МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Институт ЗВФ

Направление подготовки 09.03.01 Информатика и вычислительная техника
(код и наименование)

Направленность (профиль) образовательной программы Вычислительные машины, комплексы, системы и сети
(наименование)

Кафедра Вычислительные системы и технологии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

бакалавра
(бакалавра, магистра, специалиста)

Студента Думина М.В. группы 14ВМ
(Ф.И.О.)

на тему Приложение на платформе Android для системы мобильного обхода

СТУДЕНТ:
Думин М.В
(подпись) (фамилия, и., о.)
_____
(дата)

РУКОВОДИТЕЛЬ:
Викулова Е.Н
(подпись) (фамилия, и., о.)
_____
(дата)

РЕЦЕНЗЕНТ:
_____
_____
(подпись) (фамилия, и., о.)
_____
(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ
Мякиньков А.В.
(подпись) (фамилия, и.о.)
_____
(дата)

КОНСУЛЬТАНТЫ:
1. По _____
_____ _____
(подпись) (фамилия, и., о.)
_____
(дата)

2. По _____
_____ _____
(подпись) (фамилия, и., о.)
_____
(дата)

3. По _____
_____ _____
(подпись) (фамилия, и., о.)
_____
(дата)

ВКР защищена _____
(дата)
протокол № _____
с оценкой _____

Приложение на платформе Android для системы мобильного обхода, Выпускная квалификационная работа бакалавра по направлению: «Вычислительные машины, комплексы, системы и сети», Нижегородский государственный технический университет им. Р.Е. Алексеева, кафедра: «Вычислительные системы и технологии», Нижний Новгород, 2015. Руководитель: доцент кафедры «Вычислительные системы и технологии» Викулова Е.Н.

Работа посвящена проектированию и разработке приложения, решающей задачу контроля за совершением обхода предприятия, идентификации датчиков и получения их показаний на мобильное устройство. Описывается структура системы, методы и способы реализации модулей, входящих в систему.

В результате тестирования установлено, что разработанное приложение корректно решает поставленную задачу контроля за совершением обхода предприятия, идентификации датчиков и получения их показаний на мобильное устройство. Объем работы 71 страница. Использовано источников –18, рисунков – 15, таблиц –1.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире происходит всё большее усложнение технологических процессов, что в свою очередь требует более пристального внимания за их осуществлением. Сложных механизмов и устройств даже в рамках всего лишь одной площадки множество. С развитием промышленности увеличивается количество устройств, которые нужно контролировать и получать от них различные данные. Составные вентили и заглушки, насосы, трубопроводы, устройства пожаротушения — за всем этим надо следить, у каждого узла в нужный момент времени должны быть определенные параметры: давление в трубах, температура узла, степень открытия какой-либо заглушки и тому подобное.

Автоматизация позволяет в значительной мере облегчить контроль за рядом самых критичных параметров, но показания с датчиков в таком случае, поступают на центральный пульт управления системой, и персоналу, во время планового обхода или выполнения работ, находящемуся непосредственно рядом с элементами данной системы, приходится запрашивать данные у оператора, находящегося за пультом управления. При этом каждый из них должен четко понимать о показаниях какого датчика идёт речь и где он расположен (как непосредственно на месте, так и на пульте управления). Всё это создаёт дополнительные сложности, может привести к ошибкам в действиях персонала, и как следствие к внештатным ситуациям.

Решением данной проблемы станет возможность получения персоналом данных о состоянии отдельных элементов системы на портативное устройство, подключённое к беспроводной сети. Это позволит мастеру оперативно получать объективную и актуальную информацию, оставаясь мобильным, и полностью исключит ошибки при коммуникации с оператором. Кроме того данное устройство можно использовать для контроля за осуществлением обхода, чтобы ни один узел не остался без внимания. Устройств отвечающих требованиям по функционалу и безопасности сейчас огромное множество. Целью данной работы будет создание приложения для осуществления мониторинга системы, которое позволит точно идентифицировать датчики и получать данные об их показаниях на устройство.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Наименование разработки

Приложение на платформе Android для системы мобильного обхода

1.2 Основания для разработки

Данная работа выполняется на основании приказа на выполнение выпускной квалификационной работы.

1.3 Назначение разработки

Цель разработки — создание программной системы для автоматизации контроля за проведением обходов на предприятиях, повышения качества их осуществления, освобождение персонала от рутинных операций по заполнению журналов обхода.

1.4 Область применения разработки

Приложение предназначено для работы на предприятиях с системой централизованного получения технологической информации с объектов её инфраструктуры.

1.5. Функциональное назначение

Идентификация датчиков во время совершения обхода, получение информации с ЦП о значениях их показаний, визуализация показаний с датчика на экране мобильного устройства, отправка сообщения на ЦП о том, что объект был осмотрен.

1.6 Требования к системе.

1.6.1 Требования к функциональным характеристикам

Организация идентификации датчиков при совершении обхода;

Реализация подключения к серверу с данными о показаниях датчиков;

Организация автоматического получения данных о показаниях идентифицированных датчиков;

Сохранение информации, полученной с датчиков, в памяти устройства;

Отображение информации полученной с датчиков в виде списка датчиков и значений их показаний.

Необходимо предоставить пользователю возможность выборочного удаления записей или полную очистку приложения от данных с датчиков.

Требуется организовать автоматическую отправку данных о совершении обхода, для поддержания трудовой дисциплины.

Реализация интуитивно понятного пользовательского интерфейса.

В интерфейсе должны быть реализованы следующие возможности:

- Аутентификация;
- Настройка подключения к серверу;
- Идентификация датчиков;
- Визуализация сохранённых показаний;

1.6.2 Требования к дизайну

Мобильное приложение должно соответствие принципам Material Design[6] мобильных приложений Google. Дизайн приложения должен быть адаптирован для корректного отображения при различных разрешениях экрана. При работе с приложением пользователь должен получать обратную связь от своих действий. В приложении это можно реализовать следующими способами:

- внешний вид кнопок изменяется при нажатии на них (меняют оттенок);
- поля, заполняемые пользователем, выделяются цветом;
- для соответствия рекомендациям для приложений Android от Google, нужно придерживаться принципов Material Design.

1.6.3 Требования к входным (выходным) данным

Входные данные — имя датчика, представленное в удобном для быстрого ввода виде.

Выходные данные — показания датчика на экране мобильного устройства.

1.6.4 Требования к аппаратному и программному обеспечению

Мобильный телефон – смартфон с поддержкой передачи данных сети Wi-fi, Bluetooth, NFC, оснащённый цифровой камерой, под управлением операционной системы Android.

Чтобы построить взаимодействие с системой необходимо определить сетевой протокол для передачи данных на устройство посредством беспроводной сети. Разработать представление полученной информации приложении.

Для получения данных с датчика необходимо будет однозначно идентифицировать его в системе, поэтому нужно предусмотреть уникальное именование датчиков в системе и то, каким образом имя датчика будет передаваться в приложение.

1.7 Аналоги

Аналогами данного сервиса являются проекты вроде «Электронная система обходов оборудования»[1] или «Интерактивная система ОБХОД»[2].

Обе системы предназначены для автоматизации проведения обходов и используют специальные RFID метки для распознавания датчиков и помогают составить отчёт о проведении обхода, но данные в них необходимо вводить вручную.

2АНАЛИЗ

2.1 Аппаратное и программное обеспечение

2.1.1 Целевое устройство

В качестве устройства был выбран смартфон под управлением ОС Android, так он предоставляет широкий спектр возможностей для выполнения различных задач, при этом существует множество моделей от простых и относительно дешевых до взрывозащищённых и ударопрочных (рис. 2.1).

Рисунок 2.1 Взрывозащищённый смартфон

На объектах нефте-газовой сферы и химической промышленности одно из основных требований к электронным девайсам на площадке — взрывозащищённость, то есть устройство не должно становиться источником взрыва (не создавать искру и подобное).

2.1.2 Выбор поддерживаемых версий платформы Android

Android — свободная операционная система для мобильных телефонов, планшетных компьютеров, умных часов, телевизоров и смартбуков, использующая ядро Linux, разрабатываемая Open Handset Alliance и принадлежащая Google. С момента выхода первой версии в сентябре 2008 года произошло 40 обновлений системы. Эти обновления, как правило, касаются исправления обнаруженных ошибок и добавления новой функциональности в систему.

Для того чтобы охватить большое количество мобильных устройств, было решено провести анализ того, какие версии платформы Android наиболее часто используются и в каком процентном соотношении. В таблице 2.1 приведена статистика распространения версий платформы Android.

Таблица 2.1 – Статистика распространения версий на 7 мая 2019 года[3]:

Версия	Название	Год	Доля
2.3	*Gingerbread*	2010	0,3%
4.0	*Ice Cream Sandwich*	2011	0,3%
4.1	*Jelly Bean*	2012	1,2%
4.2		2012	1,5%
4.3		2013	0,5%
4.4	*KitKat*	2013	6,9%
5.0	*Lollipop*	2014	3%
5.1		2015	11,5%
6.0	*Marshmallow*	2015	16,9%
7.0	*Nougat*	2016	11,4%
7.1		2016	7,8%
8.0	*Oreo*	2017	12,9%
8.1		2017	15,4%
9.0	*Pie*	2018	10,4%
10.0	*Q*	2019	< 0,1%
Примечание: Учитываются только те устройства, владельцы которых запускали Google на своём аппарате хотя бы раз за период сбора информации. Версии, имеющие менее 0,1% учитываются статистикой.			

Для анализа использовались данные об относительном количестве устройств, работающих под управлением различных версий платформы Android. Данные собраны в течение 7-дневного периода, закончившегося 7 мая 2019 года. Данные собраны из приложения Google Play, которое поддерживается начиная с версии Android 2.2 и выше, так что устройств, работающих под управлением более старых версий в статистике нет.

Из таблицы видно что версия *Jelly Bean* (4.1-4.3) всё ещё используется более чем на 3% устройств.

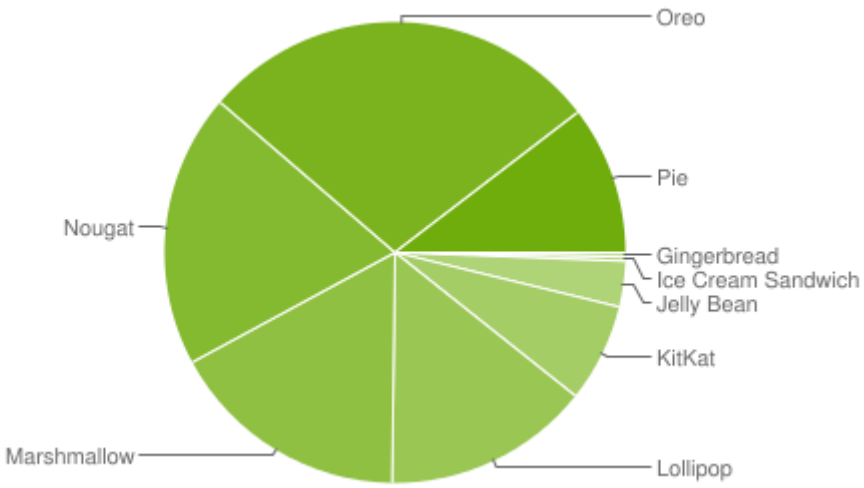


Диаграмма 2.1 – Относительное количество устройств на платформе Android

В результате проведенного анализа было решено поддерживать версии начиная с 4.1 для охвата большого количества платформ Android.

2.1.3 Идентификация датчиков в системе

Так как датчиков может быть огромное множество, а их типы и назначение повторятся, целесообразно их структурировать в виде каталогов. Имена вложенных каталогов указывать через «слеш» (/), последним будет указано название датчика, например:

'Название_системы / название_подсистемы / имя_датчика ' - это будет полное имя датчика. Полное имя датчика будет использоваться для получения информации с него.

Для того чтобы быстро и точно передать имя датчика в приложение можно использовать Bluetooth-маяки и NFC-метки, закодировав его в соответствующий сигнал.

- **Bluetooth** - открытый стандарт беспроводной связи с низким энергопотреблением, обеспечивающий передачу данных и звука между совместимыми устройствами.

- **Near field communication**, **NFC** («коммуникация ближнего поля», «ближняя бесконтактная связь») — технология беспроводной передачи данных малого радиуса действия, которая дает возможность обмена данными между устройствами, находящимися на расстоянии 4 сантиметров[4];

В рамках дипломной работы я заменил эти два способа передачи данных считыванием QR-кодов. Данная функция работает аналогично, только используется не чип NFC или bluetooth модуль, а камера смартфона. Это делает систему ещё более доступной, так как не нужно использовать дополнительное оборудование кроме принтера. Вместе с тем она имеет один существенный недостаток: недобросовестный обходчик может скопировать изображение и никуда не ходить.

2.1.4 Сетевой протокол для передачи данных на устройство

Для решения проблем взаимодействия большого количества устройств и проблем объединения устройств в одну сеть была создана концепция Интернета вещей (англ. Internet of Things, IoT) – это когда устройства объединяются по какому-то признаку в одну сеть, потом уже несколько подобных сетей объединяются в другую большую сеть и так далее[5].

Устройства в таких сетях взаимодействуют друг с другом по средствам различных интерфейсов и протоколов передачи данных. Так как мы говорим о промышленном применении концепции IoT, в которой должны использоваться промышленное оборудование со своими протоколами и аппаратными средствами, то мы переходим к концепции IIoT (Промышленного Интернета вещей).

Для взаимодействия между собой устройства используют различные промышленные протоколы, одним из популярных протоколов для этой цели является MQTT.

2.1.5 Хранение информации

Информацию сохраняемая в приложении делится на два группы:

- Параметры камеры(подсветка, автофокусировка) и настройки соединения
- Показания датчиков

Первая группа характеризуется небольшим, но постоянным объёмом данных. Поэтому для неё в качестве хранилища будет использован SharedPreferences — постоянное хранилище на платформе Android, используемое приложениями для хранения своих настроек.

Вторая группа — показания датчиков — очень динамична в процессе работы и может как увеличивать объём данных, используемых в приложении, так и уменьшать. Следовательно для хранения информации больше подходит база данных. Стандартным решением для этих целей является база данных SQLite

2.2 Используемые языки и программное обеспечение

При разработке мобильного приложения, веб-приложения и сервера рекомендуется использовать следующие технологии и языки программирования:

- Android SDK
- Java
- MQTT
- Paho Android Service
- SQLite

2.2.1 Android

Android – операционная система для смартфонов, планшетов, электронных книг, цифровых проигрывателей, наручных часов, игровых приставок, нетбуков, очков Google, телевизоров, мультимедийных систем в автомобилях и других устройств. В качестве ядра операционной системы используется Linux [7] и реализации виртуальной машины Java от Google [8].

Android позволяет создавать Java-приложения, управляющие устройством через разработанные Google библиотеки. Данная платформа была выбрана как самая популярная и широко распространенная платформа для мобильных устройств.

Для разработки используется среда Android Studio, основанная на IntelliJ IDEA от JetBrains [9]. Процесс создания Android приложения не требует дополнительных устройств, кроме, Android устройства (можно использовать эмулятор). Andriod Studio характеризуется:

- гибкой системой сборки Gradle;
- расширенной поддержкой сервисов Google и различных типов устройств;
- богатым функционалом редактором экранов приложений с поддержкой редактирования тем интерфейса;
- возможностью подписания приложений;
- встроенной поддержкой облачной платформы Google и возможности простой интеграции с Google Cloud Messaging и App Engine[10].

2.2.2 Java

Язык Java – это объектно-ориентированный язык программирования. Приложения Java транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине вне зависимости от компьютерной архитектуры. Виртуальная машина Java (JVM) – программа, обрабатывающая байтовый код и передающая инструкции оборудованию как интерпретатор.

Язык Java был выбран, т.к. платформа Android подразумевает разработку приложений на этом языке. В Android используется собственная версия JVM, которая активно поддерживается и является основной средой выполнения приложений для данной платформы.

2.2.3 QR-code

QR-код (англ. Quick Response Code — код быстрого реагирования; сокр. QR code) — товарный знак для типа матричных штрихкодов (или двумерных штрихкодов), изначально разработанных для автомобильной промышленности Японии.

Штрихкод — считываемая машиной оптическая метка, содержащая информацию об объекте, к которому она привязана. QR-код использует четыре стандартизированных режима кодирования (числовой, буквенно-цифровой, двоичный и кандзи) для эффективного хранения данных; могут также использоваться расширения[11].

QR-код состоит из чёрных квадратов, расположенных в квадратной сетке на белом фоне, которые могут считываться с помощью устройств обработки изображений, таких как камера, и обрабатываться с использованием кодов Рида — Соломона[12] до тех пор, пока изображение не будет надлежащим образом распознано. Затем необходимые данные извлекаются из шаблонов, которые присутствуют в горизонтальных и вертикальных компонентах изображения.

Основное достоинство QR-кода — это лёгкое распознавание сканирующим оборудованием, что даёт возможность использования в торговле, производстве, логистике.

Хотя обозначение «QR code» является зарегистрированным товарным знаком «DENSO Corporation», использование кодов не облагается никакими лицензионными отчислениями, а сами они описаны и опубликованы в качестве стандартов ISO.

