Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ

Зав. каф. ЭВМ

\_\_\_\_\_\_\_\_\_\_ Б.В. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту

на тему

ПРОГРАММНЫЙ КОМПЛЕКС «УМНЫЙ ДОМ» С ПРИМЕНЕНИЕМ ТЕХНОЛОГИИ ZIGBEE

БГУИР ДП 1–40 02 01 01 088 ПЗ

Студент Я.В. Антонов

Руководитель Ю.А. Луцик

Консультанты:

от кафедры ЭВМ Ю.А. Луцик

по экономической части В.Г. Горовой

Нормоконтролер А.И Стракович

Рецензент

МИНСК 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 7](#_Toc166527598)

[1 ОБЗОР ЛИТЕРАТУРЫ 8](#_Toc166527599)

[1.1 Обзор существующих аналогов 8](#_Toc166527600)

[1.2 MQTT-протокол 15](#_Toc166527601)

[1.3 Протоколы передачи данных 17](#_Toc166527602)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 20](#_Toc166527603)

[2.1 Блок пользовательского интерфейса 20](#_Toc166527604)

[2.2 Блок соединения с брокером MQTT 22](#_Toc166527605)

[2.3 Блок приема-передачи данных MQTT брокеру 22](#_Toc166527606)

[2.4 Блок обработки данных 23](#_Toc166527607)

[2.5 Блок базы данных 24](#_Toc166527608)

[2.6 Блок тестирования 24](#_Toc166527609)

[2.7 Блок координатора 25](#_Toc166527610)

[2.8 Блок конечных устройств 25](#_Toc166527611)

[2.9 Блок MQTT брокера 26](#_Toc166527612)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 27](#_Toc166527613)

[3.1 Аппаратная часть 27](#_Toc166527614)

[3.2 Программная часть 28](#_Toc166527617)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 43](#_Toc166527647)

[4.1 Android Studio 43](#_Toc166527648)

[4.2 Добавление системы 44](#_Toc166527649)

[4.3 Вывод списка систем 47](#_Toc166527650)

[4.4 Вывод списка устройств 48](#_Toc166527651)

[4.5 Добавление нового устройства 49](#_Toc166527652)

[4.6 Подключение к MQTT брокеру 52](#_Toc166527653)

[4.7 Подписка и отписка на темы (topic) 53](#_Toc166527654)

[4.7 Получение данных от MQTT брокера 55](#_Toc166527655)

[4.8 Обработка полученных данных 55](#_Toc166527656)

[4.9 Отправка данных в MQTT брокер 56](#_Toc166527657)

[4.10 Взаимодействие с базой данных 57](#_Toc166527658)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 61](#_Toc166527659)

[5.1 Руководство по использованию устройства координатор. 61](#_Toc166527660)

[5.2 Системные требования мобильного устройства 65](#_Toc166527661)

[5.3 Руководство по использованию мобильного приложения 66](#_Toc166527662)

[5.4 Добавление системы 67](#_Toc166527663)

[5.5 Изменение и удаление системы 68](#_Toc166527664)

[5.6 Открытие списка устройств и добавление устройства 69](#_Toc166527665)

[5.6 Изменение и удаление устройства 71](#_Toc166527666)

[5.6 Руководство программиста 71](#_Toc166527667)

[5.7 Установка IDE Android Studio 71](#_Toc166527668)

[5.8 Java Development Kit 73](#_Toc166527669)

[5.12 MQTT сервер 74](#_Toc166527670)

[5.13 Установка ПО ESP32 76](#_Toc166527671)

[5.14 Установка ПО E72 77](#_Toc166527672)

[5.15 Разработка и установка ПО конечного устройства 79](#_Toc166527673)

[6 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 82](#_Toc166527674)

[6.1 Модульное тестирование 82](#_Toc166527675)

[6.2 Ручное тестирование 87](#_Toc166527676)

[7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И   
 РЕАЛИЗАЦИИ НА РЫНКЕ ПРОГРАММНОГО КОМПЛЕКСА   
 «УМНЫЙ ДОМ» С ПРИМЕНЕНИЕМ ТЕХНОЛОГИИ ZIGBEE 90](#_Toc166527677)

[7.1 Описание функций, назначения и потенциальных пользователей  
 программного комплекса 90](#_Toc166527678)

[7.2 Расчет затрат на разработку программного комплекса 90](#_Toc166527679)

[7.3 Расчет экономического эффекта от реализации программного  
 комплекса на рынке 92](#_Toc166527684)

[7.4 Расчет показателей экономической эффективности разработки и   
 реализации программного средства на рынке 94](#_Toc166527685)

[7.5 Вывод об экономической целесообразности реализации проектного  
 решения 94](#_Toc166527686)

[ЗАКЛЮЧЕНИЕ 95](#_Toc166527687)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 96](#_Toc166527688)

[ПРИЛОЖЕНИЕ А 98](#_Toc166527689)

[ПРИЛОЖЕНИЕ Б 130](#_Toc166527690)

[ПРИЛОЖЕНИЕ В 131](#_Toc166527691)

[ПРИЛОЖЕНИЕ Г 132](#_Toc166527692)

# ВВЕДЕНИЕ

С появлением передовых технологий, жизнь человека становится более комфортной и продуктивной. Развитие технологий в области «умного дома» привело к тому, что в доме теперь можно автоматизировать и контролировать множество процессов, что делает жизнь более удобной, безопасной и эффективной. От управления освещением и климатом до мониторинга безопасности и энергопотребления – все это становится доступным благодаря «умным» технологиям.

Сегодня множество компаний предлагают различные решения для «умного дома». Некоторые из них специализируются на «умных замках» и системах безопасности, другие - на «умных термостатах» и системах управления климатом, а третьи - на интегрированных системах «умного освещения». Более того, с появлением стандартов связи, таких как Zigbee и Z-Wave, устройства разных производителей могут взаимодействовать между собой, что упрощает интеграцию различных компонентов системы «умного дома».

Существуют как готовые решения, представляющие собой набор устройств с уже встроенным программным обеспечением для управления ими, так и DIY-проекты, где пользователи могут самостоятельно прошивать устройства и создавать собственные решения под свои потребности.

Однако, несмотря на все преимущества «умного дома», вопрос цены остается значимым. Многие интегрированные системы все еще остаются дорогостоящими, что может отпугнуть потенциальных пользователей. Поэтому разработка доступных по цене устройств является важным направлением развития этой отрасли.

Цель дипломного проекта заключается не только в прошивке устройств и разработке приложения для управления ими, но и в создании гибкой и доступной системы «умного дома», которая бы удовлетворяла потребности разнообразных пользователей. Это включает в себя не только технические аспекты, но и удобство использования, безопасность и экономическую целесообразность.

Для достижения поставленной цели дипломного проекта важно разбить её на конкретные задачи:

– прошивка и настройка устройств сети;

– проектирование архитектуры системы «умный дом»;

– разработка программного обеспечения;

– тестирование и отладка приложения.

Каждая из этих задач важна для создания успешного проекта по разработке гибкой и доступной системы «умного дома», удовлетворяющей потребности разнообразных пользователей.

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Обзор существующих аналогов

1.1.1 Система «Хайт Про»

«Хайт Про» [1] (рисунок 1.1) – это один из аналогов систем «умного дома», который следует изучить для понимания того, какие функциональности уже реализованы на рынке и какие подходы используются в этой области.



Рисунок 1.1 – Система «Хайт Про»

Вышеназванная система представляет собой компактное устройство, которое без труда вставляется в стандартную розетку 220 В. Оно работает на специальной радиочастоте 868 МГц и функционирует без подключения к интернету. Компактный корпус этого устройства укомплектован кнопкой управления, двумя индикаторными светодиодами и двумя разъемами: RJ-45 и USB. Программирование и настройка системы, а также добавление новых устройств осуществляются при помощи приложения HiTE PRO.

Среди функциональных возможностей этого устройства выделяются: способность управлять освещением и приводами, обеспечивая открытие и закрытие штор и жалюзи. Кроме того, система обладает разнообразными датчиками, которые могут контролировать движение, температуру, влажность, протечки воды, а также открытие дверей и окон. Интегрированные сценарии позволяют создавать удобные и интуитивно понятные автоматизированные процессы.

Эта система «умного дома» легко интегрируется в пять популярных экосистем: Apple HomeKit, Google Home, HiTE PRO, Tuya Smart и «Умный дом Яндекса». Кроме того, комплексом можно управлять голосом при помощи таких голосовых ассистентов, как Google Assistant, «Яндекс.Алиса», Siri и «Маруся».

В комплект этого устройства входит все необходимое для начала использования: основа «умного устройства», реле для управления освещением, беспроводной выключатель, а также датчики открытия двери и протечки воды.

Несмотря на многочисленные функции и преимущества, пользователи отмечают некоторые недостатки:

– иногда в системе возникают сбои, и связь между блоком управления и реле теряется;

– отсутствие функций обнаружения пожара;

– обновления программного обеспечения выпускаются не так часто, как хотелось бы пользователям.

Помимо перечисленных функциональных возможностей и недостатков, важно отметить, что компания предоставляет эффективную техническую поддержку для пользователей устройства. Техническая поддержка оперативно отвечает на запросы пользователей и готова помочь с решением любых возникающих проблем или вопросов.

1.1.2 «Умный дом» от «Белтелеком»

«Умный дом» от компании «Белтелеком» [2] представляет собой простое и экономичное решение, которое включает в себя все основные функции для обеспечения комфорта и ресурсосбережения в помещениях. Эта система предоставляет абонентам информацию о состоянии различных объектов в помещениях и позволяет им управлять этими объектами в соответствии с их потребностями.

Пользователи имеют возможность настраивать алгоритмы работы системы под свои индивидуальные требования, создавая сценарии событий. В этих сценариях можно настроить систему таким образом, чтобы не только уведомлять пользователя о срабатывании датчиков, но и выполнять определенные действия с помощью устройств. Например, можно настроить систему так, чтобы она автоматически регулировала температуру в помещении при определенных условиях.

С помощью этой системы пользователи могут интегрировать датчики задымления, движения и открытия дверей/окон для обеспечения безопасности своего дома. Датчики задымления могут оперативно реагировать на возможные пожары, предупреждая об опасности и активируя аварийную сигнализацию. Датчики движения и открытия дверей/окон могут использоваться для контроля доступа и обеспечения безопасности внутри дома.

Кроме того, система поддерживает интеграцию различных устройств, таких как видеокамеры для наблюдения за обстановкой внутри и вокруг дома, сирены для предупреждения о возможных инцидентах, «умные розетки» для управления электроприборами издалека, а также датчики температуры, влажности и протечки воды для контроля за условиями в помещении и предотвращения аварийных ситуаций.

Система предлагает гибкие варианты оповещения пользователей о событиях. Оповещение может осуществляться через Push-уведомления на мобильных устройствах, SMS-сообщения или электронные письма. Пользователи могут выбирать наиболее удобный для себя способ оповещения или использовать несколько вариантов одновременно.

Благодаря своей гибкости и простоте использования, система «Умный дом» от «Белтелеком» пользуется популярностью среди пользователей, которые ценят возможность настройки системы под свои потребности и удобные варианты оповещения.



Рисунок 1.3 – «Умный дом» от «Белтелеком»

Данный продукт также имеет ряд недостатков:

– зависимость от оператора;

– процесс подключения и настройки устройств может быть сложным для некоторых пользователей.

1.1.3 «Умный дом» от «Xiaomi»

Данный производитель предоставляет широкий ассортимент разнообразной техники, в том числе технику, которая отвечает требованиям «умного дома» [3]. В их ассортименте можно найти устройства для обеспечения безопасности и поддержания чистоты в помещении, что делает их продукцию универсальным решением для создания современного «умного дома».

Для удобного управления и взаимодействия с этими устройствами предоставляется специальное приложение (см. рисунок 1.4), обеспечивающее полную совместимость устройств одной экосистемы и их простую настройку. Благодаря фокусу данного производителя на широкий потребительский рынок, все устройства легко подключаются и не требуют значительных временных затрат на настройку.

Тем не менее, стоит отметить, что у продукции этого производителя есть и некоторые недостатки. Например, некоторые датчики могут быть дешевыми и менее надежными, а обновления приложения для «умного дома» выпускаются не так часто, как хотелось бы. Кроме того, большинство устройств требуют постоянного подключения к электросети и Wi-Fi для полноценной работы.

Таким образом, данный производитель предлагает доступное решение для знакомства с концепцией «умного дома. Однако, у них также имеются и более дорогие варианты аппаратуры с расширенными функциональными возможностями и повышенной надежностью.

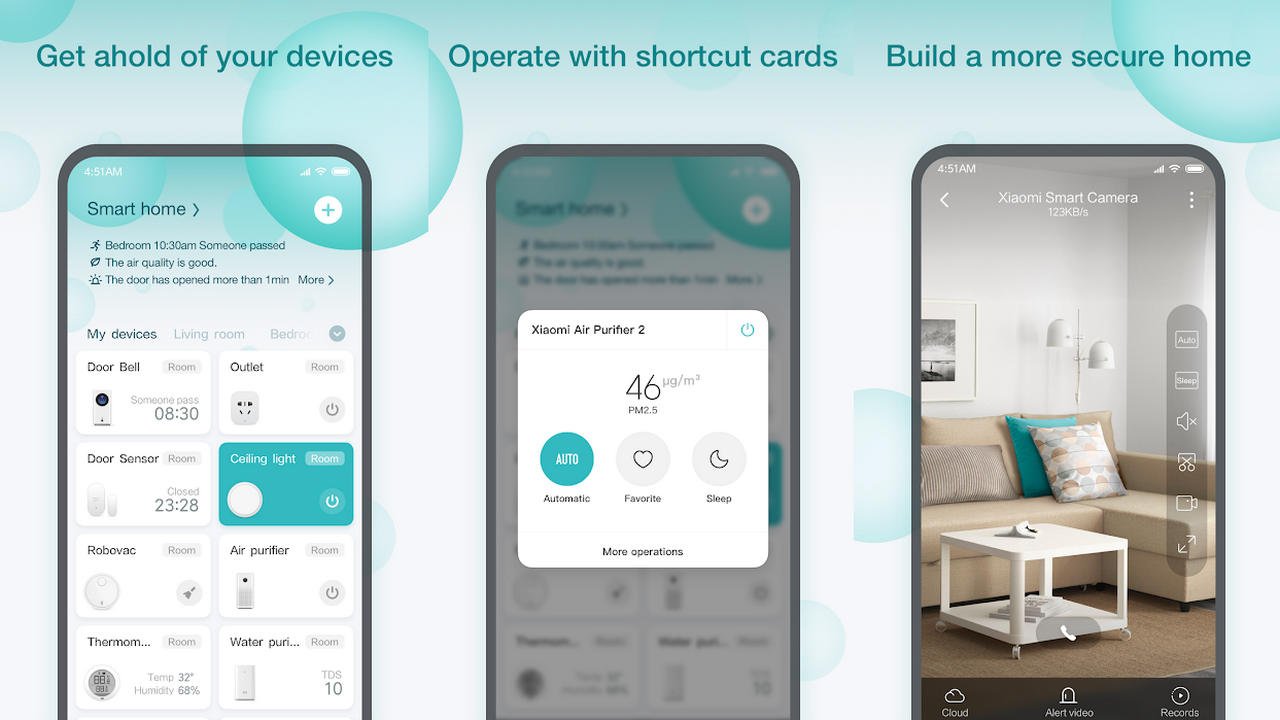


Рисунок 1.4 – Приложение Xiaomi Smart Home

1.1.4 Координатор от Яндекс

Для обеспечения функционирования всех устройств в «умном доме» необходимо наличие центрального устройства, которое выступает в качестве координатора, обеспечивая связь и взаимодействие между пользователем и системой. Одним из вариантов такого решения являются продукты компании Яндекс [4], предоставляющие широкий ассортимент устройств, способных выполнять функции координаторов (см. рисунок 1.5). Эти устройства не только обеспечивают надежное соединение и взаимодействие с другими устройствами в комплексе, но также предоставляют пользователю удобный интерфейс для управления всем комплексом. Благодаря интеграции с различными экосистемами и голосовыми ассистентами, устройства от Яндекса обеспечивают гибкость и удобство использования.

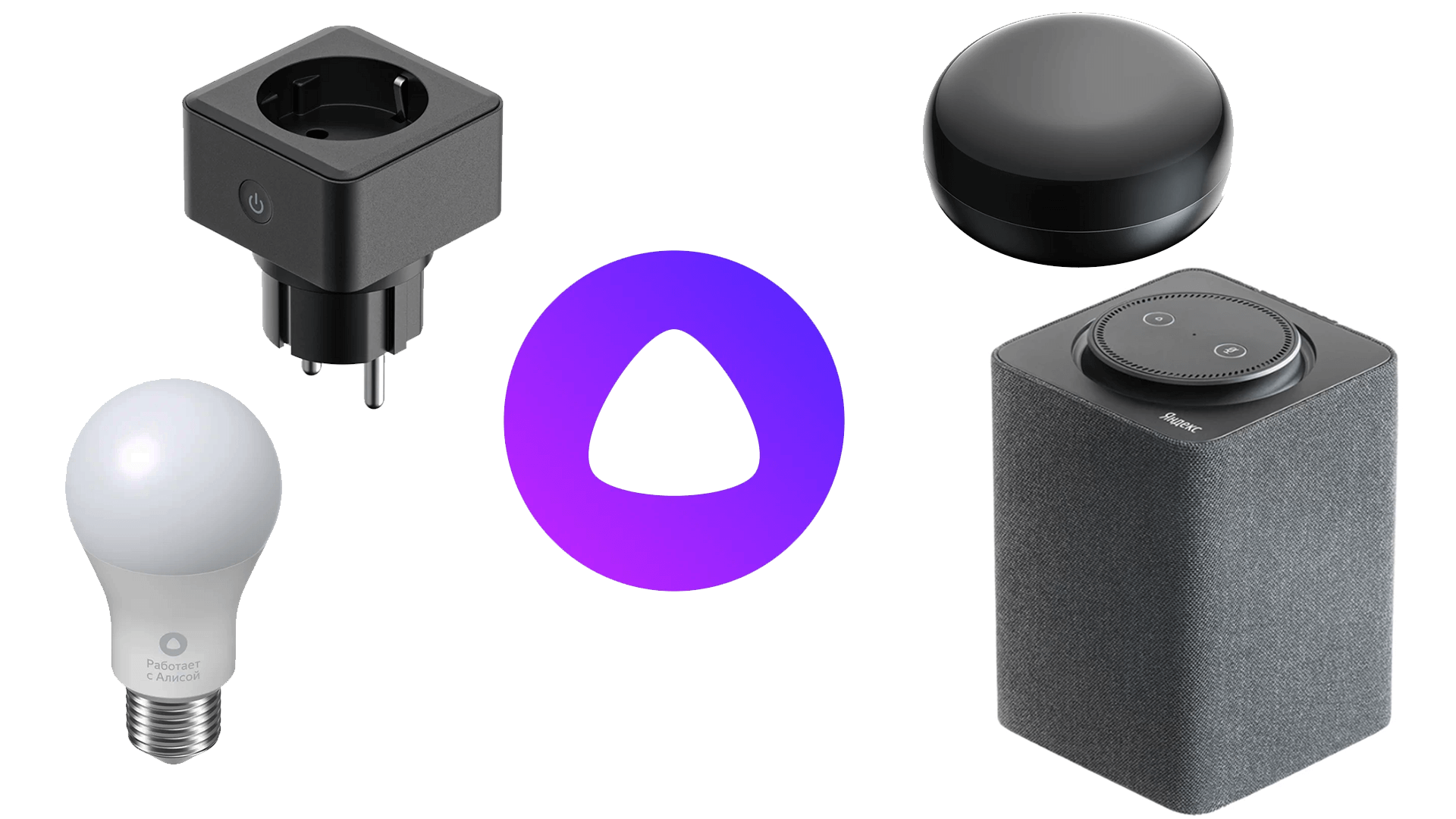


Рисунок 1.5 – «Умный дом» от Яндекс

Наиболее распространённым вариантом устройства, выступающего в роли координатора в «умном доме», является «умная колонка» с голосовым помощником Алиса (см. рисунок 1.6). Это устройство способно подключать различные оконечные устройства для управления всеми аспектами «умного дома».

«Умная колонка» Алиса предоставляет возможность управления, информирования и выполнения различных функций как с использованием мобильного приложения, так и с помощью голосовых команд. Это делает взаимодействие с «умным домом» более удобным и интуитивно понятным для пользователей, позволяя им контролировать своё пространство с легкостью и эффективностью.

Без интернет-соединения ПО неспособно функционировать должным образом. Обработка голосовых команд происходит не на устройстве, а на серверах «Яндекса». После отправки команды «Алисе» она передается на сервер для обработки, а затем возвращается на устройство для выполнения. Даже при значительной задержке процесс обработки остается быстрым и эффективным.

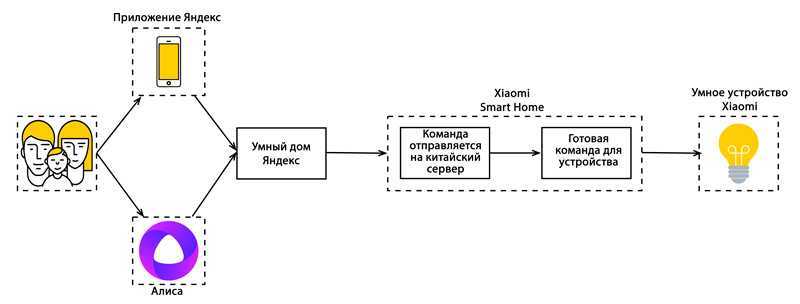


Рисунок 1.6 – Схема запросов в «Умном доме» от Яндекс

«Умный дом» от Яндекса уже интегрирован с множеством устройств от таких производителей, как Philips, Redmond, Rubetek, Samsung и Xiaomi (см. рисунок 1.7). В дополнение компания разработала и представила собственные устройства, включая «умную лампочку, розетку и пульт». Эти устройства обладают различным функционалом, который дополняет экосистему «умного дома».



Рисунок 1.7 – Совместимые устройства с координатором от Яндекс

Плюсом выбора продукта от Яндекс является то, что устройства данной компании прекрасно взаимодействуют с устройствами популярных производителей, работающими по протоколу Zigbee. Это позволяет пользователям легко интегрировать разнообразные устройства в свой «умный дом», выбирая из широкого ассортимента рынка и создавая гибкую систему под свои нужды.

Дополнительно, управление системой осуществляется с помощью мобильного приложения с современным интерфейсом (см. рисунок 1.8), что делает использование и контроль «умного дома» максимально удобным для пользователя. Это приложение позволяет настраивать интерфейс в соответствии с индивидуальными предпочтениями и потребностями, обеспечивая интуитивно понятное и персонализированное взаимодействие с системой. Такой подход делает использование «умного дома» более эффективным и удобным для каждого пользователя.



Рисунок 1.8 – Мобильное приложение от Яндекс

Однако, наряду с многочисленными преимуществами, у данного решения имеются и определенные недостатки:

1. Подписочная система. Для полноценного функционирования некоторых возможностей требуется подписка, что может стать дополнительным финансовым бременем для пользователей.
2. Высокая стоимость оборудования. Несмотря на качество и функциональность устройств, их стоимость может быть значительной, что делает данное решение недоступным для определенного сегмента потребителей.

В итоге, хотя данный производитель предлагает весьма привлекательные решения с широкими возможностями интеграции в домашнюю среду, высокая стоимость оборудования и необходимость в подписке могут создать некоторые препятствия для пользователей.

## 1.2 MQTT-протокол

В силу того, что большинство «умных домашних устройств» находятся в локальной сети, для того чтобы они могли функционировать в качестве устройств интернета вещей (IoT), необходимо осуществить их подключение к сети Интернет. Благодаря этому, собирать данные с датчиков и управлять конечными устройствами «умного дома» будет возможно из любой точки планеты. Подобный подход существенно упрощает коммуникацию между человеком и системой, ведь в настоящее время доступ в Интернет из мобильных устройств обеспечен почти во всех частях мира.

В большинстве случаев для управления конечными устройствами «умного дома» используется телефон с приложением системы, и в редких случаях веб-версии приложений данных систем. Такой подход делает мобильный телефон еще одним устройством IoT, содержащий различные возможности, включая экран для визуализации информации, полученной от датчиков, и контроллера конечных устройств.

В сети «умного дома» конечные устройства и координатор общаются при помощи протоколов Zigbee, Z-Wave и др., но для коммуникации между координатором и мобильным телефоном данный протокол не подходит. Для данной цели более приемлим протокол MQTT (Message Queue Telemetry Transport) [6], являющийся протоколом потоковой передачи информации с ограниченой производительностью процессора, то есть устройствами IoT. Протокол MQTT, построенный на основе TCP/IP, используется в сетях с низкой пропускной способностью, что делает его идеальным выбором для «умного дома».

Отличия протокола MQTT, выделяющие его среди других протоколов:

* нейтрален к содержимому сообщения;
* подходит для дистанционных коммуникаций «один ко многим» и приложений с разделением;
* имеет функцию LWT (Last Will and Testament, «последняя воля и завещание») для оповещения о нештатном отключении клиента;
* основан на протоколе TCP/IP для основных коммуникационных задач;
* разработан для доставки сообщений с определенной частотой: «максимум один раз», «минимум один раз» и «ровно один раз».

Система связи, использующая MQTT (см. рисунок 1.9), включает в себя издателей, сервер-брокер и одного или нескольких клиентов (подписчиков). Издатель не нуждается в настройке для определения количества или расположения подписчиков, получающих сообщения. Подписчики также не требуют настройки для определения конкретного издателя. В системе может присутствовать несколько брокеров, распространяющих сообщения. Издатель помещает сообщения в темы (topic), откуда их могут читать клиенты, подписанные на данную тему. Издатель и подписчик не могу коммуницировать напрямую, поэтому они не знают о существовании друг друга. В это же время подписчики могут читать сообщения из многих тем, на которые они подписаны, как и издатели имеют возможность помещать свои сообщения в различные темы.

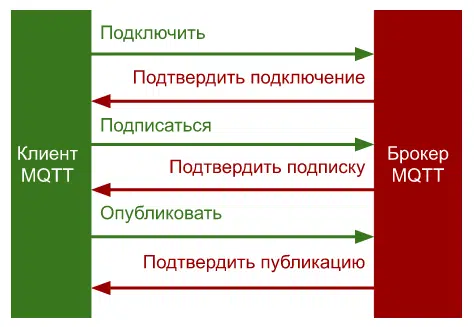


Рисунок 1.9 – Схема простого взаимодействия по протоколу MQTT

Данные, отправленные или полученные через брокер MQTT, будут представлены в двоичном формате, поскольку MQTT является бинарным протоколом. Это означает, что для того чтобы понять содержимое сообщения, необходимо выполнить интерпретацию двоичных данных.

Топики в MQTT составлены из символов в кодировке UTF8 и имеют структуру, аналогичную дереву файловой системы в UNIX. Этот подход обеспечивает удобство в названии сущностей таким образом, что они понятны человеку. Например:

– home/kitchen/temperature;

– home/sleeping-room/pressure;

– home/outdoor/light.

Такая организация позволяет наглядно отслеживать передаваемые данные и упрощает разработку и отладку кода, не требуя запоминания числовых адресов расположения данных.

Устройства, использующие MQTT, взаимодействуют с брокером посредством определенных типов сообщений. Ниже перечислены основные из них:

* Connect – установка соединения с сервером-брокером;
* Disconnect – оборвать соединение с сервером-брокером;
* Publish – опубликовать данные в топик;
* Subscribe – подписаться на топик;
* Unsubscribe – отписаться от топика.

Для удобства в уже готовых системах каждому конечному устройству назначается свой топик с определенными вложенными разделами. Именно по этим топикам происходит рассылка уведомлений о изменениях определенных параметров и управление устройствами. Когда сообщения публикуются в соответствующий топик, устройство, подписанное на этот топик, получает сообщение и реагирует на него.

## 1.3 Протоколы передачи данных

1.3.1 Протокол I2С

I2C (Inter–Integrated Circuit)[7] – это протокол последовательной связи, который обеспечивает взаимодействие между различными компонентами внутри электронных устройств.

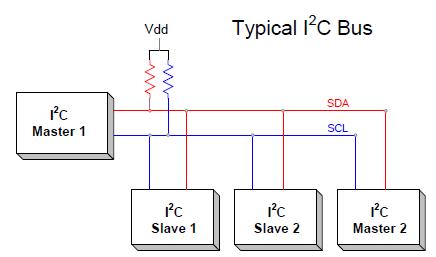


Рисунок 1.11 – Схема шины I2С

Применение данного стандарта для передачи информации способствует сокращению количества взаимосвязей между интегральными схемами, что приводит к уменьшению числа требуемых контактов и дорожек. Также встроенный I2C-протокол исключает необходимость в дополнительных средствах, таких как дешифраторы адресов, и другой внешней логике для согласования.

Для передачи информации используются лишь две линии (см. рисунок 1.11):

* SDA (для данных);
* SCL (для синхронизации).

Каждое устройство определяется как ведущее или ведомое, а также обладает уникальным (в пределах шины) адресом. Максимальное количество устройств, подключаемых к одной шине, ограничено ее емкостью – 400 пФ.

Ведущее устройство – то, которое инициирует передачу данных и контролирует сигналы синхронизации, а также завершает передачу данных. Оно может отправлять данные или запрашивать их.

Ведомое устройство – любое адресуемое устройство по отношению к ведущему. Оно может принимать данные или отправлять их по запросу ведущего.

Передатчик – устройство, передающее данные по шине.

Приемник – устройство, принимающее данные с шины.

Отношения ведущий-ведомый и приемник-передатчик не постоянны и зависят от направления передачи данных в текущий момент времени. Обычно стандарт I2C предполагает наличие одного ведущего устройства в один момент времени, однако допускается наличие нескольких ведущих устройств на шине без нарушения работы системы. В таком случае применяется процедура арбитража для предотвращения ошибок.

Каждое прикрепленное устройство имеет свой собственный адрес. Производитель задает первые три бита адреса, а последний бит указывает на запись или чтение устройства. Такая структура обеспечивает простую передачу данных. 7-битная адресация шины I2C позволяет подключить до 128 устройств.

1.3.2 Протокол Zigbee

Zigbee – это беспроводной протокол, специально разработанный для создания сетей малой дальности с низким энергопотреблением [8]. Он широко используется для передачи данных в устройствах Интернета вещей (IoT) и смарт-домах. Протокол Zigbee работает в частотном диапазоне 2,4 ГГц и использует технологию малого радиуса действия, что обеспечивает низкое энергопотребление и надежную связь. Основные характеристики протокола включают в себя низкое энергопотребление, минимальные задержки передачи данных, возможность создания крупных сетей до тысяч устройств, самоорганизацию и безопасность данных.

ZigBee устанавливает ячеистую сеть устройств, соединенных между собой с помощью маломощных радиосигналов. Всякое устройство в сети, действуя как маршрутизатор, может принимать и перенаправлять сообщения другим устройствам, что обеспечивает избыточность связи и повышает надежность сети.

Уникальная схема адресации в ZigBee позволяет однозначно идентифицировать устройства в сети. Это обеспечивает возможность использования функций безопасности, таких как шифрование и аутентификация, для защиты сети от несанкционированного доступа.

Сети Zigbee имеют несколько типов структур, которые могут быть использованы в различных сценариях в зависимости от требований к конкретному приложению. Вот некоторые из основных структур сетей Zigbee:

1. Звезда (Star). В звездной структуре одно устройство, обычно координатор сети, служит центром и связывается напрямую со всеми другими устройствами в сети. Это простая и надежная структура, но она имеет ограничения по расстоянию и масштабируемости.
2. Дерево (Tree). В данной структуре устройства связаны между собой в иерархической форме, создавая иерархию маршрутизаторов и конечных устройств. Это позволяет увеличить дальность связи и масштабируемость сети.
3. Сеть с маршрутизацией (Mesh). Сеть с маршрутизацией является наиболее распространенной структурой в Zigbee. Она позволяет устройствам действовать как маршрутизаторы, перенаправляя сообщения между устройствами, даже если они находятся вне прямой радиочастотной области. Это обеспечивает более высокую надежность и устойчивость связи.
4. Кластерные структуры. Эти структуры позволяют группировать устройства по функциональным кластерам или областям, что облегчает организацию и управление сетью.

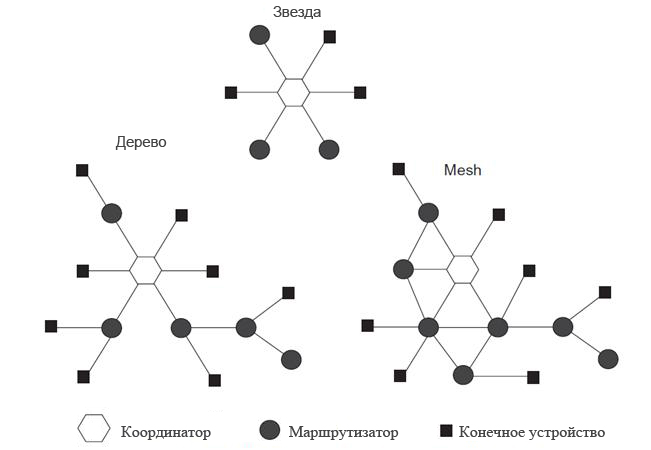


Рисунок 1.12 – Топология Zigbee

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

При разработке программного обеспечения для «умного дома» ключевым моментом является структурирование его компонентов в логические блоки, каждый из которых отвечает за определенные функциональные задачи.

Целью разбиения программного обеспечения на логические блоки является облегчение понимания его работы и структуры, а также упрощение процесса разработки и поддержки. Каждый блок выполняет определенные функции и имеет четко определенный интерфейс для взаимодействия с другими блоками.

Рассмотрим следующие основные блоки структуры программного обеспечения:

– блок интерфейса приложения;

– блок соединения с брокером MQTT;

– блок приема-передачи данных брокеру MQTT;

– блок обработки данных;

– блок базы данных;

– блок тестирования;

– блок координатора;

– блок конечных устройств;

– блок MQTT брокера.

Структурная схема, изображенная на чертеже ГУИР.400201.006 С1, иллюстрирует взаимосвязь между различными блоками программного обеспечения.

В конечном итоге, все модули блока взаимодействуют и образуют единую систему, позволяющую корректно реагировать на запросы и изменения субъекта. В числе прочего, это позволяет обеспечивать пользователю интерфейс и комфорт взаимодействия с системой умного дома, путем предоставления ему актуальной информаци и выполнения необходимых функций. Далее будет описано именно практическое назначение и способ работы каждого из блоков, для наглядности их реализации.

## 2.1 Блок пользовательского интерфейса

Блок интерфейса приложения играет ключевую роль в системе «умного дома», поскольку он обеспечивает взаимодействие пользователя с системой. Его основной задачей является создание удобного и интуитивно понятного пользовательского интерфейса, который позволяет пользователю управлять и мониторить различными компонентами «умного дома». Ниже приведено подробное описание функциональности и особенностей блока интерфейса приложения:

1. Пользовательский опыт. Блок интерфейса разрабатывается с учетом пользовательского опыта, чтобы обеспечить удобство использования для широкого круга пользователей. Это включает в себя понятную навигацию, простоту в использовании элементов управления и понятные инструкции.
2. Визуализация данных. Интерфейс приложения предоставляет пользователю визуализацию данных о состоянии устройств в системе и системы в целом. Это может быть представление в виде графиков, диаграмм, таблиц или других подходящих форматов, которые помогают пользователю лучше понять текущее состояние системы.
3. Элементы управления. Блок интерфейса содержит различные элементы управления, позволяющие пользователю взаимодействовать с компонентами «умного дома». Это могут быть кнопки, ползунки, переключатели, поля ввода и другие элементы, позволяющие изменять параметры устройств, включать и выключать функции, а также добавлять свои системы и устройства в них.
4. Отображение уведомлений. Интерфейс приложения также поддерживает отображение уведомлений пользователю о различных событиях и состояниях системы. Это могут быть предупреждения о срабатывании сигнализации, информация о текущих работающих процессах, оповещения о необходимости обслуживания устройств и т.д.
5. Настройки и персонализация. Блок интерфейса предоставляет пользователю возможность настройки и персонализации системы в соответствии с его предпочтениями. Это могут быть настройки устройств и другие параметры, которые делают систему более удобной и функциональной для конкретного пользователя.
6. Адаптивность приложения. В случае мобильного приложения блок интерфейса также оптимизирован для работы на мобильных устройствах, обеспечивая удобство использования на смартфонах и планшетах. Это включает в себя адаптивный дизайн, оптимизированный для различных размеров экранов, а также использование жестов для управления.

Благодаря функциональности и особенностям блока интерфейса приложения пользователь может легко и удобно управлять системой «умного дома», получая необходимую информацию и выполняя нужные действия в соответствии с его потребностями и предпочтениями.

При разработке проектов под Android с использованием IDE Android Studio широко используется подход с определением визуального интерфейса в специальных файлах XML. Эти XML-файлы являются ресурсами разметки и содержат определения визуального интерфейса в формате XML-кода. Этот подход аналогичен использованию HTML-файлов для определения веб-интерфейса в веб-разработке, где логика приложения отделена от визуальной составляющей и реализована с помощью JavaScript.

Использование XML для объявления пользовательского интерфейса позволяет отделить интерфейс приложения от его логики, что позволяет изменять визуальное оформление без необходимости внесения изменений в Java-код. Например, в приложении можно определить различные разметки для разных ориентаций экрана, разных размеров устройств, различных языков и так далее. Кроме того, объявление интерфейса в XML упрощает визуализацию его структуры и облегчает процесс отладки, что делает разработку более эффективной и удобной.

После объявления в XML, компоненты становятся доступными для манипуляций из основного кода приложения. Это достигается путем связывания XML-компонентов с объектами в Java-коде с помощью механизма «надувания», который позволяет получить доступ к компонентам по их уникальному идентификатору. После связывания компоненты можно изменять, управлять и реагировать на действия пользователя из основной части проекта.

