**СОДЕРЖАНИЕ**

[Введение 7](#_Toc196314386)

[1 Обзор литературы 8](#_Toc196314387)

[1.1 Обзор существующих аналогов 8](#_Toc196314388)

[1.2 Zigbee 12](#_Toc196314391)

[1.3 MQTT 12](#_Toc196314392)

[1.4 Интегрированная среда разработки Android Studio 14](#_Toc196314393)

[1.5 Архитектура приложений 15](#_Toc196314394)

[1.6 Язык программирования Kotlin 18](#_Toc196314397)

[1.7 Jetpack Compose 18](#_Toc196314398)

[2 Системное проектирование 20](#_Toc196314399)

[2.1 Блок MQTT клиента 20](#_Toc196314400)

[2.2 Блок соединения с брокером MQTT 21](#_Toc196314401)

[2.3 Блок коммуникации MQTT 22](#_Toc196314402)

[2.4 Блок обработки и генерации сообщений 22](#_Toc196314403)

[2.5 Блок базы данных 23](#_Toc196314404)

[2.6 Блок моделей 23](#_Toc196314405)

[2.7 Блок бизнес-логики 24](#_Toc196314406)

[2.8 Блок представлений 24](#_Toc196314407)

[2.9 Блок моделей представлений 25](#_Toc196314408)

[3 Функциональное проектирование 26](#_Toc196314409)

[3.1 Представления 26](#_Toc196314410)

[3.2 Модели представлений 31](#_Toc196314418)

[3.3 Модели 33](#_Toc196314421)

[3.4 Хранение данных 36](#_Toc196314426)

[3.5 MQTT и взаимодействие с брокером 40](#_Toc196314436)

[3.6 Вспомогательные компоненты системы 42](#_Toc196314442)

[4 Разработка программных модулей 44](#_Toc196314446)

[4.1 Управление соединением с MQTT-брокером 44](#_Toc196314447)

[4.2 Обработка данных полученных от MQTT-брокера 46](#_Toc196314450)

[4.3 Отправка пользовательских данных в MQTT-брокер 48](#_Toc196314454)

[4.4 Управление модулем светодиодной подсветки 49](#_Toc196314455)

[4.5 Отображение общего списка устройств системы 51](#_Toc196314458)

[4.6 Отображение списка комнат системы 53](#_Toc196314462)

[4.7 Добавление новых комнат в систему 54](#_Toc196314463)

[4.8 Управление режимом обнаружения и подключения новых устройств 55](#_Toc196314464)

[4.9 Отображение детальной информации об устройстве 56](#_Toc196314465)

[4.10 Привязка устройства к определенной комнате 57](#_Toc196314466)

[5 Руководство пользователя 58](#_Toc196314467)

[5.1 Руководство пользователя координатора 58](#_Toc196314468)

[5.2 Руководство пользователя MQTT-брокера 62](#_Toc196314472)

[5.3 Руководство пользователя мобильного приложения 63](#_Toc196314477)

[5.4 Руководство программиста 68](#_Toc196314485)

[6 Программа и методика испытаний 71](#_Toc196314488)

[6.1 Автоматическое тестирование 71](#_Toc196314489)

[6.2 Ручное тестирование 72](#_Toc196314490)

[7 Технико-экономическое обоснование разработки и реализации на рынке  
      Android-приложения для управления и мониторинга устройствами умного  
      дома …. 77](#_Toc196314494)

[7.1 Характеристика программного средства, разрабатываемого для реализации на рынке 77](#_Toc196314495)

[7.2 Расчет инвестиций в разработку программного средства 77](#_Toc196314496)

[7.3 Расчет экономического эффекта от реализации программного средства на рынке 80](#_Toc196314502)

[7.4 Расчет показателей экономической эффективности разработки и реализации программного средства на рынке 81](#_Toc196314503)

[7.5 Вывод об экономической целесообразности реализации проектного решения 82](#_Toc196314504)

[Заключение 83](#_Toc196314505)

[Приложение А 86](#_Toc196314506)

[Приложение Б 123](#_Toc196314507)

[Приложение В 124](#_Toc196314507)

ВВЕДЕНИЕ

Благодаря развитию передовых технологий повседневная жизнь человека становится удобнее и эффективнее. Развитие систем «умный дом» позволило автоматизировать большинство бытовых процессов, обеспечивая комфорт для пользователей. Современные решения позволяют управлять освещением, климатом, системами безопасности и другими элементами, создавая интеллектуальную и адаптивную среду.

Развитие универсальных стандартов связи значительно упростило взаимодействие устройств разных производителей, обеспечивая их совместимость и позволяя пользователям создавать гибкие, масштабируемые экосистемы для управления. В результате современные системы автоматизации становятся все более функциональными и доступными.

Несмотря на все преимущества, вопрос стоимости остается одним из ключевых факторов, ограничивающих массовое внедрение систем «умный дом». Высокие цены на оборудование, необходимость приобретения дополнительных шлюзов и контроллеров, а также расходы на установку и настройку могут стать серьезным барьером для пользователей. Кроме того, некоторые устройства требуют регулярного обновления или подписки на облачные сервисы, что увеличивает общую стоимость использования внедряемой системы.

Однако, помимо готовых коммерческих решений, все большую популярность приобретают DIY-устройства (Do It Yourself – «сделай сам»), позволяющие пользователям самостоятельно создавать и настраивать системы «умного дома» под свои нужды. Такой подход не только снижает затраты, но и дает возможность полной кастомизации, позволяя интегрировать исполнительные устройства в единую интеллектуальную сеть.

Целью данного дипломного проекта является разработка Android-приложения, которое обеспечит мониторинг и управление устройствами «умного дома», предоставляя пользователю интуитивно понятный интерфейс для эффективного контроля, настройки и автоматизации различных устройств.

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

* прошивка и настройка шлюза «умного дома»;
* проектирование архитектуры системы «умный дом»;
* разработка Android-приложения для yправление системой «умный дом»;
* тестирование и отладка разработанного приложения.

Данный дипломный проект выполнен мной лично, проверен на заимствования, процент оригинальности составляет 92,45% (отчет о проверке на заимствования прилагается).

1. ОБЗОР ЛИТЕРАТУРЫ
   1. Обзор существующих аналогов

Одним из первых этапов разработки приложения для управления и мониторинга «умным домом» является определение перечня необходимых функций и выбор подходящих технологий. Для принятия обоснованных решений важно проанализировать преимущества и недостатки существующих решений на рынке.

Многие пользователи не имеют полного представления о возможностях современных систем «умного дома», их функционале и технических особенностях. Такие приложения могут включать широкий спектр функций – от управления освещением и климатом до мониторинга безопасности и автоматизации сценариев. В дальнейшем будут рассмотрены существующие решения, схожие по функционалу с разрабатываемым приложением.

Системы от «Xiaomi» предоставляет широкий ассортимент разнообразной техники, в том числе технику, которая отвечает требованиям «умного дома» [1]. В их ассортименте можно найти большое количество разнообразных устройств, что делает их продукцию универсальным решением для создания современного «умного дома».

Для удобного управления и взаимодействия с этими устройствами предоставляется специальное приложение, 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.

Для построения системы «умного дома» с устройствами Xiaomi и использованием протокола MQTT необходим централизованный хаб Xiaomi Smart Home Hub 2. Этот хаб выступает в роли шлюза между беспроводными устройствами, поддерживающими Zigbee 3.0, Bluetooth Mesh, Wi-Fi и Matter, обеспечивая их взаимодействие в единой экосистеме.

Xiaomi Smart Home Hub 2 подключается к сети через Ethernet или Wi-Fi и работает от розетки. В отличие от некоторых других хабов, он не оснащен встроенным аккумулятором, поэтому при отключении питания теряет связь с устройствами. Он поддерживает локальное управление через MQTT, что позволяет интегрировать его в собственные серверные решения, такие как HomeAssistant. Благодаря современному оборудованию хаб обеспечивает стабильную работу, минимальные задержки и возможность автоматизации без облачных сервисов.

Приложение Xiaomi Home является основным инструментом для управления устройствами «умного дома» от Xiaomi, а также совместимыми гаджетами других брендов, поддерживающих экосистему Mi Home. Оно доступно для Android и iOS, обеспечивая удобное удаленное взаимодействие с устройствами.

С помощью приложения можно добавлять устройства, управлять их настройками, создавать сценарии автоматизации и получать уведомления. Приложение поддерживает групповое управление, позволяя объединять устройства по комнатам и сценариям. Также реализована интеграция с голосовыми помощниками (для совместимых устройств).

Xiaomi Home позволяет настроить локальное управление для некоторых устройств, но большая часть автоматизаций выполняется через облачные серверы Xiaomi, что требует постоянного подключения к интернету. Интерфейс приложения интуитивно понятен, но скорость работы может зависеть от количества подключенных устройств и качества интернет-соединения.

Графический интерфейс приложения представлен на рисунке 1.1.



Рисунок 1.1 – Интерфейс приложения Xiaomi Home

​Приложение Xiaomi Home предоставляет пользователям удобный способ управления устройствами умного дома от Xiaomi. Однако, на основе отзывов пользователей, можно выделить несколько недостатков:​

1. Для корректной работы некоторых устройств и функций в приложении Xiaomi Home пользователям часто приходится выбирать определенный регион, например, Китай. Это связано с тем, что некоторые устройства могут быть доступны только в определенных регионах, а их функциональность может отличаться в зависимости от выбранных настроек.
2. Для стабильной работы некоторых функций рекомендуется использовать английский язык. Это может быть неудобно для пользователей, предпочитающих русский интерфейс, так как требует дополнительной адаптации.
3. Приложение официально поддерживает русский язык, но некоторые разделы и плагины могут отображаться на китайском или английском. Это создает сложности при использовании и может привести к затруднениям в настройке устройств.

Таким образом приложение Xiaomi Home является удобным инструментом для управления устройствами умного дома, предлагая широкий функционал и интеграцию с экосистемой Xiaomi. Однако его использование может вызывать трудности из-за ограничений, связанных с региональными настройками. Несмотря на недостатки, приложение остается популярным благодаря своей универсальности, низкой стоимости и поддержке большого количества устройств.

* + 1. Решения от компании «Яндекс» представляют собой экосистему, интегрированную с голосовым помощником Алисой. В ассортименте представлены устройства, такие как умные лампы, розетки, датчики движения и видеокамеры, а также шлюзы для взаимодействия с техникой сторонних производителей (например, Xiaomi, Philips Hue) через облачную интеграцию [2]. Основное преимущество системы – глубокая интеграция с сервисами Яндекса (Музыка, Погода, Расписания) и поддержка русского языка на всех уровнях взаимодействия.

В качестве шлюза чаще всего используется Яндекс Станция (умная колонка с голосовым управлением) или Яндекс Модуль, который подключается к сети через Wi-Fi и выступает мостом между устройствами умного дома и облачными сервисами. Шлюз поддерживает протоколы Wi-Fi и Bluetooth, но не Zigbee, что ограничивает совместимость с некоторыми устройствами. Управление через MQTT или локальные решения не предусмотрено – все команды обрабатываются через облако Яндекса. Это обеспечивает простоту настройки, но создает зависимость от интернет-соединения и серверов компании.

Приложение доступно для Android и iOS и служит центральным инструментом для управления устройствами. Ключевые функции данных приложений:

* добавление и группировка устройств по комнатам;
* создание сценариев автоматизации (например, «если датчик движения сработал, включить свет»);
* интеграция с голосовым помощником Алисой для управления через речь;
* уведомления о событиях (срабатывание датчиков, отключение устройств);
* совместимость с устройствами других брендов через облачные плагины (например, TP-Link, Redmond).

Интерфейс приложения интуитивно понятен и полностью русифицирован (см. рисунок 1.2). Для настройки сценариев используется визуальный редактор, где пользователь может задавать условия и действия с помощью «перетаскивания» элементов.



Рисунок 1.2 – Интерфейс приложения «Дом с Алисой»

Приложение «Яндекс.Умный дом» обеспечивает глубокую интеграцию с экосистемой сервисов Яндекса, что позволяет пользователям управлять устройствами через голосового помощника Алису и автоматически добавлять технику по QR-коду без ручного ввода данных. Локализация интерфейса, включая полную поддержку русского языка и голосовых команд, делает решение доступным для русскоязычной аудитории. Настройка устройств максимально упрощена: приложение не требует технических знаний, что идеально подходит для новичков. Кроме того, облачные сценарии автоматизации позволяют сохранять работоспособность функций даже при выключенном телефоне, так как логика обработки событий выполняется на серверах Яндекса.

Основным ограничением системы является сильная зависимость от облачных сервисов: большинство операций, включая управление устройствами и выполнение сценариев, требуют стабильного интернет-соединения, а локальное управление (например, через LAN) недоступно. Это создает риск потери контроля над умным домом при сбоях в сети. Кроме того, приложение поддерживает ограниченный набор протоколов – отсутствие Zigbee и MQTT сужает выбор совместимых устройств, делая экосистему закрытой.

Приложение «Яндекс.Умный дом» предлагает удобное решение для пользователей, которые ценят простоту и интеграцию с русскоязычными сервисами. Однако его зависимость от облака и закрытая экосистема ограничивают гибкость, что критично для продвинутых сценариев автоматизации. В отличие от Xiaomi, система Яндекса не поддерживает локальные протоколы (например, Zigbee), что делает ее менее универсальной.

* 1. Zigbee

Беспроводной протокол Zigbee [3], предназначенный для передачи данных между устройствами в сетях с низким энергопотреблением. Он работает на частоте 2,4 ГГц и поддерживает три типа устройств:

* координатор (главное устройство, управляющее сетью);
* роутер (передает сигналы другим устройствам);
* оконечное устройство (датчики, выключатели и другие узлы, потребляющие минимум энергии).

Благодаря сетевой топологии Mesh устройства могут передавать данные друг через друга, увеличивая дальность связи.

Zigbee активно применяется в системах умного дома, промышленной автоматизации, медицине и IoT-решениях. Его ключевые преимущества –низкое энергопотребление, высокая надежность за счет самовосстанавливающейся сети и совместимость с разными производителями. Однако у него есть и недостатки: ограниченная пропускная способность делает его неподходящим для передачи мультимедиа, а необходимость координатора может усложнять настройку сети. Кроме того, возможны помехи от Wi-Fi, так как оба протокола работают на одной частоте.

* 1. MQTT

Большинство устройств «умного дома» находятся в непосредственной близости друг от друга и работают в локальной сети, но для их работы в качестве IoT-устройств необходимо подключение к сети Интернет. Это позволит организовать мониторинг и управление устройствами из любой точки планеты при наличии интернет-доступа. Этот подход значительно облегчает взаимодействие человека с системой, поскольку сегодня мобильный интернет доступен практически повсюду.

В большинстве современных систем «умный дом» управление системой производится с использованием мобильного телефона с выходом в интернет, в редких случая используются веб-приложения доступные как для телефонов, так и для персональных компьютеров с помощью веб-браузера. Такой подход делает мобильное устройство еще одним IoT-устройством.

В непосредственной близости для обмена данными между устройствами и координатором умного дома применяются такие протоколы, как Zigbee, Z-Wave и другие. Однако для связи координатора со смартфоном эти протоколы не подходят, так как они имеют ограниченный радиус действия, не поддерживаются в большинстве мобильных устройств и требуют специального оборудования для приема и передачи данных. Для этих целей используется подключение к интернету с использованием протокола MQTT.

Протокол MQTT, разработанный на основе TCP/IP, оптимизирован для потоковой передачи данных в сетях с низкой пропускной способностью, что делает его особенно востребованным при создании Android-приложений для удаленного управления системой «умный дом». Его преимущества перед другими протоколами и ключевые особенности включают:

* поддержка сохраненных сообщений (retained message), что позволяет новым клиентам получать актуальную информацию сразу после подключения;
* для работы в условиях нестабильного подключения реализован механизм QoS (Quality of Service) который обеспечивает надежность доставки сообщений;
* имеет поддержку функционала LWT (Last Will and Testament, «последняя воля и завещание»), что используется для оповещения брокера о нештатном отключении клиента;
* минимальная задержка передачи данных, благодаря легковесности передаваемых сообщений.

Система связи, использующая протокол MQTT (см. рисунок 1.3), включает в себя издателей (publisher), сервер-брокер (MQTT broker) и одного или нескольких клиентов (subscribers). Издателю не требуется дополнительная настройка для определения числа или местоположения подписчиков, которые будут получать сообщения. Также подписчики не нуждаются в настройке для определения конкретного издателя. Издатель отправляет 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.



Рисунок 1.3 – Схема взаимодействия по протоколу MQTT

В MQTT данные передаются в виде payload, которые могут быть любыми байтовыми данными. Это означает, что информация может быть представлена в виде строки, JSON, двоичных данных или даже изображений – все зависит от того, что отправитель и получатель согласуют. Однако, в большинстве случаев, данные передаются в формате JSON, так как этот формат легко обрабатывается, обладает хорошей читаемостью и широко используется для обмена структурированными данными в приложениях, связанных с IoT и умными устройствами.

После отправки, данные хранятся в топиках, которые составлены из символов в кодировке UTF8 и имеют структуру, схожую с деревом файловых UNIX подобных операционных систем. Данный подход обеспечивает дополнительную читаемость для пользователей. В некоторых случаях устройства отправляют данные в топики, которые соответствует их адресу, например:

* home/kitchen/plug;
* home/restroom/light/ceiling;
* zigbee/0xA4C138DEB58ECDEC.

Данная структура топиков изначально задана в устройстве, но может быть изменена пользователем при необходимости.

Организация топиков с использованием дружественных имен для устройств позволяет наглядно отслеживать передаваемые данные и упрощает разработку и отладку кода, не требуя запоминания числовых адресов расположения данных.

Для взаимодействия устройств с брокером предусмотрены различные типы сообщений, ниже перечислены основные типы для функционирования системы:

* connect – установка соединения клиента с брокером;
* disconnect – разрыв соединения клиента с брокером;
* publish – публикация данных клиента в топик;
* subscribe – подписка клиента на получение сообщений из топика.

Таким образом, протокол MQTT является отличным средством для обмена данными в системах «умный дом». Его простота и гибкость позволяют легко интегрировать устройства с различными уровнями взаимодействия. Возможность использования структуры топиков, которая напоминает файловую систему, делает обмен данными более удобным и читаемым для пользователя. MQTT также поддерживает различные уровни качества обслуживания (QoS), что обеспечивает надежность передачи сообщений даже в условиях нестабильной сети. Эти особенности делают MQTT идеальным выбором для создания масштабируемых и устойчивых решений в умных домах, промышленности и других областях.

* 1. Интегрированная среда разработки Android Studio

Android Studio включает в себя редакторы кода, визуальные инструменты для разработки пользовательских интерфейсов, отладчики и профилировщики производительности, а также средства для автоматизации и управления проектами.

Можно устанавливать и отлаживать приложения как в эмуляторе, так и на физических устройствах, подключенных по USB или беспроводным способом. В режиме отладки Android Studio предоставляет подробную информацию о работе приложения, позволяя приостанавливать его выполнение, анализировать состояние переменных и инспектировать пользовательский интерфейс.

Среда также включает библиотеку инструментов для работы с базами данных, мультимедийными файлами, сетевыми протоколами и графикой. Функция Live Edit позволяет мгновенно просматривать изменения в пользовательском интерфейсе Jetpack Compose без необходимости полной перекомпиляции приложения. Это помогает разработчикам оперативно тестировать идеи и настраивать внешний вид, а также проверять адаптацию интерфейса для разных устройств, ориентаций экрана и цветовых схем.

* 1. Архитектура приложений

Разработка Android-приложений опирается на несколько архитектурных подходов, которые помогают организовать код, упростить поддержку и улучшить масштабируемость проекта. Архитектура определяет принципы разделения ответственности между компонентами приложения, что особенно важно в условиях сложных пользовательских интерфейсов и многопоточных операций.

Среди наиболее распространенных архитектур для Android можно выделить MVC (Model-View-Controller) и MVVM (Model-View-ViewModel). Эти модели помогают структурировать код так, чтобы интерфейс, логика обработки данных и бизнес-логика были четко разграничены, что снижает зависимость компонентов и делает приложение более гибким.

* + 1. Model-View-Controller (MVC) – это архитектурный паттерн, который является основой для разработки приложений. Он обеспечивает разделение приложения на три основных взаимодействующих компонента: модель (Model), представление (View) и контроллер (Controller) представленных на рисунке 1.4.

Использование паттерна MVC позволяет создавать гибкие и масштабируемые приложения, где каждый компонент отвечает за свою часть функциональности.

Модель отвечает за управление данными и их обработку, включая реализацию бизнес-логики, проверку данных и другие связанные операции. Она не взаимодействует напрямую с представлением и контроллером, не имея доступа к их методам, переменным и состояниям. Взаимодействие с моделью осуществляется через заранее определенные интерфейсы, что позволяет полностью отделить ее от пользовательского интерфейса и сделать более универсальной.



Рисунок 1.4 – Архитектурный паттерн MVC

Представление отображает данные пользователю и предоставляет возможность взаимодействия с приложением через графический интерфейс. В ней определяется внешний вид и макет интерфейса, это происходит в графическом редакторе. Представление не имеет представления о существовании модели и контроллера.

Контроллер является посредником между моделью и представлением. Он получает информацию от представления, обрабатывает ее и вносит соответствующие изменения в модель. Таким образом, контроллер обеспечивает взаимодействие и согласованность между моделью и представлением, делая возможным реализацию функциональности приложения. Чаще всего контроллер является наиболее наполненным исходным кодом элементом приложения.

В контексте разработки Android-приложений паттерн MVC может использоваться для структурирования кода и разделения ответственности между компонентами. Однако в Android этот паттерн встречается реже, так как система активностей и фрагментов уже включает элементы контроллера и представления.

Хотя MVC может быть использован в Android, чаще применяются другие архитектурные паттерны, такие как MVVM (Model-View-ViewModel), которые лучше соответствуют особенностям Android-разработки и обеспечивают лучшую разделяемость кода.

* + 1. Model-View-ViewModel (MVVM) является схожим с MVC паттерном, используемым для разделения компонентов приложения на три основных уровня: Модель, Представление и Модель Представления (ViewModel).

Одним из недостатков MVVM является некоторая его избыточность для простых приложений, которые имеют небольшое количеством логики и данных. MVVM требует больше кода, чем, например, простая парадигма Model-View-Controller. При этом эти недостатки нивелируются при разработке относительно больших приложений, так как MVVM предоставляет огромные возможности для повторного использования различных элементов Представления. На начальных этапах разработка приложения, использующего платформу MVVM, может быть затратной с точки зрения количества кода и времени, однако повторное использование и универсальность стандартизированных представлений упрощают поддержку и расширение проекта в будущем.

Структурная схема взаимодействия компонентов архитектурного паттерна MVVM представлена на рисунке ниже:



Рисунок 1.5 – Архитектурный паттерн MVVM

Модель, аналогична модели MVC, не имеет информации о представлении и модели представления.

Представление, частично похоже на MVC, отображает данные пользователю и обрабатывает пользовательский ввод. При это в отличии от MVC, представление напрямую обращается к модели представления, последняя чаще всего является полем представления.

Модель представления является посредником между моделью и представлением и чем-то схожа с контроллером MVC. Модель имеет общее с Представляем данные, которые автоматически обновляются и модели представления и в самом представлении при изменении их как стороны пользователя, так и со стороны модели. В MVC же контроллер при обновлении данных запускает метод обновления Представления. В каком-то смысле представление совмещает контроллер и представление MVC. Модель представления также может обслуживать несколько представлений одновременно.

* 1. Язык программирования Kotlin

Kotlin – это современный язык программирования со статической типизацией. Разработкой языка занимается компания JetBrains. Он ориентирован на работу с JVM (Java Virtual Machine) и является полностью совместимым с Java, что позволяет использовать библиотеки и фреймворки разработанных для Java без значительных изменений в кодовой базе. Kotlin имеет удобочитаемый и краткий синтаксис, обеспечивает повышенную надежность кода и предлагает удобные конструкции, которые способствуют сокращению числа ошибок в процессе разработки.

Одним из главных достоинств языка программирования Kotlin является его защищенность при работе с null-значениями. В отличие от Java, в Kotlin реализована строгая типизация, исключающая ошибки при попытке обратиться к объекту, значение которого является null. Кроме того, язык поддерживает функциональный подход, позволяя применять анонимные функции, усовершенствованные коллекции и другие удобные механизмы для обработки данных.

Kotlin активно используется в разработке Android-приложений и официально поддерживается Google как основной язык разработки приложения для операционных систем Android.

* 1. Jetpack Compose

Jetpack Compose – это набор инструментов от Google для разработки пользовательских интерфейсов для системы Android, основанный на декларативном подходе, что позволяет описывать пользовательский интерфейс (UI) с помощью функций, а не традиционных XML-разметок. Это упрощает разработку, делает код более читаемым и снижает вероятность ошибок.

Одно из ключевых достоинств Jetpack Compose – его высокая адаптивность и структурированность. Элементы интерфейса, представленные в виде Composable-функций, можно без труда повторно использовать, комбинировать и модифицировать без необходимости заново выстраивать всю структуру экрана. Помимо этого, Jetpack Compose хорошо сочетается с архитектурными подходами, такими как MVVM, что делает его удобным решением для разработки современных Android-приложений.

Compose полностью управляет состоянием UI, что позволяет разрабатывать динамические и отзывчивые интерфейсы без сложных манипуляций с View. Разработчики могут использовать State и MutableState для отслеживания изменений и автоматического обновления экрана. Это значительно упрощает работу с анимациями, пользовательскими действиями и изменениями данных.

Еще одно важное преимущество – интеграция с существующими Android-компонентами. Jetpack Compose совместим с View-based UI, поэтому его можно постепенно внедрять в проекты, работающие на старых версиях Android. Кроме того, Google активно развивает Compose, добавляя новые возможности и улучшая производительность.

Данный фреймворк предоставляет мощные инструменты для отладки и предпросмотра интерфейсов. Благодаря функции Preview разработчики могут визуализировать компоненты прямо в Android Studio без запуска эмулятора или физического устройства. Это ускоряет итерации: изменения в коде мгновенно отражаются в превью, а поддержка тем, параметров и локалей позволяет тестировать адаптивность интерфейса в разных сценариях. Кроме того, встроенные инструменты вроде Layout Inspector помогают анализировать иерархию элементов, находить перерисовки и оптимизировать производительность, что особенно важно для сложных анимаций и динамических данных.

Jetpack Compose также поддерживает Material Design 3, что позволяет легко использовать современные UI-компоненты и стили. Это ускоряет процесс разработки и делает приложения более визуально привлекательными. В сочетании с другими библиотеками Jetpack (например, Navigation, Paging, Room) Compose становится мощным инструментом для создания надежных и масштабируемых Android-приложений.

1. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После анализа предметной области разрабатываемого приложения были сформулированы основные требования, которые необходимо учитывать при его реализации. Поскольку приложение включает обширный функционал, включая работу с базой данных, обработку и отправку данных MQTT-брокеру, работа в локальной сети с использованием HTTP, возникает необходимость в проектировании структурной схемы программного средства.

Структура проекта состоит из следующих блоков:

* блок MQTT клиента;
* блок соединения с брокером;
* блок коммуникации MQTT;
* блок обработки и генерации сообщений;
* блок базы данных;
* блок моделей;
* блок бизнес-логики;
* блок моделей представления;
* блок представления.

Структура системы была организована таким образом, чтобы каждый блок выполнял строго определенную задачу и обеспечивал стабильную работу приложения в целом.

Далее будет рассмотрен принцип работы каждого из выделенных блоков, их основные функции, а также взаимодействие между ними. Взаимосвязь между основными блоками проекта отражена на структурной схеме ГУИР.400201.060 С1.

* 1. Блок MQTT клиента

Блок MQTT клиента отвечает за первичную настройку соединения, обеспечивая корректное взаимодействие с брокером. В рамках этой настройки задаются основные параметры подключения, включая адрес брокера, номер порта, уникальный идентификатор клиента, а также параметры безопасности, такие как логин и пароль.

Дополнительно можно задать интервал Keep Alive, который определяет, как часто клиент будет отправлять сигналы активности брокеру, предотвращая разрыв соединения. Также можно настроить механизм Last Will and Testament (LWT) – заранее заданное сообщение, которое брокер опубликует в случае неожиданного отключения клиента. Это позволяет другим участникам системы получать актуальную информацию о его состоянии.

Кроме того, при настройке соединения можно указать, должна ли сессия быть очищаемой (Clean Session). Если параметр включен, при каждом новом подключении клиент начинает работу с нуля, без сохраненных подписок. Если отключен, брокер запоминает подписки и недоставленные сообщения, обеспечивая более надежную связь при временных разрывах соединения.

При первом подключении системы «умного дома» данные для подключения вводятся пользователем через блок представления. Вводимые параметры, такие как адрес брокера, логин, пароль и другие настройки, проверяются на корректность и сохраняются в базе данных только при успешном вводе и установлении соединения. Это обеспечивает их последующее использование и предотвращает сохранение некорректных данных. При следующем запуске приложения блок MQTT клиента загружает сохраненные параметры из базы данных и автоматически устанавливает соединение с брокером, исключая необходимость повторного ввода данных. Такой подход повышает удобство работы пользователя и снижает вероятность ошибок, связанных с ручной настройкой.

Благодаря этому механизму обеспечивается не только удобное управление подключением, но и стабильность работы системы, так как параметры сохраняются и могут быть использованы для восстановления соединения в случае сбоя.

* 1. Блок соединения с брокером MQTT

Блок соединения с MQTT-брокером отвечает за установление и поддержание стабильного соединения между клиентом и брокером. Его главная задача – инициировать подключение, следить за его состоянием и восстанавливать связь в случае разрыва.

При запуске системы блок загружает сохраненные параметры подключения, такие как адрес брокера, номер порта, логин, пароль, уникальный идентификатор клиента, а также дополнительные настройки, включая Keep Alive, Clean Session и LWT.

Далее создается экземпляр MQTT-клиента с заданными параметрами, после чего он отправляет запрос на подключение к брокеру, используя аутентификацию по логину и паролю. Если соединение успешно установлено, блок уведомляет другие компоненты системы, в частности блок коммуникации MQTT, который затем подписывается на нужные топики и начинает обмен сообщениями.

Для поддержания соединения используется механизм Keep Alive, который позволяет клиенту периодически отправлять сигналы активности, предотвращая разрыв связи. Если же соединение все же прерывается, блок оперативно обнаруживает это и инициирует процедуру переподключения. При повторных неудачах используется механизм увеличения задержки между попытками восстановления связи, чтобы снизить нагрузку на используемую сеть и брокер.

При каждом успешном подключении блок соединения может сохранять актуальные параметры в базе данных, чтобы использовать их при следующих запусках системы. Это позволяет исключить необходимость повторного ввода данных вручную.

Таким образом, блок соединения с MQTT-брокером выполняет ключевую функцию в системе, обеспечивая надежное подключение и автоматическое восстановление связи при сбоях.

* 1. Блок коммуникации MQTT

Блок коммуникации MQTT играет ключевую роль в системе «умного дома», обеспечивая обмен сообщениями между приложением и удаленным брокером. Его основная задача – установление защищенного соединения, подписка на топики и публикация сообщений, что позволяет передавать данные о состоянии устройств и отправлять им управляющие команды.

После подключения к брокеру с использованием аутентификации блок подписывается на определенные топики, что дает возможность получать актуальную информацию о состоянии устройств. При поступлении сообщения от брокера блок передает его в блок обработки и генерации сообщений, который анализирует содержимое и определяет дальнейшие действия. Если данные содержат информацию о новом состоянии устройства, они передаются в пользовательский интерфейс для отображения актуальной информации.

Кроме приема сообщений, блок коммуникации выполняет отправку данных, обеспечивая управление устройствами. Когда пользователь изменяет параметры в приложении, например, включает освещение или задает температуру, блок обработки формирует соответствующую команду и передает ее в блок коммуникации, который публикует сообщение в нужный топик MQTT. Это позволяет системе обеспечивать двусторонний обмен данными, необходимый для стабильной работы «умного дома».

Для поддержания соединения блок включает механизмы обработки ошибок и восстановления связи. В случае разрыва соединения он уведомляет блок соединения с брокером, который предпринимает попытки восстановления. Также он контролирует целостность передаваемых данных, предотвращая некорректные сообщения и дублирование команд.

* 1. Блок обработки и генерации сообщений

Блок обработки и генерации сообщений играет самую важную роль в системе, обеспечивая корректное взаимодействие между приложением и устройствами «умного дома» через MQTT-протокол. Он выполняет обработку входящих сообщений от брокера, анализируя их содержимое, и формирует исходящие команды для управления устройствами. Основная цель блока – преобразование сырых данных в понятный формат, фильтрация лишней информации и передача обработанных данных в соответствующие компоненты системы.

При получении сообщения от MQTT-брокера блок обработки и генерации анализирует его, проверяет на наличие ошибок и определяет, к какому устройству оно относится. Если данные содержат информацию о состоянии устройства, например, текущую температуру, уровень освещенности или статус реле, они передаются в пользовательский интерфейс для отображения актуальной информации.

Кроме обработки входящих сообщений, этот блок также отвечает за генерацию исходящих данных. Когда пользователь в приложении изменяет параметры работы устройства, блок обработки получает соответствующий запрос, преобразует его в формат MQTT-сообщения и передает в блок коммуникации для отправки брокеру. Это позволяет осуществлять двусторонний обмен данными между клиентом и устройствами «умного дома».

Дополнительно блок включает механизмы фильтрации и валидации данных, предотвращая отправку некорректных команд и обеспечивая корректную работу системы. В случае обнаружения ошибки он может отправлять уведомления в интерфейс пользователя или запускать механизмы повторной передачи сообщения. Все это делает блок обработки и генерации сообщений важным элементом системы, который обеспечивает стабильную и надежную работу обмена данными через MQTT.

* 1. Блок базы данных

Информация, получаемая из блока бизнес-логики после обработки сообщений от брокера, записывается в базу данных, где она структурируется в таблицах, предназначенных для хранения данных о системе и подключенных устройствах. В качестве хранилища используется SQLite с библиотекой Room, обеспечивающей удобную работу с базой данных через объектно-реляционное отображение (ORM).

Блок базы данных отвечает за сохранение параметров подключения к удаленному брокеру сообщений, включая адрес брокера, номер порта, учетные данные, уникальный идентификатор клиента, а также дополнительные настройки, такие как Keep Alive, Clean Session и Last Will and Testament. Это позволяет автоматически восстанавливать соединение после перезапуска приложения.