Закодировать информацию в QR-код можно несколькими способами, а выбор конкретного способа зависит от того, какие символы используются. Если используются только цифры от 0 до 9, то можно применить цифровое кодирование, если кроме цифр необходимо зашифровать буквы латинского алфавита, пробел и символы ±*/$%*.:, используется алфавитно-цифровое кодирование. Ещё существует кодирование кандзи, которое применяется для шифрования китайских и японских иероглифов, и побайтовое кодирование. Перед каждым способом кодирования создаётся пустая последовательность бит, которая затем заполняется.

Рисунок 2.2 – Пример QR-кода

Для обнаружения штрих-кода будут задействованы интерфейсы Mobile Vision, которые обеспечивают соответствующий API. Классы для обнаружения и анализа штрих-кодов доступны в пространстве имен com.google.android.gms.vision.barcode. Они считывают и декодируют множество различных типов штрих-кодов. Основным является класс BarcodeDetector[13]. Он выполняет обработку объектов Frame и возвращает массив штрих-кодов SparseArray <Barcode>.

Тип Barcode представляет собой единый общепризнанный штрих-код и его значение. В случае 1D штрихкодов, таких как коды UPC, это будет просто номер, который закодирован в штрихкоде. Его значение доступно в поле rawValue, в то время как тип штрих-кода (то есть его кодировку) можно найти в поле format.

Для 2D штрихкодов, которые содержат структурированные данные, такие как QR-коды — в поле valueFormat устанавливается определенный тип значения, соответствующего полю данных. Так, например, если обнаружен тип URL , то поле valueFormat вернет константу URL, а объект Barcode.UrlBookmark будет содержать значение URL-адреса. Помимо URL-адресов, существует множество различных типов данных, которые QR-код может хранить. Например, почтовый адрес, дату и время события календаря, мероприятие в календаре, информацию контакта, номер телефона, местоположение на карте и другие данные, полный список которых приводится в документации[14].

Использование в приложении Mobile Vision API позволяет считывать штрихкоды в любом положении.

Важно отметить, что синтаксический разбор всех штрихкодов выполняется локально, поэтому не придётся использовать соединение с сервером для чтения данных из кода. Например, при считывании линейного штрих-кода PDF-417, который может вместить до 1 КБ текста, можно сразу же получить всю закодированную в нем информацию.

2.2.4 MQTT

MQTT или Message Queue Telemetry Transport – это легкий, компактный и открытый протокол обмена данными созданный для передачи данных на удалённых локациях, где требуется небольшой размер кода и есть ограничения по пропускной способности канала. Вышеперечисленные достоинства позволяют применять его в системах M2M (Машинно-Машинное взаимодействие) и IIoT (Промышленный Интернет вещей).

Основные особенности протокола MQTT:

- Асинхронный протокол
- Компактные сообщения
- Работа в условиях нестабильной связи на линии передачи данных
- Поддержка нескольких уровней качества обслуживания (QoS)
- Легкая интеграция новых устройств

Протокол MQTT работает на прикладном уровне поверх TCP/IP (рис. 2.3)и использует по умолчанию 1883 порт (8883 при подключении через SSL).

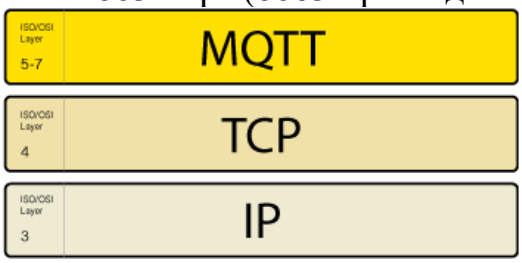


Рисунок. 2.3 – Расположение протокола MQTT относительно TCP/IP

Обмен сообщениями в протоколе MQTT осуществляется между клиентом (client), который может быть издателем или подписчиком (publisher/subscriber) сообщений, и брокером (broker) сообщений (например, Mosquitto MQTT).

Издатель отправляет данные на MQTT брокер, указывая в сообщении определенную тему, топик (topic). Подписчики могут получать разные данные от множества издателей в зависимости от подписки на соответствующие топики.

Устройства MQTT используют определенные типы сообщений для взаимодействия с брокером, ниже представлены основные:

- Connect – установить соединение с брокером;
- Disconnect – разорвать соединение с брокером;
- Publish – опубликовать данные в топик на брокере;
- Subscribe – подписаться на топик на брокере;
- Unsubscribe – отписаться от топика;

Рис 2.4 – Схема простого взаимодействия между подписчиком, издателем и брокером

Семантика топиков(тем)

Топики представляют собой символы с кодировкой UTF-8. Иерархическая структура топиков имеет формат «дерева», что упрощает их организацию и доступ к данным. Топики состоят из одного или нескольких уровней, которые разделены между собой символом «/».

Пример топика в который датчик температуры, расположенный в спальной комнате публикует данные брокеру:

/enterprise/production-hall/ foundry-shop/temperature

Подписчик может так же получать данные сразу с нескольких топиков, для этого существуют wildcard. Они бывают двух типов: одноуровневые и многоуровневые. Для более простого понимания рассмотрим в примерах каждый из них:

Одноуровневый wildcard. Для его использования применяется символ «+»

К примеру, нам необходимо получить данные о температуры во всех спальных комнатах:

/enterprise/production-hall/+/temperature

В результате получаем данные с топиков:

- */enterprise/production-hall/foundry-shop/*temperature
- */enterprise/production-hall/assembly-line/*temperature
- */enterprise/production-hall/warehouse/*temperature

Многоуровневый wildcard. Для его использования применяется символ «#» К примеру, чтобы получить данные с различных датчиков всех спален в доме:

/enterprise/production-hall/#

В результате получаем данные с топиков:

- */enterprise/production-hall/foundry-shop/*temperature
- */enterprise/production-hall/foundry-shop/*light1
- */enterprise/production-hall/foundry-shop/*light2
- */enterprise/production-hall/foundry-shop/*humidity
- */enterprise/production-hall/assembly-line/*temperature
- */enterprise/production-hall/assembly-line/*light1
- …

Качество обслуживания в протоколе MQTT (QoS)

MQTT поддерживает три уровня качества обслуживания (QoS) при передаче сообщений.

QoS 0 At most once. На этом уровне издатель один раз отправляет сообщение брокеру и не ждет подтверждения от него, то есть отправил и забыл.

Рисунок 2.5 – Уровень качества обслуживания QoS 0

QoS 1 At least once. Этот уровень гарантирует, что сообщение точно будет доставлено брокеру, но есть вероятность дублирования сообщений от издателя. После получения дубликата сообщения, брокер снова рассылает это сообщение подписчикам, а издателю снова отправляет подтверждение о получении сообщения. Если издатель не получил PUBACK сообщения от брокера, он повторно отправляет этот пакет, при этом в DUP устанавливается «1».

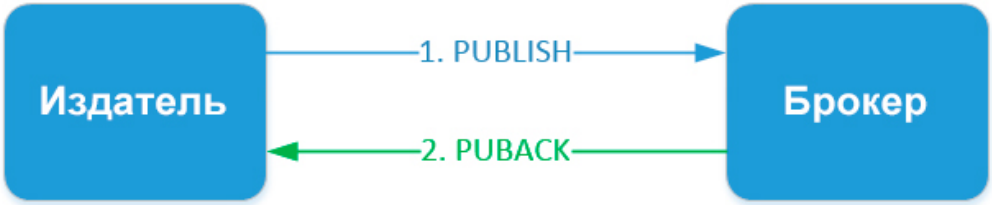


Рисунок 2.6– Уровень качества обслуживания QoS 1

QoS 2 Exactly once. На этом уровне гарантируется доставка сообщений подписчику и исключается возможное дублирование отправленных сообщений.

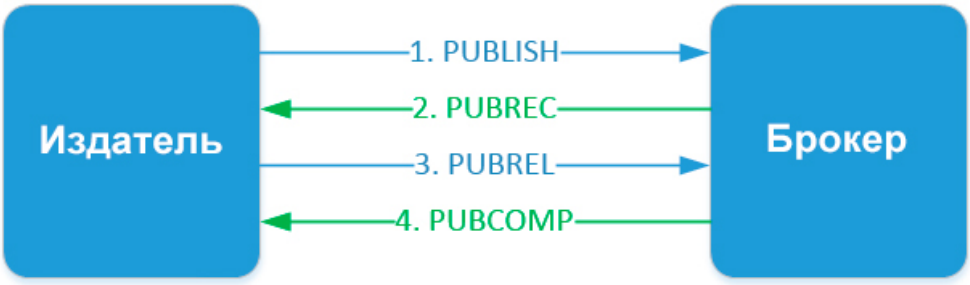


Рисунок 2.7 – Уровень качества обслуживания QoS 3

Издатель отправляет сообщение брокеру. В этом сообщении указывается уникальный Packet ID, QoS=2 и DUP=0. Издатель хранит сообщение неподтвержденным пока не получит от брокера ответ PUBREC. Брокер отвечает сообщением PUBREC в котором содержится тот же Packet ID. После его получения издатель отправляет PUBREL с тем же Packet ID. До того, как брокер получит PUBREL он должен хранить копию сообщения у себя. После получения PUBREL он удаляет копию сообщения и отправляет издателю сообщение PUBCOMP о том, что транзакция завершена.

Защита передачи данных

Для обеспечения безопасности в MQTT протоколе реализованы следующие методы защиты:

- Аутентификация клиентов. Пакет CONNECT может содержать в себе поля USERNAME и PASSWORD. При реализации брокера можно использовать эти поля для аутентификации клиента
- Контроль доступа клиентов через Client ID
- Подключение к брокеру через TLS/SSL

Оборудование для IIoT с поддержкой MQTT:

- Web-программируемые IIoT контроллеры
- IIoT модули удаленного ввода/вывода
- Шлюзы IIoT
- IIoT платформы, сервера MQTT
- Измерители CO, CO2, температуры и влажности

2.2.5 Paho Android Service

Paho Android Service — это интерфейс к клиентской библиотеке MQTT Java Paho для платформы Android. Соединение MQTT инкапсулируется в Android-сервис, который работает в фоновом режиме приложения для Android, сохраняя его, когда приложение для Android переключается между различными Activity. Этот уровень абстракции необходим для надежного получения сообщений MQTT. Поскольку служба Paho Android основана на клиентской библиотеке Java Paho, ее можно считать стабильной и использовать в производстве.

Проект Paho Android был создан для обеспечения надежной реализации открытых и стандартных протоколов обмена сообщениями с открытым исходным кодом, направленных на новые, существующие и новые приложения для межмашинного взаимодействия (M2M) и Интернета вещей (IoT). Проект активно поддерживается командой разработчиков Eclipse.

2.2.6 SQLite – это встраиваемая библиотека, в которой реализовано многое из стандарта SQL 92. Позиция функциональности SQLite где-то между MySQL и PostgreSQL[15]. Однако, на практике, SQLite нередко оказывается в 2-3 раза (и даже больше) быстрее. Такое возможно благодаря высокоупорядоченной внутренней архитектуре и устранению необходимости в соединениях типа «сервер-клиент» и «клиент-сервер».

Используя высокоэффективную инфраструктуру, SQLite может работать в крошечном объёме выделяемой для неё памяти, гораздо меньшем, чем в любых других системах БД. Это делает SQLite очень удобным инструментом с возможностью использования практически в любых задачах возлагаемых на базу данных.

Будучи файловой БД, она предоставляет отличный набор инструментов для более простой (в сравнении с серверными БД) обработки любых видов данных. Когда приложение использует SQLite, их связь производится с помощью функциональных и прямых вызовов файлов, содержащих данные (например, баз данных SQLite), а не какого-то интерфейса, что повышает скорость и производительность операций.

Поддерживаемые типы данных

- `NULL`: `NULL`-значение.
- `INTEGER`: целое со знаком, хранящееся в 1, 2, 3, 4, 6, или 8 байтах.
- `REAL`: число с плавающей запятой, хранящееся в 8-байтовом формате IEEE.
- `TEXT`: текстовая строка с кодировкой UTF-8, UTF-16BE или UTF-16LE.
- `BLOB`: тип данных, хранящийся точно в таком же виде, в каком и был получен.

Преимущества

- Файловая: вся база данных хранится в одном файле, что облегчает перемещение.
- Стандартизированная: SQLite использует SQL; некоторые функции опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), однако, есть и некоторые новые.
- Отлично подходит для разработки и даже тестирования: во время этапа разработки большинству требуется масштабируемое решение. SQLite, со своим богатым набором функций, может предоставить более чем достаточный функционал, при этом, будучи достаточно простой для работы с одним файлом и связанной библиотекой на языке Си.

Недостатки

- Отсутствие пользовательского управления: продвинутые БД предоставляют пользователям возможность управлять связями в таблицах в соответствии с привилегиями, но у SQLite такой функции нет.
- Невозможность дополнительной настройки: опять-таки, SQLite нельзя сделать более производительной, поковырявшись в настройках — так уж она устроена.

Применение SQLite

- Встроенные приложения: все портируемые не предназначенные для масштабирования приложения — например, локальные однопользовательские приложения, мобильные приложения или игры.
- Система доступа к дисковой памяти: в большинстве случаев приложения, часто производящие прямые операции чтения/записи на диск, можно перевести на SQLite для повышения производительности.
- Тестирование: отлично подойдёт для большинства приложений, частью функционала, которых является тестирование бизнес-логики.

3 ПРОЕКТИРОВАНИЕ

3.1 Разработка логики работы мобильного приложения

Исходя из анализа требований, предъявляемых к системе, определяется набор следующих функций, выполнение которых программа должна поддерживать:

- Идентификация датчиков
- Реализация подключения к серверу с данными о показаниях датчиков;
- Организация автоматического получения данных о показаниях идентифицированных датчиков;
- Сохранение информации, полученной с датчиков, в памяти устройства;
- Отображение информации полученной с датчиков в виде списка датчиков и значений их показаний.
- Автоматическая отправка данных о совершении обхода, для поддержания трудовой дисциплины.

На основе этих функция была составлена структурная схема системы, которая приведена на рисунке 3.1.

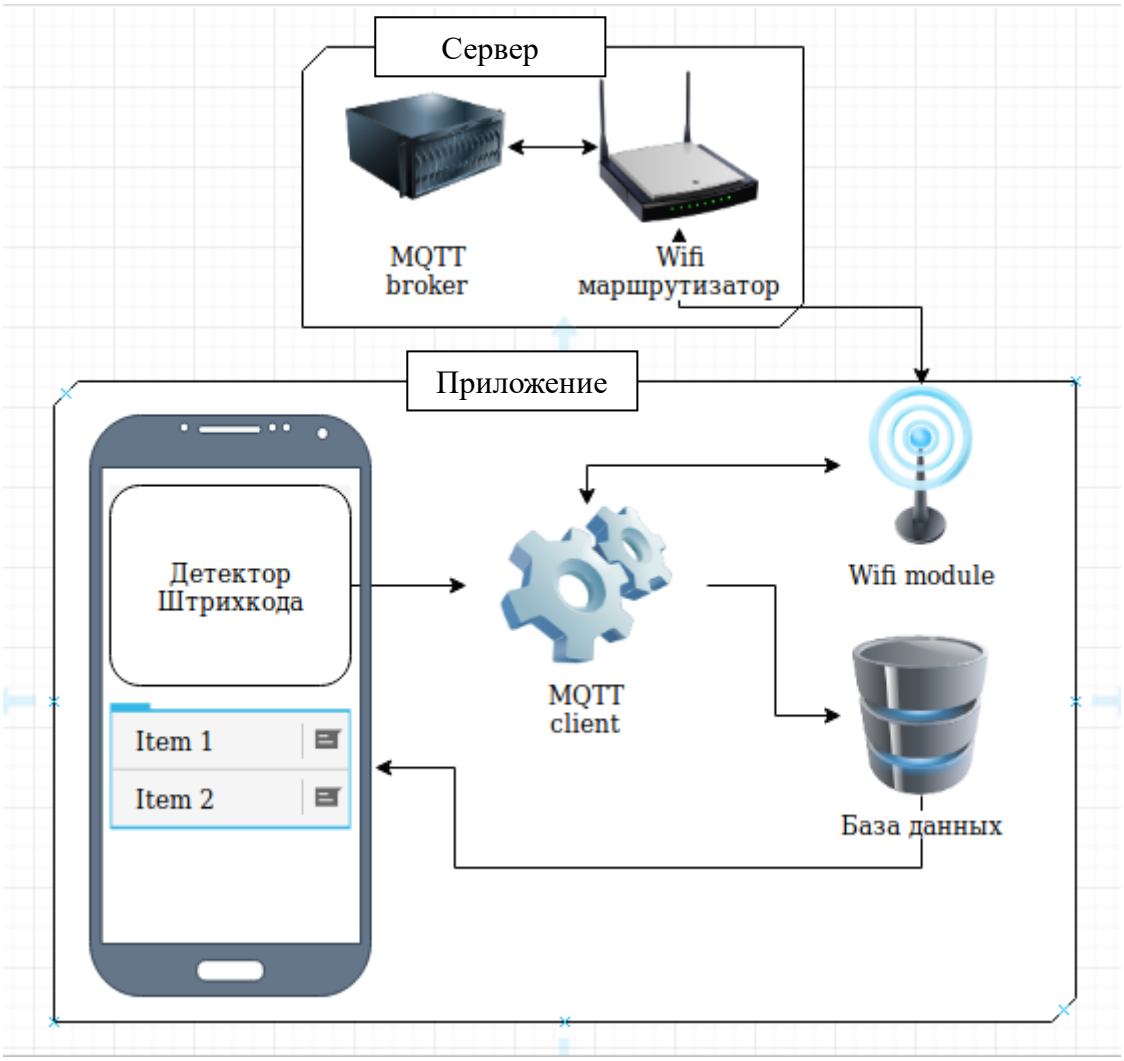


Рисунок 3.1 – Структурная схема системы

На этапе проектирования необходимо построить алгоритм работы с приложением определить основные функциональные модули, каждый из которых будет реализовывать функции определённые в алгоритме.

