клиента и сервера, а также процесс передачи данных реализован в пакете AndroidServer.HDHMainServer, который содержит четыре класса.

```
public void run() {

    Log.i(TAG, "BEGIN mConnectThread"+mmDevice.getAddress());
    setName("ConnectThread");
    // Cancel the discovery that it was launched before cause the device has been selected
    mAdapter.cancelDiscovery();
    try {
        Log.i(TAG, "Trying to connect");
        mmSocket.connect();
    } catch (IOException e) {
        try {

            Log.e(TAG, "error: connect failed");
            if (mConnectThread != null){
                Log.d(TAG, "ZZzzzZZZ");
                ConnectThread.sleep(30000);
                Log.d(TAG, "Wake Up");
                mConnectThread.cancel();
                BluetoothHDHService.this.connect(mmDevice);
            }
        }
        catch (InterruptedException e3) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            Log.e(TAG, "Thread didnt sleep", e3);
        }
        return;
    }
    // Pass the socket to the "Talking" thread and add to the queue
    Log.d(TAG, "Successful Connection ");
    HDHClientforServer client = new HDHClientforServer(mmSocket, mHandlerService);
```

На панели HDHMainServer существует форма, на которой существуют кнопки «УСТРОЙСТВА», «СЕРВЕР», «КЛИЕНТЫ».

При переходе на панель «ОСНОВНОЙ СЕРВЕР» появляется запрос IP адреса и порта основного сервера для приема данных от приложения.

В данном серверном приложении в первую очередь делается проверка наличия доступной системы хранения в мобильном устройстве (в пакете com.upna.AndroidServer.SDCard имеются классы для проверки и работы с SD-карта). Если SD-карта недоступна, приложение закрывается, и пользователь

будет уведомлен об этом сообщением, что в мобильном телефоне нет системы хранения.

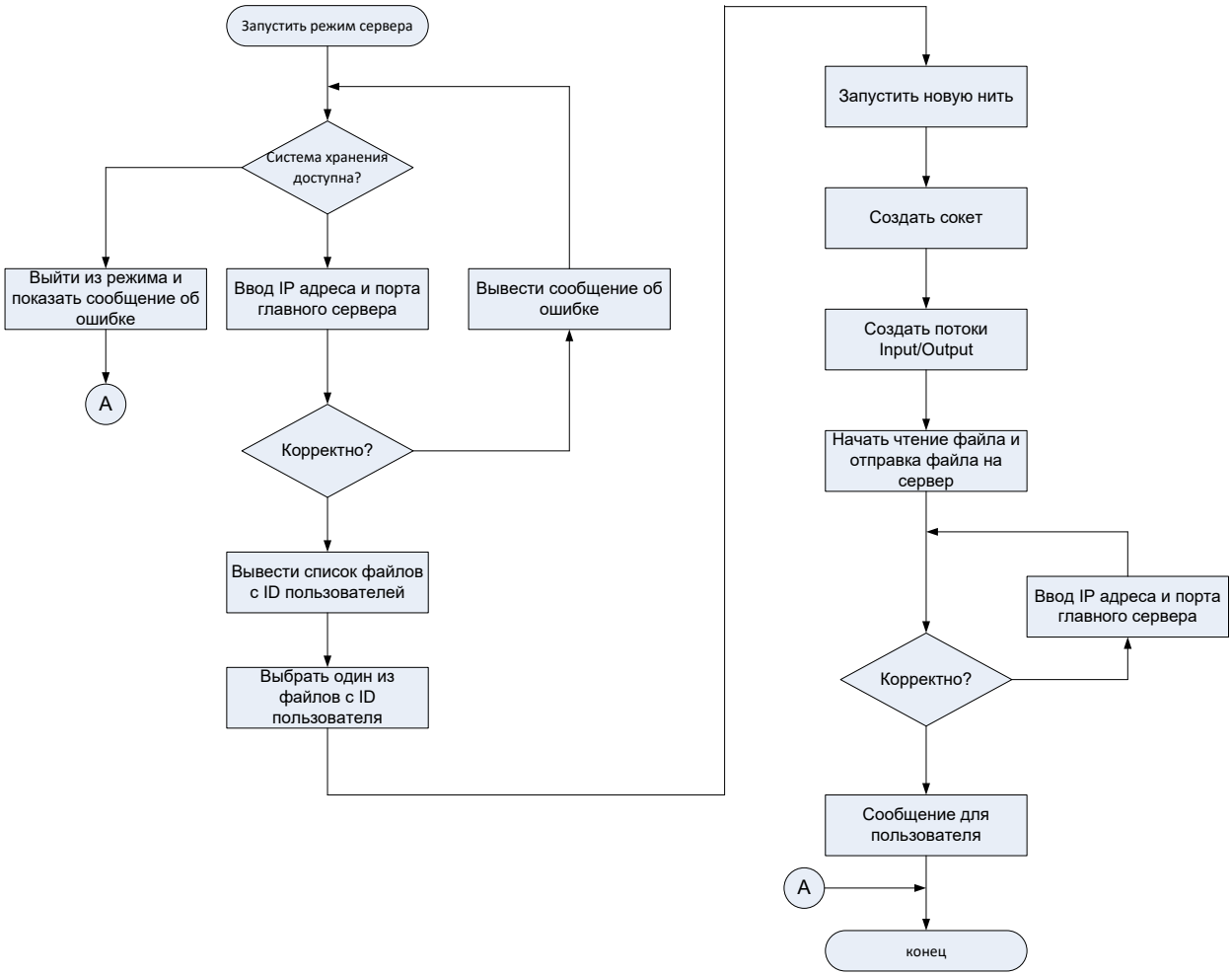


Рисунок 2.7 - ГСА работы мобильного приложения с основным сервером.

Наличие и правильность IP адресов и порты проверяются регулярными выражениями:

```
public boolean MatchIP(String IP){
    // Check a number with an IP
    final String REGEX = "\\d{1,3}.\\d{1,3}.\\d{1,3}.\\d{1,3}";
    boolean isValid=false;
    Pattern p = Pattern.compile(REGEX);
    Matcher m = p.matcher(IP);
    if((m.matches() == true)){
        String[] Iparray=IP.split("\\.");
        if(Integer.parseInt(Iparray[0])<=255 & Integer.parseInt(Iparray[1])<=255 &
                Integer.parseInt(Iparray[2])<=255 & Integer.parseInt(Iparray[3])<=255){
            isValid=true;
            return isValid;
        }else{
            return isValid;
        }
    }else{
        return isValid;
    }
}
```

Первая строка участок кода, имеет код выделенный прямоугольником, показывает, что IP будет правильным, если он удовлетворяет формату IP адреса. Под правильными IP адресами будут приниматься числа меньше 255.255.255.255.

То же самое применяется с кодом, отвечающим за проверку портов (порт числа не может превышать 65535).

Регулярное выражение для порта говорит, что порт должен иметь 4 или 5 цифр (порт сервера должен быть больше, чем общеизвестные порты: от 0 до 1023 и ниже, чем 65536).

```
public boolean MatchPort(String port){
    //Check a number with a port
    final String REGEX = "\\d{4,5}";
    boolean isValid=false;
    Pattern p = Pattern.compile(REGEX);
    Matcher m = p.matcher(port);
    if((m.matches() == true)){
        if ((Integer.parseInt(port)<=65536) & (Integer.parseInt(port)>=1024)){
            isValid=true;
            return isValid;
        }else{
            return isValid;
        }
        }else{
           return isValid;
        }
}
```

Кроме проверки IP-адреса и порта, внизу диалога видим списки файлов с сохраненной информацией. Для обработки данных файлов используются объекты классов PopUpWindow и CustomAdapter. Благодаря этим объектам карта SDCard будет освобождена от сохраненных файлов.

После выбора файла из списка, откроется новый поток запускающий объект класса HDHMainServerService, который передаст выбранные данные на сервер по выбранным IP-адресу и порту. За передачу данных в классе HDHMainServerService отвечают методы Intent и Bundle.

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    // TODO Auto-generated method stub
    super.onListItemClick(l, v, position, id);
    String portS= port.getText().toString();
    String ipS= ip.getText().toString();
    // Check if the IP or port are correct
    if(MatchIP(ipS) && MatchPort(portS)){
         Intent intent=new Intent(HDHMainServer.this, HDHMainServerService.class);
         Bundle b=new Bundle();
         b.putString("Patient", l.getItemAtPosition(position).toString());
         b.putString("IP", ipS);
         b.putString("PORT", portS);
         intent.putExtras(b);
         startService(intent);
    }else{
        Toast.makeText(this, getString(R.string.ip_port_vrong), Toast.LENGTH_SHORT).show();
    }
}
```

В методе onListItemClick класса ListActivity идет обработка IP адресов и портов. Функции startService и Toast класса Service отвечают за запуск процесса передачи данных от клиента серверу с выбранным массивом данных.

Как только объект класса HDHMainServerService создан, данные для передачи собираются в объекте HDHMainServerManager и пердаются по Wi-Fi.

```
public synchronized void connect(String ipaddress, String port, String pUpload) {
    // Start the thread to connect to the server
    if (mConnectThread == null) {
        mConnectThread = new ConnectThreadtoMainServer(ipaddress,Integer.valueOf(port),pUpload);
        mConnectThread.start();
    }
}
```

В классе HDHMainServerManager есть объект класса ConnectThreadtoMainServer, который расширяет функции класса Thread и отвечает за соединение. Работает ConnectThreadtoMainServerна основе TCP/IP.

```
public ConnectThreadtoMainServer(String ip, int port, String pUpload){
     this.ip_address=ip;
     this.port=port;
     this.pUpload=pUpload;
     this.clientSocket=new Socket();
     if (pUpload.equals("All Patients")){
          this.idsPatientUpload=SDCard.listDirPatient();
     }else{
          String[] id= {this.pUpload};
          this.idsPatientUpload=id;
     }
}
```

Ниже представлено потока создается сокет и устанавливается соединение с сервером. Если соединение установлено успешно, объекты InputStream и OutputStream будут созданы для записи и чтения из буфера Socket. Если соединение не установилось успешно, приложение закрывает поток, и пользователю будет отправлено сообщение.

```
public void run() {
    // TODO Auto-generated method stub
    super.run();
    setName("ConnectThreadtoMainServer");
    try {
        // Connecting with the server
        clientSocket.connect(new InetSocketAddress(ip_address, port));
    }catch (IOException e) {
        mHandlerServiceServer.obtainMessage(1, -1, -1, 0).sendToTarget();
        return;
    }
    if (clientSocket!=null){

        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        try {
            tmpIn = clientSocket.getInputStream();
            tmpOut = clientSocket.getOutputStream();
        } catch (IOException e) {
            mHandlerServiceServer.obtainMessage(1, -1, -1, 0).sendToTarget();
            return;
        }
        mmInStream = tmpIn;
        mmOutStream = tmpOut;
```

Поток проверяет, сколько раз была попытка отправить один и тот же файл (максимум в два раза) и не возникало ли ошибки при его отправке. Затем файл считывается с карты SDCard с помощью метода readFile класса SDCard. Данные записываются в выходной буфер и отправляются на сервер.

