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

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, кмоплексом 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.

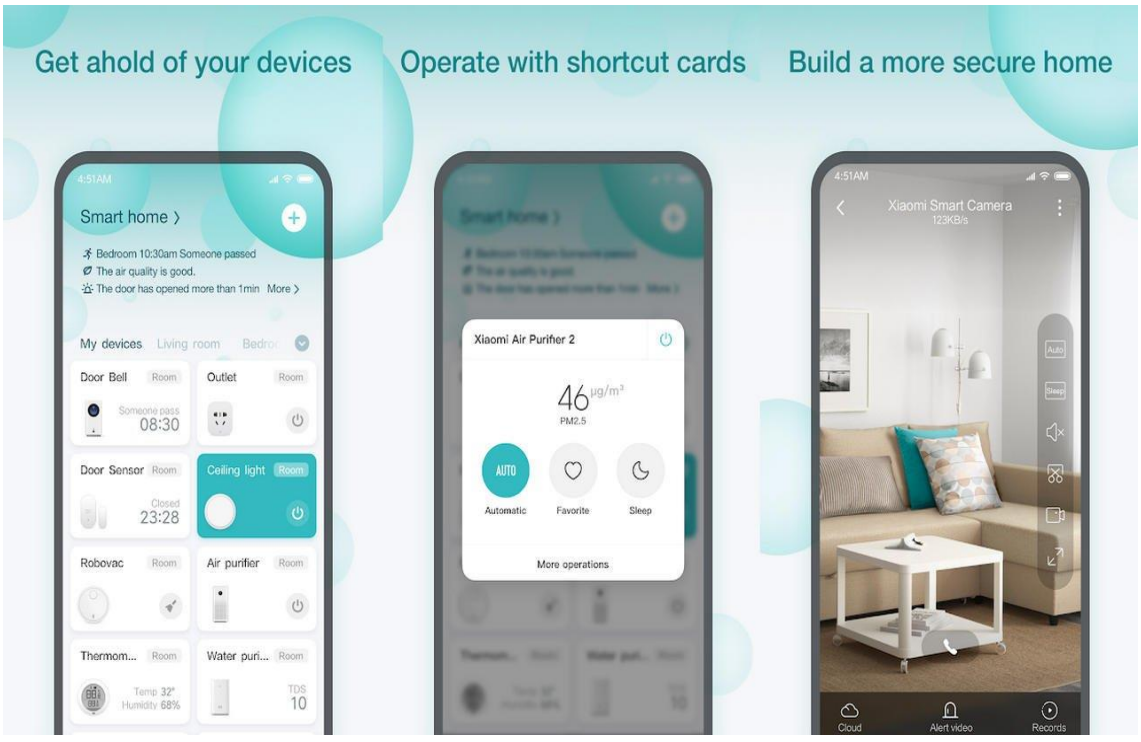

Рисунок 1.4 – Приложение Xiaomi Smart Home

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.