В таблицах хранятся сведения о подключенных устройствах, включая уникальный сетевой адрес Zigbee, последние полученные значения от MQTT-брокера, пользовательские настройки (например, дружественное имя для устройства), параметры управления (каналы переключения реле, регулировки освещения) и тип устройства. Для ускорения работы приложения и снижения нагрузки на сеть используется кеширование данных, позволяющее загружать актуальные параметры устройств без необходимости частых запросов к брокеру. Механизмы обновления и удаления данных реализованы через Room API с учетом каскадных изменений и целостности связей между таблицами.

* 1. Блок моделей

Блок моделей отвечает за структуру данных, используемых в приложении. Он содержит классы, описывающие подключенные устройства умного дома, а также параметры подключения к MQTT-брокеру. Модели обеспечивают единообразное представление данных, используемых в бизнес-логике, базе данных и пользовательском интерфейсе.

Данные об устройствах включают уникальный идентификатор, тип устройства, текущее состояние, доступные команды и другие параметры, необходимые для взаимодействия. Также в моделях хранятся сведения о подключении к MQTT-брокеру, такие как адрес сервера, логин, пароль и настройки соединения.

Модели упрощают обработку и передачу данных между различными компонентами системы, обеспечивая их согласованность и целостность. Кроме того, они позволяют эффективно управлять состоянием устройств и взаимодействовать с MQTT-брокером для получения и отправки данных в реальном времени.

* 1. Блок бизнес-логики

Блок бизнес-логики играет ключевую роль в обеспечении стабильной работы приложения, связывая между собой все основные компоненты системы. Он отвечает за обработку данных, управление MQTT-сообщениями, взаимодействие с базой данных и передачу данных в модель представления. Вся логика работы с устройствами умного дома, включая обработку пользовательских команд, обновление состояний и выполнение автоматических сценариев, сосредоточена именно здесь.

Для повышения надежности блок бизнес-логики реализует механизмы обработки ошибок и повторной отправки сообщений в случае сбоев соединения. Он также контролирует целостность данных, предотвращая их дублирование или потерю. Благодаря такому подходу обеспечивается бесперебойная работа системы и точное выполнение всех команд пользователя.

* 1. Блок представлений

Блок представлений отвечает за отображение данных и взаимодействие пользователя с системой «умного дома». В проекте используется Jetpack Compose, который позволяет создавать гибкий и переиспользуемый UI, адаптируемый под разные устройства. Представления работают в тесной связке с ViewModel, которая управляет состоянием, обрабатывает события и передает данные в интерфейс. Это обеспечивает четкое разделение ответственности, где UI остается декларативным, а бизнес-логика сосредоточена во ViewModel.

Одним из важных аспектов является тестируемость представлений. ViewModel можно проверять отдельно, используя инструменты для работы с потоками данных, а UI тестируется с помощью Compose UI Testing. Это позволяет выявлять ошибки на ранних этапах разработки. Кроме того, в Jetpack Compose предусмотрены механизмы обработки ошибок, такие как отображение уведомлений и диалогов, что помогает улучшить пользовательский опыт. Благодаря такому подходу представления остаются простыми, удобными в сопровождении и эффективно взаимодействуют с бизнес-логикой приложения.

* 1. Блок моделей представлений

В Jetpack Compose корневым представлением является композиционная функция (Composable), с которой начинается работа приложения. Это представление содержит ссылки на другие представления, формируя иерархическую структуру интерфейса. При этом всегда возможна навигация назад, а также циклические переходы, когда одно представление может ссылаться на себя.

В Jetpack Compose любое отображаемое на экране содержимое является представлением (Composable). Например, кнопка – это отдельное представление, описанное декларативным способом, а содержащий ее список также является представлением. Таким образом, одно представление может включать в себя множество других, а благодаря механизму параметров при инициализации обеспечивается гибкость и переиспользуемость компонентов.

Представления должны работать одинаково на всех устройствах Android и адаптироваться под разные размеры экранов. Для этого используется механизм адаптивного пользовательского интерфейса, обеспечивающий корректное отображение элементов на смартфонах, планшетах и других устройствах.

Представление взаимодействует с моделью представления (ViewModel), получая из нее данные и отправляя события на обработку. Однако допускается выполнение простых вычислений непосредственно в представлениях. В некоторых случаях, когда представление содержит только статические данные или не требует сложной логики, оно может работать напрямую с моделью данных без использования ViewModel. Это особенно актуально для простых компонентов, таких как кнопки или отдельные элементы списка.

Представления также могут взаимодействовать с Composition Local, получая данные из общего контекста, что упрощает управление состоянием и передачу зависимостей. Функциональность представлений в системе «умного дома»:

1. Пользовательский опыт. Важно учитывать удобство использования интерфейса: логичную навигацию, интуитивные элементы управления и четкую обратную связь при взаимодействии с системой.
2. Визуализация данных. Jetpack Compose позволяет создавать динамические и адаптивные интерфейсы для отображения данных. Это могут быть графики, диаграммы, карточки или таблицы, отображающие текущее состояние устройств системы «умного дома».
3. Элементы управления, такие как Button, Switch, Slider, TextField, позволяют пользователю управлять устройствами «умного дома». Использование состояний (State) и событий (Event) обеспечивает интерактивность и отзывчивость интерфейса.

Таким образом, использование Jetpack Compose в разработке интерфейса приложения «умного дома» позволяет создавать гибкие, адаптивные и удобные представления, обеспечивая качественный пользовательский опыт.

1. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Функциональное проектирование делает упор на создании корректно работающего приложения, определяя его ключевые возможности и структуру. В этом разделе рассматриваются логические блоки системы, используемые в них классы и методы. Также представлены диаграмма классов на чертеже ГУИР.400201.060 РР.1 и диаграмма последовательности на чертеже ГУИР.400201.060 PP.2.

* 1. Представления

Разработка приложения производится с использованием фреймворка Jetpack Compose, который основан на декларативном и функциональном подходе к созданию пользовательского интерфейса. В рамках данного подхода представления строятся с помощью функций-компонентов (Composable), которые возвращают описание UI-элементов. Каждый экран приложения представлен в виде набора таких функций, которые принимают параметры, управляют состоянием и отрисовывают элементы интерфейса.

Класс AppNavHost отвечает за управление стеком навигации, позволяя пользователю перемещаться между экранами и возвращаться назад без потери состояния. NavHostController используется для выполнения переходов и передачи параметров между экранами, что обеспечивает динамическую и удобную навигацию.

Каждый маршрут внутри NavHost связан с соответствующим экраном, а передача ViewModel в composable() гарантирует, что данные остаются актуальными при смене экранов. Такой подход позволяет эффективно управлять состоянием приложения и упрощает работу с пользовательским интерфейсом.

Основные маршруты:

1. Экран авторизации (Routes.AUTH\_SCREEN) – первый экран, если пользователь не авторизован. Передает sensorsViewModel и devicesViewModel в AuthorizationScreen для инициализации обработчика сообщений MQTTMessageHandler и управления подключением к MQTT-брокеру.
2. Главный экран (Routes.HOME\_SCREEN) – основной интерфейс пользователя для управления системой «умный дом». Здесь отображаются списки доступных устройств и комнат, полученные из devicesViewModel и roomsViewModel**.**
3. Экран деталей устройства (Routes.DEVICE\_DETAILS\_SCREEN) – получает backStackEntry для обработки параметров навигации и передает devicesViewModel и roomsViewModel для работы с конкретным устройством.
4. Экран деталей комнаты (Routes.ROOM\_DETAILS\_SCREEN) – также использует backStackEntry, передает navHostController и ViewModel, чтобы загружать информацию о комнате и ее устройствах.

Данный класс обеспечивает централизованное управление навигацией, передавая общие ViewModel в экраны, что позволяет эффективно работать с данными без дублирования их экземпляров. Это улучшает производительность приложения и упрощает управление состоянием, так как модели представлений остаются активными при переходах между экранами.

* + 1. Функция AuthorizationScreen отвечает за графический интерфейс экрана авторизации в приложении, предназначенном для управления системой «умный дом». Она предоставляет пользователю возможность ввести учетные данные MQTT-брокера, необходимые для установления соединения с сервером.

Основные компоненты:

1. Элемент Image используется для отображения логотипа приложения посредине в верхней части экрана.
2. Форма BrokerInputForm включает поля для ввода адреса (URI), порта, имени пользователя и пароля, а также кнопку (Button), вызывающую функцию onAddBroker для сохранения введенных данных и добавления нового брокера в приложении.
3. Список BrokerList отображает последний MQTT-брокер, к которому производилось подключение, с помощью BrokerItem, позволяя повторно подключиться (onLogin) или удалить запись (onDelete).
4. Компонент BrokerItem оформляет информацию о брокере в Card, включая Text для отображения URI, порта, а также (если указаны) логина и замаскированного пароля. Внизу расположены две кнопки: кнопка для подключения к брокеру и кнопка для удаления записи.

Данный интерфейс предоставляет возможность добавить информацию для подключения к новому MQTT-брокеру, используя соответствующие учетные данные, а при повторном входе – без ввода данных подключиться к последнему использованному брокеру или выбрать другой из списка сохраненных.

HomeScreen является основным экраном приложения, предоставляющий удобный интерфейс для навигации между ключевыми разделами: устройства, комнаты и настройки. Пользователь может легко переключаться между вкладками, что обеспечивает быстрый доступ к необходимым данным без задержек и перезагрузки.

Функциональные особенности:

* горизонтальная навигация реализована через LazyRow, где каждая вкладка представлена компонентом TabButton, визуально выделяющим активный раздел;
* состояние активной вкладки отслеживается через переменную selectedTab, а previousTab используется для анимации плавных переходов;
* динамическое обновление контента осуществляется с помощью AnimatedContent, который меняет отображаемый экран в зависимости от текущей вкладки;
* анимации переходов включают эффекты slideInHorizontally, slideOutHorizontally, fadeIn и fadeOut, создавая плавный и интуитивный пользовательский опыт;
* адаптация интерфейса под различные размеры экранов: если ширины экрана недостаточно, LazyRow позволяет прокручивать вкладки;
* оптимизация производительности, позволяющая избежать лишних перерисовок и снизить нагрузку на устройство, что особенно важно для слабых смартфонов.

Благодаря этим особенностям HomeScreen обеспечивает удобную и быструю работу с основным функционалом приложения.

Интерфейс DevicesScreen отвечает за отображение списка устройств, доступных в системе, и предоставляет пользователю возможность управления их состоянием.

Основные компоненты экрана:

* Scaffold – используется в качестве контейнера, обеспечивающего корректное размещение элементов и управление отступами;
* Column – главный контейнер, содержащий все элементы экрана и обеспечивающий вертикальное расположение компонентов;
* вертикальная прокрутка verticalScroll() позволяет комфортно просматривать список устройств при их большом количестве;
* DeviceCard – карточка устройства, содержащая изображение, название, тип устройства и его текущее состояние с возможностью управления.

Для загрузки данных используется devicesViewModel.devices, который с помощью collectAsState() получает актуальный список всех устройств. Фильтрация по типу выполняется с помощью функции getDevicesByTypeFlow(type), где в качестве аргумента указывается тип устройства, а актуальные состояния устройств отслеживаются в DeviceState.devicesData, что позволяет оперативно обновлять информацию.

Динамическое обновление интерфейса обеспечивается автоматическим реагированием UI на изменения в данных devicesViewModel.devices, благодаря чему состояние устройств остается актуальным без необходимости ручного обновления. Текущее состояние каждого устройства извлекается из deviceState, что гарантирует мгновенное отображение изменений.

SettingsScreen — это функция, описывающая экран настроек, отвечающий за управление дополнительными параметрами и средствами в системе умного дома. В частности, здесь реализовано управление некоторыми настройками шлюза, включая режим поиска устройств.

В данном представлении реализован функционал:

* включение и отключение режима обнаружения новых устройств «умного дома»;
* отображение времени работы режима обнаружения;
* управление настройками подсветки координатора.

Экран настроек состоит из следующих компонентов:

* основной контейнер (Column) – обеспечивает расположение элементов в столбец с отступами между ними;
* карточки (Card) – используется для отображения различных функциональных компонентов системы;
* переключатель (Switch) – позволяет производить управление переключаемых параметров координатора;
* текстовые элементы (Text) – отображают информацию о работе координатора.

Функция onDiscoverySwitchChanged(enabled: Boolean) отвечает за управление режимом обнаружения новых устройств. При передаче true вызывается функция startDiscovery(), иначе — stopDiscovery(), прекращающая поиск.

Функция LaunchedEffect(discoveryState) асинхронно отслеживает изменение состояния поиска (discoveryState). Если поиск включен, переменная remainingTime инициализируется значением 255, соответствующее 255 секундам, затем в цикле каждую секунду значение уменьшается на единицу функцией управления задержками delay(1000), пока не достигнет нуля. Если поиск отключается, значение remainingTime устанавливает значение 0.

ФункцияRoomsScreenреализует функционал управления системой группирования устройств по их территориальному расположению в помещении. Данный пользовательский интерфейс позволяет просматривать список существующих комнат, отображать количество устройств в каждой из них и добавлять новые комнаты в систему.

В данной функции описаны следующие атрибуты:

* showDialog – переменная состояния, которая отвечает за отображение диалогового окна добавления новой комнаты: если true, отображается диалоговое окно AddRoomDialog, иначе компонент скрыт;
* rooms – коллекция комнат, полученная из roomsViewModel, используемая для построения списка комнат в LazyColumn.

Для добавления новой комнаты используется компонент FloatingActionButton для отображения кнопки, которая при нажатии изменяет состояние showDialog на true, вызывая отображение AddRoomDialog.

После ввода названия комнаты и подтверждения действий, вызывается функция roomsViewModel.addRoom(roomName), которая добавляет новую комнату в список. После этого состояние отображения диалогового окна устанавливается в false.

Интерфейс DeviceDetailScreen представляет собой пользовательский интерфейс для просмотра подробной информации о всех данных предоставляемыми IoT-устройством системы. Он включает в себя элементы для изменения дружественного имени просматриваемого устройства, назначения его в определенную комнату и просмотра текущих параметров устройства.

Атрибуты, описанные в интерфейсе:

* deviceId – идентификатор текущего устройства, получаемый из backStackEntry в результате навигации между экранами приложения;
* deviceData – информация о состоянии текущего устройства, получаемого из объекта, отвечающего за хранение показаний устройств;
* rooms – список всех комнат, для возможности переназначения отношения к какой-либо комнате;
* expanded – переменная состояния для управление выпадающим списком при выборе комнат;
* showEditDialog – состояние, управляющее отображением диалогового окна для редактирования дружественного имени устройства;
* selectedRoom – переменная состояния для хранения информации о выбранной комнате.

Основные графические компоненты интерфейса:

* DeviceTitle – компонент, отображающий текущее имя устройства и кнопку для редактирования имени устройства;
* EditDialog – всплывающее окно для редактирования информации;
* ExposedDropdownMenuBox – это компонент, реализующий раскрывающееся меню для выбора комнат;
* ExposedDropdownMenu – компонент выпадающего списка для выбора комнаты;
* DropdownMenuItem – элементы списка выпадающего меню;
* DeviceDetails – composable-функция, используемая для отображения текущего состояния устройства.

Для реализации функционала в данном интерфейсе используются следующие функции:

* onEditClick – callback функция для открытия диалогового окна EditDialog при изменении имени устройства;
* updateDeviceName(deviceId, newName) – функция, которая вызывается при подтверждении нового имени устройства, обновляя его в DevicesViewModel;
* assignRoomToDevice(deviceId, roomId?) – функция, которая выполняет привязку устройства к выбранной комнате или снимает ее при выборе опции «Selected Room».

Таким образом, интерфейс DeviceDetailScreen предоставляет пользователю удобный способ просмотра данных об устройстве, а также позволяет изменять его имя и привязку к комнате.

* 1. Модели представлений

Взаимодействие пользовательского интерфейса с данными и бизнес-логикой осуществляется с использованием архитектурного компонента ViewModel. ViewModel отвечает за хранение и управление состоянием экрана, обеспечивая его устойчивость к изменениям конфигурации, таким как поворот экрана. Он предоставляет данные в Composable-функции представлений через механизмы хранения состояний, а также содержит методы для обработки пользовательских действий.

Для каждого представления реализован соответствующий фабричный класс ViewModelProvider.Factory, который отвечает за создание экземпляров классов с передачей необходимых зависимостей, обеспечивая инъекцию зависимостей и соблюдение принципов инверсии управления.

Класс AuthorizationViewModel реализует логику управления данными и состоянием экрана авторизации.

Основные атрибуты класса:

1. \_brokers – это приватное свойство типа MutableState<List<Broker>>, хранящее текущий список брокеров. Оно используется для внутреннего управления состоянием и позволяет изменять данные внутри ViewModel.
2. brokers – публичное свойство типа State<List<Broker>>, предоставляющее доступ к списку брокеров только для чтения. Оно обеспечивает реактивное обновление UI при изменении данных (например, при добавлении или удалении брокеров).

Методы класса:

1. addBroker() асинхронно добавляет нового MQTT-брокера в локальную базу данных и обновляет UI. Метод выполняется в корутине через viewModelScope.launch, создает объект Broker с переданными serverUri, serverPort, user и password (поддерживая анонимное подключение), затем сохраняет его в базе данных с помощью brokerDao.insert(), после этого вызывается метод loadBrokers(), чтобы загрузить обновленный список брокеров, что автоматически обновляет UI благодаря реактивному состоянию \_brokers.
2. deleteBroker() асинхронно удаляет информацию о брокере из базы данных и обновляет UI. Он выполняется в корутине, сначала отключает MQTT-клиент с помощью disconnect(), затем удаляет брокера через deleteBroker(broker), после чего вызывает loadBrokers().
3. handleLogin() устанавливает текущий брокер, создает MQTT-клиент и выполняет подключение. Он обновляет BrokerState с broker.id, затем переинициализирует клиент MQTTClient с указанным брокером и обработчиком сообщений. Если подключение успешно, вызывается функция onSuccess(). Далее асинхронно с использованием viewModelScope.launch(Dispatchers.IO) клиент подписывается на необходимые для работы топики.

DevicesViewModel это модель представления, которая служит для управления устройствами умного дома, обеспечивая их загрузку, обновление и взаимодействие с MQTT брокером.

Атрибут \_devices – это MutableStateFlow, хранящий список устройств и используемый для управления их состоянием. Он инициализируется пустым списком и изменяется внутри viewModelScope.launch, что позволяет выполнять загрузку устройств из базы данных в фоновом режиме и не блокировать основной поток выполнения программы. Для получения данных используется атрибут devices типа StateFlow.

Методы, используемые для работы с данными:

1. loadDevices(brokerId: Int) – асинхронно загружает устройства, связанные с брокером, подписываясь на Flow из базы данных. При обновлении данных обновляет \_devices, что автоматически отражается в пользовательском интерфейсе.
2. getDeviceIdByIeeeAddr(addr: String, callback: (Int?) -> Unit) – получает ID устройства по его адресу и передает результат в callback.
3. getDevicesByTypeFlow(type: String) – возвращает StateFlow с устройствами указанного типа. Фильтрует \_devices на основе команд, полученных через Flow.
4. getDevicesByRoomIdFlow(roomId: Int) – аналогично предыдущему методу фильтрует устройства по их типу, но дополнительно учитывает идентификатор комнаты roomId, обновляя resultFlow.
5. addDeviceIfNotExists(device: Device) – добавляет устройство в локальное хранилище, если его там нет. Проверяет наличие устройства по его физическому адресу ieeeAddr, при отсутствии добавляет новую запись с информацией об устройстве.
6. updateDeviceName(deviceId: Int, newName: String) – изменяет имя устройства, обновляя его в базе и списке \_devices.
7. addCommandIfNotExists(command: Command) – добавляет команду в базе данных, если ее еще нет, проверяя по commandTopic.
8. sendCommandToMqtt(topic: String, command: String) – публикует команду управления устройством в заданный топик.
9. onToggle(topic: String, state: Boolean) – функция для обработки управляющих команд переключателей.
10. onValueChange(topic: String, value: Int) – функция для обработки управляющих команд устройств с регулируемыми параметрами в диапазоне от 0 до 100.
11. onSelectChange(topic: String, option: String) – функция для обработки управляющих команд устройств с выбором доступных опцией.
12. assignRoomToDevice(deviceId: Int, roomId: Long?) – обновляет roomId устройства в базе.
    1. Модели

В данном подразделе представлены таблицы моделей данных и связи между ними. Типы данных указаны в нотации Kotlin, как они описаны в исходном коде. Опциональность значений обозначается вопросительным знаком после типа данных.

Для создания сущности и соответствующей ей таблицы в базе данных используется аннотация @Entity. В ее параметрах указывается служебная информация для библиотеки Room, такая как:

* tableName – название таблицы;
* foreignKeys – внешний ключ, определяющий связи между таблицами;
* indicies – список индексов, ускоряющих поиск данных при большом объеме информации.

Описание модели состоит из двух частей:

* описание атрибутов таблицы – перечень столбцов и их характеристики;
* описание связей таблицы – указание внешних ключей, индексов и каскадных правил.

Класс Device представляет собой сущность базы данных, представленной в таблице devices, предназначенную для хранения информации об устройствах в системе «умного дома».

Описание атрибутов таблицы:

1. id: Int – первичный ключ таблицы, используемый для уникальной идентификации каждой записи. Генерируется автоматически при создании новой записи благодаря аннотации @PrimaryKey(autoGenerate = true).
2. ieeeAddr: String – уникальный физический адрес устройства, используемый для его идентификации в сети. Как правило, соответствует MAC-адресу или другому идентификатору, назначенному на аппаратном уровне.
3. friendlyName: String – читаемое название устройства, которое пользователь может задать вручную. Это название отображается в интерфейсе и упрощает взаимодействие с устройством, так как заменяет сложные идентификаторы удобным именем.
4. modelId: String – уникальный идентификатор модели устройства, назначаемый производителем. Используется для определения типа устройства, его функциональных возможностей и совместимости с другими компонентами системы.
5. topic: String – MQTT-топик, в который публикуются сообщения о состоянии устройства и через который осуществляется его управление. Позволяет организовать обмен данными между устройством и сервером, используя брокер MQTT.
6. roomId: Long? – идентификатор комнаты, в которой находится устройство. Если значение null, это означает, что устройство не привязано ни к одной конкретной комнате. Используется для логической группировки устройств по помещениям.
7. brokerId: Int – идентификатор MQTT-брокера, через которого осуществляется обмен данными. Позволяет системе поддерживать работу с несколькими брокерами и маршрутизировать сообщения к нужному серверу.

Связи между таблицами определены с помощью внешних ключей. Поле brokerId связано с id таблицы Broker, и при удалении брокера все связанные устройства также удаляются (с параметром onDelete = ForeignKey.CASCADE). Параметр roomId связан с id таблицы RoomEntity, и, если комната удаляется, у всех устройств, связанных с этой комнатой, значение roomId становится null (с параметром onDelete = ForeignKey.SET\_NULL).

Объект компаньон в классе Device используется для создания нового устройства с автоматическим формированием топика. Топик генерируется на основе физического адреса устройства в формате «zigbee/0x<ieeeAddr>».

Кроме того, в таблице используется индексация по полю roomId, что значительно ускоряет поиск устройств, принадлежащих определенной комнате.

Класс Broker представляет собой сущность базы данных, отображаемую в таблице brokers, предназначенную для хранения информации о MQTT-брокерах, используемых в системе «умного дома».

Поля класса Broker:

* id: Int – первичный ключ таблицы с автоматической генерацией;
* serverUri: String – URI MQTT-брокера, который используется для подключения к нему;
* serverPort: Int – порт, через который осуществляется подключение к брокеру;
* user: String? – имя пользователя для аутентификации при подключении к брокеру (может быть null, если аутентификация не требуется);
* password: String? – пароль пользователя для подключения к брокеру (может быть null, если аутентификация не требуется).

Поле id используется в таблице devices как внешний ключ (brokerId). Если брокер удаляется, все связанные с ним устройства также удаляются, что обеспечивается параметром onDelete = ForeignKey.CASCADE в определении внешнего ключа.

В таблице brokers можно хранить несколько брокеров, что позволяет системе «умного дома» работать с разными MQTT-серверами.

Класс Command представляет собой сущность базы данных, отображаемую в таблице commands, предназначенную для хранения информации о командах управления устройствами в системе «умного дома».

Атрибуты таблицы:

* id: Int – первичный ключ таблицы;
* deviceId: Int – внешний ключ, связывающий команду с устройством из таблицы devices. При удалении устройства все связанные с ним команды также удаляются;
* commandTopic: String – MQTT-топик, в который отправляются команды для управления устройством;
* payloadOn: String? – полезная нагрузка для включения устройства (может быть null, если команда не предназначена для переключателя);
* payloadOff: String? – полезная нагрузка для выключения устройства (может быть null, если команда не предназначена для переключателя);
* options: Map<String, String>? – список доступных опций для команд выбора (например, вариантов переключения между режимами устройства), для хранения в базе данных с использованием MapTypeConvertor;
* commandTemplate: String? – шаблон команды, который может быть использован для формирования сложных сообщений в MQTT (может быть null, если команда не требует шаблонизации);
* commandType: String – тип команды (например, switch, select, custom), определяющий логику ее обработки.

Поле deviceId связано с идентификатором таблицы devices. При удалении устройства все связанные с ним команды автоматически удаляются.

Таблица commands позволяет хранить команды управления устройствами умного дома, поддерживая как простые команды (например, включение и выключение устройств), так и более сложные (например, выбор режима работы устройства).

Класс RoomEntity представляет собой сущность, информация о которой хранится в таблице rooms.

Поля сущности:

* id: Long – идентификатор, выступающий в роли первичного ключа таблицы;
* name: String – название комнаты, которое задается пользователем для удобного восприятия в системе.

Поле id используется в таблице devices как внешний ключ (roomId). Если комната удаляется, у всех связанных с ней устройств значение roomId принимает значение null (onDelete = ForeignKey.SET\_NULL), что позволяет избежать удаления устройств из системы при удалении комнаты.

Таблица rooms позволяет группировать устройства по местоположению в помещении.

* 1. Хранение данных

В приложении используется локальная база данных на основе SQLite, управляемая через библиотеку Room. Это позволяет эффективно хранить и обрабатывать информацию обеспечивая удобный доступ к данным и их целостность.

Взаимодействие с хранимой информацией осуществляется через DAO-интерфейсы. Помимо постоянного хранения данных в базе, в приложении используются singleton-объекты, отвечающие за временное хранение состояний системы. Их основная задача – обеспечение быстрого доступа к актуальной информации без необходимости выполнения запросов к базе данных. Эти состояния существуют только во время работы приложения и не сохраняются в базе, поскольку их значения могут динамически изменяться в зависимости от текущего сеанса работы системы.

Класс RoomLocalDatabase является основной точкой доступа к базе данных SQLite, реализованной с использованием библиотеки Room.

Атрибуты класса базы данных задаются в качестве параметров аннотации @Database, в которой указываются:

* entities – список моделей используемых в базе данных (Broker, Device, Command);
* version – версия базы данных;
* exportSchema – флаг экспорта схемы, установленный в значение false, что означает создание таблиц производится на основе моделей, описанных в приложении.

Методы класса:

* brokerDAO(), deviceDAO(), commandDAO(), roomEntityDAO() – абстрактные методы, предоставляющие доступ к DAO-объектам для работы с таблицами brokers, devices, commands и rooms соответственно;
* getInstance() – метод, реализующий паттерн Singleton для создания и получения экземпляра сущности базы данных.

Класс BrokerState определен с помощью ключевого слова object, что делает его singleton-объектом – он создается один раз и существует в единственном экземпляре в рамках всего приложения.

Этот класс предназначен для хранения идентификатора, текущего MQTT-брокера, который используется в данный момент.

Основные компоненты данного класса:

* \_brokerId – приватное поле типа MutableStateFlow<Int?>, хранящее идентификатор брокера и позволяющее его изменять внутри методов класса;
* brokerId – публичное свойство типа StateFlow<Int?>, предоставляющее неизменяемый поток данных, содержащий идентификатор текущего брокера;
* setBrokerId(id: Int) – метод, обновляющий идентификатор брокера, устанавливая новое значение в \_brokerId.

Класс DeviceState так же определен с помощью ключевого слова object, что делает его singleton-объектом.

Данный класс используется для хранения состояний устройств «умного дома», содержащего актуальные данные, передаваемые устройствами.

Атрибуты класса:

* \_devicesData – приватное поле типа MutableStateFlow<Map<Int, Map<String, Any>>>, хранящее в себе данные всех устройств системы, где ключом верхнего уровня является идентификатор устройства, а значением – параметры устройств, представленных в виде структуры Map;
* devicesData – публичное свойство типа StateFlow<Map<Int, Map<String, Any>>>, предоставляющее неизменяемый поток данных с текущими состояниями устройств;
* updateDeviceData(deviceId: Int, payload: String) – метод, обновляющий данные конкретного устройства, принимая идентификатор устройства и строку в формате JSON;
* parseJson(json: String) – метод, преобразующий JSON-строку в структуру Map<String, Any> используемую для хранения состояния устройства;
* getDeviceValue(deviceId: Int, key: String) – метод, возвращающий конкретное значение из состояния устройства по его идентификатору и заданному ключу key.

Класс DiscoveryState определен ключевым словом object. Данный класс используется для управления процессом обнаружения и подключения устройств в систему «умный дом». Он содержит методы для запуска и остановки процесса, а также отслеживает его текущее состояние.

Атрибуты класса:

1. \_isDiscoveryActive – поле MutableStateFlow<Boolean>, которое хранит текущее состояние процесса обнаружения устройств. Если значение равно true, это означает, что процесс обнаружения активен, если false — процесс неактивен.
2. isDiscoveryActive – атрибут типа StateFlow<Boolean>, которое предоставляет доступ к текущему состоянию процесса обнаружения устройств. Это свойство является неизменяемым, что позволяет внешним компонентам подписываться на изменения состояния, но не изменять его напрямую.
3. resetJob – приватное поле типа Job? которое хранит ссылку на корутину, отвечающую за автоматическое завершение процесса обнаружения через определенный промежуток времени. Это поле может быть null, если процесс обнаружения неактивен.

Методы, описанные в классе:

1. Метод startDiscovery() устанавливает значение true для атрибута \_isDiscoveryActive, что указывает на начало процесса обнаружения устройств. Он отправляет MQTT-сообщение в топик DISCOVERY\_TOPIC со значением true, чтобы разрешить устройствам Zigbee подключаться к сети. Если существует предыдущая корутина, она отменяется. Затем запускается новая корутина, которая через DISCOVERY\_TIME автоматически завершает процесс обнаружения, устанавливая \_isDiscoveryActive в false.
2. stopDiscovery()устанавливает значение \_isDiscoveryActive в false, что указывает на завершение процесса обнаружения. Он отправляет MQTT-сообщение в аналогичный топик со значением false, чтобы запретить устройствам Zigbee подключаться к сети. Если текущая корутина существует, она отменяется.

Класс DeviceDAO представляет собой DAO-интерфейс (Data Access Object) для работы с таблицей устройств в базе данных. Он определен с помощью аннотации @Dao и содержит методы для взаимодействия с данными.

Основные методы:

* getDevicesByBroker(brokerId: Int) – получает список информации об устройствах, связанных с указанным брокером используя его ID;
* getDevicesByBrokerFlow(brokerId: Int) – возвращает поток (Flow), содержащий информацию об устройствах для заданного брокера по его идентификационному номеру;
* getAllDevices() – загружает информацию о всех устройствах, хранящиеся в базе данных;
* getDeviceByIeeeAddr(ieeeAddr: String) – ищет устройство с использованием его физического адреса (IEEE адрес), возвращает null, если устройство не найдено;
* getDeviceById(deviceId: Int) – получает устройство по его deviceId;
* insertDevice(device: Device) – добавление устройства в базу данных, игнорируя конфликты. Возвращает deviceId добавленной записи или -1, если устройство уже существует;
* getDevicesByRoomIdFlow(roomId: Int) – возвращает поток Flow, содержащий список устройств, принадлежащих указанной комнате (roomId);
* updateDevice(device: Device) – обновляет информацию об устройстве в базе данных;
* getDeviceCountForRoom(roomId: Long) – получает количество устройств, связанных с заданной комнатой (roomId).

Класс BrokerDAO является интерфейсом для взаимодействия приложения с таблицей, содержащей информацию о брокерах.

Методы класса:

* insert(broker: Broker) – добавляет брокера в базу или обновляет его при конфликте;
* getAllBrokers() – получает список всех брокеров;
* getLastBroker() – получает последнего добавленного брокера (по убыванию ID), возвращает null, если брокеров нет;
* deleteBroker(broker: Broker) – удаляет запись содержащую информацию об указанном брокере.

Класс CommandDAO представляет собой интерфейс, обеспечивающий доступ к данным команд, хранящимся в базе.

Методы, реализуемые в данном интерфейсе:

* getCommandByCommandTopic(commandTopic: String) – получает команду по заданной теме (commandTopic), возвращает null, если такой команды нет;
* getSwitchCommandByDeviceId(deviceId: Int) – ищет команду типа «switch» по deviceId, возвращает null, если команда не найдена;
* insertCommand(command: Command) – добавляет новую команду в базу или обновляет существующую при конфликте;
* getCommandsByTypeFlow(type: String) – возвращает Flow со списком команд заданного типа.

RoomEntityDAO – класс, предназначенный для управления данными о комнатах в базе данных.

Доступные функции для работы с данными:

* getAllRooms() – возвращает поток (Flow) со списком всех комнат;
* insertRoom(room: RoomEntity) – добавляет новую комнату или обновляет существующую при конфликте.

Класс MapTypeConvertor используется в базе данных Room для преобразования объектов типа Map<String, String> в строковый формат JSON и обратно. Это необходимо, так как SQLite, на котором основана Room, не поддерживает хранение сложных структур данных, таких как Map. SQLite работает с примитивными типами, поэтому для сохранения структур ключ-значение их нужно преобразовывать в строку.

В данном классе используется класс Gson для сериализации (fromMap) и десериализации (toMap) Map<String, String> в JSON. Метод fromMap превращает карту в строку JSON, которая может быть сохранена в базе данных как TEXT. Метод toMap выполняет обратную операцию — восстанавливает карту из JSON, что позволяет использовать ее в коде как обычный объект Map<String, String>. Такой подход позволяет удобно работать с Map в Room, несмотря на ограничения SQLite.