```
int x=0;
boolean breceived=false;
while(!breceived && x<10){
    try {
        if(mmInStream.available()==0){
            breceived=false;
            Thread.sleep(1000);
            Log.d(TAG,"Nothing in the buffer... waiting...");
        }else{
            breceived=true;
            Log.d(TAG,"We have something in the buffer...");
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    x++;
}
```

По умолчанию доступен метод InputStream, который будет ожидать, пока во входном буфере не появится данные доступные для чтения.

После отправки данных приложение выдает сообщения или положительного исхода отправки сообщения или об ошибке передачи.

2.3 Интерфейсы приложения

Работа приложения начинается с авторизации или регистрации в системе, в зависимости от того есть пользователь в БД приложения или нет (рисунок 2.8 и 2.9).

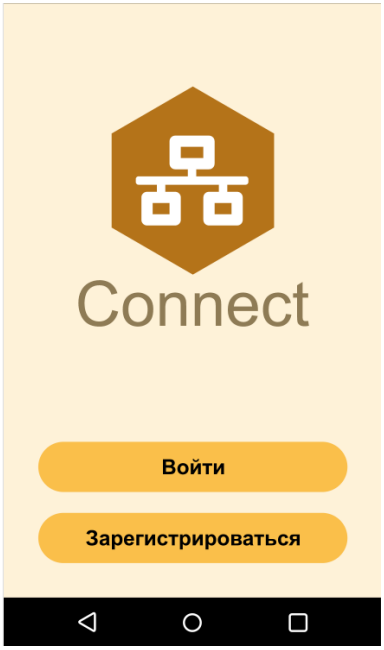


Рисунок 2.8 – Приглашение к вводу данных.

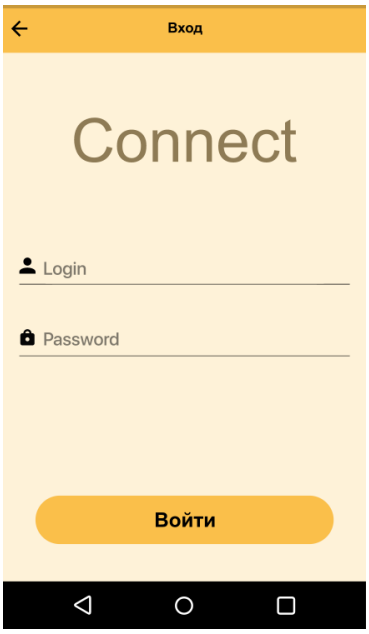


Рисунок 2.9 – Окно авторизации.

После ввода логина и пароля пользователь вызывает методы обработки пароля и логина пользователя, которые проверяют достоверность введенной информации. В случае обнаружения ошибки выводится сообщение (рисунок 2.10), которое предлагает ее исправить.

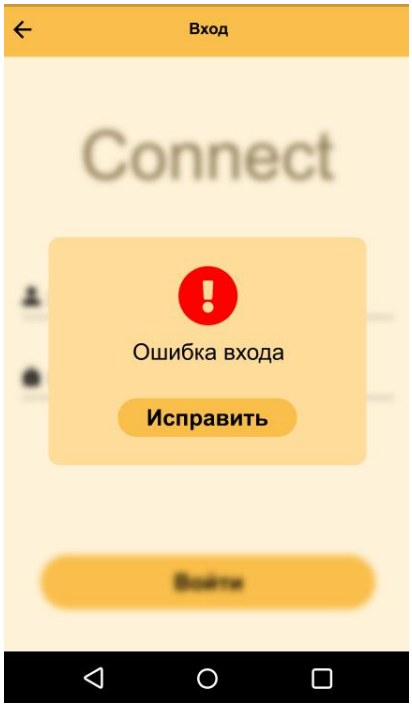


Рисунок 2.10 – Ошибка ввода.

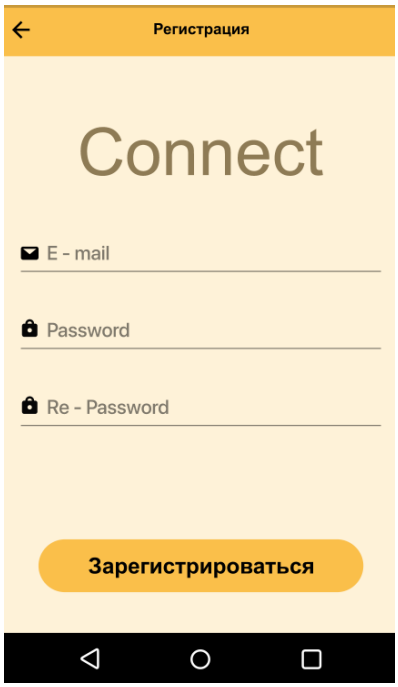


Рисунок 2.11 – Окно регистрации.

Окно регистрации позволяет зарегистрировать пользователя. Для реализации функций регистрации используется OAuth 2.0, протокол позволяет выдать приложению права на доступ к ресурсам пользователя на другом сервисе.

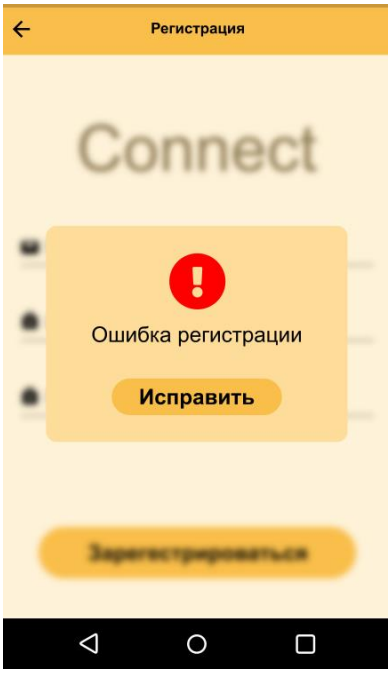


Рисунок 2.12 – Ошибка регистрации.

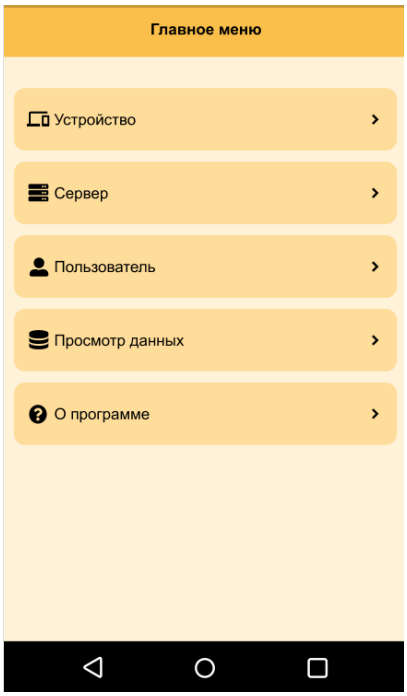


Рисунок 2.13 – Основное меню программы.

После авторизации пользователь попадает в основное меню программы (рисунок 2.13), в котором выбирает соответствующий пункт меню. В первую очередь требуется осуществить поиск устройств и найти нужное, с помощью меню «Устройство» (рисунок 2.14).

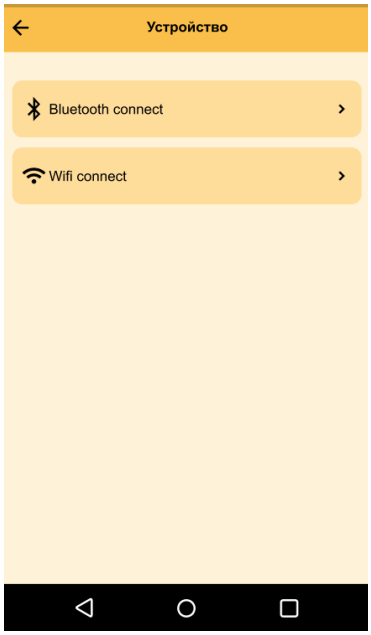


Рисунок 2.14– Меню устройство.

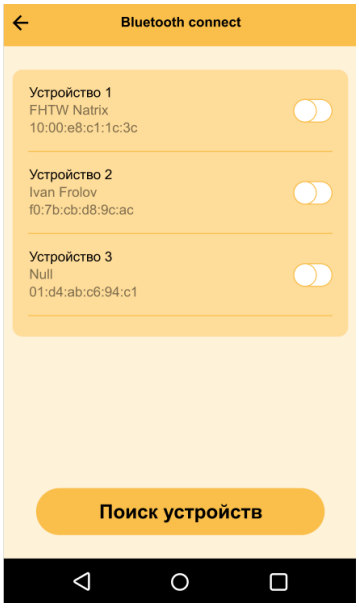


Рисунок 2.15 – Найденные устройства.

После выбора режима «Bluetooth», происходит автоматическое сканирование устройств готовых присоединиться по BLE соединению. А также появляется элемент управления, который позволяет пользователю выбрать соединение с необходимым устройством. Выбранное устройство тут же начинает обмениваться служебной информацией с мобильным устройством и передавать данные о сердечном ритме.

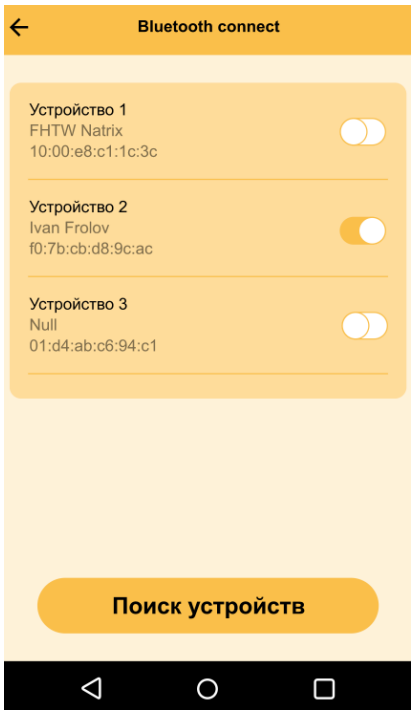


Рисунок 2.16 – Выбранное устройство.

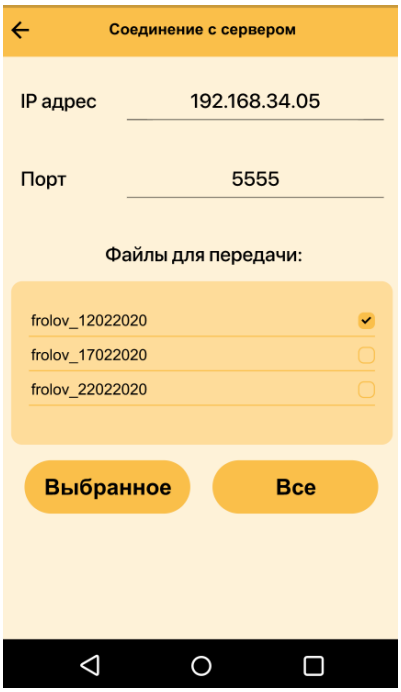


Рисунок 2.17 – Соединение с основным сервером.

Как только устройство соединилось с сервером, начинается набор данных в файл сбора статистики. Данные собираются в формате: {"sensor":"heart_rate", "androidId" : "12ca9042ebc1bf61" , "values" : {"time: ": "1467970379162" ," val: ":"71.0"}}, где первый параметр характеристика которую собирает приложение с конечного устройства, второй параметр это уникальный номер нашего мобильного устройства, третий параметр значения времени и частоты сердцебиения для формирования графики статистики.

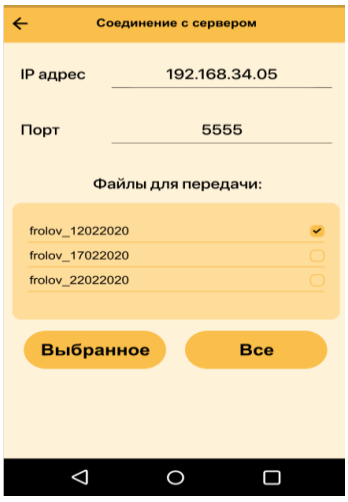


Рисунок 2.18– Файлы телеметрической информации для основного сервера.

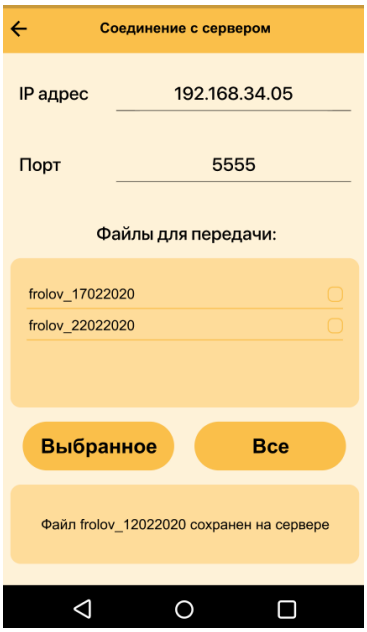


Рисунок 2.19 – Отправка файла на основной сервер.

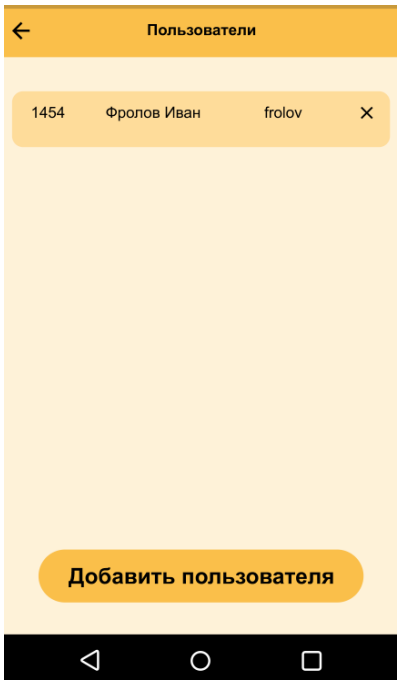


Рисунок 2.20 – Меню работы с пользователями.

Данные полученные от устройства конвертируются в файлы в формате HL7 для отправки на основной сервер. Для чего приложение использует стороннюю библиотеку HAPI, методы которой конверсируют файл перед отправкой.

3 Технология разработки приложений ОС Android в IDE Eclipse

Для того чтобы подготовиться к разработке приложения для ОС Android и взаимодействия с виртуальными модулями сторонних устройств в ОС установить следующее программное обеспечение:

IDE Eclipse - интегрированная среда разработки модульных кроссплатформенных приложений;

ADT плагин для IDE Eclipse - пользовательский плагин, для разработки приложений Android в IDE Eclipse;

JDK - комплект разработчика приложений на языке Java, включающий в себя компилятор Java, стандартные библиотеки классов Java, примеры, документацию, различные утилиты;

JRE - реализация виртуальной машины, необходимая для исполнения Java-приложений, без компилятора и других средств разработки;

Android SDK - средство разработки мобильных приложений для операционной системы Android.

3.1 Установка среды разработки Eclipse

Из раздела Downloads на официальном сайте Eclipse скачиваем версию IDE Eclipse Classic. Далее просто распакуйте архив в рабочий каталог дистрибутивов, по умолчанию в директорию C:\Eclipse.

Запускаем Eclipse, и открываем диалог установки плагина через меню Help → Install new software.

В поле Work with вписываем адрес загрузки плагина dl-ssl.google.com/android/eclipse (если не получилось, то плагин можно скачать и установить вручную по ссылке http://developer.android.com/sdk/eclipse-

adt.html), далее в таблице ниже появится Developer Tools, отмечаем его и идем далее.

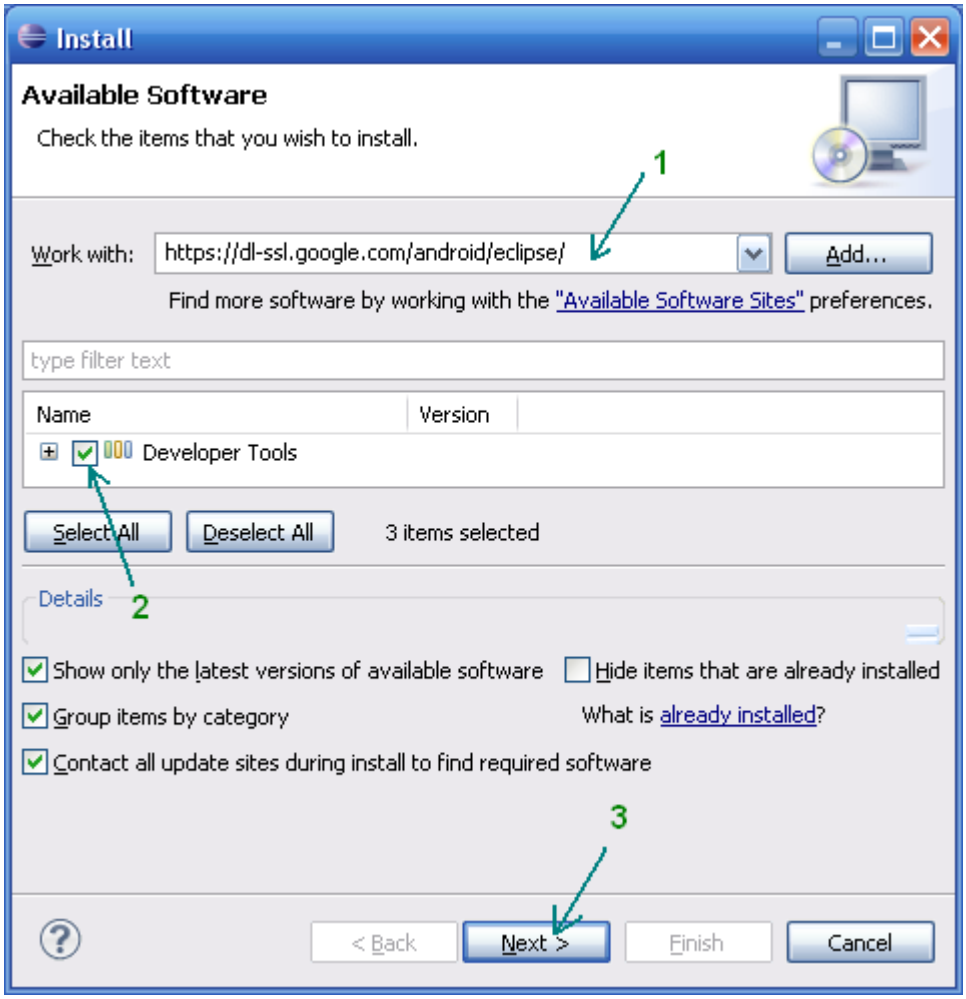


Рисунок 3.1 Диалог Developer Tools

После успешной установки, можно перезапустить среду Eclipse.

Если не установлены Java Development Kit (JDK) и Java Runtime Environment (JRE), то их необходимо установить. Сделать это можно на сайте Oracle. Скачайте и установите JDK и JRE.

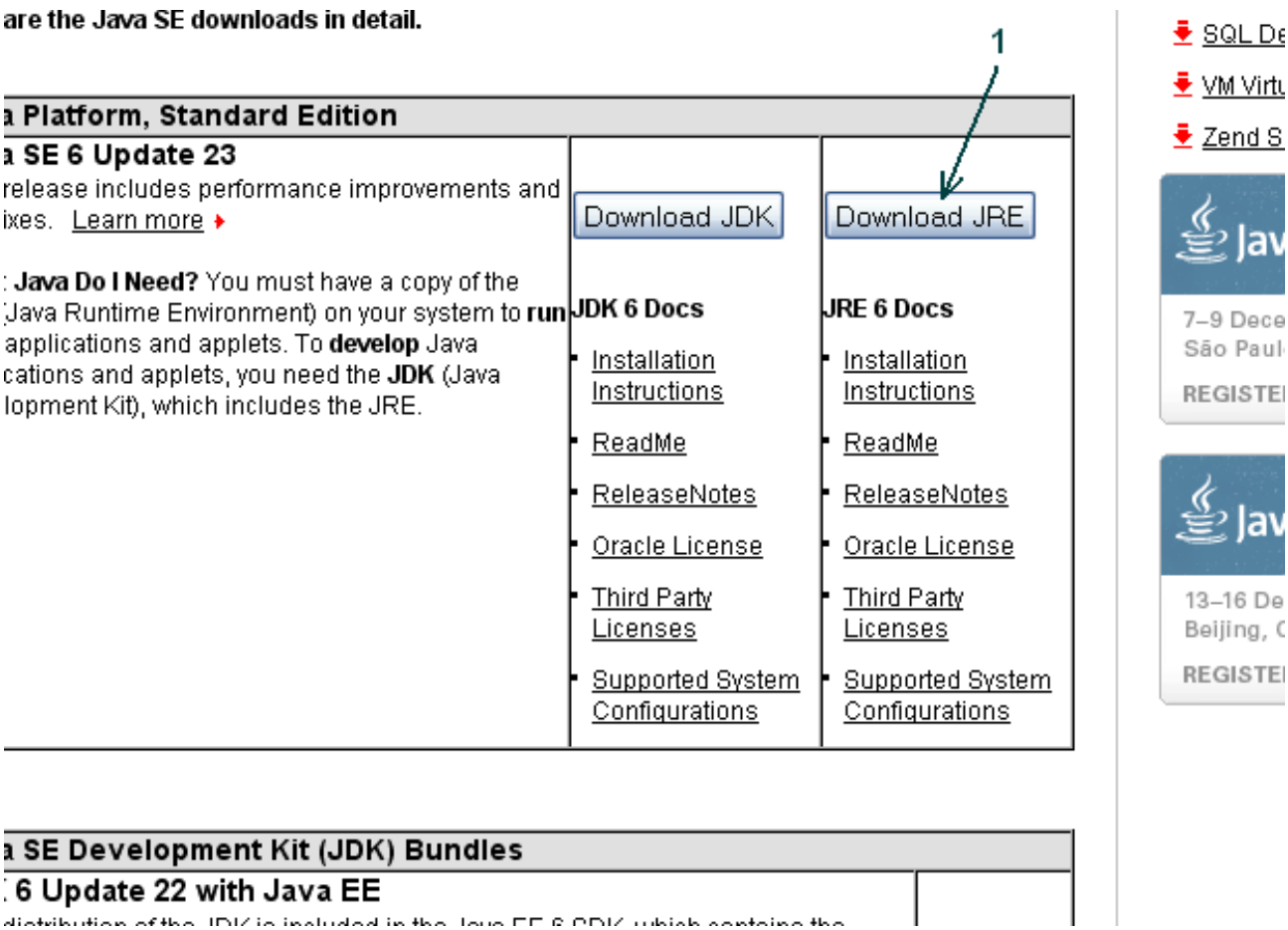


Рисунок 3.2 Установка JDK и JRE

3.2 Установка Android SDK

Далее требуется скачать и установить Android SDK. Это делается на сайте для Android разработчиков. Установим данное приложение директорию C:\Android. После этого необходимо добавить платформы и прочие дополнительные элементы SDK. При инсталляции требуется выбрать необходимые плагины, так как планируется реализовывать приложение для связи с Bluetooth устройствами и USB-драйвер.

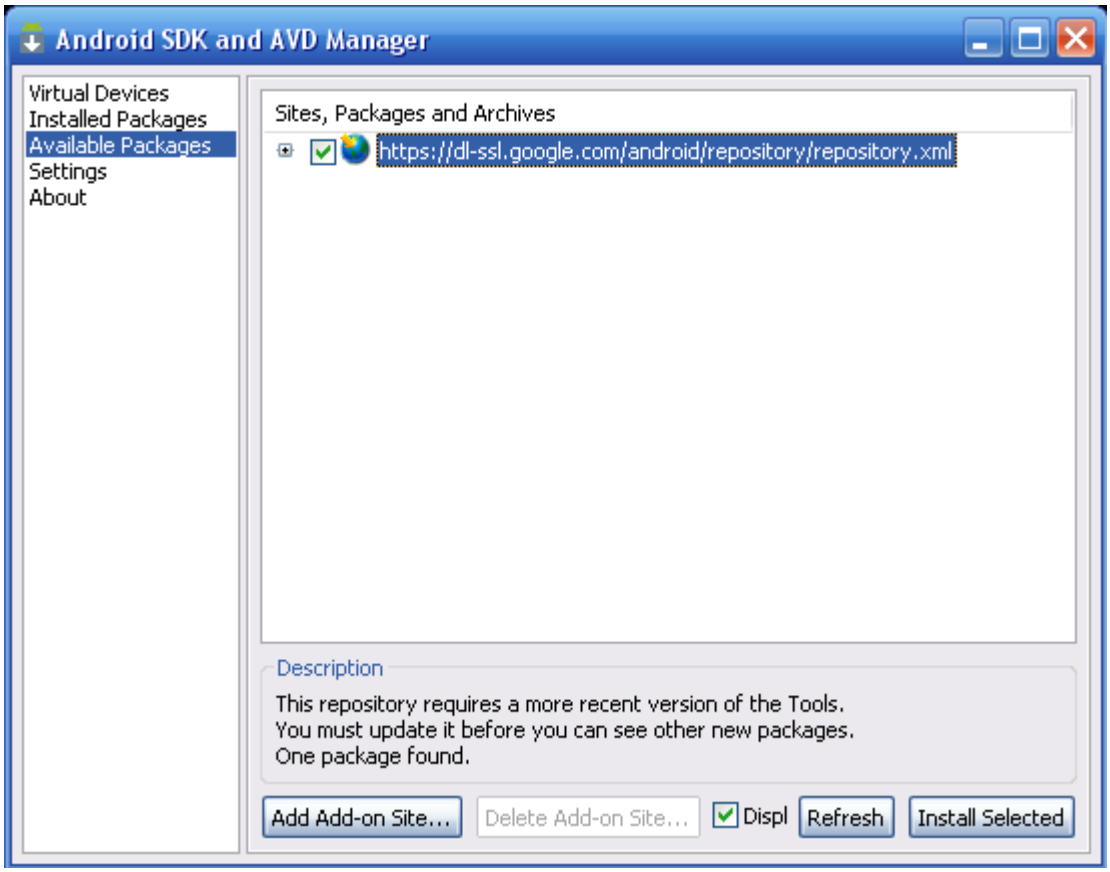


Рисунок 3.3 Android SDK

Перед тем как создать приложение, можно создать виртуальное Android-устройство, чтобы быстро тестировать на нём разрабатываемое программное обеспечение. Несколько слов по поводу Android Virtual Device (AVD). Это виртуальный смартфон с ОС Android, на котором можно запускать созданную программу. Как можно заметить, преимущество Android Virtual Device состоит в том, что наглядно можно увидеть, как будет работать программа на разных смартфонах с Android, а не покупать весь модельный ряд и тестировать приложение на каждом из них. Кроме того эта виртуальная среда способна взаимодействовать с виртуальным устройством Zephyr Bioharness.

3.3 Реализация минимального проекта

Подготовка к разработке завершена. Теперь следующий этап — создание приложения.

Приступим к созданию виртуального устройства Android. Следуем в меню по пути **Window → Android SDK and AVD Manager**.

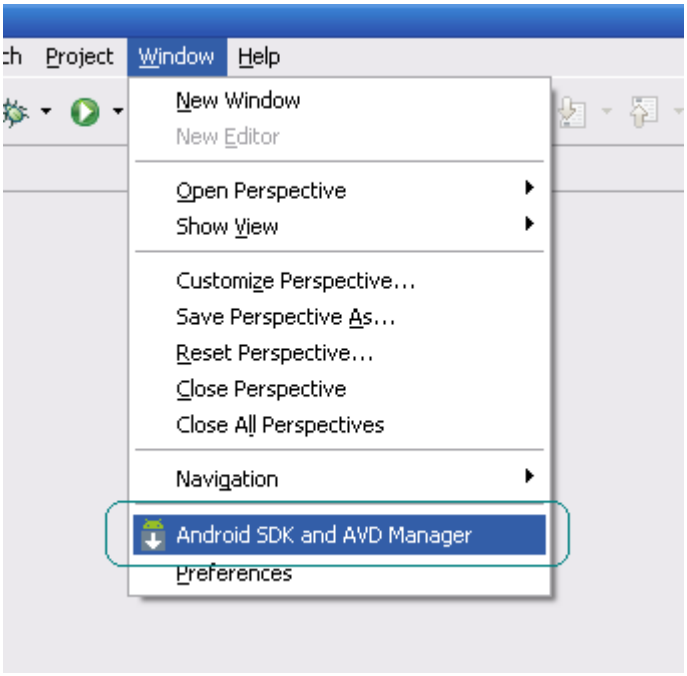


Рисунок 3.4 Созданию виртуального устройства Android

Выбираем New в правой части окна, в появившемся окне вписываем имя виртуального устройства, платформу Android, размер карты памяти (допустим 512 Mb), тип дисплея (допустим HVGA). Затем жмем внизу на кнопку Create AVD.

Теперь создадим проект. Для этого идем по пути в меню File → New → Other, в появившемся списке Android → Android Project.

В качестве примера продемонстрируем реализацию функции считывания температуры с датчиков Zephyr Bioharness Termometr, показывающей текущую температуру с датчиков, расположенных на виртуальном устройстве Zephyr Bioharness.

После создания проекта, слева увидите дерево директорий. Загрузим иконку приложения, а точнее 3 иконки под разные варианты. В папку

drawable-hdpi загрузим png-картинку с прозрачным фоном размером в 72x72, в drawable-mdpi соответственно 48x48 и в drawable-ldpi самую маленькую размером 36x36.

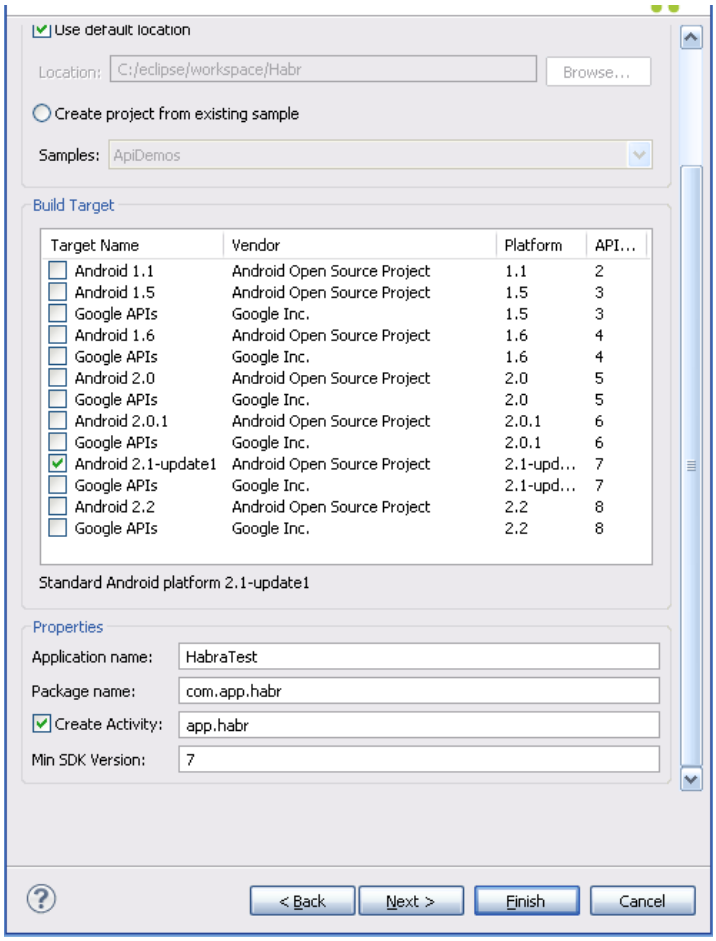


Рисунок 3.5 Обновление приложения до последней версии.

Делать это можно простым перетаскиванием файлов прямо на дерево. Следующим шагом будут элементы управления. В программе было необходимо всего три элемента интерфейса: ImageView (просто картинка логотип приложения), Button (обновить значение температуры) и TextView (вывод значения температуры). Все эти элементы управления нужно описать в специальном xml-файле. В дереве он расположен по адресу res → layout → main.xml.

Разметка элементов управления сходна с вёрсткой веб-страниц, тут есть и padding, и margin, и аналог align.

Для загрузки произвольного изображения в ImageView, требуется, в поле src указать путь, начинающийся с символа @ и далее указываете адрес по дереву. В папке res заблаговременно создать подкаталог drawable и загрузить логотип, а затем указать в коде программы путь ImageView. Непосредственно код приложения располагается по маршруту src → «ваше название package» → «название приложения».java.

По-умолчанию создается шаблон главного класса и от него уже дальше формируются другие модули. Логика работы объясняется тремя простыми шагами:

- загрузить виртуальную таблицу температуры;
- с помощью регулярных выражений загрузить температуру из таблицы температуры, расположенной в виртуальном модуле Zephyr Bioharness;
- показать в TextView значение температуры.

Когда приложению требуются какие-то системные разрешения, нужно указывать их в манифесте AndroidManifest.xml. В данном конкретном случае используем подключение к виртуальному устройству, соответственно добавляем строчку <uses-permission android:name=«android.permission.vdt»/> в указанный выше xml-файл.

Код приложения состоит из трех классов, основной из которых класс соединения с BLE устройством, реализующий указанную выше логику работы.

После того как приложение написано, можно его отлаживать и тестировать. Выбираем виртуальную машину и запускаем кнопкой меню Run → Run.

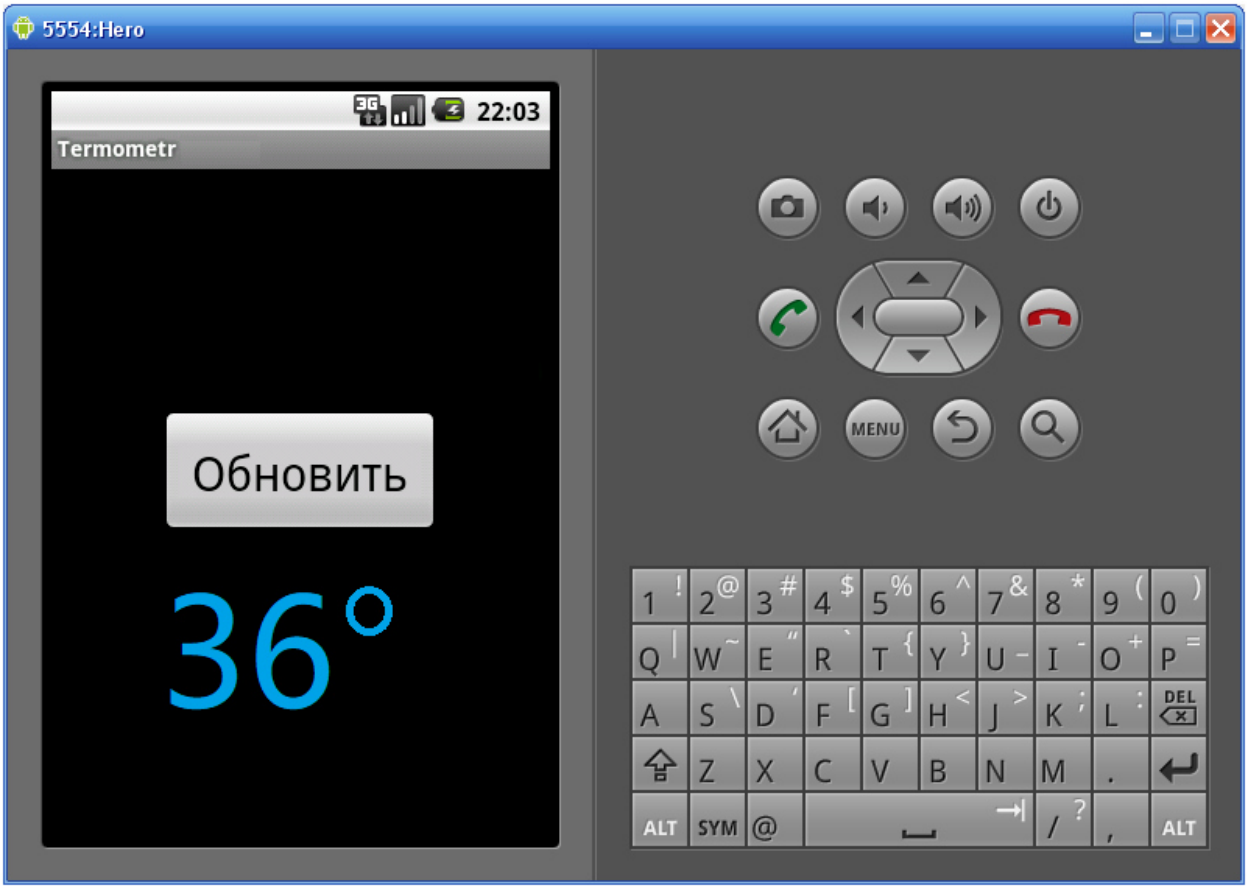


Рисунок 3.6 Результат тестирования приложения

4 Технико-экономическое обоснование разработки программного обеспечения дипломного проекта

4.1 Организация создания программного обеспечения

Целью данного дипломного проекта, является создание программных средств автоматизированной системы контроля успеваемости учащихся.

Программу, как любое техническое решение, необходимо рассматривать с точки зрения экономической целесообразности и пользы.

Программное изделие должно быть разработано так, чтобы оно выполняло свои функции без лишних затрат ресурсов (оперативной памяти ЭВМ, машинного времени, пропускной способности каналов передачи данных и др. - на стадии функционирования; время разработки и денежных ресурсов - на стадии использования программного изделия).

Содержание технико-экономического обоснования разработки программы заключается в следующем:

- расчет технико-экономических показателей;
- расчет затрат на разработку;

Приведенные в данном разделе дипломной работы данных позволяют сопоставить результаты разработки и затраты на нее, чтобы сделать вывод об эффективности проекта.

4.2 Расчет цены программного обеспечения

Расчет цены ПО содержит:

– определение затрат на оплату труда и накладных расходов;

– установление цены ПО;

установление нижнего предела цены и договорной цены с учетом тиражирования.

4.2.1 Определение затрат на разработку программного обеспечения

Затраты на разработку программного обеспечения определяются по формуле:

$$З_{р.по} = З_{р} + З_{д} + З_{с} + З_{н} + Р_{экспл}, \quad (4.1)$$

Где:

$З_р$ – заработная плата разработчиков, руб.;

$З_д$ – дополнительная заработная плата, руб.;

$З_с$ – отчисления на социальные нужды, руб.;

$З_н$ – накладные расходы, руб.;

$Р_{экспл}$ – эксплуатационные расходы, руб.

Оплата труда разработчика 8000 руб. в месяц, с учетом того, что в месяце 24 рабочих дня, получим 333 руб. в день. В общей сложности, разработка ПО велась в течение 40 рабочих дней. Таким образом, общая заработная плата за 40 дней разработки составит

$$З_р = 40 \cdot 333 = 13320 \ (руб.)$$

Дополнительная заработная плата определяется по формуле:

$$З_д = К_д \cdot З_р, \quad (4.2)$$

Где:

$К_д$ – коэффициент, учитывающий дополнительную заработную плату.

Примем $К_д$ равным 0,1, тогда получаем, что дополнительная заработная плата составляет:

$$З_д = 13320 \cdot 0{,}1 = 1332\ (руб.)$$

Расчет отчислений в страховые фонды с фонда заработной платы всех категорий промышленно-производственного персонала проводится в соответствии с установленным по действующему законодательству значением процента отчислений и определяются по формуле:

$$З_с = (З_р + З_д) \cdot 0{,}30 \qquad (4.3)$$

Учитывая ранее полученные значения заработной платы разработчика и дополнительной заработной платы, отчисления на социальные нужды составляют:

$$З_с = (13320 + 1332) \cdot 0{,}30 = 4395{,}6\ (руб.)$$

Накладные расходы определяются следующим образом:

$$З_н = \frac{З_р \cdot П}{100}, \qquad (4.4)$$

Где:

П – процент накладных расходов организации, в которой разрабатывается программа.

Примем, что П равен 45 %, получим накладные расходы в размере

$$\boldsymbol{З_н = \frac{13320 \cdot 45}{100} = 5994\ (руб.)}$$

Расчет стоимости одного часа эксплуатации ЭВМ производится по формуле:

$$Р_{экспл} = \frac{С_э + С_о + А + С_р}{Ф_г}, \quad (4.5)$$

Где:

$С_э$ – стоимость энергии, потребляемой компьютером за год, руб.;

$С_о$ – расходы на обслуживание компьютера в течение года, руб.;

А – амортизационные отчисления, руб.;

$С_р$ – расходы на текущий ремонт, руб.;

$Ф_г$ – годовой полезный фонд времени работы ЭВМ, ч.

Компьютер работает в течение 9 часов в сутки, пять дней в неделю, 249 дней в году. Годовой полезный фонд времени работы компьютера при таких условиях составляет

$$Ф_г = 9 \cdot 249 = 2241(ч.)$$

Стоимость потребляемой за год электроэнергии рассчитывается по формуле:

$$С_э = М_п \cdot Ф_г \cdot Ц_э, \quad (4.6)$$

Где:

$М_п$ – потребляемая мощность ЭВМ, кВт;

$Ц_э$ – цена 1 кВт*ч электроэнергии, руб.

Согласно последней тарифной ставке 1 кВт*ч электроэнергии стоит 3,44 руб. Потребляемая мощность компьютера 0,2 кВт. Затраты на электроэнергию составят

$$С_э = 0{,}2 \cdot 2241 \cdot 3{,}44 = 1541{,}81\ (руб.)$$

Расходы на обслуживание компьютера за год определяются по формуле:

$$С_о = \frac{ЗП_о \cdot 12}{Э_о}, \quad (4.7)$$

Где:

$ЗП_о$ – зарплата персонала, обслуживающего компьютеры, руб.;

$Э_о$ – количество обслуживаемых им единиц оборудования;

12 – количество месяцев в году.