## 2.2 Блок соединения с брокером MQTT

Блок соединения с брокером MQTT является важной частью системы «умного дома», ответственной за установление и поддержание связи с сервером MQTT. Этот блок обеспечивает надежное соединение между приложением и брокером MQTT, что позволяет обмениваться сообщениями и данными между устройствами системы.

Основные задачи блока соединения с брокером MQTT включают:

1. Установка соединения. Приложение должно установить соединение с сервером MQTT перед тем, как пользователь открывает страницу с устройствами, чтобы начать отправку и получение сообщений для отображения актуальных данных, если это возможно. Этот процесс включает в себя инициализацию сетевого соединения и передачу необходимых параметров для аутентификации и подключения к серверу MQTT.
2. Поддержание связи. После установления соединения блок должен поддерживать активное соединение с сервером MQTT, чтобы обеспечить непрерывный обмен сообщениями. Это включает в себя периодическую отправку «ping» запросов для подтверждения активности соединения и переподключение в случае его разрыва.
3. Обмен сообщениями. После успешного установления и поддержания соединения блок должен обеспечивать передачу сообщений между приложением и сервером MQTT. Это включает в себя отправку сообщений от приложения к брокеру для управления устройствами «умного дома» и получение обновлений состояния устройств от брокера.
4. Обработка ошибок. Блок должен обрабатывать возможные ошибки при установлении и поддержании соединения с брокером MQTT, такие как потеря сетевого соединения или недоступность сервера. В случае ошибок он должен предпринимать соответствующие действия для восстановления связи и продолжения работы.

## 2.3 Блок приема-передачи данных MQTT брокеру

Блок приема-передачи данных MQTT брокеру выполняет функцию обмена сообщениями между приложением и брокером MQTT в системе «умного дома». Этот блок играет важную роль в передаче управляющих команд от приложения к устройствам «умного дома» через брокер MQTT, а также в получении обновлений состояния устройств от брокера и передаче их в приложение для дальнейшей обработки.

Основные задачи блока приема-передачи данных MQTT брокеру включают отправку и получение сообщений, а также управление подписками на топики MQTT и обработку ошибок. Блок отправляет управляющие команды от приложения к устройствам «умного дома» через брокер MQTT и получает от них сообщения с обновлениями состояния устройств или другой важной информацией. Полученные сообщения обрабатываются блоком для извлечения нужной информации и принятия соответствующих действий. Блок также управляет подписками на топики MQTT, определяя, на какие топики приложение подписано для получения сообщений от брокера, что позволяет эффективно управлять потоком данных и получать только необходимую информацию. При возникновении ошибок блок обрабатывает их, предпринимая соответствующие действия для восстановления связи и обеспечения непрерывной работы системы.

## 2.4 Блок обработки данных

Информация, полученная из различных источников в блок обработки данных, где преобразовываются для последующего использования в различных блоках.

Блок получает «сырые сообщения» в виде JSON от брокера MQTT, содержащие информацию о состоянии устройств в системе «умного дома» или другие важные данные. Эти сообщения могут содержать информацию о температуре, освещенности, статусе устройств и других параметрах, которые необходимо обработать и отобразить пользователю в виде виджетов.

В функциональную часть блока также входит обязанность принимать данные, введенные пользователем через поля ввода в интерфейсе приложения. Эти данные могут включать в себя команды управления устройствами, настройки параметров системы и другую информацию, которую необходимо обработать и применить.

Кроме того, блок обрабатывает события, связанные с действиями пользователя, такие как нажатие кнопок для переключения состояния светодиодов и других управляемых устройств. Он интерпретирует эти события и принимает соответствующие действия, например, отправку сформированной команды на изменение состояния устройства.

Информация, приходящая от полей ввода, связанных с данными системы и ее устройствами передаются в блок обработки данных. Далее происходит создание и поддержка внутренней модели системы «умного дома», включая список доступных устройств, их параметры и текущее состояние. Он обновляет эту модель на основе полученной информации от MQTT и действий пользователя, чтобы обеспечить актуальное состояние системы и корректное функционирование устройств, например, добавление последних полученных данных от датчика, для последующего корректного отображения информации при перезагрузке приложения.

## 2.5 Блок базы данных

Информация, получаемая из блока обработки данных, записывается в базу данных, где она сохраняется в специальных таблицах, предназначенных для хранения данных о системе и устройствах. Чаще всего для этого используется SQLite.

В таблице «системы» хранятся данные, необходимые для установления связи с брокером MQTT, такие как адрес сервера, порт, учетные данные и название системы для удобства отображения.

В таблице «устройства» содержатся поля для хранения последних данных, полученных от MQTT брокера, уникальный адрес устройства для подключения к сети Zigbee, изображение самого устройства, выбранное пользователем, каналы управления (например, для переключения датчика или светодиода), тип устройства и его дружественное имя.

Данная информация позволяет системе эффективно управлять подключением к MQTT и обеспечивать надежную передачу данных в системе умного дома. Информация может храниться в базе данных в «сыром виде», что обеспечивает гибкую обработку данных для различных задач в приложении.

## 2.6 Блок тестирования

Блок тестирования представляет собой важный компонент разработки комплекса «умного дома», ответственный за проверку корректности работы приложения и его компонентов. В этом блоке проводятся несколько видов тестирования, включая модульное и пользовательское тестирование, с целью обнаружения и устранения ошибок и недочетов в приложении.

Модульное тестирование направлено на проверку отдельных модулей приложения, таких как блоки обработки данных, соединения с брокером MQTT и интерфейса приложения. Пользовательское тестирование проводится для оценки удобства использования приложения и выявления потенциальных проблем, с которыми может столкнуться конечный пользователь.

Блок тестирования включает в себя разработку тестовых сценариев, проведение тестов, анализ результатов и исправление выявленных ошибок. Целью этого блока является обеспечение качества и надежности приложения перед его выпуском в продакшн и после каждого обновления. Тестирование позволяет выявить и устранить проблемы до того, как они окажутся в реальных условиях эксплуатации, что способствует повышению удовлетворенности пользователей и обеспечивает успешное функционирование комплекса «умного дома». Кроме того, благодаря систематическому тестированию, становится возможным внедрять новые функции и улучшения, минимизируя риск возникновения непредвиденных проблем.

## 2.7 Блок координатора

Блок координатора ответственен за взаимодействие с устройствами Zigbee в системе умного дома. Он обеспечивает связь с устройствами Zigbee, управляет ими и получает от них данные. Координатор выполняет конвертацию данных, полученных от устройств Zigbee, в формат, понятный брокеру MQTT, и обратно, обеспечивая эффективную передачу информации между устройствами и приложением.

В состав блока координатора входит прошивка устройства, которая представляет собой программное обеспечение, устанавливаемое на устройство координатора.

Координатор подключается к брокеру MQTT, обеспечивая передачу данных между конечными устройствами и мобильным приложением пользователя. Это позволяет системе умного дома контролировать и управлять устройствами через стандартный протокол обмена сообщениями.

Сопряжение конечных устройств с координатором осуществляется через протокол Zigbee, который обеспечивает беспроводную связь между ними. После сопряжения устройства могут обмениваться информацией о состоянии с координатором и участвовать в работе комплекса.

Для дополнительной настройки и управления координатором можно использовать HTTP протокол. Кроме того, координатор позволяет использовать Lua скрипты. Lua - это легкий и гибкий скриптовый язык программирования, который позволяет создавать сценарии для автоматизации различных задач. Например, с помощью Lua скриптов можно настраивать расписания работы устройств, настраивать реакцию системы на определенные события или условия, а также выполнять другие задачи, улучшающие функциональность и удобство использования комплекса.

## 2.8 Блок конечных устройств

Блок конечных устройств в системе умного дома выполняет задачу получения данных от датчиков и формирования соответствующих кластеров для передачи в сети Zigbee. Кластеры представляют собой логические группы данных, которыми устройства Zigbee обмениваются между собой. Каждый кластер имеет свое определенное назначение, например, кластер может содержать данные о температуре, влажности, освещенности и т.д., в зависимости от типа датчика.

Конечные устройства, такие как датчики, работают на уровне протокола Zigbee и формируют данные в соответствии с необходимыми кластерами. Например, датчик температуры будет отправлять данные о текущей температуре в специально определенный кластер для температурных данных.

Полученные кластеры данных затем передаются в сеть Zigbee и доставляются до координатора. Координатор, в свою очередь, правильно интерпретирует эти данные, конвертируя их в формат, понятный брокеру MQTT.

Кроме того, для работы конечных устройств необходимо настроить контакты, которые используются для подключения датчиков или для взаимодействия с ними. Настройка контактов устройства позволяет определить, какие сигналы будут приниматься или отправляться через эти самые контакты, что важно для правильного взаимодействия между устройствами.

## 2.9 Блок MQTT брокера

Блок MQTT брокера представляет собой центральный элемент в системе умного дома, отвечающий за обмен сообщениями между всеми устройствами и компонентами системы. Основные функции блока MQTT брокера включают в себя:

1. Прием и передача сообщений. Брокер MQTT принимает сообщения от различных устройств и компонентов системы, а затем передает их нужным адресатам в соответствии с подписками на топики.
2. Маршрутизация сообщений. Он обеспечивает маршрутизацию сообщений между различными устройствами и компонентами системы, гарантируя доставку сообщений тем получателям, которые подписаны на соответствующие топики.
3. Управление подписками. Брокер управляет подписками на топики MQTT, определяя, какие устройства и компоненты получают определенные сообщения.

Установка сервера MQTT на публичный адрес обеспечивает доступ к системе умного дома из любой точки мира, где есть доступ к интернету. Это означает, что пользователи могут контролировать и управлять своими «умными устройствами» удаленно, независимо от их местоположения, используя приложение.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Функциональное проектирование направлено на разработку корректно и исправно работающего проекта. В данном разделе основное внимание уделяется определению необходимого функционала.

Будут описаны логические блоки приложения, их классы, методы и выполняемая ими функциональность. Представлена диаграмма классов на чертеже ГУИР.400201.006 РР.1 и диаграмма последовательности на чертеже ГУИР.400201.006 PP.2.

Поскольку дипломный проект использует уже реализованные ранее координатор и переферийное устройство, то следует описать данные устройства. Раздел разделен на две части: аппаратную, в которой и описываются используемые устройства, и программную.

## 3.1 Аппаратная часть

## 3.1.1 Координатор

Координатор – ключевое устройство в системе «умного дома», которое не только выстраивает и контролирует сеть, но и обеспечивает взаимодействие между всеми компонентами. Он является центром системы, отвечая за управление всей инфраструктурой и обеспечивая коммуникацию с каждым устройством.

Координатором в данном проекте выступают микроконтроллер ESP32 WROOM и многопротокольный радиомодуль E72. ESP32 WROOM имеет Wi-Fi модуль, что позволяет мобильным устройствам взаимодействовать с координатором. С другой стороны, модуль связи E72, основанный на CC2652P, способен работать как по протоколу Zigbee, так и Z-Wave.

ESP32 WROOM отвечает за передачу данных к пользователю через сеть Wi-Fi и обработку этих данных. Связь с мобильными устройствами и передача команд от пользователя к системе осуществляется через данный модуль.

Радиомодуль E72, работая по протоколу Zigbee, отвечает за связь с периферийными устройствами и передачу данных от них к координатору. Этот модуль выполняет важную функцию взаимодействия с устройствами, поддерживающими Zigbee, такими как датчики, «умные выключатели» и другие элементы системы «умного дома».

## 3.1.2 Переферийное устройство

Радиомодуль Е18 – периферийное устройство в системе «умного дома». Как и у E72, функционал E18 может работать по стандарту Zigbee. Основная особенность этого элемента двоякая: во-первых, радиомодуль считывает данные с подключенных к нему датчиков и передает полученную информацию координатору; во-вторых, микроконтроллер получает команды от координатора через стандартный Zigbee и на основании этой информации меняет режим устройства. Датчики и устройства, подключаемые к модулю связи, представляют собой вставные модули с несколькими контактами. Контакты используются для питания датчиков и передачи данных на радиомодуль. Модулями могут быть датчики температуры, влажности, движения и т. д., а также устройства управления, например, реле или сервоприводы. При получении определенного сигнала от координатора с помощью Zigbee они меняют состояние устройства или передают свое текущее состояние.

Модуль связи Е18 способен вести запись некоторых данных и по запросу отправлять их. Это является хорошим способом оптимизации процесса хранения и организации данных

## 3.2 Программная часть

## 3.2.1 Класс MainActivity

Класс MainActivity представляет основной экран приложения. Описание его методов и атрибутов:

Атрибуты:

* dbHelper – экземпляр класса DBHelper, используемый для взаимодействия с базой данных, а именно для получения списка всех добавленных систем;
* systems – список объектов класса System, представляющих системы в приложении.

Методы класса:

* onCreate(Bundle savedInstanceState) – этот метод вызывается при создании экземпляра класса, отвечающий за инициализацию пользовательского интерфейса, путем вызова встроенной функции setContentView, и установку обработчиков событий;
* setListeners() – метод, который устанавливает обработчик события нажатия на кнопку добавления новой системы, вызывающую при нажатии активность SecondActivity для добавления новой системы;
* setListItems() – этот метод отвечает за заполнение списка систем на главном экране приложения и извлекает данные о системах из базы данных, создает адаптер для списка и устанавливает обработчик событий для элементов списка;
* getAdapter() – метод, который создает адаптер для списка систем и извлекает названия всех систем из списка и создает адаптер, который отображает эти названия в списке, а также список устройств, подключенных к каждой системе в качестве подзаголовка.

Также в классе MainActivity происходит связывание с пользовательским интерфейсом, описанным в файле activity\_main.xml. Этот файл содержит разметку главного экрана приложения, включая компоненты пользовательского интерфейса, такие как кнопки, списки и текстовые поля.

## 3.2.2 Класс SecondActivity

SecondActivity является активностью в приложении и используется для отображения вторичного экрана, в котором происходит добавление системы. Этот экран создан с использованием навигации через фрагменты.

Методы класса:

1. OnCreate(Bundle savedInstanceState) – метод вызывается при создании активности. Внутри него происходит установка разметки из файла activity\_second.xml, настройка виджета Toolbar, инициализация NavController для навигации между фрагментами.
2. OnSupportNavigateUp() – метод обрабатывает нажатие на кнопку «Назад» на панели инструментов. Он находит NavController и вызывает метод navigateUp() для обработки навигации назад. Если навигация обработана, метод возвращает true, иначе вызывается метод super.onSupportNavigateUp().

SecondActivity также связан с файлом activity\_second.xml, который определяет внешний вид вторичного экрана, а также с фрагментами FirstFragment и SecondFragment, отвечающими за логику и взаимодействие пользователя с интерфейсом на данном экране.

## 3.2.3 Класс FirstFragment

Класс FirstFragment представляет собой часть пользовательского интерфейса, отвечающего за ввод информации о новой системе, в частности об MQTT брокере.

В классе используется единственный атрибут binding, который является объектом класса FragmentFirstBinding, представляющий собой привязку к элементам пользовательского интерфейса в макете фрагмента fragment\_first.xml. Этот объект используется для доступа к элементам пользовательского интерфейса (например, полям ввода, кнопкам) и их свойствам в коде фрагмента. Привязка создается при помощи метода FragmentFirstBinding.inflate(inflater, container, false) в методе onCreateView фрагмента. После создания привязка содержит ссылки на все элементы пользовательского интерфейса, определенные в макете фрагмента. Использование привязки позволяет избежать необходимости вызывать методы findViewById() для каждого элемента пользовательского интерфейса отдельно, что делает код более компактным и читаемым.

Методы класса:

1. OnCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) – метод создает и настраивает макет фрагмента, который определен в файле fragment\_first.xml. Внутри метода устанавливаются фильтры для полей ввода IP-адреса и порта.
2. ValidateData() – метод осуществляет валидацию введенных данных. Проверяется, заполнены ли обязательные поля (IP-адрес и порт), а также корректность порта. Если данные введены верно, метод возвращает Bundle, содержащий информацию об адресе MQTT брокера, а также, при наличии, логин и пароль.
3. IsValidIPAddress(String ipAddress) – метод проверяет корректность введенного IP-адреса с помощью регулярного выражения.
4. IsValidPort(String port) – метод проверяет корректность введенного порта с помощью регулярного выражения.
5. OnViewCreated(View view, Bundle savedInstanceState) – метод вызывается после того, как корневой элемент представления фрагмента был создан. Здесь устанавливаются обработчики нажатия кнопок, такие как «Далее» и «Отмена». При нажатии на кнопку «Далее» осуществляется валидация данных и, в случае успешной валидации, происходит навигация на следующий фрагмент SecondFragment. При нажатии на кнопку «Отмена» происходит навигация на главный экран приложения.

Файл fragment\_first.xml содержит макет пользовательского интерфейса фрагмента, который включает в себя поля ввода для IP-адреса, порта, логина и пароля, а также кнопки для перехода на другие экраны или выполнения действий.

## 3.2.4 Класс SecondFragment

Класс SecondFragment отвечает за второй экран в процессе добавления системы. В нем пользователь вводит название системы и сохраняет ее.

Атрибуты:

* binding – это объект привязки, который предоставляет доступ к элементам пользовательского интерфейса в макете фрагмента fragment\_second.xml;
* dbHelper – экземпляр класса DBHelper, который используется для взаимодействия с базой данных, а именно для добавления данных о системе.

Методы класса:

1. OnCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) – cоздает и возвращает представление фрагмента на основе макета fragment\_second.xml.
2. OnViewCreated(): Устанавливает слушатель нажатия на кнопку «Сохранить» (button\_save). При нажатии на кнопку «Сохранить» проверяет, было ли введено название системы, если да, то получает название системы из текстового поля ввода и данные для подключения к MQTT брокеру из переданных аргументов фрагмента. Далее создает новый объект класса System и добавляет объект System в базу данных, используя экземпляр DBHelper. Если название системы не было введено, выводится сообщение об ошибке.
3. OnDestroyView(): Освобождает ресурсы и устанавливает привязку в значение null.

## 3.2.5 Класс DevicesActivity

Класс DevicesActivity представляет активность для управления устройствами в выбранной на главном экране системе.

Атрибуты класса:

* mRecyclerView – поле типа RecyclerView, представляющее компонент для отображения списка устройств;
* Adapter – объект класса DeviceAdapter, представляющий адаптер для связывания данных устройств с RecyclerView;
* Devices – поле типа ArrayList<Devices>, являющееся списком объектов Devices, представляющих устройства в системе;
* Binding – объект класса ActivityDevicesBinding, осуществляющий привязку для доступа к компонентам пользовательского интерфейса в макете activity\_devices.xml;
* Callback – объект класса SwipeToDeleteCallback, являющийся обратным вызовом для обработки свайпа для удаления устройства;
* System – объект System, представляющий текущую систему;
* dbHelper – экземпляр класса DBHelper для работы с базой данных;
* mqttAndroidClient – объект класса MqttAndroidClient, представляющий клиент MQTT для подключения к брокеру сообщений MQTT.

Методы класса:

1. OnCreate(Bundle savedInstanceState). Создает пользовательский интерфейс и инициализирует необходимые компоненты. Получает данные о системе. Устанавливает слушатели для кнопок обновления устройств и удаления системы. Устанавливает обработчик прокрутки для RecyclerView.
2. InitCardView(). Инициализирует RecyclerView и адаптер для списка устройств. Устанавливает обработчик свайпа для удаления устройств.
3. IsInternetConnection(). Проверяет наличие интернет-соединения на устройстве.
4. ConnectToMqtt(). Отвечает за подключение к брокеру MQTT и подписку на топики устройств.
5. SubscribeToTopic(String topic). Выполняет подписку на указанный топик MQTT.
6. PublishMessage(String topic, String message). Отправляет сообщение на указанный топик MQTT.
7. ParseMqttMessage(String jsonMessage). Обрабатывает полученное от MQTT сообщение в формате JSON.
8. SetListeners(). Устанавливает слушатели для кнопок на пользовательском интерфейсе.
9. SetDevicesList(). Инициализирует RecyclerView и устанавливает адаптер для списка устройств. Сначала он извлекает данные о устройствах из базы данных, используя объект DBHelper. Затем создает экземпляр адаптера DeviceAdapter и связывает его с RecyclerView для отображения данных о каждом устройстве.
10. UpdateDataForRecycler(). Обновляет данные в RecyclerView после внесения изменений. Он отвечает за обновление отображаемой информации в RecyclerView, включая скрытие или отображение текстового сообщения о наличии или отсутствии устройств в списке.
11. OnDestroy(). Отключение от MQTT брокера при завершении активности.

## 3.2.6 Класс AddDevice

Класс AddDevice представляет собой активность, предназначенную для добавления нового устройства в систему. Он содержит функциональность для выбора изображения устройства, заполнения информации о новом устройстве, сохранения его в базе данных и закрытия активности.

Атрибуты класса:

* RESULT\_LOAD\_IMG – константа, которая используется для определения кода запроса при выборе изображения из галереи;
* dbHelper – экземпляр класса DBHelper, который используется для взаимодействия с базой данных при добавлении нового устройства;
* Img – строковая переменная img, которая используется для хранения пути к выбранному изображению устройства;
* binding – объект класса ActivityAddDeviceBinding, который представляет собой привязку к макету activity\_add\_device.xml, и используется для доступа к элементам пользовательского интерфейса этой активности, таким как кнопки, текстовые поля и изображения.

Методы класса:

* onCreate(Bundle savedInstanceState) – настраивает различные элементы пользовательского интерфейса, связываете их с данными и устанавливает обработчики событий для реагирования на действия пользователя;
* onActivityResult(int reqCode, int resultCode, Intent data) – обрабатывает результат выбора изображения из галереи, извлекает изображение из полученного пути и отображает его в ImageView;
* saveImageToInternalStorage(Bitmap bitmapImage, Uri imageUri) – сохраняет выбранное изображение во внутреннем хранилище устройства, генерируя уникальное имя файла и сохраняя изображение с этим именем во внутреннем каталоге приложения;
* getExtensionFromUri(Uri uri) – извлекает расширение файла из пути выбранного изображения, определяя тип MIME выбранного изображения и возвращая его расширение.

## 3.2.7 Класс DBHelper

Класс DBHelper является вспомогательным классом для работы с базой данных SQLite в приложении.

Атрибуты класса:

* DATABASE\_NAME – константа, определяющая имя базы данных, которая по умолчанию принимает значение «iotDb»;
* DATABASE\_VERSION – константа, определяющая версию базы данных, которая по умолчанию принимает значение «1».

Методы класса:

* onCreate(SQLiteDatabase db) – метод, вызываемый при создании базы данных, создающий таблицы «Системы» и «Устройства», если они еще не существуют;
* onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) – метод, вызываемый при обновлении базы данных, использующийся для миграции данных при изменении структуры базы данных;
* getSystemId(System system) – получает идентификатор системы;
* addSystem(System system) – добавляет новую систему в базу данных;
* getAllSystems() – получает список всех систем из базы данных;
* deleteAllSystems() – удаляет все системы из базы данных;
* addDevice(Devices device, System system) – добавляет новое устройство в базу данных, связывая его с определенной системой;
* getAllDevices(System system) – получает список всех устройств, принадлежащих определенной системе.

## 3.2.8 Класс Devices

Класс Devices представляет собой модель устройства в приложении.

Атрибуты класса:

* friendlyName – строка, представляющая дружественное имя устройства (по умолчанию пустая строка);
* deviceId – строка, содержащая идентификатор устройства;
* type – строка, обозначающая тип устройства;
* imgPath – строка, представляющая путь к изображению устройства (по умолчанию пустая строка);
* lastAcceptedData – строка, содержащая последние принятые данные от устройства (по умолчанию пустая строка);
* MqttPrefix – строка, представляющая префикс MQTT для устройства (по умолчанию пустая строка);
* diodeChannel – строка, обозначающая канал диода устройства (по умолчанию пустая строка).

Методы класса включают конструктор, getter и setter атрибутов, служащие для установки и получения значений атрибутов.

## 3.2.9 Класс Systems

Класс Systems представляет собой модель устройства в приложении.

Атрибуты класса:

* systemName – строка, содержащая название системы;
* mqtt\_url – строка, представляющая URL-адрес MQTT сервера для системы;
* mqtt\_login – строка, содержащая логин для подключения к MQTT серверу (по умолчанию «null»);
* mqtt\_password – строка, содержащая пароль для подключения к MQTT серверу (по умолчанию «null»).

Методы класса включают конструктор, getter и setter атрибутов, служащие для установки и получения значений атрибутов.

## 3.2.10 Класс DeleteConfirmationDialog

Класс DeleteConfirmationDialog представляет диалоговое окно подтверждения удаления.

Содержит метод show(Context context, String message, final DeleteDialogListener deleteListener, final CancelDialogListener cancelListener): статический метод для отображения диалогового окна подтверждения удаления. Принимает контекст приложения context, сообщение message, объект слушателя для подтверждения удаления deleteListener и объект слушателя для отмены действия cancelListener. В диалоговом окне пользователь может выбрать между удалением и отменой действия. Если пользователь подтверждает удаление, вызывается метод onDeleteConfirmed() интерфейса DeleteDialogListener. Если пользователь отменяет действие, вызывается метод onCancel() интерфейса CancelDialogListener.

## 3.2.11 Класс DeviceAdapter

Класс DeviceAdapter представляет адаптер для встроенного класса RecyclerView, который управляет отображением списка устройств.

Атрибуты класса:

* mDataset – список устройств для отображения;
* Context – контекст приложения;
* mqttAndroidClient – клиент MQTT для взаимодействия с брокером сообщений.

Методы класса:

* ViewHolder(View v) – конструктор класса ViewHolder, инициализирует представления элемента списка;
* SetDetails(Devices device) – метод для установки данных устройства в представления элемента списка;
* getView() – возвращает представление элемента списка;
* DeviceAdapter(Context context, ArrayList<Devices> myDataset, MqttAndroidClient mqttAndroidClient) – конструктор адаптера, принимает контекст приложения, список устройств и клиент MQTT;
* onCreateViewHolder(ViewGroup parent, int viewType) – создает новый объект ViewHolder при необходимости;
* onBindViewHolder(ViewHolder holder, int position) – связывает данные с представлениями элемента списка;

## 3.2.11 Класс SwipeToDeleteCallback

SwipeToDeleteCallback – это класс обратного вызова, который обрабатывает жесты свайпа для удаления элементов из списка RecyclerView.

Атрибуты класса:

* icon – значок, который будет отображаться при свайпе;
* backgroundLeft – градиентный фон для свайпа влево;
* swipeListener – слушатель для обработки события свайпа;
* context – контекст приложения.

Методы класса:

* SwipeToDeleteCallback(Context context, OnSwipeListener listener) – конструктор класса, принимает контекст и слушатель событий свайпа;
* onMove(RecyclerView recyclerView, ViewHolder viewHolder, RecyclerView.ViewHolder target) – метод, который вызывается при перемещении элемента списка;
* onSwiped(RecyclerView.ViewHolder viewHolder, int direction) – метод, который вызывается при свайпе элемента списка;
* onChildDraw(Canvas c, RecyclerView recyclerView, ViewHolder viewHolder, float dX, float dY, int actionState, boolean isCurrentlyActive) – метод, который вызывается при отрисовке элемента списка во время свайпа и устанавливает значок и фон для свайпа влево;
* getSwipeThreshold(RecyclerView.ViewHolder viewHolder) – метод, который возвращает порог для срабатывания свайпа.

## 3.2.12 Класс IPAddressFilter

IPAddressFilter – это класс, реализующий интерфейс InputFilter, который используется для фильтрации ввода IP-адреса. Метод filter(CharSequence source, int start, int end, Spanned dest, int dstart, int dend) – фильтрует ввод символов для IP-адреса. Метод проверяет, является ли ввод допустимым IP-адресом. Если нет, то возвращает пустую строку, что приводит к игнорированию ввода. В противном случае, возвращает null, что позволяет вводу продолжиться.

## 3.2.13 Класс PortFilter

Класс PortFilter представляет собой инструмент для фильтрации ввода порта в приложении. Он реализует интерфейс InputFilter, что позволяет его использовать для контроля пользовательского ввода.

Метод filter(CharSequence source, int start, int end, android.text.Spanned dest, int dstart, int dend) в классе PortFilter играет ключевую роль в определении допустимости ввода. Он анализирует ввод пользователя, проверяя, является ли он корректным портом, то есть находится ли в диапазоне от 0 до 65535. Если ввод соответствует этому условию, он допускается и возвращается null, что позволяет вводу продолжиться. В противном случае, если ввод не соответствует диапазону портов или является некорректным числом, возвращается пустая строка, и ввод игнорируется.

## 3.2.14 Интерфейс MqttConnectionLostListener

Интерфейс MqttConnectionLostListener представляет собой слушателя для обработки потери соединения с брокером MQTT в приложении. Этот интерфейс определяет метод onConnectionLost, который вызывается в случае потери соединения.

Параметр cause, передаваемый в метод onConnectionLost, представляет собой объект типа Throwable, который содержит информацию о причине потери соединения. Это может быть, например, исключение, описывающее ошибку соединения.

Реализуя интерфейс MqttConnectionLostListener в приложении и переопределяя метод onConnectionLost, можно определить необходимые действия при потере соединения с брокером MQTT.

## 3.2.15 Интерфейс MqttConnectListener

Интерфейс MqttConnectListener представляет собой слушателя для обработки результатов попыток подключения к брокеру MQTT в приложении. Он определяет два метода: onSuccess и onFailure.

Метод onSuccess вызывается в случае успешного установления соединения с брокером MQTT. Он позволяет приложению выполнить необходимые действия при успешном подключении, например, инициализировать подписку на темы или запустить отправку сообщений.

Метод onFailure вызывается, если произошла ошибка при попытке подключения к брокеру MQTT. В качестве параметра exception передается объект типа Throwable, содержащий информацию о причине неудачного подключения. Это может быть, например, исключение, описывающее ошибку соединения или аутентификации.

Используя интерфейс MqttConnectListener, можно реализовать логику обработки результатов попыток подключения к брокеру MQTT в приложении. Например, при успешном подключении можно выполнить дополнительные действия, связанные с работой с MQTT, а при неудачном подключении можно вывести сообщение об ошибке или попытаться повторить подключение.

## 3.2.16 Интерфейс MqttDeliveryCompleteListener

Интерфейс MqttDeliveryCompleteListener определяет метод onDeliveryComplete, который вызывается при завершении доставки сообщения в брокер MQTT. В качестве параметра передается объект IMqttDeliveryToken, который содержит информацию о доставленном сообщении.

Реализуя этот интерфейс, определяется логика, которая должна быть выполнена после успешной доставки сообщения. Например, можно обновить интерфейс пользователя, отобразив статус успешной доставки, или выполнить другие действия, связанные с успешной отправкой сообщения.

## 3.2.17 Интерфейс MqttMessageArrivedListener

Интерфейс MqttMessageArrivedListener определяет метод onMessageArrived, который вызывается при получении сообщения из брокера MQTT. Этот метод принимает два параметра:

* topic – строка, представляющая тему (topic) сообщения, которое было получено;
* message – объект типа MqttMessage, содержащий полученное сообщение.

Реализуя этот интерфейс, можно определить логику, которая должна быть выполнена при получении нового сообщения из брокера MQTT. Например, можно обработать полученное сообщение, отобразить его на пользовательском интерфейсе или выполнить другие действия в зависимости от содержания сообщения и его темы.

## 3.2.18 Интерфейс OnSwipeListener

Интерфейс OnSwipeListener определяет метод onSwipe, который вызывается при выполнении свайпа элемента в RecyclerView. Этот метод принимает параметр position – позиция элемента в адаптере RecyclerView, который был свайпнут.

Реализуя этот интерфейс, можно определить логику, которая должна быть выполнена при свайпе элемента списка. Например, можно удалить элемент из списка или выполнить другие действия в зависимости от позиции свайпнутого элемента.

## 3.2.19 Разметка AndroidManifest

В файле AndroidManifest.xml определены необходимые разрешения, которые приложение должно запросить перед использованием определенных функций. Вот какие разрешения указаны и их цели:

* INTERNET – это разрешение указывает на необходимость доступа к сети Интернет для приложения, а также позволяет приложению выполнять сетевые операции, такие как загрузка данных из Интернета;
* WAKE\_LOCK – это разрешение указывает на необходимость приложению управлять состоянием блокировки экрана устройства, то есть, если приложение должно оставаться активным или выполнять действия в фоновом режиме, даже когда экран устройства выключен;
* ACCESS\_NETWORK\_STATE – это разрешение позволяет приложению запрашивать информацию о текущем состоянии сетевого подключения;
* READ\_PHONE\_STATE – это разрешение дает приложению доступ к информации о состоянии телефона, такой как состояние сети, идентификатор устройства и другие телефонные параметры;
* READ\_MEDIA\_AUDIO – это разрешение позволяет приложению читать медиа- и аудиофайлы на устройстве, например, позволяет загрузить изображение конечного устройства из галереи.

Также в элементе <application> файла AndroidManifest.xml устанавливается иконка приложения, название, а также разрешения на резервное копирование и доступ к сети Интернет. Также определяются правила для резервного копирования данных и настройки темы приложения.

## 3.2.20 Разметка Nav\_graph

Этот XML-файл представляет навигационный граф в приложении. Он определяет различные фрагменты и действия между ними, что облегчает навигацию пользователя по приложению.

Каждый фрагмент (FirstFragment и SecondFragment) имеет свой собственный идентификатор, метку и макет для отображения содержимого. Фрагменты представляют отдельные экраны приложения, где пользователь может взаимодействовать с различными элементами.

Каждое действие определяет переход между фрагментами. Например, действие action\_FirstFragment\_to\_SecondFragment указывает на переход от FirstFragment к SecondFragment, а действие action\_SecondFragment\_to\_FirstFragment – на обратный переход. Также есть действия, которые напрямую переносят пользователя на главный экран приложения (MainActivity).

Общий навигационный граф организует взаимодействие между экранами и определяет, как пользователи перемещаются по приложению, что помогает создать плавный и интуитивно понятный пользовательский опыт.

## 3.2.21 Разметка Activity\_add\_device

Этот XML-файл содержит макет экрана (AddDevice), в котором происходит добавление устройства. Он использует ConstraintLayout для организации размещения элементов на экране.

На данном макете есть:

* Toolbar в верхней части экрана с настройками, такими как цвет фона и текст заголовка;
* LinearLayout с идентификатором line, который содержит группу элементов, включая изображение, кнопку, и поля для ввода текста;
* Кнопка buttonSaveDevice, расположенная внизу экрана, для сохранения устройства.

Кроме того, в разметке определены изображение, кнопка для выбора изображения, и несколько текстовых полей для ввода данных, таких как дружественное имя устройства, префикс MQTT и т. д.

## 3.2.21 Разметка Activity\_devices

XML-файл определяет макет для экрана DevicesActivity, отображающий список доступных в системе устройств.

Основные компоненты макета:

* Toolbar с идентификатором toolbar2, который содержит заголовок и кнопки для обновления списка устройств и удаления устройств;
* ImageButton с идентификатором buttonRefreshDevices для обновления списка устройств;
* ImageButton с идентификатором buttonSystemDelete для удаления систем;
* TextView с идентификатором textViewNoDevices, который отображает текст «No Devices», если список устройств пуст или не был загружен;
* ProgressBar с идентификатором progressBar, который отображается во время загрузки данных, а именно во время подключения к MQTT брокеру, связанным с выбранной системой;
* RecyclerView с идентификатором recycler для отображения списка устройств;
* ConstraintLayout с идентификатором conLayout, который содержит RecyclerView и организует его размещение на экране;
* FloatingActionButton с идентификатором addDevice, который предоставляет пользователю возможность добавления нового устройства.

## 3.2.22 Разметка Activity\_main

В данном файле описывается макет для главного экрана (MainActivity), в котором отображаются все доступные пользователю системы.

В макете присутствуют следующие компоненты:

* Toolbar (toolbar), который содержит заголовок приложения и является контейнером для других элементов;
* TextView (textViewNoSystems), отображающий текст «No Systems», если список систем пуст;
* ListView (ListOfSystems), предназначенный для отображения списка систем;
* FloatingActionButton (addSystem), который позволяет пользователю добавлять новые системы.

## 3.2.22 Разметка Activity\_second

Данный XML-файл определяет макет для второй активности (SecondActivity), в которой происходит добавление системы.

В макете используется CoordinatorLayout, который обеспечивает координацию различных компонентов пользовательского интерфейса. Внутри CoordinatorLayout содержится AppBarLayout, который представляет собой контейнер для MaterialToolbar. MaterialToolbar является основным инструментом для отображения заголовка и действий на верхней панели приложения.

Также в макете присутствует включение (include) с идентификатором include, которое ссылается на содержимое (content\_second.xml), которое будет отображаться в этой активности.

Этот макет обеспечивает простой и элегантный интерфейс для второй активности приложения, включая верхнюю панель с заголовком и содержимым, определенным в файле content\_second.xml.

## 3.2.23 Разметка Content\_second

Этот файл определяет макет для второй активности (SecondActivity) приложения, который содержит фрагмент, связанный с навигацией.

В макете используется ConstraintLayout, который обеспечивает гибкую разметку элементов пользовательского интерфейса. Внутри ConstraintLayout находится фрагмент, идентифицированный атрибутом android:id=«@+id/nav\_host\_fragment\_content\_second». Этот фрагмент представляет собой контейнер для отображения навигационных элементов, которые управляются с помощью NavHostFragment.

Атрибут app:navGraph=«@navigation/nav\_graph» связывает этот NavHostFragment с файлом навигационного графа (nav\_graph.xml), определенным в ресурсах проекта. Это графическое представление маршрутов и навигационных действий в приложении.

App:defaultNavHost=«true» устанавливает этот NavHostFragment как основной контейнер для обработки нажатий кнопок «назад» на устройстве.

Этот макет обеспечивает интеграцию навигационных возможностей во второй активности приложения, что позволяет пользователям легко перемещаться между различными экранами и функциональными областями приложения.