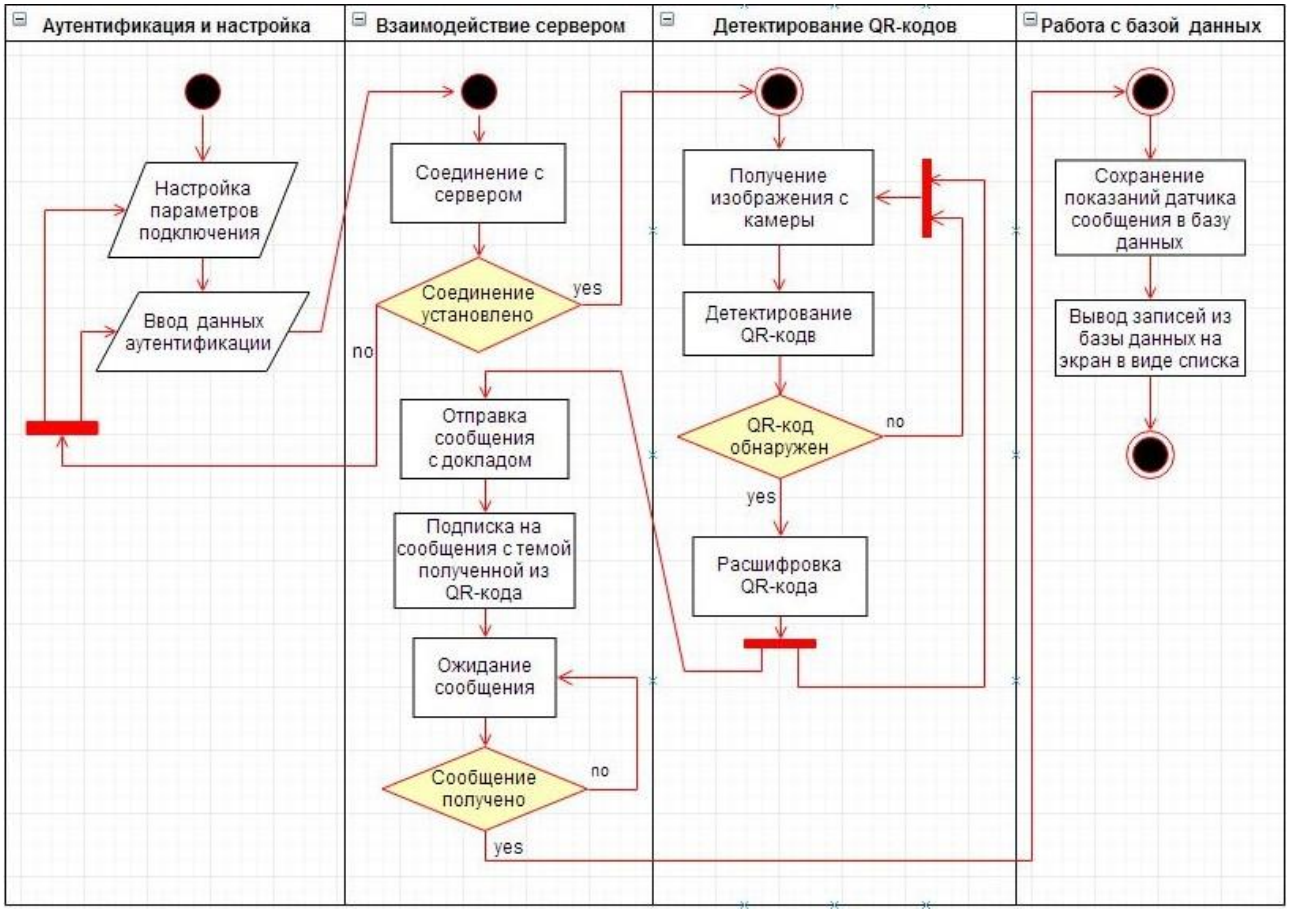


Рисунок 3.2 – Общий алгоритм работы приложения

MQTT — модуль, который будет реализовывать подключение к брокеру, передачу сообщений с докладом о идентификации датчика, получение сообщений с показаниями идентифицированных датчиков. Кроме того он позволит реализовать аутентификацию пользователя на стороне сервера.

QRDecoder.Camera — модуль, отвечающий за управление камерой и получение изображения с неё.

QRDecoder.Barcode — модуль детектирования и расшифровывания QR-кодов, полученных в виде изображений с камеры.

Database — модуль взаимодействия с базой данных SQLite

UI — модуль для реализации пользовательского интерфейса

3.2 Разработка интерфейса

Проектирование приложения включает в себя работу по планированию навигации и функционала. На этом этапе определяется внешний вид приложения и основные элементы управления.

Основная задача интерфейса приложения – предоставить пользователю функционал для максимально быстрого и лаконичного достижения его целей. Исходя из постановки задачи цели пользователя:

- Аутентификация
- Настройка подключения к серверу
- Идентификация датчиков
- Получение и визуализация показаний с идентифицированных датчиков
- Сохранение показаний датчика
- Отправка доклада о том, что датчик был осмотрен

Непосредственного взаимодействия пользователя с приложением требуют четыре цели: аутентификация, настройка подключения к серверу, идентификация датчиков и визуализация показаний их показаний.

Получение и сохранение показаний датчика, отправка доклада о том, что датчик был осмотрен, будет осуществляться сразу после идентификации датчика без необходимости дополнительных действий от пользователя.

На основании этого мною была составлена навигационная карта приложения:

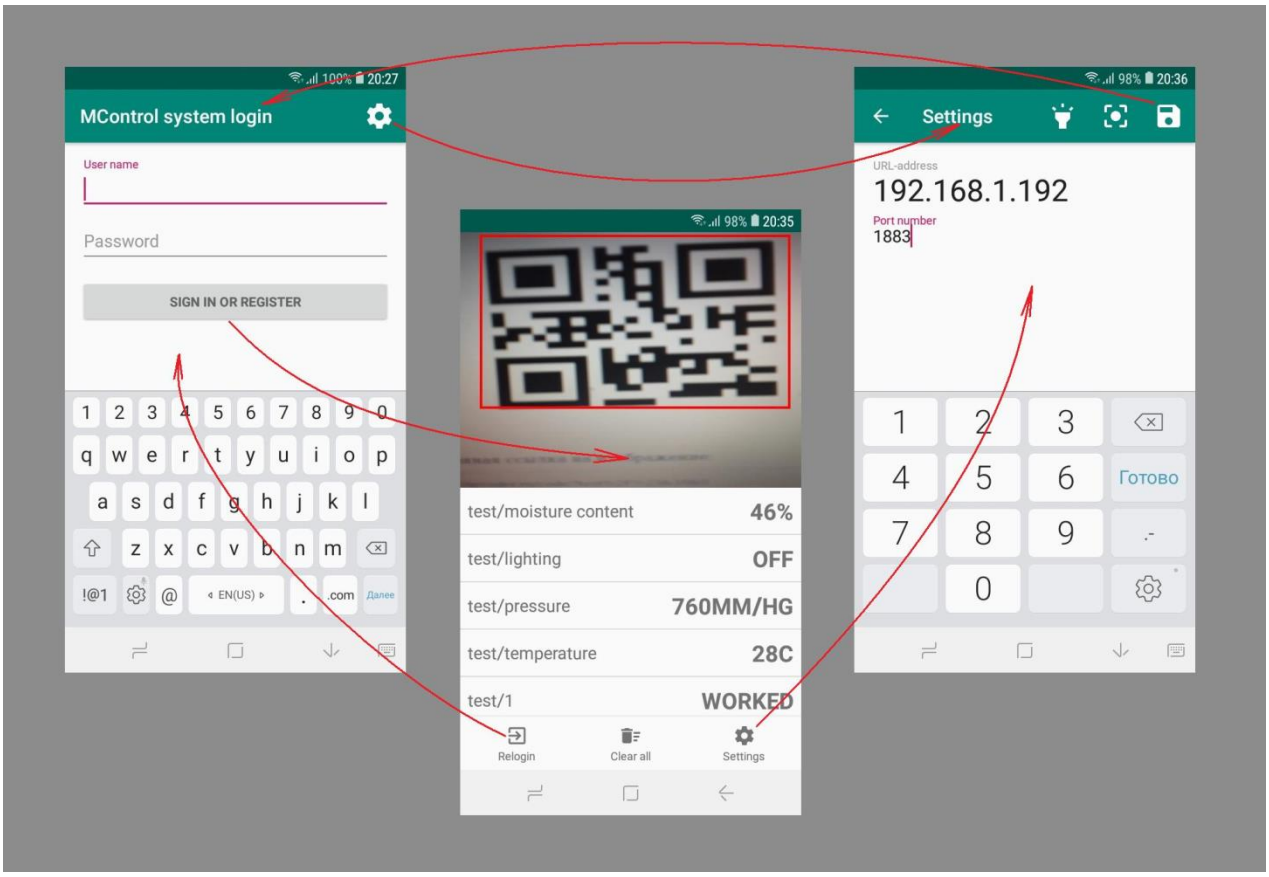


Рисунок 3.3 – Навигационная карта приложения

Навигационная карта схематично отражает все экраны, которые будут использованы в приложении, распределение информации на них. Организация схемы переходов показана стрелками.

Рассмотрим элементы более подробно:

3.2.1 Экран аутентификации

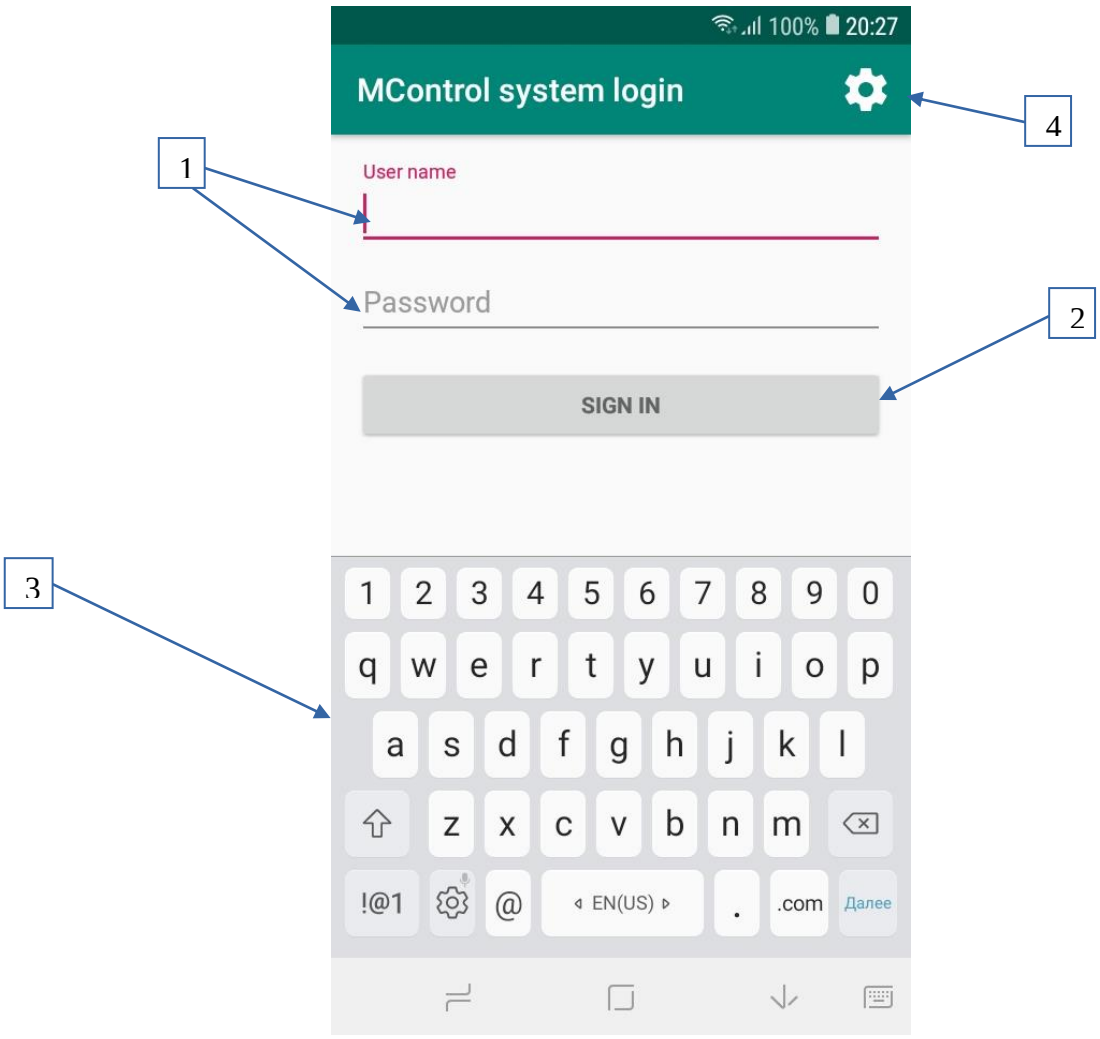


Рисунок 3.4 – Экран аутентификации

Экран аутентификации (Рис 3.4) содержит текстовые поля(1) для ввода имени пользователя и пароля, которые используются для аутентификации пользователя в системе, кнопка «Sign in»(2) при нажатии которой происходит попытка соединиться с сервером, виртуальная клавиатура(3) для ввода значений в текстовые поля. Также в верхнем меню присутствует кнопка перехода, к экрану настроек оформленная в виде значка шестерёнки(4).

Если после нажатия на кнопку «Sign in» соединение с сервером не будет установлено, на экран будет выведено сообщение об ошибке с предложением проверить настройки соединения. Для настройки соединения необходимо нажать на знак в виде шестерёнки в углу экрана. Произойдёт переход на экран настройки.

3.2.2 Экран настройки

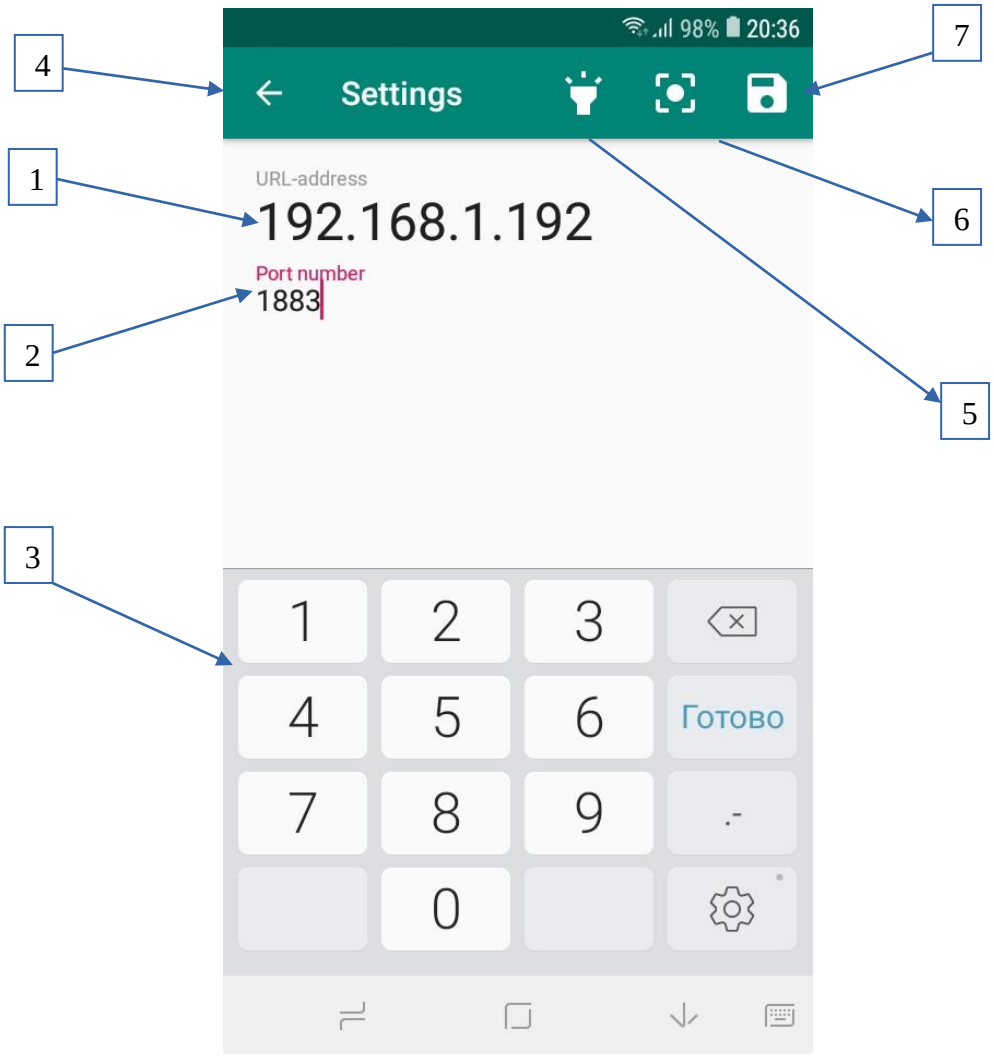


Рисунок 3.5 – Экран настройки приложения

На экране настройки также присутствуют два текстовых поля:

- Поле для ввода URL — адреса(1).
- Поле для ввода номера порта(2).