В организации используется 5 компьютеров, обслуживанием которых занимается 1 человек. Его заработная плата составляет 5000 руб. в месяц. В соответствии с формулой (4.7), затраты на обслуживание ЭВМ составляют

$$С_о = \frac{5000 \cdot 12}{5} = 12000 \ (руб.)$$

Амортизационные отчисления определяются по формуле:

$$А = \frac{Н_а \cdot С}{100}, \qquad (4.8)$$

Где:

$Н_а$ – норма амортизации. Примем ускоренную, равную 50 %;

С – балансовая стоимость компьютера, руб.

Балансовая стоимость складывается из стоимости компьютера, стоимости транспортировки и стоимости установки. Для используемого компьютера балансовая стоимость составляет 15000 руб.

Таким образом, согласно формуле (4.8), амортизационные отчисления составят:

$$А = \frac{50 \cdot 15000}{100} = 7500 \ (руб.)$$

Расходы на текущий ремонт $С_р$ определяются как 5% от балансовой стоимости компьютера. Таким образом, получаем, что расходы составят:

$$С_р = 0{,}05 \cdot 15000 = 750 \ (руб.)$$

Подставляя вычисленные значения в формулу (4.5) получаем, что стоимость одного часа эксплуатации ЭВМ равна

$$Р_{экспл/ч} = \frac{1541,81 + 12000 + 7500 + 750}{2241} = 9,72 \ (руб.)$$

Из 40 дней, затраченных на разработку ПО, 28 дней приходится на кодирование и 12 дней на отладку ПО. Учитывая, что в день компьютер работает в течение 9 часов, получаем, что эксплуатационные расходы составят:

$$Р_{экспл} = (28 + 12) \cdot 9 \cdot 9,72 = 3499,2 \ (руб.)$$

Согласно формуле (3.1) затраты на разработку программного обеспечения составят:

$$З_{р.по} = 13320 + 1332 + 4395,6 + 5994 + 3499,2 = 28540,8$$
(*руб.)*

4.2.2 Установление цены программного обеспечения

Цена разработанной программы определяется по формуле:

$$\boldsymbol{Ц = З_{р.по} \cdot (1 + Р),} \qquad (4.9)$$

Где:

Р – норматив рентабельности, позволяющий учитывать прибыль организации, разрабатывающей данную программу.

Примем норматив рентабельности равным 20%, тогда цена разрабатываемого программного продукта:

$$Ц = 28540,8 \cdot 1,2 = 34248,96_{(руб.)}$$

Итак, цена программного продукта, полученного в ходе дипломного проектирования, составляет 34248,96 руб. при норме рентабельности 20%. Себестоимость программы – 28540,8 руб.

4.2.3 Установление нижнего предела цены с учетом тиражирования

Нижний предел цены (Цнп) с учетом желаемой рентабельности (Р), тиражированием (N) и НДС (Н) определяется по формуле:

$$Ц_{НП} = \frac{З_{р.по} \cdot (1+Р)}{N} \cdot (1+Н). \qquad (4.10)$$

При установленном НДС равным 20%, и учитывая, что тиражирование программы не предполагается, рассчитываем предел нижней цены.

$$Ц_{НП} = \frac{34248{,}96 \cdot (1+0{,}20)}{1} \cdot (1+0{,}20) = 49318{,}5 \quad (руб.)$$

Окончательную цену разрабатываемого ПО примем на уровне 49318,5 рублей.

5. Безопасность жизнедеятельности

5.1 Организация безопасных условий труда на рабочем месте оператора ПЭВМ

Рабочее место - это система функционально и пространственно-организованных технических средств и предмета труда, обеспечивающая условия для успешного решения человеком - оператором поставленной перед ним задачи.

На комфортность работы оператора влияет организация рабочего места оператора, средства отображения и органы управления машиной. Они должны быть максимально удобны для человека, чтобы не создавать помех, дискомфорта в процессе работы, а также способствовать наименьшей утомляемости.

Основным способом обеспечения условий комфорта оператора ПЭВМ является рациональная организация его рабочего места.

Под рабочим местом оператора ПЭВМ понимается зона трудовой деятельности в системе «человек-машина» с техническими средствами и вспомогательным оборудованием, необходимым для решения конкретных производственных задач.

Конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учетом его количества и конструктивных особенностей, характера выполняемой работы. При этом допускается использование рабочих столов различных конструкций, отвечающих современным требованиям эргономики. Поверхность рабочего стола должна иметь коэффициент отражения 0,5 - 0,7. [1]

Рекомендуемые расположения текста, клавиатуры и экрана компьютера R=50..60 см для типовых компьютеров с размером экрана по диагонали 250..350 мм (предельно минимальное) представлены на рисунке 5.1.

5.2 Расчет освещенности на рабочем месте

Рациональное освещение помещений - один из наиболее важных факторов, от которых зависит эффективность трудовой деятельности человека. Хорошее освещение необходимо для выполнения большинства задач пользователя ЭВМ. Для того чтобы спланировать рациональную систему освещения, учитывается специфика рабочего задания, для которого создается система освещения, скорость и точность, с которой это рабочее задание должно выполняться, длительность его выполнения и различные изменения в условиях выполнения рабочих операций. [4]

Помещение, в котором находится рабочее место пользователя ЭВМ, имеет следующие характеристики:

- длина помещения 16 м;
- ширина помещения 6 м;
- высота 4 м;
- число окон 3;
- количество рабочих мест 3;
- окраска интерьера - белый потолок, бледно-зеленые стены;
- пол металлический, обтянутый линолеумом зеленого цвета.

В помещении, где находится рабочее место пользователя, используется смешанное освещение, т.е. сочетание естественного и искусственного освещения. В качестве естественного - боковое освещение через окна. Искусственное освещение используется при недостаточном естественном освещении. В данном помещении используется общее искусственное

освещение. Расчет его осуществляется по методу светового потока с учетом потока, отраженного от стен и потолка.

Нормами для данных работ установлена необходимая освещенность рабочего места Ен = 300 [лк].

Общий световой поток определяется по формуле:

$$F_{общ} = \frac{Ен \times S \times z_1 \times z_2}{V \cdot N}$$ [лм].

Где:

Ен - нормированная освещенность (Ен =300 [лк]);

S- площадь помещения;

z1 - коэффициент, учитывающий старение ламп и загрязнение светильников (z1=5).

z2 - коэффициент, учитывающий неравномерность освещения помещения (z2=1,1);

N – количество ламп в светильнике;

V- коэффициент использования светового потока, определяется в зависимости от коэффициентов отражения от стен, потолка, рабочих поверхностей, типов светильников и геометрии помещения.

Площадь помещения:

S=A*B=16*6=96 [м2].

Выберем коэффициент использования светового потока по следующим данным:

– коэффициент отражения побеленного потолка Rn = 70%;

– коэффициент отражения от стен, окрашенных в светлую краску Rcm=50%;

– коэффициент отражения от пола, покрытого линолеумом темного цвета Rp=10%;

– индекс помещения:

$$i = \frac{A * B}{h * (A + B)} = \frac{16 * 6}{4 * (16 * 6)} = 1.1$$

Найденный коэффициент V= 0,34 (для светильника ПВЛ-1). Отсюда определяется общий световой поток:

$$F_{общ} = \frac{300 * 96 * 1{,}1 * 1{,}5}{0{,}34 \cdot 2} = 69882$$ [лм].

Для организации общего искусственного освещения выберем лампы типа ЛБ40. Люминесцентные лампы имеют ряд преимуществ перед лампами накаливания: их спектр ближе к естественному; они имеют большую экономичность (больше светоотдача) и срок службы (в 10-12 раз). Наряду с этим имеются и недостатки: их работа сопровождается иногда шумом, хуже работают при низких температурах, их нельзя применять во взрывоопасных помещениях; имеют малую инерционность. Для нашего помещения люминесцентные лампы подходят. Световой поток одной лампы ЛБ40 составляет не менее Fn =3000 [лм]. Число ламп N, необходимых для организации общего освещения определяется по формуле:

$$N = \frac{F_{общ}}{F_{л}} = \frac{69882}{3000} = 24$$

В качестве светильников выбираем ПВЛ-1, 2х40 Вт. Таким образом, чтобы обеспечить световой поток Fобщ = 69882 [лм] надо использовать 12 светильников по 2 лампы ЛБ40 в каждом.

Рабочая высота подвеса определяется выражением:

$h = H - h_1 = 4 - 0.7 - 0.1 = 3.2$ [м].

Расстояние между центрами светильников:

$L = 1{,}4 \cdot h = 1{,}4 \cdot 3.2 = 4{,}48$ [м].

Число рядов светильников рассчитывается исходя из ширины помещения и расстояния между рядами:

$$n = \frac{B}{L} = \frac{6}{4.48} = 1.34 \approx 2$$

Размещаем светильники в два ряда.

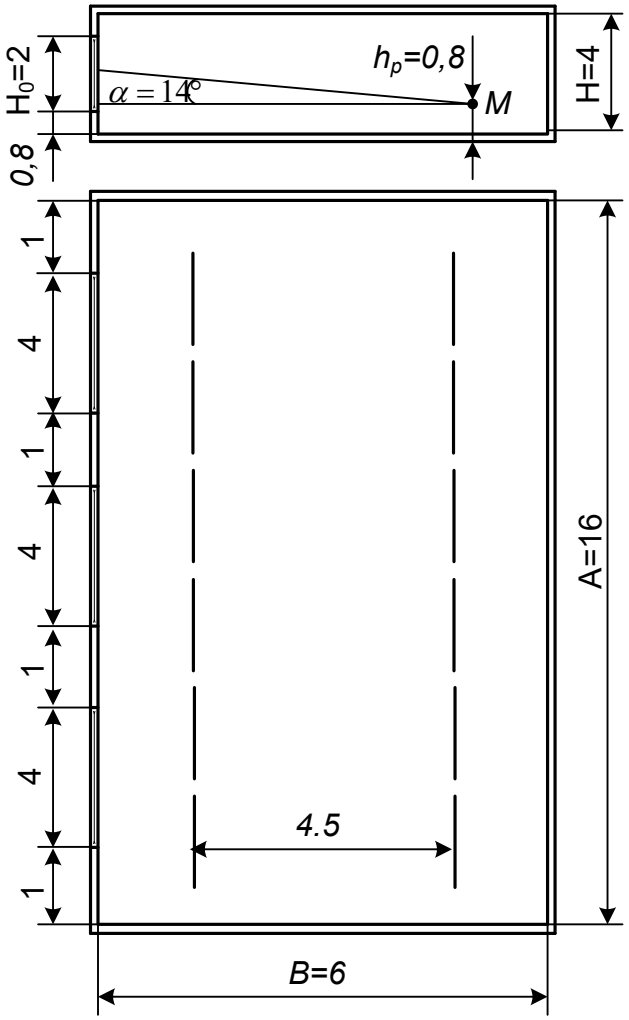


Рисунок 5.1 – Эскиз помещения с расположением светильников

5.3 Параметры микроклимата

Параметры микроклимата могут меняться в широких пределах, в то время как необходимым условием жизнедеятельности человека является поддержание постоянства температуры тела благодаря терморегуляции, т.е. способности организма регулировать отдачу тепла в окружающую среду. Принцип нормирования микроклимата – создание оптимальных условий для теплообмена тела человека с окружающей средой.

Вычислительная техника является источником существенных тепловыделений, что может привести к повышению температуры и снижению относительной влажности в помещении. В помещениях, где установлены компьютеры, должны соблюдаться определенные параметры микроклимата. В санитарных нормах установлены величины параметров микроклимата, создающие комфортные условия. Эти нормы устанавливаются в зависимости от времени года, характера трудового процесса и характера производственного помещения (см. табл. 5.1).

Объем помещений, в которых размещены работники вычислительных центров, не должен быть меньше 19,5м3/человека с учетом максимального числа одновременно работающих в смену. Нормы подачи свежего воздуха в помещения, где расположены компьютеры, приведены в табл. 5.2.

Таблица 5.1 – Параметры микроклимата для помещений, где установлены компьютеры.

Период года	Параметр микроклимата	Величина
Холодный	Температура воздуха в помещении	22…24°С
	Относительная влажность	40…60%
	Скорость движения воздуха	до 0,1м/с

Продолжение таблицы 5.1

Теплый	Температура воздуха в помещении	23…25°С
	Относительная влажность	40…60%
	Скорость движения воздуха	0,1…0,2м/с

Таблица 5.2 – Нормы подачи свежего воздуха в помещения, где расположены компьютеры.

Характеристика помещения	Объемный расход подаваемого в помещение свежего воздуха, м3 /на одного человека в час
Объем до 20м3 на человека	Не менее 30
20…40м3 на человека	Не менее 20
Более 40м3 на человека	Естественная вентиляция

Для обеспечения комфортных условий используются как организационные методы (рациональная организация проведения работ в зависимости от времени года и суток, чередование труда и отдыха), так и технические средства (вентиляция, кондиционирование воздуха, отопительная система).

5.4 Шум и вибрация

Шум ухудшает условия труда оказывая вредное действие на организм человека. Работающие в условиях длительного шумового воздействия испытывают раздражительность, головные боли, головокружение, снижение

памяти, повышенную утомляемость, понижение аппетита, боли в ушах и т.д. Такие нарушения в работе ряда органов и систем организма человека могут вызвать негативные изменения в эмоциональном состоянии человека вплоть до стрессовых. Под воздействием шума снижается концентрация внимания, нарушаются физиологические функции, появляется усталость в связи с повышенными энергетическими затратами и нервно-психическим напряжением, ухудшается речевая коммутация. Все это снижает работоспособность человека и его производительность, качество и безопасность труда. Длительное воздействие интенсивного шума (выше 80 дБ) на слух человека приводит к его частичной или полной потере. [3]

В табл. 5.3 указаны предельные уровни звука в зависимости от категории тяжести и напряженности труда, являющиеся безопасными в отношении сохранения здоровья и работоспособности.

Таблица 5.3 – Предельные уровни звука, дБ, на рабочих местах:

Категория напряженности труда	Категория тяжести труда			
	Легкая	Средняя	Тяжелая	Очень тяжелая
Мало напряженный	80	80	75	75
Умеренно напряженный	70	70	65	65
Напряженный	60	60		
Очень напряженный	50	50		

В результате вычислений и технологий БЖД было выполнено:

– организация рабочего места оператора ЭВМ с учетом требований безопасности;

– из расчета следует, что для нормальной работы пользователя рабочего места с видеотерминальным устройством необходимо общее освещение помещения со световым потоком 69882 [лм], для чего необходимо наличие 12 светильников типа ПВЛ-1 с 2-мя лампами типа ЛБ40 каждый. Светильники должны быть расположены в два ряда, что является наиболее оптимальным решением. Кроме того, рекомендуется использовать ряд специальных мер по защите пользователя от вредных факторов в виде бликов на экране монитора, например, использование занавесей на окнах;

– соблюдение условий, определяющих оптимальную организацию рабочего места инженера - программиста, позволит сохранить хорошую работоспособность в течение всего рабочего дня. Повысит, как в количественном, так и в качественном отношениях производительность труда программиста, что в свою очередь будет способствовать быстрейшей разработке и отладке программного продукта.

Заключение

В ходе выполнения дипломной работы было разработано мобильное приложение, обеспечивающее обработку телеметрических данных полученных с оконечного устройства. Данный проект это приложение на основе клиент серверного подхода, созданное на основе телеметрических данных собранных пользователем с пульсометров или иных устройств. Приложение интегрируется со сторонними службами для обеспечения безопасности работы с данными и конвертирования полученной информации на сервер визуализации данных. Приложение построено с использованием технологий, которые являются производительными, гибкими и масштабируемыми.

Список использованных источников

1. Голощапов Алексей Google Android. Программирование для мобильных устройств; БХВ-Петербург - Москва, 2017. - 448 с.

2. Дэрси Лорен , Кондер Шейн Android за 24 часа. Программирование приложений под операционную систему Google; Рид Групп - Москва, 2020. - 464 с.

3. Колисниченко Денис Программирование для Android. Самоучитель; БХВ-Петербург - Москва, 2016. - 272 с.

4. Майер Рето Android 2. Программирование приложений для планшетных компьютеров и смартфонов; Эксмо - Москва, 2015. - 672 с.

5. Майер Рето Android 4. Программирование приложений для планшетных компьютеров и смартфонов; Эксмо - Москва, 2017. - 816 с.

6. Медникс Зигард, Дорнин Лайрд, Мик Блэйк , Накамура Масуми Программирование под Android; Питер - Москва, 2018. - 560 с.

7. Немцова Т. И., Голова С. Ю., Абрамова И. В. Программирование Bluetooth под Android; Форум, Инфра-М - Москва, 2019. - 496 с.

8. Ретабоуил Сильвен Android NDK. Разработка приложений под Android на Java; ДМК Пресс - Москва, 2018. - 496 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы

AndroidManifest.xml