* 1. MQTT и взаимодействие с брокером

В данном разделе рассматривается организация взаимодействия с MQTT-брокером в разрабатываемом приложении.

Интерфейс MQTTMessaging определяет методы subscribe и publish, которые используется клиентом для взаимодействия с брокером. Метод подписки subscribe принимает в качестве параметра MQTT-топик, на который необходимо подписаться.

Метод для публикации сообщений, в свою очередь, отправляет заданное сообщение payload в указанный MQTT-топик с возможностью выбора уровня качества обслуживания qos и флага сохраненного сообщения retained.

Реализуя этот интерфейс, класс описывает логику подписки на MQTT-топики и отправки сообщений, обеспечивая взаимодействие с брокером.

Интерфейс MQTTConnection определяет методы connect и disconnect, используемые для управления подключением к MQTT-брокеру.

Метод connect выполняет установку соединения с брокером и возвращает Boolean, указывающий на успешность подключения.

Метод disconnect, в свою очередь, разрывает текущее соединение с брокером.

Реализуя этот интерфейс, класс описывает логику установки и завершения соединения, обеспечивая стабильное взаимодействие с брокером.

Класс MQTTClient представляет собой реализацию MQTT-клиента для взаимодействия с брокером. Данный класс реализует интерфейсы MQTTMessaging и MQTTConnection. Он позволяет настроить соединение с учетом параметров аутентификации, обрабатывать входящие сообщения через callback-функции и управлять подписками.

Атрибуты класса:

* mqttClient: MqttClient? – объект клиента MQTT;
* broker: Broker? – содержит информацию о брокере;
* messageHandler: MQTTMessageHandler? – обработчик входящих сообщений.

Методы, реализованные в классе:

* initialize(Broker, MQTTMessageHandler) – выполняет инициализацию клиента, устанавливая параметры подключения к брокеру и назначая обработчик сообщений;
* reinitialize(Broker, MQTTMessageHandler) – завершает текущее соединение, перенастраивает параметры подключения и повторно инициализирует клиента с новым брокером;
* connect(): Boolean – выполняет подключение к брокеру с учетом учетных данных. Возвращает true в случае успешного соединения и false при возникновении ошибки;
* subscribe(topic: String) – подписывает клиента на указанный MQTT-топик и передает полученные сообщения в messageHandler;
* publish(topic: String, payload: String, qos: Int, retained: Boolean) – отправляет сообщение в указанный топик с заданным уровнем качества обслуживания (QoS) и возможностью хранения последнего значения (retained);
* disconnect() – разрывает соединение с MQTT-брокером и освобождает ресурсы.

Класс MQTTMessageHandler отвечает за обработку входящих MQTT-сообщений и взаимодействие с моделями данных приложения.

Метод handleMessage(topic: String, payload: String) является основной функцией для обработки входящих MQTT-сообщений. Анализирует topic сообщения и вызывает соответствующий обработчик в зависимости от его типа. Это позволяет корректно маршрутизировать данные в системе.

Метод handleDeviceStateMessage(topic: String, payload: String) отвечает за обновление состояния устройств. Разбирает данные, полученные в сообщении, и обновляет соответствующую запись в модели состояния устройств. Может использоваться для синхронизации данных между физическими устройствами и интерфейсом пользователя.

Метод handleDeviceCommandMessage(topic: String, payload: String) предназначен для обработки команд, отправленных устройствам. Проверяет, существует ли уже такая команда в базе данных, и, если нет, сохраняет ее. Это позволяет избежать дублирования команд и обеспечивает корректное управление устройствами.

Метод handleDeviceListMessage(payload: String) выполняет обработку JSON-данных, содержащих список доступных устройств. Если какое-либо устройство еще не добавлено в систему, оно записывается в базу данных. Этот механизм позволяет автоматически обновлять список доступных устройств в приложении.

Данный класс обеспечивает корректную маршрутизацию MQTT сообщений и взаимодействие с внутренними структурами данных приложения.

Класс Topics содержит предопределенные константы, представляющие MQTT-топики, с которыми взаимодействует приложение. Эти топики используются для обмена данными с MQTT-брокером и управления устройствами умного дома.

Атрибуты:

* DISCOVERY\_TOPIC – топик для включения режима сопряжения новых Zigbee-устройств;
* DEVICE\_STATE\_TOPIC – шаблон топиков, содержащих информацию о состоянии устройств;
* DEVICE\_COMMANDS\_TOPIC – базовый топик для получения информации о командах для управления устройствами;
* DEVICE\_LIST\_TOPIC – топик для получения списка доступных устройств.
  1. Вспомогательные компоненты системы

В файле-разметке AndroidManifest.xml заданы необходимые разрешения, которые приложение запрашивает перед использованием определенных функций. Ниже приведен список указанных разрешений и их назначение:

1. INTERNET – разрешает приложению доступ в интернет. Требуется для работы с сетевыми запросами по протоколу MQTT.
2. ACCESS\_NETWORK\_STATE – позволяет приложению запрашивать текущее состояние о подключении к сети.
3. POST\_NOTIFICATIONS – предоставляет приложению доступ для отправки уведомлений.

Так же в данном файле задаются ключевые параметры приложения. Устанавливаются иконки, название и тема, что определяет внешний вид и стиль интерфейса. Указываются настройки резервного копирования, позволяющие сохранять данные при переустановке или переносе на другое устройство.

Класс Logger предоставляет удобный способ ведения логов в приложении. Реализован с использованием порождающего паттерна Singleton и использует Log.i() для вывода информационных сообщений.

Функции для логирования:

* log(className: KClass<\*>, message: String) – логирует сообщение, используя имя класса в качестве тега;
* log(tag: String, message: String) – логирует сообщение с указанным тегом.

Этот класс упрощает отладку, позволяя быстро идентифицировать источник сообщений в логах.

Класс Constants представляет собой объект, содержащий константы, используемые в приложении.

Основные значения, представленные в виде констант:

* типы устройств: SWITCH\_TYPE, SELECT\_TYPE, DIMMER\_TYPE;
* список вкладок интерфейса: TABS\_LIST включает «Devices», «Rooms», «Settings»;
* настройки для системы обнаружения устройств: DISCOVERY\_ENABLE, DISCOVERY\_DISABLE, DISCOVERY\_TIME.

Этот класс обеспечивает централизованное хранение значений, упрощая доступ и поддержку кода.

1. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Разработка данного приложения производилась в интегрированной среде разработки Android Studio с использованием языка программирования Kotlin и фреймворка для разработки графического интерфейса Jetpack Compose.

В приложении реализуется следующий функционал:

* управление соединением с MQTT-брокером;
* обработка данных, полученных от MQTT-брокера;
* отправка пользовательских данных в MQTT-брокер;
* управление модулем светодиодной подсветки;
* отображение общего списка устройств системы;
* отображение списка комнат системы;
* добавление новых комнат в систему;
* управление режимом обнаружения и подключения новых устройств;
* отображение полной информации об устройстве;
* привязка устройства к определенной комнате.
  1. Управление соединением с MQTT-брокером

За функционал управления соединением отвечает пользовательский интерфейс AuthorizationScreen и класс клиента MQTT-брокера MQTTClient.

Установка нового соединение состоит из двух основных этапов:

* получение данных об используемом брокере;
* создание экземпляра клиента и подключение к брокеру.

Данные об используемом брокере могут быть получены несколькими способами:

* посредством пользовательского ввода;
* из сохраненных данных в базе данных.
  + 1. В первом случае это может быть ввод данных об подключении через форму BrokerInputForm на экране AuthorizationScreen, где пользователь указывает адрес сервера, порт брокера, имя пользователя и пароль (если на сервере настроена аутентификация пользователей). Введенные данные должны пройти проверку на валидность, чтобы убедиться, что все параметры корректны (например, правильный формат адреса и порта). После успешной валидации и подключения к брокеру, данные сохраняются в базу данных, чтобы при следующих запусках приложения пользователь мог автоматически подключиться, без необходимости повторного ввода данных. Валидация и добавление информации в базу данных представлена в функции, описанной ниже:

fun addBroker(serverUri: String, serverPort: Int, user: String?, password: String?) {

if (serverUri.isBlank()) return

val uriPattern = "^(http|https|mqtt|mqtts)://[a-zA-Z0- 9.-]+(:[0-9]+)?".toRegex()

if (!uriPattern.matches(serverUri)) return

if (serverPort !in 1..65535) return

viewModelScope.launch {

val broker = Broker(

serverUri = serverUri,

serverPort = serverPort,

user = user,

password = password

)

brokerDao.insert(broker)

loadBrokers()

}

}

Метод brokerDao.insert(broker) реализован следующим образом:

@Insert(onConflict = OnConflictStrategy.REPLACE)

suspend fun insert(broker: Broker)

В данном методе происходит вставка нового объекта Broker в базу данных. @Insert(onConflict = OnConflictStrategy.REPLACE) – данная аннотация означает, что если в таблице уже существует запись с таким же первичным ключом, она будет заменена новой. Это поведение удобно для обновления настроек брокера без необходимости выполнения дополнительной операции удаления. Так же метод помечен ключевым словом suspend, что означает его асинхронное выполнение. Для вызова этого метода требуется использовать корутину.

* + 1. Если пользователь ранее уже вводил корректные данные для подключения, приложение может автоматически извлечь эти данные из базы данных и отобразить на экране авторизации в виде карточки с информацией о подключении.

@Composable

fun BrokerList(

brokers: List<Broker>,

onDelete: (Broker) -> Unit,

onLogin: (Broker) -> Unit

) {

brokers.lastOrNull()?.let { broker ->

Text("Recently used broker:", style = MaterialTheme.typography.labelMedium)

Spacer(modifier = Modifier.height(1.dp))

BrokerItem(

broker = broker,

onDelete = { onDelete(broker) },

onLogin = { onLogin(broker) }

)

}

}

В этом случае, пользователь не будет вынужден снова вводить параметры вручную, что значительно ускоряет процесс подключения. Приложение предоставит пользователю информацию о последнем использованном брокере, и при наличии интернет-соединения по нажатию кнопки входа в систему произойдет подключении к брокеру.

При нажатии кнопки входа вызывается следующая функция:

fun handleLogin(broker: Broker, onSuccess: () -> Unit) {

BrokerState.setBrokerId(broker.id)

val mqttClient = MQTTClient.reinitialize(broker, messageHandler)

val isSuccess = mqttClient.connect()

if (isSuccess) {

onSuccess()

viewModelScope.launch(Dispatchers.IO) {

mqttClient.subscribe(Topics.SUBSCRIBE\_DEVICE\_LIST\_TOPIC) mqttClient.subscribe(Topics.SUBSCRIBE\_COMMANDS\_TOPIC) mqttClient.subscribe(Topics.SUBSCRIBE\_DEVICE\_STATE\_TOPIC) mqttClient.subscribe(Topics.SUBSCRIBE\_LED\_STATE\_TOPIC)

}

}

}

В этом методе идет проверка подключения к брокеру, и в случае успешного подключения производится асинхронная подписка на ключевые темы для корректного управления и мониторинга системой «умный дом» посредством использования мобильного приложения. Подписка подразумевает собой установку соответствующего callback метода, который автоматически вызывается при получении сообщений по указанным темам. Этот метод обрабатывает входящие данные и передает их в систему для дальнейшей обработки или отображения в интерфейсе мобильного приложения.

* 1. Обработка данных полученных от MQTT-брокера

После принятия сообщения данные следует обработать корректным образом, так как они приходят в «сыром виде». При инициализации клиента MQTT-брокера определяется класс MQTTMessageHandler, который будет обрабатывать приходящие в темы сообщения.

fun handleMessage(topic: String, payload: String) {

Log.i("MQTTHandler", "Обрабатываем сообщение: $payload с топика: $topic")

when {

topic.startsWith(Topics.DEVICE\_STATE\_TOPIC) -> handleDeviceStateMessage(topic, payload)

topic.startsWith(Topics.DEVICE\_COMMANDS\_TOPIC) -> handleDeviceCommandMessage(topic, payload)

topic.startsWith(Topics.DEVICE\_LIST\_TOPIC) -> handleDeviceListMessage(payload)

topic.startsWith(Topics.LED\_STATE\_TOPIC) -> handleLEDState(payload)

else -> Log.i("MQTTHandler", "Необрабатываемый топик: $topic")

}

}

Данный метод выполняет роль оркестратора, принимая входящие сообщения и, в зависимости от темы, делегирует их обработку соответствующим методам. Этот подход предоставляет возможность для вариативной обработки событий в системе «умного дома».

* + 1. Обработка состояний устройств происходит в методе, представленном ниже:

private fun handleDeviceStateMessage(topic: String, payload: String) {

val ieeeAddr = extractIeeeAddrFromTopic(topic)

val deviceId = devicesViewModel.devices.value.find { it.ieeeAddr == ieeeAddr }?.id

if (deviceId != null) {

DeviceState.updateDeviceData(deviceId, payload)

} else {

println("Устройство с IEEE Addr $ieeeAddr не найдено")

}

}

При получении сообщения из информационной темы устройства осуществляется извлечение его физического адреса из имени темы. На основе этого адреса определяется внутренний идентификатор устройства в приложении, после чего производится обновление текущего состояния устройства во временном хранилище данных приложения.

* + 1. Для получения актуализированной информации о подключенных устройствах, в случае их сопряжения с использованием графического интерфейса координатора используется следующий метод:

private fun handleDeviceListMessage(payload: String) {

try {

val jsonObject = JSONObject(payload)

for (key in jsonObject.keys()) {

val deviceJson = jsonObject

.optJSONObject(key) ?: continue

val ieeeAddr = deviceJson.optString("ieeeAddr")

val friendlyName = deviceJson

.optString("friendly\_name")

val modelId = deviceJson

.optString("ModelId")

val device = Device.create(

ieeeAddr = ieeeAddr,

friendlyName = friendlyName,

modelId = modelId,

roomId = null,

brokerId = BrokerState.brokerId.value ?: -1

)

devicesViewModel.addDeviceIfNotExists(device)

}

} catch (e: Exception) {

Log.e("DEVICE", "Ошибка обработки списка устройств: ${e.message}")

}

}

Здесь происходит первоначальная обработка информации, получаемой от брокера, с преобразованием в формат JSON. Далее выполняется обработка каждого ключа и извлекаются данные об устройстве. Если устройство еще не добавлено в систему, оно создается и добавляется в базу данных с использованием метода addDeviceIfNotExists.

* + 1. Метод handleDeviceCommandMessage отвечает за обработку сообщений, содержащих команды для управления устройствами системы «умный дом». В этом методе извлекается информация о команде управления устройством, которая используется для формирования объекта, необходимого для создания записи в базе данных.

val jsonObject = JSONObject(payload)

val commandTopic = jsonObject

.optString("command\_topic")

.takeIf { it.isNotBlank() } ?: return

val payloadOn = jsonObject.optString("payload\_on", null)

val payloadOff = jsonObject.optString("payload\_off", null)

val commandTemplate = jsonObject

.optString("command\_template", null)

val options = jsonObject.optJSONArray("options")

?.let { array -> (

0 until array.length()).associate { index ->

array.getString(index) to

array.getString(index)

}

} ?: emptyMap()

val commandType = extractCommandTypeFromTopic(topic)

val deviceIeeeAddr = extractIeeeAddrFromTopic(topic)

Сначала происходит извлечение данных из входного сообщения в формате JSON. Затем определяется тип команды с помощью функции extractCommandTypeFromTopic и извлекается уникальный адрес устройства с помощью метода extractIeeeAddrFromTopic, что позволяет точно идентифицировать, какое устройство должно быть управляемым. После формирования объекта данные сохраняются в базу данных и привязываются к устройству соответствующему данному физическому адресу, что позволяет точно идентифицировать, какое устройство в приложении должно отображаться как управляемое.

* 1. Отправка пользовательских данных в MQTT-брокер

Для управления системой «умный дом» в приложении реализован механизм отправки пользовательских команд через MQTT-брокер. Когда пользователь взаимодействует с интерактивными элементами графического интерфейса – например, включает свет – приложение интерпретирует это как намерение изменить состояние соответствующего устройства. Для реализации такого поведения формируется и отправляется MQTT-сообщение с помощью функции, предназначенной для обработки команд управления переключаемыми устройствами:

fun onToggle(deviceId: Int, state: Boolean) {

viewModelScope.launch {

val cmd = db.commandDAO()

.getSwitchCommandByDeviceId(

deviceId)

val newState = if (!state) cmd?.payloadOff else cmd?.payloadOn

cmd?.let {

if (newState != null) {

sendCommandToMqtt(it.commandTopic, newState)

}

}

}

}

В функции запускается корутина, внутри которой из локальной базы данных извлекается объект команды, связанный с конкретным устройством. На основе переданного состояния определяется, какое значение должно быть отправлено в брокер. Таким образом, приложение формирует MQTT-сообщение, соответствующее синтаксису Home Assistant, и отправляет его по указанному в объекте команды топику.

Для непосредственной отправки сообщения используется метод publish, определенный в интерфейсе MQTTMessaging и реализованный в классе MQTTClient.

override fun publish(topic: String, payload: String, qos: Int, retained: Boolean) {

    try {

        val message =

            MqttMessage(payload.toByteArray()).apply {

                this.qos = qos

                this.isRetained = retained

            }

        mqttClient?.publish(topic, message)

        Log.i("MQTT", "Отправлено сообщение: $payload в топи к: $topic")

     } catch (e: MqttException) {

        Log.e("MQTT", "Ошибка отправки сообщения: ${e.reason Code} - ${e.message}")

}

}

Внутри метода создается объект MqttMessage, которому задаются параметры качества обслуживания и флаг сохранения сообщения. После этого сформированное сообщение публикуется в указанный топик с помощью сущности mqttClient.

* 1. Управление модулем светодиодной подсветки

Для управления светодиодной подсветкой в приложении используется два метода, в зависимости от выбранного пользователем:

* автоматический;
* ручная настройка.
  + 1. Для автоматического режима подсветки используется следующий метод:

fun setAUTOMode() {

viewModelScope.launch {

val ledAutoStatus = """

{

"state": "OFF",

"brightness": 255,

"color": {

"r": 255,

"g": 255,

"b": 255

},

"color\_mode": "rgb",

"mode": "auto"

}

""".trimIndent()

MQTTClient.publish(Topics.LED\_SET\_STATE\_TOPIC, ledAutoStatus)

}

}

В корутине создается JSON-объект, описывающий параметры подсветки в автоматическом режиме. Ключевое значение mode устанавливается в auto, что указывает устройству перейти в автоматический режим управления подсветкой. В этом режиме поведение светодиодов зависит от текущего состояния устройства: например, при активации режима сопряжения подсветка начинает мигать зеленым, сигнализируя о готовности к подключению.

* + 1. Для настройки подсветки по пользовательским параметрам пользователю предоставляется возможность вручную отрегулировать яркость и интенсивность каждого цвета подсветки с помощью отдельных карточек-интерфейсов:

BrigtnessCard(

title = "Brightness",

color = Color(brightness / 255f, brightness / 255f, brightness / 255f),

value = brightness,

onValueChange = { LEDState.setBrightness(it) },

onValueChangeFinished = { ledScreenViewModel

.sendLEDStatus()})

Данная карточка отвечает за регулировку яркости светодиодной подсветки. При изменении значения на ползунке в режиме ручной настройки изменяется цвет индикатора яркости – небольшого визуального элемента внутри карточки. Этот элемент служит для наглядного отображения текущего уровня яркости: от черного цвета, символизирующего минимально допустимый уровень подсветки, до белого – соответствующего максимальной яркости. Обновление значения яркости происходит в функции, передаваемой в параметре onValueChange, а отправку команды координатору – только после завершения действия пользователя с использованием функции onValueChangeFinished.

Для настройки интенсивности каждого цветового канала (красного, зеленого и синего) используется отдельная карточка:

ColorsCard(

red = red,

green = green,

blue = blue,

onRedChange = { LEDState.setRed(it) },

onGreenChange = { LEDState.setGreen(it) },

onBlueChange = { LEDState.setBlue(it) },

onValueChangeFinished = {

ledScreenViewModel.sendLEDStatus()

}

)

Значения цветовых компонентов изменяются в режиме реального времени, а итоговое состояние модуля подсветки координатора передается устройству также только по завершении регулировки.

* 1. Отображение общего списка устройств системы

Полный список устройств системы отображается в соответствующем разделе интерфейса с использованием DevicesScreen. В данном компоненте происходит чтение потока, содержащего информацию о всех активных устройствах системы:

val switchDevices by devicesViewModel

.getDevicesByTypeFlow("switch").collectAsState()

val selectDevices by devicesViewModel

.getDevicesByTypeFlow("select").collectAsState()

val sliderDevices by devicesViewModel

.getDevicesByTypeFlow("slider").collectAsState()

val devicesWithoutCommands by devicesViewModel

.getDevicesWithoutCommandsFlow().collectAsState()

Для отображения устройств используется DeviceCard. Данный компонент является универсальным и позволяет динамически отображать различные элементы управления, такие как переключатели, ползунки и выпадающие списки, в зависимости от типа устройства и его состояния. Каждое устройство может быть связано с различными параметрами, такими как яркость, состояние включения/выключения или выбор параметра из списка.

* + 1. В случае отображения устройства с возможностью переключения состояния, например, включения и выключения:

"switch" -> {

val switchColors = SwitchDefaults.colors(

checkedThumbColor = Color.White,

checkedTrackColor = Color(0xFF4CAF50),

uncheckedThumbColor = Color.White,

uncheckedTrackColor = Color(0xFFF44336))

Switch(

checked = checked,

onCheckedChange = {

checked = it

onToggle?.invoke(it)

},

colors = switchColors

)

* + 1. Для устройств с возможностью регулирования значения, например, яркости освещения, значение может быть представлено в диапазоне от 0 до 100. В случае, если параметр устройства не находится в пределах этого диапазона, его значение делится на 100 равных промежутков. Каждый из этих промежутков затем используется для вычисления корректного значения в пределах от 0 до 100, что позволяет точно управлять параметром, даже если он изначально имеет другой диапазон. Эта логика представлена в коде ниже.

"slider" -> {

val sliderColors = SliderDefaults.colors(

thumbColor = Color(0xFF8A9F9B),

activeTrackColor = Color(0xFFA6B6A9))

Slider(

value = sliderValue,

onValueChange = {

sliderValue = it

onSliderChange?.invoke(it)

},

valueRange = 0f..100f,

colors = sliderColors)

}

* + 1. Для устройств с возможностью выбора параметра из списка, например, режим работы устройства, где значение может быть представлено в виде заранее определенного набора опций, используется следующий элемент управления:

"select" -> ExposedDropdownMenuBox(

    expanded = expanded,

    onExpandedChange = {

expanded = it }) {

    TextField(

        value = selectedOption,

        onValueChange = {},

        readOnly = true,

        modifier = Modifier.menuAnchor(),

        label = { Text("Выберите") },

        trailingIcon = {

ExposedDropdownMenuDefaults

.TrailingIcon(expanded = expanded) })

    ExposedDropdownMenu(

        expanded = expanded,

        onDismissRequest = { expanded = false }) {

        options.forEach {

option ->

            DropdownMenuItem(

                text = { Text(option) },

                onClick = {

                    selectedOption = option

                    expanded = false

                    onSelectChange?.invoke(option)})}}}

* 1. Отображение списка комнат системы

Для получения списка комнат системы необходимо перейти на вторую вкладку главного экрана приложения. После нажатия на кнопку «Rooms» откроется интерфейс, который отобразит карточки комнат, каждая из которых будет содержать название и количество устройств, связанных с данной комнатой. За данный функционал отвечает следующий код:

items(rooms) { room ->

var deviceCount by remember {

mutableIntStateOf(0)}

LaunchedEffect(room.id) {

roomsViewModel.getDeviceCount(room.id) {

count -> deviceCount = count

}

}

RoomCard(

roomId = room.id.toInt(),

roomName = room.name,

deviceCount = deviceCount,

navHostController = navHostController,

onDelete = { roomId ->

roomsViewModel.deleteRoom(roomId)

}

)

}

Для каждой комнаты мы создаем карточку, используя Composable-компонент RoomCard. В этот компонент передается информация об комнате, а также функции для навигации, которые позволяют перейти на новый экран с информацией об устройствах, относящихся к этой комнате. Также реализована функция для удаления комнаты, которая вызывается при нажатии на соответствующую кнопку внутри карточки.

Внутри компонента формируется карточка при нажатии на которую пользователя перенаправляет на экран информации об устройствах, относящихся к выбранной комнате. Для этого используется следующий код:

Card(

shape = RoundedCornerShape(16.dp),

modifier = Modifier

.fillMaxWidth()

.padding(8.dp)

.clickable { navHostController

.navigate("room\_details/$roomId") },

elevation = CardDefaults

.cardElevation(6.dp),

colors = CardDefaults.cardColors(

containerColor = Color(0xFFE3F2FD)

)

)

Отображение устройств, привязанных к выбранной комнате, реализовано аналогично отображению всех устройств, однако при этом накладывается дополнительный фильтр, учитывающий привязку устройства к комнате. Пример фильтрации для устройств с режимом «переключения» состояния:

val switchDevices by devicesViewModel

.getDevicesByTypeFlow("switch")

.collectAsState()

val roomSwitchDevices = switchDevices

.filter { it -> it.id in roomDevices.map {

it.id }}

После фильтрации на экране RoomDetailsScreen отобразятся все устройства, связанные с этой комнатой.

* 1. Добавление новых комнат в систему

На экране RoomsScreen реализован функционал добавления новой комнаты с помощью плавающей кнопки (FloatingActionButton), расположенной в правом нижнем углу интерфейса. Кнопка оформлена в соответствии с Material Design 3 и содержит иконку «плюс», обозначающую действие добавления.

FloatingActionButton(

onClick = { showDialog = true },

containerColor = MaterialTheme.colorScheme.primary,

modifier = Modifier

.align(Alignment.BottomEnd)

.padding(16.dp)

) {

Icon(imageVector = Icons.Default.Add, contentDescription = "Добавить комнату")

}

При нажатии на кнопку состояние отображения диалогового окна showDialog становится true, и на экране отображается диалоговое окно AddRoomDialog. Это диалоговое окно позволяет пользователю ввести название новой комнаты.

if (showDialog) {

AddRoomDialog(

onDismiss = { showDialog = false },

onConfirm = { roomName ->

roomsViewModel.addRoom(roomName)

showDialog = false

Log.i("Rooms", "Комната добавлена: $roomName")}

)

}

При заполнении информации в диалоговом окне предусмотрены два действия:

* отмена – закрытие диалогового окна без сохранения данных;
* подтвердить – введенное имя комнаты передается в roomsViewModel.addRoom, после чего диалог закрывается и начинается процесс сохранения комнаты в базе данных.

Функция addRoom во ViewModel отвечает за асинхронное создание и сохранение новой комнаты в базе данных:

fun addRoom(name: String) {

viewModelScope.launch {

db.roomEntityDAO().insertRoom(

RoomEntity(name = name))}}

При вызове этого метода:

* создается объект RoomEntity с указанным именем;
* выполняется вставка в базу данных с помощью DAO (insertRoom);
* операция запускается внутри viewModelScope.launch, что обеспечивает асинхронное выполнение без блокировки основного потока интерфейса.
  1. Управление режимом обнаружения и подключения новых устройств

На экране настроек приложения реализован элемент управления режимом обнаружения новых устройств – переключатель Switch, размещенный внутри компонента Card. Данный переключатель предоставляет пользователю возможность вручную активировать или деактивировать режим «обнаружения устройств» в системе умного дома.

Switch(

checked = discoveryState,

onCheckedChange = { onSwitchChanged(it) },

colors = SwitchDefaults.colors(

checkedThumbColor = Color.White,

checkedTrackColor = Color(0xFF4CAF50),

uncheckedThumbColor = Color.White,

uncheckedTrackColor = Color(0xFFF44336)

)

)

Изменение состояния переключателя вызывает функцию onSwitchChanged, которая в зависимости от значения параметра enabled инициирует соответствующее действие:

fun onSwitchChanged(enabled: Boolean) {

if (enabled) {

DiscoveryState.startDiscovery()

} else {

DiscoveryState.stopDiscovery()

}

}

Функциональность включения и выключения режима обнаружения реализована в объекте DiscoveryState.

fun startDiscovery() {

\_isDiscoveryActive.value = true

    MQTTClient.publish(

Topics.DISCOVERY\_TOPIC, Constants.DISCOVERY\_ENABLE

)

    resetJob?.cancel()

    resetJob = CoroutineScope(Dispatchers.Default)

.launch {

        delay(Constants.DISCOVERY\_TIME)

        \_isDiscoveryActive.value = false}}

Данная функция отвечает за активацию режима обнаружения устройств. Она устанавливает значение флага \_isDiscoveryActive в true, публикует соответствующее сообщение в тему DISCOVERY\_TOPIC, а также запускает корутину, которая запускает таймер на 255 секунд. При этом, если ранее уже был запущен таймер, он отменяется, чтобы избежать конфликтов и повторных срабатываний.

* 1. Отображение детальной информации об устройстве

Для просмотра подробной информации о состоянии устройства и показаниях датчиков необходимо нажать на карточку соответствующего устройства. В компонент DeviceCard передается NavHostController.

DeviceCard(

deviceId = device.id,

imageRes = R.drawable.mqtt\_logo,

name = device.friendlyName,

type = Constants.SWITCH\_TYPE,

value = (deviceData?.get("state") as? String)?

.toBooleanState() ?: false,

navController = navHostController,

onToggle = { state ->

devicesViewModel.onToggle(device.id, state)

}

)

Обработка навигации осуществляется внутри компонента DeviceCard с использованием модификатора clickable, с помощью которого при нажатии осуществляется переход на экран с деталями: вызывается навигация на маршрут device\_details/$deviceId, где deviceId – идентификатор выбранного устройства.

Card(

shape = RoundedCornerShape(16.dp),

modifier = Modifier

.fillMaxWidth()

.padding(8.dp)

.clickable { navController.navigate("device\_details/$deviceId") },

elevation = CardDefaults.cardElevation(6.dp)

)

После перехода на соответствующий маршрут отображается экран с подробной информацией об устройстве. На экране доступно изменение дружественного имени устройства и выпадающее меню для привязки к комнате, а также отображается текущая привязанная комната. Ниже выводится информация о всех доступных значениях датчиков, а также служебные данные устройства, такие как last\_seen. Все значения получаются из временного хранилища DeviceState.

* 1. Привязка устройства к определенной комнате

Привязка устройства к комнате подразумевает наличие заранее созданной комнаты в системе. Пользователь может выбрать одну из доступных комнат из выпадающего списка на экране подробной информации об устройстве.

ExposedDropdownMenu(

    expanded = expanded,

    onDismissRequest = { expanded = false }

) {

    DropdownMenuItem(

        text = { Text("Select room") },

        onClick = {

            selectedRoom = "Select room"

            expanded = false

            devicesViewModel

.assignRoomToDevice(deviceId, null)

        }

    )

    rooms.forEach { room ->

        DropdownMenuItem(

            text = { Text(room.name) },

            onClick = {

                selectedRoom = room.name

                expanded = false

                devicesViewModel

.assignRoomToDevice(

deviceId, room.id)})}}

Данный компонент отвечает за отображение выпадающего меню с доступными комнатами. Он реализован с использованием ExposedDropdownMenu, внутри которого каждый элемент представляет одну из заранее созданных комнат. При выборе комнаты обновляется локальное состояние selectedRoom, закрывается меню и вызывается метод assignRoomToDevice у devicesViewModel, передающий идентификатор выбранной комнаты для привязки устройства.

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В данном разделе описано руководство по установке и взаимодействию с системой «умный дом». Это руководство разделено на три части по настройке, поскольку в рамках дипломного проекта используется как ранее реализованная аппаратная часть с прошивкой, так и разрабатываемое программная часть:

1. Руководство пользователя координатора. Этот раздел описывает процесс подготовки устройства координатора к работе в системе «умный дом». Здесь представлены инструкции по прошивке координатора, подключению к источнику питания и сети, а также первоначальной настройке для корректной работы с приложением.
2. Руководство пользователя MQTT-брокера. В этом разделе описывается настройка серверной части системы, выполняющей функцию брокера сообщений. Приведены инструкции по установке и конфигурации брокера, настройке параметров подключения и безопасности.
3. Руководство пользователя мобильного приложения. Этот раздел посвящен использованию мобильного приложения для управления системой. Здесь будут представлены инструкции по загрузке, установке и запуску приложения. Так же будут рассмотрены основные функции приложения, включая добавление системы в приложение, просмотр информации об устройствах, а также управление системными модулями.
   1. Руководство пользователя координатора

После первого включения координатора, при условии успешной прошивки микроконтроллера ESP32-WROVER-IE и модуля RF-STAR RF-BM-2652P2, можно приступить к его первоначальной настройке.

Для этого необходимо подключить питание через разъем microUSB. После подачи питания координатор автоматически перейдет в режим точки доступа, что позволяет выполнить начальную настройку доступа в интернет без необходимости перепрошивки устройства. В доступных сетях Wi-Fi появится точка доступа с названием формата «SLS XXXX» (см рисунок 5.1).

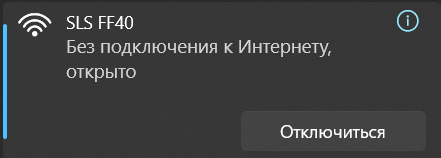


Рисунок 5.1 – Wi-Fi для подключения к координатору

Для подключения к координатору подойдет любое устройство, поддерживающее Wi-Fi не ниже версии 4. После подключения к сети необходимо открыть веб-браузер и перейти на локальный сервер шлюза по адресу «http://192.168.100.1», где и осуществляется дальнейшая настройка подключения координатора к интернету. Интерфейс подключения к сети изображен на рисунке 5.2.

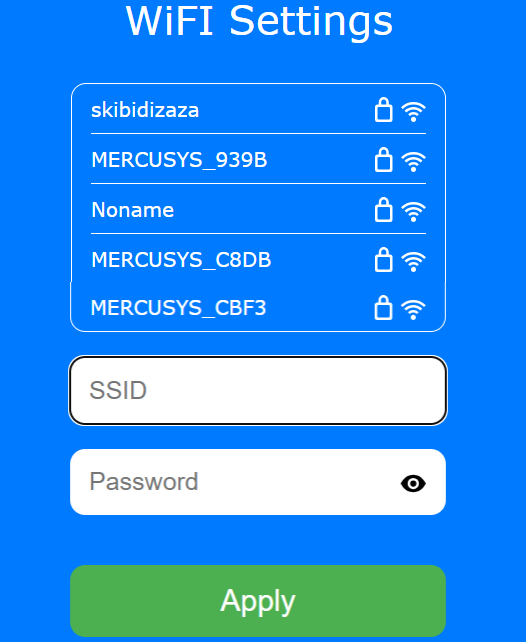


Рисунок 5.2 – Интерфейс подключения к Wi-Fi

В открывшейся веб-странице необходимо выбрать сеть Wi-Fi из списка, отображающего доступные в данный момент сети, которые видит координатор. Также предусмотрена возможность ручного ввода названия сети (SSID) и пароля, если нужная сеть не отображается в списке.