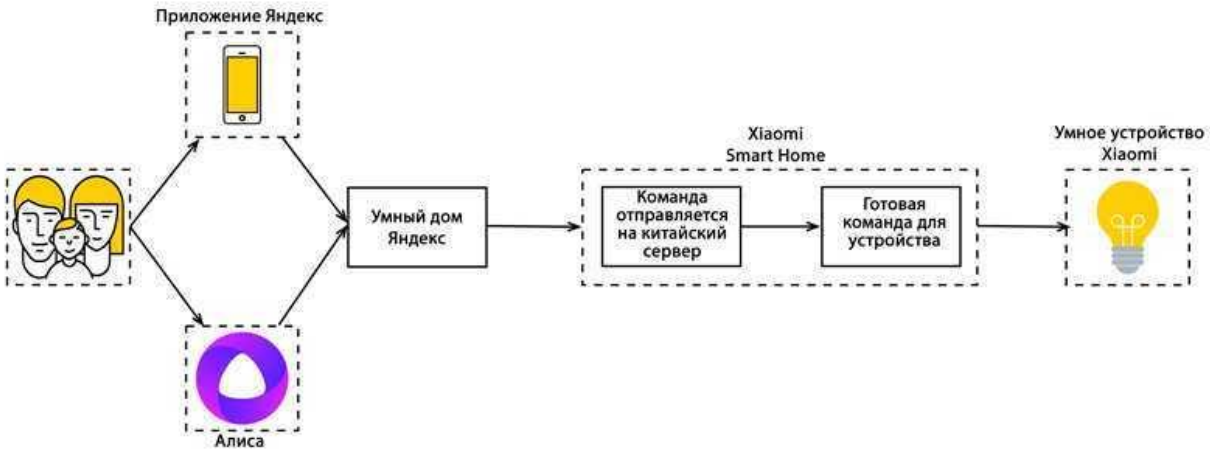

Рисунок 1.6 – Схема запросов в «Умном доме» от Яндекс

«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 I2C

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

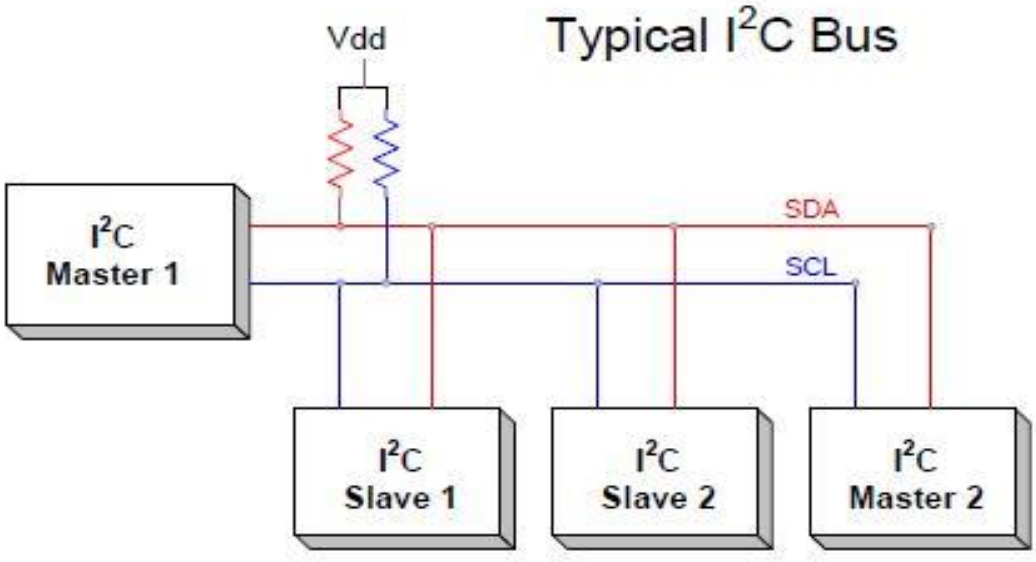

Рисунок 1.11 – Схема шины I2C

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.

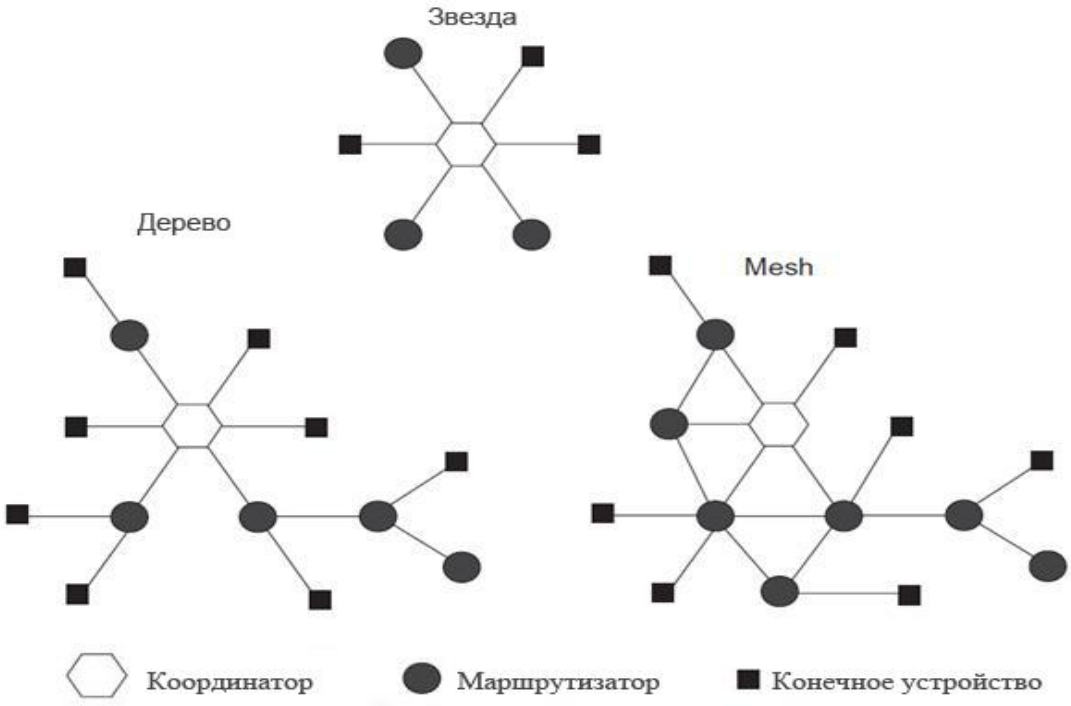

Рисунок 1.12 – Топология Zigbee

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 РР.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 E18 – 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 E18 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)` – создает 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)` – инициализирует пользовательский интерфейс и обработчики событий;
- `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;

– getItemCount() – возвращает количество элементов в списке устройств.

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.

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

dest, int dstart, int dend) – метод, который 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.