```
    <?xml version="1.0" encoding="utf-8"?>
    <manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.project.bluetooth.monitoring"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
/>
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"
    />
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission
android:name="android.permission.UPDATE_DEVICE_STATS"
    />
    <uses-permission
android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-sdk
    android:minSdkVersion="11"
    android:targetSdkVersion="15" />
    <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <!-- com.project.bluetooth.addons -->
    <activity
    android:name="com.project.bluetooth.addons.AboutActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name"
    android:screenOrientation="portrait"
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
    <activity
    android:name="com.project.bluetooth.addons.SplashActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
    <activity
    android:name="com.project.bluetooth.addons.MainActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar" />
    <activity
    android:name="com.project.bluetooth.addons.Help"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name" />
    <activity
    android:name="com.project.bluetooth.addons.HelpTopics"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name"
```

```
        android:theme="@android:style/Theme.Light" />
        <activity
        android:name="com.project.bluetooth.addons.Configuration"
        android:enabled="true"
        android:label="Settings"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <!-- com.project.bluetooth.drawing -->
        <activity
        android:name="com.project.bluetooth.drawing.DataToDrawECG"
        android:label="@string/app_name"
        android:screenOrientation="portrait"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <activity
    android:name="com.project.bluetooth.drawing.DataToDrawAccelerometer"
        android:label="@string/app_name"
        android:screenOrientation="portrait"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <activity
        android:name="com.project.bluetooth.drawing.DataToDrawRespiration"
        android:label="@string/app_name"
        android:screenOrientation="portrait"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <!-- com.project.bluetooth.login -->
        <activity
        android:name="com.project.bluetooth.login.PatientLogin"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >
        <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER"
        />
        </intent-filter>
        </activity>
        <activity
        android:name="com.project.bluetooth.login.PatientRegister"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
        <!-- com.project.bluetooth.monitoring -->
        <activity