После подключения к сети Wi-Fi необходимо произвести полную перезагрузку устройства, отключив и повторно подключив его через разъем microUSB. После перезагрузки устройство автоматически перейдет в режим координатора с собственным веб-сервером, доступным по IP-адресу, который будет выдан маршрутизатором. Узнать IP-адрес координатора можно через веб-интерфейс маршрутизатора в списке подключенных устройств или с помощью сетевых сканеров, таких как nmap с использованием команды nmap -sn 192.168.0.1/24, где необходимо указать адрес локальной подсети.

Для корректной работы координатора необходимо произвести настройку номеров контактов подключения модуля радиосвязи к модулю микроконтроллера. Для этого необходимо перейти во вкладку Settings → Hardware. Откроется интерфейс настройки периферии, который необходимо заполнить следующим образом, как показано на рисунке 5.3.

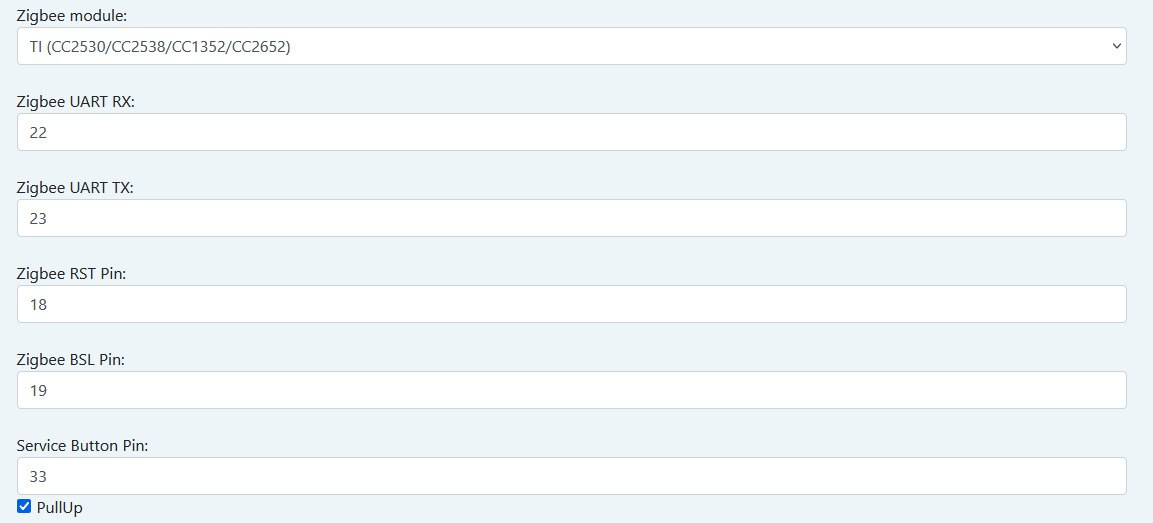


Рисунок 5.3 – Интерфейс аппаратной конфигурации модуля радиосвязи

Для настройки модуля светодиодной подсветки необходимо заполнить информацию в форме ниже. Данные для настройки представлены на рисунке 5.4. После ввода данных необходимо нажать кнопку «Save» для сохранения параметров.

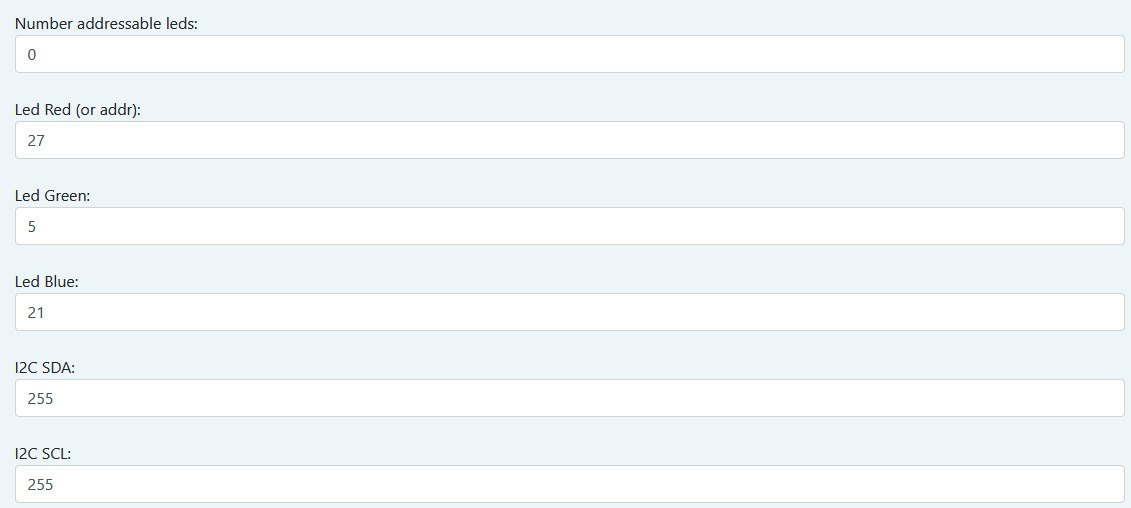


Рисунок 5.4 – Интерфейс аппаратной конфигурации модуля светодиодной подсветки

После настройки конфигурации необходимо выполнить перезагрузку устройства с помощью соответствующей кнопки в графическом интерфейсе. При успешной настройке и корректном подключении модуль радиосвязи будет обнаружен, и его статус отобразится как **Status: 9 OK**, как показано на рисунке 5.5. В случае ошибки или некорректной конфигурации в этом же поле будет выведено сообщение с указанием типа ошибки.

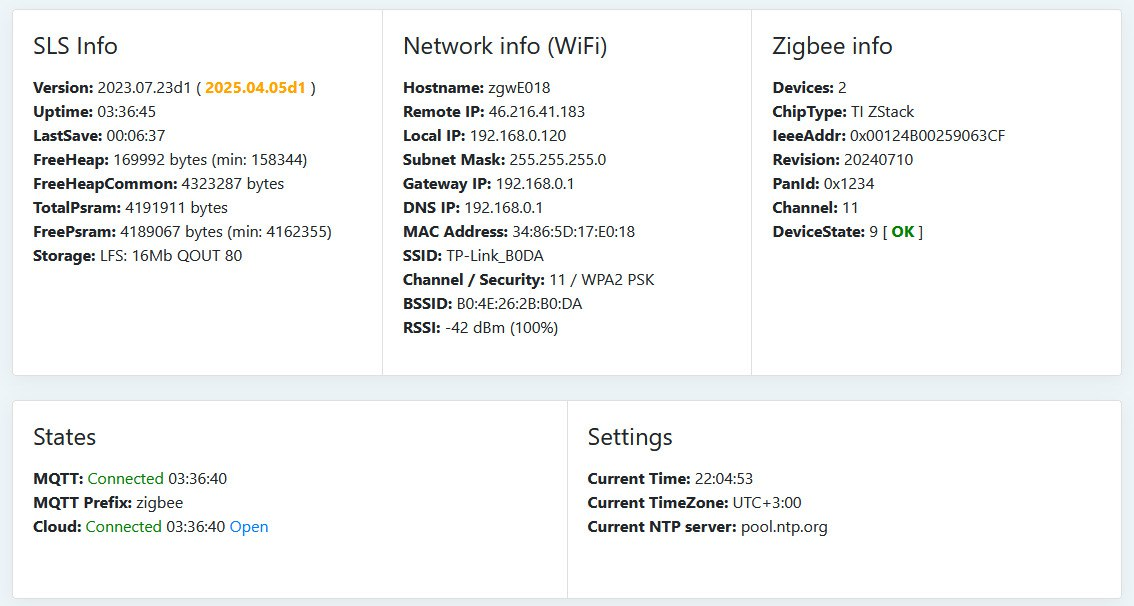


Рисунок 5.5 – Успешное подключение модуля радиосвязи

После успешной настройки радиомодуля необходимо подключить координатор к брокеру MQTT. Для этого необходимо перейти во вкладку Settings → Link → MQTT Setup. В открывшемся окне необходимо выставить чек-бокс Enable в активное состояние. Ниже необходимо заполнить поля, содержащие информацию о брокере. Поля «User» и «Password» необходимо заполнить в случае настроенной авторизации. В поле «System prefix topic» необходимо ввести значение «zigbee». Поля состояний «Retain States» и «Home Assistant MQTT Discovery» перевести в активное состояние. Интерфейс с введенными данными должен выглядеть так, как показано на рисунке 5.6.

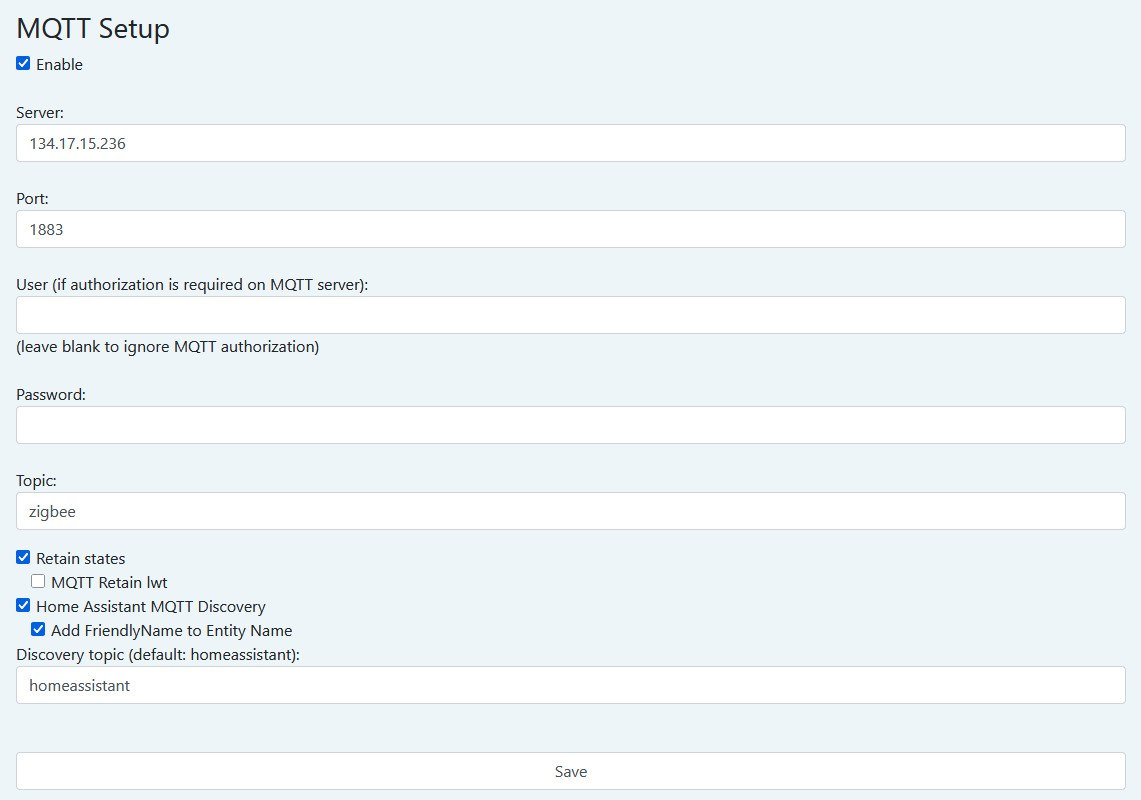


Рисунок 5.6 – Интерфейс настройки подключения к MQTT брокеру

Если данные для подключения указаны верно, то на главном экране веб-интерфейса координатора появится окно «**States»**, в котором будет отображен статус подключения.

* 1. Руководство пользователя MQTT-брокера

Для корректной работы брокера MQTT сообщений необходима операционная система на базе Linux удовлетворяющая следующим условиям:

* операционная система: дистрибутив на базе ядра Linux версии 3.10 и выше;
* процессор: 1-ядерный, с тактовой частотой не менее 2,6 ГГц;
* оперативная память: не менее 1 ГБ;
* свободное место на диске: не менее 12 ГБ;
* наличие стабильного подключения к сети Интернет со скоростью не ниже 1 Мбит/с;
* поддержка виртуализации.

Для непрерывной и стабильной работы брокера сообщений используется инструмент контейнеризации Docker. Для установки необходимого программного пакета следует воспользоваться официальной инструкцией [6], соответствующей используемой операционной системе. Выбор инструкции в соответствии с платформой представлен на рисунке 5.7

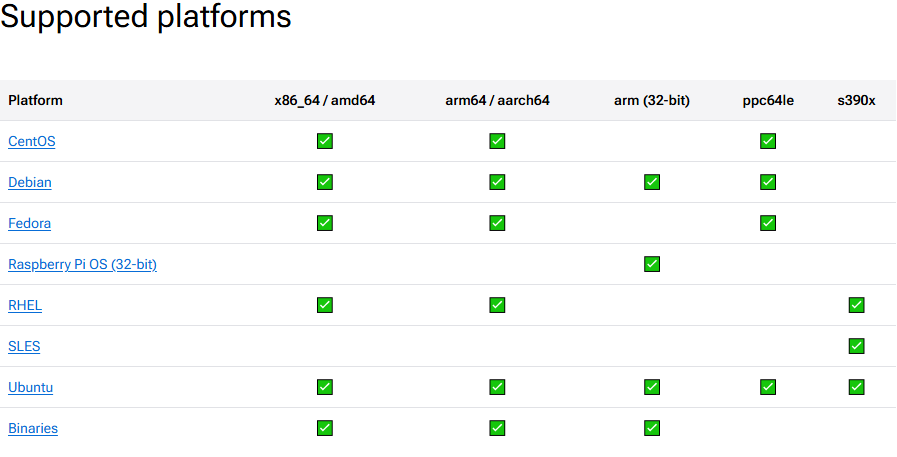


Рисунок 5.7 – Выбор инструкции в соответствии с используемой платформой для установки Docker

После успешной установки Docker Engine, необходимо установить плагин Docker Compose [7] в соответствии с операционной системой. Установка плагина производится с использованием пакетного менеджера.

После установки инструментов контейнеризации необходимо подготовить следующую структуру проекта:

* каталог log – используется для хранения логов брокера;
* каталог config – используется для хранения конфигурационного файла брокера mosquitto.conf и файла с учетными данными для авторизации пользователей password.txt.

Содержимое конфигурационного файла mosquitto.conf:

listener 1883

persistence true

persistence\_location /mosquitto/data/

log\_dest file /mosquitto/log/mosquitto.log

В корне проекта необходимо создать конфигурационный файл docker-compose.yaml со следующим содержимым:

version: '3.9'

services:

mosquitto:

image: eclipse-mosquitto:latest

container\_name: mqtt-broker

restart: unless-stopped

ports:

- "1883:1883"

- "9001:9001"

volumes:

- ./config/mosquitto.conf:

/mosquitto/config/mosquitto.conf:ro

- ./config/password.txt:

/mosquitto/config/password.txt:ro

- data:/mosquitto/data

- ./log:/mosquitto/log

volumes:

data:

name: mqtt-broker-data

В готовом проекте с использованием терминала необходимо запустить проект, используя Docker Compose и следующую команду:

$ docker-compose up -d

После выполнения команды будет загружен образ брокера, контейнер запустится в фоновом режиме и станет доступен для подключения по порту 1883. Для корректного подключения из внешней сети может потребоваться дополнительная настройка сетевого оборудования.

* 1. Руководство пользователя мобильного приложения

Для корректной работы мобильного приложения необходимо соблюдение следующих условий:

* наличие стабильного подключения к сети Интернет (Wi-Fi или мобильная передача данных) со скоростью не ниже 1 Мбит/с;
* операционная система Android версии 7.0 или выше.

Для установки мобильного приложения необходимо запустить исполняемый файл с расширением .apk на устройстве. После запуска установки следует подтвердить действие и дождаться завершения стандартной процедуры установки.

В случае, если система заблокирует установку, необходимо разрешить установку приложений из неизвестных источников. Для этого может потребоваться перейти в настройки устройства и активировать соответствующую опцию, которая обычно находится в разделе «Безопасность» или «Для разработчиков».

Процесс установки может незначительно отличаться в зависимости от версии операционной системы Android или модели устройства.

После завершения установки приложение готово к использованию. Для запуска необходимо нажать на иконку с соответствующим названием, отображенную на рабочем столе устройства показанную на рисунке 5.8.



Рисунок 5.8 – Установленное приложение

После запуска откроется начальный экран приложения (см. рисунок 5.9), содержащий окно авторизации и подключения к MQTT-брокеру системы «умный дом».

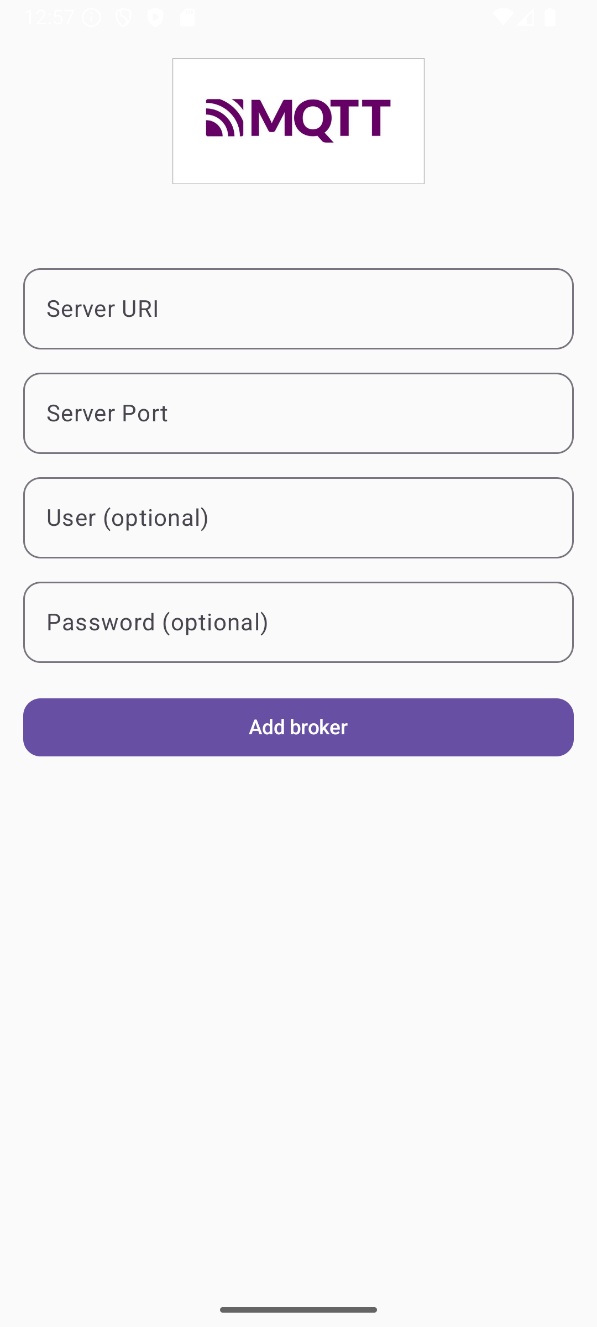


Рисунок 5.9 – Экран авторизации приложения

На экране авторизации отображается форма для ввода данных, необходимых для подключения к системе «умный дом». Форма содержит четыре поля:

* IP-адрес брокера – адрес сервера, на котором работает MQTT-брокер;
* порт – номер порта, используемого для соединения с брокером;
* имя пользователя – логин для авторизации (если требуется);
* пароль – пароль для авторизации (если требуется).

После заполнения обязательных полей пользователь инициирует проверку валидности данных, нажав кнопку добавления брокера. Приложение выполняет попытку установить соединение с указанным MQTT-брокером. В случае успешного подключения информация о системе сохраняется в локальную базу данных устройства, после чего на главном экране, под формой авторизации, появляется соответствующая карточка (см. рисунок 5.10) с информацией о системе и следующими элементами управления:

* кнопка «войти» – для перехода к управлению системой;
* кнопка «удалить» – для удаления записи о доступном подключении.

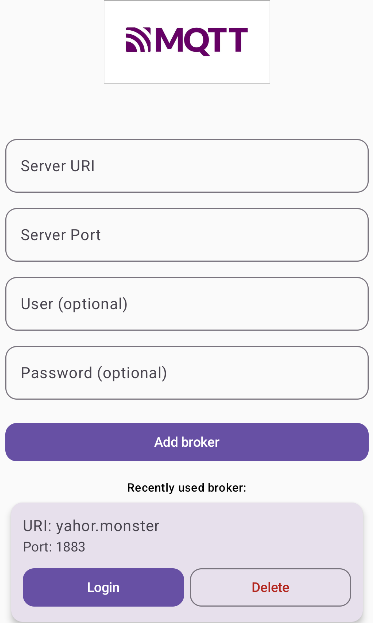


Рисунок 5.10 – Экран авторизации при успешном подключении

На этом этапе пользователю необходимо нажать на кнопку «войти» расположенное в карточке с информацией о доступном брокере.

* + 1. После успешного входа в систему пользователь попадает на главный экран управления системой умного дома. На этом экране отображаются все устройства, подключенные к координатору, с возможностью просмотра информации и управления. В случае отсутствия подключенных устройств, то на экране будет отображаться надпись «No connected devices» (см. рисунок 5.11).

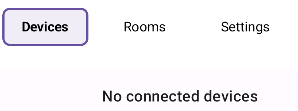


Рисунок 5.11 – Главный экран приложения

Для добавления новых устройств необходимо ознакомиться со списком поддерживаемых устройств на официальном сайте поставщика программного обеспечения SLS Smart Home [5]. Если устройство присутствует в списке поддерживаемых, необходимо перейти во вкладку «Settings», где доступна функция обнаружения и добавления новых устройств. (см. рисунок 5.12).

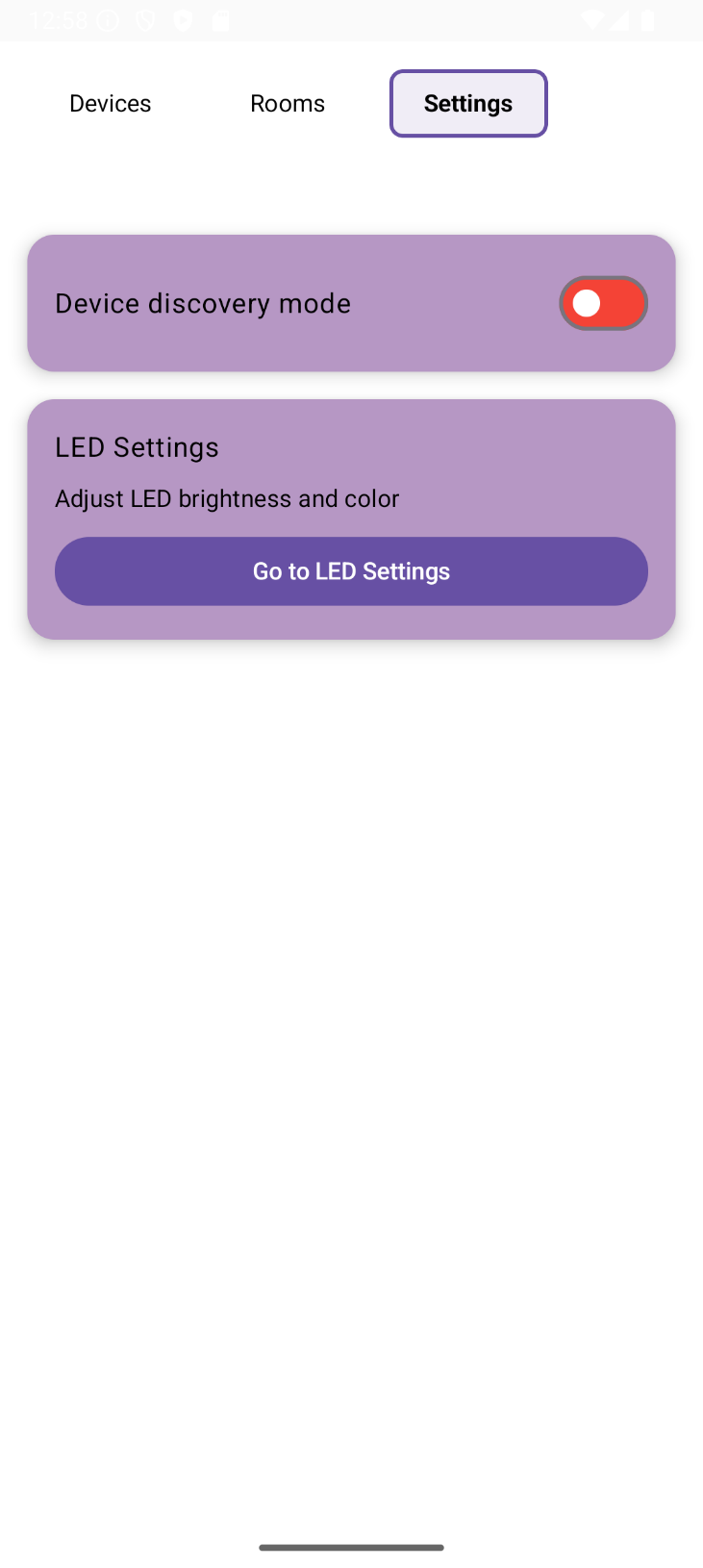


Рисунок 5.12 – Экран настроек

Для включения режима обнаружения необходимо установить значение переключателя в положение **«Включено»** – в этом случае он подсветится **зеленым цветом**. После активации на карточке отобразится **обратный отсчет времени (см. рисунок 5.13)**, в течение которого система будет искать новые устройства и устанавливать с ними соединение для обмена информацией.

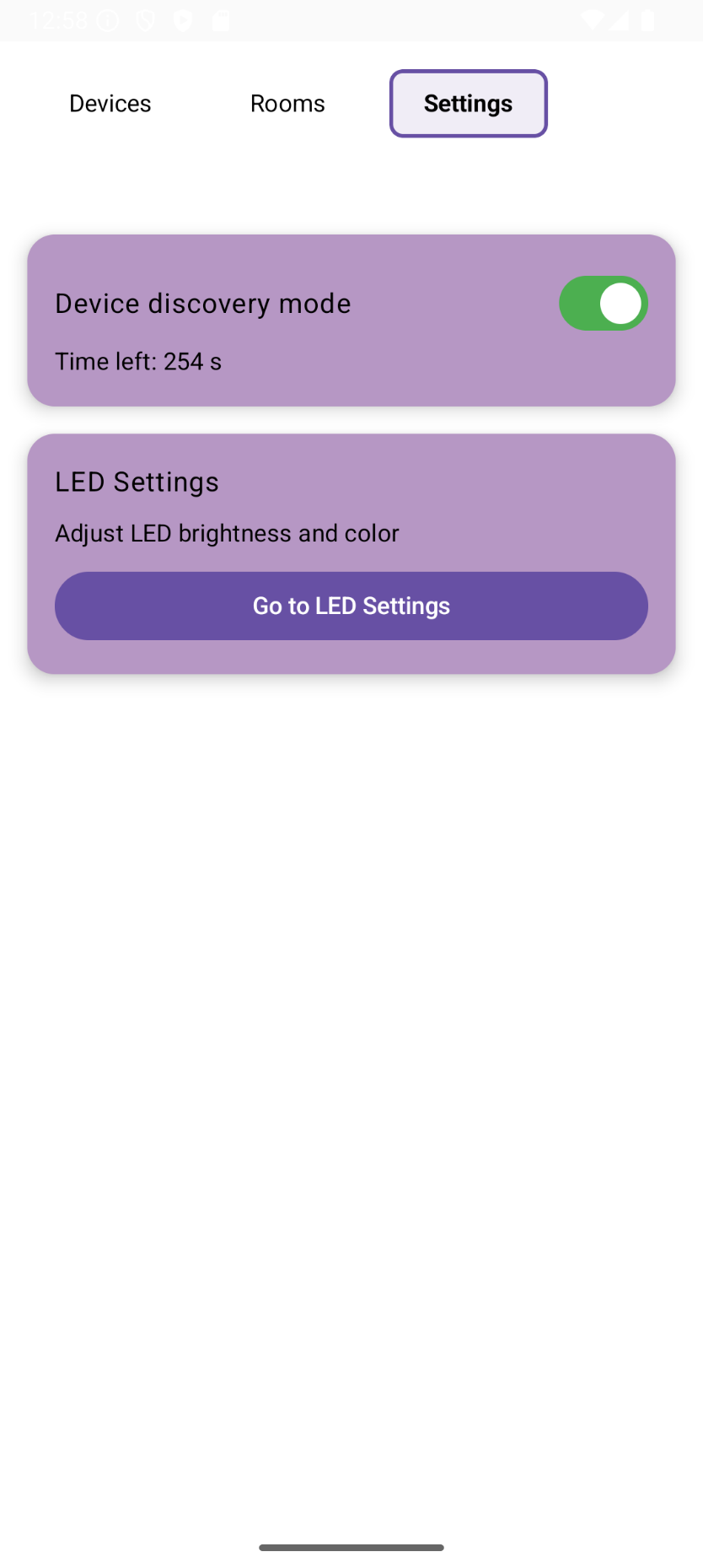


Рисунок 5.13 – Включенный режим обнаружения и сопряжения устройств

После активации режима обнаружения в мобильном приложении необходимо на оконечном устройстве выполнить действия для перевода его в режим сопряжения. Конкретный способ активации режима сопряжения зависит от типа устройства и его производителя – например, может потребоваться зажать кнопку на устройстве или выполнить определенную комбинацию действий, указанную в инструкции к устройству.

После выполнения вышеописанных действий устройство, как правило, появляется на главном экране приложения в течение нескольких секунд.

**После успешного добавления устройства** оно становится доступным для взаимодействия через мобильное приложение (см. рисунок 5.14).



Рисунок 5.14 – Главный экран приложения после подключения устройства

В случае, если устройство поддерживает управление, на его карточке в главном экране отображается соответствующий элемент управления. Как показано на рисунке 5.14, для розетки доступен **переключатель состояния** – при его нажатии розетка включается или выключается, а ее текущее состояние визуально отображается в приложении с помощью изменения цвета элемента управления.

Для просмотра дополнительной информации о конкретном устройстве пользователь может нажать на его карточку. После этого откроется экран с детализированной информацией, где можно изменить название устройства в приложении, привязать его к комнате для удобной территориальной группировки, а также просмотреть доступные параметры устройства. Пример такого экрана приведен на рисунке 5.15.

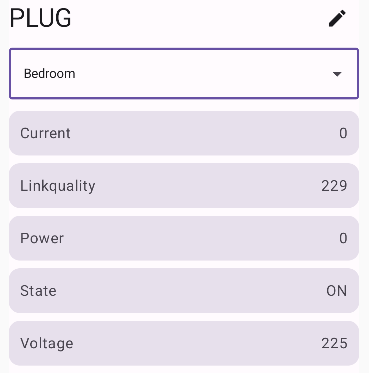


Рисунок 5.15 – Экран информации об устройстве

На данном экране пользователь может изменить привязку устройства к комнате, а также задать дружественное имя устройства в приложении для его более удобной идентификации.

* 1. Руководство программиста

Для сборки приложения, а также внесения изменений в его исходный код, необходимо использовать среду разработки Android Studio. Рекомендуемые системные требования для стабильной работы Android Studio следующие:

* операционная система: Windows 10 и выше;
* процессор: 4-ядерный, с тактовой частотой 2,6 ГГц;
* оперативная память: не менее 8 ГБ;
* свободное место на диске: не менее 15 ГБ.

Перед началом работы необходимо установить Android Studio, загрузить необходимые SDK и инструменты, а также импортировать проект в среду разработки.

Для установки IDE Android Studio необходимо перейти на официальный сайт разработчиков программного обеспечения [8] и на главной странице нажать «Download Android Studio» (см. рисунок 5.16) после чего начнется загрузка ПО. По завершению загрузки необходимо запустить исполняемый файл установщика.