## 3.2.24 Разметка Fragment\_first

Данный файл определяет макет для первого фрагмента (FirstFragment) приложения для добавления информации об MQTT сервере при создании новой системы. Фрагмент содержит NestedScrollView, обеспечивающий прокрутку содержимого в случае необходимости. NestedScrollView содержит ConstraintLayout, который используется для организации размещения элементов пользовательского интерфейса.

Внутри ConstraintLayout содержатся следующие элементы:

* ImageView для отображения изображения;
* TextView с текстом о добавлении подключения MQTT брокера;
* LinearLayout с двумя TextInputLayout для ввода IP-адреса и порта MQTT;
* два TextInputLayout для ввода имени пользователя и пароля;
* два Button для выполнения действий «Отмена» и «Далее».

Этот макет предоставляет пользователю интерфейс для добавления MQTT подключения с вводом IP-адреса, порта, имени пользователя и пароля. Пользователь может использовать кнопки «Отмена» и «Далее» для выполнения соответствующих действий.

## 3.2.25 Разметка Fragment\_second

Этот файл определяет макет для фрагмента (SecondFragment) приложения, отображающий последние поля для ввода данных о системе. Внутри NestedScrollView находится ConstraintLayout, который используется для организации размещения элементов пользовательского интерфейса.

В контейнере ConstraintLayout используется LinearLayout с идентификатором linearlay, который содержит следующие элементы:

* ImageView для отображения изображения;
* TextView с названием системы;
* TextInputLayout с вложенным TextInputEditText для ввода названия системы;
* Button с идентификатором button\_save, который используется для сохранения введенной информации о системе.

Этот макет предоставляет пользователю интерфейс для ввода названия системы и сохранения этой информации.

## 3.2.25 Разметка Fragment\_second

Этот файл определяет макет для фрагмента (SecondFragment) приложения, отображающий последние поля для ввода данных о системе. Внутри NestedScrollView находится ConstraintLayout, который используется для организации размещения элементов пользовательского интерфейса.

В контейнере ConstraintLayout используется LinearLayout с идентификатором linearlay, который содержит следующие элементы:

* ImageView для отображения изображения;
* TextView с названием системы;
* TextInputLayout с вложенным TextInputEditText для ввода названия системы;
* Button с идентификатором button\_save, который используется для сохранения введенной информации о системе.

Этот макет предоставляет пользователю интерфейс для ввода названия системы и сохранения этой информации.

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Данное приложение разрабатывалось в IDE Android Studio с использованием технологий Java, Sqlite3, Mqtt. В приложении реализованы следующие функции:

* добавление системы;
* вывод списка существующих систем;
* вывод списка привязанных к системе устройств;
* добавление в систему нового устройства;
* подключение к MQTT брокеру;
* подписка и отписка на темы (topics) в брокере;
* получение данных от MQTT брокера;
* обработка полученных данных;
* отправка данных в MQTT брокер;
* взаимодействие с базой данных.

Рассмотрим среду разработки и результаты работы мобильного приложения.

## 4.1 Android Studio

В рамках проекта используется интегрированная среда разработки (IDE) Android Studio, основанная на языке программирования Java. Android Studio предоставляет разработчикам широкий спектр инструментов и функций, специально адаптированных для создания мобильных приложений под платформу Android. Эта IDE обеспечивает удобную среду для написания кода, автоматическую проверку синтаксиса, управление проектом, визуальное проектирование пользовательского интерфейса, отладку приложений, а также интеграцию с множеством сторонних библиотек и инструментов.

Использование Android Studio и языка Java значительно упрощает процесс разработки мобильных приложений для Android. Эти инструменты предлагают обширный набор функций, которые помогают создавать высококачественные и эффективные приложения, такие как быструю разработку интерфейсов пользователя, интеграцию с облачными сервисами, обеспечение безопасности данных и оптимизацию производительности. Также использование Android Studio и Java позволяет легко адаптировать приложение под различные устройства и версии операционной системы Android, обеспечивая широкий охват аудитории и повышая конкурентоспособность продукта на рынке мобильных приложений.

Проект в IDE Android Studio разделен на следующие элементы (см. рисунок 4.1):

1. Дерево файлов проекта.
2. Редактор кода.
3. Эмулятор устройств.



Рисунок 4.1 – Элементы Android Studio

## 4.2 Добавление системы

За добавление системы отвечает классы System и SecondActivity, который состоит из FirstFragment и SecondFragment. Добавление происходит в два этапа:

* ввод данных, связанных с MQTT брокером;
* ввод названия системы, которое будет отображаться на главном экране.

В начальном этапе пользователь вводит адрес и порт MQTT брокера, а также логин и пароль, если они необходимы. После этого введенные данные проходят проверку на правильность, и только затем передаются для последующего использования на следующем этапе. Данный метод выполняет указанные действия.

public Bundle validateData(){

Bundle bundle = new Bundle();

if(binding.editTextInputIP.getText().toString().isEmpty() || Objects.requireNonNull(binding.editTextInputPort.getText()).toString().isEmpty())

{Toast.makeText(requireContext(), "Required IP Address and port", Toast.LENGTH\_LONG).show();

return null;}

else {

String ipAddress = binding.editTextInputIP.getText().toString();

String port = binding.editTextInputPort.getText().toString();

if (isValidPort(port)) {

Boolean loginEmpty = binding.editTextInputLogin.getText() == null;

Boolean passwordEmpty = binding.editTextPassword.getText() == null;

if (loginEmpty && passwordEmpty) {

bundle.putString("address", "tcp://" + ipAddress + ":" + port);

return bundle;

} else if (!loginEmpty && !passwordEmpty) {

bundle.putString("address", "tcp://" + ipAddress + ":" + port);

bundle.putString("login", binding.editTextInputLogin.getText().toString());

bundle.putString("password", binding.editTextPassword.getText().toString());

return bundle;

} else {

Toast.makeText(requireContext(), "Required full credentials or none", Toast.LENGTH\_LONG).show();

return null;}}

else{

Toast.makeText(requireContext(), "Incorrect port", Toast.LENGTH\_LONG).show();

return null;

}}}

Кроме того, на формы ввода прикрепляются фильтры, например для ввода IP адреса и номера порта.

@Override

public CharSequence filter(CharSequence source, int start, int end, Spanned dest, int dstart, int dend) {

if (end > start) {

String destTxt = dest.toString();

String resultingTxt = destTxt.substring(0, dstart)

+ source.subSequence(start, end)

+ destTxt.substring(dend);

if (!resultingTxt.matches("^\\d{1,3}(\\.(\\d{1,3}(\\.(\\d{1,3}(\\.(\\d{1,3})?)?)?)?)?)?")) {

return "";

} else {

String[] splits = resultingTxt.split("\\.");

for (int i = 0; i < splits.length; i++) {

if (Integer.valueOf(splits[i]) > 255) {

return "";}}}}

return null;}

@Override

public CharSequence filter(CharSequence source, int start, int end,

android.text.Spanned dest, int dstart, int dend) {

if (end > start) {

String destTxt = dest.toString();

String resultingTxt = destTxt.substring(0, dstart)

source.subSequence(start, end)

destTxt.substring(dend);

try {

int port = Integer.parseInt(resultingTxt);

if (port < 0 || port > 65535) {return "";}

} catch (NumberFormatException e) {

return "";}}

return null;}

После успешной проверки всех полей на корректность данные передаются в SecondFragment, где пользователю предоставляется возможность завершить добавление системы. После ввода названия системы и нажатия кнопки «Сохранить» система добавляется в базу данных мобильного устройства.

binding.buttonSave.setOnClickListener(v -> {

Boolean nameEmpty = binding.textEditSystemName.getText().toString().isEmpty();

if(!nameEmpty) {

String sysName = binding.textEditSystemName.getText().toString();

Bundle args = getArguments();

if (args != null)

{

String address = "";

String login = "";

String password = "";

address = args.getString("address");

if (args.containsKey("login")) {

login = args.getString("login");}

if (args.containsKey("password")) {

password = args.getString("password");}

dbHelper = new DBHelper(requireContext());

System mySys = new System(sysName, address);

if(!login.isEmpty() && !password.isEmpty()){

mySys.setMqtt\_login(login);

mySys.setMqtt\_password(password);}

Log.i("MQTT add system", address + " " + login + " " + password);

dbHelper.addSystem(mySys);

Toast.makeText(requireContext(), "Successfully added system!",

Toast.LENGTH\_LONG).show(); NavHostFragment.findNavController(SecondFragment.this).navigate(R.id.action\_SecondFragment\_to\_mainActivity)}

}else Toast.makeText(requireContext(), "System name field should be filled", Toast.LENGTH\_LONG).show();});

После всех манипуляций с данными происходит переход на основной экран со списком всех систем.

## 4.3 Вывод списка систем

Запуская приложение пользователя встречает список систем, которые был добавлены ранее. Если список пуст, то в середине экрана будет выведена надпись «Нет систем». Для реализации функции вывода списка систем используются классы MainActivity и SystemAdapter. В классе MainActivity создается объект адаптера и иницаилизируется списком систем, найденых функцией getAllSystems класса DBHelper. В объекте самого виджета списка происходит установка данного адаптера функцией setAdapter.

public void setListItems(){

mRecyclerView = findViewById(R.id.recyclerSystem);

mRecyclerView.setLayoutManager(new LinearLayoutManager((this)));

systems = dbHelper.getAllSystems();

adapter = new SystemAdapter(this, new ArrayList<>(systems));

mRecyclerView.setAdapter(adapter);

updateDataForRecycler();

В классе SystemAdapter происходит установка значений объектов систем в поля на форме виджета, например, установка названия системы и названия всех устройств, связанных с данной системой.

public void SetDetails(System system){

textNameSystem.setText(system.getSystemName());

ArrayList<Devices> devices = dbHelper.getAllDevices(system);

StringBuilder devStr = new StringBuilder();

devices.forEach((x)->devStr.append(x.getFriendlyName()).append(" "));

if(!devStr.toString().isEmpty())

textDevicesSystem.setText(devStr);}

К тому же, в данном классе происходит обработка нажатия на элемент списка с системами, где происходит вызов активности с выводом списка устройств.

holder.itemView.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

int position = holder.getAdapterPosition();

if (position != RecyclerView.NO\_POSITION) {

System clickedSystem = mDataset.get(position);

Intent intent = getIntent(clickedSystem);

context.startActivity(intent);}}});

Кроме всего прочего, в классе MainActivity происходит прикрепление обработчика свайпов элемента списка. Данный функционал позволяет добавить изменение и удаление системы без добавления новых кнопок.

ItemTouchHelper itemTouchHelper = getItemTouchHelper();

itemTouchHelper.attachToRecyclerView(mRecyclerView);

@NonNull

private ItemTouchHelper getItemTouchHelper() {

SwipeToDeleteCallback callback = new SwipeToDeleteCallback(this, this, this);

ItemTouchHelper itemTouchHelper = new ItemTouchHelper(callback);

return itemTouchHelper;}

## ****4.4 Вывод списка устройств****

**После того, как пользователь выбирает систему из списка на главном экране, данные о выбранной системе передаются в активность DeviceActivity. В этой активности происходит проверка соединения с Интернетом и попытка подключения к указанному MQTT брокеру, указанному в выбранной системе. Если проверка неудачна, выводится соответствующее сообщение.**

**После этого, при первом запуске приложения, пользователю отображается пустой список устройств с надписью «Нет устройств», если приложение смогло успешно подключиться к MQTT брокеру. Только после добавления пользователем устройства список будет инициализирован. Ниже приведена функция, в которой происходит получение списка всех устройств из базы данных и создание адаптера, который инициализирует виджет списка.**

**private void setDevicesList(){**

**mRecyclerView = findViewById(R.id.recycler);**

**mRecyclerView.setLayoutManager(new LinearLayoutManager((this)));**

**devices = dbHelper.getAllDevices(system);**

**adapter = new DeviceAdapter(this, devices, mqttAndroidClient);**

**mRecyclerView.setAdapter(adapter);**

**updateDataForRecycler();}**

**После создания объекта адаптера, он инициализирует виджет списка, передаваемый ему в качестве параметра, массивом устройств. В классе DeviceAdapter происходит заполнение текстовых полей, таких как название, идентификатор, тип устройства, а также изображение и последние полученные данные от устройства. Если при создании устройства был указан канал для взаимодействия, также инициализируется кнопка переключения. Вот пример инициализации некоторых полей:**

**public void SetDetails(Devices device){**

**setFriendlyName(device);**

**setDiodeChannelAndLastData(device);**

**setImage(device);}**

**public void setFriendlyName(Devices device){**

**if(device.getFriendlyName() != null)**

**textNameCard.setText(device.getFriendlyName() + "\n(" + device.getDeviceId() + ")");**

**else textDataCard.setText(device.getType());}**

**public void setImage(Devices device){**

**if (device.getImgPath() != null){**

**if (!device.getImgPath().isEmpty()) {**

**try {**

**File file = new File(device.getImgPath());**

**if (file.exists()) {**

**Bitmap bitmap = BitmapFactory.decodeFile(file.getAbsolutePath());**

**imageView.setImageBitmap(bitmap);}**

**} catch (Exception e) {**

**e.printStackTrace();}}}}**

## 4.5 Добавление нового устройства

При добавлении нового устройства, аналогично системам, пользователю необходимо заполнить определенные поля данными: имя устройства для отображения в списке, префикс топика в MQTT брокере, уникальный идентификатор устройства, тип устройства и канал для взаимодействия. Кроме того, пользователь может добавить изображение устройства, выбрав его из галереи. Формат изображений может быть как PNG, так и JPEG. После заполнения данных и нажатия кнопки «Сохранить», вызывается функция-слушатель, которая добавляет данные об устройстве в базу данных. Ниже представлен алгоритм добавления устройства:

buttonSaveDevice.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

boolean isFriendlyName = Objects.requireNonNull(binding.textInputFriendlyName.getText()).toString().isEmpty();

boolean isDeviceIeeeid = Objects.requireNonNull(binding.textInputDeviceIeeeid.getText()).toString().isEmpty();

boolean isDeviceType = Objects.requireNonNull(binding.textInputDeviceType.getText()).toString().isEmpty();

boolean isMqttPrefix = Objects.requireNonNull(binding.textInputMqttPrefix.getText()).toString().isEmpty();

boolean isDiodeChannel = Objects.requireNonNull(binding.textInputDeviceSwitchChannel.getText()).toString().isEmpty();

if(!isFriendlyName && !isDeviceIeeeid && !isDeviceType && !isMqttPrefix){

dbHelper = new DBHelper(getApplicationContext());

Intent intent = getIntent();

String systemName = intent.getStringExtra("systemName");

String mqttUrl = intent.getStringExtra("mqttUrl");

System system = new System(systemName, mqttUrl);

String friendlyName = binding.textInputFriendlyName.getText().toString();

String deviceIeeeid = binding.textInputDeviceIeeeid.getText().toString();

String deviceType = binding.textInputDeviceType.getText().toString();

String deviceMqttPrefix = binding.textInputMqttPrefix.getText().toString();

Devices device = new Devices(deviceIeeeid, deviceType, img);

device.setFriendlyName(friendlyName);

device.setMqttPrefix(deviceMqttPrefix);

if(!isDiodeChannel){

String diodeChannel = binding.textInputDeviceSwitchChannel.getText().toString();

device.setDiodeChannel(diodeChannel);}

dbHelper.addDevice(device, system);

dbHelper.close();}

else Toast.makeText(getApplicationContext(), "All fields should be filled", Toast.LENGTH\_LONG).show();}});

В процессе происходит проверка заполнения полей ввода данных, после чего информация собирается из этих полей и сохраняется в соответствующие параметры. Затем создается объект устройства device и объект базы данных dbHelper, которые передаются в функцию сохранения устройства.

В следующей части кода реализовано открытие галереи, отображение выбранного изображения в приложении, и вызов функции saveImageToInternalStorage, в которой происходит сохранение выбранного изображения в директорию изображений:

buttonUpload.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent photoPickerIntent = new Intent(Intent.ACTION\_PICK);

photoPickerIntent.setType("image/\*");

startActivityForResult(photoPickerIntent, RESULT\_LOAD\_IMG);}});

@Override

protected void onActivityResult(int reqCode, int resultCode, Intent data) {

super.onActivityResult(reqCode, resultCode, data);

if (resultCode == RESULT\_OK) {

try {

final Uri imageUri = data.getData();

final InputStream imageStream = getContentResolver().openInputStream(imageUri);

final Bitmap selectedImage = BitmapFactory.decodeStream(imageStream);

img = saveImageToInternalStorage(selectedImage, imageUri);

Log.i("sql", img);

ImageView imgView = findViewById(R.id.imageViewDeviceAdd);

imgView.setImageBitmap(selectedImage);

} catch (FileNotFoundException e) {

e.printStackTrace();

Toast.makeText(this, "Something went wrong", Toast.LENGTH\_LONG).show();}

}else {

Toast.makeText(this, "You haven't picked Image",Toast.LENGTH\_LONG).show();}}

Сама функция saveImageToInternalStorage представляет собой следующий код:

private String saveImageToInternalStorage(Bitmap bitmapImage, Uri imageUri){

ContextWrapper cw = new ContextWrapper(getApplicationContext());

File directory = cw.getDir("imageDir", Context.MODE\_PRIVATE);

String uniqueFileName = "profile\_" + UUID.randomUUID().toString();

String extension = getExtensionFromUri(imageUri);

if (extension != null && !extension.isEmpty()) {

uniqueFileName += "." + extension;

} else {

uniqueFileName += ".jpg";}

File mypath = new File(directory, uniqueFileName);

FileOutputStream fos = null;

try {

fos = new FileOutputStream(mypath);

if (extension != null && extension.equalsIgnoreCase("png")) {

bitmapImage.compress(Bitmap.CompressFormat.PNG, 100, fos);

} else {

bitmapImage.compress(Bitmap.CompressFormat.JPEG, 100, fos);}

} catch (Exception e) {

e.printStackTrace();

} finally

{

try {

if (fos != null) {

fos.close();}

} catch (IOException e) {

e.printStackTrace();}}

return directory.getAbsolutePath() + "/" + uniqueFileName;}

В данной функции происходит сохранение изображения под уникальным индентификатором типа UUID. Также перед сохранением производится сжатие изображения с определенным форматом для чего находится расширение изображения в функции getExtensionFromUri.

private String getExtensionFromUri(Uri uri) {

ContentResolver contentResolver = getContentResolver();

MimeTypeMap mimeTypeMap = MimeTypeMap.getSingleton();

String extension;

if (contentResolver == null || uri == null) {

return null;}

String mimeType = contentResolver.getType(uri);

if (mimeType == null) {

String uriString = uri.toString();

int extensionIndex = uriString.lastIndexOf('.');

if (extensionIndex != -1 && extensionIndex < uriString.length() - 1) {

extension = uriString.substring(extensionIndex + 1);

return extension.toLowerCase();}

return null;}

extension = mimeTypeMap.getExtensionFromMimeType(mimeType);

return extension;}

## 4.6 Подключение к MQTT брокеру

При подключении к брокеру необходимо выполнить следующие действия:

* получить данные для подключения;
* отправить серверу запрос о подключении;
* получить ответ от сервера.

Первое действие осуществляется путем принятия данных о системе из предыдущей активности. Следующий код выполняет создание объекта системы и заполняет его:

Intent intent = getIntent();

system = new System(intent.getStringExtra("systemName"), intent.getStringExtra("mqttUrl"));

if (intent.hasExtra("mqtt\_login") && intent.hasExtra("mqtt\_password")) {

system.setMqtt\_login(intent.getStringExtra("mqtt\_login"));

system.setMqtt\_password(intent.getStringExtra("mqtt\_password"))}

После этого создается объект MQTT клиента и заполняется параметрами из объекта system. В случае, если логин и пароль, представленные полями mqtt\_login и mqtt\_password соответственно, не равняются null, то указываются как параметры подключения.

String clientId = MqttClient.generateClientId();

mqttAndroidClient = new MqttAndroidClient(getApplicationContext(), system.getMqtt\_url(), clientId);

MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();

mqttConnectOptions.setAutomaticReconnect(true);

mqttConnectOptions.setCleanSession(false);

if(system.getMqtt\_login() != null && system.getMqtt\_password() != null){

mqttConnectOptions.setUserName(system.getMqtt\_login());

mqttConnectOptions.setPassword(system.getMqtt\_password().toCharArray());}

try {

mqttAndroidClient.connect(mqttConnectOptions, null, new IMqttActionListener())

} catch (Exception ex) {

ex.printStackTrace();

Log.i("MQTT", ex.toString());}

## 4.7 Подписка и отписка на темы (topic)

Функции, обрабатывающие данные действия, активируются только после успешного подключения к брокеру MQTT. Сначала происходит подписка на темы во время загрузки списка устройств после установления связи с брокером для получения обновлений данных от устройств и их отображения. Затем, после добавления нового устройства, происходит дополнительная подписка на соответствующие темы.

Как упомянуто ранее, подписка происходит после установления соединения. Для этого определен слушатель, который при успешном подключении обрабатывает все устройства, формируя строку подписки на основе значений их префиксов и уникальных идентификаторов в методе subscribeToTopic. Ниже приведен код слушателя.

@Override

public void onSuccess(IMqttToken asyncActionToken) {

loadingSpinner.setVisibility(View.INVISIBLE);

constraintLayout.setVisibility(View.VISIBLE);

addDevice.setVisibility(View.VISIBLE);

Toast.makeText(getApplicationContext(), "Mqtt connection", Toast.LENGTH\_LONG).show();

initCardView();

for (Devices device : devices) {

String prefixMqtt = device.getMqttPrefix();

String deviceId = device.getDeviceId();

subscribeToTopic(prefixMqtt + "/" + deviceId);}}

Метод subscribeToTopic представляет собой вызов метода объекта mqttAndroidClient, созданный при подключении к брокеру, обработку ошибок подписки, обработку в случае успешного и безуспешного подключений:  
  
 public void subscribeToTopic(String topic) {

try {

mqttAndroidClient.subscribe(topic, 0, null, new IMqttActionListener() {

@Override

public void onSuccess(IMqttToken asyncActionToken) {

Log.i("MQTT", "CONNECTED TO TOPIC");}

@Override

public void onFailure(IMqttToken asyncActionToken, Throwable exception) {

Log.i("MQTT", "NOT CONNECTED TO TOPIC");}});

} catch (Exception ex) {

Log.i("MQTT", ex.toString());

ex.printStackTrace();}}

Метод отписки от темы имеет следующее представление в коде:

public void unsubscribeFromTopic(String topic) {

try {

mqttAndroidClient.unsubscribe(topic, null, new IMqttActionListener() {

@Override

public void onSuccess(IMqttToken asyncActionToken) {

Log.i("MQTT", "UNSUBSCRIBED FROM TOPIC");}

@Override

public void onFailure(IMqttToken asyncActionToken, Throwable exception) {

Log.i("MQTT", "FAILED TO UNSUBSCRIBE FROM TOPIC");}});

} catch (Exception ex) {

Log.i("MQTT", ex.toString());

ex.printStackTrace();}}

В обоих методах используется конструкция try-catch, которая обрабатывает появление всевозможных критических ошибок и в случаях их возникновения выводит соответствующее сообщение.

## 4.7 Получение данных от MQTT брокера

При инициализации объекта подключения к MQTT брокеру определяется слушатель, который будет обрабатывать приходящие в темы сообщения. Слушатель обрабатывает только темы, на которые были оформлены подписки. Функция обработчик реализована следующим образом:

@Override

public void messageArrived(String topic, MqttMessage message) throws Exception {

String[] parts = topic.split("/");

String deviceId = parts[parts.length - 1];

Devices dev = new Devices(deviceId, "smth", "");

dbHelper.updateLastDataForDevice(dev, parseMqttMessage(message.toString()));

setDevicesList();

Log.i("MQTT", deviceId + " " + message.toString());}

В данной функции происходит обработка приходящего сообщения, а именно: поиск уникального идентификатора устройства из названия темы и обновление объекта устройства. После обновления последних приятых данных, хранящихся в базе данных, происходит обновление полей в списке устройств.

## 4.8 Обработка полученных данных

После принятия сообщения от MQTT брокера данные следует обработать так, как они приходят в «сыром виде». В данном методе происходит создание форматированного сообщения, путем выделения ключей и их значений, для сохранения в базе данных:

public String parseMqttMessage(String jsonMessage){

StringBuilder formattedMessage = new StringBuilder();

try {

JSONObject jsonObject = new JSONObject(jsonMessage);

Iterator<String> keys = jsonObject.keys();

while (keys.hasNext()) {

String key = keys.next();

String value = jsonObject.getString(key);

formattedMessage.append(key).append(": ").append(value).append("\n");}

} catch (JSONException e) {

e.printStackTrace();}

return formattedMessage.toString();}

Затем данная информация используется для отображения в поле данных в списке устройств:

if (device.getLastAcceptedData() != null) {

String[] keyValuePairs = device.getLastAcceptedData().split("\n");

StringBuilder newDataStringBuilder = new StringBuilder();

for (String pair : keyValuePairs) {

String[] keyValue = pair.split(":");

if (keyValue.length == 2) {

String key = keyValue[0].trim();

if (!key.equals("state\_" + device.getDiodeChannel())) {

newDataStringBuilder.append(pair).append("\n");

}}}

textDataCard.setText(device.getType() + "\n\n" + newDataStringBuilder.toString());

В случае, если устройство содержит переключаемый датчик, то в сообщении по ключу находится текущее состояние датчика, например «state\_l1» = «OFF» и устанавливается соответствующее значение в кнопке, отвечающей за переключение состояния датчика.

for (String pair : keyValuePairs) {

String[] keyValue = pair.split(":");

if (keyValue.length == 2) {

String key = keyValue[0].trim();

String value = keyValue[1].trim();

if (key.equals("state\_" + device.getDiodeChannel())) {

switchButton.setChecked(!value.equals("OFF"));

break;}}}

## 4.9 Отправка данных в MQTT брокер

Отправка сообщений в MQTT брокер выполняется только при переключении состояния кнопок, находящихся в списке устройств, в которых указаны каналы переключающихся датчиков. Для обработки переключения, в момент создания каждого объекта, в списке устройств определяются функции-обработчики, в которых происходит создание объекта сообщения, с соответстующим данными внутри, и отправка данного сообщения в тему, связанную с устройством. Обработчик реализован следующим образом:

holder.switchButton.setOnCheckedChangeListener((buttonView, isChecked) -> {

MqttMessage mqttMessage = new MqttMessage();

if (isChecked) {

mqttMessage.setPayload("On".getBytes());

Log.i("button", "On " + holder.getAdapterPosition());

} else {

Log.i("button", "Off " + holder.getAdapterPosition());

mqttMessage.setPayload("Off".getBytes());}

try {

Log.i("MQTT", device.getMqttPrefix());

mqttClient.publishMessage(device.getMqttPrefix() + "/" + device.getDeviceId() + "/" + device.getDiodeChannel() + "/set", mqttMessage);

} catch (MqttException e) {

throw new RuntimeException(e);}});

В обработчике используется метод отправки сообщения publishMessage объекта класса MqttHelper:

public void publishMessage(String topic, MqttMessage message) {

try {

mqttAndroidClient.publish(topic, mqttMessage);

} catch (Exception ex) {

ex.printStackTrace();}}

Как видно из метода, используется так же конструкция try-catch для предотвращения ошибок отправки.

## 4.10 Взаимодействие с базой данных

В базе данных созданы две таблицы: «устройства», которая ссылается на таблицу «системы». Для данных таблиц определены все CRUD операции (создание, чтение, обновление, удаление). При создания объекта класса DBHelper вызывается метод onCreate, в котором происходит создание таблиц, если они еще не созданы. Ниже приведен реализация метода:

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL("CREATE TABLE IF NOT EXISTS systems "

"(\_id INTEGER PRIMARY KEY AUTOINCREMENT, system\_name TEXT NOT NULL UNIQUE, "

"mqtt\_url TEXT NOT NULL, mqtt\_login TEXT, mqtt\_password TEXT)");

db.execSQL("CREATE TABLE IF NOT EXISTS devices "

"(\_id INTEGER PRIMARY KEY AUTOINCREMENT, friendly\_name TEXT, "

"device\_id TEXT NOT NULL UNIQUE, "

"device\_type TEXT NOT NULL, last\_data TEXT,"

"system\_id INTEGER NOT NULL, "

"img\_path TEXT," +

"mqtt\_prefix Text," +

"diode\_channel Text," +

"FOREIGN KEY (system\_id) REFERENCES systems(\_id) ON DELETE CASCADE)");}

В самом методе используется операция SQL «CREATE TABLE», в которой определяются все данные, необходимые для хранения на постоянной основе, для обоих таблиц.

Далее рассмотрим CRUD операции на основе таблицы «системы», так как описание методов для двух таблиц будет избыточно, и методы немногим схожи логикой между собой.

Для создания системы используется метод addSystem, принимающий объект system. Чтобы выполнить запись в базу данных создается объект db, через который происходит вставка строки. При этом вытягиваются значения из объекта system. Если логин и пароль неопределены, то в базе данных данные параметры будут принимать значение null. Реализация метода:

public void addSystem(System system) {

SQLiteDatabase db = getWritableDatabase();

ContentValues values = new ContentValues();

values.put("system\_name", system.getSystemName());

values.put("mqtt\_url", system.getMqtt\_url());

if(system.getMqtt\_login() != null && system.getMqtt\_password() != null){

values.put("mqtt\_login", system.getMqtt\_login());

values.put("mqtt\_password", system.getMqtt\_password());

Log.i("MQTT addSystem DB", system.getMqtt\_url() + " " + system.getMqtt\_login() + " " + system.getMqtt\_password());}

db.insert("systems", null, values);

db.close();}

При отображении списка систем необходимо получить данные об всех системах. Для этого применяется метод getAllSystems, возвращающий список добавленных систем. В процессе выполнени так же создается объект db, с помощью которого выполняется операция чтения. Затем в цикле происходит обход элементов списка результата выборки с занесением данных в созданные объекты sys, использующиеся для занесения информации в конечный список, возращаемый после завершения функции.

public List<System> getAllSystems() {

List<System> systems = new ArrayList<>();

SQLiteDatabase db = getReadableDatabase();

Cursor cursor = db.rawQuery("SELECT system\_name, mqtt\_url, mqtt\_login, mqtt\_password FROM systems", null);

if (cursor.moveToFirst()) {

do {

String systemName = cursor.getString(0);

String mqtt\_url = cursor.getString(1);

System sys = new System(systemName, mqtt\_url);

sys.setMqtt\_login(cursor.getString(2));

sys.setMqtt\_password(cursor.getString(3));

systems.add(sys);

} while (cursor.moveToNext());}

cursor.close();

db.close();

return systems;}

Метод deleteSystem, как следует из названия, выполняет удаление системы из базы данных. Внутри метода происходит вызов функции getSystemId, принимающая объект system для того, чтобы найти уникальный индентификатор записи и выполнить удаление записи по этому идентификатору. Перед удалением системы происходит удаление и устройств, связанных с данной системой.

public void deleteSystem(System system) {

String sysId = getSystemId(system);

SQLiteDatabase db = getWritableDatabase();

db.delete("devices", "system\_id = ?", new String[]{sysId});

db.delete("systems", "\_id = ?", new String[]{sysId});

db.close();}

Операция обновления системы реализована в функции updateSystem. Параметрами функции являются объекты класса System newSystem и oldSystem. Объект oldSystem представляет собой систему с неизмененными данными и предназначен для того, чтобы найти идентификатор системы в базе данных, необходимый для создания sql-запроса для обновления. Как и предыдущих методах из объекта newSystem извлекаются параметры системы и испльзуются дальше в sql-запросе.

public void updateSystem(System newSystem, System oldSystem){

String id = getSystemId(oldSystem);

SQLiteDatabase db = getWritableDatabase();

ContentValues values = new ContentValues();

values.put("mqtt\_url", newSystem.getMqtt\_url());

values.put("system\_name", newSystem.getSystemName());

if (newSystem.getMqtt\_login() != null && newSystem.getMqtt\_password() != null) {

values.put("mqtt\_login", newSystem.getMqtt\_login());

values.put("mqtt\_password", newSystem.getMqtt\_password());}

db.update("systems", values, "\_id = ?", new String[]{id});

db.close();}

В дополнение к этому реализованы вспомогательные методы, такие как проверка существующих названий системы, которые должны быть уникальными.

public boolean isSystemExists(String sysName) {

SQLiteDatabase db = getReadableDatabase();

Cursor cursor = db.rawQuery("SELECT COUNT(\*) FROM systems WHERE system\_name = ?", new String[]{sysName});

int count = 0;

if (cursor.moveToFirst()) {

count = cursor.getInt(0);}

cursor.close();

db.close();

return count > 0;}

В методе выполняется запрос с подсчетом строк с указанным именем и возвращается булевое значение true, если количество строк больше нуля, или false, если количество строк меньше нуля.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В данном разделе будет представлено руководство по взаимодействию с программным комплексом системы «Умный дом». Это руководство разделено на три части, поскольку в рамках дипломного проекта используется как ранее реализованная аппаратная часть, так и разрабатываемая программная часть:

* Руководство пользователя устройства координатора и конечного устройства. Этот раздел описывает процесс установки, настройки и использования устройства координатора, а также конечных устройств, таких как датчики, исполнительные устройства и другие. Здесь будут рассмотрены основные функции каждого устройства, способы подключения к сети и взаимодействия с другими устройствами в системе.
* Руководство пользователя мобильного приложения. Этот раздел посвящен использованию мобильного приложения для управления системой «Умный дом». Здесь будут представлены инструкции по загрузке, установке и запуску приложения на мобильных устройствах. Также будут рассмотрены основные функции приложения, включая добавление и удаление устройств, систем и упрвление ими.

## 5.1 Руководство по использованию устройства координатор.

Настройка координатора проходит в несколько этапов:

1. Для работы устройства требуется батарейка типа CR2032 либо подача питания через внешний источник, соединенный с координатором через USB.

2. После поключения к питанию, устройство создаёт точку доступа Wifi (см. рисунок 5.1), к которой необходимо подключиться с любого устройства, способного использовать сеть Wifi.

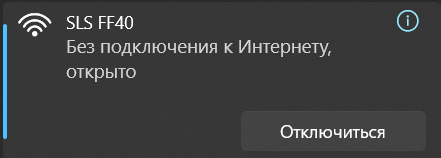


Рисунок 5.1 – Сеть Wifi, созданная координатором

3. Следующим действием, которое нужно сделать после подключения к точке доступа, необходимо зайти в веб-браузер и перейти по ссылке http:// http://192.168.100.1/.

4. На открывшейся странице (см. рисунок 5.2) будут изображены все Wifi сети, находящиеся рядом. В полях ввода необходимо ввести название и пароль рабочей сети пользователя, в пределах которой будет работать координатор.

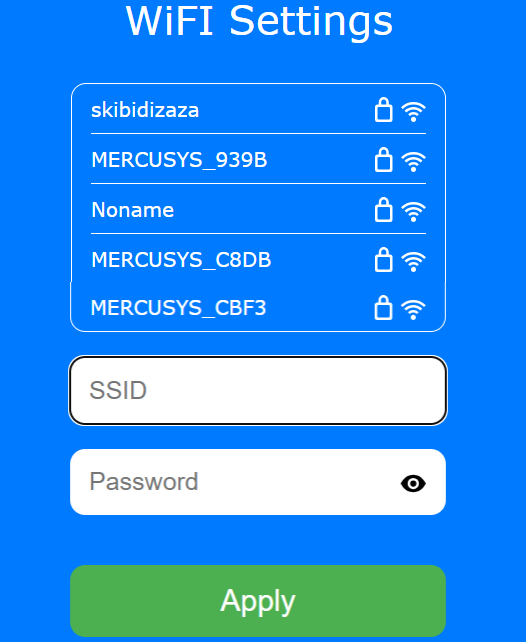


Рисунок 5.2 – Подключение координатора к рабочей сети Wifi

5. После того, как пользователь подключит координатор к своей сети Wifi, ему необходимо перезагрузить координатор путем отключения и включения питания. После этого координатор уже не будет создавать свою сеть Wifi, а будет подключен к определенной рабочей сети. Далее необходимо перейти по IP адресу координатора в рабочей сети. Чтобы его найти можно перейти на сайт маршрутизатора сети, где будут отображены подключенные устройства, либо использовать вспомогательные приложения, например, Advanced IP Scanner.

1. После перехода по IP адресу координатора откроется главная страница, где будет отображаться вся информация о конфигурации координатора (см. рисунок 5.3).