Виртуальная клавиатура(3) зависит от текстового поля. Если активно поле URL — адреса, выводится полноценная клавиатура с основными буквенное-символьными элементами. Если активно поле для ввода номера порта — выводится только цифровая клавиатура.

В верхнем меню активируются следующие пункты

«Back arrow» (4) — осуществляет возврат на предыдущий экран.

«lighting»(5) — активирует подсветку с помощью камеры (если такой поддерживается устройством).

«Autofocus»(6) — активирует режим автофокусировки камеры (если такой поддерживается устройством).

«Save»(7) — сохраняет введённые настройки.

3.2.3 Основной экран приложения

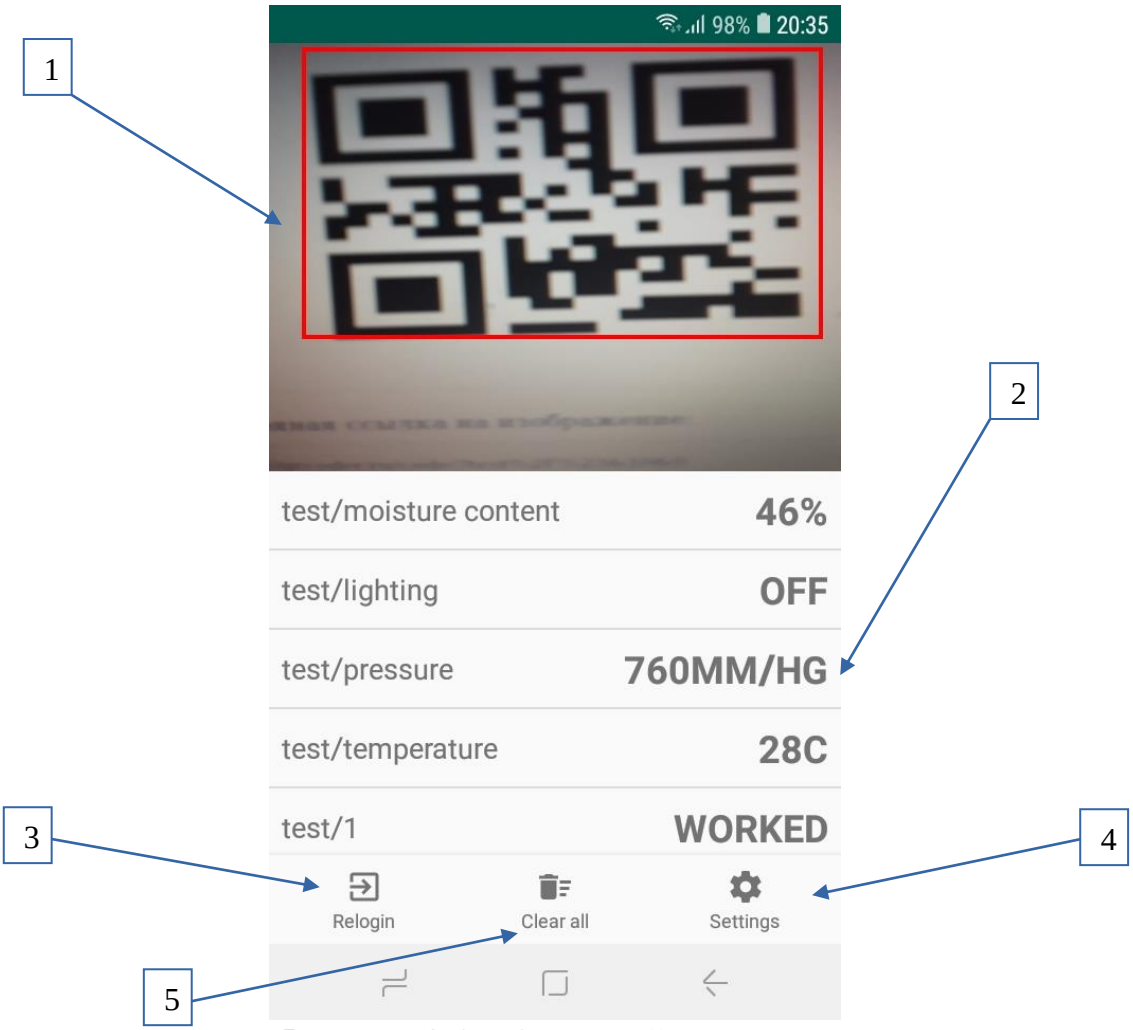


Рисунок 3.6 – Основной экран приложения

Основной экран приложения объединяет область для идентификации датчика и список.

Для идентификации датчика используется изображение с камеры(1), которую пользователь направляет на QR - код. Ниже отображается список идентифицированных датчиков(2) в порядке обратном времени идентификации. Каждый элемент списка содержит название датчика и его показания.

Внизу экрана расположено горизонтальное меню, которое состоит из пунктов:
- «Relogin» - переход на экран аутентификации;
- «Settings» - переход на экран настроек;
- «Clear all» - удаление всех записей и отключение запросов.

Кроме того специальные движения позволяют регулировать zoom на камере, а продолжительное нажатие на элемент списка вызывает контекстное меню, через которое можно удалить данный элемент.

3.3Разработка хранилища и проектирование базы данных

3.3.1 Хранение настроек

Информация в мобильном приложении должна храниться в специальном хранилище, доступном только этому приложению.

В файле SharedPreferences будут созданы поля вида:

```
<map>
        <string name="Ключ">Значени</string>
</map>
```

Ключи и значения параметров будут задаваться при старте приложения и хранится в закрытом хранилище. Планируется использовать следующие ключи:

autoFocus — отвечает за использование автофокусировки, принимает значения:

ON— автофокусировка активирована

OFF— автофокусировка деактивирована

useFlash — отвечает за включение / выключение подсветки камеры, принимает значения:

ON— подсветка камеры включена

OFF— подсветка камеры выключена

url_address— соответствует значению url-адреса сервера;

port_number— соответствует значению номера порта;

3.3.2 Хранение показаний датчиков

Для хранения информации о показаниях датчиков приложение использует базу данных SQLite.

SQLite позволяет реализовать реляционную модель данных.

Реляционная модель – это модель, которая описывает, как организовать данные в таблицах и как определить связи между этими таблицами. Ниже приведены некоторые характеристики реляционных баз данных и реляционной модели данных.

Использование ключей. Каждая строка данных в таблице идентифицируется уникальным “ключом”, который называется первичным ключом. Зачастую, первичный ключ это автоматически увеличиваемое (автоинкрементное) число (1,2,3,4 и т.д). Данные в различных таблицах могут быть связаны вместе при использовании ключей. Значения первичного ключа одной таблицы могут быть добавлены в строки (записи) другой таблицы, тем самым, связывая эти записи вместе.

Отсутствие избыточности данных. В проекте базы данных, которая создана с учетом правил реляционной модели данных, каждый кусочек информации, например, имя пользователя, хранится только в одном месте. Это позволяет устранить необходимость работы с данными в нескольких местах. Дублирование данных называется избыточностью данных и этого следует избегать в хорошем проекте базы данных.

Ограничение ввода. Используя реляционную базу данных, можно определить, какой вид данных позволено сохранять в столбце. Можно создать поле, которое содержит целые числа, десятичные числа, небольшие фрагменты текста, большие фрагменты текста, даты и т.д.

Поддержание целостности данных. Настраивая свойства полей, связывая таблицы между собой и настраивая ограничения, можно увеличить сохранность и связность данных.

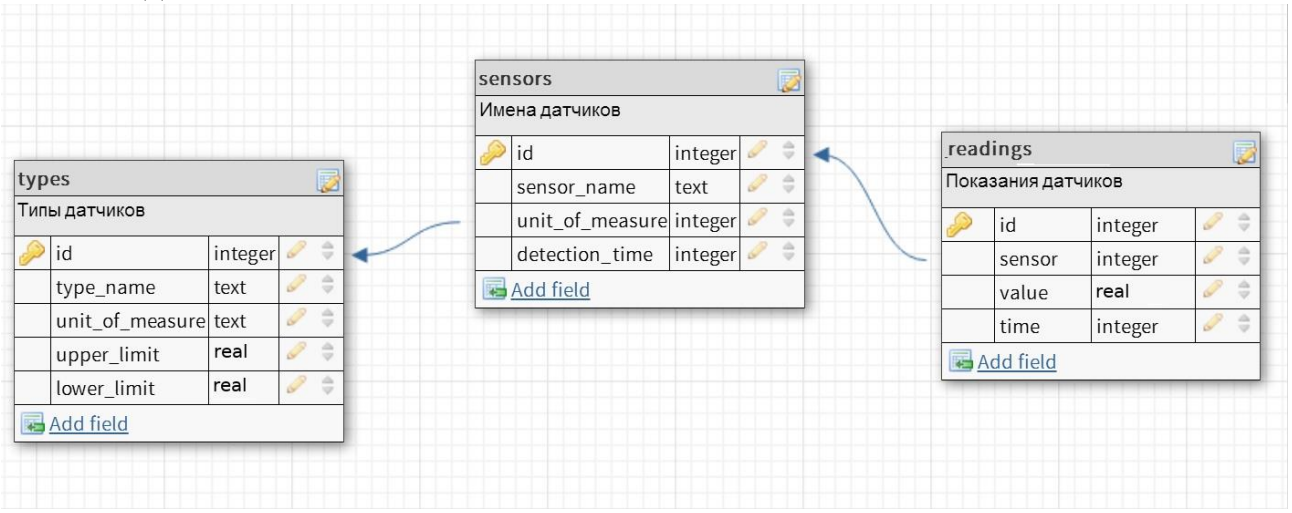


Рисунок 3.7 – Модель базы данных приложения

3.3.3 Структура базы данных приложения

База данных приложения будет состоять из трёх таблиц:

- types — содержит основные типы датчиков
- sensors — содержит уникальные имена датчиков
- readings — хранит показания датчиков

В таблице types, кроме поля id, которое является первичным ключом, присутствуют ещё четыре поля:

- type_neme — текстовое поле с названием типа датчиков
- unit_of_measure — текстовое поле с названием единиц измерения
- upper_limit — информация о верхней границеизмерений, тип real
- lover_limit — информация о нижней границеизмерений, тип real

В таблице sensors, помимо поля id, которое является первичным ключом, имеются три поля:

- sensor_name — текстовое поле с именем датчика
- unit_of_measure — foreign key соответствующий id из таблицы types требуется для сопоставления датчика и единиц измерения соответствующих данному типу датчика.
- detection time — время обнаружения и распознавания датчика, необходимо для сортировки данных при выводе в виде INT(long).

В третей таблице readings также присутствует поле id(первичный ключи), и три поля:

- sensor — foreign key соответствующий id из таблицы sensors,
- value — хранит значение полученное от сервера
- time — содержит значение времени с последнего обновления значения.

4 РЕАЛИЗАЦИЯ

4.1 Взаимодействие с сервером

Работа приложения построена на взаимодействии с сервером, поэтому первое что необходимо реализовать — подключение к MQTT брокеру.

4.1.1 Подключение библиотеки Paho Android Service

Android использует Gradle в качестве системы сборки и управления зависимостями, поэтому служба Paho Android Service может быть добавлена в приложение через Gradle. Чтобы добавить последнюю версию Paho Android Service в качестве зависимости к приложению, необходимо добавить следующие записи в файл gradle:

Листинг 4.1.1 — Подключение библиотеки Paho Android Service

```
repositories {
      maven {
        url "https://repo.eclipse.org/content/repositories/paho-snapshots/"
  }
}

dependencies {
  implementation('org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0')
  implementation('org.eclipse.paho:org.eclipse.paho.android.service:1.1.1')
}
```

Первая часть добавляет репозиторий выпуска Paho в конфигурацию gradle, чтобы Gradle мог найти упакованный Paho Android Service JAR. Вторая часть добавляет службу Paho Android Service для настройки зависимости приложения. Исключение библиотеки поддержки Android `exclude module: 'support-v4'` необходимо только в том случае, если приложение Android использует библиотеку поддержки Android для резервного копирования новых функций Android в более старые версии Android.

4.1.2Установка соединения

Paho Android Service инкапсулирует соединение MQTT и предлагает API для этого. Чтобы создать привязку к Paho Android Service, service должен быть объявлен в `AndroidManifest.xml`. Для этого нужно добавить в тег `<application>`следующую запись:

Листинг 4.1.2 — Объявление сервиса в манифесте приложения

```
<service android:name="org.eclipse.paho.android.service.MqttService">
</service>
```

Кроме этого для работы службы Paho Android необходимы следующие разрешения:

Листинг 4.1.3 — Разрешения для приложения

```
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

Для создания подключения к брокеру был спроектирован класс Connection.java, реализующий паттерн проектирования Singleton. Сокращенный код этого класса приведен в листинге 2.14.

Листинг 4.1.4 — Сокращенный код класса Connection.java

```
public class Connection {

    enum Status { DISCONNECT, CONNECT }
  private Status status = Status.DISCONNECT;

  private String clientId;
  private MqttAndroidClient client;
  private String clientUserName;

  private static Connection instance = null;

  private Connection() { }
  public static void initInstance() { }
  public static Connection getInstance() { }
  public Status getStatus() { }
  public String getClientUserName() { }
  public void connect(Application appContext, String serverURI,
  String userName, String password, Intent intent) { }
  public void publish(String topic, String payload) { }
  public void subscribe(Context appContext, String topic) { }
  public void unsubscribe(final String topic){ }
  public void disconnect() { }

  interface MessagesReceiver{
      void receivingMessage(String topic, MqttMessage message);
  }
      MessagesReceiver messagesReceiver;
      public void setMessagesListener(MessagesReceiver messagesListener){ }
      public void receiveMessages( ) { }
}
```

4.1.3 Основные методы для работы с MQTT брокером:

- connect;
- publish;
- subscribe;
- unsubscribe;
- disconnect;

Для подключения к брокеру используется метод connect;

Листинг 4.1.5 — Сокращённый код метода connect

```
{
    if(status == Status.CONNECT) { disconnect(); }
        String clientId = MqttClient.generateClientId();
        MqttAndroidClient client =
            new MqttAndroidClient(appContext, serverURI, clientId);
    try{
        IMqttToken token = client.connect();
        token.setActionCallback(new IMqttActionListener(){
    @Override
    publicvoidonSuccess(IMqttToken asyncActionToken){
                clientUserName = userName;

                // Здесь будет запускаться код для перехода на
                // основной экран.
    }

    @Override
    publicvoidonFailure(IMqttToken asyncActionToken, Throwable exception){
        // Соединение не уставлено!
        Log.d(TAG,"onFailure");

    }
        });
    }catch(MqttException e){
      e.printStackTrace();
    }
}
```

В первой строке вспомогательная функция клиента Paho MQTT используется для генерации случайного идентификатора пользователя. Вторая строка создает экземпляр клиента Android MQTT, который будет привязан к службе Paho Android.

При вызове метода connect, MqttAndroidClient клиент асинхронно попытается подключиться к брокеру MQTT и вернуть маркер. Этот маркер можно использовать для регистрации обратных вызовов, чтобы получать уведомления при подключении MQTT-соединения или возникновении ошибки.

Так как для подключения к брокеру требуется ввести имя пользователя и пароль, необходимо передать их в виде дополнительных опций:

Листинг 4.1.6 — Опции при установке соединения

```
// Connect with Username / Password
MqttConnectOptions options = new MqttConnectOptions();
options.setUserName(userName);
options.setPassword(password.toCharArray());
```

Где `userName` — строка с именем пользователя, а `password` — пароль.

Метод `publish` `MqttAndroidClient` позволяет публиковать сообщения с помощью метода `publish(topic, MqttMessage).` `MqttAndroidClient` не ставит в очередь все сообщения. Если клиент не подключен метод выдаст ошибку при попытке отправить сообщение. Когда клиент находится в автономном режиме `client.isConnected() == false`.

После первого подключения, когда соединение MQTT установлено успешно, вызывается метод `ImqttActionListener.onSuccess`.

Листинг 4.1.7 — Сокращённый метод `publish`

```
{
    byte[] encodedPayload =newbyte[0];
    try{
        encodedPayload = payload.getBytes("UTF-8");
        MqttMessage message =new MqttMessage(encodedPayload);
        client.publish(topic, message);
    }catch(UnsupportedEncodingException | MqttException e){
        e.printStackTrace();
    }
}
```

Для отправки сохраненного сообщения MQTT используется `MqttMessage.retained`атрибут может быть установлен в `true` через `MqttMessage.setRetained (true).`

`MqttAndroidClient.subscribe` метод принимает тему и QOS в качестве параметров и возвращает`IMqttToken`. Токен можно использовать для отслеживания успешной или неудачной подписки. Сокращённое тело метода приведёно в листинге 2.1.8.

Листинг 4.1.8 — Сокращённый метод `subscribe`

```
{
    int qos = 1;
    try{
        IMqttToken subToken = client.subscribe(topic, qos);
        subToken.setActionCallback(new IMqttActionListener(){
            @Override
            publicvoidonSuccess(IMqttToken asyncActionToken){