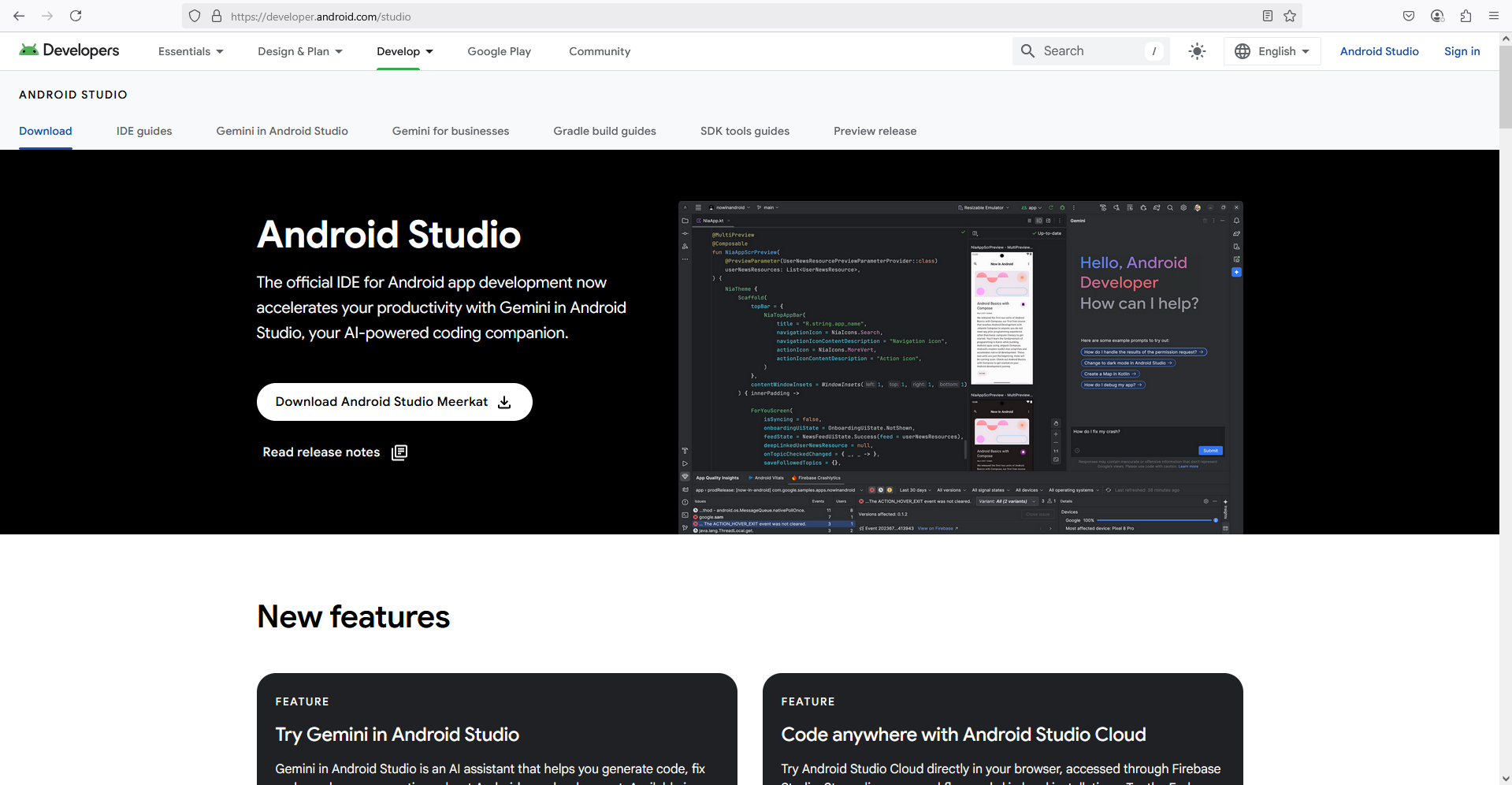


Рисунок 5.16 – Страница скачивания IDE Android Studio

В процессе подготовки среды разработки необходимо установить Android Studio — основную интегрированную среду для написания и отладки Android-приложений. Также требуется установить компонент Android Virtual Device (AVD), который позволяет создавать и запускать виртуальные устройства с различными конфигурациями для тестирования работы приложения без физического мобильного устройства Android. Установка и настройка этих компонентов показана на рисунке 5.17.

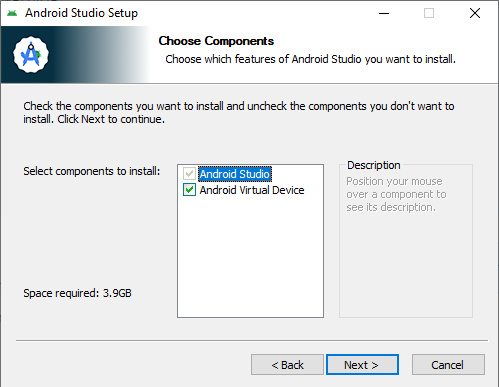


Рисунок 5.17 – Установка необходимых компонентов приложения

После завершения установки запустите Android Studio и при появлении окна настройки проекта выберите стандартные параметры (см. рисунок 5.18): это включает установку Android SDK и необходимых зависимостей, конфигурацию Gradle и подключение рекомендуемых плагинов для работы с разными версиями Android; после загрузки всех компонентов среда разработки будет готова к использованию.

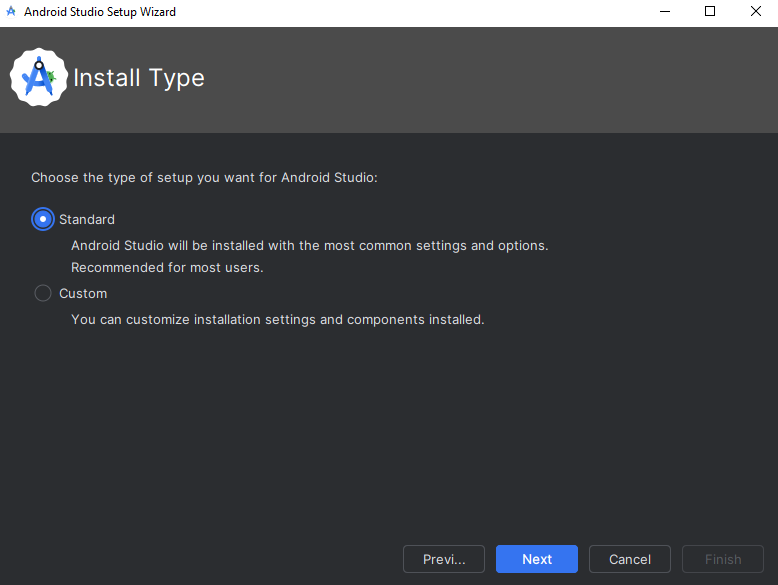


Рисунок 5.18 – Выбор режима установки приложения

Для работы с проектом необходимо открыть корневой каталог проекта «ha-application». На главном экране Android Studio нажмите кнопку «Open Project», затем в открывшемся диалоговом окне выберите папку с проектом и нажмите OK. После этого проект откроется в Android Studio, как показано на рисунке 5.19.

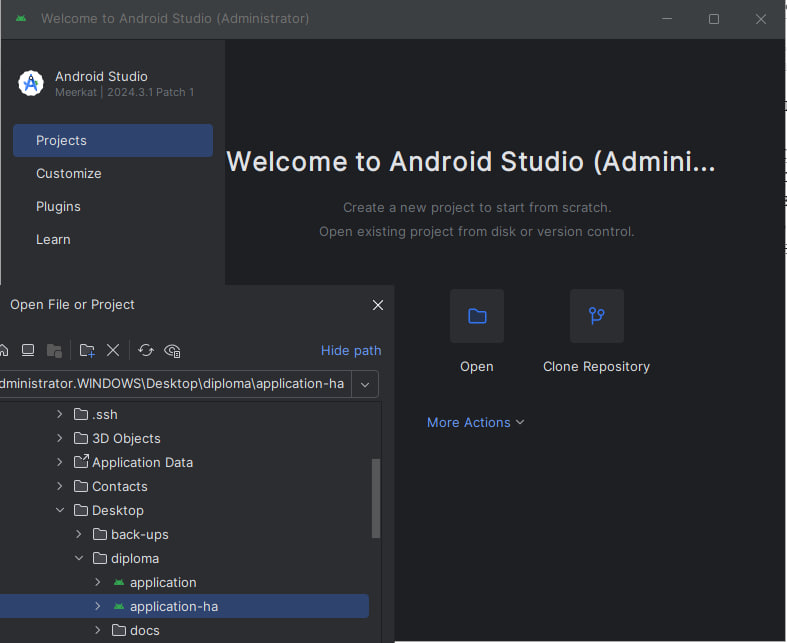


Рисунок 5.19 – Пример открытия проекта

После того как Android Studio завершит индексацию файлов и загрузку зависимостей проекта, можно приступать к запуску и внесению изменений в проект.

1. ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Этот раздел описывает программу и методику испытаний приложения. Целью тестирования является проверка правильности работы основных функций приложения, стабильности работы в различных условиях и соответствия требованиям, предъявляемым к пользовательскому интерфейсу и взаимодействию с устройствами по протоколу MQTT.

* 1. Автоматическое тестирование

Первый этап тестирования приложения – это автоматическое тестирование, направленное на проверку корректности работы отдельных компонентов на уровне исходного кода. Такие тесты позволяют выявить ошибки в логике до сборки и запуска приложения, что упрощает отладку и ускоряет процесс разработки.

В рамках автоматического тестирования производится проверка исходных файлов проекта, включая их соответствие стандартам кодирования, а также проверка версий используемых библиотек и зависимостей для выявления возможных несовместимостей.

Начало тестирования приложения включает проверку компонентов на уровне исходного кода. В качестве сборщика проекта используется Gradle, который выполняет следующие функции:

* проверка исходного кода на наличие логических или синтаксических ошибок;
* управление зависимостями проекта, включая проверку наличия библиотек, их версий и совместимости с другими библиотеками проекта;
* сборка проекта и компиляция исходного кода в единый исполняемый файл приложения;
* автоматический запуск unit-тестов.

Использование Gradle в качестве инструмента сборки проекта так же позволяет создавать детализированные отчёты о результатах тестирования и сборки приложения и результатов проверки стиля кода, что помогает улучшить качество приложения на протяжении всего жизненного цикла разработки.

На следующем этапе было проведено Unit-тестирование ключевых функциональных блоков приложения. Основное внимание уделялось проверке корректности подключения приложения к брокеру MQTT, а также обеспечению стабильного взаимодействия с базой данных.

Для этого были разработаны тесты, охватывающие следующий функционал:

* возможность подключения к MQTT брокеру;
* возможность публикации MQTT сообщений в топик;
* возможность подписки и получения сообщений из топика;
* корректность выполнения операций чтения, записи, изменения и удаления в базе данных.

Результаты тестирования представлены в таблице 6.1.

Таблица 6.1 – Результаты Unit-тестирования

|  |  |  |
| --- | --- | --- |
| Тестируемый функционал | Краткое описание теста | Результат тестирования |
| Подключение к MQTT брокеру | Проверка успешного подключения к MQTT брокеру с валидными данными | Успешно |
| Публикация MQTT сообщения | Проверка успешной публикации MQTT сообщений в топик «test/topic» | Успешно |
| Подписка на MQTT сообщения | Проверка возможности подписки на получение сообщений | Успешно |
| Получение MQTT сообщения | Проверка успешного получения MQTT сообщения из топика «test/topic» | Успешно |
| Проверка подключения к базе данных | Проверка успешного подключения к базе данных Room | Успешно |
| Проверка операций записи в базу данных | Проверка возможности добавления записей в таблицу Brokers | Успешно |
| Проверка операций чтения из базы данных | Проверка возможности получения данных из таблицы Brokers | Успешно |
| Проверка операций изменения в базе данных | Проверка возможности изменения данных в таблице Brokers | Успешно |
| Проверка операций удаления в базе данных | Проверка возможности удаления данных из таблицы Brokers | Успешно |

По результатам Unit-тестирования был успешно проверен весь функционал, связанный с получением, отправкой и обработкой данных в системе умного дома.

* 1. Ручное тестирование

На втором этапе проводилось ручное тестирование приложения, направленное на проверку его работы в условиях, приближенных к реальному использованию. Тестирование выполнялось как на виртуальном Android-устройстве (AVD), так и на физическом смартфоне, что позволило оценить поведение интерфейса, стабильность соединения и корректность обработки пользовательских действий.

Особое внимание уделялось взаимодействию приложения с реальными устройствами умного дома, подключенными в сеть Zigbee. В ходе тестирования использовались такие устройства, как:

* умная zigbee розетка \_TZ3000\_zloso4jk (TS011F) от производителя Tuya [9];
* датчик температуры и влажности \_TZ3000\_fllyghyj (TS0201) от производителя Tuya [10].

Каждый тест предполагает, что все функции, кроме тестируемой, работают корректно. Каждый тест-кейс включает описание шагов, которые необходимо выполнить для проверки конкретной функциональности приложения.

В таблице 6.2 представлено описание тест-кейсов, разработанных для проведения всестороннего испытания модуля авторизации пользователя и подключения приложения к MQTT-брокеру. Тест-кейсы охватывают основные сценарии использования модуля.

Таблица 6.2 – Описание тест-кейсов модуля авторизации и подключения

|  |  |  |
| --- | --- | --- |
| Наименование тест-кейса | Выполняемые шаги | Ожидаемый результат |
| Ввод не валидных данных для подключения | 1 Ввести существующий адрес брокера  2 Ввести отрицательное значение порта для подключения к брокеру 3 Нажать кнопку «Add Broker» | Toast-уведомление о некорректном вводе данных |
| Ввод валидных данных для подключения | 1 Ввести существующий адрес брокера  2 Ввести значение порта для подключения к брокеру  3 Нажать кнопку «Add Broker» | Добавление карточки для входа в систему на экране авторизации под формой ввода данных для подключения |
| Вход в систему | 1 Нажать кнопку «Login» на карточке последнего используемого брокера | Отображение главного экрана системы |
| Удаление данных о последнем используемом брокере | 1 Нажать кнопку «Delete» на карточке последнего используемого брокера | Удаление карточки последнего используемого брокера с экрана авторизации |

Приведённые тест-кейсы обеспечивают всестороннюю проверку функционирования модуля авторизации и подключения к MQTT-брокеру, включая обработку некорректного ввода, успешное подключение, вход в систему и управление сохранёнными данными о брокерах.

В таблице 6.3 представлены тест-кейсы для проверки функционала отвечающего за взаимодействие с устройствами умного дома. Эти тесты проверяют основные аспекты для работы с физическими устройствами, включая корректность отправки команд и получение текущего статуса устройств.

Таблица 6.3 – Описание тест-кейсов модуля устройств

|  |  |  |
| --- | --- | --- |
| Наименование тест-кейса | Выполняемые шаги | Ожидаемый результат |
| Просмотр списка подключенных устройств | 1 Открыть вкладку «Devices» | Отображение списка подключенных устройств к системе |
| Просмотр детальной информации о конкретном устройстве | 1 Открыть вкладку «Devices»  2 Нажать на карточку устройства | Отображение дружественного имени устройства, параметры привязки устройства к комнате, текущие параметры устройства |
| Изменение дружественного имени устройства в системе | 1 Открыть вкладку «Devices»  2 Нажать на карточку устройства  3 Нажать на иконку карандаша  4 Ввести новое имя устройства в форме 5 Нажать кнопку сохранить | Обновленное отображение нового имени устройства во вкладке детальной информации об устройстве и главном экране со списком устройств |
| Обновление параметра привязки устройства к комнате | 1 Открыть вкладку «Devices»  2 Нажать на карточку устройства  3 Нажать на выпадающий список комнат 4 Выбрать новую комнату для привязки устройства | Обновленное отображение комнаты в карточке устройства, устройство перемещено во вкладку новой комнаты на экране «Rooms» |
| Изменение состояния управляемого устройства | 1 Открыть вкладку «Devices» 2 На карточке умной розетки «Plug» нажать на соответствующий виджет | Изменение отображения виджета управления и режима работы умной розетки согласно команде: включено или выключено |

По результатам тестирования модуля взаимодействия с устройствами умного дома была подтверждена корректная работа всех основных функций. Проверено правильное отображение списка подключённых устройств и детальной информации о них, а также надёжность управления состоянием устройств. В процессе тестирования успешно выполнены операции по изменению настроек, таким как переименование устройств и обновление привязки к комнатам. Все тесты показали отсутствие ошибок и стабильную работу системы в условиях реального использования.

В таблице 6.4 представлено описание тестовых кейсов, разработанных для проверки функционала распределения устройств по комнатам внутри приложения. Эти тесты направлены на проверку корректности выполнения операций добавления, удаления и изменения привязки устройств к различным комнатам, а также на обеспечение правильного обновления данных в интерфейсе приложения при изменении конфигурации помещений.

Таблица 6.4 – Описание тест-кейсов модуля комнат

|  |  |  |
| --- | --- | --- |
| Наименование тест-кейса | Выполняемые шаги | Ожидаемый результат |
| Добавление новой комнаты в систему | 1 Открыть вкладку «Rooms»  2 Нажать на иконку «плюс» в нижнем правом углу экрана 3 Заполнить форму с названием новой комнаты 4 Нажать на кнопку добавить | Добавление карточки новой комнаты во вкладке «Rooms» |
| Удаление комнаты без привязанных устройств из системы | 1 Открыть вкладку «Rooms»  2 Нажать на иконку «крестик» на карточке комнаты к которой привязано 0 устройств | Удаление выбранной карточки комнаты |
| Удаление комнаты с привязанными устройствами из системы | 1 Открыть вкладку «Rooms»  2 Нажать на иконку «крестик» на карточке комнаты к которой привязано 1 и более устройств | Удаление выбранной карточки комнаты, при этом привязка у устройств, ранее связанных с этой комнатой, изменяется на «не привязано» |

По результатам тестирования функционала распределения устройств по комнатам все ключевые операции, такие как добавление и удаление комнат, были успешно выполнены. Система корректно обрабатывала удаление комнат с привязанными и без привязанных устройствами

В таблице 6.5 представлен процесс тестирования модуля настроек приложения, включая проверку работы с модулем LED-подсветки и системой обнаружения устройств.

Таблица 6.5 – Описание тест-кейсов модуля настроек

|  |  |  |
| --- | --- | --- |
| Наименование тест-кейса | Выполняемые шаги | Ожидаемый результат |
| 1 | 2 | 3 |
| Добавление устройства в систему | 1 Открыть вкладку «Settings»  2 Нажать на переключатель карточки «Enable Discovery Mode»  3 Включить режим сопряжения на устройстве | Отображение добавленного устройства во вкладке «Devices» |

Продолжение таблицы 6.5

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Включение автоматического режима настройки LED модуля | 1 Открыть вкладку «Settings»  2 На карточке «LED Settings» нажать на кнопку «Go to LED Settings» 3 В открывшейся вкладке нажать на выпадающий список и выбрать режим «AUTO» | Отключение модуля подсветки, активность только в случае изменение состояний координатора, например, включение режима обнаружения устройств |
| Включение ручного режима настройки LED модуля | 1 Открыть вкладку «Settings»  2 На карточке «LED Settings» нажать на кнопку «Go to LED Settings» 3 В открывшейся вкладке нажать на выпадающий список и выбрать режим «MANUAL» | Включение модуля подсветки с последними используемыми параметрами в ручном режиме, добавление ползунков для изменения яркости светодиодов и интенсивности каждого из цветов RGB |
| Изменение параметров подсветки LED модуля в ручном режиме | 1 Открыть вкладку «Settings»  2 На карточке «LED Settings» нажать на кнопку «Go to LED Settings» 3 Включить режим «MANUAL» 4 Передвинуть все ползунки в крайнее правое положение | Включение модуля LED с максимальной яркостью и интенсивности цветов |

По результатам тестирования модуль настроек приложения работал корректно: режим обнаружения устройств добавлял новые устройства в систему, автоматический режим LED-подсветки («AUTO») активировался при изменении состояния координатора, а ручной режим («MANUAL») обеспечивал точную регулировку яркости и цвета без ошибок.

* 1. Результаты тестирования

В результате проведённого тестирования приложение продемонстрировало стабильную и корректную работу во всех ключевых функциональных областях. Все тесты, включая проверки взаимодействия с устройствами, работу с комнатами, настройки и подсветку LED-модуля, были успешно пройдены без выявления критических ошибок. Таким образом, приложение соответствует заявленным требованиям и готово к использованию в рабочей среде.

1. ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ НА РЫНКЕ ANDROID-ПРИЛОЖЕНИЯ ДЛЯ УПРАВЛЕНИЯ И МОНИТОРИНГА УСТРОЙСТВАМИ УМНОГО ДОМА

* 1. Характеристика программного средства, разрабатываемого для реализации на рынке

Созданный дипломный проект представляет собой нативное приложение для операционной системы Android, которое позволяет пользователям управлять устройствами «умного дома» через протокол MQTT для DIY систем.

Целью разработки проекта является упрощение управления устройствами «умного дома» и обработки данных о их состоянии. Приложение предназначено для пользователей, позволяя им эффективно контролировать устройства, настраивать автоматизации и оперативно получать информацию об их работе.

Целевой аудиторией данного приложения являются пользователи систем «умного дома», которым необходим удобный инструмент для управления устройствами через протокол MQTT. Также потенциальными пользователями могут быть энтузиасты DIY-решений, использующие платформы, такие как HomeAssistant, а также владельцы экосистем Aqara, Xiaomi, Tuya и Яндекс.

На момент разработки существует большое количество решений для управления устройствами «умного дома», однако большинство из них ориентированы на конкретные экосистемы, такие как Aqara, Xiaomi, Tuya и Яндекс. Существующие приложения часто имеют ограниченный функционал и не поддерживают интеграцию с DIY-системами.

Планируется распространение приложения через Google Play с возможностью монетизации, включая бесплатную и расширенную платную версии.

* 1. Расчет инвестиций в разработку программного средства

Расчет зарплат на основную заработную плату разработчиков производится исходя из количества людей, которые занимаются разработкой программного продукта, месячной зарплаты каждого участника процесса разработки и сложности выполняемой ими работы. Затраты на основную заработную плату рассчитаны по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (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.

Разработкой всего приложения занимается инженер-программист, обязанности тестирования приложения лежат на инженере-тестировщике. Задачами инженера-программиста, который занимается являются создание модели данных, графического интерфейса, связи между моделью данных и графическим интерфейсом. Инженер-тестировщик занимается выявлением неработоспособных частей приложения.

Месячная заработная плата основана на медианных показателях для Junior инженера-программиста за 2024 год по Республике Беларусь, которая составляет примерно 807 долларов США в месяц, а для Junior инженера-тестировщика – 466 долларов США [11]. По состоянию на 21 февраля 2025 года, 1 доллар США по курсу Национального Банка Республики Беларусь составляет 3,2239 белорусских рубля.

В перерасчете на белорусские рубли месячные оклады для инженера-программиста и инженера-тестировщика составляют 2601,68 и 1502,33 белорусских рублей соответственно.

Часовой оклад исполнителей высчитывается путем деления месячного оклада на количество рабочих часов в месяце, то есть 160 часов.

За количество рабочих часов в месяце для инженера-программиста и инженера-тестировщика принято соответственно 196 и 32 часа. Коэффициент премии приравнивается к единице, так как она входит в общую сумму заработной платы. Затраты на заработную плату приведены в таблице:

Таблица 7.1 – Затраты на основную заработную плату сотрудников

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Месячный оклад, р | Часовой оклад, р | Трудоемкость работ, ч | Итого, р |
| Инженер-программист | 2601,68 | 16,26 | 196 | 3186,96 |
| Инженер-тестировщик | 1502,33 | 9,39 | 32 | 300,46 |
| Итого | | | | 3487,42 |
| Премия и стимулирующие зарплаты (0%) | | | | 0 |
| Общие затраты на основную заработную плату разработчиков | | | | 3487,42 |

Расчет затрат на дополнительную заработную плату разработчиков, 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 10%.

Размер дополнительной заработной составил:

Расчет отчислений на социальные нужды 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:

Расчет затрат на прочие расходов был произведен по формуле 7.4:

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

где Нпр – норматив прочих расходов.

Норматив прочих расходов был принят равным 30%.

Размер прочих расходов составил:

Расчет расходов на реализацию рассчитан по формуле 7.5:

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

где Нр – норматив расходов на реализацию

Норматив расходов на реализацию был принят 3%.

Размер расходов на реализацию 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 | 3 487,42 |
| Дополнительная заработная плата разработчиков | 348,72 |
| Отчисления на социальные нужды | 1 342,65 |
| Прочие расходы | 1 046,22 |
| Расходы на реализацию | 104,622 |
| Общая сумма инвестиций (затрат) на разработку | 6 329,64 |

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

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

Соответственно необходимо создать обоснование возможного объема продаж, количество проданных лицензий расширенной версии программного средства, купленного пользователями.

В Беларуси проживает около 9,2 миллиона человек, из которых примерно 8,48 миллионов являются активными интернет-пользователями [12]. По данным [13], доля пользователей Android среди мобильных ОС в Беларуси на 2024 год составляет 83%, что делает платформу наиболее популярной среди владельцев смартфонов.

Учитывая, что использование систем «умный дом» требует наличия совместимых устройств, примем, что 3% от общего числа активных интернет-пользователей уже обладают устройствами умного дома. Из них 80% (будут использовать приложения аналоги от производителей устройств, а оставшиеся 20% (42 200 человек) установят разработанное программное средство. Из них 5000 пользователей приобретут расширенную версию программного обеспечения.

С учетом цены на расширенную версию приложения, которая составляет 2,99 долларов США, и с учетом обменного курса доллара к белорусскому рублю, отпускная стоимость программного средства составит примерно 9,62 белорусских рубля.

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

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

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

– отпускная цена копии программного средства, р.;

 – 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 ставки налога, посчитаем НДС:

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

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

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

где *N*       – количество копий 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 с 1 января 2025 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 после внедрения программного обеспечения.

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 расчета рентабельности инвестиций (Return on Investment, ROI). Формула для расчета ROI:

|  |  |  |
| --- | --- | --- |
|  |  | (7.8) |

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 6 329,64 белорусских рублей, а отпускная цена установлена на уровне 9,62 белорусских рублей.

Прогнозируемый прирост чистой прибыли за год, основанный на предполагаемом объеме продаж в размере 5 000 расширенных версий в год, составляет 9 620 белорусских рублей. Рентабельность инвестиций за год оценивается в 20%.

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 возможной недостаточной оценкой продукта со стороны потребителей.

Такой показатель рентабельности может быть приемлем для стабильного бизнеса, но для увеличения доходности следует рассмотреть стратегии оптимизации затрат, повышения ценности продукта для пользователей и расширения рынка сбыта. Дополнительные инвестиции в маркетинг, улучшение функционала и внедрение подписочных моделей могут способствовать росту рентабельности в будущем.

## ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта было разработано мобильное приложение для операционной системы Android, предназначенное для мониторинга состояния и управления устройствами умного дома. Для обмена данными между мобильным устройством и координатором умного дома использовался протокол MQTT.

В ходе выполнения дипломного проекта были достигнуты следующие результаты:

* выполнена прошивка координатора и модуля Zigbee, а также проведена настройка шлюза «умного дома»;
* была спроектирована архитектура приложения системы «умный дом», учитывающая современные архитектурные подходы в разработке мобильных приложений;
* было разработано приложение для мобильных устройств операционной системы Android позволяющая управлять и просматривать параметры устройств системы «умный дом»;
* произведена отладка мобильного приложения и его тестирование в реальных пользовательских сценариях для управления и мониторинга параметров устройств.

В настоящее время отрасль умного дома развивается с огромной скоростью, что делает проект особенно актуальным и востребованным. Быстрое внедрение новых технологий и растущий интерес со стороны пользователей к автоматизации домашних процессов подчеркивают значимость данного решения на рынке.

В будущем планируется продолжить развитие проекта, расширив его функциональные возможности и улучшив производительность системы. Также будет проводиться дополнительное тестирование и доработка приложения с целью достижения максимальной стабильности и удобства для пользователей.

Таким образом, программное обеспечение было разработано в соответствии с поставленными задачами, весь заявленный функционал реализован в полном объеме. Архитектура приложения обеспечивает гибкость для добавления нового функционала без нарушения работы основной системы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] «Умный дом» Xiaomi [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.mi.com/ru/smart-home/. – Дата доступа: 12.02.2025.

[2] Умный дом с Алисой – комфортный и безопасный [Электронный ресурс]. – Электронные данные. – Режим доступа: https://alice.yandex.ru/smart-home. – Дата доступа: 13.02.2025.

[3] The ZigBee Protocol [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.netguru.com/blog/the-zigbee-protocol. – Дата доступа: 13.02.2025.

[4] MQTT: The Standart for IoT Messaging [Электронный ресурс]. – Электронные данные. – Режим доступа: https://mqtt.org. – Дата доступа: 14.02.2025.

[5] Supported Devices [Электронный ресурс]. – Электронные данные. – Режим доступа: https://slsys.io/en/action/supported\_devices. – Дата доступа: 15.04.2025.

[6] Install Docker Engine [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.docker.com/engine/install/. – Дата доступа: 18.04.2025.

[7] Install the Docker Compose [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.docker.com/compose/install/linux/. – Дата доступа: 18.02.2025.

[8] Android Studio [Электронный ресурс]. – Электронные данные. – Режим доступа: https://developer.android.com/studio. – Дата доступа: 19.02.2025.

[9] TS011F Plug [Электронный ресурс]. – Электронные данные. – Режим доступа: https://slsys.io/en/action/supported\_devices?device=66. – Дата доступа: 22.04.2025.

[10] TS0201\_TH [Электронный ресурс]. – Электронные данные. – Режим доступа: https://slsys.io/en/action/supported\_devices?device=239. – Дата доступа: 22.04.2025.

[11] Зарплата в IT | dev.by [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://salaries.devby.io/>. – Дата доступа: 23.02.2025.

[12] Какими смартфонами чаще всего пользуются Белорусы [Электронный ресурс]. – Электронные данные. – Режим доступа: https://bvn.by/2024/04/15/kakimi-smartfonami-chashhe-vsego-polzujutsja-belorusy. – Дата доступа: 23.02.2025.

[13] Статистика интернета и соцсетей на 2024 год – цифры и тренды в Беларуси [Электронный ресурс]. – Электронные данные. – Режим доступа: https://myfin.by/article/tekhnologii/tiktok-nabiraet-popularnost-u-belorusov-issledovanie. – Дата доступа: 23.02.2025.

[14] MQTT [Электронный ресурс]. – Электронные данные. – Режим доступа: https://en.wikipedia.org/wiki/MQTT. – Дата доступа: 14.02.2025.

[15] MQTT Broker: How It Works, Popular Options, and Quickstart [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.emqx.com/en/blog/the-ultimate-guide-to-mqtt-broker-comparison. – Дата доступа: 15.02.2025.

[16] Eclipse Mosquitto – An open source MQTT broker [Электронный ресурс]. – Электронные данные. – Режим доступа: https://mosquitto.org/. – Дата доступа: 15.02.2025.

[18] Comparing Software Architecture Patterns MVC Vs. MVVM Vs. MVP [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.masaischool.com/blog/comparing-software-architecture-patterns. – Дата доступа: 18.02.2025.

[19] Что такое MVVM-шаблон: как работает, преимущества и примеры использования [Электронный ресурс]. – Электронные данные. – Режим доступа: https://gitverse.ru/blog/articles/development/116-chto-takoe-mvvm-shablon-kak-rabotaet-preimushestva-i-primery-ispolzovaniya. – Дата доступа: 18.02.2025.

[20] Что такое MVC: рассказываем простыми словами [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.hexlet.io/blog/posts/chto-takoe-mvc-rasskazyvaem-prostymi-slovami. – Дата доступа: 18.02.2025.

[21] Kotlin Programming Language [Электронный ресурс]. – Электронные данные. – Режим доступа: https://kotlinlang.org/. – Дата доступа: 19.02.2025.

[22] Build Better apps faster with Jetpack Compose [Электронный ресурс]. – Электронные данные. – Режим доступа: https://developer.android.com/compose. – Дата доступа: 19.02.2025.

[23] Eclipse Paho [Электронный ресурс]. – Электронные данные. – Режим доступа: https://github.com/eclipse-paho. – Дата доступа: 10.03.2025.

ПРИЛОЖЕНИЕ А

(обязательное)

Текст программы

BrokerDAOTest.kt

0001. package com.example.iot\_ha

0002.

0003. import android.content.Context

0004. import androidx.room.Room

0005. import androidx.test.core.app.ApplicationProvider

0006. import androidx.test.ext.junit.runners.AndroidJUnit4

0007. import com.example.iot\_ha.data.local.RoomLocalDatabase

0008. import com.example.iot\_ha.data.local.broker.Broker

0009. import com.example.iot\_ha.data.local.broker.BrokerDAO

0010. import kotlinx.coroutines.runBlocking

0011. import org.junit.After

0012. import org.junit.Before

0013. import org.junit.Test

0014. import org.junit.runner.RunWith

0015. import com.google.common.truth.Truth.assertThat

0016.

0017. @RunWith(AndroidJUnit4::class)

0018. class BrokerDAOTest {

0019.

0020. private lateinit var database: RoomLocalDatabase

0021. private lateinit var brokerDao: BrokerDAO

0022.

0023. @Before

0024. fun setup() {

0025. val context = ApplicationProvider.getApplicationContext<Context>()

0026. database = Room.inMemoryDatabaseBuilder(

0027. context,

0028. RoomLocalDatabase::class.java

0029. ).allowMainThreadQueries()

0030. .build()

0031.

0032. brokerDao = database.brokerDAO()

0033. }

0034.

0035. @After

0036. fun teardown() {

0037. database.close()

0038. }

0039.

0040. @Test

0041. fun insertBroker\_andGetAllBrokers() = runBlocking {

0042. val broker = Broker(

0043. serverUri = "tcp://example.com",

0044. serverPort = 1883,

0045. user = "user",

0046. password = "pass"

0047. )

0048.

0049. brokerDao.insert(broker)

0050.

0051. val brokers = brokerDao.getAllBrokers()

0052.

0053. assertThat(brokers).isNotEmpty()

0054. assertThat(brokers.first().serverUri).isEqualTo("tcp://example.com")

0055. assertThat(brokers.first().serverPort).isEqualTo(1883)

0056. }

0057.

0058. @Test

0059. fun insertMultipleBrokers\_andGetLastBroker() = runBlocking {

0060. val broker1 = Broker(serverUri = "tcp://first.com", serverPort = 1883, user = null, password = null)

0061. val broker2 = Broker(serverUri = "tcp://second.com", serverPort = 1884, user = "admin", password = "adminpass")

0062.

0063. brokerDao.insert(broker1)

0064. brokerDao.insert(broker2)

0065.

0066. val lastBroker = brokerDao.getLastBroker()

0067.

0068. assertThat(lastBroker).isNotNull()

0069. assertThat(lastBroker?.serverUri).isEqualTo("tcp://second.com")

0070. }

0071.

0072. @Test

0073. fun insertAndDeleteBroker() = runBlocking {

0074. val broker = Broker(

0075. serverUri = "tcp://delete.com",

0076. serverPort = 1885,

0077. user = "delete",

0078. password = "deletepass"

0079. )

0080.

0081. brokerDao.insert(broker)

0082.

0083. val brokersBeforeDelete = brokerDao.getAllBrokers()

0084. assertThat(brokersBeforeDelete).hasSize(1)

0085.

0086. brokerDao.deleteBroker(brokersBeforeDelete.first())

0087.

0088. val brokersAfterDelete = brokerDao.getAllBrokers()

0089. assertThat(brokersAfterDelete).isEmpty()

0090. }

0091. }

ExampleInstrumentedTest.kt

0092. package com.example.iot\_ha

0093.

0094. import androidx.test.ext.junit.runners.AndroidJUnit4

0095. import androidx.test.platform.app.InstrumentationRegistry

0096. import org.junit.Assert.assertEquals

0097. import org.junit.Test

0098. import org.junit.runner.RunWith

0099.

0100. /\*\*

0101. \* Instrumented test, which will execute on an Android device.

0102. \*

0103. \* See [testing documentation](http://d.android.com/tools/testing).

0104. \*/

0105. @RunWith(AndroidJUnit4::class)

0106. class ExampleInstrumentedTest {

0107. @Test

0108. fun useAppContext() {

0109. // Context of the app under test.