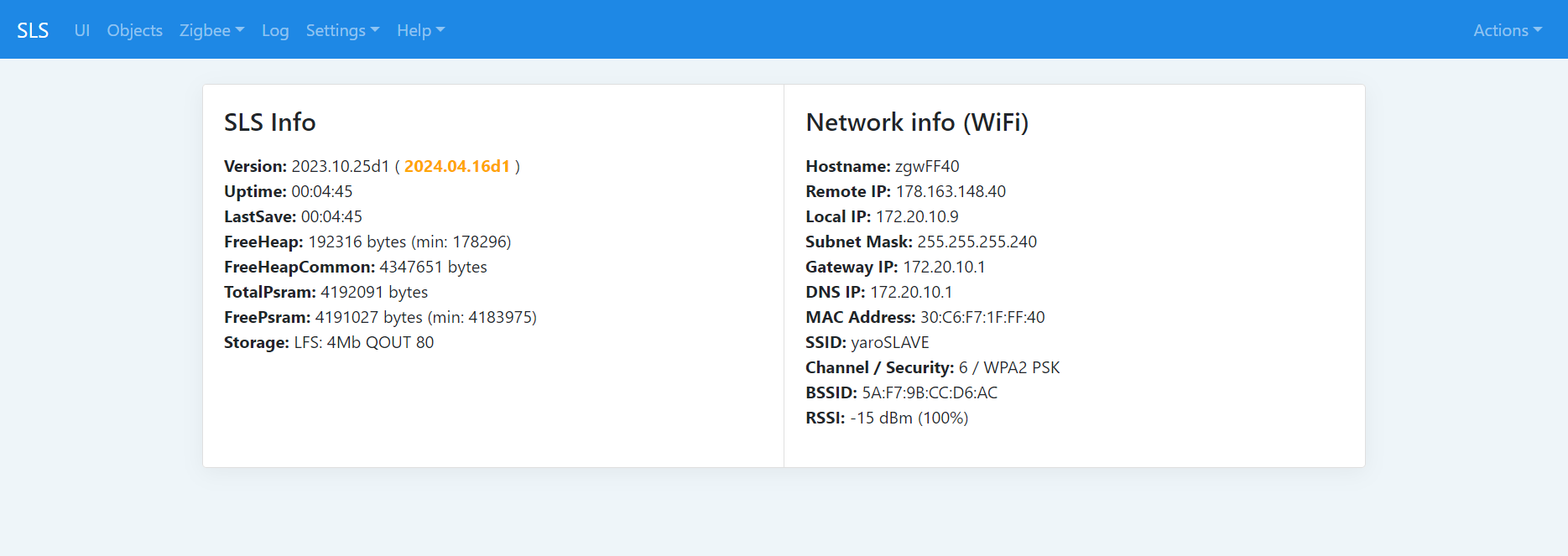


Рисунок 5.3 – Главная страница координатора

1. Для работы координатора следует настроить подключение к Mqtt брокеру. Для этого необходимо перейти на вкладку Settings > Link > MQTT Setup. Далее следует нажать на поле «Enable» под строкой «MQTT Setup», заполнить все данные для подключения и нажать кнопку «Save» для сохранения конфигурации (см. рисунок 5.4).



Рисунок 5.4 – Настройка подключения к Mqtt брокеру

1. Последним шагом будет указание контактов для взаимодействия ESP32 и E72, являющийся ведущим устройством в сети Zigbee, находящийся в составе координатора. Для этого необходимо перейти на вкладку Settings > Hardware. На открывшейся странице необходимо указать тип Zigbee модуля и номера контактов ESP32 работающие по протоколу UART с E72 (см. рисунок 5.5). После ввода информации необходимо нажать на кнопку «Save» в самом внизу страницы.



Рисунок 5.5 – Настройка подключения к E72

В результате настроек подключения к Mqtt брокеру и к Zigbee модулю E72 главная страница координатора должа выглядеть следующим образом (см. рисунок 5.6).

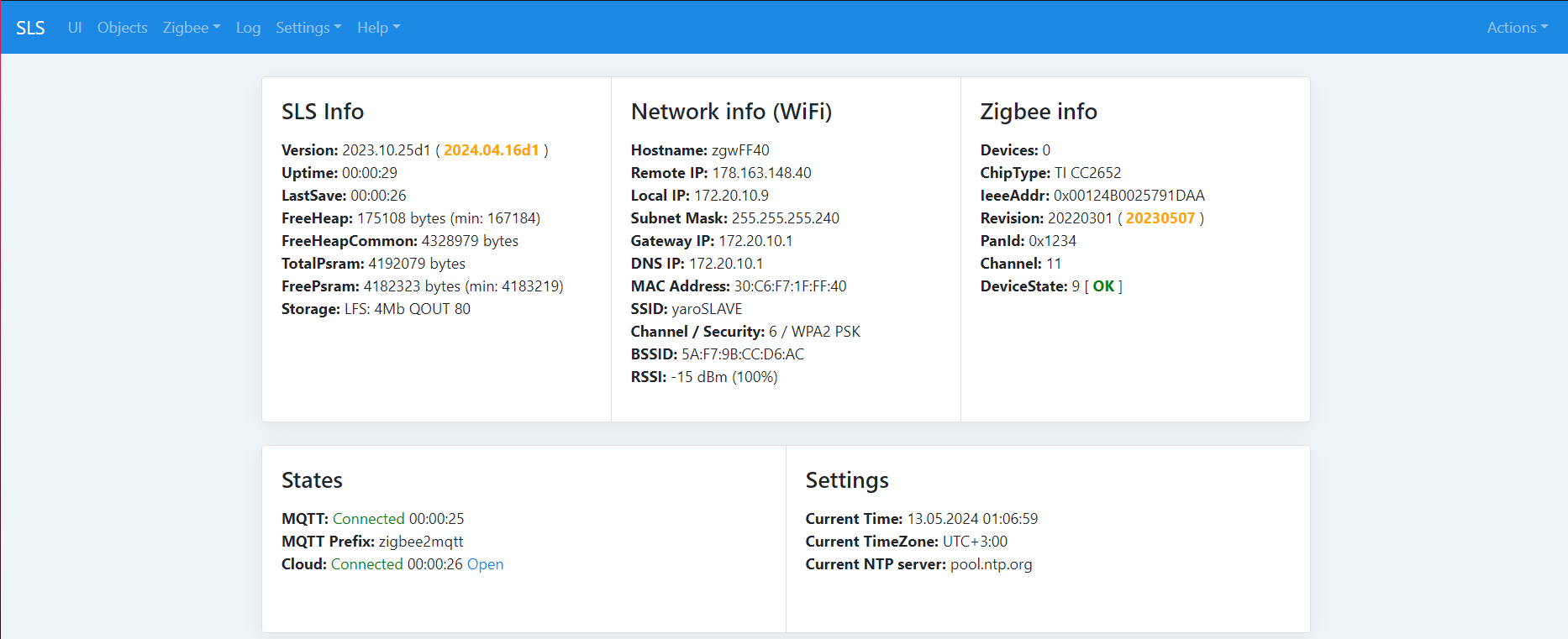


Рисунок 5.6 – Главная страница координатора после полной настройки

Если настройки верны, то в столбце «Zigbee info» в строке «DeviceState» дожно быть написано «OK» и в столбце «States» состояние подключения к брокеру Mqtt должно быть «Connected».

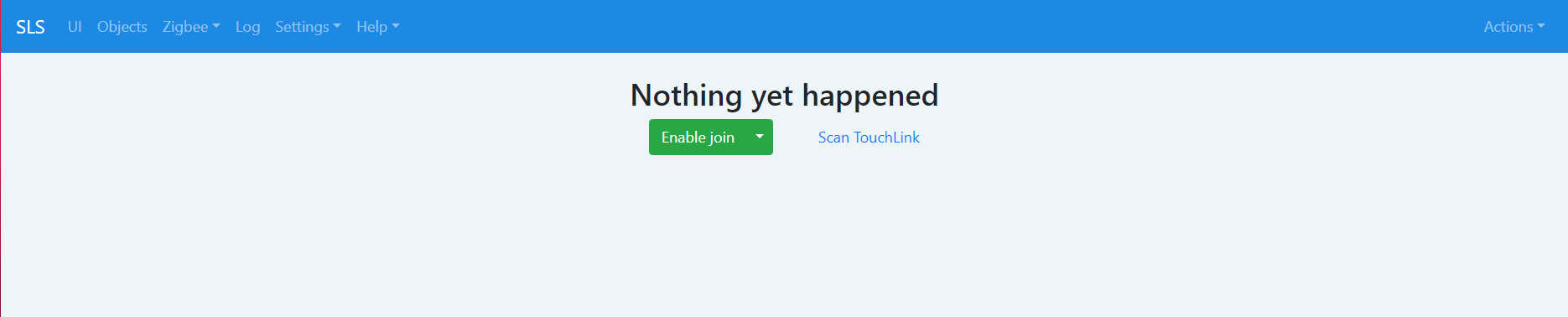
Для подключения конечного устройства необходимо перейти во кладку Zigbee > Join. Пользователь увидит следующую страницу (см. рисунок 5.7). 

Рисунок 5.7 – Страница подключения конечных устройств

Для поиска устройства необходимо нажать на кнопку «Enable join». В течении некоторого времени устройство будет обнаружено и подключено (см. рисунок 5.8).

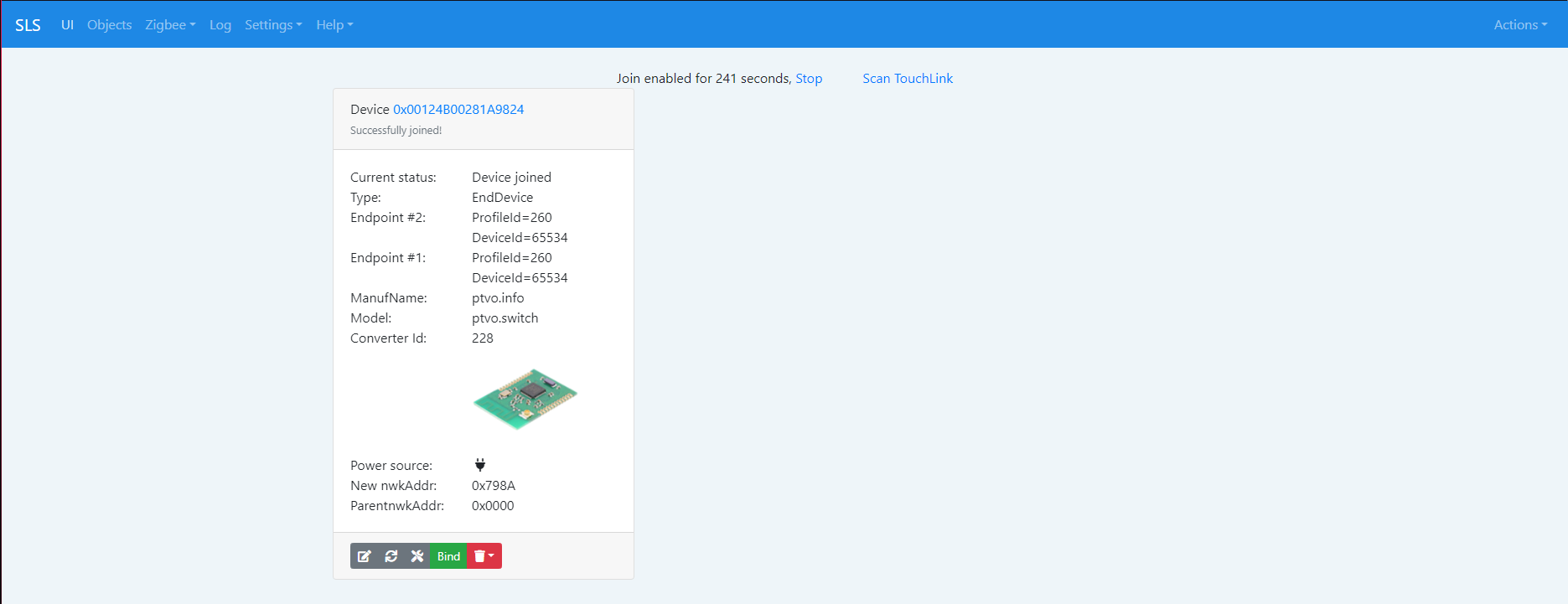


Рисунок 5.8 – Сопряжение с конечным устройством

## 5.2 Системные требования мобильного устройства

В этом разделе будут рассмотрены действия и необходимые требования к мобильным устройствам для корректной работы приложения. Для оптимальной работы мобильного приложения следует удостовериться в следующих условиях:

* на устройстве должна быть установлена операционная система Android версии 8.0 или более поздняя;
* доступ в интернет обязателен для мобильного устройства (мобильный интернет или сеть Wifi);
* наличие приложения «IotSystem» (или «Умный дом», в зависимости от установленного языка в телефоне) на мобильном устройстве также необходимо.

Для установки данного приложения, следует запустить установочный apk файл и выполнить стандартную процедуру установки. Если телефон откажется устанавливать приложение, необходимо перейти в настройки телефона и активировать режим разработчика. Процедура может отличаться в зависимости от модели устройства.

## 5.3 Руководство по использованию мобильного приложения

После выполнения вышеперечисленных действий можно приступать к работе с приложением. Для этого необходимо его запустить, нажав на появившуюся иконку с соответствующим названием (см. рисунок 5.9).



Рисунок 5.9 – Иконка приложения

После первого запуска приложения пользователю открывается экран с пустым списком систем, где будет написано «Нет систем» и кнопкой добавления новой системы внизу экрана (см. рисунок 5.10).

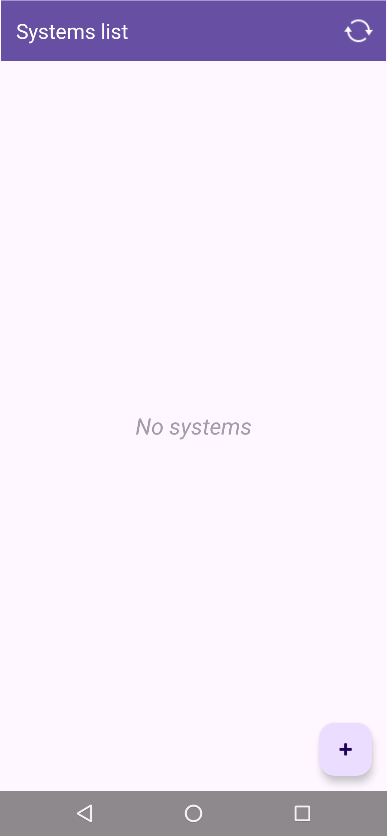


Рисунок 5.10 – Главный экран приложения

## 5.4 Добавление системы

На главном экране приложения со списком систем присутствует кнопка добавления системы внизу экрана. После того как пользователь нажмет на данную кнопку, откроется экран с формой для добавления данных о системе (см. рисунок 5.11 а). В первой форме должна быть введена информации об подключении к Mqtt брокеру:

* IP адрес или домен сервера;
* номер порта сервера;
* логин и пароль, если на сервере включена авторизация.

Для перехода к следующему этапу добавления необходимо нажать на кнопку «Далее». Если в полях ввода были введены некорректные данные, то после нажатия на кнопку переход не осуществиться и отобразиться уведомление с информацией об ошибке заполнения.

На конечном этапе добавления будет всего одно поле с вводом названия системы (см. рисунок 5.11 б), которое будет отображаться на главном экране. Название должно быть уникальным для каждой системы. После заполнения названия системы следует нажать на кнопку «Сохранить», в результате чего данные системы сохраняться в приложении и пользователю откроется главный экран с добавленной системой (см. рисунок 5.12).

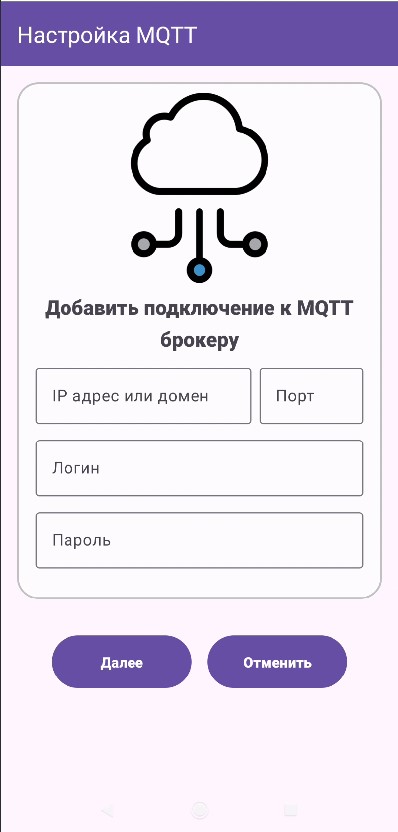
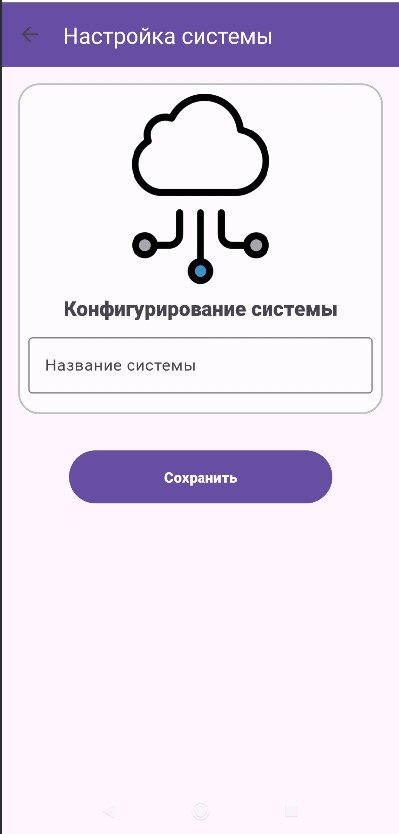
 

Рисунок 5.11 а) – Добавление информации для подключения к Mqtt брокеру, б) – Добавление названия системы

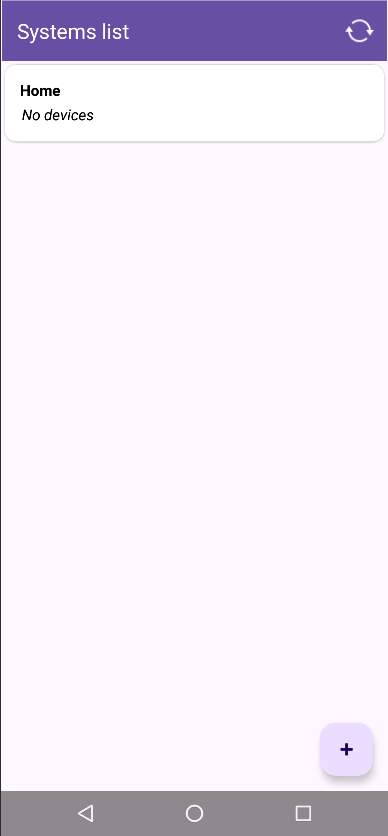


Рисунок 5.12 – Главный экран после добавления системы

## 5.5 Изменение и удаление системы

Возможности изменения и удаления системы реализованы свайпами в разные стороны данной системы в списке систем. При свайпе влево появиться иконка мусорной корзины на красном фоне (см. рисунок 5.13 а), что будет означать удаление объекта. После чего появится диалоговое окно с подтверждением удаления, где еще будет возможность отменить действие (см. рисунок 5.13 б).

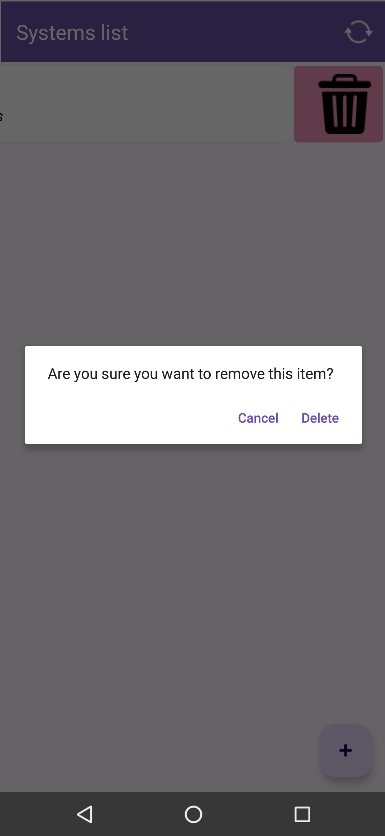
 

Рисунок 5.13 а) – Вызов окна для удаления системы, б) – Диалоговое окно при удалении системы

При свайпе вправо появиться иконка карандаша на зеленом фоне (см. рисунок 5.14 а), после чего откроется экран с формами ввода, которые уже были при добавлении системы (см. рисунок 5.14 б). Действия для редактирования и добавления системы схожи.

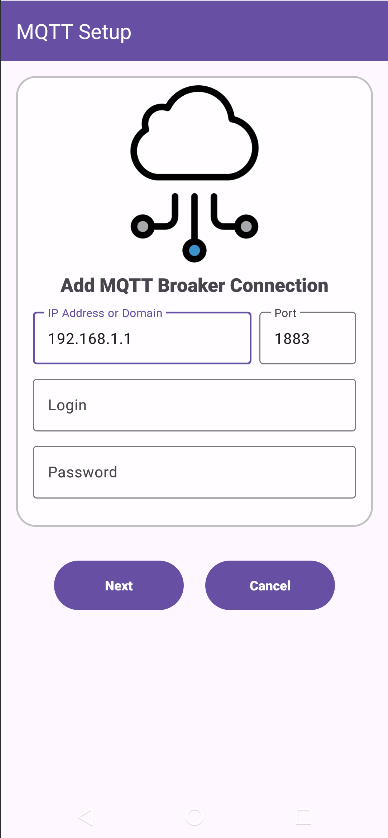
 

Рисунок 5.14 а) – Вызов экрана для редактирования данных системы, б) – Открывшийся экран после вызова редактирования

## 5.6 Открытие списка устройств и добавление устройства

Для того, чтобы открыть список устройств, связанных с системой, необходимо быть подключеным к сети Интернет и кликнуть на систему в списке после чего откроется экран со списком устройств (см. рисунок 5.15 а). Если устройства еще не были добавлены, то на экране будет отображаться надпись «Нет устройств». Для того, чтобы добавить устройство, в случае как и с добавлением системы, следует нажать на кнопку внизу экрана, в следствии чего откроется экран с формами ввода (см. рисунок 5.15 б).

Для добавления устройства требуется ввести следующие данные:

* название устройства, которое будет отображаться в списке устройств;
* префикс Mqtt, который был указан при настройке подключения координатора к Mqtt серверу;
* уникальный идентификатор устройства;
* тип устройства;
* канал диода или другого управляемого датчика.

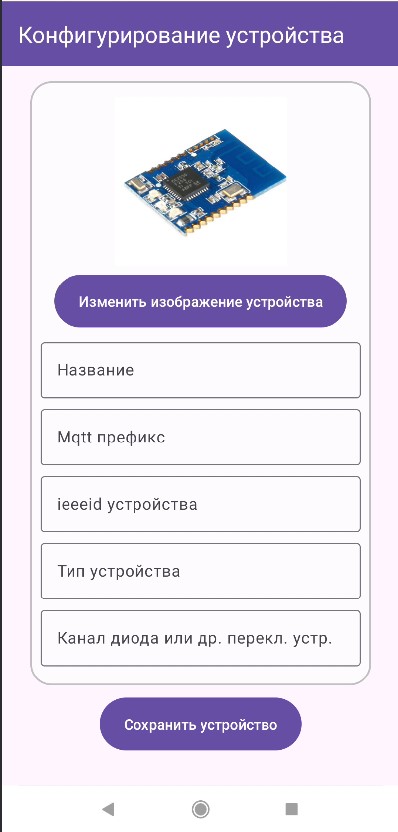
 

Рисунок 5.15 а) – Список устройств, б) – Добавление нового устройства

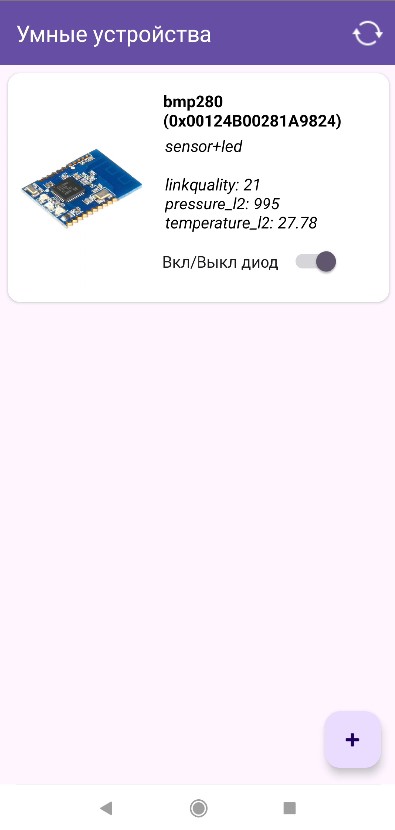


Рисунок 5.16 – Добавленное устройство

В результате при добавлении корректной информации, пользователю отобразиться список с добавленным устройством, где в течении некоторого времени будет добавлена полученая информация от самого датчика (см. рисунок 5.16).

## 5.6 Изменение и удаление устройства

Возможности изменения и удаления устройства реализованы аналогично системам с использованием свайпов в разные стороны в списке устройств. При свайпе влево появляется иконка мусорной корзины на красном фоне, указывающая на удаление объекта. После этого открывается диалоговое окно с подтверждением удаления, предоставляя возможность отменить действие.

При свайпе вправо появляется иконка карандаша на зеленом фоне, после чего открывается экран с формами ввода, аналогично процессу добавления устройства. Действия для редактирования и добавления устройства аналогичны тем, которые выполняются для системы.

## 5.6 Руководство программиста

Для работы с исходным кодом данного мобильного приложения следует произвести ряд действий по установке и настройке ПО. Базовыми требованиями являются:

1. Наличие Интернет соединения.

2. Свободное место на диске.

## 5.7 Установка IDE Android Studio

После выполнения вышеперечисленных шагов следующим этапом будет загрузка среды разработки Android Studio. Для этого необходимо перейти по ссылке на сайт разработчиков данной IDE: https://developer.android.com/studio/. На странице прокрутить вниз до раздела «Android Studio downloads», выбрать подходящую версию установщика и скачать её. Перед началом загрузки может появиться диалоговое окно, в котором необходимо установить галочку напротив согласия с условиями использования, после чего начнется загрузка (см. рисунок 5.17).

По завершении загрузки следует запустить скачанный файл. На экране компьютера появится окно приветствия, которое не влияет на процесс установки, поэтому пользователю следует нажать кнопку «Next». Далее будет предложен выбор устанавливаемых компонентов (см. рисунок 5.18). В нашем случае все компоненты должны быть выбраны, как на скриншоте. Затем появится окно с лицензионным соглашением. Необходимо принять условия лицензионного соглашения, нажав кнопку «I Agree». После этого появится окно выбора директории для установки программы (см. рисунок 5.19). Для корректной установки программы следует выбрать диск, на котором достаточно свободной памяти, поскольку данная программа может подгружать определённые ресурсы уже после установки.

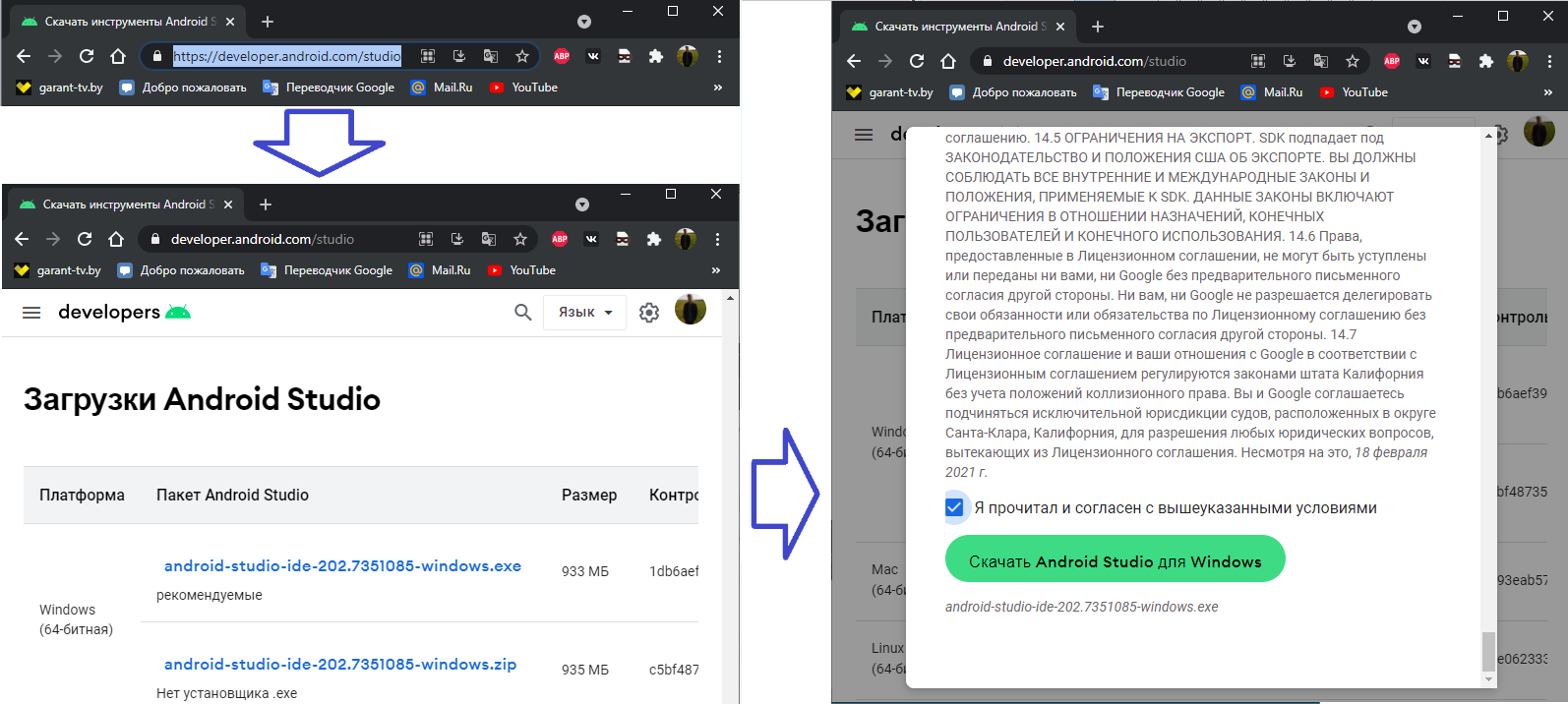


Рисунок 5.17 – скачивание Android Studio

После выбора директории необходимо нажать кнопку «I Agree». Затем продолжить нажимать клавишу «Next», пока не появится кнопка «Install». После этого окно с кнопкой «Install» будет финальным, и процесс установки начнется после её нажатия.

По завершении установки появится информационное окно. Следует нажать кнопку «Next», а затем «Finish». Теперь установка Android Studio завершена.

После завершения установки Android Studio рекомендуется перейти к установке дополнительных программных пакетов для возможности компиляции проекта под Android. Один из таких пакетов – JDK.

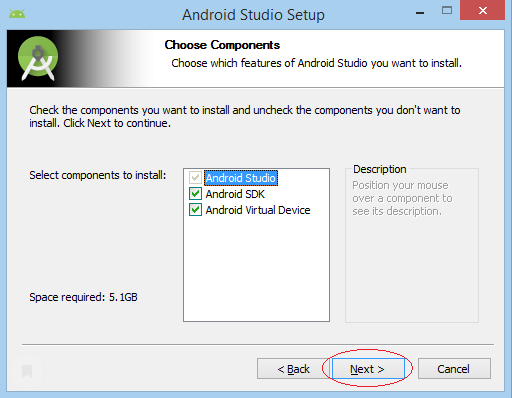
****

Рисунок 5.18 – Окно выбора компонентов Android Studio

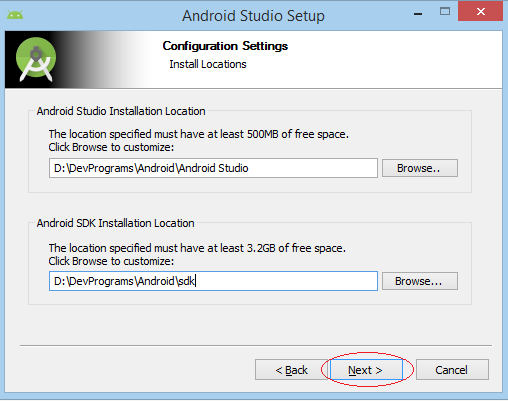
****

Рисунок 5.19 – Окно выбора директории Android Studio

## 5.8 Java Development Kit

Для начала пользователю необходимо скачать файл установщика, перейдя по ссылке https://www.oracle.com/java/technologies/downloads/#jdk20 и выбрав соответствующий вариант для операционной системы (см. рисунок 5.20). Обычно большинство пользователей предпочитают 64-разрядную версию ОС Windows, поэтому рекомендуется выбрать файл jdk-8u371-windows-x64.exe. После выбора файла начнется загрузка выбранного дистрибутива.

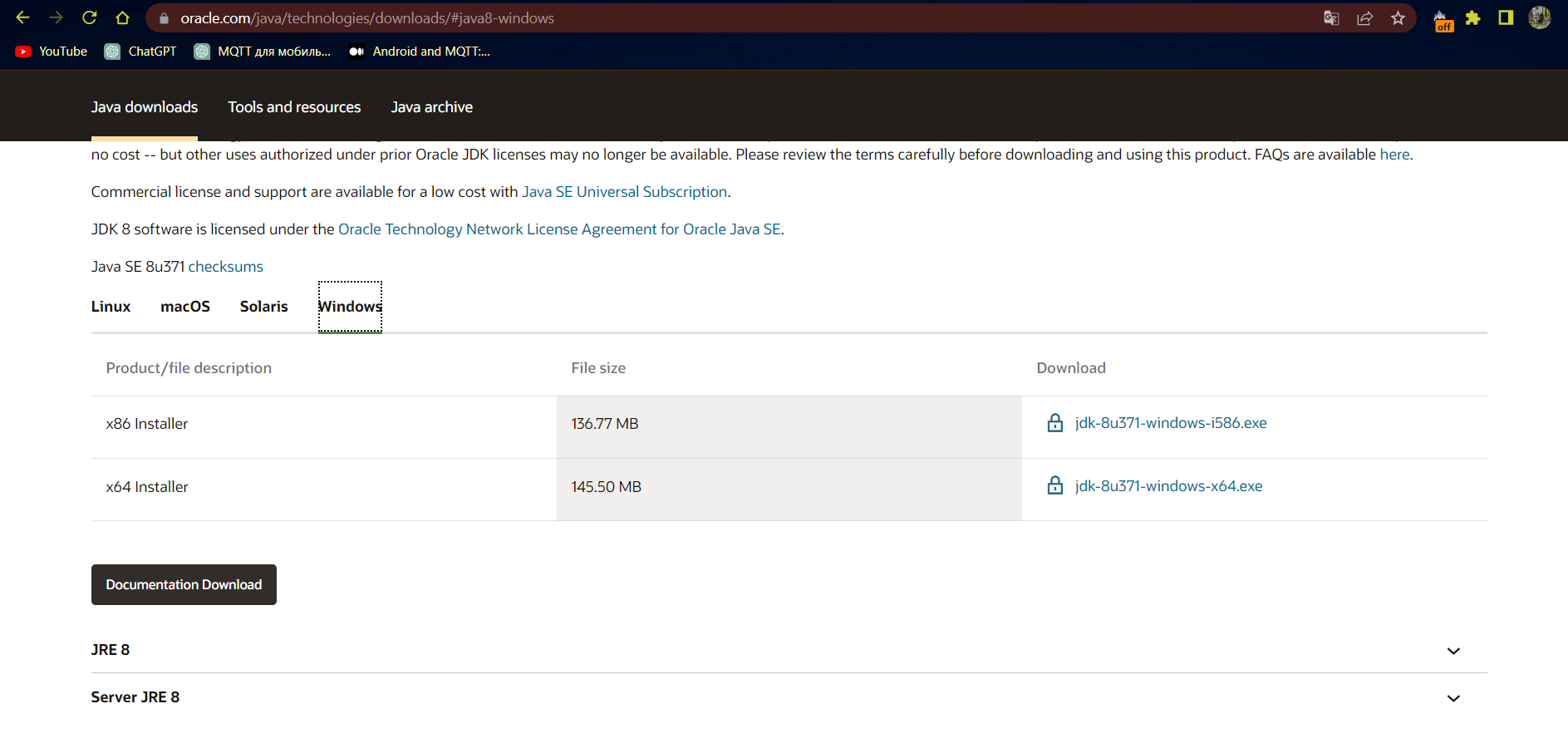


Рисунок 5.20 – Сайт загрузки JDK

После завершения загрузки слуедует запустить скачанный файл, чтобы начать процесс установки. Обратите внимание, что на диске, на который устанавливается JDK, должно быть достаточно свободного места. Поэтому при выборе директории установки необходимо убедиться, что выбранный диск соответствует этим требованиям. Также важно избегать использования пробелов и кириллических символов в пути установки (см. рисунок 5.21).



Рисунок 5.21 – Выбор места для JDK

## 5.12 MQTT сервер

Поскольку взаимодействие между комплексом и пользователем осуществляется через протокол MQTT, необходимо инициировать работу сервера, который будет выступать в качестве MQTT брокера.

5.12.1 Установка ПО

Для запуска сервера необходимо сделать следующее:

1. Перейти по ссылке https://mosquitto.org/download/ и нажать на mosquitto-2.0.18-install-windows-x64.exe, после чего должно начаться скачивание установочного файла;
2. Открыть загруженный файл, в результате чего должно появиться приветствующее окно, на котором следует нажать кнопку «Next»;
3. Далее откроется окно с выбором компонентов Mqtt брокера. Выбор элементов должен совпадать с выбором на рисунке 5.22, после чего следует нажать кнопку «Next»;

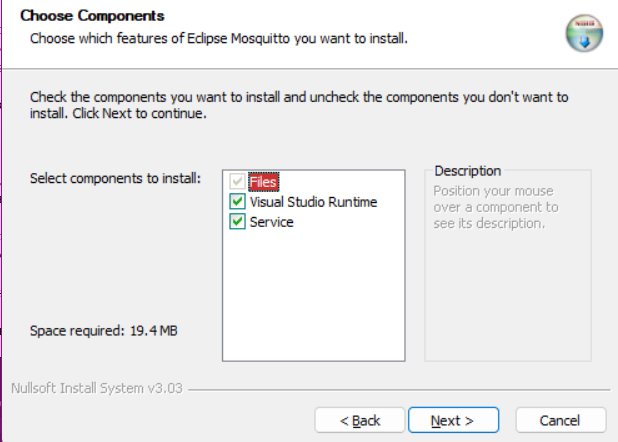


Рисунок 5.22 – Окно выбора элементов для установки

Далее следует окно с выбором директории установки, на котором пользователь должен указать место для установки;

Определившись с местом пользователь должен нажать кнопку «Install». После чего установка считается завершённой.

5.12.2 Настройка сервера

Перед запуском сервера требуется поменять его конфигурацию. Конфигурирование сервера достигается путем изменения файла mosquitto.conf, который находится в папке, куда производилась установка приложения.