    android:name="com.project.bluetooth.monitoring.BluetoothMonitoring"
        android:configChanges="orientation|keyboardHidden"
        android:screenOrientation="portrait" />
        <activity
        android:name="com.project.bluetooth.monitoring.DeviceListActivity"
        android:configChanges="orientation|keyboardHidden"
        android:label="@string/select_device"
        android:theme="@android:style/Theme.Holo.Dialog" />
        <service

    android:name="com.project.bluetooth.monitoring.BluetoothRfcommService"
        android:exported="false" />
        <!-- com.project.bluetooth.storage -->
        <activity
        android:name="com.project.bluetooth.storage.SQLView"
        android:label="@string/app_name" />
        <!-- com.project.bluetooth.upload -->
```

```
    <activity
    android:name="com.project.bluetooth.upload.AllProductsActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name" />
    <activity
    android:name="com.project.bluetooth.upload.NewProductActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name" />
    <activity
    android:name="com.project.bluetooth.upload.EditProductActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name" />
    <activity

android:name="com.project.bluetooth.upload.AndroidDetectInternetConnec
tionActivity"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/app_name" />
    </application>
</manifest>
```

BluetoothMonitoring.java

```
/*
    package com.project.bluetooth.monitoring;
    import java.io.BufferedReader;
    import java.io.File;
    import java.io.FileNotFoundException;
    import java.io.FileReader;
    import java.text.SimpleDateFormat;
    import java.util.Date;
    import com.project.bluetooth.addons.Configuration;
    import com.project.bluetooth.drawing.*;
    import
com.project.bluetooth.monitoring.BluetoothRfcommService.LocalBinder;
    import com.project.bluetooth.storage.DatabaseHelperClass;
    import android.annotation.SuppressLint;
    import android.app.Activity;
    import android.app.AlertDialog;
    import android.app.Dialog;
    import android.bluetooth.BluetoothAdapter;
    import android.bluetooth.BluetoothDevice;
    import android.content.ActivityNotFoundException;
    import android.content.BroadcastReceiver;
    import android.content.ComponentName;
    import android.content.Context;
    import android.content.DialogInterface;
    import android.content.Intent;
    import android.content.IntentFilter;
    import android.content.ServiceConnection;
    import android.content.SharedPreferences;
    import
android.content.SharedPreferences.OnSharedPreferenceChangeListener;
    import android.net.Uri;
    import android.os.Bundle;
    import android.os.Environment;
    import android.os.Handler;
    import android.os.IBinder;
```

```
import android.os.Message;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.WindowManager.LayoutParams;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;
public class BluetoothMonitoring extends Activity implements
OnClickListener, OnSharedPreferenceChangeListener {
    private static final String TAG = "BluetoothMonitoring";
    private static final boolean D = true;
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
    public static final int MESSAGE_DEVICE_NAME = 4;
    public static final int MESSAGE_TOAST = 5;
    public static final int MESSAGE_GET_DATA = 6;

    public static final String DEVICE_NAME = "device_name";
    public static final String TOAST = "toast";

    private static final int REQUEST_CONNECT_DEVICE = 1;
    private static final int REQUEST_ENABLE_BT = 2;

    private String mConnectedDeviceName = null;
    private BluetoothAdapter mBluetoothAdapter = null;
    private BluetoothRfcommService mRfcommClient = null;
    private Button button_call;
    private Button button_nextview;
    private ImageButton imageButton_position;
    private TextView panel_heart_rate;
    private TextView panel_respiration_rate;
    private TextView panel_temperature_c;
    private TextView panel_bottom1;
    private TextView panel_bottom2;
    private TextView panel_bottom3;
    private TextView panel_bottom4;
    private double maximumBR = 50;
    private double minimumBR = 8;
    private double maximumHR = 180;
    private double minimumHR = 40;
    private double maximumT = 40;
    private double minimumT = 34;
    boolean mExternalStorageAvailable = false;
    boolean mExternalStorageWriteable = false;
    String state = Environment.getExternalStorageState();
    File file;
    FileReader filereader;
    BufferedReader bufferedreader_ECG1;
    String s = new String();
    String p = new String();
```

```
        static int test = 0;
        private boolean mBound = false;

        private SharedPreferences preferences = null;
        String dataMessage;
        public String bluedata;
        private String position;
        private String heartRate;
        private String respirationRate;
        private String temperature;
        private Button button_ind_1;
        private Button button_ind_2;
        private Button button_ind_3;
        private Button button_ind_4;
        private Button button_ind_5;
        private Button button_ind_7;
        private Button button_ind_6;
        private Button button_ind_8;
        private Button button_ind_9;
        private Button button_ind_10;
        private Button button_ind_11;
        private Button button_ind_12;
        private Button button_ecg_go;
        private Button button_resp_go;
        private Button button_acc_go;

        @SuppressLint("NewApi")
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            if (D)
                Log.e(TAG, "+++ ON CREATE +++");
            preferences =
PreferenceManager.getDefaultSharedPreferences(this);

preferences.registerOnSharedPreferenceChangeListener(this);
            setContentView(R.layout.monitoring_main);
            Context.BIND_AUTO_CREATE);
            getWindow().addFlags(LayoutParams.FLAG_KEEP_SCREEN_ON);
            setupUI();
            mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
            if (mBluetoothAdapter == null) {
                Toast.makeText(this, "Bluetooth is not available",
                Toast.LENGTH_LONG).show();
                finish();
                return;
            }

            if (Environment.MEDIA_MOUNTED.equals(state)) {
                    mExternalStorageAvailable =
mExternalStorageWriteable = true;
            } else if
(Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {

            mExternalStorageAvailable = true;
            mExternalStorageWriteable = false;
            } else {
```

```
                    mExternalStorageAvailable =
mExternalStorageWriteable = false;
            }

            try {
                    file = new
File(Environment.getExternalStorageDirectory() + "/project/data.txt");
                    filereader = new FileReader(file);
                    bufferedreader_ECG1 = new
BufferedReader(filereader);
            } catch (FileNotFoundException e) {
                    e.printStackTrace();
      }
}

public void onClick(final View v) {
    int[] array = { test };
    if (v == button_call) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage("Are you sure you want to do an emergency
call?").setCancelable(false)
        builder.setPositiveButton("Yes", new
DialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int id) {
            p = preferences.getString("phonenumber", "");
            if (p.equals("")) {
                Toast toast = Toast.makeText(
                BluetoothMonitoring.this, "Please enter a phone
number", Toast.LENGTH_LONG);
                toast.show();
                Intent i = new Intent(v.getContext(),
Configuration.class);
                startActivity(i);
       }
       emergencyCall();
    }
}

    .setNegativeButton("No", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
    }
};

            builder.show();


        }
        if (v == button_nextview) {
            Intent i = new Intent(this, DataToDrawECG.class);
            Bundle e = new Bundle();
            e.putString("key1", s);
            e.putIntArray("key2", array);
            i = i.putExtras(e);
            startActivity(i);
        }
```

```
        if (v == button_resp_go) {
            Intent i = new Intent(this, DataToDrawRespiration.class);
            Bundle e = new Bundle();
            e.putString("key1", s);
            e.putIntArray("key2", array);
            i = i.putExtras(e);
            startActivity(i);
        }
        if (v == button_acc_go) {
            Intent i = new Intent(this,
DataToDrawAccelerometer.class);
            Bundle e = new Bundle();
            e.putString("key1", s);
            e.putIntArray("key2", array);
            i = i.putExtras(e);
            startActivity(i);
        }
        if (v == button_ecg_go) {
            Intent i = new Intent(this, DataToDrawECG.class);
            Bundle e = new Bundle();
            e.putString("key1", s);
            e.putIntArray("key2", array);
            i = i.putExtras(e);
            startActivity(i);
        }
public void emergencyCall() {
    try {
            Intent intent = new Intent();
            intent.setAction(Intent.ACTION_CALL);
            intent.setData(Uri.parse("tel:" + p));
            startActivity(intent);
    } catch (ActivityNotFoundException e) {
        Log.e("Error: ", "failed to make a call. Please check Call
settings in Configuration menu", e);

}
}
@Override
public void onStart() {
    super.onStart();
    if (D)
        Log.e(TAG, "++ ON START ++");
    myReceiver = new MyReceiver();

    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(BluetoothRfcommService.MY_ACTION);
    registerReceiver(myReceiver, intentFilter);


    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableIntent = new Intent(
        BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
    } else {
    if (!mBound)
        setupMonitoring();
    }
```

```
}
@Override
public synchronized void onResume() {
    super.onResume();
    if (D)
    Log.e(TAG, "+ ON RESUME +");
    //
    //
    if (mRfcommClient != null) {
    if (mRfcommClient.getState() == BluetoothRfcommService.STATE_NONE)
{
        mRfcommClient.start();
    }
}

private void setupMonitoring() {
    Log.d(TAG, "setupMonitoring()");
    if (preferences.getBoolean("syncEnabled", false) && !mBound) {
        Intent intent = new Intent(this,
BluetoothRfcommService.class);
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
    }
}
@Override
public synchronized void onPause() {
    super.onPause();
    if (D)
    Log.e(TAG, "- ON PAUSE -");
}

@Override
public void onStop() {
    unregisterReceiver(myReceiver);
    super.onStop();
    if (D)
    Log.e(TAG, "-- ON STOP --");
    if (mBound) {
        unbindService(mConnection);
        mBound = false;
    }
}

@Override
public void onDestroy() {

    super.onDestroy();
    if (D)
        Log.e(TAG, "--- ON DESTROY ---");
    if (mBound) {
        unbindService(mConnection);
    mBound = false;
}