            }

            @Override
            publicvoidonFailure(IMqttToken asyncActionToken,
                                                    Throwable exception){
            // Не удалось подписаться не тему
            }
        });
    }catch(MqttException e){
        e.printStackTrace();
    }
}
```

Обратным методу `subscribe` является метод `unsubscribe`.

Листинг 4.1.9 — Сокращённый метод `unsubscribe`

```
{
    try{
        IMqttToken unsubToken = client.unsubscribe(topic);
        unsubToken.setActionCallback(new IMqttActionListener(){
            @Override
            publicvoidonSuccess(IMqttToken asyncActionToken){

            }
            @Override
            publicvoidonFailure(IMqttToken asyncActionToken,
                                                Throwable exception){
            }
        });
    }catch(MqttException e){
        e.printStackTrace();
    }
}
```

Для отключения клиента необходимо вызвать метод disconnect объекта MqttAndroidClient. В приведенном ниже примере сохраняется токен, возвращенный методом disconnect, и добавляется IMqttActionListener для получения уведомления об успешном отключении клиента.

Листинг 4.1.10 — Сокращённый метод `disconnect`

```
{
    try{
        IMqttToken disconToken = client.disconnect();
        disconToken.setActionCallback(new IMqttActionListener(){
            @Override
            publicvoidonSuccess(IMqttToken asyncActionToken){

            }
            @Override
            publicvoidonFailure(IMqttToken asyncActionToken,
                Throwable exception){
            }
        });
    }catch(MqttException e){
        e.printStackTrace();
    }
}
```

В классе Connection.java определен статический экземпляр этого же класса. В методе getInstance(), если этот экземпляр еще не был инициализирован, то вызывается конструктор. Сам конструктор определен как private, а значит, мы не можем вне этого класса создать экземпляр с помощью оператора new.

Инициализированный объект класса Connection.java должен быть доступен из любой части приложения. Вполне очевидным решением для этого является создание экземпляра класса Connection.java (первый вызов getInstance()) в какой-либо Activity, которая будет жить на протяжении всего времени работы приложения. Но система android не может гарантировать, что неактивная Activity не будет уничтожена в случае нехватки памяти. Поэтому написать реализацию класса Application.java. Класс Application - это базовый класс приложения Android. При запуске программы вначале создается экземпляр этого класса, а потом уже необходимые деятельности.

Листинг 4.1.11 — Класс MControlApplication.java

```
public class MControlApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        Connection.initInstance();
    }
}
```

Также надо прописать класс MControlApplication в AndroidManifest.xml

Листинг 4.1.12 — Класс MControlApplication.javав AndroidManifest.xml

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:name=".MControlApplication">
    ...
</application>
```

4.2 Идентификация датчиков

Идентификатором датчика в системе будет уникальное имя датчика, ввод которого будет осуществлён с помощью камеры устройства. Для управления камерой и вывода изображения с неё на экран, специалистами Google был спроектирован модуль, состоящий из классов:

- CameraSource.java — предоставляет управление камерой для получения предварительного просмотра. Получает изображение предварительного просмотра с камеры с определённой частотой, отправляя кадры детектору по мере того как он их обрабатывать. Кадры могут быть отброшены, если детектор не успевает обрабатывать все кадры, генерируемые камерой. Частоту кадров можно указать с помощью метода setRequestedFps (float)

- CameraSourcePreview.java — отвечает за отображение кадров предварительного просмотра камеры на экран.

- GraphicOverlay.java — отображает графические объекты поверх связанного предварительного просмотра камеры. Представление, отображающее серию пользовательских графических объектов, накладываемых поверх связанного предварительного просмотра (т. е. камеры). Позволяет добавлять графические объекты, обновлять их и удалять их, вызывая и отключая видимость соответствующего рисунка в пределах view. Поддерживает масштабирование и зеркальное отображение графики относительно свойств предварительного просмотра камеры. Размеры элементов выражаются в значениях размера предварительного просмотра, но должны быть масштабированы к полному размеру экрана.

Классы для расшифровки QR-кодов, был также выделены в отдельный модуль — Barcode:

- Класс BarcodeTrackerFactory.java реализует паттерн «Фабрика» и используется для создания трекеров штрих-кода — по одному для каждого штрих-кода.

- Класс BarcodeGraphicTracker.java это трекер, который используется для обнаружения штрихкодов на экране, и их отслеживания для наложения графики, а также удаления графики, когда штрих-код покидает зону видимости.

- Класс BarcodeGraphic.java используется для отрисовки экземпляра накладываемого на штрих-код изображения с учетом его положения, размера и идентификатора.

Эти классы используют интерфейсы Mobile Vision, которые обеспечивают API для обнаружения штрих-кода.

4.3 Взаимодействие с базой данных

Работу с БД вынесем в отдельный класс DataBase.java. В нём будет описано создание БД, управление подключением и методы по чтению, добавлению, обновлению и удалению записей.

Листинг 4.3.1 — Основные поля и методы класса DataBase.java

```
public class DataBase {
    private final Context mCtx;
    private DBHelper mDBHelper;
    private SQLiteDatabase mDB;

    public DataBase(Context ctx) { }
    public void open() { }
    public void close() { }
    public String getName(long id) { }
    public Cursor getAllData() { }
    public String[] getAllNames() { }
    public void addRec(String name, String value) { }
    public  int update(String name, String value) { }
    public void delRec(long id) { }
    public void delRec(String name) { }
    public void delAll() { }

    private class DBHelper extends SQLiteOpenHelper { }
}
```

Где:

- Context mCtx — содержит контекст Activity или Aplication создавшего объект класса.
- mDBHelper — хранит ссылку на объект DBHelper
- mDB — хранит объект SQLiteDatabase, который позволяет работать с базой данных
- open() - открыть подключение к БД
- close() - закрыть подключение к БД
- getName(long id) — получить по id имя датчика в виде строки
- getAllData() - получить все данные из таблиц БД в виде объекта типа Cursor
- getAllNames() – получить все имена в виде массива типа String
- addRec(String name, String value) добавить запись в БД
- update(String name, String value) - обновить запись в БД
- delRec(long id) - удалить запись из БД по id
- delRec(String name) - удалить запись из БД по полю name
- delAll() - удалить все записи из БД
- class DBHelper extends SQLiteOpenHelper - класс по созданию и управлению БД

При подключении к БД необходимо указать имя БД и версию. При этом могут возникнуть следующие ситуации:

1) БД не существует. Это может быть, например, в случае первичной установки программы. В этом случае приложение должно само создать БД и все таблицы в ней. И далее оно уже работает с только что созданной БД.

2) БД существует, но ее версия устарела. Это может быть в случае обновления программы. Например, новой версии программы нужны дополнительные поля в старых таблицах или новые таблицы. В этом случае приложение должно обновить существующие таблицы и создать новые, если это необходимо.

3) БД существует и ее версия актуальна. В этом случае приложение успешно подключается к БД и работает.

Для обработки описанных выше ситуаций, нужно создать вложенный в класс DBHelper, являющийся наследником для SQLiteOpenHelper. Этот класс предоставит методы для создания или обновления БД в случаях ее отсутствия или устаревания:

Листинг 4.3.2 — Методы класса DBHelper.java

```
private class DBHelper extends SQLiteOpenHelper {
        public DBHelper(Context context, String name,
                        SQLiteDatabase.CursorFactory factory, int version) { }
        public void onCreate(SQLiteDatabase db) { }
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) { }
}
```

onCreate - метод, который будет вызван, если БД, к которой мы хотим подключиться – не существует

onUpgrade - будет вызван в случае, если мы пытаемся подключиться к БД более новой версии, чем существующая

4.4 Реализация пользовательского интерфейса

Для управления визуализацией данных приложения и определения действий, которые может производить пользователь. В Android-приложении должен быть как минимум один класс, который был расширен (extends) от родительского класса Activity. Почти все экземпляры и разновидности Activity напрямую взаимодействуют с пользователем, так что класс Activity отвечает за создание окна, в котором можно разместить визуальный интерфейс GUI вызовом setContentView(View).

На этапе проектирования были определены три экрана GUI:

- Экран аутентификации
- Экран настройки
- Основной экран приложения.

В приложении каждому из них будут соответствовать Activity из UI-модуля

4.4.1 LoginActivity

При запуске приложения первым будет отображаться экран аутентификации. За его отвечает класс LoginActivity.java, наследуемый от AppCompatActivity, методы которого приведены в листинге 4.4.1:

Листинг 4.4.1 — Поля и методы класса LoginActivity.java

```
public class LoginActivity extends AppCompatActivity {
    private Connection connection;
    private EditText mPasswordView, mUserName;
    Button btnLogin;

    protected void onCreate(Bundle savedInstanceState) { }
    private String getURI() { }
    public boolean onCreateOptionsMenu(Menu menu) { }
    public boolean onOptionsItemSelected(MenuItem item) { }
    public boolean setOnClickListener(View view) { }
}
```

Где:

- `connection` — содержит ссылку на объект класса `Connection;`
- `mUserName` — поле для ввода имени пользователя;
- `mPasswordView` — поле для ввода пароля;
- `btnLogin` — кнопка на экране деля подтверждения ввода;
- `onCreate(Bundle savedInstanceState)` — метод вызываемый системой при создании Activity. В нём проводится инициализация всех полей класса, а также вызов методов отвечающих за отображение элементов экрана. Код метода в в листинге 4.4.2;
- `getURI()` - вспомогательный метод для получения и оформления URL-адреса из значений текстовых полей;
- `onCreateOptionsMenu(Menu menu)` — метод вызываемый системой при создании Activity для отрисовки элементов верхнего меню
- `onOptionsItemSelected(MenuItem item)` — callback-метод для обработки нажатий пользователем на элементы меню.

Листинг 4.4.2 —Метод `onCreate` класса LoginActivity.java

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

    ActionBar actionBar = getSupportActionBar();
    actionBar.setTitle(R.string.mc_login);

    connection = Connection.getInstance();
    if(connection.getStatus() == Connection.Status.CONNECT) {
        actionBar.setHomeButtonEnabled(true);
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
```

```
    mUserName = (EditText) findViewById(R.id.txtUserName);
    mPasswordView = (EditText) findViewById(R.id.txtPassword);

    btnLogin = (Button) findViewById(R.id.btnLogin);
    btnLogin.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
            String UserName = mUserName.getText().toString();
            String Pwd = mPasswordView.getText().toString();
            if((mUserName.length() != 0) && (mPasswordView.length() != 0)){
                Intent MainIntent = new Intent(LoginActivity.this,
                        MainActivity.class).addFlags(
                        Intent.FLAG_ACTIVITY_CLEAR_TASK).addFlags(
                        Intent.FLAG_ACTIVITY_NEW_TASK);
                connection.connect((Application) getApplicationContext(),
                                                    getURI(), UserName, Pwd, MainIntent);
            }
        }
    });
}
```

Вызов `setContentView(R.layout.activity_login)` в первой строке этого методаустанавливает содержимое Activity из layout-файла. Но в качестве аргумента мы указываем не путь к layout-файлу (`res/layout/activity_login.xml`), а константу, которая является ID файла. Эта константа генерируется автоматически в файле R.java.

Листинг 4.4.3 — основные элементы layout-файлаactivity_login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <android.support.design.widget.TextInputLayout>
        <EditText    android:id="@+id/txtUserName"/>
    </android.support.design.widget.TextInputLayout>

    <android.support.design.widget.TextInputLayout>
        <EditText    android:id="@+id/txtPassword"/>
    </android.support.design.widget.TextInputLayout>

    <Button android:id="@+id/btnLogin" />
</LinearLayout>
```

Анологичным образом инициализируются остальные элементы экрана.

Выражение `btnLogin = (Button) findViewById(R.id.btnLogin);` инициализирует кнопку `btnLogin`. В следующей строке определяется `setOnClickListener(View view)` – callback-метод для обработки нажатий пользователем на кнопку `btnLogin`.

При нажатии на кнопку выполняется попытка создать соединение, при успешном соединении будет запущено `MainActivity`. Для этого в метод `connect` в классе `Connection` передаётся `Intent` и при успешном подключении вызывается метод `startActivity(Intent)`.

При выборе в верхнем меню пункта `Settings` обработчик нажатий `onCreateOptionsMenu(Menu menu)` создаст `Intent` с вызовом `SettingsActivity` и вызовет метод `startActivity(Intent)`, который запустит `SettingsActivity`.

4.4.2 SettingsActivity

Для управления экраном настроек предназначен класс `SettingsActivity.java`. Он также как и `LoginActivity` наследуется от `AppCompatActivity`.

Листинг 4.4.4 — Поля и методы класса LoginActivity.java

```
public class SettingsActivity extends AppCompatActivity {
        private EditText url_address, port_number;

        protected void onCreate(Bundle savedInstanceState) {}
        private void getSettings() { }
        private void saveSettings() { }
        public boolean onCreateOptionsMenu(Menu menu) { }
        public boolean onOptionsItemSelected(MenuItem item) { }
        private void setOnFocusChangeListener(final EditText editText) { }
}
```

Где:

- `url_address, port_number` — поля для ввода URL-адреса и номера порта
- `onCreate(Bundle savedInstanceState)` — метод вызываемый системой при создании Activity. В нём проводится инициализация всех полей класса, а также вызов методов отвечающих за отображение элементов экрана.
- `getSettings()` - метод для получения настроек из файла SharedPreferences
- `saveSettings()` - метод для сохранения настроек в файл SharedPreferences
- `onCreateOptionsMenu(Menu menu)` — метод вызываемый системой при создании Activity для отрисовки элементов верхнего меню
- `onOptionsItemSelected(MenuItem item)` — callback-метод для обработки нажатий пользователем на элементы меню.

Настройки вносятся пользователем в текстовые поля, при выбора пункта меню Save настройки сохранятся в SharedPreferences, и запускается LoginActivity. При повторном запуске SettingsActivity данные из файла SharedPreferences загружаются в соответствующие поля, чтобы пользователь мог их отредактировать.

4.4.3 MainActivity

В файле activity_main.xml определяется расположение элементов основного экрана приложения.

Листинг 4.4.5 — основные элементы layout-файла ctivity_main.xml

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/container">

    <LinearLayout
        <ru.ngtu.mcontrol.qrdecoder.camera.CameraSourcePreview
            android:id="@+id/preview">
            <ru.ngtu.mcontrol.qrdecoder.camera.GraphicOverlay
android:id="@+id/graphicOverlay"/>
        </ru.ngtu.mcontrol.qrdecoder.camera.CameraSourcePreview>

        <ListView
            android:id="@+id/lvData"
        </ListView>
    </LinearLayout>

    <android.support.design.widget.BottomNavigationView
android:id="@+id/navigation"
app:menu="@menu/navigation" />

</android.support.constraint.ConstraintLayout>
```

В верхней части экрана будут отображаться кадрыпредварительного просмотра с камеры(элемент с `id="@+id/preview"`). С их помощью пользователь сможет сканировать штрихкод. Поверх изображения с камеры будут отрисованы графические элементы, в виде рамок создаваемые в классе `GraphicOverlay`.

Под превью камеры расположен элемент `ListView`. `ListView` представляет собой прокручиваемый список элементов. Очень популярен на мобильных устройства из-за своего удобства. В нём будет отображаться список датчиков полученных из базы данных.

Наконец в самом низу расположено `BottomNavigationView` — нижнее меню навигации. С помощью его элементов можно будет переходить на другие экраны или же управлять списком.

`Main Activity.java` — это класс, который отвечает за работу основного экрана приложения. В нем активируется камера устройства, превью которой отображается в верхней части экрана, запускается процесс поиска и расшифровки QR-кодов, ListView заполняется элементами из базы данных.

Листинг 4.4.6 — Поля и методы класса LoginActivity.java

```
public class MainActivity extends FragmentActivity implements
        BarcodeGraphicTracker.BarcodeUpdateListener,
    LoaderManager.LoaderCallbacks<Cursor>, Connection.MessagesReceiver {
    private Application appContext;
    private ListView lvData;
    private DataBase dataBase;
    private SimpleCursorAdapter scAdapter;
    private Connection connection;
    private CameraSource mCameraSource;
    private CameraSourcePreview mPreview;
    private GraphicOverlay<BarcodeGraphic> mGraphicOverlay;
    private ScaleGestureDetector scaleGestureDetector;
    private GestureDetector gestureDetector;
    protected void onCreate(Bundle savedInstanceState) { }
    private void requestCameraPermission() { }
    public boolean onTouchEvent(MotionEvent e) { }
    private void createCameraSource(boolean autoFocus, boolean useFlash) { }
    protected void onResume() { }
    protected void onPause() { }
    protected void onDestroy() { }
    public void onRequestPermissionsResult(int requestCode,
    String[] permissions, int[] grantResults) { }
    private void startCameraSource() throws SecurityException { }
    private boolean onTap(float rawX, float rawY) { }
    private class CaptureGestureListener extends {
        GestureDetector.SimpleOnGestureListener {
            public boolean onSingleTapConfirmed(MotionEvent e) { }
        }
    }
    private class ScaleListener implements
        ScaleGestureDetector.OnScaleGestureListener {
            public boolean onScale(ScaleGestureDetector detector) { }
            public boolean onScaleBegin(ScaleGestureDetector detector) { }
            public void onScaleEnd(ScaleGestureDetector detector) { }
    }
    public void onBarcodeDetected(Barcode barcode) { }
    public void receivingMessage(String topic, MqttMessage message) { }
    public Loader<Cursor> onCreateLoader(int id, Bundle bndl) { }
    public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) { }
    public void onLoaderReset(Loader<Cursor> loader) { }
    static class MyCursorLoader extends CursorLoader {
            DataBase db;
            public MyCursorLoader(Context context, DataBase db) {      }
                public Cursor loadInBackground() { }
        }
    public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenu.ContextMenuInfo menuInfo) { }
```