Разработчику рекомендуется вписать такие же параметры, что и на рисунке 5.23. При такой конфигурации, подключение к серверу производится без авторизации. Если установка и настройка была произведена правильно, то при вводе команды netstat -an | findstr 1883 в командной строке PowerShell должно вывестись сообщение как на рисунке 5.24.

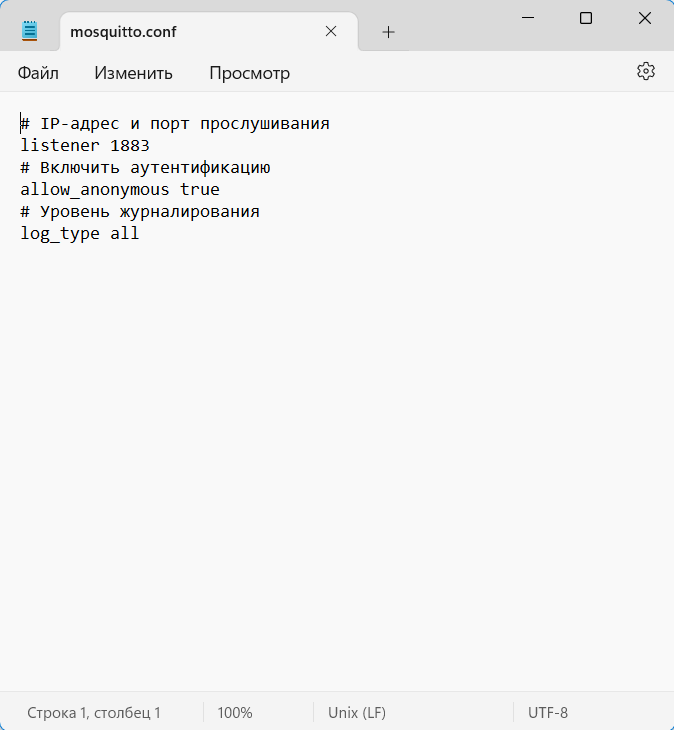


Рисунок 5.23 – Настройки конфигурации

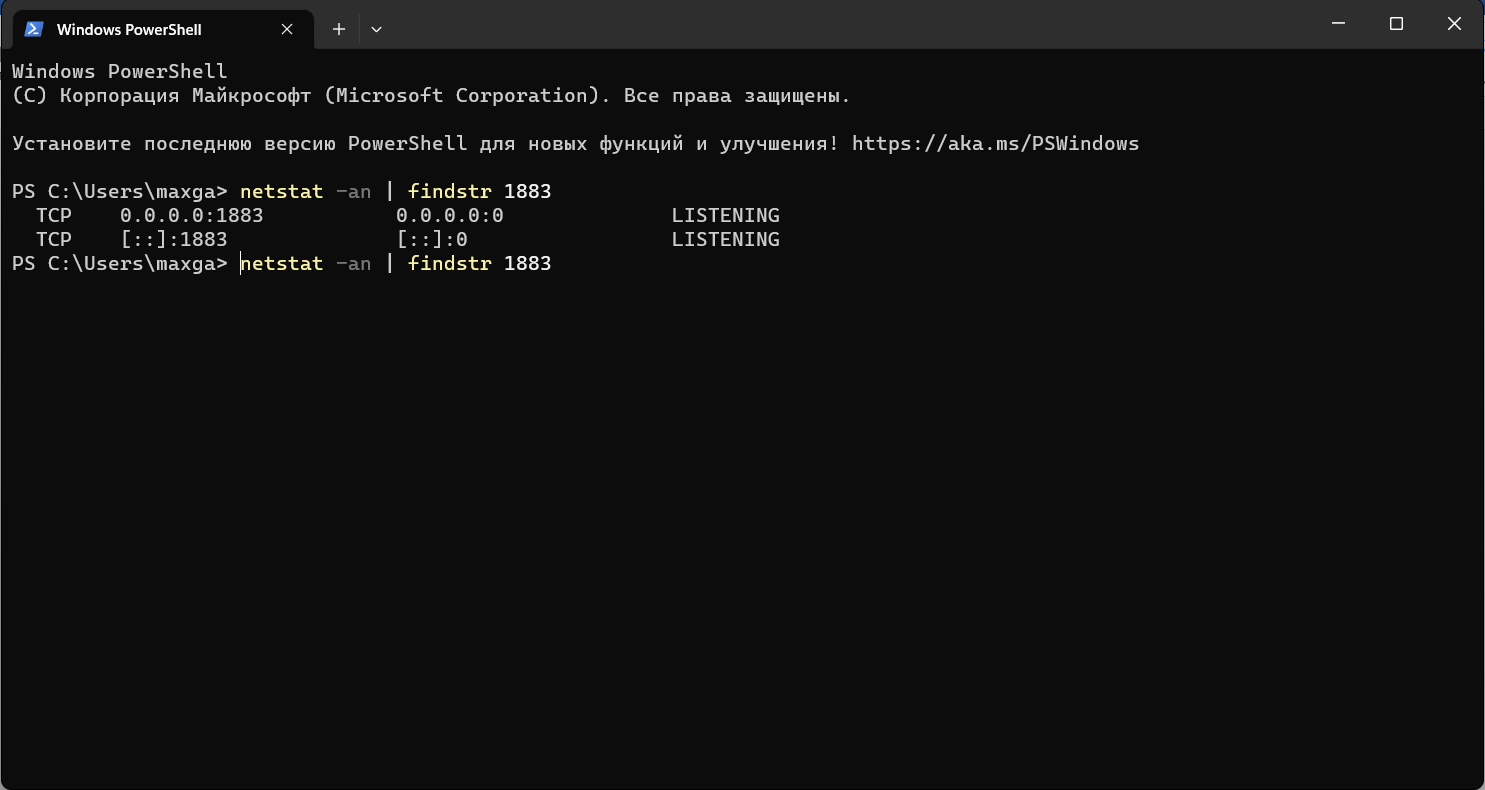


Рисунок 5.24 – командная строка PowerShell

Также запустить сервер можно при помощи служб Windows. При правильной установке и соблюдении всех пунктов в службах Windows должна появиться строка Mosquitto Broker (см. рисунок 5.25) со значениями столбцов:

– состояние: выполняется;

– тип запуска: автоматически;

– вход от имени: локальная система.

Если такая строка содержится в таблице, но имеет другие параметры, то их можно изменить, кликнув правой кнопкой мышки и выбрав нужный атрибут.

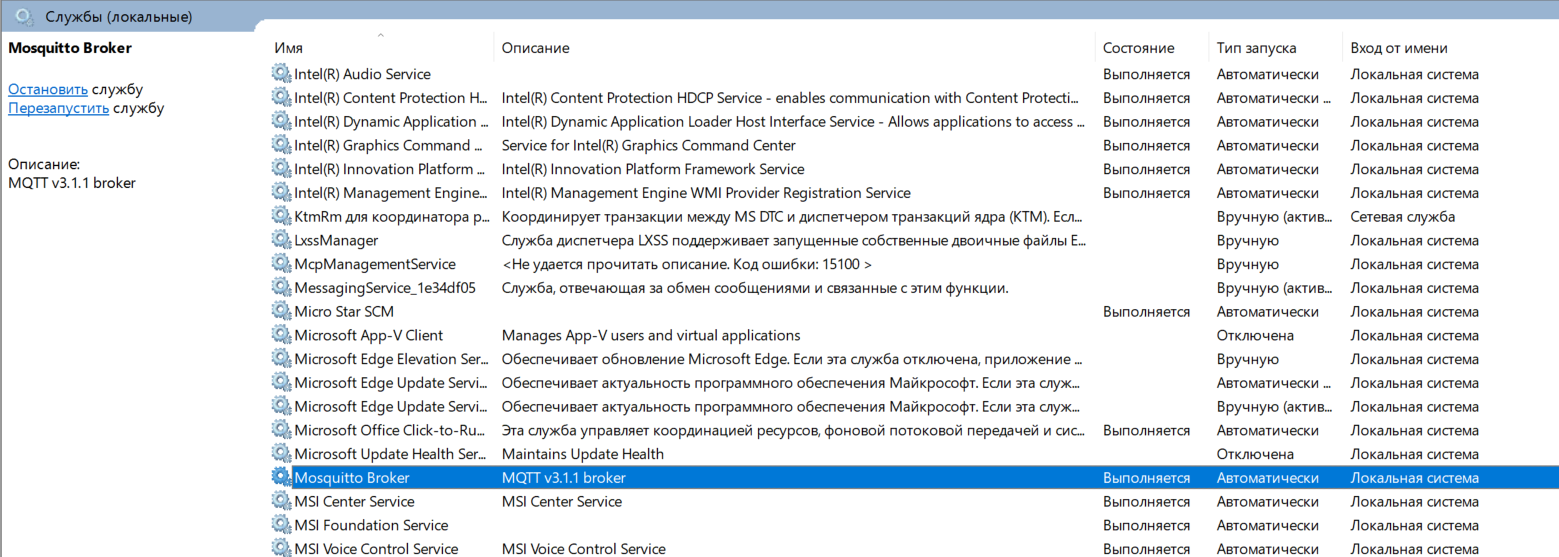


Рисунок 5.25 – Службы Windows

## 5.13 Установка ПО ESP32

Первоначально следует загрузить прошивку для ESP32, связывающий сеть Zigbee и сеть интернет. Для этого необходимо произвести следующие действия:

* перейти по ссылке: https://github.com/slsys/Gateway/tree/master/rom;
* нажать на файл 20230723\_full.zip (см рисунок 5.26);
* нажать на иконку с изображенной стрелкой вниз (с названием «Download raw file»).

Если действия были выполнены верно, то начнется скачивание архива.

После загрузки необходимо распаковать скачанный архив в любую папку на компьютере. Затем следует зайти в папку, в которую было выполнено распаковывание архива. В данной папке будет находиться еще четыре папки с названиями операционных систем. В зависимости от используемой операционной системы компьютером, необходимо выбрать папку с таким же названием, в которой будет находится ПО для координатора соответствующее данной системе. В дипломном проекте использовалась операционная система Windows. Перед пользователем появятся четыре необходимых файла расширения bat.

Перед установкой прошивки следует зажать кнопку Boot на ESP32 и подключить микроконтроллер через USB к компьютеру. Далее необходимо запустить файл Clear.bat, который очистит ESP32 от ранее установленых прошивок. После запуска файла откроется консольное окно, в котором будет отображаться информация об процессе очистки. В результате очистка должна завершиться успешно. Перед прошивкой устройства требуется отключить USB от компьютера, и, как делалось ранее, с зажатой кнопкой Boot снова подключить USB.

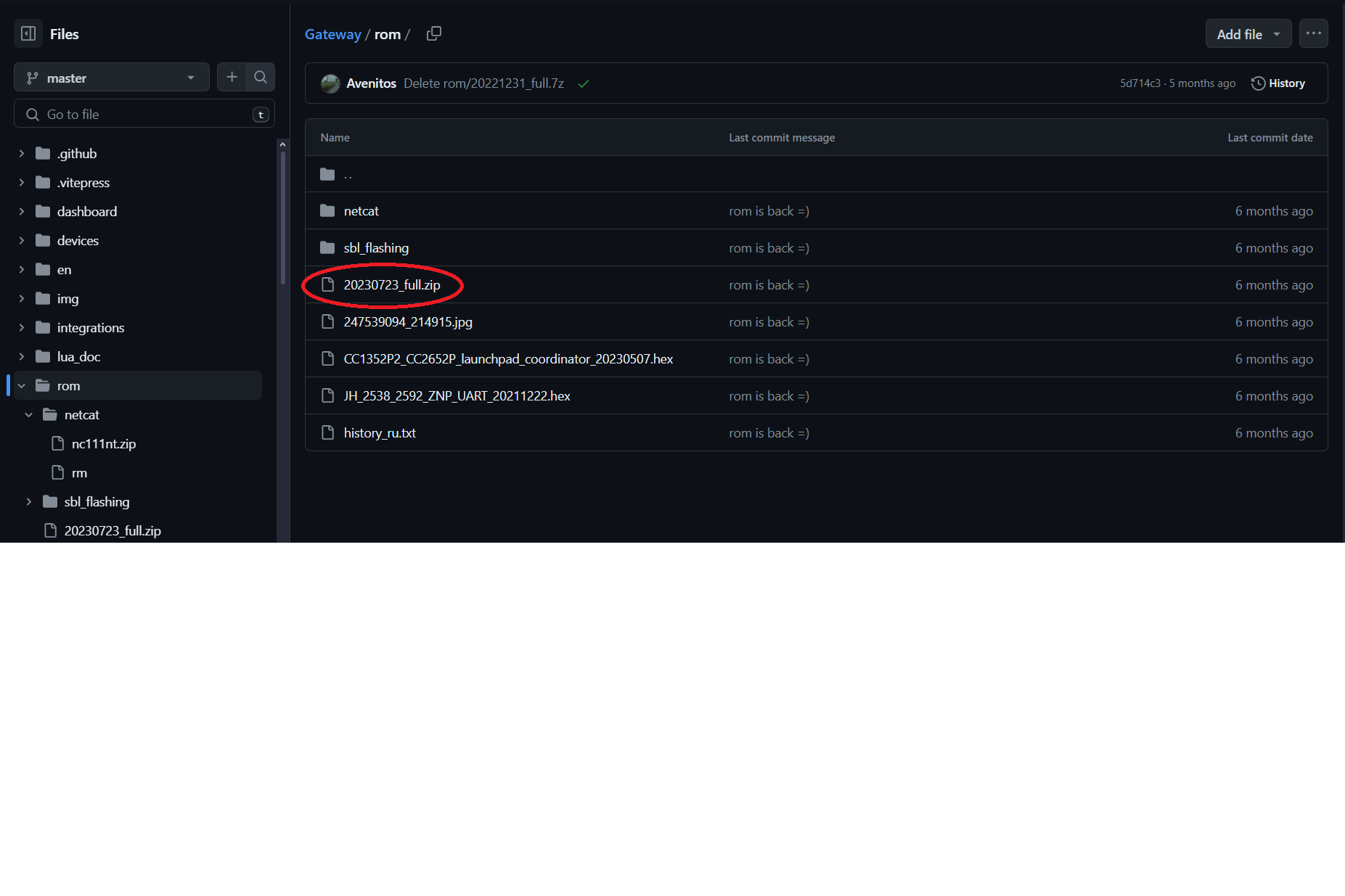


Рисунок 5.26 – Страница сайта GitHub

Чтобы прошить устройство, нужно запустить файл Flash\_4mb.bat, расположенный в той же папке, что и Clear.bat. После запуска этого файла откроется консольное окно, в котором будет отображаться информация о процессе прошивки. По завершении прошивки будет выведено сообщение об успешном завершении установки, после чего окно закроется автоматически.

## 5.14 Установка ПО E72

Для скачивания файлов прошивки E72 требуется перейти по https://github.com/egony/cc2652p\_E72-2G4M20S1E/blob/master/, нажать на вкладку «firmware», затем на «coordinator» и скачать архив. После загрузки архива его следует распаковать в любое место на компьютере.

Для загрузки прошивки на E72 требуется использовать устройство USB TTL, контакты которого необходимо подключить к контактам платы E72 следующим образом: 3V3 > 3V3, GND > GND, RXD > TX, TXD > RX. Также для загрузки прошивки требуется установить приложение Flash Programmer 2. Установка данного приложения включает лишь загрузку установочного файла с сайта https://www.ti.com/tool/download/FLASH-PROGRAMMER-2/1.8.2, открытие файла и указание пути установки. После открытия приложения пользователь встретит следующий интерфейс (см. рисунок 5.27).

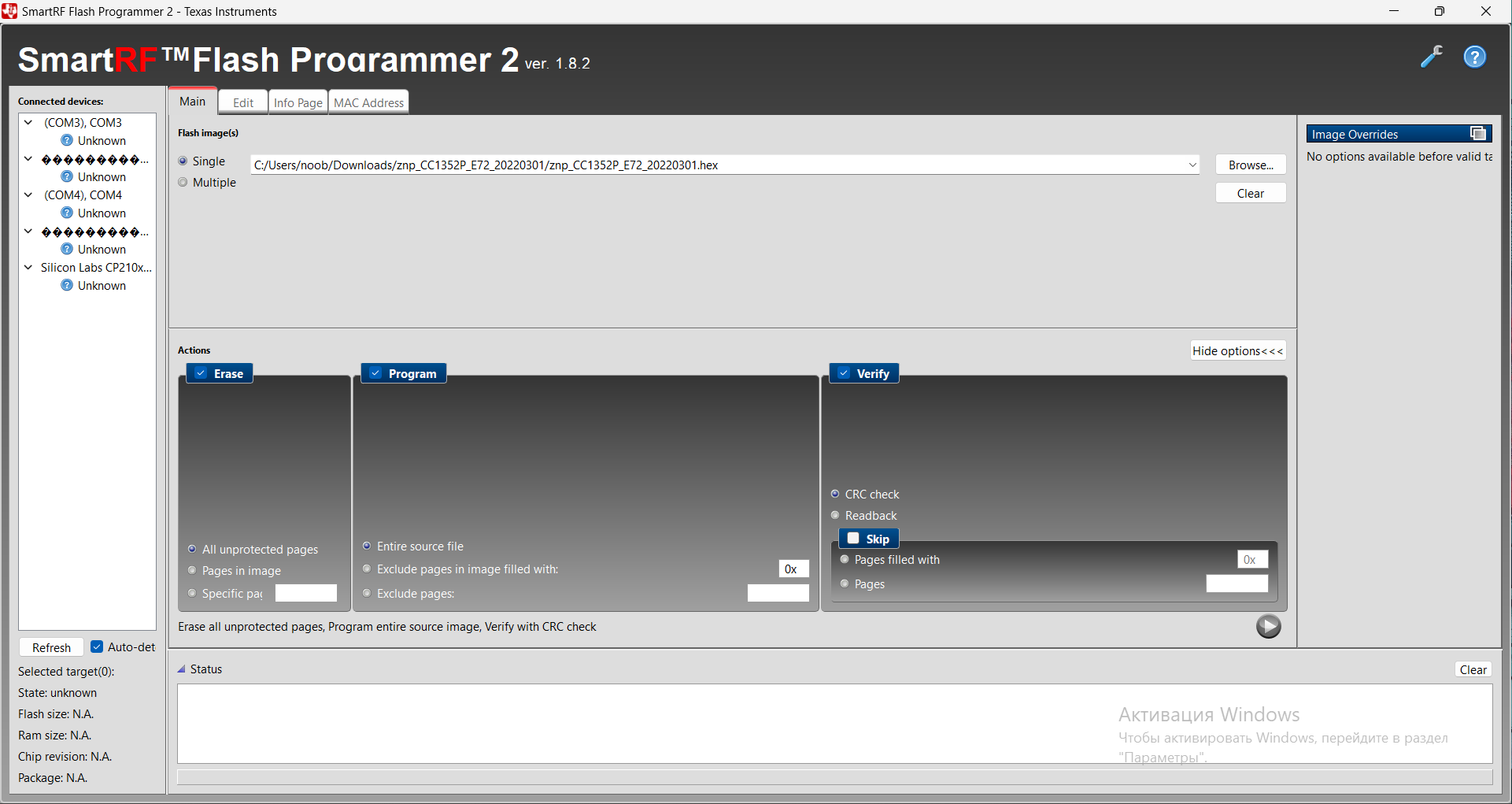


Рисунок 5.27 – Приложение Flash Programmer 2

На левой стороне интерфейса находится список подключенных устройств, где нужно найти и выбрать устройство USB TTL, обозначенное производителем. Затем в левом нижнем углу экрана нужно выбрать тип подключаемого устройства, который в дипломном проекте представлен как CC1352P. После этого пользователь должен выбрать устройство из списка и нажать на него, а затем выбрать опцию «Connect» (см. рисунок 5.28).

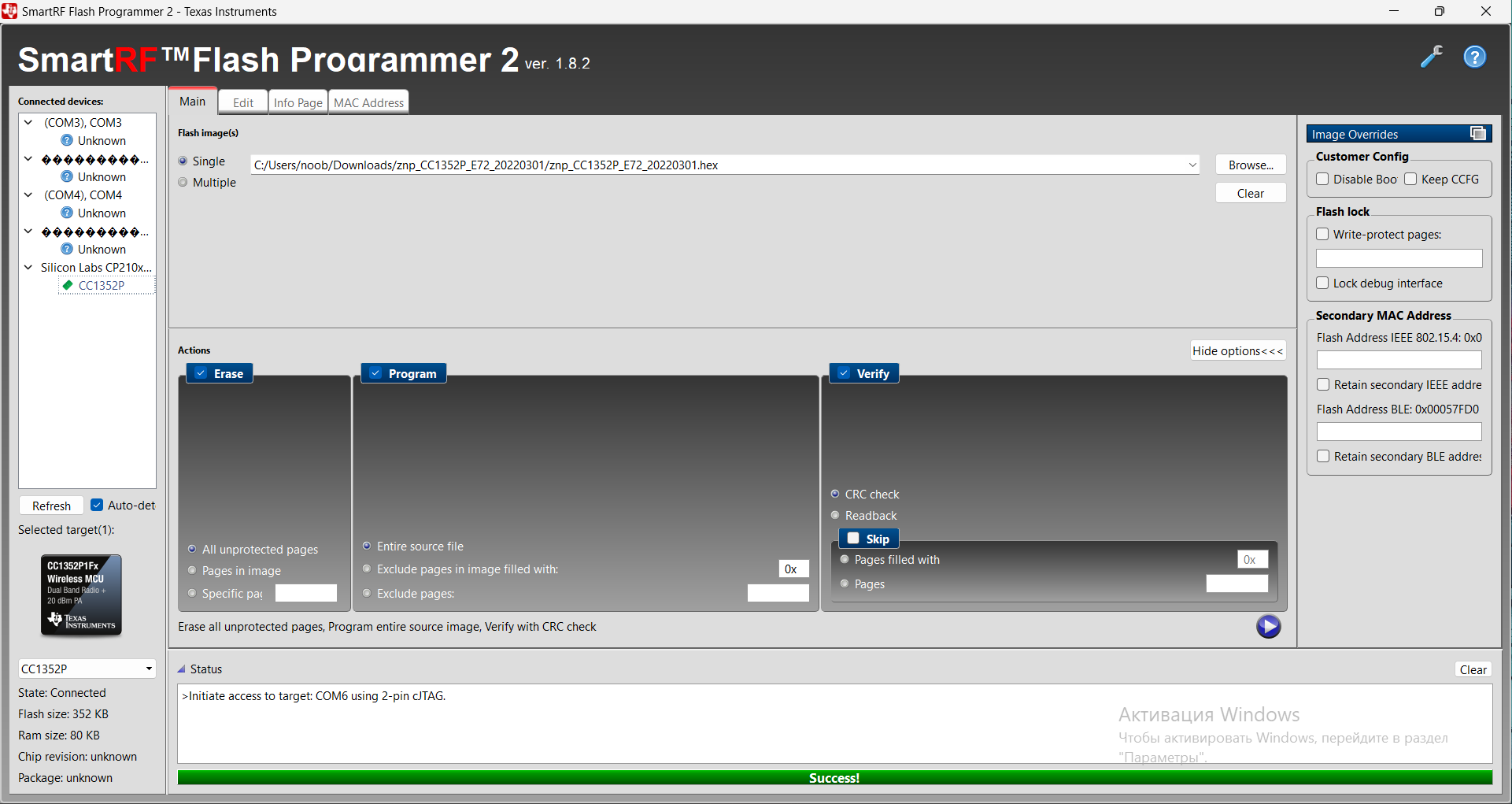


Рисунок 5.28 – Подключение устройства

В разделе «Flash images» необходимо указать путь к распакованному файлу прошивки, который был загружен ранее. Под пунктом выбора файла прошивки находятся настройки, где нужно установить галочки, аналогично изображению 5.28. Завершающим шагом будет нажатие на синюю кнопку с изображением треугольника, что запустит процесс прошивки. После завершения этого процесса устройство будет успешно настроено.

## 5.15 Разработка и установка ПО конечного устройства

Для разработки прошивки для конечного устройства рекомендуется использовать программу PTVO, которую можно скачать по ссылке https://ptvo.info/download/ptvo-firmware-latest.zip. Действия для установки данной программы такие же как и для установок предыдущих программ. После запуска программы пользователь встретить простой интерфейс (см. рисунок 5.29).

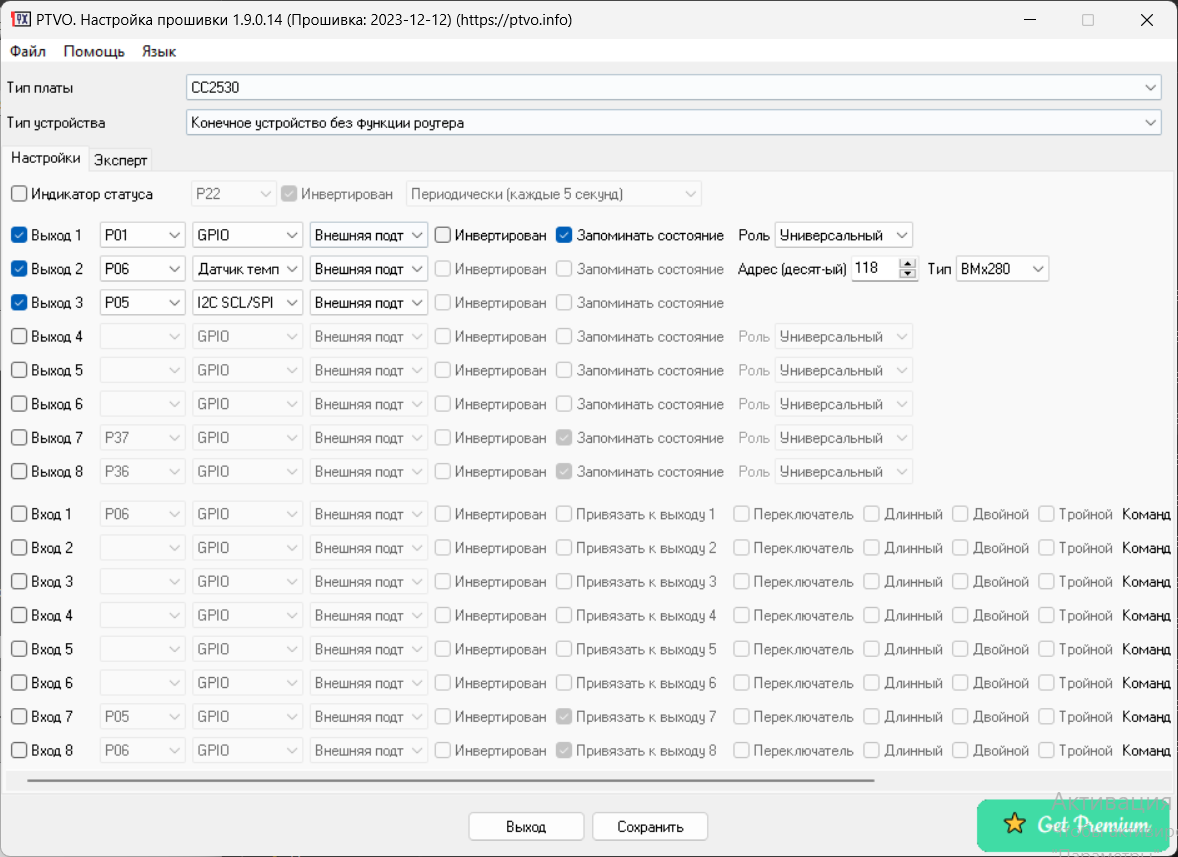


Рисунок 5.29 – Интерфейс программы PTVO

Для разработки прошивки под устройство, используемое в дипломном проекте, необходимо в полях «тип платы» и «тип устройства» выбрать опции, аналогичные показанным на рисунке 5.29. В приложении предусмотрена настройка контактов устройства как входных, так и выходных. Под выходными контактами подразумеваются виртуальные контакты, откуда будет поступать информация на координатор. При добавлении датчика или светодиода необходимо выбрать контакт, к которому он подключен, тип сигнала и тип подтяжки. В зависимости от типа подключаемого датчика, могут появляться различные опции для настройки. При добавлении светодиода следует выбрать тип устройства «GPIO», а при добавлении датчика – название датчика из списка. Если датчик подключается по протоколу I2C, то рекомендуется указать контакт SDL как «Датчик температуры» с дальнейшим выбором типа устройства и адреса датчика на линии I2C, а контакт SCL указать как «I2C SCL/SPI». После разработки прошивки следует сохранить ее, нажав на кнопку «Сохранить», после чего появится окно с выбором папки для сохранения. Файл прошивки будет иметь расширение hex. Представленная конфигурация на рисунке 5.29 является рабочей и предназначена для получения и отправки данных от светодиода и датчика BMP280.

Для установки разработанного файла прошивки на само конечное устройство типа E18 требуется скачать вспомогательную программу VLK DIY Multi\_Flasher. Скачать данную программу можно по ссылке http://82.146.46.112/file/VLK\_DIY\_Multi\_Flasher.rar. Распаковка и установка программы была уже описана ранее. Интерфейс программы включает форму для подключения конечного устройства и способы прошивки (см. рисунок 5.30).

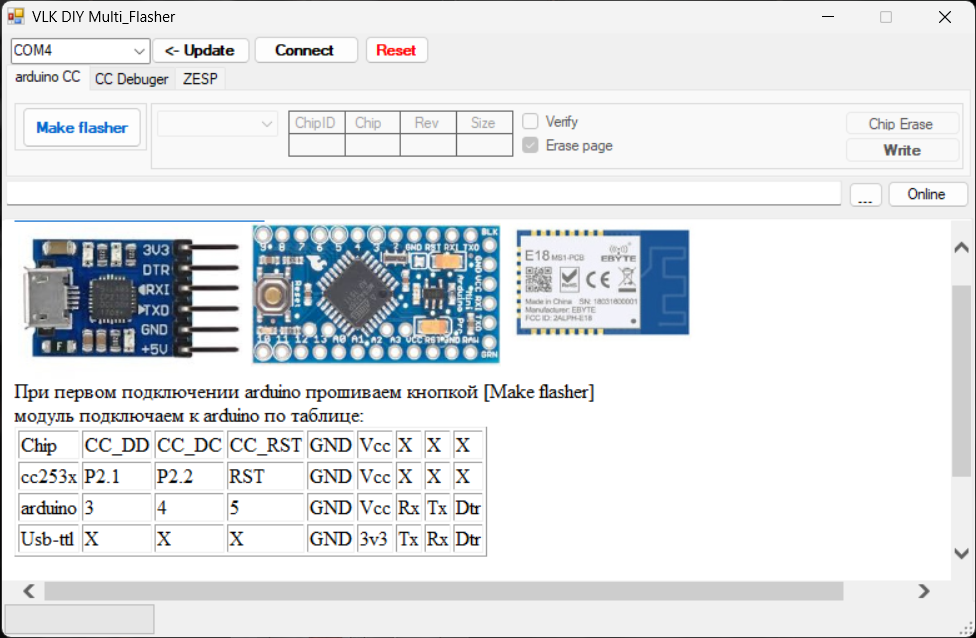


Рисунок 5.30 – Интерфейс программы VLK DIY Multi\_Flasher

В дипломном проекте прошивка конечного устройства производилась с помощью arduino. Для прошивки требуется подключить контакты arduino к контактам конечного устройства как показано в таблице на рисунке 5.30. После подключения контактов необходимо выбрать COM порт через, через который осуществлено подключение и нажать кнопку «Connect». Далее требуется нажать на кнопку «Make flasher» для того, чтобы aduino стало загрузочным устройством. Следующим шагом станет выбор файла прошивки после нажатия на кнопку, находящуюся слева от кнопки «Online» (см. рисунок 5.31). Затем необходимо очистить устройство от предыдущих прошивок путем нажатия на кнопку «Chip Erase» и загрузить новую прошивку нажатием на кнопку «Write». Процесс займет около полторы минуты. После чего устройство будет успешно прошито.

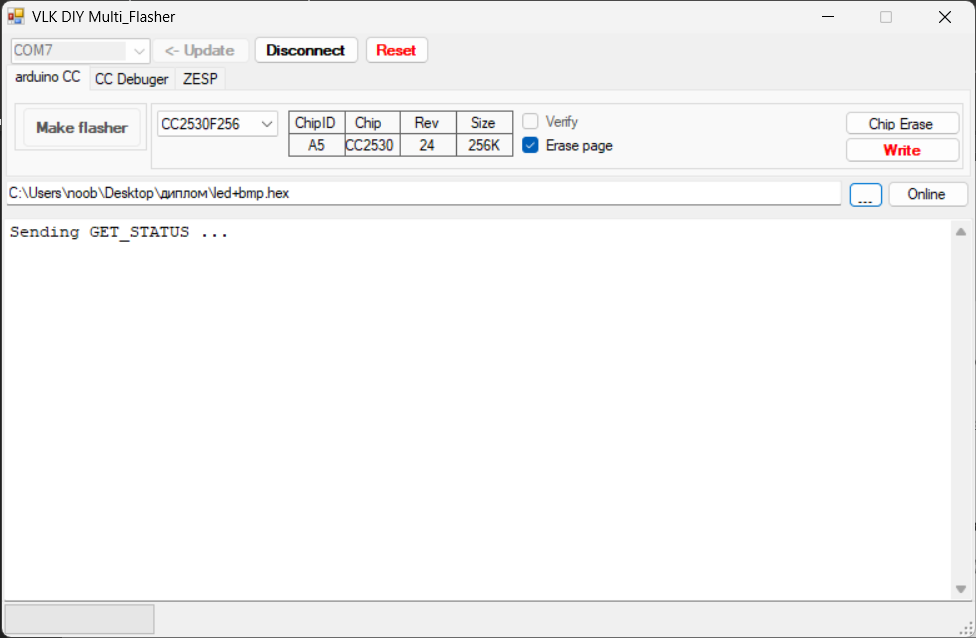


Рисунок 5.31 – Интерфейс приложения перед прошивкой

# 6 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Любая программа состоит из функций, которые взаимодействуют друг с другом, принимая различные параметры или объекты. Это взаимодействие может привести к изменению данных или формированию определенного ответа. Однако, из-за взаимодействия с человеком вводимые данные могут быть некорректными, что может привести к ошибкам или некорректным ответам. Чтобы избежать подобных ситуаций, разработчик обязан предусмотреть проверки или защитные механизмы.

Этот подход хорошо работает в маленьких программах, где функций немного и возможно ручное отслеживание ошибок. Однако, в больших программах с множеством классов и методов это становится трудоемким. Поэтому существует множество методов и техник тестирования программного обеспечения.

Само тестирование программного обеспечения - это процесс анализа программы и сопутствующей информации для выявления дефектов и повышения качества продукта. Обычно программа разбивается на модули, которые тестируются отдельно. Эти модули обычно соответствуют отдельным классам и методам.

В данном разделе представлена информация и руководство по проведению испытаний, направленных на достижение высокого уровня качества и надежности приложения перед его выпуском.

## 6.1 Модульное тестирование

Первый этап тестирования приложения – модульное тестирование. Оно направлено на изолированную проверку каждого модуля приложения для выявления корректности и нахождения неисправностей в конкретных участках программы. Модульное тестирование позволяет проверить отдельные методы или классы, что ускоряет процесс обнаружения и устранения ошибок. Такие тесты выполняются быстро и эффективно, поэтому их стоит проводить после каждого изменения в приложении, особенно в участках кода, где вносятся изменения.

Для проведения модульных тестов в данной работе использовались следующие технологии:

* JUnit Jupiter был использован для написания тестовых методов и управления порядком выполнения инструкций.
* Mockito применялся для создания мок-объектов, записи и проверки поведения объектов в процессе тестирования.
* EasyRandom использовался для заполнения объектов классов случайными значениями, что способствует разнообразию входных данных и позволяет более полно исследовать функциональность приложения.

В рамках модульного тестирования также проводилась проверка граничных случаев и неправильных входных данных для обеспечения корректной работы каждого метода в различных сценариях использования. Это позволяет выявить потенциальные уязвимости или неожиданное поведение программы при нестандартных условиях и обеспечивает более надежное и стабильное функционирование приложения. Также были применены методы анализа кода и статического анализа для выявления возможных ошибок и неправильного использования API. В результате модульного тестирования было обеспечено полное покрытие кода тестами, что повысило уверенность в его качестве и устойчивости к изменениям.

6.1.1 Класс ConnectTest

Данный класс используется для тестирования метода connect класса MqttHelper. Описание методов приведено в таблице в таблице 6.1.

Таблица 6.1 – Модульный тест ConnectTest

|  |  |
| --- | --- |
| Содержание теста | Ожидаемый результат |
| ConnectTest ValidInput | |
| Содержание тела запроса для осуществления подключения к серверу с корректными параметрами. | Метод должен вернуть сообщение «Successfylly connected» в лог программы. |
| ConnectTest InvalidInput | |
| Содержание тела запроса для осуществления подключения к серверу с ошибочными значениями. | Метод должен вернуть сообщение «Failed to connect» в лог программы с указанием ошибки. |

6.1.2 Класс DisconnectTest

С помощью данного класса производится проверка отключения от сервера в методе disconnect класса MqttHelper. Описание методов приведено в таблице 6.2.