@SuppressWarnings("unused")
private void sendMessage(String Message) {
```

```
        if (mRfcommClient.getState() !=
    BluetoothRfcommService.STATE_CONNECTED) {
        Toast.makeText(this, R.string.not_connected,
    Toast.LENGTH_SHORT).show();
        return;
        }

        if (Message.length() > 0) {
            byte[] send = Message.getBytes();
            mRfcommClient.write(send);
        }
    }

    private void setupUI() {
        panel_heart_rate = (TextView) findViewById(R.id.heart_bpm);
        panel_heart_rate.setText(this.getString(R.string.test3));
        panel_respiration_rate = (TextView) findViewById(R.id.
        respiration_bpm);
        panel_respiration_rate.setText(this.getString(R.string.test2));
        panel_temperature_c = (TextView) findViewById(R.id.temperature_C);
        panel_temperature_c.setText(this.getString(R.string.test1));
        panel_bottom1 = (TextView) findViewById(R.id.testtext1);
        panel_bottom1.setText(this.getString(R.string.bottom_test1));
        panel_bottom2 = (TextView) findViewById(R.id.testtext2);
        panel_bottom2.setText(this.getString(R.string.bottom_test2));
        panel_bottom3 = (TextView) findViewById(R.id.testtext3);
        panel_bottom3.setText(this.getString(R.string.bottom_test3));
        panel_bottom4 = (TextView) findViewById(R.id.testtext4);
        panel_bottom4.setText(this.getString(R.string.bottom_test4));
        button_nextview = (Button) findViewById(R.id.nextbutton);
        button_nextview.setOnClickListener(this);
        button_call = (Button) findViewById(R.id.button_call_emergency);
        button_call.setOnClickListener(this);
        imageButton_position = (ImageButton) findViewById(R.id.
        imageButton_body_position);
        imageButton_position.setOnClickListener(this);
        button_ind_1 = (Button) findViewById(R.id.indicatorbutton1);
        button_ind_2 = (Button) findViewById(R.id.indicatorbutton2);
        button_ind_3 = (Button) findViewById(R.id.indicatorbutton3);
        button_ind_4 = (Button) findViewById(R.id.indicatorbutton4);
        button_ind_5 = (Button) findViewById(R.id.indicatorbutton5);
        button_ind_6 = (Button) findViewById(R.id.indicatorbutton6);
        button_ind_7 = (Button) findViewById(R.id.indicatorbutton7);
        button_ind_8 = (Button) findViewById(R.id.indicatorbutton8);
        button_ind_9 = (Button) findViewById(R.id.indicatorbutton9);
        button_ind_10 = (Button) findViewById(R.id.indicatorbutton10);
        button_ind_11 = (Button) findViewById(R.id.indicatorbutton11);
        button_ind_12 = (Button) findViewById(R.id.indicatorbutton12);
        button_ecg_go = (Button) findViewById(R.id.button_ecg);
        button_ecg_go.setOnClickListener(this);
        button_resp_go = (Button) findViewById(R.id.button_resp);
        button_resp_go.setOnClickListener(this);
        button_acc_go = (Button) findViewById(R.id.button_acc);
        button_acc_go.setOnClickListener(this);
    }

    private final Handler mHandler = new Handler() {
```

```
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case MESSAGE_WRITE:
            break;
            case MESSAGE_READ:
                byte[] readBuf = (byte[]) msg.obj;
                String readMessage = new String(readBuf, 0, msg.arg1);
                displayTemperature(readMessage);
                displayRespirationRate(readMessage);
                displayHeartRate(readMessage);
                displayPosition(readMessage);
                displayMisc(readMessage);
            break;
            case MESSAGE_DEVICE_NAME:
                mConnectedDeviceName =
msg.getData().getString(DEVICE_NAME);
                Toast.makeText(getApplicationContext(), "Connected to
" + mConnectedDeviceName,
                    Toast.LENGTH_SHORT).show();
                break;
            case MESSAGE_TOAST:
                Toast.makeText(getApplicationContext(),
                msg.getData().getString(TOAST),
Toast.LENGTH_SHORT).show();
            break;
            case MESSAGE_GET_DATA:
                test = msg.arg1;
            break;
        }
};

private void displayPosition(String Message) {
    for (int i = 0; i < Message.length() - 3; i++) {
        if (Message.charAt(i) == 'n') {
        char msg1 = Message.charAt(i + 1);
        char msg2 = Message.charAt(i + 2);
        byte bmsg1 = (byte) (msg1 - 0x20);
        byte bmsg2 = (byte) (msg2 - 0x20);
        int sensorValue = (0x000000ff & bmsg1)
        | ((0x000000ff & bmsg2) << 6);
        positionTransfer(sensorValue);

        switch (sensorValue) {
            case 18:// upright
            //

setCompoundDrawablesWithIntrinsicBounds(orientation_upright_img,
                imageButton_position
                .setImageResource(R.drawable.position_upright);
            break;

            case 19:// prone
                imageButton_position
                    .setImageResource(R.drawable.position_prone);
            break;
```

```
                case 20:// supine
                    imageButton_position
                        .setImageResource(R.drawable.position_supine);
                break;

                case 21:// inverted
                    imageButton_position
                        .setImageResource(R.drawable.position_inverted);
                break;

                case 22:// side
                    imageButton_position
                        .setImageResource(R.drawable.position_side);
                break;
                case 23:// unknown
                    //

imageButton_position.setImageResource(R.drawable.position_unknown);
                break;

                case 152:// falling

imageButton_position.setImageResource(R.drawable.position_falling);
                break;

                case 165:// moving (slowly)

imageButton_position.setImageResource(R.drawable.position_moving);
                break;
}


protected void displayMisc(String readMessage) {
    for (int i = 0; i < readMessage.length() - 3; i++) {
        if (readMessage.charAt(i) == 'n') {
                char msg1 = readMessage.charAt(i + 1);
                char msg2 = readMessage.charAt(i + 2);
                byte bmsg1 = (byte) (msg1 - 0x20);
                byte bmsg2 = (byte) (msg2 - 0x20);
                int sensorValue = (0x000000ff & bmsg1)
                | ((0x000000ff & bmsg2) << 6);

                switch (sensorValue) {
                    case 14:// battery low

panel_bottom1.setBackgroundDrawable(getResources().getDrawable
                            (R.drawable.frame_light_black_outline_red));
                    break;
                    case 15:// battery ok
                        panel_bottom1.setBackgroundDrawable(getResources()
                            .getDrawable(R.drawable.
                            frame_light_black_outline));
                    break;
                    case 24:// Apnea
                        panel_bottom2.setBackgroundDrawable(getResources()

.getDrawable(R.drawable.frame_light_black_outline_red));
```

```
                    break;

                    case 27:// vital sign red
                        panel_bottom3.setBackgroundDrawable(getResources()

.getDrawable(R.drawable.frame_light_black_outline_red));
                    break;

                    case 183:// irregular rythm
                        panel_bottom4.setBackgroundDrawable(getResources()

.getDrawable(R.drawable.frame_light_black_outline_red));
                    break;
                }
            }
        }
    }

    private String positionTransfer(int sensorValue) {
        position = String.valueOf(sensorValue);
        return position;
    }

    private void displayHeartRate(String Message) {
        for (int i = 0; i < Message.length() - 3; i++) {
            if (Message.charAt(i) == 'h') {
                char msg1 = Message.charAt(i + 1);
                char msg2 = Message.charAt(i + 2);
                byte bmsg1 = (byte) (msg1 - 0x20);
                byte bmsg2 = (byte) (msg2 - 0x20);
                double sensorValue = (0x000000ff & bmsg1)
                | ((0x000000ff & bmsg2) << 6);
                double formattedValue = (sensorValue * 0.1);
                int endValue = (int) Math.round(formattedValue);
                heartRateTransfer(endValue);
                panel_heart_rate.setText("HR(bpm):" + endValue);
                if (endValue > maximumHR) {
                    panel_heart_rate.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.
                    frame_light_black_outline_red));
                    button_ind_1.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_green));
                    button_ind_2.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_green));
                    button_ind_3.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_yellow));
                    button_ind_4.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_red));
                } else {
                if (endValue < minimumHR) {
                    panel_heart_rate.setBackgroundDrawable(getResources()

.getDrawable(R.drawable.frame_light_black_outline_red));
                    button_ind_1.setBackgroundDrawable(getResources()
                        .getDrawable(R.drawable.button_red));
                } else {
                if (endValue > 120) {
```

```
                button_ind_1.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_green));
                button_ind_2.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_green));
                button_ind_3.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_yellow));
            } else {
                panel_heart_rate
                    .setBackgroundDrawable(getResources()

.getDrawable(R.drawable.frame_light_black_outline));
                button_ind_1.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_green));
                button_ind_2.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_green));
            }
        }
    }
}

private String heartRateTransfer(int endValue) {
    heartRate = String.valueOf(endValue);
    return heartRate;
}

private void displayRespirationRate(String Message) {
    for (int i = 0; i < Message.length() - 3; i++) {
        if (Message.charAt(i) == 'u') {
            char msg1 = Message.charAt(i + 1);
            char msg2 = Message.charAt(i + 2);
            byte bmsg1 = (byte) (msg1 - 0x20);
            byte bmsg2 = (byte) (msg2 - 0x20);
            double sensorValue = (0x000000ff & bmsg1)
            | ((0x000000ff & bmsg2) << 6);
double formattedValue = (sensorValue * 0.1);
            int endValue = (int) Math.round(formattedValue);
            respirationRateTransfer(endValue);
            panel_respiration_rate.setText("BR(bpm):" + endValue);

            if (endValue > maximumBR) {
                panel_respiration_rate.setBackgroundDrawable(
                getResources().getDrawable(R.drawable.
                    frame_light_black_outline_red));
                button_ind_5.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
                button_ind_6.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
                button_ind_7.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_yellow));
                button_ind_8.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_red));
            } else {
            if (endValue < minimumBR) {
                    panel_respiration_rate
                    .setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.
                    frame_light_black_outline_red));
```

```
                    button_ind_5.setBackgroundDrawable(getResources()
                    .getDrawable(R.drawable.button_green));
            } else {
            if (endValue > 30) {
                button_ind_5.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
                button_ind_6.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
                button_ind_7.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_yellow));
            } else {
                panel_respiration_rate
                .setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.frame_light_black_outline));
                button_ind_5.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
                button_ind_6.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
            }
        }
    }
}

private String respirationRateTransfer(int endValue) {
    respirationRate = String.valueOf(endValue);
    return respirationRate;
}

private void displayTemperature(String Message) {
    for (int i = 0; i < Message.length() - 3; i++) {
        if (Message.charAt(i) == 't') {
            char msg1 = Message.charAt(i + 1);
            char msg2 = Message.charAt(i + 2);
            byte bmsg1 = (byte) (msg1 - 0x20);
            byte bmsg2 = (byte) (msg2 - 0x20);
            double sensorValue = (0x000000ff & bmsg1)
            | ((0x000000ff & bmsg2) << 6);
            double formattedValue = (sensorValue * 0.1);
            int endValue = (int) Math.round(formattedValue);
            temperatureTransfer(endValue);
            panel_temperature_c.setText("Temp(C°):" + endValue);
        if (endValue > maximumT) {
            panel_temperature_c.setBackgroundDrawable(getResources()
            .getDrawable(R.drawable.frame_light_black_outline_red));
            button_ind_9.setBackgroundDrawable(getResources()
            .getDrawable(R.drawable.button_green));
            button_ind_10.setBackgroundDrawable(getResources()
            .getDrawable(R.drawable.button_green));
            button_ind_11.setBackgroundDrawable(getResources()
            .getDrawable(R.drawable.button_yellow));
            button_ind_12.setBackgroundDrawable(getResources()
            .getDrawable(R.drawable.button_red));
        } else {
            if (endValue < minimumT) {
                panel_respiration_rate
                .setBackgroundDrawable(getResources()
```

```
.getDrawable(R.drawable.frame_light_black_outline_red));
                button_ind_9.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
        } else {
            if (endValue > 30) {
                button_ind_9.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
                button_ind_10.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
                button_ind_11.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_yellow));
            } else {

panel_respiration_rate.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.frame_light_black_outline));
                button_ind_9.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
                button_ind_10.setBackgroundDrawable(getResources()
                .getDrawable(R.drawable.button_green));
            }
        }
    }
}

private String temperatureTransfer(int endValue) {
    temperature = String.valueOf(endValue);
    return temperature;
}

public void onActivityResult(int requestCode, int resultCode,
Intentdata) {
    if (D)
        Log.d(TAG, "onActivityResult " + resultCode);
    switch (requestCode) {
        case REQUEST_CONNECT_DEVICE:
        if (resultCode == Activity.RESULT_OK) {
            String address = data.getExtras().getString(
            DeviceListActivity.EXTRA_DEVICE_ADDRESS);
            BluetoothDevice device = mBluetoothAdapter
            .getRemoteDevice(address);
            mRfcommClient.connect(device);
        }
        break;
        case REQUEST_ENABLE_BT:
            if (resultCode == Activity.RESULT_OK) {
                setupMonitoring();
            } else {
                Log.d(TAG, "BT not enabled");
                Toast.makeText(this, R.string.bt_not_enabled_leaving,
                Toast.LENGTH_SHORT).show();
                finish();
            }
        }
    }
}
```

```
public void onSharedPreferenceChanged(SharedPreferences prefs, String
key) {
    if (key.equals("syncEnabled")) {
        if (!prefs.getBoolean("syncEnabled", false) && mBound) {
            Log.d(TAG,"Preferences changed, BluetoothRfcommService
should now unbind and stop");
            killBluetoothChatService();
        }
        if (prefs.getBoolean("syncEnabled", false) && !mBound) {
            startBluetoothChatService();
        }
    }
}

private void startBluetoothChatService() {
    if (preferences.getBoolean("syncEnabled", false)) {
        Intent intent = new Intent(this,
BluetoothRfcommService.class);
        startService(intent);
        setupMonitoring();
    }
}

private void killBluetoothChatService() {
    unbindService(mConnection);
    mBound = false;
    Intent intent = new Intent(this, BluetoothRfcommService.class);
    stopService(intent);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.option_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.scan:
            if (mBound) {
            Intent serverIntent = new Intent(this,
DeviceListActivity.class);
            startActivityForResult(serverIntent,
REQUEST_CONNECT_DEVICE);
            } else
            Toast.makeText(getBaseContext(),
            "BluetoothRfcommService is not enabled!",
            Toast.LENGTH_LONG).show();
            return true;
        case R.id.store:
            boolean didItWork = true;
            String b = "nirgends";
        try {

            for (int i = 0; i < bluedata.length() - 3; i++) {
```

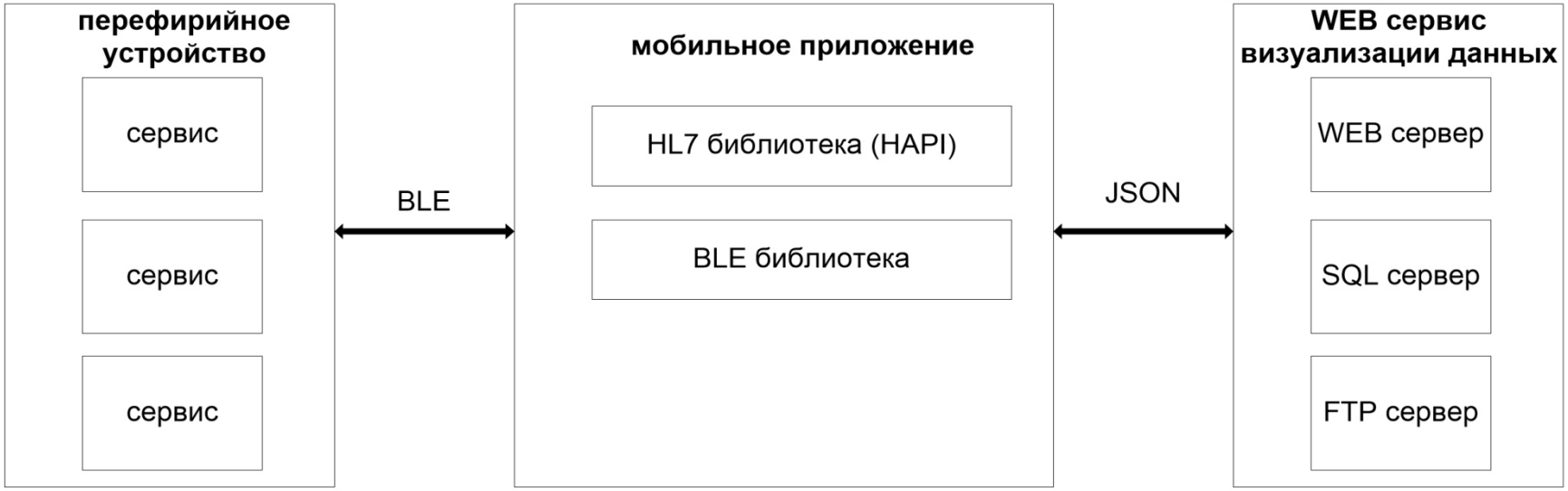
```
                DatabaseHelperClass DHC = new DatabaseHelperClass(
                BluetoothMonitoring.this);
                String dateTime = new SimpleDateFormat("yyyy-MM-dd
HH:mmZ").format(new Date());
                respirationRate,
                b =
BluetoothMonitoring.this.getDatabasePath("EquivitalDb").getPath();
        }