0110. val appContext = InstrumentationRegistry.getInstrumentation().targetContext

0111. assertEquals("com.example.iot\_ha", appContext.packageName)

0112. }

0113. }

MQTTClientTest.kt

0114. package com.example.iot\_ha

0115.

0116. import android.util.Log

0117. import com.example.iot\_ha.data.local.broker.Broker

0118. import com.example.iot\_ha.data.mqtt.MQTTClient

0119. import com.example.iot\_ha.data.mqtt.domain.MQTTMessageHandler

0120. import org.eclipse.paho.client.mqttv3.MqttConnectOptions

0121. import org.eclipse.paho.client.mqttv3.MqttException

0122. import org.eclipse.paho.client.mqttv3.MqttMessage

0123. import org.junit.Before

0124. import org.junit.Test

0125. import org.junit.Assert.\*

0126. import java.util.concurrent.CountDownLatch

0127.

0128. class MQTTClientTest {

0129.

0130. private lateinit var mqttClient: MQTTClient

0131. private lateinit var broker: Broker

0132. private lateinit var messageHandler: MQTTMessageHandler

0133.

0134. @Before

0135. fun setUp() {

0136. broker = Broker(

0137. id = 1,

0138. serverUri = "yahor.monster",

0139. serverPort = 1883,

0140. user = null,

0141. password = null

0142. )

0143.

0144. mqttClient = MQTTClient.initialize(broker, null)

0145. }

0146.

0147. @Test

0148. fun testConnect() {

0149. val latch = CountDownLatch(1)

0150. val success = mqttClient.connect()

0151. assertTrue(success)

0152. assertNotNull(mqttClient)

0153. mqttClient.publish("test/topic", "Hello MQTT!", 1, false)

0154. mqttClient.disconnect()

0155. }

0156.

0157. @Test

0158. fun testSubscribeAndReceiveMessage() {

0159. val latch = CountDownLatch(1)

0160. mqttClient.connect()

0161. mqttClient.subscribe("test/topic")

0162. mqttClient.publish("test/topic", "Test message", 1, false)

0163. mqttClient.disconnect()

0164. }

0165. }

MainActivity.kt

0166. package com.example.iot\_ha

0167.

0168. import android.os.Bundle

0169. import androidx.activity.ComponentActivity

0170. import androidx.activity.compose.setContent

0171. import androidx.navigation.compose.rememberNavController

0172. import com.example.iot\_ha.ui.navigation.AppNavHost

0173.

0174. class MainActivity : ComponentActivity() {

0175. override fun onCreate(savedInstanceState: Bundle?) {

0176. super.onCreate(savedInstanceState)

0177.

0178. setContent {

0179. val navController = rememberNavController()

0180. AppNavHost(navController, "auth")

0181. }

0182. }

0183. }

0184.

MapTypeConvertor.kt

0185. package com.example.iot\_ha.data.local

0186.

0187. import androidx.room.TypeConverter

0188. import com.google.gson.Gson

0189. import com.google.gson.reflect.TypeToken

0190.

0191. class MapTypeConvertor {

0192. private val gson = Gson()

0193.

0194. @TypeConverter

0195. fun fromMap(map: Map<String, String>?): String? {

0196. return gson.toJson(map)

0197. }

0198.

0199. @TypeConverter

0200. fun toMap(json: String?): Map<String, String>? {

0201. return json?.let {

0202. val type = object : TypeToken<Map<String, String>>() {}.type

0203. gson.fromJson(it, type)

0204. }

0205. }

0206. }

0207.

RoomLocalDatabase.kt

0208. package com.example.iot\_ha.data.local

0209.

0210. import android.content.Context

0211. import androidx.room.Database

0212. import androidx.room.Room

0213. import androidx.room.RoomDatabase

0214. import androidx.room.TypeConverters

0215. import com.example.iot\_ha.data.local.broker.Broker

0216. import com.example.iot\_ha.data.local.broker.BrokerDAO

0217. import com.example.iot\_ha.data.local.command.Command

0218. import com.example.iot\_ha.data.local.command.CommandDAO

0219. import com.example.iot\_ha.data.local.device.Device

0220. import com.example.iot\_ha.data.local.device.DeviceDAO

0221. import com.example.iot\_ha.data.local.room.RoomEntity

0222. import com.example.iot\_ha.data.local.room.RoomEntityDAO

0223.

0224. @Database(

0225. entities = [Broker::class, Device::class, Command::class, RoomEntity::class],

0226. version = 1,

0227. exportSchema = false

0228. )

0229. @TypeConverters(MapTypeConvertor::class)

0230. abstract class RoomLocalDatabase : RoomDatabase() {

0231. abstract fun brokerDAO(): BrokerDAO

0232. abstract fun deviceDAO(): DeviceDAO

0233. abstract fun commandDAO(): CommandDAO

0234. abstract fun roomEntityDAO(): RoomEntityDAO

0235.

0236. companion object {

0237. @Volatile

0238. private var INSTANCE: RoomLocalDatabase? = null

0239.

0240. fun getInstance(context: Context): RoomLocalDatabase {

0241. return INSTANCE ?: synchronized(this) {

0242. val instance = Room.databaseBuilder(

0243. context.applicationContext,

0244. RoomLocalDatabase::class.java,

0245. "room\_local\_database"

0246. ).build()

0247. INSTANCE = instance

0248. instance

0249. }

0250. }

0251. }

0252. }

Broker.kt

0253. package com.example.iot\_ha.data.local.broker

0254.

0255. import androidx.room.Entity

0256. import androidx.room.PrimaryKey

0257.

0258. @Entity(tableName = "brokers")

0259. data class Broker(

0260. @PrimaryKey(autoGenerate = true) val id: Int = 0,

0261. val serverUri: String,

0262. val serverPort: Int,

0263. val user: String?,

0264. val password: String?

0265. )

0266.

BrokerDAO.kt

0267. package com.example.iot\_ha.data.local.broker

0268.

0269. import androidx.room.Dao

0270. import androidx.room.Delete

0271. import androidx.room.Insert

0272. import androidx.room.OnConflictStrategy

0273. import androidx.room.Query

0274.

0275. @Dao

0276. interface BrokerDAO {

0277. @Insert(onConflict = OnConflictStrategy.REPLACE)

0278. suspend fun insert(broker: Broker)

0279.

0280. @Query("SELECT \* FROM brokers")

0281. suspend fun getAllBrokers(): List<Broker>

0282.

0283. @Query("SELECT \* FROM brokers ORDER BY id DESC LIMIT 1")

0284. suspend fun getLastBroker(): Broker?

0285.

0286. @Delete

0287. suspend fun deleteBroker(broker: Broker)

0288. }

BrokerState.kt

0289. package com.example.iot\_ha.data.local.broker

0290.

0291. import kotlinx.coroutines.flow.MutableStateFlow

0292. import kotlinx.coroutines.flow.StateFlow

0293.

0294. object BrokerState {

0295. private val \_brokerId = MutableStateFlow<Int?>(null)

0296. val brokerId: StateFlow<Int?> = \_brokerId

0297.

0298. fun setBrokerId(id: Int) {

0299. \_brokerId.value = id

0300. }

0301. }

DiscoveryState.kt

0302. package com.example.iot\_ha.data.local.broker.discovery

0303.

0304. import com.example.iot\_ha.data.mqtt.MQTTClient

0305. import com.example.iot\_ha.data.mqtt.util.Topics

0306. import com.example.iot\_ha.utils.Constants

0307. import kotlinx.coroutines.CoroutineScope

0308. import kotlinx.coroutines.Dispatchers

0309. import kotlinx.coroutines.Job

0310. import kotlinx.coroutines.delay

0311. import kotlinx.coroutines.flow.MutableStateFlow

0312. import kotlinx.coroutines.flow.StateFlow

0313. import kotlinx.coroutines.launch

0314.

0315. object DiscoveryState {

0316. private val \_isDiscoveryActive = MutableStateFlow(false)

0317. val isDiscoveryActive: StateFlow<Boolean> = \_isDiscoveryActive

0318.

0319. private var resetJob: Job? = null

0320.

0321. fun startDiscovery() {

0322. \_isDiscoveryActive.value = true

0323. MQTTClient.publish(Topics.DISCOVERY\_TOPIC, Constants.DISCOVERY\_ENABLE)

0324.

0325. resetJob?.cancel()

0326.

0327. resetJob = CoroutineScope(Dispatchers.Default).launch {

0328. delay(Constants.DISCOVERY\_TIME)

0329. \_isDiscoveryActive.value = false

0330. }

0331. }

0332.

0333. fun stopDiscovery() {

0334. \_isDiscoveryActive.value = false

0335. MQTTClient.publish(Topics.DISCOVERY\_TOPIC, Constants.DISCOVERY\_DISABLE)

0336. resetJob?.cancel()

0337. }

0338. }

Command.kt

0339. package com.example.iot\_ha.data.local.command

0340.

0341. import androidx.room.Entity

0342. import androidx.room.ForeignKey

0343. import androidx.room.PrimaryKey

0344. import androidx.room.TypeConverters

0345. import com.example.iot\_ha.data.local.MapTypeConvertor

0346. import com.example.iot\_ha.data.local.device.Device

0347.

0348. @Entity(

0349. tableName = "commands",

0350. foreignKeys = [ForeignKey(

0351. entity = Device::class,

0352. parentColumns = ["id"],

0353. childColumns = ["deviceId"],

0354. onDelete = ForeignKey.CASCADE

0355. )]

0356. )

0357. data class Command(

0358. @PrimaryKey(autoGenerate = true) val id: Int = 0,

0359. val deviceId: Int,

0360.

0361. val commandTopic: String,

0362.

0363. // for switch commands

0364. val payloadOn: String?,

0365. val payloadOff: String?,

0366.

0367. // for select commands

0368. @TypeConverters(MapTypeConvertor::class) val options: Map<String, String>?,

0369. val commandTemplate: String?,

0370.

0371.

0372. val commandType: String,

0373. )

CommandDAO.kt

0374. package com.example.iot\_ha.data.local.command

0375.

0376. import androidx.room.Dao

0377. import androidx.room.Insert

0378. import androidx.room.OnConflictStrategy

0379. import androidx.room.Query

0380. import kotlinx.coroutines.flow.Flow

0381.

0382. @Dao

0383. interface CommandDAO {

0384.

0385. @Query("SELECT \* FROM commands WHERE commandTopic = :commandTopic LIMIT 1")

0386. suspend fun getCommandByCommandTopic(commandTopic: String): Command?

0387.

0388. @Query("SELECT \* FROM commands WHERE deviceId = :deviceId AND commandType = 'switch' LIMIT 1")

0389. suspend fun getSwitchCommandByDeviceId(deviceId: Int): Command?

0390.

0391. @Insert(onConflict = OnConflictStrategy.REPLACE)

0392. suspend fun insertCommand(command: Command)

0393.

0394. @Query("SELECT \* FROM commands WHERE commandType = :type")

0395. fun getCommandsByTypeFlow(type: String): Flow<List<Command>>

0396.

0397. @Query("SELECT \* FROM commands")

0398. fun getAllCommandsFlow(): Flow<List<Command>>

0399. }

Device.kt

0400. package com.example.iot\_ha.data.local.device

0401.

0402. import androidx.room.Entity

0403. import androidx.room.ForeignKey

0404. import androidx.room.Index

0405. import androidx.room.PrimaryKey

0406. import com.example.iot\_ha.data.local.broker.Broker

0407. import com.example.iot\_ha.data.local.room.RoomEntity

0408.

0409. @Entity(

0410. tableName = "devices",

0411. foreignKeys = [

0412. ForeignKey(

0413. entity = Broker::class,

0414. parentColumns = ["id"],

0415. childColumns = ["brokerId"],

0416. onDelete = ForeignKey.CASCADE

0417. ),

0418. ForeignKey(

0419. entity = RoomEntity::class,

0420. parentColumns = ["id"],

0421. childColumns = ["roomId"],

0422. onDelete = ForeignKey.SET\_NULL

0423. )

0424. ],

0425. indices = [Index(value = ["roomId"])]

0426. )

0427. data class Device(

0428. @PrimaryKey(autoGenerate = true) val id: Int = 0,

0429. val ieeeAddr: String,

0430. val friendlyName: String,

0431. val modelId: String,

0432. val topic: String,

0433. val roomId: Long?,

0434. val brokerId: Int

0435. ) {

0436. companion object {

0437. fun create(

0438. ieeeAddr: String,

0439. friendlyName: String,

0440. modelId: String,

0441. roomId: Long?,

0442. brokerId: Int

0443. ): Device {

0444. val topic = "zigbee/0x$ieeeAddr"

0445. val name = friendlyName.ifEmpty { "0x$ieeeAddr" }

0446. return Device(

0447. ieeeAddr = "0x$ieeeAddr",

0448. friendlyName = name,

0449. modelId = modelId,

0450. topic = topic,

0451. roomId = roomId,

0452. brokerId = brokerId

0453. )

0454. }

0455. }

0456. }

DeviceDAO.kt

0457. package com.example.iot\_ha.data.local.device

0458.

0459. import androidx.room.Dao

0460. import androidx.room.Delete

0461. import androidx.room.Insert

0462. import androidx.room.OnConflictStrategy

0463. import androidx.room.Query

0464. import androidx.room.Update

0465. import kotlinx.coroutines.flow.Flow

0466.

0467. @Dao

0468. interface DeviceDAO {

0469. @Query("SELECT \* FROM devices WHERE brokerId = :brokerId")

0470. suspend fun getDevicesByBroker(brokerId: Int): List<Device>

0471.

0472. @Query("SELECT \* FROM devices WHERE brokerId = :brokerId")

0473. fun getDevicesByBrokerFlow(brokerId: Int): Flow<List<Device>>

0474.

0475. @Query("SELECT \* FROM devices")

0476. suspend fun getAllDevices(): List<Device>

0477.

0478. @Query("SELECT \* FROM devices WHERE ieeeAddr = :ieeeAddr LIMIT 1")

0479. suspend fun getDeviceByIeeeAddr(ieeeAddr: String): Device?

0480.

0481. @Query("SELECT \* FROM devices WHERE id = :deviceId LIMIT 1")

0482. suspend fun getDeviceById(deviceId: Int): Device // ? был тут

0483.

0484. @Insert(onConflict = OnConflictStrategy.IGNORE)

0485. suspend fun insertDevice(device: Device): Long

0486.

0487. @Query("SELECT \* FROM devices WHERE roomId = :roomId")

0488. fun getDevicesByRoomIdFlow(roomId: Int): Flow<List<Device>>

0489.

0490. @Update

0491. suspend fun updateDevice(device: Device)

0492.

0493. @Query("SELECT COUNT(\*) FROM devices WHERE roomId = :roomId")

0494. suspend fun getDeviceCountForRoom(roomId: Long): Int

0495.

0496. @Query("SELECT \* FROM devices")

0497. fun getAllDevicesFlow(): Flow<List<Device>>

0498.

0499. @Delete

0500. suspend fun deleteDevice(device: Device)

0501.

0502. @Query("DELETE FROM devices WHERE id = :deviceId")

0503. suspend fun deleteDeviceById(deviceId: Int)

0504.

0505. }

DeviceState.kt

0506. package com.example.iot\_ha.data.local.device

0507.

0508. import kotlinx.coroutines.flow.MutableStateFlow

0509. import kotlinx.coroutines.flow.StateFlow

0510. import kotlinx.coroutines.flow.update

0511.

0512. object DeviceState {

0513. private val \_devicesData = MutableStateFlow<Map<Int, Map<String, Any>>>(emptyMap())

0514. val devicesData: StateFlow<Map<Int, Map<String, Any>>> = \_devicesData

0515.

0516. fun updateDeviceData(deviceId: Int, payload: String) {

0517. val parsedData: Map<String, Any> = parseJson(payload)

0518.

0519. \_devicesData.update { currentData ->

0520. currentData.toMutableMap().apply { this[deviceId] = parsedData }

0521. }

0522. }

0523.

0524.

0525. private fun parseJson(json: String): Map<String, Any> {

0526. return try {

0527. val jsonObject = org.json.JSONObject(json)

0528. jsonObject.keys().asSequence().associateWith { jsonObject.get(it) }

0529. } catch (e: Exception) {

0530. emptyMap()

0531. }

0532. }

0533.

0534. fun getDeviceValue(deviceId: Int, key: String): Any? {

0535. return \_devicesData.value[deviceId]?.get(key)

0536. }

0537. }

LEDState.kt

0538. package com.example.iot\_ha.data.local.led

0539.

0540. import kotlinx.coroutines.flow.MutableStateFlow

0541. import kotlinx.coroutines.flow.asStateFlow

0542.

0543. object LEDState {

0544. private val \_brightness = MutableStateFlow(127f)

0545. private val \_red = MutableStateFlow(127f)

0546. private val \_green = MutableStateFlow(127f)

0547. private val \_blue = MutableStateFlow(127f)

0548.

0549. val brightness = \_brightness.asStateFlow()

0550. val red = \_red.asStateFlow()

0551. val green = \_green.asStateFlow()

0552. val blue = \_blue.asStateFlow()

0553.

0554. fun setBrightness(value: Float) { \_brightness.value = value }

0555. fun setRed(value: Float) { \_red.value = value }

0556. fun setGreen(value: Float) { \_green.value = value }

0557. fun setBlue(value: Float) { \_blue.value = value }

0558. }

LEDStatus.kt

0559. package com.example.iot\_ha.data.local.led

0560.

0561. import org.json.JSONObject

0562.

0563. data class LEDStatus(

0564. val state: String = "ON",

0565. val brightness: Int,

0566. val red: Int,

0567. val green: Int,

0568. val blue: Int,

0569. val colorMode: String = "rgb",

0570. val mode: String = "manual"

0571. ) {

0572. fun toJson(): String {

0573. return JSONObject().apply {

0574. put("state", state)

0575. put("brightness", brightness)

0576. put("color", JSONObject().apply {

0577. put("r", red)

0578. put("g", green)

0579. put("b", blue)

0580. })

0581. put("color\_mode", colorMode)

0582. put("mode", mode)

0583. }.toString()

0584. }

0585. }

RoomEntity.kt

0586. package com.example.iot\_ha.data.local.room

0587.

0588. import androidx.room.Entity

0589. import androidx.room.PrimaryKey

0590.

0591. @Entity(tableName = "rooms")

0592. data class RoomEntity(

0593. @PrimaryKey(autoGenerate = true) val id: Long = 0,

0594. val name: String

0595. )

RoomEntityDAO.kt

0596. package com.example.iot\_ha.data.local.room

0597.

0598. import androidx.room.Dao

0599. import androidx.room.Insert

0600. import androidx.room.OnConflictStrategy

0601. import androidx.room.Query

0602. import kotlinx.coroutines.flow.Flow

0603.

0604. @Dao

0605. interface RoomEntityDAO {

0606. @Query("SELECT \* FROM rooms")

0607. fun getAllRooms(): Flow<List<RoomEntity>>

0608.

0609. @Insert(onConflict = OnConflictStrategy.REPLACE)

0610. suspend fun insertRoom(room: RoomEntity)

0611.

0612. @Query("DELETE FROM rooms WHERE id = :roomId")

0613. suspend fun deleteRoom(roomId: Int)

0614. }

MQTTClient.kt

0615. package com.example.iot\_ha.data.mqtt

0616.

0617. import android.util.Log

0618. import com.example.iot\_ha.data.local.broker.Broker

0619. import com.example.iot\_ha.data.mqtt.domain.MQTTMessageHandler

0620. import com.example.iot\_ha.data.mqtt.interfaces.MQTTConnection

0621. import com.example.iot\_ha.data.mqtt.interfaces.MQTTMessaging

0622. import com.example.iot\_ha.utils.logging.Logger

0623. import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken

0624. import org.eclipse.paho.client.mqttv3.MqttCallback

0625. import org.eclipse.paho.client.mqttv3.MqttClient

0626. import org.eclipse.paho.client.mqttv3.MqttConnectOptions

0627. import org.eclipse.paho.client.mqttv3.MqttException

0628. import org.eclipse.paho.client.mqttv3.MqttMessage

0629.

0630. object MQTTClient: MQTTMessaging, MQTTConnection {

0631. private var mqttClient: MqttClient? = null

0632. private var broker: Broker? = null

0633. private var messageHandler: MQTTMessageHandler? = null

0634.

0635. fun initialize(broker: Broker, handler: MQTTMessageHandler?): MQTTClient {

0636. this.broker = broker

0637. this.messageHandler = handler

0638. return this

0639. }

0640.

0641. fun reinitialize(newBroker: Broker, handler: MQTTMessageHandler): MQTTClient {

0642. disconnect()

0643. broker = newBroker

0644. messageHandler = handler

0645. return this

0646. }

0647.

0648. fun getInstance(): MQTTClient {

0649. return this

0650. }

0651.

0652. override fun connect(): Boolean {

0653. return try {

0654. val clientId = MqttClient.generateClientId()

0655. mqttClient =

0656. MqttClient("tcp://${broker?.serverUri}:${broker?.serverPort}", clientId, null)

0657.

0658. val options = MqttConnectOptions().apply {

0659. isAutomaticReconnect = true

0660. isCleanSession = true

0661. }

0662.

0663. mqttClient?.connect(options)

0664. Logger.log(MQTTClient::class, "Подключение успешно!")

0665. true

0666. } catch (e: MqttException) {

0667. Logger.log(MQTTClient::class, "Ошибка подключения: ${e.reasonCode} - ${e.message}")

0668. false

0669. }

0670. }

0671.

0672. override fun subscribe(topic: String) {

0673. try {

0674. mqttClient?.setCallback(object : MqttCallback {

0675. override fun connectionLost(cause: Throwable?) {

0676. Log.e("MQTT", "Соединение потеряно: ${cause?.message}")

0677. }

0678.

0679. override fun messageArrived(topic: String, message: MqttMessage) {

0680. val payload = message.toString()

0681. Log.i("MQTT", "📩 Получено сообщение: $message на топик: $topic")

0682.

0683. messageHandler?.handleMessage(topic, payload)

0684. }

0685.

0686. override fun deliveryComplete(token: IMqttDeliveryToken) {

0687. Log.i("MQTT", "Сообщение отправлено")

0688. }

0689. })

0690.

0691. mqttClient?.subscribe(topic)

0692. Log.i("MQTT", "📡 Подписался на $topic")

0693. } catch (e: MqttException) {

0694. Log.e("MQTT", "Ошибка подписки: ${e.reasonCode} - ${e.message}")

0695. }

0696. }

0697.

0698. override fun unsubscribe(topic: String) {

0699. try {

0700. mqttClient?.unsubscribe(topic)

0701. } catch (e: MqttException) {

0702. Log.e("MQTT", "Ошибка отписки: ${e.reasonCode} - ${e.message}")

0703. }

0704. }

0705.

0706. override fun publish(topic: String, payload: String, qos: Int, retained: Boolean) {

0707. try {

0708. val message = MqttMessage(payload.toByteArray()).apply {

0709. this.qos = qos

0710. this.isRetained = retained

0711. }

0712. mqttClient?.publish(topic, message)

0713. Log.i("MQTT", "📤 Отправлено сообщение: $payload в топик: $topic")

0714. } catch (e: MqttException) {

0715. Log.e("MQTT", "Ошибка отправки сообщения: ${e.reasonCode} - ${e.message}")

0716. }

0717. }

0718.

0719. override fun disconnect() {

0720. try {

0721. mqttClient?.disconnect()

0722. Log.i("MQTT", "🔌 Отключен от брокера")

0723. } catch (e: MqttException) {

0724. Log.e("MQTT", "Ошибка отключения: ${e.message}")

0725. }

0726. }

0727. }

MQTTMessageHandler.kt

0728. package com.example.iot\_ha.data.mqtt.domain

0729.

0730. import android.util.Log

0731. import com.example.iot\_ha.data.local.broker.BrokerState

0732. import com.example.iot\_ha.data.local.command.Command

0733. import com.example.iot\_ha.data.local.device.Device

0734. import com.example.iot\_ha.data.local.device.DeviceState

0735. import com.example.iot\_ha.data.local.led.LEDState

0736. import com.example.iot\_ha.data.mqtt.util.Topics

0737. import com.example.iot\_ha.ui.viewmodels.shared.DevicesViewModel

0738. import com.example.iot\_ha.ui.viewmodels.shared.SensorsViewModel

0739. import com.example.iot\_ha.utils.Constants

0740. import org.json.JSONObject

0741.

0742. class MQTTMessageHandler(

0743. private val sensorsViewModel: SensorsViewModel,

0744. private val devicesViewModel: DevicesViewModel,

0745. ) {

0746. fun handleMessage(topic: String, payload: String) {

0747. Log.i("MQTTHandler", "📩 Обрабатываем сообщение: $payload с топика: $topic")

0748.

0749. when {

0750. topic.startsWith(Topics.DEVICE\_STATE\_TOPIC) -> handleDeviceStateMessage(topic, payload)

0751. topic.startsWith(Topics.DEVICE\_COMMANDS\_TOPIC) -> handleDeviceCommandMessage(topic, payload)

0752. topic.startsWith(Topics.DEVICE\_LIST\_TOPIC) -> handleDeviceListMessage(payload)

0753. topic.startsWith(Topics.LED\_STATE\_TOPIC) -> handleLEDState(payload)

0754. else -> Log.i("MQTTHandler", "⚠ Необрабатываемый топик: $topic")

0755. }

0756. }

0757.

0758. private fun handleDeviceStateMessage(topic: String, payload: String) {

0759. val ieeeAddr = extractIeeeAddrFromTopic(topic)

0760. val deviceId = devicesViewModel.devices.value.find { it.ieeeAddr == ieeeAddr }?.id

0761.

0762. if (deviceId != null) {

0763. DeviceState.updateDeviceData(deviceId, payload)

0764. } else {

0765. println("Устройство с IEEE Addr $ieeeAddr не найдено")

0766. }

0767. }

0768.

0769. private fun handleLEDState(payload: String) {

0770. try {

0771. val json = JSONObject(payload)

0772. if (json.getString("state") == "ON") {

0773. LEDState.setBrightness(json.optInt("brightness", 127).toFloat())

0774. json.optJSONObject("color")?.let { color ->

0775. LEDState.setRed(color.optInt("r", 127).toFloat())

0776. LEDState.setGreen(color.optInt("g", 127).toFloat())

0777. LEDState.setBlue(color.optInt("b", 127).toFloat())

0778. }

0779. }

0780. } catch (e: Exception) {

0781. e.printStackTrace()

0782. }

0783. }

0784.

0785.

0786. private fun handleDeviceCommandMessage(topic: String, payload: String) {

0787. try {

0788. val jsonObject = JSONObject(payload)

0789.

0790. val commandTopic =

0791. jsonObject.optString("command\_topic").takeIf { it.isNotBlank() } ?: return

0792. val payloadOn = jsonObject.optString("payload\_on", null)

0793. val payloadOff = jsonObject.optString("payload\_off", null)

0794. val commandTemplate = jsonObject.optString("command\_template", null)

0795.

0796. val options = jsonObject.optJSONArray("options")?.let { array ->

0797. (0 until array.length()).associate { index ->

0798. array.getString(index) to array.getString(

0799. index

0800. )

0801. }

0802. } ?: emptyMap()

0803.

0804. val commandType = extractCommandTypeFromTopic(topic)

0805. val deviceIeeeAddr = extractIeeeAddrFromTopic(topic)

0806.

0807. if (deviceIeeeAddr != null && commandType != "unknown") {

0808. devicesViewModel.getDeviceIdByIeeeAddr(deviceIeeeAddr) { deviceId ->

0809. if (deviceId != null) {

0810. val command = Command(

0811. deviceId = deviceId,

0812. commandTopic = commandTopic,

0813. payloadOn = payloadOn,

0814. payloadOff = payloadOff,

0815. options = options,

0816. commandTemplate = commandTemplate,

0817. commandType = commandType

0818. )

0819.

0820. devicesViewModel.addCommandIfNotExists(command)

0821. Log.i("MQTTHandler", "📥 Команда сохранена: $command")

0822. } else {

0823. Log.e("DeviceId", "Device not found for IEEE Address: $deviceIeeeAddr")

0824. }

0825. }

0826. }

0827. } catch (e: Exception) {

0828. Log.e("MQTTHandler", "Ошибка обработки команды: ${e.message}")

0829. }

0830. }

0831.

0832. private fun extractCommandTypeFromTopic(topic: String): String {

0833. return when {

0834. topic.contains("/switch/", ignoreCase = true) -> Constants.SWITCH\_TYPE

0835. // topic.contains("/select/", ignoreCase = true) -> "select"

0836. // topic.contains("/light/", ignoreCase = true) -> "dimmer"

0837. else -> "unknown"

0838. }

0839. }

0840.

0841. private fun extractIeeeAddrFromTopic(topic: String): String? {

0842. val regex = Regex("0x[0-9A-Fa-f]+")

0843. return regex.find(topic)?.value

0844. }

0845.

0846. private fun handleDeviceListMessage(payload: String) {

0847. try {

0848. val jsonObject = JSONObject(payload)

0849.

0850. for (key in jsonObject.keys()) {

0851. val deviceJson = jsonObject.optJSONObject(key) ?: continue

0852.

0853. val ieeeAddr = deviceJson.optString("ieeeAddr")

0854. val friendlyName = deviceJson.optString("friendly\_name")

0855. val modelId = deviceJson.optString("ModelId")

0856.

0857. val device = Device.create(

0858. ieeeAddr = ieeeAddr,

0859. friendlyName = friendlyName,

0860. modelId = modelId,

0861. roomId = null,

0862. brokerId = BrokerState.brokerId.value ?: -1

0863. )

0864.

0865. Log.i("DEVICE", "📥 Получено устройство: $device")

0866.

0867. devicesViewModel.addDeviceIfNotExists(device)

0868. }

0869.

0870. } catch (e: Exception) {

0871. Log.e("DEVICE", "Ошибка обработки списка устройств: ${e.message}")

0872. }

0873. }

0874.

0875. }

0876.

MQTTConnection.kt

0877. package com.example.iot\_ha.data.mqtt.interfaces

0878.

0879. interface MQTTConnection {

0880. fun connect(): Boolean

0881. fun disconnect()

0882. }

MQTTMessaging.kt

0883. package com.example.iot\_ha.data.mqtt.interfaces

0884.

0885. interface MQTTMessaging {

0886. fun subscribe(topic: String)

0887. fun unsubscribe(topic: String)

0888. fun publish(topic: String, payload: String, qos: Int = 1, retained: Boolean = false)

0889. }

Topics.kt

0890. package com.example.iot\_ha.data.mqtt.util

0891.

0892. object Topics {

0893. const val DISCOVERY\_TOPIC = "zigbee/bridge/config/permit\_join"

0894.

0895. const val DEVICE\_STATE\_TOPIC = "zigbee/0x"

0896. const val SUBSCRIBE\_DEVICE\_STATE\_TOPIC = "zigbee/#"

0897.

0898. const val DEVICE\_COMMANDS\_TOPIC = "homeassistant/"

0899. const val SUBSCRIBE\_DEVICE\_COMMANDS\_TOPIC = "homeassistant/#"

0900.

0901. const val DEVICE\_LIST\_TOPIC = "devicelist"

0902. const val SUBSCRIBE\_DEVICE\_LIST\_TOPIC = "devicelist"

0903.

0904. const val LED\_STATE\_TOPIC = "zigbee/led"

0905. const val SUBSCRIBE\_LED\_STATE\_TOPIC = "zigbee/led"

0906. const val LED\_SET\_STATE\_TOPIC = "zigbee/led/set"

0907. }

BrokerInputForm.kt

0908. package com.example.iot\_ha.ui.components.broker

0909.

0910. import androidx.compose.foundation.layout.Column

0911. import androidx.compose.foundation.layout.Spacer

0912. import androidx.compose.foundation.layout.fillMaxWidth

0913. import androidx.compose.foundation.layout.height

0914. import androidx.compose.foundation.shape.RoundedCornerShape

0915. import androidx.compose.material3.Button

0916. import androidx.compose.material3.ButtonDefaults

0917. import androidx.compose.material3.MaterialTheme

0918. import androidx.compose.material3.Text

0919. import androidx.compose.runtime.Composable

0920. import androidx.compose.ui.Modifier

0921. import androidx.compose.ui.graphics.Color

0922. import androidx.compose.ui.unit.dp

0923. import com.example.iot\_ha.ui.components.common.CustomOutlinedTextField

0924.

0925. @Composable

0926. fun BrokerInputForm(

0927. serverUri: String,

0928. serverPort: String,

0929. user: String,

0930. password: String,

0931. onServerUriChange: (String) -> Unit,

0932. onServerPortChange: (String) -> Unit,

0933. onUserChange: (String) -> Unit,

0934. onPasswordChange: (String) -> Unit,

0935. onAddBroker: () -> Unit

0936. ) {

0937. Column(modifier = Modifier.fillMaxWidth()) {

0938. CustomOutlinedTextField(

0939. value = serverUri,

0940. label = "Server URI",

0941. onValueChange = onServerUriChange

0942. )

0943. CustomOutlinedTextField(

0944. value = serverPort,

0945. label = "Server Port",

0946. onValueChange = onServerPortChange

0947. )

0948. CustomOutlinedTextField(

0949. value = user,

0950. label = "User (optional)",

0951. onValueChange = onUserChange

0952. )

0953. CustomOutlinedTextField(

0954. value = password,

0955. label = "Password (optional)",

0956. onValueChange = onPasswordChange

0957. )

0958.

0959. Spacer(modifier = Modifier.height(12.dp))

0960.

0961. Button(

0962. onClick = onAddBroker,

0963. modifier = Modifier.fillMaxWidth(),

0964. shape = RoundedCornerShape(12.dp),

0965. colors = ButtonDefaults.buttonColors(containerColor = MaterialTheme.colorScheme.primary)

0966. ) {

0967. Text("Add broker", color = Color.White)

0968. }

0969. }

0970. }

BrokerItem.kt

0971. package com.example.iot\_ha.ui.components.broker

0972.

0973. import androidx.compose.foundation.layout.Arrangement

0974. import androidx.compose.foundation.layout.Column

0975. import androidx.compose.foundation.layout.Row

0976. import androidx.compose.foundation.layout.Spacer

0977. import androidx.compose.foundation.layout.fillMaxWidth

0978. import androidx.compose.foundation.layout.height

0979. import androidx.compose.foundation.layout.padding

0980. import androidx.compose.foundation.layout.width

0981. import androidx.compose.foundation.shape.RoundedCornerShape

0982. import androidx.compose.material3.Button

0983. import androidx.compose.material3.ButtonDefaults

0984. import androidx.compose.material3.Card

0985. import androidx.compose.material3.CardDefaults

0986. import androidx.compose.material3.MaterialTheme

0987. import androidx.compose.material3.OutlinedButton

0988. import androidx.compose.material3.Text

0989. import androidx.compose.runtime.Composable

0990. import androidx.compose.ui.Modifier

0991. import androidx.compose.ui.draw.shadow

0992. import androidx.compose.ui.graphics.Color

0993. import androidx.compose.ui.unit.dp

0994. import com.example.iot\_ha.data.local.broker.Broker

0995.