```
public boolean onContextItemSelected(MenuItem item) { }

    private BottomNavigationView.OnNavigationItemSelectedListener
mOnNavigationItemSelectedListener
= new BottomNavigationView.OnNavigationItemSelectedListener() {
        public boolean onNavigationItemSelected(MenuItem item) { }
    };
}
```

Для детектирования QR-кодов класс MainAcrivity.java реализует интерфейс `BarcodeGraphicTracker.BarcodeUpdateListener`, а так же определяет следующие методы:

- `requestCameraPermission`() - обрабатывает запрос на разрешения использования камеры.
- `onTouchEvent(MotionEvent e)` — обрабатывает касания по области предварительного просмотра;
- `createCameraSource(boolean autoFocus, boolean useFlash)` - Создает и запускает камеру. В этом же методе создается BarcodeDetector для отслеживания штрихкодов. С ним связывается `MultiProcessor.Builder` для получения результатов обнаружения штрих-кода, отслеживания штрихкодов и графического обозначения каждого штрих-кода на экране;
- `onResume`() - метод вызываемый системой при возобновлении работы Activity. В нём же перезапускается камера;
- `onPause()` - метод вызываемый системой при сворачивании Activity с экрана. В нём же останавливается работа камеры;
- `onDestroy`() Высвобождает ресурсы, связанные с работой камеры, соответствующими детекторами;
- `onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)` - Обратный вызов для результата запроса разрешений. Этот метод вызывается для каждого вызова `requestPermissions (String[], int);`
- `startCameraSource() throws SecurityException` Запуск или перезапуск фотокамеры, если она есть. Если камер отсутствует (например, если onResume был вызван до создания `CameraSource`), будет вызван этот метод.
- `onTap(float rawX, float rawY) возвращает результат(boolean)` обнаружения QR-кода в вызвавшее Activity
- `class CaptureGestureListener extends GestureDetector.SimpleOnGestureListener {`
- `onSingleTapConfirmed(MotionEvent e)` — обрабатывает касания по экрану

}

- `class ScaleListener implements`

ScaleGestureDetector.OnScaleGestureListener {

- onScale(ScaleGestureDetector detector) - Реагирует на события для выполняемого жеста масштабирования. Сообщается движением указателя.

- onScaleBegin(ScaleGestureDetector detector) – Реагирует на начало жеста масштабирования.

- onScaleEnd(ScaleGestureDetector detector) - Реагирует на завершение жеста масштабирования.

- onBarcodeDetected(Barcode barcode) - callback-метод, вызываемый при обнаружении и расшифровывании QR-кода

Интерфейс Connection.MessagesReceiver класс MainActivity.java реализует для взаимодействия с MQTT-брокером с помощью обратного вызова метода

public void receivingMessage(String topic, MqttMessage message) - вызывается при получении сообщения от брокера. Сохраняет данные в базу, вызывает метод для обновления отображения списка

Для того чтобы отобразить содержимое БД в виде списка, класс MainActivity.javaнаследуется от FragmentActivity и реализует интерфейс LoaderManager.LoaderCallbacks<Cursor>

- onCreateLoader(int id, Bundle bndl) - создаётзагрузчик и даем ему на вход объект для работы с БД.

- onLoadFinished(Loader<Cursor> loader, Cursor cursor) - выполняется по окончании загрузки данных. В нём получаем результат работы загрузчика – новый курсор с данными. Этот курсор мы отдаем адаптеру методом swapCursor.

- public void onLoaderReset(Loader<Cursor> loader) - вызывается при «сбросе» состояния загрузчика. Здесь данные обнуляются, и нам нужно удалить все имеющиеся ссылки на них.

Чтобы связать данные со списком, используется CursorLoader. CursorLoader представляет собой наследника класса AsyncTaskLoader<Cursor> и по умолчанию настроен на работу с ContentProvider, т.к. при работе требует Uri. Здесь же он используется для работы с БД. Для этого необходимо его расширить и добавить свою реализацию в MainActivity:

```
static class MyCursorLoader extends CursorLoader {
        public MyCursorLoader(Context context, DataBase db) { }
        public Cursor loadInBackground() { }
    }
```

- onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) – создание контекстного меню для элемнтов списка

- onContextItemSelected(MenuItem item) - реализуем удаление записи из БД. И после удаления снова запрашиваем у загрузчика новый курсор с данными.

4.5 Создание тестового сервера

Для проверки работоспособности приложения потребуется:
- Компьютер с возможностью подключения к сети и сетевой ОС;
- Маршрутизатор WiFi;
- MQTT - брокер;
- Скрипт для моделирования генерации показаний датчиков;
- QR-коды с зашифрованными в них именами датчиков.

4.5.1 Выбор операционной системы

В качестве ОС была выбрана система на основе ядра Linux.

Во-первых это ПО с открытым исходным, что обеспечивает лучшую защиту и переносимость на все типы компьютеров. Во-вторых такие системы уже применяются в устройствах, которое работает как центральный контроллер в системе автоматизации или мониторинга. Например: контроллеры **Wiren Board**[16], которые применяются в задачах мониторинга серверного и климатического оборудования, диспетчеризации и сбора данных с приборов учёта, в качестве основы для «умного дома». Контроллеры могут работать в составе облачной платформы «WB Cloud». Оборудование производится на заводе в России.

Рис. 4.1 контроллер Wiren Board

4.5.2 Установка MQTT-сервера

Для организации передачи сообщений по сети необходимо установить Mosquitto – это популярный MQTT-сервер (или брокер). Он прост в установке и настройке и активно поддерживается сообществом разработчиков.

Устанавливаем сервер и клиент с помощью консольных команд

```
sudo apt-get install mosquitto
sudo apt-get install mosquitto-clients
```

По умолчанию сервис Mosquitto запускается сразу после установки.

Для настройки пароля в консоль вводим:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd majordomo
sudo nano /etc/mosquitto/conf.d/default.conf
```

В открывшемся файле необходимо указать путь к файлу с именем пользователя и хэшем пароля

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd majordomo
sudo nano /etc/mosquitto/conf.d/default.conf
```

После сохранения файла, нужно перезапустить сервер

```
sudo systemctl restart mosquitto
```

4.5.3 Написание скрипта

Для моделирования работы системы был написан скрипт на языке bush:

Листинг 4.4.7 — Скрипт для генерации тестовых значений

```
#!/bin/bash
echo "Рестарт сервера"
sudo systemctl stop mosquitto
sleep 0.5
sudo systemctl start mosquitto
sleep 0.5
echo "Получим адрес сервера"
ipm="$(ip -4 addr show scope global | awk '$1 ~ /^inet/ {print $2}')"
ip="$(echo $ipm | awk -F"/"'{print $1}')"
echo "ip = "$ip

echo "Запускаем цикл генерации значений"
echo "---------------------------------"
n=32
i=0
while [ 1 ]
do
    let "number = ($RANDOM % 100)"
    number=0.7$number
    number+="Bar"
    mosquitto_pub -h $ip -t "ОНП/Маш.Зал/давление масла" -m $number -u "8host" -P "1234"

    let "number = ($RANDOM % 10 + 1495)"
    number+="об/с"
    mosquitto_pub -h $ip -t "ОНП/Маш.Зал/обороты" -m $number -u "8host" -P "1234"

    let "number = ($RANDOM % 10)"
    number=34.3$number
    number+="C"
    mosquitto_pub -h $ip -t "ОНП/Литейный цех/температура" -m $number -u "8host" -P
    "1234"

    let "number = ($RANDOM % 10)"
    number=2.5$number\
    number+="м/с"
    mosquitto_pub -h $ip -t "ОНП/Сборочный конвейер/скорость" -m $number -u "8host" -P
    "1234"
```

В бесконечном цикле скрипт генерирует случайные значения и посредством команды `mosquitto_pub` отправляет их на сервер, который перенаправляет эти сообщения клиентам, подписавшимся на соответствующие темы.

5 ТЕСТИРОВАНИЕ

5.1Тестирование программного продукта

Тестирование является одним из наиболее устоявшихся способов обеспечения качества разработки программного обеспечения и входит в набор эффективных средств современной системы обеспечения качества программного продукта.

Один из основных видов тестирования –функциональное тестирование. Под функциональным тестированием понимается проверка соответствия программного продукта функциональным требованиям, указанным в техническом задании на создание это продукта.

Для проведения функционального тестирования методика испытаний функционала приложения (ПМИ). Документ ПМИ содержит перечень сценариев тестирования программного продукта (`test cases`) с подробным описанием шагов. Каждый шаг сценария тестирования характеризуется действиями пользователя (специалиста по тестированию) и ожидаемыми результатами – ответной реакции программы на эти действия. Программа и методика испытаний обязана имитировать эксплуатацию программного продукта в реальном режиме.

Тестирование системы производилось в домашних условиях на мобильном телефоне Samsung J2 с использованием персонального компьютера с операционной системой Linux Mint с установленным сервером Mosquitto, с использованием заранее написанного скрипта, запускающего генерацию тестовых показаний датчиков. Для идентификации датчиков использовались заранее подготовленные QR-коды, в которых были зашифрованы их имена.

Характеристики смартфона **Samsung J2:**

Операционная система	Android
Процессор	
Производитель	Qualcomm
Тип	MSM8917
Количество ядер	4
Частота	1.4 ГГц
Память	
Объем оперативной памяти	1.5 ГБ
Объем встроенной памяти	16 ГБ

Сценарий 1.

Цель: проверка аутентификации на правильных данных.

Последовательность действий: запустить приложение, ввести имя, зарегистрированного на сервере пользователя и пароль в соответствующие поля на экране смартфона.

Ожидаемый результат: сообщение «Connect» и переход на основной экран.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден.

Сценарий 2.

Цель: проверка аутентификации на неправильных данных.

Последовательность действий: запустить приложение, ввести имя, незарегистрированного на сервере пользователя и пароль в соответствующие поля на экране смартфона.

Ожидаемый результат: сообщение «Connection failed!» и переход в ожидание ввода других данных.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден.

Сценарий 3.

Цель: проверка подключения при правильных настройках.

Последовательность действий: запустить приложение, открыть настройки через меню, ввести адрес тестового сервера и номер порта, выбрать пункт меню «Save» чтобы сохранить настройки, после перехода на экран аутентификации, ввести имя, зарегистрированного на сервере пользователя и пароль в соответствующие поля на экране смартфона.

Ожидаемый результат: сообщение «Connect» и переход на основной экран.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден.

Сценарий 4.

Цель: проверка сохранения настроек сервера.

Последовательность действий: запустить приложение, открыть настройки через меню, ввести адрес тестового сервера и номер порта, выбрать пункт меню «Save» чтобы сохранить настройки. Выйти из приложения и снова зайти на экран настроек.

Ожидаемый результат: введённые ранее настройки указаны в соответствующих полях.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден.

Сценарий 5.

Цель: проверка подключения при неправильных настройках.

Последовательность действий: запустить приложение, открыть настройки через меню, ввести адрес тестового сервера и номер порта, выбрать пункт меню «Save» чтобы сохранить настройки, после перехода на экран аутентификации, ввести имя, зарегистрированного на сервере пользователя и пароль в соответствующие поля на экране смартфона.

Ожидаемый результат: сообщение «Connection failed! Check your connection settings» и переход в ожидание ввода других данных.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден

Сценарий 6.

Цель: проверка идентификации данных с QR-кода и отправка сообщения с отчётом об идентификации на сервер.

Последовательность действий: после прохождения аутентификации навести камеру устройства на QR-код с имением датчика «test/7».

Ожидаемый результат: сообщение «inspects the test/7» в консоли сервера.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден

Сценарий 7.

Цель: проверка идентификации данных с QR-кода и получение показаний датчика.

Последовательность действий: после прохождения аутентификации навести камеру устройства на QR-код с имением датчика «ОНП/Маш.Зал/давление масла».

Ожидаемый результат: появление сроки с именем датчика и показаниями напротив него на экране мобильного устройства.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден

Сценарий 8.

Цель: проверка удаления данных показания датчика из приложения

Последовательность действий: после получения показания датчика, нажать на строку с именем датчика в списке датчиков и не отпуска пока не появится контекстное меню. В контекстном меню выбрать пункт — «удалить».

Ожидаемый результат: исчезновение сроки с именем датчика и показаниями с экрана мобильного устройства.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден

Сценарий 9.

Цель: проверка идентификации данных с QR-кода и получение показаний группы датчиков.

Последовательность действий: после прохождения аутентификации навести камеру устройства на QR-код с имением датчика «ОНП/#».

Ожидаемый результат: появление нескольких срок с именами датчиков «ОНП/*» и показаниями напротив него на экране мобильного устройства.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден

Сценарий 10.

Цель: проверка удаления данных показаний всех датчиков из приложения

Последовательность действий: после получения показания датчиков, нажать на пункт меню с пиктограммой «Корзина».

Ожидаемый результат: исчезновение всех срок с именами датчиков и показаниями с экрана мобильного устройства.

Полученный результат: соответствует ожидаемому.

Вывод: тест пройден

Вывод: приложение удовлетворяет всем функциональным требованиям

Стресс тестирование – проверка приложения с целью выявить предел нормального функционирования.

Для проведения стресс тестирования был также написан скрипт, который в цикле увеличивал число датчиков с регулярно обновляемыми показаниями.

```
#!/bin/bash

echo "Получим адрес сервера"
ipm="$(ip -4 addr show scope global | awk '$1 ~ /^inet/ {print $2}')"
ip="$(echo $ipm | awk -F"/" '{print $1}')"
echo "ip = "$ip

echo "    Запускаем цикл генерации значений"
echo "----------------------------------------"

for ((loop1=0; 1; loop1++))
do
    for ((loop2=0; loop2 <= loop1; loop2++))
    do
        topic="sensor/"$loop2
        msg="loop"$loop1
        sudo mosquitto_pub -h $ip -t $topic -m $msg -u "8host" -P "1234"