        } catch (Exception e) {
            didItWork = false;
            String error = e.toString();
            Dialog d = new Dialog(this);
            d.setTitle("Dang it!");
            TextView tv = new TextView(this);
            tv.setText(error);
            d.setContentView(tv);
            d.show();
        } finally {
        if (didItWork) {
            Dialog d = new Dialog(this);
            d.setTitle("Storage!");
            TextView tv = new TextView(this);
            tv.setText("Beginn to save, DB stored in" + b);
            d.setContentView(tv);
            d.show();
        }
    }
    return true;
}

private ServiceConnection mConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder
    service) {
    LocalBinder binder = (LocalBinder) service;
    mRfcommClient = binder.getService(mHandler);
    mBound = true;
    if ((preferences.getString("autoconnect_device", null) != null)
        && (preferences.getBoolean("autoconnect_enabled", false))
        && (mRfcommClient.getState() != BluetoothRfcommService.
        STATE_CONNECTED)) {
    String macAddress = preferences.getString("autoconnect_device",
null);
    BluetoothDevice device =
mBluetoothAdapter.getRemoteDevice(macAddress);
    mRfcommClient.connect(device);
    }
}

public void onServiceDisconnected(ComponentName argo) {
    mBound = false;
};

private class MyReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context arg0, Intent arg1) {
        }
```