0996. @Composable

0997. fun BrokerItem(broker: Broker, onDelete: () -> Unit, onLogin: () -> Unit) {

0998. Card(

0999. modifier = Modifier

1000. .fillMaxWidth()

1001. .padding(6.dp)

1002. .shadow(4.dp, shape = RoundedCornerShape(12.dp)),

1003. elevation = CardDefaults.elevatedCardElevation(8.dp)

1004. ) {

1005. Column(modifier = Modifier.padding(12.dp)) {

1006. Text(text = "URI: ${broker.serverUri}", style = MaterialTheme.typography.bodyLarge)

1007. Text(text = "Port: ${broker.serverPort}", style = MaterialTheme.typography.bodyMedium)

1008. broker.user?.let {

1009. Text(

1010. text = "User: $it",

1011. style = MaterialTheme.typography.bodyMedium

1012. )

1013. }

1014. broker.password?.let {

1015. Text(

1016. text = "Password: ${"\*".repeat(it.length)}",

1017. style = MaterialTheme.typography.bodyMedium

1018. )

1019. }

1020.

1021. Spacer(modifier = Modifier.height(8.dp))

1022.

1023. Row(

1024. modifier = Modifier.fillMaxWidth(),

1025. horizontalArrangement = Arrangement.SpaceBetween

1026. ) {

1027. Button(

1028. onClick = onLogin,

1029. modifier = Modifier.weight(1f),

1030. shape = RoundedCornerShape(12.dp),

1031. colors = ButtonDefaults.buttonColors(containerColor = MaterialTheme.colorScheme.primary)

1032. ) {

1033. Text("Login", color = Color.White)

1034. }

1035. Spacer(modifier = Modifier.width(6.dp))

1036. OutlinedButton(

1037. onClick = onDelete,

1038. modifier = Modifier.weight(1f),

1039. shape = RoundedCornerShape(12.dp),

1040. colors = ButtonDefaults.outlinedButtonColors(contentColor = MaterialTheme.colorScheme.error)

1041. ) {

1042. Text("Delete")

1043. }

1044. }

1045. }

1046. }

1047. }

BrokerList.kt

1048. package com.example.iot\_ha.ui.components.broker

1049.

1050.

1051. import androidx.compose.foundation.layout.Spacer

1052. import androidx.compose.foundation.layout.height

1053. import androidx.compose.material3.MaterialTheme

1054. import androidx.compose.material3.Text

1055. import androidx.compose.runtime.Composable

1056. import androidx.compose.ui.Modifier

1057. import androidx.compose.ui.unit.dp

1058. import com.example.iot\_ha.data.local.broker.Broker

1059.

1060. @Composable

1061. fun BrokerList(

1062. brokers: List<Broker>,

1063. onDelete: (Broker) -> Unit,

1064. onLogin: (Broker) -> Unit

1065. ) {

1066. brokers.lastOrNull()?.let { broker ->

1067. Text("Recently used broker:", style = MaterialTheme.typography.labelMedium)

1068. Spacer(modifier = Modifier.height(1.dp))

1069. BrokerItem(

1070. broker = broker,

1071. onDelete = { onDelete(broker) },

1072. onLogin = { onLogin(broker) }

1073. )

1074. }

1075. }

CustomOutlinedTextField.kt

1076. package com.example.iot\_ha.ui.components.common

1077.

1078. import androidx.compose.foundation.layout.Spacer

1079. import androidx.compose.foundation.layout.fillMaxWidth

1080. import androidx.compose.foundation.layout.height

1081. import androidx.compose.foundation.shape.RoundedCornerShape

1082. import androidx.compose.material3.OutlinedTextField

1083. import androidx.compose.material3.Text

1084. import androidx.compose.runtime.Composable

1085. import androidx.compose.ui.Modifier

1086. import androidx.compose.ui.unit.dp

1087.

1088. @Composable

1089. fun CustomOutlinedTextField(

1090. value: String,

1091. label: String,

1092. onValueChange: (String) -> Unit,

1093. ) {

1094. OutlinedTextField(

1095. value = value,

1096. onValueChange = onValueChange,

1097. label = { Text(label) },

1098. shape = RoundedCornerShape(12.dp),

1099. modifier = Modifier.fillMaxWidth()

1100. )

1101. Spacer(modifier = Modifier.height(8.dp))

1102. }

TabButton.kt

1103. package com.example.iot\_ha.ui.components.common

1104.

1105. import androidx.compose.foundation.BorderStroke

1106. import androidx.compose.foundation.clickable

1107. import androidx.compose.foundation.layout.Arrangement

1108. import androidx.compose.foundation.layout.Row

1109. import androidx.compose.foundation.layout.height

1110. import androidx.compose.foundation.layout.padding

1111. import androidx.compose.foundation.shape.RoundedCornerShape

1112. import androidx.compose.material3.MaterialTheme

1113. import androidx.compose.material3.Surface

1114. import androidx.compose.material3.Text

1115. import androidx.compose.runtime.Composable

1116. import androidx.compose.ui.Alignment

1117. import androidx.compose.ui.Modifier

1118. import androidx.compose.ui.draw.clip

1119. import androidx.compose.ui.graphics.Color

1120. import androidx.compose.ui.text.font.FontWeight

1121. import androidx.compose.ui.unit.dp

1122.

1123. @Composable

1124. fun TabButton(title: String, isSelected: Boolean, onClick: () -> Unit) {

1125. val borderColor = if (isSelected) MaterialTheme.colorScheme.primary else Color.Transparent

1126. val textColor = Color.Black

1127. Surface(

1128. modifier = Modifier

1129. .padding(horizontal = 4.dp, vertical = 4.dp)

1130. .height(40.dp)

1131. .clip(RoundedCornerShape(8.dp))

1132. .clickable { onClick() },

1133. color = if (isSelected) MaterialTheme.colorScheme.primary.copy(alpha = 0.1f) else Color.Transparent,

1134. shape = RoundedCornerShape(8.dp),

1135. border = BorderStroke(if (isSelected) 2.dp else 0.dp, borderColor)

1136. ) {

1137. Row(

1138. modifier = Modifier.padding(horizontal = 20.dp, vertical = 8.dp),

1139. verticalAlignment = Alignment.CenterVertically,

1140. horizontalArrangement = Arrangement.Center

1141. ) {

1142. Text(

1143. text = title,

1144. color = textColor,

1145. style = MaterialTheme.typography.labelLarge,

1146. fontWeight = if (isSelected) FontWeight.Bold else FontWeight.Normal

1147. )

1148. }

1149. }

1150. }

DeviceCard.kt

1151. package com.example.iot\_ha.ui.components.devices

1152.

1153. import android.util.Log

1154. import androidx.compose.foundation.Image

1155. import androidx.compose.foundation.clickable

1156. import androidx.compose.foundation.layout.Column

1157. import androidx.compose.foundation.layout.Row

1158. import androidx.compose.foundation.layout.Spacer

1159. import androidx.compose.foundation.layout.fillMaxWidth

1160. import androidx.compose.foundation.layout.padding

1161. import androidx.compose.foundation.layout.size

1162. import androidx.compose.foundation.layout.width

1163. import androidx.compose.foundation.shape.RoundedCornerShape

1164. import androidx.compose.material3.Card

1165. import androidx.compose.material3.CardDefaults

1166. import androidx.compose.material3.DropdownMenuItem

1167. import androidx.compose.material3.ExperimentalMaterial3Api

1168. import androidx.compose.material3.ExposedDropdownMenuBox

1169. import androidx.compose.material3.ExposedDropdownMenuDefaults

1170. import androidx.compose.material3.Slider

1171. import androidx.compose.material3.SliderDefaults

1172. import androidx.compose.material3.Switch

1173. import androidx.compose.material3.SwitchDefaults

1174. import androidx.compose.material3.Text

1175. import androidx.compose.material3.TextField

1176. import androidx.compose.runtime.Composable

1177. import androidx.compose.runtime.getValue

1178. import androidx.compose.runtime.mutableFloatStateOf

1179. import androidx.compose.runtime.mutableStateOf

1180. import androidx.compose.runtime.remember

1181. import androidx.compose.runtime.setValue

1182. import androidx.compose.ui.Alignment

1183. import androidx.compose.ui.Modifier

1184. import androidx.compose.ui.draw.clip

1185. import androidx.compose.ui.graphics.Color

1186. import androidx.compose.ui.res.painterResource

1187. import androidx.compose.ui.text.font.FontWeight

1188. import androidx.compose.ui.unit.dp

1189. import androidx.compose.ui.unit.sp

1190. import androidx.navigation.NavHostController

1191.

1192. @OptIn(ExperimentalMaterial3Api::class)

1193. @Composable

1194. fun DeviceCard(

1195. deviceId: Int,

1196. imageRes: Int,

1197. name: String,

1198. type: String? = null,

1199. value: Any? = null,

1200. navController: NavHostController,

1201. onToggle: ((Boolean) -> Unit)? = null,

1202. onSliderChange: ((Float) -> Unit)? = null,

1203. onSelectChange: ((String) -> Unit)? = null,

1204. options: List<String> = emptyList()

1205. ) {

1206. var checked by remember { mutableStateOf(value as? Boolean ?: false) }

1207. var sliderValue by remember { mutableFloatStateOf(value as? Float ?: 0f) }

1208. var selectedOption by remember {

1209. mutableStateOf(value as? String ?: options.firstOrNull().orEmpty())

1210. }

1211. var expanded by remember { mutableStateOf(false) }

1212.

1213. Card(

1214. shape = RoundedCornerShape(16.dp),

1215. modifier = Modifier

1216. .fillMaxWidth()

1217. .padding(8.dp)

1218. .clickable { navController.navigate("device\_details/$deviceId") },

1219. elevation = CardDefaults.cardElevation(6.dp)

1220. ) {

1221. Row(

1222. verticalAlignment = Alignment.CenterVertically,

1223. modifier = Modifier.padding(16.dp)

1224. ) {

1225. Image(

1226. painter = painterResource(id = imageRes),

1227. contentDescription = name,

1228. modifier = Modifier

1229. .size(64.dp)

1230. .clip(RoundedCornerShape(12.dp))

1231. )

1232. Spacer(modifier = Modifier.width(16.dp))

1233. Column(modifier = Modifier.weight(1f)) {

1234. Text(

1235. text = name,

1236. fontSize = 20.sp,

1237. fontWeight = FontWeight.Bold,

1238. color = Color.Black

1239. )

1240.

1241. when (type) {

1242. "switch" -> Switch(

1243. checked = checked,

1244. onCheckedChange = {

1245. checked = it

1246. onToggle?.invoke(it)

1247. },

1248. colors = SwitchDefaults.colors(

1249. checkedThumbColor = Color.White,

1250. checkedTrackColor = Color(0xFF4CAF50),

1251. uncheckedThumbColor = Color.White,

1252. uncheckedTrackColor = Color(0xFFF44336)

1253. )

1254. )

1255.

1256. "slider" -> Slider(

1257. value = sliderValue,

1258. onValueChange = {

1259. sliderValue = it

1260. onSliderChange?.invoke(it)

1261. },

1262. valueRange = 0f..100f,

1263. colors = SliderDefaults.colors(

1264. thumbColor = Color(0xFF8A9F9B),

1265. activeTrackColor = Color(0xFFA6B6A9)

1266. )

1267. )

1268.

1269. "select" -> ExposedDropdownMenuBox(

1270. expanded = expanded,

1271. onExpandedChange = { expanded = it }

1272. ) {

1273. TextField(

1274. value = selectedOption,

1275. onValueChange = {},

1276. readOnly = true,

1277. modifier = Modifier.menuAnchor(),

1278. label = { Text("Выберите") },

1279. trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = expanded) }

1280. )

1281. ExposedDropdownMenu(

1282. expanded = expanded,

1283. onDismissRequest = { expanded = false }

1284. ) {

1285. options.forEach { option ->

1286. DropdownMenuItem(

1287. text = { Text(option) },

1288. onClick = {

1289. selectedOption = option

1290. expanded = false

1291. onSelectChange?.invoke(option)

1292. }

1293. )

1294. }

1295. }

1296. }

1297.

1298. else -> {

1299. value?.let {

1300. Text(text = it.toString(), fontSize = 14.sp, color = Color.Black)

1301. }

1302. }

1303. }

1304. }

1305. }

1306. }

1307. }

1308.

DeviceDetails.kt

1309. package com.example.iot\_ha.ui.components.devices

1310.

1311. import androidx.compose.foundation.background

1312. import androidx.compose.foundation.layout.Arrangement

1313. import androidx.compose.foundation.layout.Box

1314. import androidx.compose.foundation.layout.Column

1315. import androidx.compose.foundation.layout.Row

1316. import androidx.compose.foundation.layout.fillMaxWidth

1317. import androidx.compose.foundation.layout.padding

1318. import androidx.compose.foundation.shape.RoundedCornerShape

1319. import androidx.compose.material3.MaterialTheme

1320. import androidx.compose.material3.Text

1321. import androidx.compose.runtime.Composable

1322. import androidx.compose.ui.Alignment

1323. import androidx.compose.ui.Modifier

1324. import androidx.compose.ui.draw.clip

1325. import androidx.compose.ui.unit.dp

1326. import androidx.compose.ui.unit.sp

1327.

1328. @Composable

1329. fun DeviceDetails(deviceData: Map<String, Any>) {

1330. Column(modifier = Modifier.fillMaxWidth()) {

1331. deviceData.forEach { (key, value) ->

1332. Box(

1333. modifier = Modifier

1334. .fillMaxWidth()

1335. .padding(vertical = 4.dp)

1336. .clip(RoundedCornerShape(12.dp))

1337. .background(MaterialTheme.colorScheme.surfaceVariant)

1338. .padding(12.dp)

1339. ) {

1340. Row(

1341. modifier = Modifier.fillMaxWidth(),

1342. horizontalArrangement = Arrangement.SpaceBetween,

1343. verticalAlignment = Alignment.CenterVertically

1344. ) {

1345. Text(

1346. text = key.replaceFirstChar { if (it.isLowerCase()) it.titlecase() else it.toString() },

1347. style = MaterialTheme.typography.bodyLarge.copy(fontSize = 16.sp),

1348. color = MaterialTheme.colorScheme.onSurfaceVariant

1349. )

1350. Text(

1351. text = value.toString(),

1352. style = MaterialTheme.typography.bodyLarge.copy(fontSize = 16.sp),

1353. color = MaterialTheme.colorScheme.onSurfaceVariant

1354. )

1355. }

1356. }

1357. }

1358. }

1359. }

DeviceNotFound.kt

1360. package com.example.iot\_ha.ui.components.devices

1361.

1362. import androidx.compose.foundation.layout.Box

1363. import androidx.compose.foundation.layout.fillMaxSize

1364. import androidx.compose.material3.MaterialTheme

1365. import androidx.compose.material3.Text

1366. import androidx.compose.runtime.Composable

1367. import androidx.compose.ui.Alignment

1368. import androidx.compose.ui.Modifier

1369.

1370. @Composable

1371. fun DeviceNotFoundMessage() {

1372. Box(

1373. modifier = Modifier.fillMaxSize(),

1374. contentAlignment = Alignment.Center

1375. ) {

1376. Text(

1377. text = "Device not found",

1378. style = MaterialTheme.typography.bodyLarge,

1379. color = MaterialTheme.colorScheme.error

1380. )

1381. }

1382. }

DeviceTitle.kt

1383. package com.example.iot\_ha.ui.components.devices

1384.

1385. import androidx.compose.foundation.layout.Row

1386. import androidx.compose.foundation.layout.fillMaxWidth

1387. import androidx.compose.foundation.layout.padding

1388. import androidx.compose.foundation.layout.statusBarsPadding

1389. import androidx.compose.material.icons.Icons

1390. import androidx.compose.material.icons.filled.Edit

1391. import androidx.compose.material3.Icon

1392. import androidx.compose.material3.IconButton

1393. import androidx.compose.material3.MaterialTheme

1394. import androidx.compose.material3.Text

1395. import androidx.compose.runtime.Composable

1396. import androidx.compose.ui.Alignment

1397. import androidx.compose.ui.Modifier

1398. import androidx.compose.ui.unit.dp

1399.

1400. @Composable

1401. fun DeviceTitle(

1402. friendlyName: String,

1403. onEditClick: () -> Unit

1404. ) {

1405. Row(

1406. verticalAlignment = Alignment.CenterVertically,

1407. modifier = Modifier.fillMaxWidth()

1408. ) {

1409. Text(

1410. text = friendlyName,

1411. style = MaterialTheme.typography.headlineMedium,

1412. modifier = Modifier.weight(1f)

1413. )

1414. IconButton(

1415. onClick = onEditClick,

1416. modifier = Modifier.padding(start = 8.dp)

1417. ) {

1418. Icon(

1419. imageVector = Icons.Default.Edit,

1420. contentDescription = "Edit device name"

1421. )

1422. }

1423. }

1424. }

AddRoomDialog.kt

1425. package com.example.iot\_ha.ui.components.rooms

1426.

1427. import androidx.compose.foundation.layout.Column

1428. import androidx.compose.material3.AlertDialog

1429. import androidx.compose.material3.Button

1430. import androidx.compose.material3.Text

1431. import androidx.compose.runtime.Composable

1432. import androidx.compose.runtime.getValue

1433. import androidx.compose.runtime.mutableStateOf

1434. import androidx.compose.runtime.remember

1435. import androidx.compose.runtime.setValue

1436. import com.example.iot\_ha.ui.components.common.CustomOutlinedTextField

1437.

1438. @Composable

1439. fun AddRoomDialog(

1440. onDismiss: () -> Unit,

1441. onConfirm: (String) -> Unit

1442. ) {

1443. var roomName by remember { mutableStateOf("") }

1444.

1445. AlertDialog(

1446. onDismissRequest = onDismiss,

1447. confirmButton = {

1448. Button(onClick = {

1449. if (roomName.isNotBlank()) {

1450. onConfirm(roomName)

1451. onDismiss()

1452. }

1453. }) {

1454. Text("Добавить")

1455. }

1456. },

1457. dismissButton = {

1458. Button(onClick = onDismiss) {

1459. Text("Отмена")

1460. }

1461. },

1462. title = { Text("Добавить комнату") },

1463. text = {

1464. Column {

1465. Text("Введите название комнаты:")

1466. CustomOutlinedTextField(

1467. value = roomName,

1468. label = "Название комнаты",

1469. onValueChange = { roomName = it }

1470. )

1471. }

1472. }

1473. )

1474. }

RoomCard.kt

1475. package com.example.iot\_ha.ui.components.rooms

1476.

1477. import androidx.compose.foundation.Image

1478. import androidx.compose.foundation.background

1479. import androidx.compose.foundation.clickable

1480. import androidx.compose.foundation.layout.Box

1481. import androidx.compose.foundation.layout.Column

1482. import androidx.compose.foundation.layout.Row

1483. import androidx.compose.foundation.layout.Spacer

1484. import androidx.compose.foundation.layout.fillMaxWidth

1485. import androidx.compose.foundation.layout.padding

1486. import androidx.compose.foundation.layout.size

1487. import androidx.compose.foundation.layout.width

1488. import androidx.compose.foundation.shape.RoundedCornerShape

1489. import androidx.compose.material.icons.Icons

1490. import androidx.compose.material.icons.filled.Delete

1491. import androidx.compose.material3.Card

1492. import androidx.compose.material3.CardDefaults

1493. import androidx.compose.material3.Icon

1494. import androidx.compose.material3.IconButton

1495. import androidx.compose.material3.MaterialTheme

1496. import androidx.compose.material3.Text

1497. import androidx.compose.runtime.Composable

1498. import androidx.compose.ui.Alignment

1499. import androidx.compose.ui.Modifier

1500. import androidx.compose.ui.draw.clip

1501. import androidx.compose.ui.graphics.Color

1502. import androidx.compose.ui.res.painterResource

1503. import androidx.compose.ui.text.font.FontWeight

1504. import androidx.compose.ui.unit.dp

1505. import androidx.compose.ui.unit.sp

1506. import androidx.navigation.NavHostController

1507. import com.example.iot\_ha.R

1508.

1509. @Composable

1510. fun RoomCard(

1511. roomId: Int,

1512. roomName: String,

1513. deviceCount: Int,

1514. navHostController: NavHostController,

1515. onDelete: (Int) -> Unit

1516. ) {

1517. Card(

1518. shape = RoundedCornerShape(16.dp),

1519. modifier = Modifier

1520. .fillMaxWidth()

1521. .padding(8.dp)

1522. .clickable { navHostController.navigate("room\_details/$roomId") },

1523. elevation = CardDefaults.cardElevation(6.dp),

1524. colors = CardDefaults.cardColors(containerColor = Color(0xFFE3F2FD))

1525. ) {

1526. Row(

1527. verticalAlignment = Alignment.CenterVertically,

1528. modifier = Modifier.padding(16.dp)

1529. ) {

1530. Image(

1531. painter = painterResource(id = R.drawable.mqtt\_logo),

1532. contentDescription = roomName,

1533. modifier = Modifier

1534. .size(64.dp)

1535. .clip(RoundedCornerShape(12.dp))

1536. )

1537. Spacer(modifier = Modifier.width(16.dp))

1538. Column(modifier = Modifier.weight(1f)) {

1539. Text(

1540. text = roomName,

1541. fontSize = 20.sp,

1542. fontWeight = FontWeight.Bold,

1543. color = Color.Black

1544. )

1545. }

1546. Spacer(modifier = Modifier.width(16.dp))

1547. Box(

1548. contentAlignment = Alignment.Center,

1549. modifier = Modifier

1550. .size(32.dp)

1551. .clip(RoundedCornerShape(16.dp))

1552. .background(MaterialTheme.colorScheme.primary)

1553. ) {

1554. Text(

1555. text = deviceCount.toString(),

1556. color = Color.White,

1557. fontSize = 14.sp,

1558. fontWeight = FontWeight.Bold

1559. )

1560. }

1561. Spacer(modifier = Modifier.width(8.dp))

1562. IconButton(onClick = { onDelete(roomId) }) {

1563. Icon(imageVector = Icons.Default.Delete, contentDescription = "Удалить комнату", tint = Color.Red)

1564. }

1565. }

1566. }

1567. }

BrightnessCard.kt

1568. package com.example.iot\_ha.ui.components.settings.led

1569.

1570. import androidx.compose.foundation.background

1571. import androidx.compose.foundation.border

1572. import androidx.compose.foundation.layout.Box

1573. import androidx.compose.foundation.layout.Column

1574. import androidx.compose.foundation.layout.Row

1575. import androidx.compose.foundation.layout.Spacer

1576. import androidx.compose.foundation.layout.fillMaxWidth

1577. import androidx.compose.foundation.layout.height

1578. import androidx.compose.foundation.layout.padding

1579. import androidx.compose.foundation.layout.size

1580. import androidx.compose.foundation.layout.width

1581. import androidx.compose.material3.Card

1582. import androidx.compose.material3.MaterialTheme

1583. import androidx.compose.material3.Slider

1584. import androidx.compose.material3.Text

1585. import androidx.compose.runtime.Composable

1586. import androidx.compose.ui.Alignment

1587. import androidx.compose.ui.Modifier

1588. import androidx.compose.ui.graphics.Color

1589. import androidx.compose.ui.unit.dp

1590.

1591. @Composable

1592. fun BrigtnessCard(

1593. title: String,

1594. color: Color,

1595. value: Float,

1596. onValueChange: (Float) -> Unit,

1597. onValueChangeFinished: () -> Unit

1598. ) {

1599. Card(

1600. modifier = Modifier

1601. .fillMaxWidth()

1602. .padding(8.dp),

1603. shape = MaterialTheme.shapes.medium

1604. ) {

1605. Column(

1606. modifier = Modifier

1607. .padding(16.dp)

1608. ) {

1609. Text(text = title, style = MaterialTheme.typography.bodyLarge)

1610.

1611. Spacer(modifier = Modifier.height(8.dp))

1612.

1613. Row(

1614. verticalAlignment = Alignment.CenterVertically,

1615. modifier = Modifier.fillMaxWidth()

1616. ) {

1617. Box(

1618. modifier = Modifier

1619. .size(40.dp)

1620. .background(color)

1621. .border(2.dp, Color.Black)

1622. )

1623.

1624. Spacer(modifier = Modifier.width(16.dp))

1625.

1626. Slider(

1627. value = value,

1628. onValueChange = onValueChange,

1629. onValueChangeFinished = onValueChangeFinished,

1630. valueRange = 0f..255f,

1631. modifier = Modifier.fillMaxWidth()

1632. )

1633. }

1634. }

1635. }

1636. }

ColorsCard.kt

1637. package com.example.iot\_ha.ui.components.settings.led

1638.

1639. import androidx.compose.foundation.background

1640. import androidx.compose.foundation.border

1641. import androidx.compose.foundation.layout.Box

1642. import androidx.compose.foundation.layout.Column

1643. import androidx.compose.foundation.layout.Row

1644. import androidx.compose.foundation.layout.Spacer

1645. import androidx.compose.foundation.layout.fillMaxWidth

1646. import androidx.compose.foundation.layout.height

1647. import androidx.compose.foundation.layout.padding

1648. import androidx.compose.foundation.layout.size

1649. import androidx.compose.foundation.layout.width

1650. import androidx.compose.material3.Card

1651. import androidx.compose.material3.MaterialTheme

1652. import androidx.compose.material3.Slider

1653. import androidx.compose.material3.Text

1654. import androidx.compose.runtime.Composable

1655. import androidx.compose.ui.Alignment

1656. import androidx.compose.ui.Modifier

1657. import androidx.compose.ui.graphics.Color

1658. import androidx.compose.ui.unit.dp

1659.

1660. @Composable

1661. fun ColorsCard(

1662. red: Float,

1663. green: Float,

1664. blue: Float,

1665. onRedChange: (Float) -> Unit,

1666. onGreenChange: (Float) -> Unit,

1667. onBlueChange: (Float) -> Unit,

1668. onValueChangeFinished: () -> Unit

1669. ) {

1670. Card(

1671. modifier = Modifier

1672. .fillMaxWidth()

1673. .padding(8.dp),

1674. shape = MaterialTheme.shapes.medium

1675. ) {

1676. Column(

1677. modifier = Modifier

1678. .padding(16.dp)

1679. ) {

1680. Text(text = "Colors Status", style = MaterialTheme.typography.bodyLarge)

1681.

1682. Spacer(modifier = Modifier.height(8.dp))

1683.

1684. listOf(

1685. Triple("Red", Color(red / 255f, 0f, 0f), onRedChange to red),

1686. Triple("Green", Color(0f, green / 255f, 0f), onGreenChange to green),

1687. Triple("Blue", Color(0f, 0f, blue / 255f), onBlueChange to blue)

1688. ).forEach { (label, color, setterWithValue) ->

1689. val (setter, value) = setterWithValue

1690.

1691. Row(

1692. verticalAlignment = Alignment.CenterVertically,

1693. modifier = Modifier.fillMaxWidth()

1694. ) {

1695. Box(

1696. modifier = Modifier

1697. .size(20.dp)

1698. .background(color)

1699. .border(1.dp, Color.Black)

1700. )

1701.

1702. Spacer(modifier = Modifier.width(8.dp))

1703.

1704. Column(modifier = Modifier.weight(1f)) {

1705. Text(text = label)

1706. Slider(

1707. value = value,

1708. onValueChange = setter,

1709. onValueChangeFinished = onValueChangeFinished,

1710. valueRange = 0f..255f,

1711. modifier = Modifier.fillMaxWidth()

1712. )

1713. }

1714. }

1715. Spacer(modifier = Modifier.height(8.dp))

1716. }

1717. }

1718. }

1719. }

AppNavHost.kt

1720. package com.example.iot\_ha.ui.navigation

1721.

1722. import com.example.iot\_ha.ui.screens.home.settings.LEDScreen

1723. import androidx.compose.runtime.Composable

1724. import androidx.compose.ui.platform.LocalContext

1725. import androidx.lifecycle.viewmodel.compose.viewModel

1726. import androidx.navigation.NavHostController

1727. import androidx.navigation.compose.NavHost

1728. import androidx.navigation.compose.composable

1729. import com.example.iot\_ha.data.local.RoomLocalDatabase

1730. import com.example.iot\_ha.ui.screens.AuthorizationScreen

1731. import com.example.iot\_ha.ui.screens.HomeScreen

1732. import com.example.iot\_ha.ui.screens.home.DeviceDetailScreen

1733. import com.example.iot\_ha.ui.screens.home.RoomDetailScreen

1734. import com.example.iot\_ha.ui.viewmodels.factory.DevicesViewModelFactory

1735. import com.example.iot\_ha.ui.viewmodels.factory.RoomsViewModelFactory

1736. import com.example.iot\_ha.ui.viewmodels.factory.SensorsViewModelFactory

1737. import com.example.iot\_ha.ui.viewmodels.shared.DevicesViewModel

1738. import com.example.iot\_ha.ui.viewmodels.shared.RoomsViewModel

1739. import com.example.iot\_ha.ui.viewmodels.shared.SensorsViewModel

1740.

1741. @Composable

1742. fun AppNavHost(

1743. navController: NavHostController,

1744. startDestination: String

1745. ) {

1746. val db = RoomLocalDatabase.getInstance(LocalContext.current)

1747.

1748. val sensorsViewModel: SensorsViewModel = viewModel(factory = SensorsViewModelFactory())

1749. val devicesViewModel: DevicesViewModel = viewModel(factory = DevicesViewModelFactory(db))

1750. val roomsViewModel: RoomsViewModel = viewModel(factory = RoomsViewModelFactory(db))

1751.

1752. NavHost(navController = navController, startDestination = startDestination) {

1753. composable(Routes.AUTH\_SCREEN) {

1754. AuthorizationScreen(

1755. navHostController = navController,

1756. sensorsViewModel = sensorsViewModel,

1757. devicesViewModel = devicesViewModel

1758. )

1759. }

1760.

1761. composable(Routes.HOME\_SCREEN) {

1762. HomeScreen(

1763. navHostController = navController,

1764. devicesViewModel = devicesViewModel,

1765. roomsViewModel = roomsViewModel

1766. )

1767. }

1768.

1769. composable(Routes.DEVICE\_DETAILS\_SCREEN) { backStackEntry ->

1770. DeviceDetailScreen(

1771. backStackEntry = backStackEntry,

1772. devicesViewModel = devicesViewModel,

1773. roomsViewModel = roomsViewModel

1774. )

1775. }

1776.

1777. composable(Routes.ROOM\_DETAILS\_SCREEN) { backStackEntry ->

1778. RoomDetailScreen(

1779. backStackEntry = backStackEntry,

1780. navHostController = navController,

1781. devicesViewModel = devicesViewModel,

1782. roomsViewModel = roomsViewModel

1783. )

1784. }

1785.

1786. composable(Routes.LED\_SETTINGS\_SCREEN) {

1787. LEDScreen()

1788. }

1789. }

1790. }

1791.

Routes.kt

1792. package com.example.iot\_ha.ui.navigation

1793.

1794. object Routes {

1795. const val AUTH\_SCREEN = "auth"

1796. const val HOME\_SCREEN = "home"

1797.

1798. const val DEVICE\_DETAILS\_SCREEN = "device\_details/{deviceId}"

1799. const val ROOM\_DETAILS\_SCREEN = "room\_details/{roomId}"

1800.

1801. const val LED\_SETTINGS\_SCREEN = "home/led"

1802. }

AuthorizationScreen.kt

1803. package com.example.iot\_ha.ui.screens

1804.

1805. import androidx.compose.foundation.Image

1806. import androidx.compose.foundation.layout.Column

1807. import androidx.compose.foundation.layout.Spacer

1808. import androidx.compose.foundation.layout.aspectRatio

1809. import androidx.compose.foundation.layout.fillMaxSize

1810. import androidx.compose.foundation.layout.fillMaxWidth

1811. import androidx.compose.foundation.layout.height

1812. import androidx.compose.foundation.layout.padding

1813. import androidx.compose.foundation.shape.RoundedCornerShape

1814. import androidx.compose.runtime.Composable

1815. import androidx.compose.runtime.getValue

1816. import androidx.compose.runtime.mutableStateOf

1817. import androidx.compose.runtime.remember

1818. import androidx.compose.runtime.setValue

1819. import androidx.compose.ui.Alignment

1820. import androidx.compose.ui.Modifier

1821. import androidx.compose.ui.draw.clip

1822. import androidx.compose.ui.platform.LocalContext

1823. import androidx.compose.ui.res.painterResource

1824. import androidx.compose.ui.unit.dp

1825. import androidx.lifecycle.viewmodel.compose.viewModel

1826. import androidx.navigation.NavHostController

1827. import com.example.iot\_ha.R

1828. import com.example.iot\_ha.data.local.RoomLocalDatabase

1829. import com.example.iot\_ha.data.mqtt.domain.MQTTMessageHandler

1830. import com.example.iot\_ha.ui.components.broker.BrokerInputForm

1831. import com.example.iot\_ha.ui.components.broker.BrokerList

1832. import com.example.iot\_ha.ui.viewmodels.AuthorizationViewModel

1833. import com.example.iot\_ha.ui.viewmodels.factory.AuthorizationViewModelFactory

1834. import com.example.iot\_ha.ui.viewmodels.shared.DevicesViewModel

1835. import com.example.iot\_ha.ui.viewmodels.shared.SensorsViewModel

1836.

1837. @Composable

1838. fun AuthorizationScreen(

1839. navHostController: NavHostController,

1840. sensorsViewModel: SensorsViewModel,

1841. devicesViewModel: DevicesViewModel

1842. ) {

1843. var serverUri by remember { mutableStateOf("") }

1844. var serverPort by remember { mutableStateOf("") }

1845. var user by remember { mutableStateOf("") }

1846. var password by remember { mutableStateOf("") }

1847.

1848. val db = RoomLocalDatabase.getInstance(LocalContext.current)

1849. val messageHandler = remember { MQTTMessageHandler(sensorsViewModel, devicesViewModel) }

1850. val authorizationViewModel: AuthorizationViewModel =

1851. viewModel(factory = AuthorizationViewModelFactory(db, messageHandler))

1852.