Таблица 6.2 – Модульный тест DisconnectTest

|  |  |
| --- | --- |
| Содержание теста | Ожидаемый результат |
| DisconnectTest ValidInput | |
| Формирование запроса для разрыва соединения между приложением и сервером с корректными значениями и последующий вызов метода disconnect. | Метод должен вернуть сообщение «Disconnected» в лог программы. |
| DisconnectTest InvalidInput | |
| Формирование запроса для разрыва соединения между с сервером с неверными значениями и последующий вызов disconnect. | Метод должен вернуть сообщение «Failed to disconnect» в лог программы с демонстрацией ошибки. |

6.1.3 Класс PublishTest

Данный класс реализует тестирование метода publish класса MqttHelper. Описание методов приведено в таблице 6.3

Таблица 6.3 – Модульный тест PublishTest

|  |  |
| --- | --- |
| Содержание теста | Ожидаемый результат |
| PublishTest ValidInput | |
| Создание тела запроса для отправки сообщения с корректными значениями и вызов метода publish. | Метод должен вернуть сообщение «Publish message» в лог программы. |
| PublishTest InvalidInput | |
| Формирование тела запроса для отправки сообщения с неверными значениями и вызов метода publish. | Метод должен вернуть сообщение «Failed to publish» в лог программы. |

6.1.4 Класс SubscribeTest

С помощью данного класса производится тестирование метода subscribeToTopic класса MqttHelper. Описание методов приведено в таблице 6.4

Таблица 6.4 – Модульный тест SubscribeTest

|  |  |
| --- | --- |
| Содержание теста | Ожидаемый результат |
| SubscribeTest ValidInput | |
| Формирование тела запроса для отправки сообщения с корректными значениями и вызов метода subscribeToTopic. | Метод должен вернуть сообщение «Subscribed» в лог программы. |
| SubscribeTest InvalidInput | |
| Создание тела запроса для отправки сообщения с ошибочными значениями и вызов метода subscribeToTopic. | Метод должен вернуть сообщение «Failed to subscribe» в лог программы. |

6.1.5 Класс UnsubscribeTest

Этот класс разработан с целью тестирования метода unSubscribeToTopic класса MqttHelper. Этот метод используется для отмены подписки на указанную тему MQTT. Входным параметром для этого метода является строка с названием топика, от которого нужно отписаться. Описание других методов класса приведено в таблице 6.5, где каждый метод предназначен для определенного теста.

Таблица 6.5 – Модульный тест unSubscribeToTopicTest

|  |  |
| --- | --- |
| Содержание теста | Ожидаемый результат |
| unSubscribeToTopicTest ValidInput | |
| Формирование тела запроса для отправки сообщения с корректными значениями и вызов метода unSubscribeToTopic. | Метод должен вернуть сообщение «Unsubscribed» в лог программы. |
| unSubscribeToTopicTest InvalidInput | |
| Создание тела запроса для отправки сообщения с неверными значениями и вызов метода unSubscribeToTopic. | Метод должен вернуть сообщение «Failed to unsubscribe» в лог программы. |

6.1.6 Класс isInternetConnectionTest

Этот класс выполняет проверку метода isInternetConnection в классе MainActivity. Этот метод не требует входных параметров и оперирует только системными данными. Описание методов приведено в таблице 6.6

Таблица 6.6 – Модульный тест isInternetConnectionTest

|  |  |
| --- | --- |
| Содержание теста | Ожидаемый результат |
| isInternetConnectionTest ValidInput | |
| Генерация корректных значений параметров системы. | Функция возвращает значение true. |
| isInternetConnectionTest InvalidInput | |
| Генерация ошибочных значений параметров системы. | Функция возвращает значение false и выводит сообщение «No internet connection» в лог программы. |

6.1.7 Класс DBHelperTest

С помощью данного класса выполняется тестирование методов класса DBHelper. Описание методов приведено в таблице 6.7. Методы класса DBHelper выполняют операции создания, чтения, обновления, удаления.

Таблица 6.7 – Модульный тест DBHelperTest

|  |  |
| --- | --- |
| Содержание теста | Ожидаемый результат |
| 1 | 2 |
| testDBHelperOnCreate | |
| Создание объекта класса DBHelper. | Создание таблиц Systems и Devices. |
| testDBHelperAddSystem | |
| Генерация объекта класса System и его передача в метод addSystem. | Добавление объекта в таблицу Systems. |
| testDBHelperAddDevice | |

Продолжение таблицы 6.7

|  |  |
| --- | --- |
| 1 | 2 |
| Генерация объекта класса System с передачей его в метод addSystem, затем генерация объекта класса Device, который передается в метод addDevice совместно с объектом класса System. | Добавление объекта класса Device в таблицу devices, с привязкой к добавленному объекту System. |
| testDBHelperGetSystemId | |
| Передача в качестве параметра метода определенный объект System c заготовленными параметрами и вызов getSystemId. | Возвращение в качестве результата работы функции идентификатор объекта System в таблице. |
| testDBHelperGetDeviceId | |
| Передача в качестве параметра метода определенный объект Device и вызов метода getDeviceId. | Возвращение в качестве результата работы функции идентификатор объекта Device в таблице. |
| testDBHelperGetAllSystems | |
| Генерация вызова метода getAllSystems. | Возвращение в качестве результата работы функции списка объектов System. |
| testDBHelperGetAllDevices | |
| Генерация вызова метода getAllDevices c передачей определенного объекта System. | Возвращение в качестве результата работы функции списка объектов Device, привязанных к переденному объекту System. |
| testDBHelperDeleteSystem | |
| Передача в качестве параметра метода определенный объект System и вызов метода deleteSystem. | Удаление объекта System из таблицы systems и всех связанных с ним объектов Device из таблицы devices. |
| testDBHelperDeleteDevice | |
| Передача в качестве параметра метода определенный объект Device и вызов метода deleteDevice. | Удаление объекта Device из таблицы devices. |
| testDBHelperUpdateSystem | |
| Генерация объектов newSystem, oldSystem и передача в качестве параметра в метод updateSystem. | Обновление объекта oldSystem в таблице systems. |
| testDBHelperUpdateDevice | |
| Генерация объектов newDevice, oldDevice с определенными атрибутами и передача в качестве параметра в метод updateDevice. | Обновление объекта oldDevice в таблице devices. |

6.1.9 Класс NotificationTest

Данный класс тестирует метод showNotification класса NotificationHelper. Описание методов приведено в таблице 6.9

Таблица 6.9 – Модульный тест NotificationTest

|  |  |
| --- | --- |
| Содержание теста | Ожидаемый результат |
| NotificationTest ValidInput | |
| Формирование тела запроса для отправки сообщения с корректными значениями и вызов метода showNotification. | Вывод сгенерированного текста в оповещениях. |
| NotificationTest InvalidInput | |
| Формирование тела запроса для отправки сообщения с неверными значениями и вызов метода showNotification. | Вывод в лог программы «Invalid input» |

6.1.10 Покрытие модульных тестов

Результаты модульных тестов имеют важное значение для оценки качества мобильного приложения. Прохождение этих тестов позволяет определить, насколько эффективно функционируют различные модули приложения и соответствуют ли они ожиданиям. Полученные результаты помогают выявить ошибки и уязвимые места в работе приложения, что в конечном итоге способствует улучшению его стабильности и надёжности. Результаты модульных тестов могут быть использованы для выявления и устранения дефектов до выпуска приложения и для поддержания его работоспособности в долгосрочной перспективе.

## 6.2 Ручное тестирование

В ручном тестировании, завершающем этап проверки программного обеспечения, происходит имитация реальных сценариев использования приложения. Цель этого этапа - проверить его функциональность, удобство и надежность.

В рамках данного дипломного проекта был разработан список сценариев для ручного тестирования. Этот список предназначен для проверки тех же функциональностей приложения, но уже в реальных условиях эксплуатации.

Для каждого теста предполагается, что все остальные функции, за исключением тестируемой, работают корректно. Каждый тест-кейс содержит описание действий, которые должны быть выполнены для проверки определенной функциональности приложения. Эти действия подробно описаны в таблице 6.10.

Таблица 6.10 – Описание тест-кейсов

|  |  |  |
| --- | --- | --- |
| Наименование тест-кейса | Выполняемые шаги | Ожидаемый результат |
| 2 | 3 | 4 |
| 1 Запуск приложения | 1 Запустить приложение. | 1 Отображение главного экрана. |
| 2 Добавление системы | 1 Запустить приложение.  2 Ввести данные системы.  3 Нажать кнопку «Сохранить». | 1 Переход на гланый экран.  2 Отображение системы в списке систем. |
| 3 Проверка кнопки обновления списка систем на главном экране | 1 Запустить приложение.  2 Нажать кнопку обновления. | 1 Обновления списка систем. |
| 4 Проверка вывода списка устройств, связанных с системой | 1 Запустить приложение.  2 Нажать на элемент списка систем. | 1 Переход на экран со списком устройств.  2 Проверка соединения с интернетом  3 Загрузка добавленных устройств |
| 5 Проверка удаления системы | 1 Запустить приложение.  2 Свайпнуть элемент списка систем влево.  3 Нажать на кнопку «Удалить». | 1 Удаление системы из списка. |
| 6 Проверка изменения данных системы | 1 Запустить приложение.  2 Свайпнуть элемент списка систем вправо.  3 Заполнить новые данные.  4 Нажать кнопку «Сохранить». | 1 Сохранение новых данных системы.  2 Переход на главный экран.  3 Вывод списка систем с измененными данными. |
| 7 Проверка добавления устройства | 1 Запустить приложение.  2 Нажать на элемент из списка систем.  3 Нажать на кнопку добавления системы.  4 Ввести данные устройства.  5 Нажать на кнопку «Сохранить». | 1 Переход на экран со списком устройств.  2 Вывод добавленного устройства в списке. |
| 8 Проверка удаления устройства | 1 Запустить приложение.  2 Нажать на систему.  3 Свайпнуть элемент списка устройств влево.  4 Нажать кнопку «Удалить». | 1 Удаление устройства из списка. |

Продолжение таблицы 6.10

|  |  |  |
| --- | --- | --- |
| 9 Проверка изменения данных устройства | 1 Запустить приложение.  2 Нажать на элемент из списка систем.  3 Свайпнуть элемент списка устройств влево.  4 Ввести новые данные.  5 Нажать кнопку «Сохранить». | 1 Переход на экран со списком устройств.  2 Отображение списка устройств с измененными данными. |

Тесты были выполнены как на эмуляторе, так и на реальных мобильных устройствах, чтобы проверить совместимость приложения с различными версиями Android и корректность реализованного функционала. В процессе тестирования было удостоверено, что приложение работает стабильно и без ошибок на различных устройствах и версиях операционной системы Android. Все тесты были успешно завершены, возвращая ожидаемые результаты, что подтверждает работоспособность и надежность приложения.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ПРОГРАММНОГО КОМПЛЕКСА «УМНЫЙ ДОМ» С ПРИМЕНЕНИЕМ ТЕХНОЛОГИИ ZIGBEE

* 1. Описание функций, назначения и потенциальных пользователей программного комплекса

Программное средство представляет собой систему «Умный дом», разработанную для автоматизации и управления различными компонентами жилых помещений. Его цель – обеспечить комфорт, безопасность и эффективность в управлении различными устройствами, такими как освещение, отопление, кондиционирование воздуха, безопасность и другие.

Программный комплекс предоставляет следующие ключевые возможности:

* прошивка координатора и конечных устройств, необходимая для обеспечения их совместимости с определенным протоколом;
* автоматизация управления устройствами;
* мониторинг состояния системы и устройств;
* дистанционное управление через мобильное приложение.

Целевая аудитория включает владельцев частных домов, квартир и офисов, а также строительные компании и разработчиков жилых комплексов, желающих предложить своим клиентам современные решения в области «умного дома».

Программный комплекс поддерживает DIY (сделай сам) системы, основанные на протоколе Zigbee и взаимодействующие с помощью MQTT, что является ключевым отличием от многих других компаний на рынке. В отличие от закрытых систем, которые предлагают свои собственные проприетарные решения, разрабатываемое мобильное приложение позволяет пользователям создавать и модифицировать свои собственные «умные системы», используя открытые стандарты и протоколы.

* 1. Расчет затрат на разработку программного комплекса  
     1. **Расчет затрат на основную заработную плату разработчиков**

Для расчета основной заработной платы разработчиков использовалась формула 7.1:

|  |  |  |
| --- | --- | --- |
|  |  | (7.1) |

где n – количество исполнителей, занятых разработкой конкретного ПО;

Кпр – коэффициент, учитывающий процент премий;

Зч.i – часовая заработная плата i-го исполнителя, р.;

ti – трудоемкость работ, выполняемых i-м исполнителем, ч.

Размер месячной заработной платы исполнителя каждой категории соответствует сложившемуся на рынке труда размеру заработной платы для категорий работников, участвующих в разработке [22].

Часовая заработная плата рассчитывалась путем деления месячной заработной платы на 168 рабочих часов в месяце. Размер премии был выбран равным 20% от размера основной заработной платы.

Расчет затрат на основную заработную плату представлен в таблице 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату разработчиков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Месячный оклад, р. | Часовой оклад, р. | Трудоемкость работ, р. | Итого, р. |
| Инженер-программист | 2 299,34 | 13,68 | 184 | 2 517,12 |
| Тестировщик | 1 478,15 | 8,79 | 48 | 421,92 |
| Итог | | | | 2 939,04 |
| Премия (0%) | | | | 0 |
| Всего основная заработная плата | | | | 2 939,04 |

* + 1. **Расчет затрат на дополнительную заработную плату разработчиков**

Расчет дополнительных выплат, предусмотренных законодательством о труде, осуществлялся по формуле 7.2:

|  |  |  |
| --- | --- | --- |
|  |  | (7.2) |

где Зо – затраты на основную заработную плату, р.;

– норматив дополнительной заработной платы.

Норматив дополнительной заработной платы был принят равным 15%.

Размер дополнительной заработной составил:

* + 1. **Расчет отчислений на социальные нужды**

Расчет производился в соответствии с действующими законодательными актами по формуле 7.3:

|  |  |  |
| --- | --- | --- |
|  |  | (7.3) |

где Нсоц – норматив отчислений от фонда оплаты труда.

Норматив отчислений от фонда оплаты труда дополнительной заработной платы был принят равным 35%.

Размер дополнительной заработной платы составил:

* + 1. **Расчет прочих расходов**

Прочие расходов были рассчитаны по формуле 7.4:

|  |  |  |
| --- | --- | --- |
|  |  | (7.4) |

где Нпр – норматив прочих расходов.

Норматив прочих расходов был принят равным 30%.

Размер прочих расходов составил:

Полная сумма затрат на разработку программного средства представлена в таблице 7.2.

Таблица 7.2 – Полная сумма затрат на разработку программного средства

|  |  |
| --- | --- |
| Наименование статьи затрат | Сумма, р. |
| Основная заработная плата разработчиков | 2 939,04 |
| Дополнительная заработная плата разработчиков | 440,86 |
| Отчисления на социальные нужды | 1 182,97 |
| Прочие расходы | 881,71 |
| Общая сумма инвестиций (затрат) на разработку | 5 444,58 |

* 1. Расчет экономического эффекта от реализации программного комплекса на рынке

Для расчёта экономического эффекта организации-разработчика программного средства, а именно чистой прибыли, необходимо знать такие параметры как объем продаж, цену реализации и затраты на разработку.

Для оценки экономического эффекта программного продукта, предположим, что на рынке мобильных устройств в Беларуси доля пользователей Android составляет около 65% [23].

Исходя из общего числа пользователей устройств Android в Беларуси, можно ожидать, что около 20 000 человек будут заинтересованы в данном приложении. Из них, предположим, 10 000 человек установят приложение, а 2000 приобретут расширенную версию.

С учетом цены на расширенную версию приложения, которая составляет 5 долларов США, и с учетом обменного курса доллара к белорусскому рублю, цена одной копии программного средства составит примерно 16,32 белорусских рубля.

Для расчёта прироста чистой прибыли необходимо учесть налог на добавленную стоимость, который высчитывается по следующей формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.5) |

где N – количество копий(лицензий) программного продукта, реализуемое за год, шт.;

– отпускная цена копии программного средства, р.;

– количество приобретённых лицензий;

– ставка налога на добавленную стоимость, %.

Ставка налога на добавленную стоимость по состоянию на 15 апреля 2024 года, в соответствии с действующим законодательством Республики Беларусь, составляет 20%. Используя данное значение, посчитаем НДС:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Посчитав налог на добавленную стоимость, можно рассчитать прирост чистой прибыли, которую получит разработчик от продажи программного продукта. Для этого используется формулу:

|  |  |  |
| --- | --- | --- |
|  |  | (7.6) |

где *N* – количество копий(лицензий) программного продукта, реализуемое за год, шт.;

Цотп – отпускная цена копии программного средства, р.;

– сумма налога на добавленную стоимость, р.;

Нп – ставка налога на прибыль, %;

– рентабельность продаж копий.

Ставка налога на прибыль, согласно действующему законодательству, по состоянию на 14.04.2024 равна 20%. Рентабельность продаж копий взята в размере 30%. Зная ставку налога и рентабельность продаж копий (лицензий), рассчитывается прирост чистой прибыли для разработчика:

|  |  |  |
| --- | --- | --- |
|  |  |  |

* 1. Расчет показателей экономической эффективности разработки и реализации программного средства на рынке

Для того, чтобы оценить экономическую эффективность разработки и реализации программного средства на рынке, необходимо рассмотреть результат сравнения затрат на разработку данного программного продукта, а также полученный прирост чистой прибыли за год.

Сумма затрат на разработку меньше суммы годового экономического эффекта, поэтому можно сделать вывод, что такие инвестиции окупятся менее, чем за один год.

Таким образом, оценка экономической эффективности инвестиций производится при помощи расчёта рентабельности инвестиций (Return on Investment, ROI). Формула для расчёта ROI:

|  |  |  |
| --- | --- | --- |
|  |  | ((7.7) |

где – прирост чистой прибыли, полученной от реализации программного средства на рынке информационных технологий, р.;

Зр – затраты на разработку и реализацию программного средства, р.

|  |  |  |
| --- | --- | --- |
|  |  |  |

* 1. Вывод об экономической целесообразности реализации проектного решения

Проведенные расчеты технико-экономического обоснования позволяют сделать предварительный вывод о целесообразности разработки программного продукта для умного дома. Общая сумма затрат на его разработку и реализацию составила 5 444,58 белорусских рублей, а отпускная цена установлена на уровне 16,32 белорусских рублей.

Прогнозируемый прирост чистой прибыли за год, основанный на предполагаемом объеме продаж в размере 2000 расширенных версий в год, составляет 6528 белорусских рублей. Рентабельность инвестиций за год оценивается в 19,89%.

Такие результаты говорят о том, что разработка данного программного продукта является перспективной и имеет экономическое обоснование. Однако, необходимо учитывать возможные риски, связанные с конкуренцией на рынке и возможным недооцениванием продукта со стороны потребителей.

Высокая рентабельность инвестиций связана с определенными рисками, поэтому важно принимать меры по их минимизации и обеспечению успешного продвижения продукта на рынке. Поддержка проекта и эффективное продвижение могут увеличить его успех и прибыльность.

## ЗАКЛЮЧЕНИЕ

В процессе разработки дипломного проекта была разработана гибкая и простая в использовании система «умного дома», которая ориентирована на потребности различных категорий пользователей. Это представляет собой актуальное и перспективное направление в современной промышленности.

В ходе выполнения проекта были достигнуты следующие результаты:

* разработана архитектура системы умного дома, учитывающая разнообразие функциональных возможностей и потребностей пользователей;
* прошиты и настроены устройства сети, обеспечивающие взаимодействие с приложением и друг с другом;
* создано приложение для управления системой умного дома, обладающее удобным интерфейсом и расширенным функционалом;
* проведено тестирование и отладка приложения, что позволило обнаружить и устранить возможные ошибки.

Результаты работы могут быть применены в различных областях, включая домашнее хозяйство, офисные помещения, общественные учреждения и промышленные объекты. Гибкая и доступная система «умного дома» способна значительно упростить повседневную жизнь пользователей, повысить уровень комфорта, безопасности и энергоэффективности.

В дальнейшем планируется продолжить развитие проекта, расширив функциональность системы и оптимизировав её производительность. Также предполагается дальнейшее тестирование и совершенствование приложения, с целью обеспечения максимальной стабильности и удобства использования для конечных пользователей.

В целом, выполнение данного дипломного проекта позволило получить ценный опыт в области разработки систем умного дома, а также подготовило основу для дальнейших исследований и инноваций в этой области.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] «Умный дом» Hite Pro [Электронный ресурс]. – Электронные данные. – Режим доступа: https://hite-pro.by – Дата доступа: 10.03.2024.

[2] «Умный дом» от «Белтелеком» [Электронный ресурс]. – Электронные данные. – Режим доступа: https://beltelecom.by/umnyi-dom-0. – Дата доступа: 10.03.2024.

[4] «Умный дом» Xiaomi [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.mi.com/ru/smart-home/. – Дата доступа: 10.03.2024.

[5] Датчики Яндекса для умного дома [Электронный ресурс]. – Электронные данные. – Режим доступа: https://yandex.ru/alice/smart-home/sensors?utm\_campaign=smart\_home\_description&utm\_medium=facts&utm\_source=serp. – Дата доступа: 10.03.2024.

[6] Протокол MQTT: концептуальное погружение знать [Электронный ресурс]. – Электронные данные. – Режим доступа: https://habr.com/ru/articles/463669/. – Дата доступа: 10.03.2024.

[7] Описание шины I2C [Электронный ресурс]. – Электронные данные. – Режим доступа: https://itt-ltd.com/reference/ref\_i2c.html. – Дата доступа: 12.03.2024.

[8] Беспроводные сети Zigbee. Часть 1 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://habr.com/ru/companies/efo/articles/281048/. – Дата доступа: 12.03.2024.

[9] Android and MQTT: A Simple Guide [Электронный ресурс]. – Электронные данные. – Режим доступа: https://medium.com/swlh/android-and-mqtt-a-simple-guide-cb0cbba1931c. – Дата доступа: 15.03.2024.

[10] VLK DIY Multi Flasher [Электронный ресурс]. – Электронные данные. – Режим доступа: https://zigbee.wiki/books/%D0%BF%D1%80%D0%BE%D1%88%D0%B8%D0%B2%D0%BA%D0%B8/page/vlk-diy-multi-flasher. – Дата доступа: 15.03.2024.

[11] ZESP [Электронный ресурс]. – Электронные данные. – Режим доступа: https://zigbee.wiki/books/%D1%88%D0%BB%D1%8E%D0%B7%D1%8B-%D0%B8-%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D1%8B/page/zesp. – Дата доступа: 15.03.2023.

[12] SLS Zigbee Gateway [Электронный ресурс]. – Электронные данные. – Режим доступа: https://zigbee.wiki/books/%D1%88%D0%BB%D1%8E%D0%B7%D1%8B-%D0%B8-%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D1%8B/page/sls-zigbee-gateway. – Дата доступа: 15.03.2024.

[13] mosquitto.conf man page [Электронный ресурс]. – Электронные данные. – Режим доступа: https://mosquitto.org/man/mosquitto-conf-5.html. – Дата доступа: 16.03.2024.

[14] E72-2G4M20S1E [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.ebyte.com/en/product-view-news.html?id=1002. – Дата доступа: 16.03.2024.

[14] Переопределение методов и свойств [Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com/kotlin/tutorial/4.10.php. – Дата доступа: 16.03.2024.

[16] Пишем прошивку под TI cc2530 на Z-Stack 3.0 для Zigbee реле Sonoff BASICZBR3 с датчиком ds18b20 свойств [Электронный ресурс]. – Электронные данные. – Режим доступа: https://habr.com/ru/companies/iobroker/articles/495926/. – Дата доступа: 17.03.2024.

[17] ESP32 WROOM DevKit v1: распиновка, схема подключения и программирование [Электронный ресурс]. – Электронные данные. – Режим доступа: http://wiki.amperka.ru/products:esp32-wroom-wifi-devkit-v1/. – Дата доступа: 18.03.2024.

[18] Что такое IoT и что о нем следует знать [Электронный ресурс]. – Электронные данные. – Режим доступа: https://habr.com/ru/companies/otus/articles/549550/. – Дата доступа: 18.03.2023.

[19] MQTT | What is MQTT | MQTT in Depth | QoS | FAQs | MQTT Introduction знать [Электронный ресурс]. – Электронные данные. – Режим доступа: https://medium.com/@harshhvm/mqtt-what-is-mqtt-mqtt-in-depth-qos-faqs-mqtt-introduction-810ff477dc83/. – Дата доступа: 18.03.2024.

[20] Универсальный ZigBee модуль с батарейным питанием на основе E18-MS1-PCB [Электронный ресурс]. – Электронные данные. – Режим доступа: https://kasito.ru/universalnyj-zigbee-modul-s-batarejnym-pitaniem-na-osnove-e18-ms1-pcb/. – Дата доступа: 19.03.2024.

[21] Умный дом на ESP32 через протокол MQTT [Электронный ресурс]. – Электронные данные. – Режим доступа: https://voltiq.ru/how-to-use-mqtt-for-smart-home-arduino-esp32/. – Дата доступа: 22.03.2024.

[22] Сколько зарабытвают Андроид-разработчики на приложениях и играх [Электронный ресурс]. – Электронные данные. – Режим доступа: https://istudy.by/blog/skolko-zarabatyvaut-android-razrabotciki-na-prilozeniah-i-igrah-v-2023-godu. – Дата доступа: 15.04.2024.

[23] Digital 2024: Belarus [Электронный ресурс]. – Электронные данные. – Режим доступа: https://datareportal.com/reports/digital-2024-belarus. – Дата доступа: 15.04.2024.

Вычислительные машины, системы и сети: дипломное проектирование

(методическое пособие) [Электронный ресурс] : Минск БГУИР 2019. –

Электронные данные. – Режим доступа: https://www.bsuir.by/ m/12\_100229\_1\_136308.pdf – Дата доступа: 26.03.2024.

Экономика проектных решений: методические указания по экономическому обоснованию дипломных проектов [Электронный ресурс] : Минск БГУИР 2021. – Электронные данные. – Режим доступа:

https://www.bsuir.by/m/12\_100229\_1\_161144.pdf – Дата доступа: 10.04.2024.

## ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

0001. Файл AddDevice.java

0002.

0003. package com.example.diplomaapp;

0004.

0005. import android.content.ContentResolver;

0006. import android.content.Context;

0007. import android.content.ContextWrapper;

0008. import android.content.Intent;

0009. import android.graphics.Bitmap;

0010. import android.graphics.BitmapFactory;

0011. import android.net.Uri;

0012. import android.os.Bundle;

0013. import android.util.Log;

0014. import android.view.View;

0015. import android.webkit.MimeTypeMap;

0016. import android.widget.Button;

0017. import android.widget.ImageView;

0018. import android.widget.Toast;

0019.

0020. import androidx.activity.EdgeToEdge;

0021. import androidx.appcompat.app.AppCompatActivity;

0022. import androidx.core.graphics.Insets;

0023. import androidx.core.view.ViewCompat;

0024. import androidx.core.view.WindowInsetsCompat;

0025.

0026. import com.example.diplomaapp.dataClasses.DBHelper;

0027. import com.example.diplomaapp.dataClasses.Devices;

0028. import com.example.diplomaapp.dataClasses.Storage;

0029. import com.example.diplomaapp.dataClasses.System;

0030. import com.example.diplomaapp.databinding.ActivityAddDeviceBinding;

0031.

0032. import java.io.File;

0033. import java.io.FileNotFoundException;

0034. import java.io.FileOutputStream;

0035. import java.io.IOException;

0036. import java.io.InputStream;

0037. import java.util.Objects;

0038. import java.util.UUID;

0039.

0040. public class AddDevice extends AppCompatActivity {

0041.

0042. private static final int RESULT\_LOAD\_IMG = 5164654;

0043. private DBHelper dbHelper;

0044. private String img = "";

0045. private ActivityAddDeviceBinding binding;

0046.

0047. @Override

0048. protected void onCreate(Bundle savedInstanceState) {

0049. super.onCreate(savedInstanceState);

0050. EdgeToEdge.enable(this);

0051. setContentView(R.layout.activity\_add\_device);

0052.

0053. binding = ActivityAddDeviceBinding.inflate(getLayoutInflater());

0054. setContentView(binding.getRoot());

0055.

0056. if(Storage.device != null)

0057. {

0058. binding.textInputFriendlyName.setText(Storage.device.getFriendlyName());

0059. binding.textInputMqttPrefix.setText(Storage.device.getMqttPrefix());

0060. binding.textInputDeviceIeeeid.setText(Storage.device.getDeviceId());

0061. binding.textInputDeviceType.setText(Storage.device.getType());

0062. if(!Objects.equals(Storage.device.getDiodeChannel(), ""))

0063. binding.textInputDeviceSwitchChannel.setText(Storage.device.getDiodeChannel());

0064. if (Storage.device.getImgPath() != null){

0065. if (!Storage.device.getImgPath().isEmpty()) {

0066. try {

0067. File file = new File(Storage.device.getImgPath());

0068. if (file.exists()) {

0069. Bitmap bitmap = BitmapFactory.decodeFile(file.getAbsolutePath());

0070. ImageView imgView = findViewById(R.id.imageViewDeviceAdd);

0071. imgView.setImageBitmap(bitmap);

0072. }

0073. } catch (Exception e) {

0074. e.printStackTrace();

0075. }

0076. }

0077. }

0078. }

0079.

0080. Button buttonUpload = findViewById(R.id.buttonChangeImg);

0081. Button buttonSaveDevice = findViewById(R.id.buttonSaveDevice);

0082. buttonUpload.setOnClickListener(new View.OnClickListener() {

0083. @Override

0084. public void onClick(View v) {

0085. Intent photoPickerIntent = new Intent(Intent.ACTION\_PICK);

0086. photoPickerIntent.setType("image/\*");

0087. startActivityForResult(photoPickerIntent, RESULT\_LOAD\_IMG);

0088. }

0089. });

0090.

0091. buttonSaveDevice.setOnClickListener(new View.OnClickListener() {

0092. @Override

0093. public void onClick(View v) {

0094. boolean isFriendlyName = Objects.requireNonNull(binding.textInputFriendlyName.getText()).toString().isEmpty();

0095. boolean isDeviceIeeeid = Objects.requireNonNull(binding.textInputDeviceIeeeid.getText()).toString().isEmpty();

0096. boolean isDeviceType = Objects.requireNonNull(binding.textInputDeviceType.getText()).toString().isEmpty();

0097. boolean isMqttPrefix = Objects.requireNonNull(binding.textInputMqttPrefix.getText()).toString().isEmpty();

0098. boolean isDiodeChannel = Objects.requireNonNull(binding.textInputDeviceSwitchChannel.getText()).toString().isEmpty();

0099.

0100. if(!isFriendlyName && !isDeviceIeeeid && !isDeviceType && !isMqttPrefix)

0101. {

0102. dbHelper = new DBHelper(getApplicationContext());

0103. String deviceIeeeid = binding.textInputDeviceIeeeid.getText().toString();

0104.

0105. if(Storage.device == null || !Objects.equals(Storage.device.getDeviceId(), deviceIeeeid)){

0106. if(dbHelper.isDeviceExists(deviceIeeeid))

0107. Toast.makeText(getApplicationContext(), "This device id is already exists", Toast.LENGTH\_LONG).show();

0108. else saveDevice(deviceIeeeid, isDiodeChannel);

0109. }else saveDevice(deviceIeeeid, isDiodeChannel);

0110.

0111. }

0112. else Toast.makeText(getApplicationContext(), "All fields should be filled", Toast.LENGTH\_LONG).show();

0113. }

0114. });

0115.

0116. ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {

0117. Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());

0118. v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);

0119. return insets;

0120. });

0121. }

0122.

0123. private void saveDevice(String deviceIeeeid, Boolean isDiodeChannel){

0124. Intent intent = getIntent();

0125.

0126. String systemName = intent.getStringExtra("systemName");

0127. String mqttUrl = intent.getStringExtra("mqttUrl");

0128.

0129. System system = new System(systemName, mqttUrl);

0130.

0131. String friendlyName = binding.textInputFriendlyName.getText().toString();

0132.

0133. String deviceType = binding.textInputDeviceType.getText().toString();

0134. String deviceMqttPrefix = binding.textInputMqttPrefix.getText().toString();

0135.

0136. Devices device;

0137.

0138. if(Storage.device != null)

0139. if(!Objects.equals(img, ""))

0140. device = new Devices(deviceIeeeid, deviceType, img);

0141. else device = new Devices(deviceIeeeid, deviceType, Storage.device.getImgPath());

0142. else device = new Devices(deviceIeeeid, deviceType, img);

0143.

0144. device.setFriendlyName(friendlyName);

0145. device.setMqttPrefix(deviceMqttPrefix);

0146. if(!isDiodeChannel)

0147. {

0148. String diodeChannel = binding.textInputDeviceSwitchChannel.getText().toString();

0149. device.setDiodeChannel(diodeChannel);

0150. }

0151. if(Storage.device != null){

0152. dbHelper.updateDevice(device, Storage.device);

0153. Toast.makeText(getApplicationContext(), "Successfully updated device!", Toast.LENGTH\_LONG).show();

0154. }else{

0155. dbHelper.addDevice(device, system);

0156. Toast.makeText(getApplicationContext(), "Successfully added device!", Toast.LENGTH\_LONG).show();

0157. }

0158. dbHelper.close();

0159. finish();

0160. }

0161.

0162. @Override

0163. protected void onActivityResult(int reqCode, int resultCode, Intent data) {

0164. super.onActivityResult(reqCode, resultCode, data);

0165. if (resultCode == RESULT\_OK) {

0166. try {

0167. final Uri imageUri = data.getData();

0168. final InputStream imageStream = getContentResolver().openInputStream(imageUri);

0169. final Bitmap selectedImage = BitmapFactory.decodeStream(imageStream);

0170.

0171. // Сохраняем изображение в файловой системе устройства

0172. img = saveImageToInternalStorage(selectedImage, imageUri);

0173. Log.i("sql img", img);

0174. // Загружаем изображение в ImageView

0175. ImageView imgView = findViewById(R.id.imageViewDeviceAdd);

0176. imgView.setImageBitmap(selectedImage);

0177. } catch (FileNotFoundException e) {

0178. e.printStackTrace();

0179. Toast.makeText(this, "Something went wrong", Toast.LENGTH\_LONG).show();

0180. }

0181.

0182. }else {

0183. Toast.makeText(this, "You haven't picked Image",Toast.LENGTH\_LONG).show();

0184. }

0185. }

0186.

0187. private String saveImageToInternalStorage(Bitmap bitmapImage, Uri imageUri){

0188. ContextWrapper cw = new ContextWrapper(getApplicationContext());

0189. File directory = cw.getDir("imageDir", Context.MODE\_PRIVATE);

0190. String uniqueFileName = "profile\_" + UUID.randomUUID().toString();

0191.

0192. // Извлекаем расширение из URI

0193. String extension = getExtensionFromUri(imageUri);

0194. if (extension != null && !extension.isEmpty()) {

0195. uniqueFileName += "." + extension;

0196. } else {

0197. // Если не удалось извлечь расширение, используем JPEG по умолчанию

0198. uniqueFileName += ".jpg";

0199. }

0200.

0201. File mypath = new File(directory, uniqueFileName);

0202.

0203. FileOutputStream fos = null;

0204. try {

0205. fos = new FileOutputStream(mypath);

0206. // Сжимаем изображение в соответствии с расширением

0207. if (extension != null && extension.equalsIgnoreCase("png")) {

0208. bitmapImage.compress(Bitmap.CompressFormat.PNG, 100, fos);

0209. } else {

0210. bitmapImage.compress(Bitmap.CompressFormat.JPEG, 100, fos);

0211. }

0212. } catch (Exception e) {

0213. e.printStackTrace();

0214. } finally {

0215. try {

0216. if (fos != null) {

0217. fos.close();

0218. }

0219. } catch (IOException e) {

0220. e.printStackTrace();

0221. }

0222. }

0223. return directory.getAbsolutePath() + "/" + uniqueFileName;

0224. }

0225.

0226. private String getExtensionFromUri(Uri uri) {

0227. ContentResolver contentResolver = getContentResolver();

0228. MimeTypeMap mimeTypeMap = MimeTypeMap.getSingleton();

0229. String extension;

0230.

0231. if (contentResolver == null || uri == null) {

0232. return null;

0233. }

0234.

0235. String mimeType = contentResolver.getType(uri);

0236. if (mimeType == null) {

0237. String uriString = uri.toString();

0238. int extensionIndex = uriString.lastIndexOf('.');

0239. if (extensionIndex != -1 && extensionIndex < uriString.length() - 1) {

0240. extension = uriString.substring(extensionIndex + 1);

0241. return extension.toLowerCase();

0242. }

0243. return null;

0244. }

0245.

0246. // Получаем расширение из MIME типа

0247. extension = mimeTypeMap.getExtensionFromMimeType(mimeType);

0248. return extension;

0249. }

0250.

0251. @Override

0252. protected void onDestroy() {

0253. Storage.device = null;

0254. binding = null;

0255. super.onDestroy();

0256. }

0257.

0258. }

0259.

0260. Файл DevicesActivity.java

0261.

0262. package com.example.diplomaapp;

0263.

0264. import android.annotation.SuppressLint;

0265. import android.content.Context;

0266. import android.content.Intent;

0267. import android.net.ConnectivityManager;

0268. import android.net.Network;

0269. import android.net.NetworkCapabilities;

0270. import android.os.Bundle;

0271. import android.util.Log;

0272. import android.view.View;

0273. import android.widget.ImageButton;

0274. import android.widget.ProgressBar;

0275. import android.widget.TextView;

0276. import android.widget.Toast;

0277.

0278. import androidx.activity.EdgeToEdge;

0279. import androidx.appcompat.app.AppCompatActivity;

0280. import androidx.constraintlayout.widget.ConstraintLayout;

0281. import androidx.core.graphics.Insets;

0282. import androidx.core.view.ViewCompat;

0283. import androidx.core.view.WindowInsetsCompat;

0284. import androidx.recyclerview.widget.ItemTouchHelper;

0285. import androidx.recyclerview.widget.LinearLayoutManager;

0286. import androidx.recyclerview.widget.RecyclerView;

0287.

0288. import com.example.diplomaapp.dataClasses.DBHelper;

0289. import com.example.diplomaapp.dataClasses.DeleteConfirmationDialog;

0290. import com.example.diplomaapp.dataClasses.Devices;

0291. import com.example.diplomaapp.dataClasses.DeviceAdapter;

0292. import com.example.diplomaapp.dataClasses.NotificationHelper;

0293. import com.example.diplomaapp.dataClasses.Storage;

0294. import com.example.diplomaapp.dataClasses.SwipeToDeleteCallback;

0295. import com.example.diplomaapp.dataClasses.System;

0296. import com.example.diplomaapp.databinding.ActivityDevicesBinding;

0297. import com.google.android.material.floatingactionbutton.FloatingActionButton;

0298.