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.

`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 устройств;

– TextView с идентификатором textViewNoDevices, который отображает текст «No Devices», если список устройств пуст;

– ProgressBar с идентификатором progressBar, который отображается во время загрузки данных;

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

4.1 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.

4.2 Расчет затрат на разработку программного комплекса

4.2.1 Расчет затрат на основную заработную плату разработчиков

Для расчета основной заработной платы разработчиков использовалась формула 7.1:

$$З_о = К_{пр} \sum_{i=1}^{n} З_{чi} \cdot t_i \,, \qquad (7.1)$$

где n – количество исполнителей, занятых разработкой конкретного ПО;

$К_{пр}$ – коэффициент, учитывающий процент премий;

$З_{ч.i}$ – часовая заработная плата i-го исполнителя, р.;
t_i – 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

4.2.2 Расчет затрат на дополнительную заработную плату разработчиков

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

$$З_д = \frac{З_о \cdot Н_д}{100\%}, \quad (7.2)$$

где Зо – затраты на основную заработную плату, р.;
$Н_д$ – норматив дополнительной заработной платы.

Норматив дополнительной заработной платы был принят равным 15%.
Размер дополнительной заработной составил:

$$З_д = \frac{З_о \cdot Н_д}{100\%} = \frac{2\,939{,}04 \cdot 15}{100\%} = 440{,}86 \text{ р.}$$

4.2.3 Расчет отчислений на социальные нужды

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

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100\%}, \quad (7.3)$$

где $Н_{соц}$ – норматив отчислений от фонда оплаты труда.

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

Размер дополнительной заработной платы составил:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100\%} = \frac{(2\,939{,}04 + 440{,}86) \cdot 35}{100\%} = 1\,182{,}97 \text{ р.}$$

4.2.4 Расчет прочих расходов

Прочие расходов были рассчитаны по формуле 7.4:

$$Р_{пр} = \frac{З_о \cdot Н_{пр}}{100\%}, \quad (7.4)$$

где $Н_{пр}$ – норматив прочих расходов.

Норматив прочих расходов был принят равным 30%.

Размер прочих расходов составил:

$$Р_{пз} = \frac{З_о \cdot Н_{пр}}{100\%} = \frac{2\,939{,}04 \cdot 30}{100\%} = 881{,}71 \text{ р.}$$

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

4.3 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:

$$НДС = \frac{Ц_{отп} \cdot N \cdot Н_{д.с}}{100\% + Н_{д.с}}, \quad (0.1)$$

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

$Ц_{отп}$ – отпускная цена копии программного средства, р. ;

N – количество приобретённых лицензий;

$Н_{д.с}$ – 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:

$$НДС = \frac{16{,}32 \cdot 2\,000 \cdot 20\%}{100\% + 20\%} = 5\,440 \text{ р.}$$

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

$$\Delta П_{ч}^{р} = (Ц_{отп} \cdot N - НДС) \cdot Р_{пр} \cdot \left(1 - \frac{Н_{п}}{100}\right), \quad (0.2)$$

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

$Ц_{отп}$ – отпускная цена копии программного средства, р.;

НДС – сумма налога на добавленную стоимость, р.;

$Н_{п}$ – ставка налога на прибыль, %;

$Р_{пр}$ – 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:

$$\Delta П_{ч}^{р} = (16{,}32 \cdot 2\,000 - 5\,440) \cdot 30\% \cdot \left(1 - \frac{20}{100}\right) = 6528$$

4.4 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:

$$ROI = \frac{\Delta П_{ч}^{р} - З_{р}}{З_{р}} \cdot 100\% \qquad 0.3)$$

где $\Delta П_{ч}^{р}$ – прирост чистой прибыли, полученной от реализации программного средства на рынке информационных технологий, р.;

$З_{р}$ – затраты на разработку и реализацию программного средства, р.

$$ROI = \frac{6528 - 5\,444{,}58}{5\,444{,}58} \cdot 100\% = 19{,}89\%$$

4.5 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 значительно увеличить его успех и прибыльность.

ЗАКЛЮЧЕНИЕ

В ходе преддипломной практики была выполнена основная часть разработки программного комплекса «Умный дом». Это включало в себя обзор аналогов комплекса, выбор сервиса для общения между конечными устройствами и мобильным телефоном и написание мобильного приложения. Также были проведены расчеты экономической эффективности разработки.

Результаты этой работы представляют собой прочный фундамент для дальнейшей разработки комплекса. Архитектура системы организована таким образом, чтобы обеспечить гибкость и возможность расширения новыми компонентами по мере необходимости. Планируется проведение тестирования и испытаний системы, а также доработка программного обеспечения для обеспечения стабильной работы.

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)

Вводный плакат

ПРИЛОЖЕНИЕ Б
(обязательное)

Схема структурная

ПРИЛОЖЕНИЕ В
(обязательное)

Диаграмма классов