```
        }
    }
```

ПРИЛОЖЕНИЕ Б

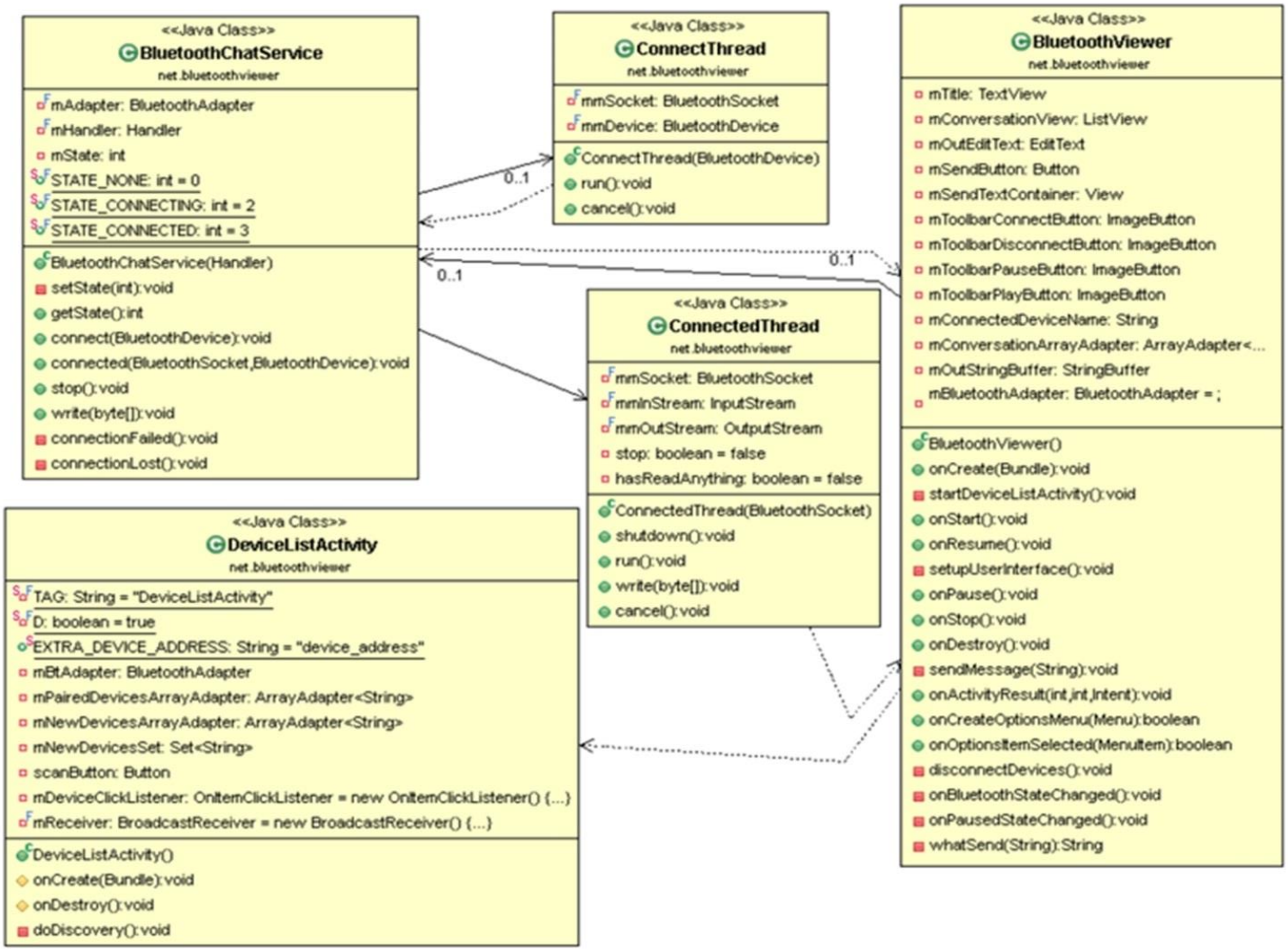
(обязательное)

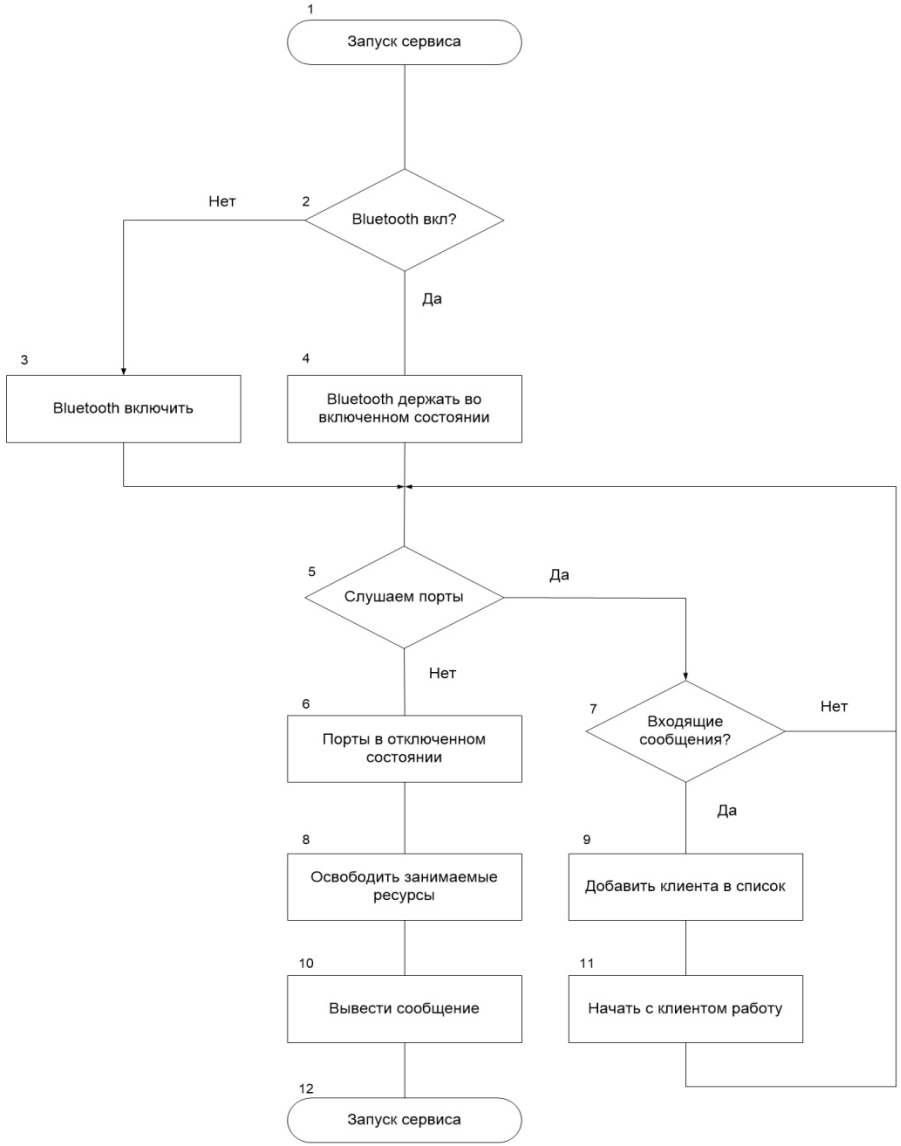
Схемы программы

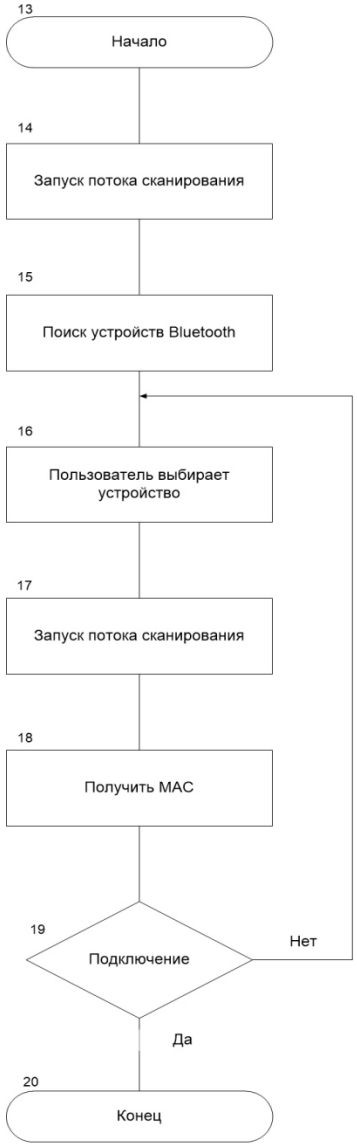


					ПГУ 1.090501.09.002 Д01	
Изм.	Лист.	№ докум.	Подпись	Дата		
Разраб.		Милованов Д.В.			Приложение обработки телеметрической информации Структурная схема	Лит. / Масса / Масш.
Провер.		Федонин Р.Н.				
Т.контр.						Лист 1 / Листов 1
Реценз.		Печерский С.В.				
Н.контр.		Кучин А.В.				ФВТ, каф. ВТ группа 15ВО2
Утв.		Митрохин М.А.				

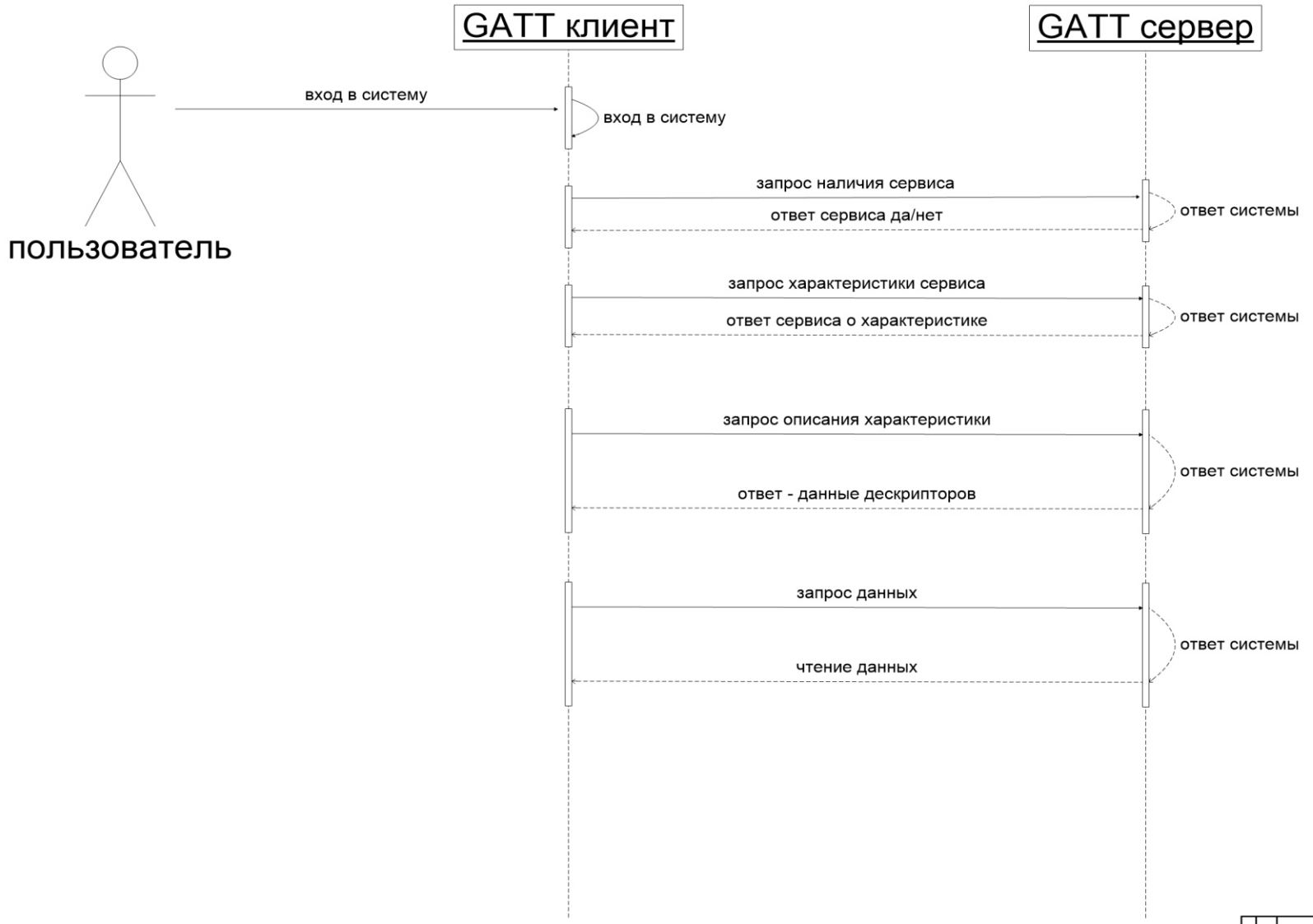
Диаграмма классов







					ПГУ 1.090501.09.004 Д03	
Инв. Лист	№ Докум.	Подпись	Дата			
Разраб.	Замозаков Д.В.			Приложение обработки телеметрической информации Схема программы	Лит.	Масса	Масш.
Провер.	Федосеев Р.Н.						
Т. контр.					Лист 1	Листов 1	
Реценз.	Печерский С.В.						
Н. контр.	Кучин А.В.				ФВТ, каф. ВТ группа 15ВО2		
Утв.	Митрохин М.А.						



						ПГУ 1.090501.09.005 Д04	
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Яковлев Д.В.			Приложение обработки телеметрической информации Диаграмма последовательности	Лит.	Масса	Масш.
Провер.		Федосеев Р.Л.						
Т. контр.						Лист 1	Листов 1	
Реценз.		Печерский С.В.				ФВТ, каф. ВТ группа 15ВО2		
Н. контр.		Кучин А.В.						
Утв.		Литохин М.А.						

ПРИЛОЖЕНИЕ В

(обязательное)

Презентация

Тема работы

Приложение обработки телеметрической информации

Автор работы: студент гр. 15ВО2 Милованов Д.В.
Руководитель ВКР: к.т.н., доцент каф. ВТ, Федюнин Р.Н.

2020

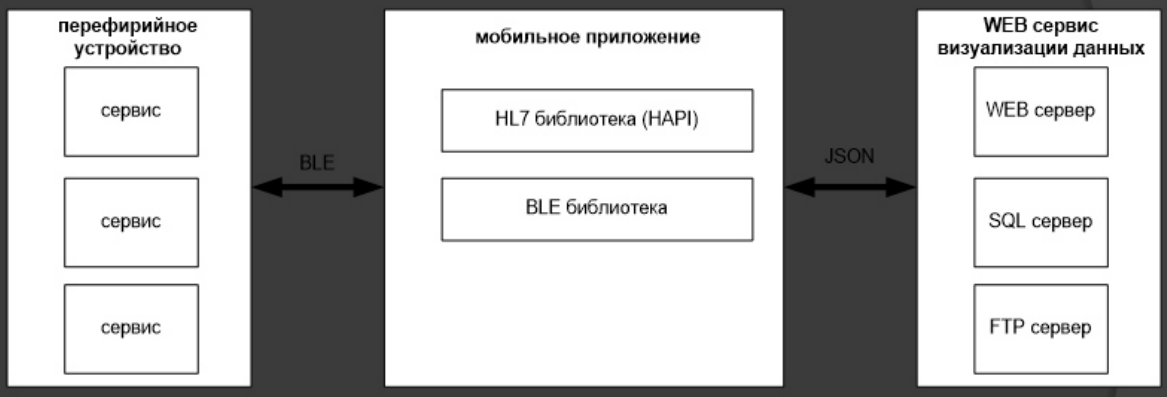
ЦЕЛИ И ЗАДАЧИ

- **Цель работы – разработка приложения, которое работает с носимым датчиком, передающим полученные данные клиенту и серверу.**
- **Задачи:**
  - **Изучить предметную область**
  - **Выбрать средства и технологии для разработки**
  - **Разработать приложение**
  - **Проверить работоспособность приложения**

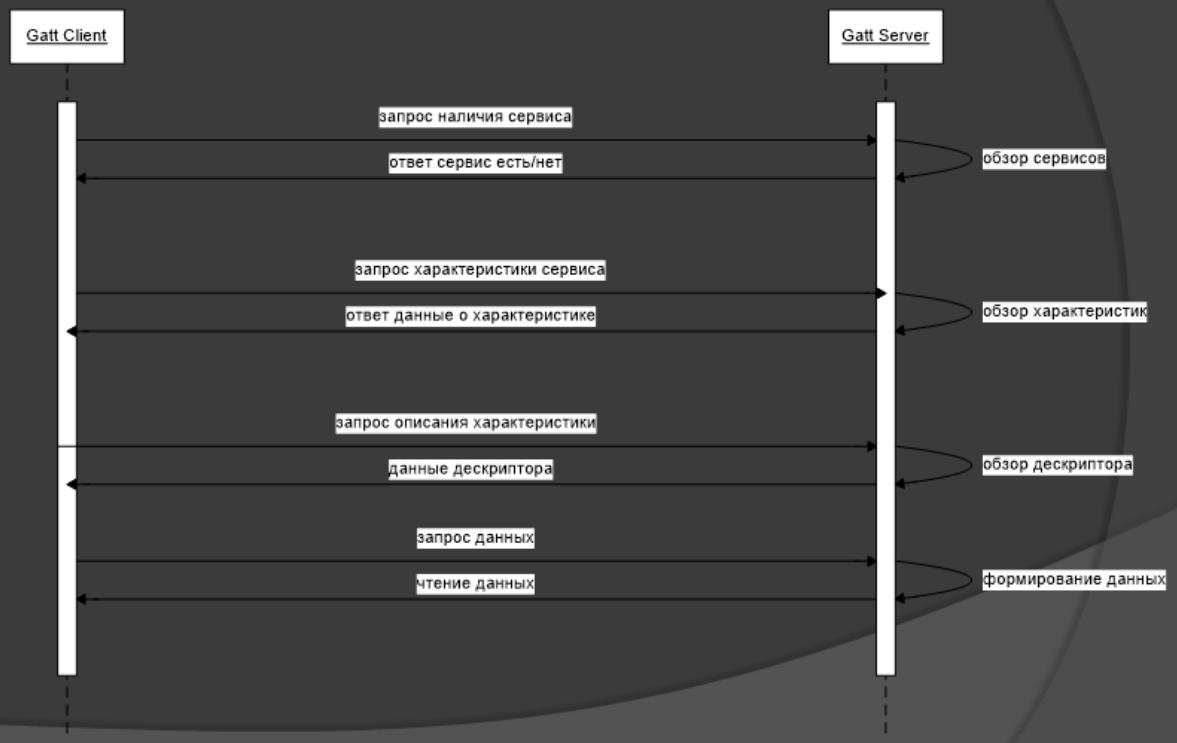
ОСНОВНЫЕ ТРЕБОВАНИЯ ДЛЯ ПРИЛОЖЕНИЯ

- Прием информации с конечных устройств
- Сохранение информации на мобильном устройстве
- Передача информации на сервер хранения данных

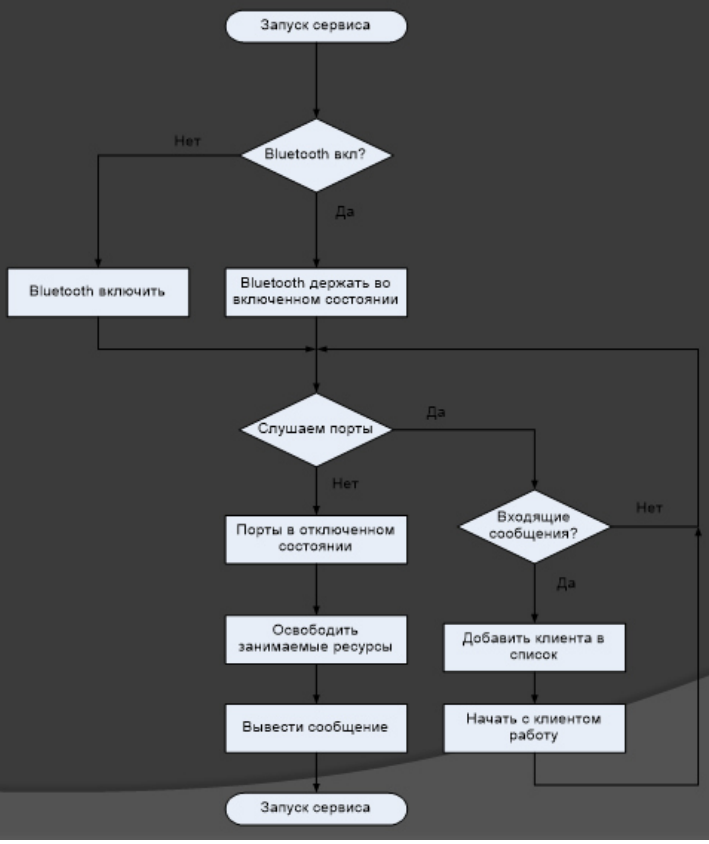
Аппаратно-программная система

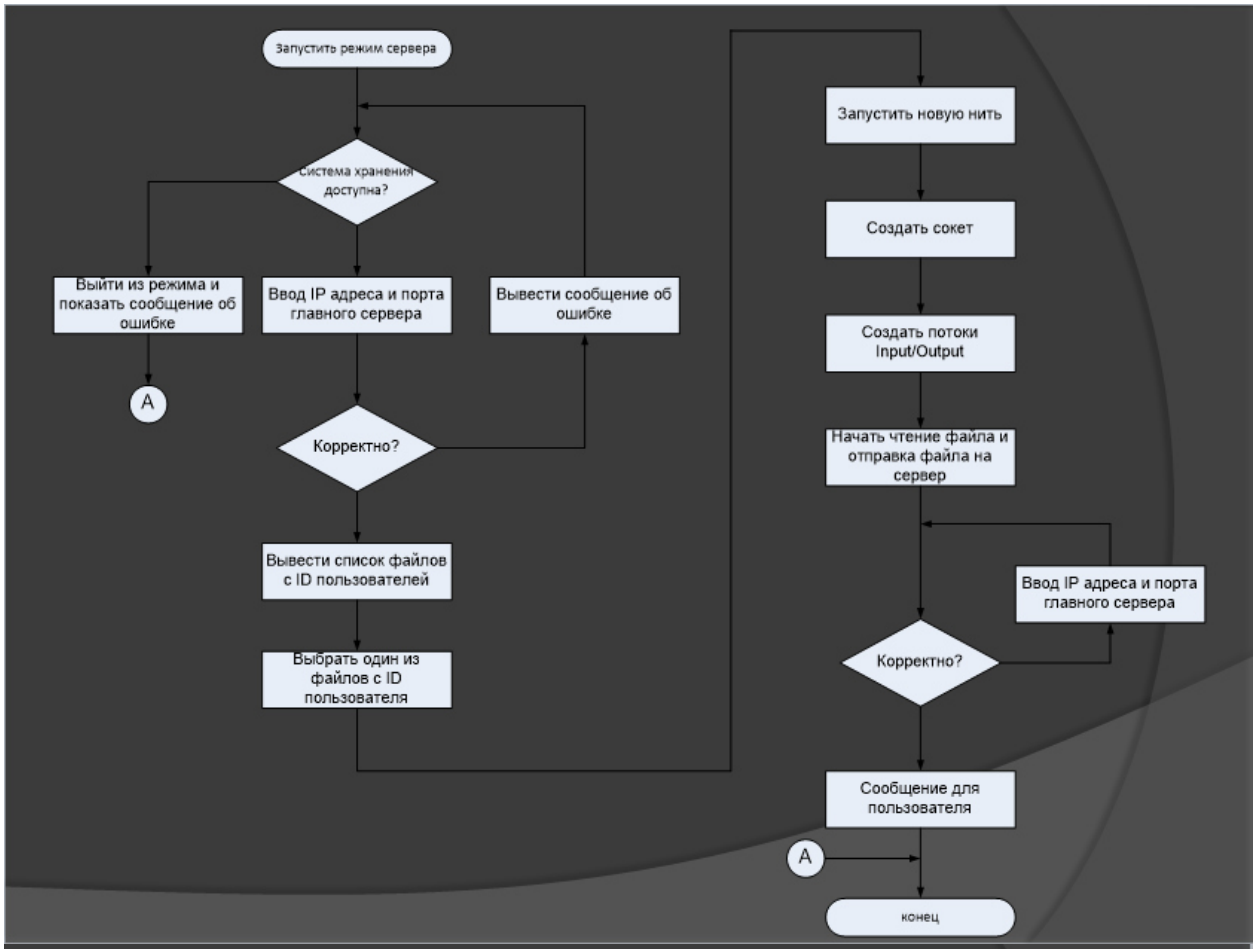


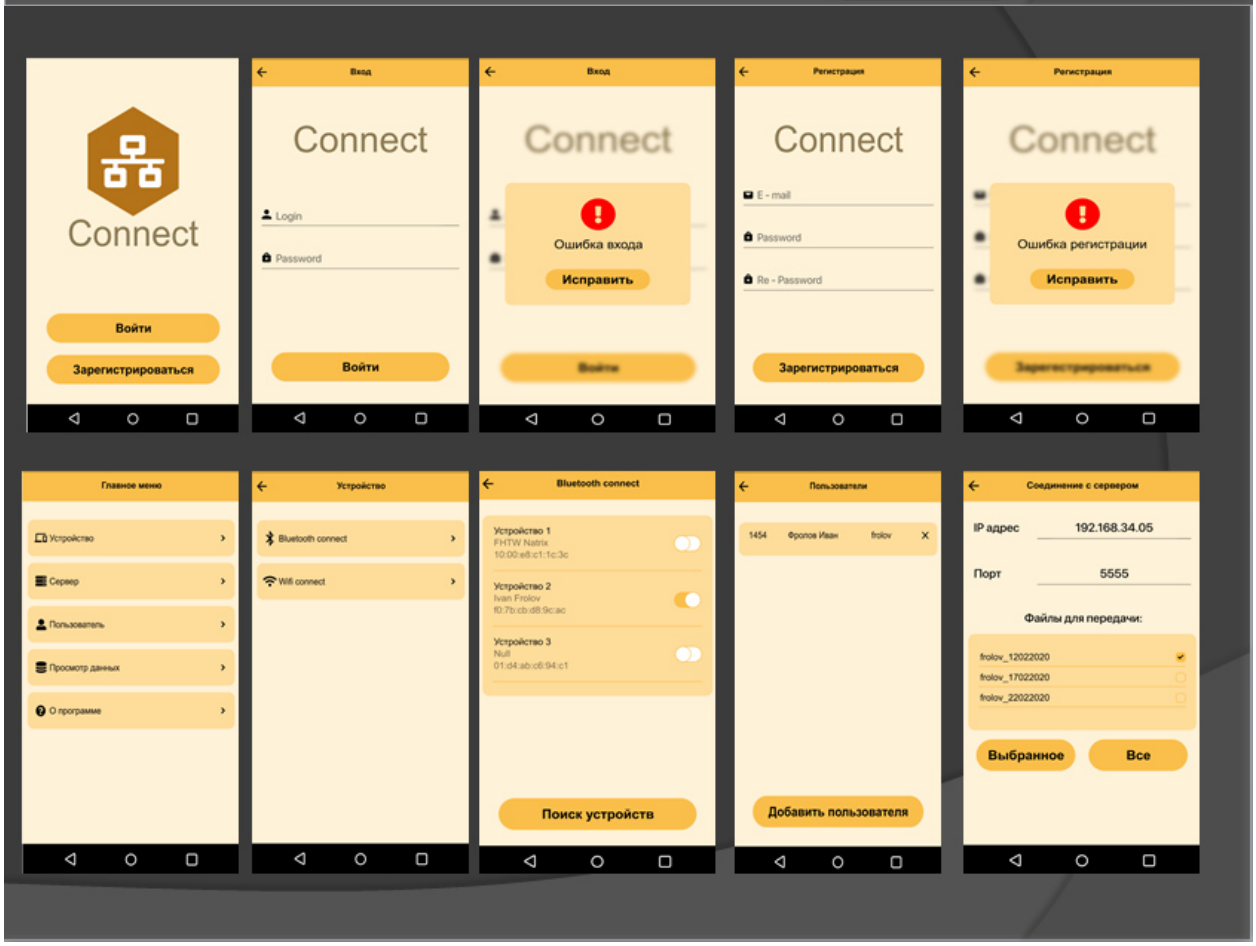
Протокол взаимодействия клиент/сервер



Обобщенная работа приложения в режиме ожидания




Запустить режим сервера
Система хранения доступна?
Выйти из режима и показать сообщение об ошибке
A
Ввод IP адреса и порта главного сервера
Вывести сообщение об ошибке
Корректно?
Вывести список файлов с ID пользователей
Выбрать один из файлов с ID пользователя
Запустить новую нить
Создать сокет
Создать потоки Input/Output
Начать чтение файла и отправка файла на сервер
Ввод IP адреса и порта главного сервера
Корректно?
Сообщение для пользователя
A
конец


Connect
Войти
Зарегистрироваться
Вход
Connect
Login
Password
Войти
Вход
Connect
Ошибка входа
Исправить
Регистрация
Connect
E - mail
Password
Re - Password
Зарегистрироваться
Регистрация
Connect
Ошибка регистрации
Исправить
Главное меню
Устройство
Сервер
Пользователь
Просмотр данных
О программе
Устройство
Bluetooth connect
Wifi connect
Bluetooth connect
Устройство 1
FHTW Natrix
10:00:e8:c1:1c:3c
Устройство 2
Ivan Frolov
f0:7b:cb:d8:9c:ac
Устройство 3
Null
01:d4:ab:c6:94:c1
Поиск устройств
Пользователи
1454 Фролов Иван frolov
Добавить пользователя
Соединение с сервером
IP адрес 192.168.34.05
Порт 5555
Файлы для передачи:
frolov_12022020
frolov_17022020
frolov_22022020
Выбранное
Все

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы было разработано мобильное приложение, обеспечивающее обработку телеметрических данных полученных с оконечного устройства.

Приложение построено с использованием технологий, которые являются производительными, гибкими и масштабируемыми.

Данное приложение может быть использовано для обмена необходимой информацией в целях диагностики, лечения и профилактики заболеваний и травм, проведения исследований и оценок.

СПАСИБО ЗА ВНИМАНИЕ