0299. import org.eclipse.paho.android.service.MqttAndroidClient;

0300. import org.eclipse.paho.client.mqttv3.IMqttActionListener;

0301. import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;

0302. import org.eclipse.paho.client.mqttv3.IMqttToken;

0303. import org.eclipse.paho.client.mqttv3.MqttCallback;

0304. import org.eclipse.paho.client.mqttv3.MqttClient;

0305. import org.eclipse.paho.client.mqttv3.MqttConnectOptions;

0306. import org.eclipse.paho.client.mqttv3.MqttMessage;

0307. import org.json.JSONException;

0308. import org.json.JSONObject;

0309.

0310. import java.util.ArrayList;

0311. import java.util.Iterator;

0312.

0313. public class DevicesActivity extends AppCompatActivity implements SwipeToDeleteCallback.OnSwipeLeftListener, SwipeToDeleteCallback.OnSwipeRightListener{

0314.

0315. private RecyclerView mRecyclerView;

0316. private DeviceAdapter adapter;

0317. private ArrayList<Devices> devices;

0318. private System system;

0319. private DBHelper dbHelper;

0320. private MqttAndroidClient mqttAndroidClient;

0321.

0322. @Override

0323. protected void onCreate(Bundle savedInstanceState) {

0324. super.onCreate(savedInstanceState);

0325. EdgeToEdge.enable(this);

0326. setContentView(R.layout.activity\_devices);

0327. dbHelper = new DBHelper(this);

0328. Intent intent = getIntent();

0329.

0330. system = new System(intent.getStringExtra("systemName"), intent.getStringExtra("mqttUrl"));

0331.

0332. if (intent.hasExtra("mqtt\_login") && intent.hasExtra("mqtt\_password")) {

0333. system.setMqtt\_login(intent.getStringExtra("mqtt\_login"));

0334. system.setMqtt\_password(intent.getStringExtra("mqtt\_password"));

0335. }

0336.

0337. connectToMqtt();

0338.

0339. setListeners();

0340.

0341. ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.mainlayout), (v, insets) -> {

0342. Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());

0343. v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);

0344. return insets;

0345. });

0346. }

0347.

0348. private void initCardView() {

0349. setDevicesList();

0350. SwipeToDeleteCallback callback = new SwipeToDeleteCallback(this, this, this);

0351. ItemTouchHelper itemTouchHelper = new ItemTouchHelper(callback);

0352. itemTouchHelper.attachToRecyclerView(mRecyclerView);

0353. }

0354.

0355. @SuppressLint("NotifyDataSetChanged")

0356. private void updateDataForRecycler() {

0357. TextView textViewNoDevices = findViewById(R.id.textViewNoDevices);

0358. if(!devices.isEmpty())

0359. textViewNoDevices.setVisibility(View.GONE);

0360. else

0361. textViewNoDevices.setVisibility(View.VISIBLE);

0362.

0363. adapter.notifyDataSetChanged();

0364. }

0365.

0366. private boolean isInternetConnection(){

0367. ConnectivityManager connectivityManager = (ConnectivityManager) getApplicationContext().getSystemService(Context.CONNECTIVITY\_SERVICE);

0368. if(connectivityManager != null) {

0369. for (Network net : connectivityManager.getAllNetworks()) {

0370. NetworkCapabilities nc = connectivityManager.getNetworkCapabilities(net);

0371. if (nc != null && ((nc.hasTransport(NetworkCapabilities.TRANSPORT\_WIFI)

0372. || nc.hasTransport(NetworkCapabilities.TRANSPORT\_CELLULAR)) // Добавляем проверку на мобильный интернет

0373. && nc.hasCapability(NetworkCapabilities.NET\_CAPABILITY\_INTERNET)))

0374. return true;

0375. }

0376. }

0377. return false;

0378. }

0379.

0380.

0381. private void connectToMqtt(){

0382. ProgressBar loadingSpinner = findViewById(R.id.progressBar);

0383. ConstraintLayout constraintLayout = findViewById(R.id.conLayout);

0384. FloatingActionButton addDevice = findViewById(R.id.addDevice);

0385. TextView textViewNoDevices = findViewById(R.id.textViewNoDevices);

0386.

0387. textViewNoDevices.setVisibility(View.GONE);

0388. loadingSpinner.setVisibility(View.VISIBLE);

0389. constraintLayout.setVisibility(View.GONE);

0390. addDevice.setVisibility(View.INVISIBLE);

0391.

0392. if(isInternetConnection()) {

0393. try {

0394. String clientId = MqttClient.generateClientId();

0395. mqttAndroidClient = new MqttAndroidClient(getApplicationContext(), system.getMqtt\_url(), clientId);

0396.

0397. mqttAndroidClient.setCallback(new MqttCallback() {

0398.

0399. @Override

0400. public void connectionLost(Throwable cause) {}

0401.

0402. @Override

0403. public void messageArrived(String topic, MqttMessage message) throws Exception {

0404. String[] parts = topic.split("/");

0405. String deviceId = parts[parts.length - 1];

0406. Devices dev = new Devices(deviceId, "smth", "");

0407. String data = parseMqttMessage(message.toString());

0408. dbHelper.updateLastDataForDevice(dev, data);

0409. Devices dv = dbHelper.getDeviceByDeviceId(deviceId);

0410. setDevicesList();

0411. NotificationHelper.showNotification(getApplicationContext(), dv.getFriendlyName() + " (" + dv.getDeviceId() + ")", data, dbHelper.getIdByDeviceId(deviceId));

0412. }

0413.

0414. @Override

0415. public void deliveryComplete(IMqttDeliveryToken token) {

0416.

0417. }

0418. });

0419.

0420. MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();

0421. mqttConnectOptions.setAutomaticReconnect(true);

0422. mqttConnectOptions.setCleanSession(false);

0423.

0424. if(system.getMqtt\_login() != null && system.getMqtt\_password() != null)

0425. {

0426. mqttConnectOptions.setUserName(system.getMqtt\_login());

0427. mqttConnectOptions.setPassword(system.getMqtt\_password().toCharArray());

0428. }

0429.

0430. try {

0431. mqttAndroidClient.connect(mqttConnectOptions, null, new IMqttActionListener() {

0432. @Override

0433. public void onSuccess(IMqttToken asyncActionToken) {

0434. loadingSpinner.setVisibility(View.INVISIBLE);

0435. constraintLayout.setVisibility(View.VISIBLE);

0436. addDevice.setVisibility(View.VISIBLE);

0437.

0438. Toast.makeText(getApplicationContext(), "Mqtt connection", Toast.LENGTH\_LONG).show();

0439. initCardView();

0440.

0441. for (Devices device : devices) {

0442. String prefixMqtt = device.getMqttPrefix();

0443. String deviceId = device.getDeviceId();

0444.

0445. subscribeToTopic(prefixMqtt + "/" + deviceId);

0446. }

0447. }

0448.

0449. @Override

0450. public void onFailure(IMqttToken asyncActionToken, Throwable exception) {

0451. Toast.makeText(getApplicationContext(), "Failed to connect", Toast.LENGTH\_LONG).show();

0452.

0453. }

0454. });

0455. } catch (Exception ex) {

0456. ex.printStackTrace();

0457. Log.i("MQTT", ex.toString());

0458. }

0459.

0460. } catch (Exception e) {

0461. Log.i("mqtt", e.getMessage());

0462. }

0463. }else {

0464. Toast.makeText(getApplicationContext(), "No internet connection", Toast.LENGTH\_LONG).show();

0465. }

0466. }

0467.

0468. public void setListeners(){

0469. ImageButton refreshButton = findViewById(R.id.buttonRefreshDevices);

0470. refreshButton.setBackground(null);

0471. refreshButton.setOnClickListener(new View.OnClickListener() {

0472. @Override

0473. public void onClick(View v) {

0474. setDevicesList();

0475. for (Devices device : devices) {

0476. String prefixMqtt = device.getMqttPrefix();

0477. String deviceId = device.getDeviceId();

0478.

0479. subscribeToTopic(prefixMqtt + "/" + deviceId);

0480. }

0481. Toast.makeText(getApplicationContext(), "Refreshed", Toast.LENGTH\_SHORT).show();

0482. }

0483. });

0484.

0485. FloatingActionButton addDeviceButton = findViewById(R.id.addDevice);

0486. addDeviceButton.setOnClickListener(v -> {

0487. Intent intent = new Intent(".AddDevice").setClassName(getPackageName(), "com.example.diplomaapp.AddDevice");

0488. intent.putExtra("systemName", system.getSystemName());

0489. intent.putExtra("mqttUrl", system.getMqtt\_url());

0490. startActivity(intent);

0491. });

0492. }

0493.

0494. private void setDevicesList(){

0495. mRecyclerView = findViewById(R.id.recycler);

0496. mRecyclerView.setLayoutManager(new LinearLayoutManager((this)));

0497. devices = dbHelper.getAllDevices(system);

0498. adapter = new DeviceAdapter(this, devices, mqttAndroidClient);

0499. mRecyclerView.setAdapter(adapter);

0500. updateDataForRecycler();

0501. }

0502.

0503. public void subscribeToTopic(String topic) {

0504. try {

0505. mqttAndroidClient.subscribe(topic, 0, null, new IMqttActionListener() {

0506. @Override

0507. public void onSuccess(IMqttToken asyncActionToken) {

0508. Log.i("MQTT", "CONNECTED TO TOPIC");

0509. }

0510.

0511. @Override

0512. public void onFailure(IMqttToken asyncActionToken, Throwable exception) {

0513. Log.i("MQTT", "NOT CONNECTED TO TOPIC");

0514. }

0515. });

0516. } catch (Exception ex) {

0517. Log.i("MQTT", ex.toString());

0518. ex.printStackTrace();

0519. }

0520. }

0521.

0522. public void unSubscribeToTopic(String topic) {

0523. try {

0524. mqttAndroidClient.unsubscribe(topic);

0525. } catch (Exception ex) {

0526. Log.i("MQTT", ex.toString());

0527. ex.printStackTrace();

0528. }

0529. }

0530.

0531. public void publishMessage(String topic, String message) {

0532. try {

0533. MqttMessage mqttMessage = new MqttMessage();

0534. mqttMessage.setPayload(message.getBytes());

0535. mqttAndroidClient.publish(topic, mqttMessage);

0536. } catch (Exception ex) {

0537. ex.printStackTrace();

0538. }

0539. }

0540.

0541. public String parseMqttMessage(String jsonMessage){

0542. StringBuilder formattedMessage = new StringBuilder();

0543. try {

0544. JSONObject jsonObject = new JSONObject(jsonMessage);

0545.

0546. // Получаем все ключи объекта

0547. Iterator<String> keys = jsonObject.keys();

0548.

0549.

0550. // Проходим по каждому ключу и добавляем его и соответствующее значение в форматированное сообщение

0551. while (keys.hasNext()) {

0552. String key = keys.next();

0553. String value = jsonObject.getString(key);

0554.

0555. formattedMessage.append(key).append(": ").append(value).append("\n");

0556. }

0557. } catch (JSONException e) {

0558. e.printStackTrace();

0559. }

0560. return formattedMessage.toString();

0561. }

0562.

0563. @Override

0564. protected void onDestroy() {

0565. try {

0566. mqttAndroidClient.disconnect();

0567. } catch (Exception ex) {

0568. ex.printStackTrace();

0569. }

0570. super.onDestroy();

0571. }

0572.

0573. @Override

0574. public void onSwipeLeft(int position) {

0575. DeleteConfirmationDialog.show(this, "Are you sure you want to remove this item?", () -> {

0576. Devices dev = devices.get(position);

0577. dbHelper.deleteDevice(dev);

0578. devices.remove(position);

0579. setDevicesList();

0580. Toast.makeText(getApplicationContext(), "System deleted", Toast.LENGTH\_SHORT).show();

0581. }, () -> {

0582. setDevicesList();

0583. Toast.makeText(getApplicationContext(), "Canceled", Toast.LENGTH\_SHORT).show();

0584. });

0585. }

0586.

0587. @Override

0588. public void onSwipeRight(int position) {

0589. Intent intent = new Intent(getApplicationContext(), AddDevice.class);

0590. Storage.device = devices.get(position);

0591. startActivity(intent);

0592.

0593. for (Devices device : devices) {

0594. String prefixMqtt = device.getMqttPrefix();

0595. String deviceId = device.getDeviceId();

0596.

0597. unSubscribeToTopic(prefixMqtt + "/" + deviceId);

0598. }

0599. setDevicesList();

0600.

0601. for (Devices device : devices) {

0602. String prefixMqtt = device.getMqttPrefix();

0603. String deviceId = device.getDeviceId();

0604.

0605. subscribeToTopic(prefixMqtt + "/" + deviceId);

0606. }

0607. }

0608.

0609. }

0610.

0611.

0612. Файл FirstFragment.java

0613.

0614. package com.example.diplomaapp;

0615.

0616. import android.os.Bundle;

0617. import android.text.InputFilter;

0618. import android.util.Log;

0619. import android.view.LayoutInflater;

0620. import android.view.View;

0621. import android.view.ViewGroup;

0622. import android.widget.Toast;

0623.

0624. import androidx.annotation.NonNull;

0625. import androidx.fragment.app.Fragment;

0626. import androidx.navigation.fragment.NavHostFragment;

0627.

0628. import com.example.diplomaapp.dataClasses.DBHelper;

0629. import com.example.diplomaapp.dataClasses.PortFilter;

0630. import com.example.diplomaapp.dataClasses.Storage;

0631. import com.example.diplomaapp.dataClasses.System;

0632. import com.example.diplomaapp.databinding.FragmentFirstBinding;

0633.

0634. import java.net.URI;

0635. import java.net.URISyntaxException;

0636. import java.util.Objects;

0637.

0638. public class FirstFragment extends Fragment {

0639. private FragmentFirstBinding binding;

0640.

0641. private static System system;

0642.

0643. @Override

0644. public View onCreateView(

0645. @NonNull LayoutInflater inflater, ViewGroup container,

0646. Bundle savedInstanceState

0647. ) {

0648. binding = FragmentFirstBinding.inflate(inflater, container, false);

0649.

0650. // InputFilter[] filters = new InputFilter[1];

0651. // filters[0] = new IPAddressFilter();

0652. // binding.editTextInputIP.setFilters(filters);

0653.

0654. InputFilter[] filters = new InputFilter[1];

0655. filters[0] = new PortFilter();

0656. binding.editTextInputPort.setFilters(filters);

0657. return binding.getRoot();

0658. }

0659.

0660. public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {

0661. super.onViewCreated(view, savedInstanceState);

0662. try {

0663. if (Storage.system != null) {

0664. try {

0665. URI uri = new URI(Storage.system.getMqtt\_url());

0666. binding.editTextInputIP.setText(uri.getHost());

0667. binding.editTextInputPort.setText(String.valueOf(uri.getPort()));

0668. } catch (URISyntaxException e) {

0669. e.printStackTrace();

0670. }

0671. binding.editTextInputLogin.setText(Storage.system.getMqtt\_login());

0672. binding.editTextPassword.setText(Storage.system.getMqtt\_password());

0673. }

0674. }catch (Exception e){

0675. e.printStackTrace();

0676. }

0677.

0678. binding.buttonFirst.setOnClickListener(v ->{

0679. Bundle bndl = validateData();

0680. if(bndl != null)

0681. NavHostFragment.findNavController(FirstFragment.this)

0682. .navigate(R.id.action\_FirstFragment\_to\_SecondFragment, bndl);

0683. }

0684. );

0685. binding.buttonSecond.setOnClickListener(v ->{

0686. Storage.system = null;

0687. NavHostFragment.findNavController(FirstFragment.this)

0688. .navigate(R.id.action\_FirstFragment\_to\_mainActivity);

0689. }

0690. );

0691. }

0692.

0693. public Bundle validateData(){

0694. Bundle bundle = new Bundle();

0695.

0696. if(binding.editTextInputIP.getText().toString().isEmpty() || Objects.requireNonNull(binding.editTextInputPort.getText()).toString().isEmpty()) {

0697. Toast.makeText(requireContext(), "Required IP Address and port", Toast.LENGTH\_LONG).show();

0698. return null;

0699. }

0700. else {

0701. String ipAddress = binding.editTextInputIP.getText().toString();

0702. String port = binding.editTextInputPort.getText().toString();

0703. if (isValidPort(port)) {

0704. Boolean loginEmpty = binding.editTextInputLogin.getText() == null;

0705. Boolean passwordEmpty = binding.editTextPassword.getText() == null;

0706. if (loginEmpty && passwordEmpty) {

0707. bundle.putString("address", "tcp://" + ipAddress + ":" + port);

0708. return bundle;

0709. } else if (!loginEmpty && !passwordEmpty) {

0710. bundle.putString("address", "tcp://" + ipAddress + ":" + port);

0711. bundle.putString("login", binding.editTextInputLogin.getText().toString());

0712. bundle.putString("password", binding.editTextPassword.getText().toString());

0713. return bundle;

0714. } else {

0715. Toast.makeText(requireContext(), "Required full credentials or none", Toast.LENGTH\_LONG).show();

0716. return null;

0717. }

0718. }

0719. else{

0720. Toast.makeText(requireContext(), "Incorrect port", Toast.LENGTH\_LONG).show();

0721. return null;

0722. }

0723. }

0724. }

0725.

0726. private boolean isValidIPAddress(String ipAddress) {

0727. // Регулярное выражение для проверки IP-адреса

0728. String ipRegex = "^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$";

0729. return ipAddress.matches(ipRegex);

0730. }

0731.

0732. private boolean isValidPort(String port) {

0733. // Регулярное выражение для проверки порта (число от 1 до 65535)

0734. String portRegex = "^([1-9]|[1-9][0-9]{1,3}|[1-5][0-9]{4}|6[0-5]{2}[0-3][0-5])$";

0735. return port.matches(portRegex);

0736. }

0737.

0738. @Override

0739. public void onDestroyView() {

0740. super.onDestroyView();

0741. binding = null;

0742. }

0743.

0744. }

0745.

0746. Файл MainActivity.java

0747.

0748. package com.example.diplomaapp;

0749.

0750. import android.annotation.SuppressLint;

0751. import android.content.Intent;

0752. import android.os.Bundle;

0753. import android.util.Log;

0754. import android.view.View;

0755. import android.view.ViewGroup;

0756. import android.widget.ArrayAdapter;

0757. import android.widget.ImageButton;

0758. import android.widget.ListView;

0759. import android.widget.TextView;

0760. import android.widget.Toast;

0761.

0762. import androidx.activity.EdgeToEdge;

0763. import androidx.annotation.NonNull;

0764. import androidx.annotation.Nullable;

0765. import androidx.appcompat.app.AppCompatActivity;

0766. import androidx.core.graphics.Insets;

0767. import androidx.core.view.ViewCompat;

0768. import androidx.core.view.WindowInsetsCompat;

0769.

0770. import com.example.diplomaapp.dataClasses.DBHelper;

0771. import com.example.diplomaapp.dataClasses.DeleteConfirmationDialog;

0772. import com.example.diplomaapp.dataClasses.DeviceAdapter;

0773. import com.example.diplomaapp.dataClasses.Devices;

0774. import com.example.diplomaapp.dataClasses.NotificationHelper;

0775. import com.example.diplomaapp.dataClasses.Storage;

0776. import com.example.diplomaapp.dataClasses.SwipeToDeleteCallback;

0777. import com.example.diplomaapp.dataClasses.System;

0778. import com.example.diplomaapp.dataClasses.SystemAdapter;

0779. import com.google.android.material.floatingactionbutton.FloatingActionButton;

0780. import androidx.appcompat.widget.Toolbar;

0781. import androidx.recyclerview.widget.ItemTouchHelper;

0782. import androidx.recyclerview.widget.LinearLayoutManager;

0783. import androidx.recyclerview.widget.RecyclerView;

0784.

0785. import java.util.ArrayList;

0786. import java.util.List;

0787.

0788. public class MainActivity extends AppCompatActivity implements SwipeToDeleteCallback.OnSwipeLeftListener, SwipeToDeleteCallback.OnSwipeRightListener {

0789. private DBHelper dbHelper;

0790. private RecyclerView mRecyclerView;

0791. private SystemAdapter adapter;

0792. private List<System> systems;

0793.

0794. @Override

0795. protected void onCreate(Bundle savedInstanceState) {

0796. super.onCreate(savedInstanceState);

0797. EdgeToEdge.enable(this);

0798. setContentView(R.layout.activity\_main);

0799. Storage.device = null;

0800. dbHelper = new DBHelper(this);

0801.

0802. Toolbar toolbar = findViewById(R.id.toolbar);

0803. setSupportActionBar(toolbar);

0804.

0805. ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.mainlayout), (v, insets) -> {

0806. Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());

0807. v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);

0808.

0809. setListeners();

0810. setListItems();

0811.

0812. ItemTouchHelper itemTouchHelper = getItemTouchHelper();

0813. itemTouchHelper.attachToRecyclerView(mRecyclerView);

0814.

0815. return insets;

0816. });

0817. }

0818.

0819. @NonNull

0820. private ItemTouchHelper getItemTouchHelper() {

0821. SwipeToDeleteCallback callback = new SwipeToDeleteCallback(this, this, this);

0822. ItemTouchHelper itemTouchHelper = new ItemTouchHelper(callback);

0823. return itemTouchHelper;

0824. }

0825.

0826. public void setListeners(){

0827. ImageButton refreshButton = findViewById(R.id.buttonRefreshSystems);

0828. refreshButton.setBackground(null);

0829.

0830. refreshButton.setOnClickListener(v -> {

0831. setListItems();

0832. Toast.makeText(getApplicationContext(), "Refreshed", Toast.LENGTH\_SHORT).show();

0833. });

0834.

0835. FloatingActionButton addSystemButton = findViewById(R.id.addSystem);

0836. addSystemButton.setOnClickListener(v -> {

0837. Storage.device = null;

0838. Intent intent = new Intent(".SecondActivity");

0839. startActivity(intent);

0840. setListItems();

0841. });

0842. }

0843.

0844. public void setListItems(){

0845. mRecyclerView = findViewById(R.id.recyclerSystem);

0846. mRecyclerView.setLayoutManager(new LinearLayoutManager((this)));

0847. systems = dbHelper.getAllSystems();

0848. adapter = new SystemAdapter(this, new ArrayList<>(systems));

0849. mRecyclerView.setAdapter(adapter);

0850. updateDataForRecycler();

0851.

0852. // listView.setOnItemClickListener((parent, view, position, id) -> {

0853. // System selectedSystem = systems.get(position);

0854. //

0855. // Intent intent = new Intent(".DevicesActivity");

0856. // intent.putExtra("systemName", selectedSystem.getSystemName());

0857. // intent.putExtra("mqttUrl", selectedSystem.getMqtt\_url());

0858. // if(selectedSystem.getMqtt\_login() != null && selectedSystem.getMqtt\_password() != null){

0859. // intent.putExtra("mqtt\_login", selectedSystem.getMqtt\_login());

0860. // intent.putExtra("mqtt\_password", selectedSystem.getMqtt\_password());

0861. // }

0862. // Log.i("MQTT main", selectedSystem.getMqtt\_url() + selectedSystem.getMqtt\_login() + selectedSystem.getMqtt\_password());

0863. // startActivity(intent);

0864. // setListItems();

0865. // });

0866.

0867. }

0868.

0869. @SuppressLint("NotifyDataSetChanged")

0870. private void updateDataForRecycler() {

0871. TextView textViewNoSystems = findViewById(R.id.textViewNoSystems);

0872.

0873. if (!systems.isEmpty())

0874. textViewNoSystems.setVisibility(View.GONE);

0875. else textViewNoSystems.setVisibility(View.VISIBLE);

0876.

0877. adapter.notifyDataSetChanged();

0878. }

0879.

0880. @Override

0881. public void onSwipeLeft(int position) {

0882. DeleteConfirmationDialog.show(this, "Are you sure you want to remove this item?", () -> {

0883. System sys = systems.get(position);

0884. dbHelper.deleteSystem(sys);

0885. systems.remove(position);

0886. setListItems();

0887. }, this::setListItems);

0888. Toast.makeText(getApplicationContext(), "System deleted", Toast.LENGTH\_SHORT).show();

0889. }

0890.

0891. @Override

0892. public void onSwipeRight(int position) {

0893. Intent intent = getIntentChangeSystem(systems.get(position));

0894. startActivity(intent);

0895. setListItems();

0896. }

0897.

0898. @NonNull

0899. private Intent getIntentChangeSystem(System clickedSystem) {

0900. Intent intent = new Intent(getApplicationContext(), SecondActivity.class);

0901. Storage.system = clickedSystem;

0902. return intent;

0903. }

0904. }

0905.

0906. Файл SecondActivity.java

0907.

0908. package com.example.diplomaapp;

0909.

0910. import static androidx.core.content.ContentProviderCompat.requireContext;

0911.

0912. import android.os.Bundle;

0913.

0914. import androidx.appcompat.app.AppCompatActivity;

0915.

0916. import androidx.navigation.NavController;

0917. import androidx.navigation.Navigation;

0918. import androidx.navigation.ui.AppBarConfiguration;

0919. import androidx.navigation.ui.NavigationUI;

0920.

0921. import com.example.diplomaapp.databinding.ActivitySecondBinding;

0922.

0923. public class SecondActivity extends AppCompatActivity {

0924.

0925. private AppBarConfiguration appBarConfiguration;

0926. private ActivitySecondBinding binding;

0927.

0928. @Override

0929. protected void onCreate(Bundle savedInstanceState) {

0930. super.onCreate(savedInstanceState);

0931.

0932. binding = ActivitySecondBinding.inflate(getLayoutInflater());

0933.

0934. setContentView(binding.getRoot());

0935. setSupportActionBar(binding.toolbar);

0936. NavController navController = Navigation.findNavController(this, R.id.nav\_host\_fragment\_content\_second);

0937. appBarConfiguration = new AppBarConfiguration.Builder(navController.getGraph()).build();

0938. NavigationUI.setupActionBarWithNavController(this, navController, appBarConfiguration);

0939.

0940.

0941. }

0942.

0943. @Override

0944. public boolean onSupportNavigateUp() {

0945. NavController navController = Navigation.findNavController(this, R.id.nav\_host\_fragment\_content\_second);

0946. return NavigationUI.navigateUp(navController, appBarConfiguration)

0947. || super.onSupportNavigateUp();

0948. }

0949. }

0950.

0951. Файл SecondFragment.java

0952.

0953. package com.example.diplomaapp;

0954.

0955. import android.os.Bundle;

0956. import android.util.Log;

0957. import android.view.LayoutInflater;

0958. import android.view.View;

0959. import android.view.ViewGroup;

0960. import android.widget.Toast;

0961.

0962. import androidx.annotation.NonNull;

0963. import androidx.fragment.app.Fragment;

0964. import androidx.navigation.fragment.NavHostFragment;

0965.

0966. import com.example.diplomaapp.dataClasses.DBHelper;

0967. import com.example.diplomaapp.dataClasses.Storage;

0968. import com.example.diplomaapp.dataClasses.System;

0969. import com.example.diplomaapp.databinding.FragmentSecondBinding;

0970.

0971. import java.net.URI;

0972. import java.net.URISyntaxException;

0973. import java.util.Objects;

0974.

0975. public class SecondFragment extends Fragment {

0976.

0977. private FragmentSecondBinding binding;

0978.

0979. private DBHelper dbHelper;

0980.

0981. @Override

0982. public View onCreateView(

0983. @NonNull LayoutInflater inflater, ViewGroup container,

0984. Bundle savedInstanceState

0985. ) {

0986. binding = FragmentSecondBinding.inflate(inflater, container, false);

0987.

0988. return binding.getRoot();

0989. }

0990.

0991. public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {

0992. super.onViewCreated(view, savedInstanceState);

0993. try {

0994. if (Storage.system != null) {

0995. binding.textEditSystemName.setText(Storage.system.getSystemName());

0996. }

0997. }catch (Exception e){

0998. e.printStackTrace();

0999. }

1000. binding.buttonSave.setOnClickListener(v -> {

1001. Boolean nameEmpty = binding.textEditSystemName.getText().toString().isEmpty();

1002. if(!nameEmpty) {

1003. dbHelper = new DBHelper(requireContext());

1004. if(Storage.system == null || !Objects.equals(Storage.system.getSystemName(), binding.textEditSystemName.getText().toString())){

1005. Log.i("sql", String.valueOf(dbHelper.isSystemExists(binding.textEditSystemName.getText().toString())));

1006. if(dbHelper.isSystemExists(binding.textEditSystemName.getText().toString()))

1007. Toast.makeText(requireContext(), "This system name is already exists", Toast.LENGTH\_LONG).show();

1008. else saveSystem();

1009. }else saveSystem();

1010. }else Toast.makeText(requireContext(), "System name field should be filled", Toast.LENGTH\_LONG).show();

1011. });

1012.

1013. }

1014.

1015. private void saveSystem(){

1016. String sysName = binding.textEditSystemName.getText().toString();

1017. Bundle args = getArguments();

1018.

1019. if (args != null) {

1020. String address = "";

1021. String login = "";

1022. String password = "";

1023. address = args.getString("address");

1024. if (args.containsKey("login")) {

1025. login = args.getString("login");

1026. }

1027. if (args.containsKey("password")) {

1028. password = args.getString("password");

1029. }

1030.

1031. System mySys = new System(sysName, address);

1032.

1033. if(!login.isEmpty() && !password.isEmpty())

1034. {

1035. mySys.setMqtt\_login(login);

1036. mySys.setMqtt\_password(password);

1037. }

1038.

1039. Log.i("MQTT add system", address + " " + login + " " + password);

1040.

1041. if(Storage.system != null){

1042. dbHelper.updateSystem(mySys, Storage.system);

1043. Toast.makeText(requireContext(), "Successfully updated system!", Toast.LENGTH\_LONG).show();

1044. }else{

1045. dbHelper.addSystem(mySys);

1046. Toast.makeText(requireContext(), "Successfully added system!", Toast.LENGTH\_LONG).show();

1047. }

1048. NavHostFragment.findNavController(SecondFragment.this)

1049. .navigate(R.id.action\_SecondFragment\_to\_mainActivity);

1050. }

1051. }

1052.

1053. @Override

1054. public void onDestroyView() {

1055. super.onDestroyView();

1056. Storage.system = null;

1057. binding = null;

1058. }

1059.

1060. }

1061.

1062. Файл DBHelper.java

1063.

1064. package com.example.diplomaapp.dataClasses;

1065.

1066. import android.content.ContentValues;

1067. import android.content.Context;

1068. import android.database.Cursor;

1069. import android.database.sqlite.SQLiteDatabase;

1070. import android.database.sqlite.SQLiteOpenHelper;

1071. import android.util.Log;

1072.

1073. import java.util.ArrayList;

1074. import java.util.List;

1075. import java.util.Objects;

1076.

1077. public class DBHelper extends SQLiteOpenHelper {

1078.

1079. private static final String DATABASE\_NAME = "iotDb";

1080. private static final int DATABASE\_VERSION = 1;

1081.

1082. public DBHelper(Context context) {

1083. super(context, DATABASE\_NAME, null, DATABASE\_VERSION);

1084. }

1085.

1086. @Override

1087. public void onCreate(SQLiteDatabase db) {

1088. db.execSQL("CREATE TABLE IF NOT EXISTS systems " +

1089. "(\_id INTEGER PRIMARY KEY AUTOINCREMENT, system\_name TEXT NOT NULL UNIQUE, " +

1090. "mqtt\_url TEXT NOT NULL, mqtt\_login TEXT, mqtt\_password TEXT)");

1091. db.execSQL("CREATE TABLE IF NOT EXISTS devices " +

1092. "(\_id INTEGER PRIMARY KEY AUTOINCREMENT, friendly\_name TEXT, " +

1093. "device\_id TEXT NOT NULL UNIQUE, " +

1094. "device\_type TEXT NOT NULL, last\_data TEXT, " +

1095. "system\_id INTEGER NOT NULL, " +

1096. "img\_path TEXT," +

1097. "mqtt\_prefix Text," +

1098. "diode\_channel Text," +

1099. "FOREIGN KEY (system\_id) REFERENCES systems(\_id) ON DELETE CASCADE)");

1100. }

1101.

1102. @Override

1103. public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

1104. // Метод вызывается при обновлении версии базы данных

1105. // Можно провести миграцию данных при необходимости

1106. }

1107.

1108. public String getSystemId(System system){

1109. SQLiteDatabase db = getReadableDatabase();

1110. Cursor cursor = db.rawQuery("SELECT \_id FROM systems WHERE system\_name = ? and mqtt\_url = ?",

1111. new String[]{system.getSystemName(), system.getMqtt\_url()});

1112. String systemId = "";

1113. if (cursor.moveToFirst()) {

1114. systemId = cursor.getString(0);

1115. }

1116. cursor.close();

1117. db.close();

1118. return systemId;

1119. }

1120.

1121. public boolean isSystemExists(String sysName) {

1122. SQLiteDatabase db = getReadableDatabase();

1123. Cursor cursor = db.rawQuery("SELECT COUNT(\*) FROM systems WHERE system\_name = ?", new String[]{sysName});

1124. int count = 0;

1125. if (cursor.moveToFirst()) {

1126. count = cursor.getInt(0);

1127. }

1128. cursor.close();

1129. db.close();

1130. return count > 0;

1131. }

1132.

1133. public boolean isDeviceExists(String deviceId) {

1134. SQLiteDatabase db = getReadableDatabase();

1135. Cursor cursor = db.rawQuery("SELECT COUNT(\*) FROM devices WHERE device\_id = ?", new String[]{deviceId});

1136. int count = 0;

1137. if (cursor.moveToFirst()) {

1138. count = cursor.getInt(0);

1139. }

1140. cursor.close();

1141. db.close();

1142. return count > 0;

1143. }

1144.

1145. public void addSystem(System system) {

1146. SQLiteDatabase db = getWritableDatabase();

1147. ContentValues values = new ContentValues();

1148. values.put("system\_name", system.getSystemName());

1149. values.put("mqtt\_url", system.getMqtt\_url());

1150. if(system.getMqtt\_login() != null && system.getMqtt\_password() != null){

1151. values.put("mqtt\_login", system.getMqtt\_login());

1152. values.put("mqtt\_password", system.getMqtt\_password());

1153. Log.i("MQTT addSystem DB", system.getMqtt\_url() + " " + system.getMqtt\_login() + " " + system.getMqtt\_password());

1154. }

1155. db.insert("systems", null, values);

1156. db.close();

1157. }

1158.

1159. public void updateSystem(System newSystem, System oldSystem) {

1160. String id = getSystemId(oldSystem);

1161. SQLiteDatabase db = getWritableDatabase();

1162. ContentValues values = new ContentValues();

1163. values.put("mqtt\_url", newSystem.getMqtt\_url());

1164. values.put("system\_name", newSystem.getSystemName());

1165. if (newSystem.getMqtt\_login() != null && newSystem.getMqtt\_password() != null) {

1166. values.put("mqtt\_login", newSystem.getMqtt\_login());

1167. values.put("mqtt\_password", newSystem.getMqtt\_password());

1168. }

1169. db.update("systems", values, "\_id = ?", new String[]{id});

1170. db.close();

1171. }

1172.

1173. public List<System> getAllSystems() {

1174. List<System> systems = new ArrayList<>();

1175. SQLiteDatabase db = getReadableDatabase();

1176. Cursor cursor = db.rawQuery("SELECT system\_name, mqtt\_url, mqtt\_login, mqtt\_password FROM systems", null);

1177. if (cursor.moveToFirst()) {

1178. do {

1179. String systemName = cursor.getString(0);

1180. String mqtt\_url = cursor.getString(1);

1181. System sys = new System(systemName, mqtt\_url);

1182. sys.setMqtt\_login(cursor.getString(2));

1183. sys.setMqtt\_password(cursor.getString(3));

1184. systems.add(sys);

1185. } while (cursor.moveToNext());

1186. }

1187. cursor.close();

1188. db.close();

1189. return systems;

1190. }

1191.

1192. public System getSystem(String systemName) {

1193. SQLiteDatabase db = getReadableDatabase();

1194. System system = null;

1195. Cursor cursor = db.rawQuery("SELECT mqtt\_url, mqtt\_login, mqtt\_password FROM systems WHERE system\_name = ?", new String[]{systemName});

1196. if (cursor.moveToFirst()) {

1197. String mqtt\_url = cursor.getString(0);

1198. system = new System(systemName, mqtt\_url);

1199. system.setMqtt\_login(cursor.getString(1));

1200. system.setMqtt\_password(cursor.getString(2));

1201. }

1202. cursor.close();

1203. db.close();

1204. return system;

1205. }

1206.

1207. public void deleteAllSystems() {

1208. SQLiteDatabase db = getWritableDatabase();

1209. db.delete("systems", null, null);

1210. db.close();

1211. }

1212.

1213. public void addDevice(Devices device, System system){

1214. Log.i("sql", device.getDiodeChannel());

1215. String id = getSystemId(system);

1216. SQLiteDatabase db = getWritableDatabase();

1217. String img = device.getImgPath();

1218. String friendlyName = device.getFriendlyName();

1219. ContentValues values = new ContentValues();

1220. values.put("device\_id", device.getDeviceId());

1221. values.put("device\_type", device.getType());

1222. values.put("system\_id", id);

1223. values.put("mqtt\_prefix", device.getMqttPrefix());

1224. if(!img.isEmpty())

1225. values.put("img\_path", img);

1226. if(!friendlyName.isEmpty())

1227. values.put("friendly\_name", friendlyName);

1228. if(!device.getDiodeChannel().isEmpty())

1229. values.put("diode\_channel", device.getDiodeChannel());

1230. db.insert("devices", null, values);

1231. db.close();

1232. }

1233.