        echo -en "\r                                                    "
        echo -en "\r$topic $msg"
        sleep 0.1
    done
done
```

Изменяемые параметры: количество датчиков, интервалы между отправкой сообщений с сервера. Результат оценивался субъективно по скорости обновления списка с датчиками.

Количество датчиков от 1 до 100, интервал между сообщениями 0.1 секунды полученный результат: обновление списка происходит синхронно с отпавкой сообщений с сервера

Количество датчиков от 1 до 100, интервал между сообщениями 0.01 секунды полученный результат: обновление списка происходит не синхронно с отпавкой сообщений с сервера, появляются задержки в работе интерфейса.

Количество датчиков от 1 до 100, интервал между сообщениями 0.001 секунды полученный результат: сразу же видно, что обновление списка происходит с большой задержкой.

Вывод: общее количество датчиков в системе существенного влияния на функционирование системы не оказывают. При этом работа приложения существенно замедляется при уменьшении интервала между сообщениями, отправляемыми с сервера. Оптимальный интервал между сообщениями 0.1 секунды. Допустимый интервал: 0.01 секунды.

ЗАКЛЮЧЕНИЕ.

В результате проделанной работы было спроектировано и реализовано удобное и простое в использовании мобильное приложение для систематического обхода предприятия.

Мобильное приложение поддерживает множество версий платформы Android, при этом оно реализовано с применением принципов Material Design, характерных для современных версий платформы. Реализован основной функционал, которого достаточно для получения показаний датчиков и отметки об осмотре узла или агрегата. К основному функционалу относится возможность подключения к серверу, аутентификация на стороне сервера, идентификация датчиков, отправка доклада об осмотре, получение показаний с идентифицированных датчиков, сохранение показаний датчика в базе данных, вывод показаний датчиков на экран в виде списка, удаление показаний датчика из общего списка, как по отдельности, так и всех сразу.

Для идентификации датчиков отслеживания их показаний не нужно перемещаться с одного экрана на другой, всё реализовано в одном Activity. Навигация внизу главного экрана позволяет переключаться на остальные экраны приложения и не перекрывает при этом изображение с камеры. Это позволяет максимально эффективно использовать место на экране и концентрировать внимание пользователя на идентификации датчика и её результатах.

В результате тестирования установлено, что разработанное приложение корректно решает поставленную задачу контроля за совершением обхода предприятия, идентификации датчиков и получения их показаний на мобильное устройство. Общее количество датчиков в системе существенного влияния на функционирование системы не оказывают. При этом работа приложения существенно замедляется при уменьшении интервала между сообщениями, отправляемыми с сервера. Оптимальный интервал между сообщениями 0.1 секунды. Допустимый интервал: 0.01 секунды.

В случае дальнейшего развития приложения возможно добавление нового функционала, без существенного изменения принципов взаимодействия мобильного приложения с сервером. При взаимодействии с сервером происходит обмен стандартными сообщениями протокола MQTT. В случае расширения функционала приложения в протоколе обмена информацией с сервером может быть сохранена обратная совместимость. При разработке БД была учтена возможность хранения более детальной информации об идентифицированных датчиках и последующего добавления в приложение новых экранов для отображения статистики и построения графиков для составления подробного отчёта об обходе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. «Электронная система обходов оборудования» [Электронный ресурс]. – Режим доступа: http://obhody.ru/ – Загл. с экрана. – яз. рус.
2. «Интерактивная система ОБХОД» [Электронный ресурс]. – Режим доступа: http://obxod.ru/ – Загл. с экрана. – яз. рус.
3. Distribution dashboard [Электронный ресурс]. – Режим доступа: https://developer.android.com/about/dashboards/ – Загл. с экрана. – яз. англ.
4. Near Field Communication [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Near_Field_Communication – Загл. с экрана. – яз. англ.
5. Описание протокола MQTT [Электронный ресурс]. – Режим доступа: https://support.smart-mac.com/knowledge-bases/2/articles/41-opisanie-protokola-mqtt – Загл. с экрана. – яз. рус.
6. Material Design Guide Line [Электронный ресурс]. – Режим доступа: https://www.google.com/design/spec/material-design/introduction.html – Загл. с экрана. – яз. англ.
7. Android [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Android_(operating_system) – Загл. с экрана. – яз. англ.
8. Dalvik [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Dalvik_(software) – Загл. с экрана. – яз. англ.
9. Intellij IDEA [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/idea/features – Загл. с экрана. – яз. англ.
10. Google App Engine [Электронный ресурс]. – Режим доступа: https://cloud.google.com/appengine – Загл. с экрана. – яз. англ.
11. QR-код [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/QR-код – Загл. с экрана. – яз. рус.
12. Код Рида — Соломона [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Код_Рида_—_Соломона – Загл. с экрана. – яз. рус.
13. BarcodeDetector [Электронный ресурс]. – Режим доступа: https://developers.google.com/android/reference/com/google/android/gms/vision/barcode/BarcodeDetector – Загл. с экрана. – яз. англ.
14. com.google.android.gms.vision [Электронный ресурс]. – Режим доступа: https://developers.google.com/android/reference/com/google/android/gms/vision/package-summary – Загл. с экрана. – яз. англ.
15. SQLite [Электронный ресурс]. – Режим доступа: https://ru.bmstu.wiki/SQLite – Загл. с экрана. – яз. рус.
16. Wiren Board [Электронный ресурс]. – Режим доступа: https://wirenboard.com/wiki/index.php/Wiren_Board – Загл. с экрана. – яз. Рус.
17. Бьюли А.Изучаем SQL. – Пер. с англ. – СПб: СимволПлюс, 2007. – 312 с., ил.ISBN13: 9785932860519
18. Шилдт Герберт. Java. Полное руководство 10-е издание. — М.: Диалектика; СПб.: Альфа-книга, 2018. — 1488 с. — ISBN 978-5-6040043-6-4.

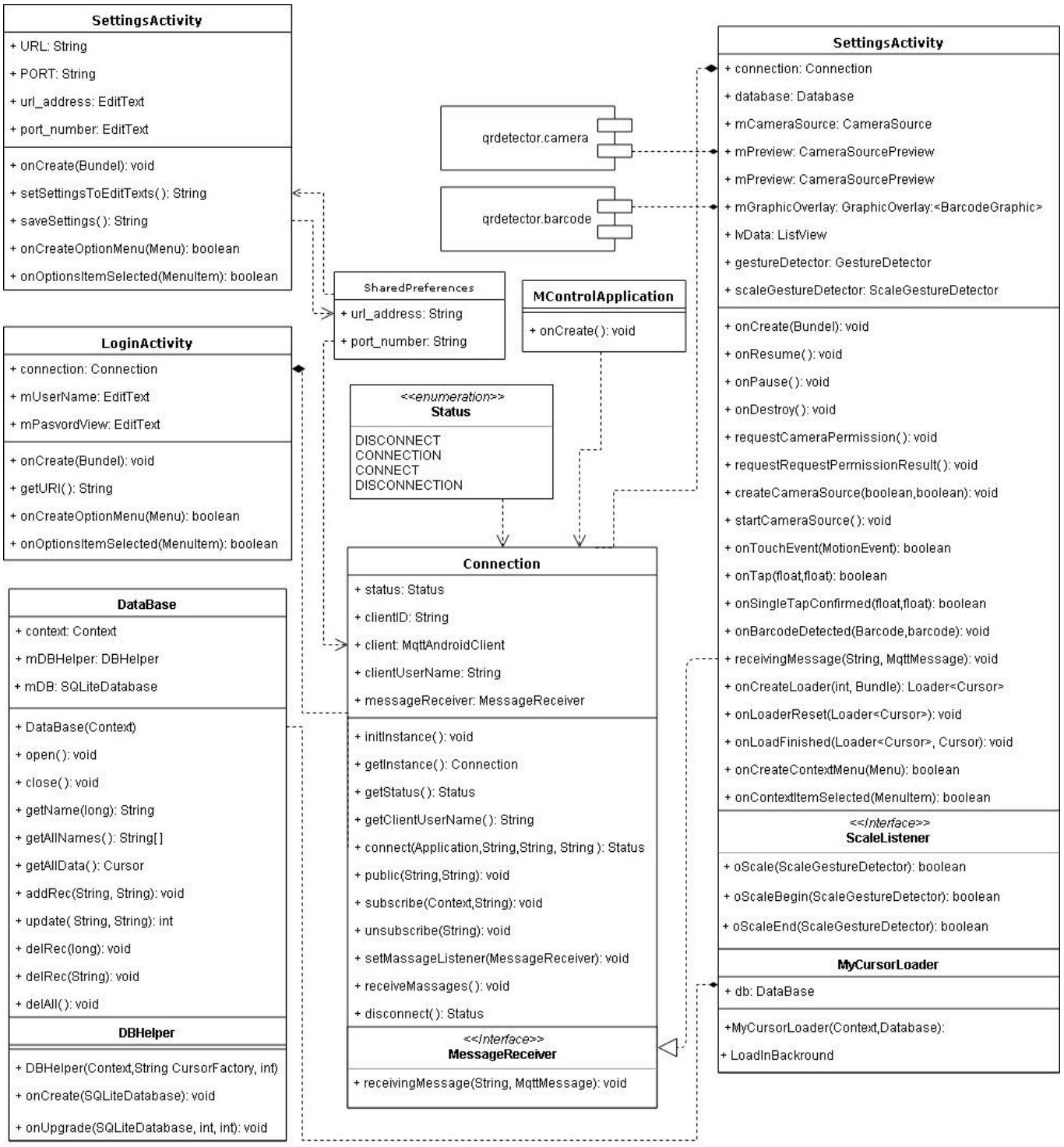
ПРИЛОЖЕНИЕ А

Наиболее употребляемые текстовые сокращения

API – application programming interface
JVM – Java virtual machine
JS – JavaScript
БД – база данных
ОС - операционная система
ПК - персональный компьютер
ПО - программное обеспечение
РСУБД - Реляционная система управления базами данных
ЦП — центральный пульт

ПРИЛОЖЕНИЕ Б

Диаграммы классов мобильного приложения



ПРИЛОЖЕНИЕ Б
Исходный код класса Connection.java

```
package ru.ngtu.mcontrol.mqtt;
import android.app.Application;
import android.content.Context;
import android.content.Intent;
import android.support.annotation.NonNull;
import android.util.Log;
import android.widget.Toast;
import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import java.io.UnsupportedEncodingException;

public class Connection {
    private static final String logTag = "CONNECTION";
    public enum Status {
        DISCONNECT,
        CONNECTING,
        CONNECT,
        DISCONNECTING
    }
    private Status status = Status.DISCONNECT;
    private String clientId;
    private MqttAndroidClient client;
    private String clientUserName;
    private static Connection instance = null;
    private Connection() {
        clientId = MqttClient.generateClientId();     }
    public static void initInstance() {
        if (instance == null) { instance = new Connection(); }
    }
    public static Connection getInstance() {
            if(instance == null)
        return instance;
    }
    public Status getStatus() {
        Status tmp = status;
        return tmp;
    }
    public String getClientUserName() {
        return new String(clientUserName); }
```

```
    public void connect(@NonNull final Application appContext, String
serverURI, final String userName, String password,
                        final Intent intent) {
        if(status == Status.CONNECT) {
            disconnect();
        }
        client = new MqttAndroidClient(appContext, serverURI, clientId);
        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName(userName);
        options.setPassword(password.toCharArray());
        try {
            IMqttToken token = client.connect(options);
            token.setActionCallback(new IMqttActionListener(){
                public void onSuccess(IMqttToken asyncActionToken) {
                    clientUserName = userName;
                    Toast.makeText(appContext,
                            "Connect!",
                            Toast.LENGTH_SHORT).show();
                    status = Status.CONNECT;
                    appContext.startActivity(intent);
                }
                public void onFailure(IMqttToken asyncActionToken, Throwable
exception) {
                            Toast.makeText(appContext,
                             "\t\t\t Connection failed!\n" +
                                     "Check your connection settings!",
                             Toast.LENGTH_SHORT).show();
                }
            });
        } catch (MqttException e) {
            Toast.makeText(appContext,
                    "Error!",
                    Toast.LENGTH_SHORT).show();
            e.printStackTrace();
        }
    }
    public void publish(String topic, String payload) {
        byte[] encodedPayload = new byte[0];
        try {
            encodedPayload = payload.getBytes("UTF-8");
            MqttMessage message = new MqttMessage(encodedPayload);
            client.publish(topic, message);
        } catch (UnsupportedEncodingException | MqttException e) {
            e.printStackTrace();
        }
    }
```

```
public void subscribe(@NonNull final Context appContext, final String topic)
{
        int qos = 1;
        try {
            IMqttToken subToken = client.subscribe(topic, qos);
            subToken.setActionCallback(new IMqttActionListener() {
                public void onSuccess(IMqttToken asyncActionToken) {
                }
                public void onFailure(IMqttToken asyncActionToken,
                                      Throwable exception) {
                    Toast.makeText(appContext,
                            "Failed subscribing to " + topic,
                            Toast.LENGTH_SHORT).show();
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
    public void unsubscribe(final String topic){
        try {
            IMqttToken unsubToken = client.unsubscribe(topic);
            unsubToken.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                }
                @Override
                public void onFailure(IMqttToken asyncActionToken,
                                      Throwable exception) {
                }
            });
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
    public interface MessagesReceiver{
        void receivingMessage(String topic, MqttMessage message);
    }
    MessagesReceiver messagesReceiver;
    public void setMessagesListener(MessagesReceiver messagesListener){
        this.messagesReceiver = messagesListener;
    }
    public void receiveMessages( ) {
        if(client != null) {
            client.setCallback(new MqttCallback() {
                public void connectionLost(Throwable cause) {
                }
                public void messageArrived(String topic, MqttMessage
message) throws Exception {
                    // отпарвляем сообщение получателю
                    messagesReceiver.receivingMessage(topic, message);
```

```
                }
                @Override
                public void deliveryComplete(IMqttDeliveryToken token) {
                }
            });
        }
    }
    public void disconnect() {
        if(status != Status.DISCONNECT) {
            try {
                IMqttToken disconToken = client.disconnect();
                disconToken.setActionCallback(new IMqttActionListener() {
                    @Override
                    public void onSuccess(IMqttToken asyncActionToken) {
                                        status = Status.DISCONNECT;
                    }
                    public void onFailure(IMqttToken asyncActionToken,
                                          Throwable exception) {
                    }
                });
            } catch (MqttException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Исходный код класса DataBase.java

```
package ru.ngtu.mcontrol.database;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DataBase {
    private static final String DB_NAME = "MCData";
    private static final int DB_VERSION = 1;
    private static final String DB_TABLE = "Sensors";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_NAME = "name";
    public static final String COLUMN_VALUE = "value";
    public static final String COLUMN_UM = "unit_of_measure";
    public static final String COLUMN_TIME = "time";
    private static final String DB_CREATE =
            "create table " + DB_TABLE + "(" +
                    COLUMN_ID + " integer primary key autoincrement, " +
COLUMN_NAME + " text, " +  COLUMN_VALUE + " text, " +   COLUMN_TIME + " int"
+ ");";
    private static final String ORDER = COLUMN_TIME + " DESC";
    private final Context mCtx;
    private DBHelper mDBHelper;
    private SQLiteDatabase mDB;
    public DataBase(Context ctx) {
        mCtx = ctx;
    }
    public void open() {
        mDBHelper = new DBHelper(mCtx, DB_NAME, null, DB_VERSION);
        mDB = mDBHelper.getWritableDatabase();
    }
    public void close() {
        if (mDBHelper!=null) mDBHelper.close();
    }
    public String getName(long id) {
        Cursor cursor = mDB.query(DB_TABLE, null, COLUMN_ID + " = ?", new
String[] {String.valueOf(id)}, null, null,null);
        int columnIndex = cursor.getColumnIndex(COLUMN_NAME);
        if(cursor.moveToFirst()){
            return new String(cursor.getString(columnIndex));
        }
        return "";
    }

public Cursor getAllData() {
        return mDB.query(DB_TABLE, null, null, null, null, null, ORDER);
    }
    public String[] getAllNames() {
```

```
        Cursor cursor = mDB.query(DB_TABLE, new String[] { COLUMN_NAME },
null, null, null, null, null);
        int columnIndex = cursor.getColumnIndex(COLUMN_NAME);
        String[] names = new String[cursor.getCount()];
        int i = 0;
        for (cursor.moveToFirst(); !cursor.isAfterLast();
cursor.moveToNext()){
          names[i++] = cursor.getString(columnIndex);
        }
        return names;
    }
    public void addRec(String name, String value) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_NAME, name);
        cv.put(COLUMN_VALUE, value);
        cv.put(COLUMN_TIME, System.currentTimeMillis());
        mDB.insert(DB_TABLE, null, cv);
    }
    public  int update(String name, String value) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_NAME, name);
        cv.put(COLUMN_VALUE, value);
        int updCount = mDB.update(DB_TABLE, cv, COLUMN_NAME + " = ?",
                new String[] { name });
        return updCount;
    }
   public void delRec(long id) {
        mDB.delete(DB_TABLE, COLUMN_ID + " = " + id, null);    }
public void delRec(String name) {
        mDB.delete(DB_TABLE, COLUMN_NAME + " = ?", new String[] { name }); }
    public void delAll() {
        mDB.delete(DB_TABLE, null, null);}
private class DBHelper extends SQLiteOpenHelper {
        public DBHelper(Context context, String name,
SQLiteDatabase.CursorFactory factory, int version) {
            super(context, name, factory, version); }
        public void onCreate(SQLiteDatabase db) {
            db.execSQL(DB_CREATE);}
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        }
    }
}
```

Исходный код класса LoginActivity.java

```
package ru.ngtu.mcontrol.ui;
import android.app.Application;
import android.content.Intent;
import android.content.SharedPreferences;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import ru.ngtu.mcontrol.R;
import ru.ngtu.mcontrol.mqtt.Connection;

public class LoginActivity extends AppCompatActivity {
    private Connection connection;
    private EditText mPasswordView, mUserName;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        ActionBar actionBar = getSupportActionBar();
        actionBar.setTitle(R.string.mc_login);
        connection = Connection.getInstance();
        if(connection.getStatus() == Connection.Status.CONNECT) {
            actionBar.setHomeButtonEnabled(true);
            actionBar.setDisplayHomeAsUpEnabled(true);
        }
        mUserName = (EditText) findViewById(R.id.txtUserName);
        mPasswordView = (EditText) findViewById(R.id.txtPassword);
        Button btnLogin = (Button) findViewById(R.id.btnLogin);
        btnLogin.setOnClickListener(new OnClickListener() {
           public void onClick(View view) {
                String UserName = mUserName.getText().toString();
                String Pwd = mPasswordView.getText().toString();
                if((mUserName.length() != 0) && (mPasswordView.length() !=
0)){
                    Intent MainIntent = new Intent(LoginActivity.this,
MainActivity.class).addFlags(
                        Intent.FLAG_ACTIVITY_CLEAR_TASK).addFlags(
                        Intent.FLAG_ACTIVITY_NEW_TASK);
                    connection.connect((Application)
getApplicationContext(), getURI(), UserName, Pwd, MainIntent);
                }   }       });    }

    private String getURI() {
        SharedPreferences sharedPreferences;
        sharedPreferences =
getSharedPreferences("MControlPref",MODE_PRIVATE);
```

```
        String uri = "tcp://" +
sharedPreferences.getString(SettingsActivity.URL, "")
                + (":" + sharedPreferences.getString(SettingsActivity.PORT,
""));
        return new String(uri);
    }
     public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.login, menu);
        return true;
    }
    public boolean onOptionsItemSelected(MenuItem item)
    {
       switch (item.getItemId())
        {
            case android.R.id.home:
                this.finish();
                return true;
            case R.id.settings:
                Intent intent = new Intent(LoginActivity.this,
SettingsActivity.class);
                startActivity(intent);
                return true;

            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

Исходный код класса SettingsActivity.java

```
package ru.ngtu.mcontrol.ui;
import android.annotation.TargetApi;
import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Build;
import android.support.annotation.NonNull;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.inputmethod.EditorInfo;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import java.util.LinkedList;
import ru.ngtu.mcontrol.R;

public class SettingsActivity extends AppCompatActivity {
    static final String URL = "url_address";
    static final String PORT = "port_number";
    private EditText url_address, port_number;
    private LinkedList<EditText> activeEdit = null;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);
        ActionBar actionBar =getSupportActionBar();
        actionBar.setHomeButtonEnabled(true);
        actionBar.setDisplayHomeAsUpEnabled(true);
        actionBar.setTitle(R.string.title_Settings);
        url_address = (EditText) findViewById(R.id.editURL);
        port_number = (EditText) findViewById(R.id.editPort);
        setOnFocusChangeListener(url_address);
        setOnFocusChangeListener(port_number);
        setSettingsToEditTexts();
    }

private void setSettingsToEditTexts() {
        SharedPreferences sharedPreferences;
        sharedPreferences =
getSharedPreferences("MControlPref",MODE_PRIVATE);
        url_address.setText(sharedPreferences.getString(URL, ""));
        port_number.setText(sharedPreferences.getString(PORT, ""));
    }
    private void saveSettings() {
        SharedPreferences sharedPreferences;
```