1234. public ArrayList<Devices> getAllDevices(System system) {

1235. String sys\_id = getSystemId(system);

1236. ArrayList<Devices> devicesList = new ArrayList<>();

1237. SQLiteDatabase db = this.getReadableDatabase();

1238.

1239. String selectQuery = "SELECT devices.\* FROM devices " +

1240. "INNER JOIN systems ON devices.system\_id = systems.\_id " +

1241. "WHERE systems.\_id = ?";

1242.

1243. Cursor cursor = db.rawQuery(selectQuery, new String[]{sys\_id});

1244.

1245. if (cursor != null && cursor.moveToFirst()) {

1246. do {

1247. // Получение данных о устройстве из курсора и добавление в список

1248. int deviceIdIndex = cursor.getColumnIndex("device\_id");

1249. int deviceTypeIndex = cursor.getColumnIndex("device\_type");

1250. int deviceImgIndex = cursor.getColumnIndex("img\_path");

1251. int deviceFriendlyNameIndex = cursor.getColumnIndex("friendly\_name");

1252. int deviceLastDataIndex = cursor.getColumnIndex("last\_data");

1253. int deviceMqttPrefixIndex = cursor.getColumnIndex("mqtt\_prefix");

1254. int deviceDiodeChannelIndex = cursor.getColumnIndex("diode\_channel");

1255.

1256. String deviceId = cursor.getString(deviceIdIndex);

1257. String deviceType = cursor.getString(deviceTypeIndex);

1258. String deviceImg = cursor.getString(deviceImgIndex);

1259. String deviceFriendlyName = cursor.getString(deviceFriendlyNameIndex);

1260. String deviceLastData = cursor.getString(deviceLastDataIndex);

1261. String deviceMqttPrefix = cursor.getString(deviceMqttPrefixIndex);

1262. String deviceDiodeChannel = cursor.getString(deviceDiodeChannelIndex);

1263.

1264. Log.i("sql get all devices", "device id: " + deviceId +

1265. ", type: " + deviceType + ", img path: " + deviceImg + ", friendlyName: " +

1266. deviceFriendlyName + ", friendlyName: " + deviceMqttPrefix +

1267. ", diode channel: " + deviceDiodeChannel);

1268.

1269. Devices device = new Devices(deviceId, deviceType, deviceImg);

1270. device.setFriendlyName(deviceFriendlyName);

1271. device.setLastAcceptedData(deviceLastData);

1272. device.setMqttPrefix(deviceMqttPrefix);

1273. device.setDiodeChannel(deviceDiodeChannel);

1274. devicesList.add(device);

1275. } while (cursor.moveToNext());

1276.

1277. cursor.close();

1278. }

1279. db.close();

1280. return devicesList;

1281. }

1282.

1283. public void deleteSystem(System system) {

1284. String sysId = getSystemId(system);

1285. SQLiteDatabase db = getWritableDatabase();

1286. db.delete("devices", "system\_id = ?", new String[]{sysId});

1287. db.delete("systems", "\_id = ?", new String[]{sysId});

1288. db.close();

1289. }

1290.

1291. public void deleteDevice(Devices device) {

1292. String devId = getDeviceId(device);

1293. SQLiteDatabase db = getWritableDatabase();

1294. db.delete("devices", "\_id = ?", new String[]{devId});

1295. db.close();

1296. }

1297.

1298. public String getDeviceId(Devices device) {

1299. SQLiteDatabase db = getReadableDatabase();

1300. Log.i("sql", device.getDeviceId());

1301. Cursor cursor = db.rawQuery("SELECT \_id FROM devices WHERE device\_id = ?",

1302. new String[]{device.getDeviceId()});

1303. String deviceId = "";

1304. if (cursor.moveToFirst()) {

1305. deviceId = cursor.getString(0);

1306. }

1307. cursor.close();

1308. db.close();

1309. return deviceId;

1310. }

1311.

1312. public int getIdByDeviceId(String deviceId) {

1313. SQLiteDatabase db = getReadableDatabase();

1314. Cursor cursor = db.rawQuery("SELECT \_id FROM devices WHERE device\_id = ?", new String[]{deviceId});

1315. String id = "";

1316. if (cursor.moveToFirst()) {

1317. id = cursor.getString(0);

1318. }

1319. cursor.close();

1320. db.close();

1321. return Integer.parseInt(id);

1322. }

1323.

1324. public Devices getDeviceByDeviceId(String deviceId) {

1325. SQLiteDatabase db = getReadableDatabase();

1326. Devices device = null;

1327. Cursor cursor = db.rawQuery("SELECT \* FROM devices WHERE device\_id = ?", new String[]{deviceId});

1328. if (cursor.moveToFirst()) {

1329. // Получение данных о устройстве из курсора

1330. int deviceTypeIndex = cursor.getColumnIndex("device\_type");

1331. int deviceImgIndex = cursor.getColumnIndex("img\_path");

1332. int deviceFriendlyNameIndex = cursor.getColumnIndex("friendly\_name");

1333. int deviceLastDataIndex = cursor.getColumnIndex("last\_data");

1334. int deviceMqttPrefixIndex = cursor.getColumnIndex("mqtt\_prefix");

1335. int deviceDiodeChannelIndex = cursor.getColumnIndex("diode\_channel");

1336.

1337. String deviceType = cursor.getString(deviceTypeIndex);

1338. String deviceImg = cursor.getString(deviceImgIndex);

1339. String deviceFriendlyName = cursor.getString(deviceFriendlyNameIndex);

1340. String deviceLastData = cursor.getString(deviceLastDataIndex);

1341. String deviceMqttPrefix = cursor.getString(deviceMqttPrefixIndex);

1342. String deviceDiodeChannel = cursor.getString(deviceDiodeChannelIndex);

1343.

1344. // Создание объекта устройства и инициализация его полей

1345. device = new Devices(deviceId, deviceType, deviceImg);

1346. device.setFriendlyName(deviceFriendlyName);

1347. device.setLastAcceptedData(deviceLastData);

1348. device.setMqttPrefix(deviceMqttPrefix);

1349. device.setDiodeChannel(deviceDiodeChannel);

1350. }

1351. cursor.close();

1352. db.close();

1353. return device;

1354. }

1355.

1356.

1357. public void updateDevice(Devices device, Devices oldDevice) {

1358. String id = getDeviceId(oldDevice);

1359. SQLiteDatabase db = getWritableDatabase();

1360. ContentValues values = new ContentValues();

1361. values.put("device\_type", device.getType());

1362. values.put("img\_path", device.getImgPath());

1363. values.put("friendly\_name", device.getFriendlyName());

1364. values.put("mqtt\_prefix", device.getMqttPrefix());

1365. values.put("diode\_channel", device.getDiodeChannel());

1366. if(!Objects.equals(device.getImgPath(), ""))

1367. values.put("img\_path", device.getImgPath());

1368. if(!device.getFriendlyName().isEmpty())

1369. values.put("friendly\_name", device.getFriendlyName());

1370. if(!device.getDiodeChannel().isEmpty())

1371. values.put("diode\_channel", device.getDiodeChannel());

1372. db.update("devices", values, "\_id = ?", new String[]{id});

1373. db.close();

1374. }

1375.

1376. public void updateLastDataForDevice(Devices device, String newData) {

1377. SQLiteDatabase db = getWritableDatabase();

1378. ContentValues values = new ContentValues();

1379. values.put("last\_data", newData);

1380. db.update("devices", values, "device\_id = ?", new String[]{device.getDeviceId()});

1381. db.close();

1382. }

1383.

1384.

1385. }

1386.

1387.

1388. Файл DeleteConfirmationDialog.java

1389.

1390. package com.example.diplomaapp.dataClasses;

1391.

1392. import android.content.Context;

1393. import android.content.DialogInterface;

1394. import androidx.appcompat.app.AlertDialog;

1395.

1396. public class DeleteConfirmationDialog {

1397.

1398. public interface DeleteDialogListener {

1399. void onDeleteConfirmed();

1400. }

1401.

1402. public interface CancelDialogListener {

1403. void onCancel();

1404. }

1405.

1406. public static void show(Context context, String message, final DeleteDialogListener deleteListener,

1407. final CancelDialogListener cancelListener) {

1408. AlertDialog.Builder builder = new AlertDialog.Builder(context);

1409. builder.setMessage(message)

1410. .setPositiveButton("Delete", new DialogInterface.OnClickListener() {

1411. @Override

1412. public void onClick(DialogInterface dialogInterface, int i) {

1413. if (deleteListener != null) {

1414. deleteListener.onDeleteConfirmed();

1415. }

1416. }

1417. })

1418. .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {

1419. @Override

1420. public void onClick(DialogInterface dialogInterface, int i) {

1421. dialogInterface.dismiss();

1422. if (cancelListener != null) {

1423. cancelListener.onCancel();

1424. }

1425. }

1426. });

1427. AlertDialog dialog = builder.create();

1428. dialog.show();

1429. }

1430. }

1431.

1432.

1433.

1434. Файл DeviceAdapter.java

1435.

1436. package com.example.diplomaapp.dataClasses;

1437.

1438. import android.content.Context;

1439. import android.graphics.Bitmap;

1440. import android.graphics.BitmapFactory;

1441. import android.util.Log;

1442. import android.view.LayoutInflater;

1443. import android.view.View;

1444. import android.view.ViewGroup;

1445. import android.widget.ImageView;

1446. import android.widget.Switch;

1447. import android.widget.TextView;

1448.

1449. import androidx.annotation.NonNull;

1450. import androidx.recyclerview.widget.RecyclerView;

1451.

1452. import com.example.diplomaapp.R;

1453.

1454. import org.eclipse.paho.android.service.MqttAndroidClient;

1455. import org.eclipse.paho.client.mqttv3.MqttException;

1456. import org.eclipse.paho.client.mqttv3.MqttMessage;

1457.

1458. import java.io.File;

1459. import java.util.ArrayList;

1460.

1461. public class DeviceAdapter extends RecyclerView.Adapter<DeviceAdapter.ViewHolder> {

1462. private ArrayList<Devices> mDataset;

1463. private Context context;

1464. private MqttAndroidClient mqttAndroidClient;

1465.

1466. public static class ViewHolder extends RecyclerView.ViewHolder {

1467. public TextView textNameCard, textDataCard;

1468. public Switch switchButton;

1469. public ImageView imageView;

1470. public View view;

1471.

1472. public ViewHolder(View v) {

1473. super(v);

1474. textNameCard = v.findViewById(R.id.textNameCard);

1475. textDataCard = v.findViewById(R.id.textDataCard);

1476. switchButton = v.findViewById(R.id.switchBtn);

1477. imageView = v.findViewById(R.id.imageView);

1478. }

1479.

1480. public void SetDetails(Devices device){

1481. setFriendlyName(device);

1482. setDiodeChannelAndLastData(device);

1483. setImage(device);

1484. }

1485.

1486. public void setFriendlyName(Devices device){

1487. if(device.getFriendlyName() != null)

1488. textNameCard.setText(device.getFriendlyName() + "\n(" + device.getDeviceId() + ")");

1489. else textDataCard.setText(device.getType());

1490. }

1491.

1492. public void setDiodeChannelAndLastData(Devices device){

1493. if(device.getDiodeChannel() == null){

1494. switchButton.setVisibility(View.INVISIBLE);

1495. }

1496. else {

1497. if (device.getLastAcceptedData() != null) {

1498.

1499. String[] keyValuePairs = device.getLastAcceptedData().split("\n");

1500. StringBuilder newDataStringBuilder = new StringBuilder();

1501. for (String pair : keyValuePairs) {

1502. String[] keyValue = pair.split(":");

1503. if (keyValue.length == 2) {

1504. String key = keyValue[0].trim();

1505.

1506. if (!key.equals("state\_" + device.getDiodeChannel())) {

1507. newDataStringBuilder.append(pair).append("\n");

1508. }

1509. }

1510. }

1511. int ind = newDataStringBuilder.lastIndexOf("\n");

1512. newDataStringBuilder.delete(ind, newDataStringBuilder.length());

1513.

1514. textDataCard.setText(device.getType() + "\n\n" + newDataStringBuilder.toString());

1515.

1516. for (String pair : keyValuePairs) {

1517. String[] keyValue = pair.split(":");

1518. if (keyValue.length == 2) {

1519. String key = keyValue[0].trim();

1520. String value = keyValue[1].trim();

1521.

1522. if (key.equals("state\_" + device.getDiodeChannel())) {

1523. switchButton.setChecked(!value.equals("OFF"));

1524. break;

1525. }

1526. }

1527. }

1528. }else{

1529. textDataCard.setText(device.getType());

1530. }

1531. }

1532. }

1533.

1534. public void setImage(Devices device){

1535. if (device.getImgPath() != null){

1536. if (!device.getImgPath().isEmpty()) {

1537. try {

1538. File file = new File(device.getImgPath());

1539. if (file.exists()) {

1540. Bitmap bitmap = BitmapFactory.decodeFile(file.getAbsolutePath());

1541. imageView.setImageBitmap(bitmap);

1542. }

1543. } catch (Exception e) {

1544. e.printStackTrace();

1545. }

1546. }

1547. }

1548. }

1549. public View getView() {

1550. return view;

1551. }

1552. }

1553.

1554. // Provide a suitable constructor (depends on the kind of dataset)

1555. public DeviceAdapter(Context context, ArrayList<Devices> myDataset, MqttAndroidClient mqttAndroidClient) {

1556. this.mDataset = myDataset;

1557. this.context = context;

1558. this.mqttAndroidClient = mqttAndroidClient;

1559. }

1560.

1561. // Create new views (invoked by the layout manager)

1562. @NonNull

1563. @Override

1564. public DeviceAdapter.ViewHolder onCreateViewHolder(ViewGroup parent,

1565. int viewType) {

1566. View v = LayoutInflater.from(context).inflate(R.layout.item\_card, parent, false);

1567. return new ViewHolder(v);

1568. }

1569.

1570. @Override

1571. public void onBindViewHolder(@NonNull ViewHolder holder, int position) {

1572. Devices device = mDataset.get(position);

1573. holder.SetDetails(device);

1574.

1575. holder.switchButton.setOnCheckedChangeListener((buttonView, isChecked) -> {

1576. MqttMessage mqttMessage = new MqttMessage();

1577. if (isChecked) {

1578. mqttMessage.setPayload("On".getBytes());

1579. Log.i("button", "On " + holder.getAdapterPosition());

1580. } else {

1581. Log.i("button", "Off " + holder.getAdapterPosition());

1582. mqttMessage.setPayload("Off".getBytes());

1583. }

1584. try {

1585. Log.i("MQTT", device.getMqttPrefix());

1586. mqttAndroidClient.publish(device.getMqttPrefix() + "/" + device.getDeviceId() + "/" + device.getDiodeChannel() + "/set", mqttMessage);

1587. } catch (MqttException e) {

1588. throw new RuntimeException(e);

1589. }

1590. });

1591. }

1592.

1593. // Return the size of your dataset (invoked by the layout manager)

1594. @Override

1595. public int getItemCount() {

1596. return mDataset.size();

1597. }

1598. }

1599.

1600.

1601. Файл Devices.java

1602.

1603. package com.example.diplomaapp.dataClasses;

1604.

1605.

1606. public class Devices {

1607. private String friendlyName = "";

1608. private String deviceId;

1609. private String type;

1610. private String imgPath = "";

1611. private String lastAcceptedData = "";

1612. private String MqttPrefix = "";

1613. private String diodeChannel = "";

1614.

1615. public Devices(String deviceId, String type, String imgPath) {

1616. this.deviceId = deviceId;

1617. this.type = type;

1618. this.imgPath = imgPath;

1619. }

1620.

1621. public String getDeviceId() {

1622. return deviceId;

1623. }

1624.

1625. public void setDeviceId(String deviceId) {

1626. this.deviceId = deviceId;

1627. }

1628.

1629. public String getLastAcceptedData() {

1630. return lastAcceptedData;

1631. }

1632.

1633. public void setLastAcceptedData(String lastAcceptedData) {

1634. this.lastAcceptedData = lastAcceptedData;

1635. }

1636.

1637. public String getType() {

1638. return type;

1639. }

1640.

1641. public void setType(String type) {

1642. this.type = type;

1643. }

1644.

1645. public String getImgPath() {

1646. return imgPath;

1647. }

1648.

1649. public void setImgPath(String imgPath) {

1650. this.imgPath = imgPath;

1651. }

1652.

1653. public String getFriendlyName() {

1654. return friendlyName;

1655. }

1656.

1657. public void setFriendlyName(String friendlyName) {

1658. this.friendlyName = friendlyName;

1659. }

1660.

1661. public String getMqttPrefix() {

1662. return MqttPrefix;

1663. }

1664.

1665. public void setMqttPrefix(String mqttPrefix) {

1666. MqttPrefix = mqttPrefix;

1667. }

1668.

1669. public String getDiodeChannel() {

1670. return diodeChannel;

1671. }

1672.

1673. public void setDiodeChannel(String diodeChannel) {

1674. this.diodeChannel = diodeChannel;

1675. }

1676. }

1677.

1678.

1679. Файл IPAddressFilter.java

1680.

1681. package com.example.diplomaapp.dataClasses;

1682.

1683. import android.text.InputFilter;

1684. import android.text.Spanned;

1685.

1686. public class IPAddressFilter implements InputFilter {

1687. @Override

1688. public CharSequence filter(CharSequence source, int start, int end, Spanned dest, int dstart, int dend) {

1689. if (end > start) {

1690. String destTxt = dest.toString();

1691. String resultingTxt = destTxt.substring(0, dstart)

1692. + source.subSequence(start, end)

1693. + destTxt.substring(dend);

1694. if (!resultingTxt.matches("^\\d{1,3}(\\.(\\d{1,3}(\\.(\\d{1,3}(\\.(\\d{1,3})?)?)?)?)?)?")) {

1695. return "";

1696. } else {

1697. String[] splits = resultingTxt.split("\\.");

1698. for (int i = 0; i < splits.length; i++) {

1699. if (Integer.valueOf(splits[i]) > 255) {

1700. return "";

1701. }

1702. }

1703. }

1704. }

1705. return null;

1706. }

1707. }

1708.

1709.

1710.

1711. Файл MqttHelper.java

1712.

1713. package com.example.diplomaapp.dataClasses;

1714.

1715. import android.content.Context;

1716. import android.util.Log;

1717. import android.widget.Toast;

1718.

1719. import com.example.diplomaapp.listeners.MqttConnectListener;

1720.

1721. import org.eclipse.paho.android.service.MqttAndroidClient;

1722. import org.eclipse.paho.client.mqttv3.IMqttActionListener;

1723. import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;

1724. import org.eclipse.paho.client.mqttv3.IMqttToken;

1725. import org.eclipse.paho.client.mqttv3.MqttCallback;

1726. import org.eclipse.paho.client.mqttv3.MqttClient;

1727. import org.eclipse.paho.client.mqttv3.MqttConnectOptions;

1728. import org.eclipse.paho.client.mqttv3.MqttMessage;

1729.

1730. public class MqttHelper {

1731. private MqttAndroidClient mqttAndroidClient;

1732.

1733. public MqttHelper(Context appContext, String serverUri, final String username, final String password, MqttConnectListener connectListener) {

1734. String clientId = MqttClient.generateClientId();

1735. mqttAndroidClient = new MqttAndroidClient(appContext, serverUri, clientId);

1736. mqttAndroidClient.setCallback(new MqttCallback() {

1737.

1738. @Override

1739. public void connectionLost(Throwable cause) {}

1740.

1741. @Override

1742. public void messageArrived(String topic, MqttMessage message) throws Exception {

1743. Log.i("MQTT", message.toString());

1744. }

1745.

1746. @Override

1747. public void deliveryComplete(IMqttDeliveryToken token) {

1748.

1749. }

1750. });

1751.

1752. MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();

1753. mqttConnectOptions.setAutomaticReconnect(true);

1754. mqttConnectOptions.setCleanSession(false);

1755. if(username != null && password != null)

1756. {

1757. mqttConnectOptions.setUserName(username);

1758. mqttConnectOptions.setPassword(password.toCharArray());

1759. }

1760.

1761. try {

1762. mqttAndroidClient.connect(mqttConnectOptions, null, new IMqttActionListener() {

1763. @Override

1764. public void onSuccess(IMqttToken asyncActionToken) {

1765. Log.i("MQTT", "Successfully connected");

1766. if (connectListener != null) {

1767. connectListener.onSuccess(); // Уведомляем об успешном подключении

1768. }

1769.

1770. subscribeToTopic("zigbee2mqtt/0x00124B00281A9824");

1771. // publishMessage("zigbee2mqtt/0x123456789/l1/state", "Off");

1772. }

1773.

1774. @Override

1775. public void onFailure(IMqttToken asyncActionToken, Throwable exception) {

1776. Toast.makeText(appContext.getApplicationContext(), "Failed to connect: " + exception.toString(), Toast.LENGTH\_LONG).show();

1777. if (connectListener != null) {

1778. connectListener.onFailure(new Throwable()); // Уведомляем об успешном подключении

1779. }

1780. }

1781. });

1782. } catch (Exception ex) {

1783. ex.printStackTrace();

1784. Log.i("MQTT", ex.toString());

1785. }

1786. }

1787.

1788. public void subscribeToTopic(String topic) {

1789. try {

1790. mqttAndroidClient.subscribe(topic, 0, null, new IMqttActionListener() {

1791. @Override

1792. public void onSuccess(IMqttToken asyncActionToken) {

1793. Log.i("MQTT", "CONNECTED TO TOPIC");

1794. }

1795.

1796. @Override

1797. public void onFailure(IMqttToken asyncActionToken, Throwable exception) {

1798. Log.i("MQTT", "NOT CONNECTED TO TOPIC");

1799. }

1800. });

1801. } catch (Exception ex) {

1802. Log.i("MQTT", ex.toString());

1803. ex.printStackTrace();

1804. }

1805. }

1806.

1807. public void publishMessage(String topic, String message) {

1808. try {

1809. MqttMessage mqttMessage = new MqttMessage();

1810. mqttMessage.setPayload(message.getBytes());

1811. mqttAndroidClient.publish(topic, mqttMessage);

1812. } catch (Exception ex) {

1813. ex.printStackTrace();

1814. }

1815. }

1816.

1817. public void disconnect() {

1818. try {

1819. mqttAndroidClient.disconnect();

1820. } catch (Exception ex) {

1821. ex.printStackTrace();

1822. }

1823. }

1824.

1825. public boolean isConnected() {

1826. return mqttAndroidClient != null && mqttAndroidClient.isConnected();

1827. }

1828. }

1829.

1830.

1831.

1832. Файл NotificationHelper.java

1833.

1834. package com.example.diplomaapp.dataClasses;

1835.

1836. import android.app.Notification;

1837. import android.app.NotificationChannel;

1838. import android.app.NotificationManager;

1839. import android.content.Context;

1840. import android.os.Build;

1841. import androidx.core.app.NotificationCompat;

1842.

1843. import com.example.diplomaapp.R;

1844.

1845. public class NotificationHelper {

1846.

1847. private static final String CHANNEL\_ID = "my\_app\_notifications";

1848. private static final String CHANNEL\_NAME = "My App Notifications";

1849.

1850. public static void showNotification(Context context, String title, String message, int notificationId) {

1851. NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION\_SERVICE);

1852.

1853. // Проверяем версию Android для создания канала уведомлений только для Android 8.0 и выше

1854. if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

1855. createNotificationChannel(notificationManager);

1856. }

1857.

1858. // Создаем уведомление

1859. NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL\_ID)

1860. .setSmallIcon(R.drawable.safe) // Устанавливаем иконку уведомления

1861. .setContentTitle(title) // Устанавливаем заголовок уведомления

1862. .setContentText(message) // Устанавливаем текст уведомления

1863. .setPriority(NotificationCompat.PRIORITY\_DEFAULT); // Устанавливаем приоритет уведомления

1864.

1865. // Отображаем уведомление

1866. notificationManager.notify(notificationId, builder.build());

1867. }

1868.

1869. // Метод для создания канала уведомлений

1870. private static void createNotificationChannel(NotificationManager notificationManager) {

1871. // Создаем канал уведомлений только для Android 8.0 и выше

1872. if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {

1873. NotificationChannel channel = new NotificationChannel(CHANNEL\_ID, CHANNEL\_NAME, NotificationManager.IMPORTANCE\_DEFAULT);

1874. notificationManager.createNotificationChannel(channel);

1875. }

1876. }

1877. }

1878.

1879.

1880.

1881. Файл PortFilter.java

1882.

1883. package com.example.diplomaapp.dataClasses;

1884.

1885. import android.text.InputFilter;

1886.

1887. public class PortFilter implements InputFilter {

1888. @Override

1889. public CharSequence filter(CharSequence source, int start, int end,

1890. android.text.Spanned dest, int dstart, int dend) {

1891. if (end > start) {

1892. String destTxt = dest.toString();

1893. String resultingTxt = destTxt.substring(0, dstart)

1894. + source.subSequence(start, end)

1895. + destTxt.substring(dend);

1896. try {

1897. int port = Integer.parseInt(resultingTxt);

1898. if (port < 0 || port > 65535) {

1899. return "";

1900. }

1901. } catch (NumberFormatException e) {

1902. return "";

1903. }

1904. }

1905. return null;

1906. }

1907. }

1908.

1909.

1910.

1911. Файл Storage.java

1912.

1913. package com.example.diplomaapp.dataClasses;

1914.

1915. public class Storage {

1916.

1917. public static System system;

1918. public static Devices device;

1919. }

1920.

1921.

1922. Файл SwipeToDeleteCallback.java

1923.

1924. package com.example.diplomaapp.dataClasses;

1925.

1926. import android.content.Context;

1927. import android.graphics.Canvas;

1928. import android.graphics.Color;

1929. import android.graphics.drawable.Drawable;

1930. import android.graphics.drawable.GradientDrawable;

1931. import android.view.View;

1932.

1933. import androidx.annotation.NonNull;

1934. import androidx.core.content.ContextCompat;

1935. import androidx.recyclerview.widget.ItemTouchHelper;

1936. import androidx.recyclerview.widget.RecyclerView;

1937.

1938. import com.example.diplomaapp.R;

1939. import com.example.diplomaapp.listeners.OnSwipeListener;

1940.

1941.

1942. public class SwipeToDeleteCallback extends ItemTouchHelper.SimpleCallback {

1943. private final Drawable iconDelete;

1944. private final Drawable iconChange;

1945. private final GradientDrawable backgroundLeft;

1946. private final GradientDrawable backgroundRight;

1947. private final OnSwipeLeftListener swipeLeftListener;

1948. private final OnSwipeRightListener swipeRightListener;

1949. private Context context;

1950.

1951. public interface OnSwipeLeftListener {

1952. void onSwipeLeft(int position);

1953. }

1954.

1955. public interface OnSwipeRightListener {

1956. void onSwipeRight(int position);

1957. }

1958.

1959.

1960.

1961. public SwipeToDeleteCallback(Context context, OnSwipeLeftListener swipeLeftListener, OnSwipeRightListener swipeRightListener) {

1962. // super(0, ItemTouchHelper.LEFT);

1963. super(0, ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT);

1964. this.swipeLeftListener = swipeLeftListener;

1965. this.swipeRightListener = swipeRightListener;

1966. this.context = context;

1967.

1968. iconDelete = ContextCompat.getDrawable(this.context, R.drawable.bin\_black);

1969. iconChange = ContextCompat.getDrawable(this.context, R.drawable.draw);

1970.

1971. backgroundLeft = new GradientDrawable();

1972. backgroundLeft.setColor(Color.argb(100,223, 0, 76)); // Устанавливаем цвет

1973. backgroundLeft.setCornerRadius(12); // Закругляем углы

1974.

1975. // Создаем GradientDrawable для фона свайпа вправо

1976. backgroundRight = new GradientDrawable();

1977. backgroundRight.setColor(Color.argb(100,164,255,164)); // Устанавливаем цвет

1978. backgroundRight.setCornerRadius(12); // Закругляем углы

1979. }

1980.

1981. @Override

1982. public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull RecyclerView.ViewHolder viewHolder, @NonNull RecyclerView.ViewHolder target) {

1983. return false;

1984. }

1985.

1986. @Override

1987. public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {

1988. int position = viewHolder.getAdapterPosition();

1989. if (direction == ItemTouchHelper.LEFT && swipeLeftListener != null) {

1990. swipeLeftListener.onSwipeLeft(position);

1991. } else if (direction == ItemTouchHelper.RIGHT && swipeRightListener != null) {

1992. swipeRightListener.onSwipeRight(position);

1993. }

1994. }

1995.

1996. @Override

1997. public void onChildDraw(@NonNull Canvas c, @NonNull RecyclerView recyclerView, @NonNull RecyclerView.ViewHolder viewHolder, float dX, float dY, int actionState, boolean isCurrentlyActive) {

1998. int maxSwipeDistance = 250;

1999. if (Math.abs(dX) > maxSwipeDistance) {

2000. dX = Math.signum(dX) \* maxSwipeDistance;

2001. }

2002. super.onChildDraw(c, recyclerView, viewHolder, dX, dY, actionState, isCurrentlyActive);

2003. View itemView = viewHolder.itemView;

2004.

2005. if (dX > 0) { // Swipe вправо

2006. int iconMargin = (itemView.getHeight() - iconChange.getIntrinsicHeight()) / 2;

2007. int iconTop = itemView.getTop() + (itemView.getHeight() - iconChange.getIntrinsicHeight()) / 2;

2008. int iconBottom = iconTop + iconChange.getIntrinsicHeight();

2009.

2010. int iconLeft = itemView.getLeft() + iconMargin;

2011. int iconRight = itemView.getLeft() + iconMargin + iconChange.getIntrinsicWidth();

2012. iconChange.setBounds(iconLeft, iconTop, iconRight, iconBottom);

2013.

2014. backgroundRight.setBounds(itemView.getLeft(), itemView.getTop(), itemView.getLeft() + ((int) dX) + backgroundRight.getIntrinsicWidth(), itemView.getBottom());

2015. backgroundRight.draw(c);

2016. } else if (dX < 0) { // Swipe влево

2017. int iconMargin = (itemView.getHeight() - iconDelete.getIntrinsicHeight()) / 2;

2018. int iconTop = itemView.getTop() + (itemView.getHeight() - iconDelete.getIntrinsicHeight()) / 2;

2019. int iconBottom = iconTop + iconDelete.getIntrinsicHeight();

2020.

2021. int iconLeft = itemView.getRight() - iconMargin - iconDelete.getIntrinsicWidth();

2022. int iconRight = itemView.getRight() - iconMargin;

2023. iconDelete.setBounds(iconLeft, iconTop, iconRight, iconBottom);

2024.

2025. backgroundLeft.setBounds(itemView.getRight() + ((int) dX) - backgroundLeft.getIntrinsicWidth(), itemView.getTop(), itemView.getRight(), itemView.getBottom());

2026. backgroundLeft.draw(c);

2027. }

2028. iconDelete.draw(c);

2029. iconChange.draw(c);

2030. }

2031.

2032. @Override

2033. public float getSwipeThreshold(@NonNull RecyclerView.ViewHolder viewHolder) {

2034. return 0.4f;

2035. }

2036.

2037.

2038. }

2039.

2040.

2041.

2042. Файл System.java

2043.

2044. package com.example.diplomaapp.dataClasses;

2045.

2046. public class System {

2047. private String systemName;

2048. private String mqtt\_url;

2049. private String mqtt\_login;

2050. private String mqtt\_password;

2051.

2052. public System(String systemName, String mqtt\_url) {

2053. this.systemName = systemName;

2054. this.mqtt\_url = mqtt\_url;

2055. }

2056.

2057. public String getSystemName() {

2058. return systemName;

2059. }

2060.

2061. public void setSystemName(String systemName) {

2062. this.systemName = systemName;

2063. }

2064.

2065. public String getMqtt\_url() {

2066. return mqtt\_url;

2067. }

2068.

2069. public void setMqtt\_url(String mqtt\_url) {

2070. this.mqtt\_url = mqtt\_url;

2071. }

2072.

2073. public String getMqtt\_login() {

2074. return mqtt\_login;

2075. }

2076.

2077. public void setMqtt\_login(String mqtt\_login) {

2078. this.mqtt\_login = mqtt\_login;

2079. }

2080.

2081. public String getMqtt\_password() {

2082. return mqtt\_password;

2083. }

2084.

2085. public void setMqtt\_password(String mqtt\_password) {

2086. this.mqtt\_password = mqtt\_password;

2087. }

2088. }

2089.

2090.

2091. Файл SystemAdapter.java

2092.

2093. package com.example.diplomaapp.dataClasses;

2094.

2095. import android.content.Context;

2096. import android.content.Intent;

2097. import android.util.Log;

2098. import android.view.LayoutInflater;

2099. import android.view.View;

2100. import android.view.ViewGroup;

2101. import android.widget.TextView;

2102.

2103. import androidx.annotation.NonNull;

2104. import androidx.recyclerview.widget.RecyclerView;

2105.

2106. import com.example.diplomaapp.DevicesActivity;

2107. import com.example.diplomaapp.R;

2108. import java.util.ArrayList;

2109.

2110. public class SystemAdapter extends RecyclerView.Adapter<SystemAdapter.ViewHolder> {

2111. private static DBHelper dbHelper;

2112. private ArrayList<System> mDataset;

2113. private Context context;

2114.

2115. public static class ViewHolder extends RecyclerView.ViewHolder {

2116. public TextView textNameSystem, textDevicesSystem;

2117. public View view;

2118.

2119. public ViewHolder(View v) {

2120. super(v);

2121. textNameSystem = v.findViewById(R.id.textNameSystem);

2122. textDevicesSystem = v.findViewById(R.id.textDevicesSystem);

2123. }

2124.

2125. public void SetDetails(System system){

2126. textNameSystem.setText(system.getSystemName());

2127. ArrayList<Devices> devices = dbHelper.getAllDevices(system);

2128. StringBuilder devStr = new StringBuilder();

2129. devices.forEach((x)->devStr.append(x.getFriendlyName()).append(" "));

2130. if(!devStr.toString().isEmpty())

2131. textDevicesSystem.setText(devStr);

2132. }

2133. public View getView() {

2134. return view;

2135. }

2136. }

2137.

2138. // Provide a suitable constructor (depends on the kind of dataset)

2139. public SystemAdapter(Context context, ArrayList<System> myDataset) {

2140. this.mDataset = myDataset;

2141. this.context = context;

2142. dbHelper = new DBHelper(context.getApplicationContext());

2143. }

2144.

2145. // Create new views (invoked by the layout manager)

2146. @NonNull

2147. @Override

2148. public SystemAdapter.ViewHolder onCreateViewHolder(ViewGroup parent,

2149. int viewType) {

2150. View v = LayoutInflater.from(context).inflate(R.layout.system\_card, parent, false);

2151. return new ViewHolder(v);

2152. }

2153.

2154. @Override

2155. public void onBindViewHolder(@NonNull ViewHolder holder, int position) {

2156. System system = mDataset.get(position);

2157. holder.SetDetails(system);

2158. holder.itemView.setOnClickListener(new View.OnClickListener() {

2159. @Override

2160. public void onClick(View v) {

2161. int position = holder.getAdapterPosition();

2162. if (position != RecyclerView.NO\_POSITION) {

2163. System clickedSystem = mDataset.get(position);

2164.

2165. Intent intent = getIntent(clickedSystem);

2166. Log.i("MQTT main", clickedSystem.getMqtt\_url() + clickedSystem.getMqtt\_login() + clickedSystem.getMqtt\_password());

2167. context.startActivity(intent);

2168. }

2169. }

2170. });

2171. }

2172.

2173. @NonNull

2174. private Intent getIntent(System clickedSystem) {

2175. Intent intent = new Intent(context, DevicesActivity.class);

2176. intent.putExtra("systemName", clickedSystem.getSystemName());

2177. intent.putExtra("mqttUrl", clickedSystem.getMqtt\_url());

2178.

2179. if(clickedSystem.getMqtt\_login() != null && clickedSystem.getMqtt\_password() != null){

2180. intent.putExtra("mqtt\_login", clickedSystem.getMqtt\_login());

2181. intent.putExtra("mqtt\_password", clickedSystem.getMqtt\_password());

2182. }

2183. return intent;

2184. }

2185.

2186. @Override

2187. public int getItemCount() {

2188. return mDataset.size();

2189. }

2190. }

2191.

2192.

2193.

2194. Файл MqttConnectionLostListener.java

2195.

2196. package com.example.diplomaapp.listeners;

2197.

2198. public interface MqttConnectionLostListener {

2199. void onConnectionLost(Throwable cause);

2200. }

2201.

2202.

2203. Файл MqttConnectListener.java

2204.

2205. package com.example.diplomaapp.listeners;

2206.

2207. public interface MqttConnectListener {

2208. void onSuccess(); // Метод для уведомления об успешном подключении

2209. void onFailure(Throwable exception); // Метод для уведомления о неудачном подключении

2210. }

2211.

2212.

2213. Файл MqttDeliveryCompleteListener.java

2214.

2215. package com.example.diplomaapp.listeners;

2216.

2217. import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;

2218.

2219. public interface MqttDeliveryCompleteListener {

2220. void onDeliveryComplete(IMqttDeliveryToken token);

2221. }

2222.

2223.

2224. Файл MqttMessageArrivedListener.java

2225.

2226. package com.example.diplomaapp.listeners;

2227.

2228. import org.eclipse.paho.client.mqttv3.MqttMessage;

2229.

2230. public interface MqttMessageArrivedListener {

2231. void onMessageArrived(String topic, MqttMessage message);

2232. }

2233.

2234.

2235. Файл OnSwipeListener.java

2236.

2237. package com.example.diplomaapp.listeners;

2238.

2239. public interface OnSwipeListener {

2240. void onSwipe(int position);

2241. }

## ПРИЛОЖЕНИЕ Б

(обязательное)

Схема структурная

## ПРИЛОЖЕНИЕ В

(обязательное)

Схема функциональная

## ПРИЛОЖЕНИЕ Г

*(обязательное)*

Спецификация

## ПРИЛОЖЕНИЕ Д

*(обязательное)*

Ведомость документов