```
        sharedPreferences =
getSharedPreferences("MControlPref",MODE_PRIVATE);
        SharedPreferences.Editor spEditor = sharedPreferences.edit();
        spEditor.putString(URL, url_address.getText().toString());
        spEditor.putString(PORT, port_number.getText().toString());
        spEditor.commit();

        Toast.makeText(this,
                "Save settings",
                Toast.LENGTH_SHORT).show();
        Intent LoginIntent = new Intent(this, LoginActivity.class);
        startActivity(LoginIntent);
    }
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.settings, menu);
        SharedPreferences preferences =
getSharedPreferences("MControlPref",MODE_PRIVATE);

        if(preferences.getString(MainActivity.USEFLASH, "OFF").equals("ON"))
{
            menu.getItem(0).setIcon(R.drawable.ic_highlight_white_off_32dp);
        }
        if(preferences.getString(MainActivity.AUTOFOCUS,
"OFF").equals("ON")) {

menu.getItem(1).setIcon(R.drawable.ic_center_focus_weak_white_32dp);
        }
        return true;
    }
    public boolean onOptionsItemSelected(MenuItem item) {
        SharedPreferences preferences =
getSharedPreferences("MControlPref",MODE_PRIVATE);
        SharedPreferences.Editor spEditor = preferences.edit();
        switch (item.getItemId()) {
            case android.R.id.home:
                this.finish();
                return true;
            case R.id.save:
                saveSettings();
                return true;
            case R.id.light:
                if(preferences.getString(MainActivity.USEFLASH,
"OFF").equals("ON")) {
                    item.setIcon(R.drawable.ic_highlight_white_on_32dp);
                    spEditor.putString(MainActivity.USEFLASH, "OFF");
                    spEditor.commit();
                } else{
                    item.setIcon(R.drawable.ic_highlight_white_off_32dp);
                    spEditor.putString(MainActivity.USEFLASH, "ON");
                    spEditor.commit();
                }
```

```
                return true;
            case R.id.autofocus:
                if(preferences.getString(MainActivity.AUTOFOCUS,
"OFF").equals("ON")) {

item.setIcon(R.drawable.ic_center_focus_strong_white_32dp);
                    spEditor.putString(MainActivity.AUTOFOCUS, "OFF");
                    spEditor.commit();
                } else{

item.setIcon(R.drawable.ic_center_focus_weak_white_32dp);
                    spEditor.putString(MainActivity.AUTOFOCUS, "ON");
                    spEditor.commit();
                }
                return true;

            default:
                return super.onOptionsItemSelected(item);
        }
    }
    private void setOnFocusChangeListener(@NonNull final EditText editText)
{

        if(activeEdit == null) { activeEdit = new LinkedList<>(); }

        editText.setOnFocusChangeListener(new View.OnFocusChangeListener() {
            public void onFocusChange(View v, boolean hasFocus) {
                if (hasFocus) {
                    startEditText((EditText)v);
                } else {
                    ((EditText)v).setCursorVisible(false);
                }
            }
        });
    }

@TargetApi(Build.VERSION_CODES.LOLLIPOP)
    void startEditText(@NonNull final EditText editText) {
        editText.setCursorVisible(true);
        if(activeEdit.contains(editText)) {
            for(EditText id : activeEdit) {
                if(id.getId() == editText.getId()) {
                    EditText tmp = id;
                    activeEdit.remove(id);
                    activeEdit.addLast(tmp);
                    break;
                }
            }
        } else {
            activeEdit.addLast(editText);
```

```
            editText.setOnEditorActionListener(new
TextView.OnEditorActionListener() {
                public boolean onEditorAction(TextView v, int actionId,
KeyEvent event) {
                    if (actionId == EditorInfo.IME_ACTION_DONE) {
                        String str = v.getText().toString();
                        if(str.length() != 0) {
                            for(EditText id : activeEdit) {
                                if(id.getId() == v.getId()) {
                                    activeEdit.remove(id);
                                    break;
                                }
                            }
                            if(activeEdit.isEmpty()){
                                v.clearFocus();
                                hideKeyboard(v);
                            } else {
                                activeEdit.peekLast().requestFocus();
                            }
                        }
                        return true;
                    }
                    return false;
                }
            });
        }
    }
    public void hideKeyboard(@NonNull View view) {
        InputMethodManager inputMethodManager
=(InputMethodManager)getSystemService(Activity.INPUT_METHOD_SERVICE);
        inputMethodManager.hideSoftInputFromWindow(view.getWindowToken(),
0);
    }
}
```

Исходный код главного экрана мобильного приложения - MainActivity.java

```
package ru.ngtu.mcontrol.ui;
import android.Manifest;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Application;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.graphics.Point;
import android.hardware.Camera;
import android.os.Build;
import android.support.annotation.NonNull;
import android.support.design.widget.BottomNavigationView;
import android.support.design.widget.Snackbar;
import android.support.v4.app.ActivityCompat;
import android.content.Intent;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.LoaderManager;
import android.support.v4.content.CursorLoader;
import android.support.v4.content.Loader;
import android.support.v4.widget.SimpleCursorAdapter;
import android.os.Bundle;
import android.util.Log;
import android.view.ContextMenu;
import android.view.Display;
import android.view.GestureDetector;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.ScaleGestureDetector;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.Toast;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GoogleApiAvailability;
import com.google.android.gms.vision.MultiProcessor;
import com.google.android.gms.vision.barcode.Barcode;
import com.google.android.gms.vision.barcode.BarcodeDetector;
import org.eclipse.paho.client.mqttv3.MqttMessage;
public class MainActivity extends FragmentActivity implements
BarcodeGraphicTracker.BarcodeUpdateListener,
    LoaderManager.LoaderCallbacks<Cursor>, Connection.MessagesReceiver {
    private Application appContext;
    private static final int CM_DELETE_ID = 1;
    private ListView lvData;
```

```
    private DataBase dataBase;
    private SimpleCursorAdapter scAdapter;
    private static final String TAG = "Barcode-reader";
    static final String AUTOFOCUS = "AutoFocus";
    static final String USEFLASH = "UseFlash";
    private Connection connection;
    private CameraSource mCameraSource;
    private CameraSourcePreview mPreview;
    private GraphicOverlay<BarcodeGraphic> mGraphicOverlay;
    private static final int RC_HANDLE_GMS = 9001;
    private static final int RC_HANDLE_CAMERA_PERM = 2;
    private ScaleGestureDetector scaleGestureDetector;
    private GestureDetector gestureDetector;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        appContext = getApplication();
        dataBase = new DataBase(appContext);
        dataBase.open();
        dataBase.delRec("trrte");
        String[] from = new String[] { DataBase.COLUMN_NAME,
DataBase.COLUMN_VALUE };
        int[] to = new int[] { R.id.name, R.id.value };
        scAdapter = new SimpleCursorAdapter(this, R.layout.item, null, from,
to, 0);
        lvData = (ListView) findViewById(R.id.lvData);
        lvData.setAdapter(scAdapter);
        registerForContextMenu(lvData);
        getSupportLoaderManager().initLoader(0, null, this);
        connection = Connection.getInstance();
        connection.setMessagesListener(this);
        connection.receiveMessages();
        mPreview = (CameraSourcePreview) findViewById(R.id.preview);
        mGraphicOverlay = (GraphicOverlay<BarcodeGraphic>)
findViewById(R.id.graphicOverlay);



     boolean autoFocus = false;
        boolean useFlash = false;
        SharedPreferences preferences =
getSharedPreferences("MControlPref",MODE_PRIVATE);
        if(preferences.getString(USEFLASH, "OFF").equals("ON")) useFlash =
true;
        if(preferences.getString(AUTOFOCUS, "OFF").equals("ON")) autoFocus =
true;
        int rc = ActivityCompat.checkSelfPermission(this,
Manifest.permission.CAMERA);
        if (rc == PackageManager.PERMISSION_GRANTED) {
            createCameraSource(autoFocus, useFlash);
```

```
        } else {
            requestCameraPermission();
        }
        gestureDetector = new GestureDetector(this, new
CaptureGestureListener());
        scaleGestureDetector = new ScaleGestureDetector(this, new
ScaleListener());
        BottomNavigationView navigation = (BottomNavigationView)
findViewById(R.id.navigation);
navigation.setOnNavigationItemSelectedListener(mOnNavigationItemSelectedList
ener);       }
private void requestCameraPermission() {
final String[] permissions = new String[]{Manifest.permission.CAMERA};
   if (!ActivityCompat.shouldShowRequestPermissionRationale(this,
                Manifest.permission.CAMERA)) {
            ActivityCompat.requestPermissions(this, permissions,
RC_HANDLE_CAMERA_PERM);
            return;
        }
      final Activity thisActivity = this;
      View.OnClickListener listener = new View.OnClickListener() {
      public void onClick(View view) {
                ActivityCompat.requestPermissions(thisActivity, permissions,
                        RC_HANDLE_CAMERA_PERM);
            }
        };
        findViewById(R.id.container).setOnClickListener(listener);
        Snackbar.make(mGraphicOverlay, R.string.permission_camera_rationale,
                Snackbar.LENGTH_INDEFINITE)
                .setAction(R.string.ok, listener)
                .show();
    }
   public boolean onTouchEvent(MotionEvent e) {
        boolean b = scaleGestureDetector.onTouchEvent(e);
        boolean c = gestureDetector.onTouchEvent(e);
        return b || c || super.onTouchEvent(e);
    }


    private void createCameraSource(boolean autoFocus, boolean useFlash) {
        Context context = getApplicationContext();
        BarcodeDetector barcodeDetector = new
BarcodeDetector.Builder(context).build();
        BarcodeTrackerFactory barcodeFactory = new
BarcodeTrackerFactory(mGraphicOverlay, this);
        barcodeDetector.setProcessor(
                new MultiProcessor.Builder<>(barcodeFactory).build());
        if (!barcodeDetector.isOperational()) {
            IntentFilter lowstorageFilter = new
IntentFilter(Intent.ACTION_DEVICE_STORAGE_LOW);
            boolean hasLowStorage = registerReceiver(null, lowstorageFilter)
```

```
!= null;
            if (hasLowStorage) {
                Toast.makeText(this, R.string.low_storage_error,
Toast.LENGTH_LONG).show();
            }
        }
        Display display = getWindowManager().getDefaultDisplay();
        Point size = new Point();
        display.getSize(size);
        float width = size.x;
        float height = size.y;
        float ratio = height / 2 / width;
        CameraSource.Builder builder = new
CameraSource.Builder(getApplicationContext(), barcodeDetector)
                .setFacing(CameraSource.CAMERA_FACING_BACK)
                .setRequestedPreviewSize(720, (int) (480 * ratio)) //TODO
Поменял местами
                .setRequestedFps(15.0f);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH)
{
            builder = builder.setFocusMode(
                    autoFocus ?
Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE : null);
        }
        mCameraSource = builder
                .setFlashMode(useFlash ? Camera.Parameters.FLASH_MODE_TORCH
: null)
                .build();
    }
    protected void onResume() {
        super.onResume();
        startCameraSource();
    }



    protected void onPause() {
        super.onPause();
        if (mPreview != null) {
            mPreview.stop();
        }
    }
    protected void onDestroy() {
        super.onDestroy();
        dataBase.close();
        if (mPreview != null) {
            mPreview.release();
        }
        connection.disconnect();
    }
    public void onRequestPermissionsResult(int requestCode,  String[]
```

```
permissions,  int[] grantResults) {
        if (requestCode != RC_HANDLE_CAMERA_PERM) {
            super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
            return;
        }
        if (grantResults.length != 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            boolean autoFocus = false;
            boolean useFlash = false;
            SharedPreferences preferences =
getSharedPreferences("MControlPref",MODE_PRIVATE);
            if(preferences.getString(USEFLASH, "OFF").equals("ON")) useFlash
= true;
            if(preferences.getString(AUTOFOCUS, "OFF").equals("ON"))
autoFocus = true;
            createCameraSource(autoFocus, useFlash);
            return;
        }
        DialogInterface.OnClickListener listener = new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
               finish();
            }
        };
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("Multitracker sample")
                .setMessage(R.string.no_camera_permission)
                .setPositiveButton(R.string.ok, listener)
                .show();
    }



            int code =
GoogleApiAvailability.getInstance().isGooglePlayServicesAvailable(
                    getApplicationContext());
        if (code != ConnectionResult.SUCCESS) {
            Dialog dlg =
GoogleApiAvailability.getInstance().getErrorDialog(this, code,
RC_HANDLE_GMS);
            dlg.show();
        }
        if (mCameraSource != null) {
            try {
                mPreview.start(mCameraSource, mGraphicOverlay);
            } catch (IOException e) {
                mCameraSource.release();
                mCameraSource = null;
            }
        }
```

```
    }
    private boolean onTap(float rawX, float rawY) {
        int[] location = new int[2];
        mGraphicOverlay.getLocationOnScreen(location);
        float x = (rawX - location[0]) /
mGraphicOverlay.getWidthScaleFactor();
        float y = (rawY - location[1]) /
mGraphicOverlay.getHeightScaleFactor();
        Barcode best = null;
        float bestDistance = Float.MAX_VALUE;
        for (BarcodeGraphic graphic : mGraphicOverlay.getGraphics()) {
            Barcode barcode = graphic.getBarcode();
            if (barcode.getBoundingBox().contains((int) x, (int) y)) {
                best = barcode;
                break;
            }
            float dx = x - barcode.getBoundingBox().centerX();
            float dy = y - barcode.getBoundingBox().centerY();
            float distance = (dx * dx) + (dy * dy);  // actually squared
distance
            if (distance < bestDistance) {
                best = barcode;
                bestDistance = distance;
            }
        }
        return false;
    }
    private class CaptureGestureListener extends
GestureDetector.SimpleOnGestureListener {
        public boolean onSingleTapConfirmed(MotionEvent e) {
            return onTap(e.getRawX(), e.getRawY()) ||
super.onSingleTapConfirmed(e);
        }    }


    private class ScaleListener implements
ScaleGestureDetector.OnScaleGestureListener {
        public boolean onScale(ScaleGestureDetector detector) {
           return false;
        }
       public boolean onScaleBegin(ScaleGestureDetector detector) {
            return true;
        }
        public void onScaleEnd(ScaleGestureDetector detector) {
            mCameraSource.doZoom(detector.getScaleFactor());
        }
    }
    public void onBarcodeDetected(Barcode barcode) {
        connection.subscribe(this, barcode.rawValue);
        connection.publish(connection.getClientUserName(),
                connection.getClientUserName() + " inspected: " +
```

```
barcode.rawValue);
    }
    public void receivingMessage(String topic, MqttMessage message) {
        if(dataBase.update(topic, new String(message.getPayload()))<1) {
            dataBase.addRec(topic, new String(message.getPayload()));
        }
        getSupportLoaderManager().getLoader(0).forceLoad();
    }
    public Loader<Cursor> onCreateLoader(int id, Bundle bndl) {
        return new MyCursorLoader(this, dataBase);
    }
    public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
        scAdapter.swapCursor(cursor);
    }
    public void onLoaderReset(Loader<Cursor> loader) {    }
    static class MyCursorLoader extends CursorLoader {
        DataBase db;
        public MyCursorLoader(Context context, DataBase db) {
            super(context);
            this.db = db;
        }
        public Cursor loadInBackground() {
            Cursor cursor = db.getAllData();
            return cursor;
        }
    }



    public void onCreateContextMenu(ContextMenu menu, View v,
                                    ContextMenu.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        menu.add(0, CM_DELETE_ID, 0, getString(R.string.delete_record));
    }
    public boolean onContextItemSelected(MenuItem item) {
        if (item.getItemId() == CM_DELETE_ID) {
            AdapterView.AdapterContextMenuInfo acmi =
(AdapterView.AdapterContextMenuInfo) item
                    .getMenuInfo();
            connection.unsubscribe(dataBase.getName(acmi.id));
            dataBase.delRec(acmi.id);
            getSupportLoaderManager().getLoader(0).forceLoad();
            return true;
        }
        return super.onContextItemSelected(item);
    }
    private BottomNavigationView.OnNavigationItemSelectedListener
mOnNavigationItemSelectedListener
            = new BottomNavigationView.OnNavigationItemSelectedListener() {
        public boolean onNavigationItemSelected(@NonNull MenuItem item) {
          Intent intent;
```

```
            switch (item.getItemId()) {
                case R.id.navigation_relogin:
                    intent = new Intent(MainActivity.this,
LoginActivity.class);
                    startActivity(intent);
                    return true;
                case R.id.navigation_clear_all:
                    if(dataBase != null) {
                        if(connection !=null) {
                            for(String name: dataBase.getAllNames()) {
connection.unsubscribe(name); }
                        }
                        dataBase.delAll();
                        getSupportLoaderManager().getLoader(0).forceLoad();
                    }
                    return true;
                case R.id.navigation_settings:
                    intent = new Intent(MainActivity.this,
SettingsActivity.class);
                    startActivity(intent);
                    return true;
            }
            return false;
        }    };
}
```