1853. val brokers = authorizationViewModel.brokers.value

1854.

1855. Column(

1856. modifier = Modifier

1857. .fillMaxSize()

1858. .padding(horizontal = 16.dp),

1859. horizontalAlignment = Alignment.CenterHorizontally

1860. ) {

1861. Spacer(modifier = Modifier.height(40.dp))

1862.

1863. Image(

1864. painter = painterResource(id = R.drawable.mqtt\_logo),

1865. contentDescription = "Broker Logo",

1866. modifier = Modifier

1867. .fillMaxWidth()

1868. .padding(horizontal = 16.dp)

1869. .aspectRatio(4f)

1870. .clip(RoundedCornerShape(12.dp)),

1871. )

1872.

1873. Spacer(modifier = Modifier.height(50.dp))

1874.

1875. BrokerInputForm(

1876. serverUri = serverUri,

1877. serverPort = serverPort,

1878. user = user,

1879. password = password,

1880. onServerUriChange = { serverUri = it },

1881. onServerPortChange = { serverPort = it },

1882. onUserChange = { user = it },

1883. onPasswordChange = { password = it },

1884. onAddBroker = {

1885. authorizationViewModel.addBroker(

1886. serverUri,

1887. serverPort.toIntOrNull() ?: 1883,

1888. user.takeIf { it.isNotBlank() },

1889. password.takeIf { it.isNotBlank() }

1890. )

1891. serverUri = ""

1892. serverPort = ""

1893. user = ""

1894. password = ""

1895. }

1896. )

1897.

1898. Spacer(modifier = Modifier.height(16.dp))

1899.

1900. BrokerList(

1901. brokers = brokers,

1902. onDelete = { authorizationViewModel.deleteBroker(it) },

1903. onLogin = { broker ->

1904. authorizationViewModel.handleLogin(broker) {

1905. navHostController.navigate("home")

1906. }

1907. }

1908. )

1909. }

1910. }

HomeScreen.kt

1911. package com.example.iot\_ha.ui.screens

1912.

1913. import androidx.compose.animation.AnimatedContent

1914. import androidx.compose.animation.ExperimentalAnimationApi

1915. import androidx.compose.animation.fadeIn

1916. import androidx.compose.animation.fadeOut

1917. import androidx.compose.animation.slideInHorizontally

1918. import androidx.compose.animation.slideOutHorizontally

1919. import androidx.compose.animation.togetherWith

1920. import androidx.compose.foundation.background

1921. import androidx.compose.foundation.layout.Arrangement

1922. import androidx.compose.foundation.layout.Column

1923. import androidx.compose.foundation.layout.PaddingValues

1924. import androidx.compose.foundation.layout.Spacer

1925. import androidx.compose.foundation.layout.fillMaxSize

1926. import androidx.compose.foundation.layout.fillMaxWidth

1927. import androidx.compose.foundation.layout.height

1928. import androidx.compose.foundation.layout.padding

1929. import androidx.compose.foundation.layout.statusBarsPadding

1930. import androidx.compose.foundation.lazy.LazyRow

1931. import androidx.compose.foundation.lazy.itemsIndexed

1932. import androidx.compose.runtime.Composable

1933. import androidx.compose.runtime.getValue

1934. import androidx.compose.runtime.mutableIntStateOf

1935. import androidx.compose.runtime.remember

1936. import androidx.compose.runtime.setValue

1937. import androidx.compose.ui.Modifier

1938. import androidx.compose.ui.graphics.Color

1939. import androidx.compose.ui.unit.dp

1940. import androidx.navigation.NavHostController

1941. import com.example.iot\_ha.ui.components.common.TabButton

1942. import com.example.iot\_ha.ui.screens.home.DevicesScreen

1943. import com.example.iot\_ha.ui.screens.home.RoomsScreen

1944. import com.example.iot\_ha.ui.screens.home.SettingsScreen

1945. import com.example.iot\_ha.ui.viewmodels.shared.DevicesViewModel

1946. import com.example.iot\_ha.ui.viewmodels.shared.RoomsViewModel

1947. import com.example.iot\_ha.utils.Constants

1948.

1949. @OptIn(ExperimentalAnimationApi::class)

1950. @Composable

1951. fun HomeScreen(

1952. navHostController: NavHostController,

1953. devicesViewModel: DevicesViewModel,

1954. roomsViewModel: RoomsViewModel

1955. ) {

1956. var selectedTab by remember { mutableIntStateOf(0) }

1957. var previousTab by remember { mutableIntStateOf(0) }

1958.

1959. Column(

1960. modifier = Modifier

1961. .fillMaxSize()

1962. .statusBarsPadding()

1963. .background(Color.White)

1964. ) {

1965. LazyRow(

1966. modifier = Modifier

1967. .fillMaxWidth()

1968. .padding(vertical = 12.dp, horizontal = 8.dp),

1969. contentPadding = PaddingValues(horizontal = 8.dp),

1970. horizontalArrangement = Arrangement.spacedBy(8.dp)

1971. ) {

1972. itemsIndexed(Constants.TABS\_LIST) { index, title ->

1973. TabButton(

1974. title = title,

1975. isSelected = selectedTab == index,

1976. onClick = {

1977. previousTab = selectedTab

1978. selectedTab = index

1979. }

1980. )

1981. }

1982. }

1983.

1984. Spacer(modifier = Modifier.height(8.dp))

1985.

1986. AnimatedContent(

1987. targetState = selectedTab,

1988. transitionSpec = {

1989. if (targetState > previousTab) {

1990. slideInHorizontally { width -> width } + fadeIn() togetherWith

1991. slideOutHorizontally { width -> -width } + fadeOut()

1992. } else {

1993. slideInHorizontally { width -> -width } + fadeIn() togetherWith

1994. slideOutHorizontally { width -> width } + fadeOut()

1995. }

1996. },

1997. label = "Tab Animation"

1998. ) { tab ->

1999. when (tab) {

2000. 0 -> DevicesScreen(

2001. navHostController = navHostController,

2002. devicesViewModel = devicesViewModel

2003. )

2004.

2005. 1 -> RoomsScreen(

2006. navHostController = navHostController,

2007. roomsViewModel = roomsViewModel

2008. )

2009.

2010. // 2 -> ScheduleScreen()

2011.

2012. 2 -> SettingsScreen(

2013. navHostController = navHostController

2014. )

2015.

2016. }

2017. }

2018. }

2019. }

DeviceDetailScreen.kt

2020. package com.example.iot\_ha.ui.screens.home

2021.

2022. import androidx.compose.foundation.layout.Column

2023. import androidx.compose.foundation.layout.Spacer

2024. import androidx.compose.foundation.layout.fillMaxSize

2025. import androidx.compose.foundation.layout.fillMaxWidth

2026. import androidx.compose.foundation.layout.height

2027. import androidx.compose.foundation.layout.padding

2028. import androidx.compose.material3.AlertDialog

2029. import androidx.compose.material3.DropdownMenuItem

2030. import androidx.compose.material3.ExperimentalMaterial3Api

2031. import androidx.compose.material3.ExposedDropdownMenuBox

2032. import androidx.compose.material3.ExposedDropdownMenuDefaults

2033. import androidx.compose.material3.MaterialTheme

2034. import androidx.compose.material3.OutlinedTextField

2035. import androidx.compose.material3.Surface

2036. import androidx.compose.material3.Text

2037. import androidx.compose.material3.TextButton

2038. import androidx.compose.runtime.Composable

2039. import androidx.compose.runtime.collectAsState

2040. import androidx.compose.runtime.getValue

2041. import androidx.compose.runtime.mutableStateOf

2042. import androidx.compose.runtime.remember

2043. import androidx.compose.runtime.setValue

2044. import androidx.compose.ui.Alignment

2045. import androidx.compose.ui.Modifier

2046. import androidx.compose.ui.unit.dp

2047. import androidx.navigation.NavBackStackEntry

2048. import com.example.iot\_ha.data.local.device.DeviceState

2049. import com.example.iot\_ha.ui.components.devices.DeviceDetails

2050. import com.example.iot\_ha.ui.components.devices.DeviceNotFoundMessage

2051. import com.example.iot\_ha.ui.components.devices.DeviceTitle

2052. import com.example.iot\_ha.ui.viewmodels.shared.DevicesViewModel

2053. import com.example.iot\_ha.ui.viewmodels.shared.RoomsViewModel

2054.

2055. @OptIn(ExperimentalMaterial3Api::class)

2056. @Composable

2057. fun DeviceDetailScreen(

2058. backStackEntry: NavBackStackEntry,

2059. devicesViewModel: DevicesViewModel,

2060. roomsViewModel: RoomsViewModel

2061. ) {

2062. val deviceId = backStackEntry.arguments?.getString("deviceId")?.toIntOrNull() ?: -1

2063.

2064. val devices by devicesViewModel.devices.collectAsState()

2065. val deviceState by DeviceState.devicesData.collectAsState()

2066. val deviceData = deviceState[deviceId]

2067.

2068. val rooms by roomsViewModel.rooms.collectAsState()

2069. val device = devices.find { it.id == deviceId }

2070. val currentRoom = rooms.find { it.id == device?.roomId }

2071. var selectedRoom by remember { mutableStateOf(currentRoom?.name ?: "Select room") }

2072. var expanded by remember { mutableStateOf(false) }

2073.

2074. var showEditDialog by remember { mutableStateOf(false) }

2075. var newName by remember { mutableStateOf(device?.friendlyName ?: "") }

2076.

2077. if (showEditDialog) {

2078. AlertDialog(

2079. onDismissRequest = { showEditDialog = false },

2080. title = { Text("Edit device name") },

2081. text = {

2082. OutlinedTextField(

2083. value = newName,

2084. onValueChange = { newName = it },

2085. label = { Text("New name") }

2086. )

2087. },

2088. confirmButton = {

2089. TextButton(

2090. onClick = {

2091. devicesViewModel.updateDeviceName(deviceId, newName)

2092. showEditDialog = false

2093. }

2094. ) {

2095. Text("Save")

2096. }

2097. },

2098. dismissButton = {

2099. TextButton(

2100. onClick = { showEditDialog = false }

2101. ) {

2102. Text("Cancel")

2103. }

2104. }

2105. )

2106. }

2107.

2108. Surface(

2109. modifier = Modifier

2110. .fillMaxSize()

2111. .padding(16.dp),

2112. color = MaterialTheme.colorScheme.background

2113. ) {

2114. Column(

2115. modifier = Modifier.fillMaxSize(),

2116. horizontalAlignment = Alignment.CenterHorizontally

2117. ) {

2118. DeviceTitle(

2119. friendlyName = device?.friendlyName?.uppercase() ?: "Unknown Device",

2120. onEditClick = { showEditDialog = true }

2121. )

2122.

2123. Spacer(modifier = Modifier.height(8.dp))

2124.

2125. ExposedDropdownMenuBox(

2126. expanded = expanded,

2127. onExpandedChange = { expanded = !expanded }

2128. ) {

2129. OutlinedTextField(

2130. value = selectedRoom,

2131. onValueChange = {},

2132. readOnly = true,

2133. modifier = Modifier

2134. .fillMaxWidth()

2135. .menuAnchor(),

2136. trailingIcon = {

2137. ExposedDropdownMenuDefaults.TrailingIcon(expanded = expanded)

2138. }

2139. )

2140. ExposedDropdownMenu(

2141. expanded = expanded,

2142. onDismissRequest = { expanded = false }

2143. ) {

2144. DropdownMenuItem(

2145. text = { Text("Select room") },

2146. onClick = {

2147. selectedRoom = "Select room"

2148. expanded = false

2149. devicesViewModel.assignRoomToDevice(deviceId, null)

2150. }

2151. )

2152. rooms.forEach { room ->

2153. DropdownMenuItem(

2154. text = { Text(room.name) },

2155. onClick = {

2156. selectedRoom = room.name

2157. expanded = false

2158. devicesViewModel.assignRoomToDevice(deviceId, room.id)

2159. }

2160. )

2161. }

2162. }

2163. }

2164.

2165. Spacer(modifier = Modifier.height(8.dp))

2166.

2167. if (deviceData != null) {

2168. DeviceDetails(deviceData)

2169. } else {

2170. DeviceNotFoundMessage()

2171. }

2172. }

2173. }

2174. }

DevicesScreen.kt

2175. package com.example.iot\_ha.ui.screens.home

2176.

2177. import androidx.compose.foundation.layout.Arrangement

2178. import androidx.compose.foundation.layout.Column

2179. import androidx.compose.foundation.layout.Spacer

2180. import androidx.compose.foundation.layout.fillMaxSize

2181. import androidx.compose.foundation.layout.height

2182. import androidx.compose.foundation.layout.padding

2183. import androidx.compose.foundation.rememberScrollState

2184. import androidx.compose.foundation.verticalScroll

2185. import androidx.compose.material3.Scaffold

2186. import androidx.compose.runtime.Composable

2187. import androidx.compose.runtime.collectAsState

2188. import androidx.compose.runtime.getValue

2189. import androidx.compose.ui.Alignment

2190. import androidx.compose.ui.Modifier

2191. import androidx.compose.ui.unit.dp

2192. import androidx.navigation.NavHostController

2193. import com.example.iot\_ha.R

2194. import com.example.iot\_ha.data.local.device.DeviceState

2195. import com.example.iot\_ha.ui.components.devices.DeviceCard

2196. import com.example.iot\_ha.ui.viewmodels.shared.DevicesViewModel

2197. import com.example.iot\_ha.utils.Constants

2198. import com.example.iot\_ha.utils.toBooleanState

2199.

2200. @Composable

2201. fun DevicesScreen(navHostController: NavHostController, devicesViewModel: DevicesViewModel) {

2202. val switchDevices by devicesViewModel.getDevicesByTypeFlow("switch").collectAsState()

2203. val devicesWithoutCommands by devicesViewModel.getDevicesWithoutCommandsFlow().collectAsState()

2204. val deviceState by DeviceState.devicesData.collectAsState()

2205.

2206. Scaffold { paddingValues ->

2207. Column(

2208. modifier = Modifier

2209. .fillMaxSize()

2210. .padding(paddingValues)

2211. .verticalScroll(rememberScrollState())

2212. .padding(16.dp)

2213. ) {

2214. if (devicesWithoutCommands.isEmpty() && switchDevices.isEmpty()) {

2215. Column(

2216. modifier = Modifier

2217. .fillMaxSize(),

2218. verticalArrangement = Arrangement.Center,

2219. horizontalAlignment = Alignment.CenterHorizontally

2220. ) {

2221. androidx.compose.material3.Text(

2222. text = "No connected devices",

2223. style = androidx.compose.material3.MaterialTheme.typography.titleMedium

2224. )

2225. }

2226. } else {

2227. devicesWithoutCommands.forEach { device ->

2228. DeviceCard(

2229. deviceId = device.id,

2230. imageRes = R.drawable.mqtt\_logo,

2231. name = device.friendlyName,

2232. navController = navHostController,

2233. onToggle = { state ->

2234. devicesViewModel.onToggle(device.id, state)

2235. }

2236. )

2237. Spacer(modifier = Modifier.height(8.dp))

2238. }

2239.

2240. switchDevices.forEach { device ->

2241. val deviceData = deviceState[device.id]

2242.

2243. DeviceCard(

2244. deviceId = device.id,

2245. imageRes = R.drawable.mqtt\_logo,

2246. name = device.friendlyName,

2247. type = Constants.SWITCH\_TYPE,

2248. value = (deviceData?.get("state") as? String)?.toBooleanState() ?: false,

2249. navController = navHostController,

2250. onToggle = { state ->

2251. devicesViewModel.onToggle(device.id, state)

2252. }

2253. )

2254. Spacer(modifier = Modifier.height(8.dp))

2255. }

2256. }

2257. }

2258. }

2259. }

RoomDetailScreen.kt

2260. package com.example.iot\_ha.ui.screens.home

2261.

2262. import androidx.compose.foundation.layout.Column

2263. import androidx.compose.foundation.layout.Spacer

2264. import androidx.compose.foundation.layout.fillMaxSize

2265. import androidx.compose.foundation.layout.height

2266. import androidx.compose.foundation.layout.padding

2267. import androidx.compose.foundation.rememberScrollState

2268. import androidx.compose.foundation.verticalScroll

2269. import androidx.compose.material3.Scaffold

2270. import androidx.compose.material3.Text

2271. import androidx.compose.runtime.Composable

2272. import androidx.compose.runtime.collectAsState

2273. import androidx.compose.runtime.getValue

2274. import androidx.compose.ui.Modifier

2275. import androidx.compose.ui.unit.dp

2276. import androidx.navigation.NavBackStackEntry

2277. import androidx.navigation.NavHostController

2278. import com.example.iot\_ha.R

2279. import com.example.iot\_ha.data.local.device.DeviceState

2280. import com.example.iot\_ha.ui.components.devices.DeviceCard

2281. import com.example.iot\_ha.ui.viewmodels.shared.DevicesViewModel

2282. import com.example.iot\_ha.ui.viewmodels.shared.RoomsViewModel

2283. import com.example.iot\_ha.utils.Constants

2284. import com.example.iot\_ha.utils.toBooleanState

2285.

2286. @Composable

2287. fun RoomDetailScreen(

2288. backStackEntry: NavBackStackEntry,

2289. roomsViewModel: RoomsViewModel,

2290. devicesViewModel: DevicesViewModel,

2291. navHostController: NavHostController

2292. ) {

2293. val roomId = backStackEntry.arguments?.getString("roomId")?.toIntOrNull() ?: -1

2294.

2295. val roomDevices by devicesViewModel.getDevicesByRoomIdFlow(roomId).collectAsState()

2296. val switchDevices by devicesViewModel.getDevicesByTypeFlow("switch").collectAsState()

2297.

2298. val roomSwitchDevices = switchDevices.filter { it -> it.id in roomDevices.map { it.id } }

2299.

2300. val deviceState by DeviceState.devicesData.collectAsState()

2301.

2302. Scaffold { paddingValues ->

2303. Column(

2304. modifier = Modifier

2305. .fillMaxSize()

2306. .padding(paddingValues)

2307. .verticalScroll(rememberScrollState())

2308. .padding(16.dp)

2309. ) {

2310. roomSwitchDevices.forEach { device ->

2311. val deviceData = deviceState[device.id]

2312.

2313. DeviceCard(

2314. deviceId = device.id,

2315. imageRes = R.drawable.mqtt\_logo,

2316. name = device.friendlyName,

2317. type = Constants.SWITCH\_TYPE,

2318. value = (deviceData?.get("state") as? String)?.toBooleanState() ?: false,

2319. navController = navHostController,

2320. onToggle = { state ->

2321. devicesViewModel.onToggle(device.id, state)

2322. }

2323. )

2324. Spacer(modifier = Modifier.height(8.dp))

2325. }

2326. }

2327. }

2328. }

2329.

2330.

RoomsScreen.kt

2331. package com.example.iot\_ha.ui.screens.home

2332.

2333. import android.util.Log

2334. import androidx.compose.foundation.layout.Box

2335. import androidx.compose.foundation.layout.Column

2336. import androidx.compose.foundation.layout.PaddingValues

2337. import androidx.compose.foundation.layout.fillMaxSize

2338. import androidx.compose.foundation.layout.padding

2339. import androidx.compose.foundation.lazy.LazyColumn

2340. import androidx.compose.foundation.lazy.items

2341. import androidx.compose.material.icons.Icons

2342. import androidx.compose.material.icons.filled.Add

2343. import androidx.compose.material3.FloatingActionButton

2344. import androidx.compose.material3.Icon

2345. import androidx.compose.material3.MaterialTheme

2346. import androidx.compose.runtime.Composable

2347. import androidx.compose.runtime.LaunchedEffect

2348. import androidx.compose.runtime.collectAsState

2349. import androidx.compose.runtime.getValue

2350. import androidx.compose.runtime.mutableIntStateOf

2351. import androidx.compose.runtime.mutableStateOf

2352. import androidx.compose.runtime.remember

2353. import androidx.compose.runtime.setValue

2354. import androidx.compose.ui.Alignment

2355. import androidx.compose.ui.Modifier

2356. import androidx.compose.ui.unit.dp

2357. import androidx.navigation.NavHostController

2358. import com.example.iot\_ha.ui.components.rooms.AddRoomDialog

2359. import com.example.iot\_ha.ui.components.rooms.RoomCard

2360. import com.example.iot\_ha.ui.viewmodels.shared.RoomsViewModel

2361.

2362. @Composable

2363. fun RoomsScreen(

2364. navHostController: NavHostController,

2365. roomsViewModel: RoomsViewModel

2366. ) {

2367. var showDialog by remember { mutableStateOf(false) }

2368. val rooms by roomsViewModel.rooms.collectAsState()

2369.

2370. Box(modifier = Modifier.fillMaxSize()) {

2371. Column(modifier = Modifier.fillMaxSize()) {

2372. LazyColumn(

2373. modifier = Modifier.weight(1f),

2374. contentPadding = PaddingValues(16.dp)

2375. ) {

2376. items(rooms) { room ->

2377. var deviceCount by remember { mutableIntStateOf(0) }

2378.

2379. LaunchedEffect(room.id) {

2380. roomsViewModel.getDeviceCount(room.id) { count ->

2381. deviceCount = count

2382. }

2383. }

2384.

2385. RoomCard(

2386. roomId = room.id.toInt(),

2387. roomName = room.name,

2388. deviceCount = deviceCount,

2389. navHostController = navHostController,

2390. onDelete = { roomId ->

2391. roomsViewModel.deleteRoom(roomId)

2392. }

2393. )

2394. }

2395.

2396. }

2397. }

2398.

2399. FloatingActionButton(

2400. onClick = { showDialog = true },

2401. containerColor = MaterialTheme.colorScheme.primary,

2402. modifier = Modifier

2403. .align(Alignment.BottomEnd)

2404. .padding(16.dp)

2405. ) {

2406. Icon(imageVector = Icons.Default.Add, contentDescription = "Добавить комнату")

2407. }

2408. }

2409.

2410. if (showDialog) {

2411. AddRoomDialog(

2412. onDismiss = { showDialog = false },

2413. onConfirm = { roomName ->

2414. roomsViewModel.addRoom(roomName)

2415. showDialog = false

2416. Log.i("Rooms", "Комната добавлена: $roomName")

2417. }

2418. )

2419. }

2420. }

2421.

ScheduleScreen.kt

2422. package com.example.iot\_ha.ui.screens.home

2423.

2424. import androidx.compose.material3.Text

2425. import androidx.compose.runtime.Composable

2426.

2427. @Composable

2428. fun ScheduleScreen() {

2429. Text("Schedule Screen")

2430. }

SettingsScreen.kt

2431. package com.example.iot\_ha.ui.screens.home

2432.

2433. import androidx.compose.foundation.layout.Arrangement

2434. import androidx.compose.foundation.layout.Column

2435. import androidx.compose.foundation.layout.Row

2436. import androidx.compose.foundation.layout.Spacer

2437. import androidx.compose.foundation.layout.fillMaxSize

2438. import androidx.compose.foundation.layout.fillMaxWidth

2439. import androidx.compose.foundation.layout.height

2440. import androidx.compose.foundation.layout.padding

2441. import androidx.compose.foundation.shape.RoundedCornerShape

2442. import androidx.compose.material3.Card

2443. import androidx.compose.material3.CardDefaults

2444. import androidx.compose.material3.MaterialTheme

2445. import androidx.compose.material3.Switch

2446. import androidx.compose.material3.SwitchDefaults

2447. import androidx.compose.material3.Text

2448. import androidx.compose.runtime.Composable

2449. import androidx.compose.runtime.LaunchedEffect

2450. import androidx.compose.runtime.collectAsState

2451. import androidx.compose.runtime.getValue

2452. import androidx.compose.runtime.mutableIntStateOf

2453. import androidx.compose.runtime.remember

2454. import androidx.compose.runtime.setValue

2455. import androidx.compose.ui.Alignment

2456. import androidx.compose.ui.Modifier

2457. import androidx.compose.ui.graphics.Color

2458. import androidx.compose.ui.unit.dp

2459. import androidx.navigation.NavController

2460. import com.example.iot\_ha.data.local.broker.discovery.DiscoveryState

2461. import com.example.iot\_ha.ui.navigation.Routes

2462. import kotlinx.coroutines.delay

2463.

2464. @Composable

2465. fun SettingsScreen(navHostController: NavController) {

2466. val discoveryState by DiscoveryState.isDiscoveryActive.collectAsState()

2467.

2468. var remainingTime by remember { mutableIntStateOf(0) }

2469.

2470. fun onSwitchChanged(enabled: Boolean) { // rename to onDiscoverySwitchChanged

2471. if (enabled) {

2472. DiscoveryState.startDiscovery()

2473. } else {

2474. DiscoveryState.stopDiscovery()

2475. }

2476. }

2477.

2478. LaunchedEffect(discoveryState) {

2479. if (discoveryState) {

2480. remainingTime = 255

2481. while (remainingTime > 0) {

2482. delay(1000)

2483. remainingTime -= 1

2484. }

2485. } else {

2486. remainingTime = 0

2487. }

2488. }

2489.

2490. Column(

2491. modifier = Modifier

2492. .fillMaxSize()

2493. .padding(16.dp),

2494. verticalArrangement = Arrangement.Top,

2495. horizontalAlignment = Alignment.CenterHorizontally

2496. ) {

2497.

2498. Spacer(modifier = Modifier.height(16.dp))

2499.

2500. Card(

2501. shape = RoundedCornerShape(16.dp),

2502. colors = CardDefaults.cardColors(containerColor = Color(0xFFb697c4)),

2503. elevation = CardDefaults.cardElevation(6.dp),

2504. modifier = Modifier.fillMaxWidth()

2505. ) {

2506. Column(

2507. modifier = Modifier

2508. .fillMaxWidth()

2509. .padding(16.dp)

2510. ) {

2511. Row(

2512. verticalAlignment = Alignment.CenterVertically

2513. ) {

2514. Text(

2515. text = "Device discovery mode",

2516. modifier = Modifier.weight(1f),

2517. style = MaterialTheme.typography.bodyLarge

2518. )

2519. Switch(

2520. checked = discoveryState,

2521. onCheckedChange = { onSwitchChanged(it) },

2522. colors = SwitchDefaults.colors(

2523. checkedThumbColor = Color.White,

2524. checkedTrackColor = Color(0xFF4CAF50),

2525. uncheckedThumbColor = Color.White,

2526. uncheckedTrackColor = Color(0xFFF44336)

2527. )

2528. )

2529. }

2530.

2531. if (discoveryState) {

2532. Text(

2533. text = "Time left: $remainingTime s",

2534. style = MaterialTheme.typography.bodyMedium,

2535. color = Color.Black

2536. )

2537. }

2538. }

2539. }

2540.

2541. Spacer(modifier = Modifier.height(16.dp))

2542.

2543. Card(

2544. shape = RoundedCornerShape(16.dp),

2545. colors = CardDefaults.cardColors(containerColor = Color(0xFFb697c4)),

2546. elevation = CardDefaults.cardElevation(6.dp),

2547. modifier = Modifier.fillMaxWidth()

2548. ) {

2549. Column(

2550. modifier = Modifier

2551. .fillMaxWidth()

2552. .padding(16.dp)

2553. ) {

2554. Row(

2555. verticalAlignment = Alignment.CenterVertically

2556. ) {

2557. Text(

2558. text = "LED Settings",

2559. modifier = Modifier.weight(1f),

2560. style = MaterialTheme.typography.bodyLarge

2561. )

2562. }

2563.

2564. Text(

2565. text = "Adjust LED brightness and color",

2566. style = MaterialTheme.typography.bodyMedium,

2567. color = Color.Black,

2568. modifier = Modifier.padding(top = 8.dp)

2569. )

2570.

2571. Spacer(modifier = Modifier.height(8.dp))

2572.

2573. androidx.compose.material3.Button(

2574. onClick = { navHostController.navigate(Routes.LED\_SETTINGS\_SCREEN) },

2575. modifier = Modifier.fillMaxWidth()

2576. ) {

2577. Text(text = "Go to LED Settings")

2578. }

2579.

2580. }

2581. }

2582. }

2583. }

LEDScreen.kt

2584. package com.example.iot\_ha.ui.screens.home.settings

2585.

2586. import androidx.compose.foundation.layout.Column

2587. import androidx.compose.foundation.layout.Spacer

2588. import androidx.compose.foundation.layout.fillMaxSize

2589. import androidx.compose.foundation.layout.fillMaxWidth

2590. import androidx.compose.foundation.layout.height

2591. import androidx.compose.foundation.layout.padding

2592. import androidx.compose.foundation.rememberScrollState

2593. import androidx.compose.foundation.verticalScroll

2594. import androidx.compose.material3.\*

2595. import androidx.compose.runtime.\*

2596. import androidx.compose.ui.Alignment

2597. import androidx.compose.ui.Modifier

2598. import androidx.compose.ui.graphics.Color

2599. import androidx.compose.ui.unit.dp

2600. import androidx.lifecycle.viewmodel.compose.viewModel

2601. import com.example.iot\_ha.data.local.led.LEDState

2602. import com.example.iot\_ha.ui.components.settings.led.BrigtnessCard

2603. import com.example.iot\_ha.ui.components.settings.led.ColorsCard

2604. import com.example.iot\_ha.ui.viewmodels.LEDScreenViewModel

2605. import com.example.iot\_ha.ui.viewmodels.factory.LEDScreenViewModelFactory

2606.

2607. @OptIn(ExperimentalMaterial3Api::class)

2608. @Composable

2609. fun LEDScreen() {

2610. val brightness by LEDState.brightness.collectAsState()

2611. val red by LEDState.red.collectAsState()

2612. val green by LEDState.green.collectAsState()

2613. val blue by LEDState.blue.collectAsState()

2614.

2615. val ledScreenViewModel: LEDScreenViewModel = viewModel(factory = LEDScreenViewModelFactory())

2616.

2617. var selectedMode by remember { mutableStateOf("AUTO") }

2618. var expanded by remember { mutableStateOf(false) }

2619.

2620. Column(

2621. modifier = Modifier

2622. .fillMaxSize()

2623. .padding(16.dp)

2624. .verticalScroll(rememberScrollState()),

2625. horizontalAlignment = Alignment.CenterHorizontally

2626. ) {

2627. Text(text = "Edit LED Settings", style = MaterialTheme.typography.headlineSmall)

2628.

2629. Spacer(modifier = Modifier.height(16.dp))

2630.

2631. ExposedDropdownMenuBox(

2632. expanded = expanded,

2633. onExpandedChange = { expanded = !expanded }

2634. ) {

2635. OutlinedTextField(

2636. value = selectedMode,

2637. onValueChange = {},

2638. readOnly = true,

2639. modifier = Modifier

2640. .fillMaxWidth()

2641. .menuAnchor(),

2642. trailingIcon = {

2643. ExposedDropdownMenuDefaults.TrailingIcon(expanded = expanded)

2644. }

2645. )

2646.

2647. ExposedDropdownMenu(

2648. expanded = expanded,

2649. onDismissRequest = { expanded = false }

2650. ) {

2651. DropdownMenuItem(

2652. text = { Text("AUTO") },

2653. onClick = {

2654. ledScreenViewModel.setAUTOMode()

2655. selectedMode = "AUTO"

2656. expanded = false

2657. }

2658. )

2659. DropdownMenuItem(

2660. text = { Text("MANUAL") },

2661. onClick = {

2662. selectedMode = "MANUAL"

2663. expanded = false

2664. }

2665. )

2666. }

2667. }

2668.

2669. Spacer(modifier = Modifier.height(16.dp))

2670.

2671. if (selectedMode == "MANUAL") {

2672. BrigtnessCard(

2673. title = "Brightness",

2674. color = Color(brightness / 255f, brightness / 255f, brightness / 255f),

2675. value = brightness,

2676. onValueChange = { LEDState.setBrightness(it) },

2677. onValueChangeFinished = { ledScreenViewModel.sendLEDStatus() }

2678. )

2679.

2680. Spacer(modifier = Modifier.height(16.dp))

2681.

2682. ColorsCard(

2683. red = red,

2684. green = green,

2685. blue = blue,

2686. onRedChange = { LEDState.setRed(it) },

2687. onGreenChange = { LEDState.setGreen(it) },

2688. onBlueChange = { LEDState.setBlue(it) },

2689. onValueChangeFinished = { ledScreenViewModel.sendLEDStatus() }

2690. )

2691. } else {

2692. Text(text = "LED settings are in AUTO mode. No adjustments available.", style = MaterialTheme.typography.bodyMedium)

2693. }

2694. }

2695. }

Color.kt

2696. package com.example.iot\_ha.ui.theme

2697.

2698. import androidx.compose.ui.graphics.Color

2699.

2700. val Purple80 = Color(0xFFD0BCFF)

2701. val PurpleGrey80 = Color(0xFFCCC2DC)

2702. val Pink80 = Color(0xFFEFB8C8)

2703.

2704. val Purple40 = Color(0xFF6650a4)

2705. val PurpleGrey40 = Color(0xFF625b71)

2706. val Pink40 = Color(0xFF7D5260)

Theme.kt

2707. package com.example.iot\_ha.ui.theme

2708.

2709. import android.os.Build

2710. import androidx.compose.foundation.isSystemInDarkTheme

2711. import androidx.compose.material3.MaterialTheme

2712. import androidx.compose.material3.darkColorScheme

2713. import androidx.compose.material3.dynamicDarkColorScheme

2714. import androidx.compose.material3.dynamicLightColorScheme

2715. import androidx.compose.material3.lightColorScheme

2716. import androidx.compose.runtime.Composable

2717. import androidx.compose.ui.platform.LocalContext

2718.

2719. private val DarkColorScheme = darkColorScheme(

2720. primary = Purple80,

2721. secondary = PurpleGrey80,

2722. tertiary = Pink80

2723. )

2724.

2725. private val LightColorScheme = lightColorScheme(

2726. primary = Purple40,

2727. secondary = PurpleGrey40,

2728. tertiary = Pink40

2729.

2730. /\* Other default colors to override

2731. background = Color(0xFFFFFBFE),

2732. surface = Color(0xFFFFFBFE),

2733. onPrimary = Color.White,

2734. onSecondary = Color.White,

2735. onTertiary = Color.White,

2736. onBackground = Color(0xFF1C1B1F),

2737. onSurface = Color(0xFF1C1B1F),

2738. \*/

2739. )

2740.

2741. @Composable

2742. fun IothaTheme(

2743. darkTheme: Boolean = isSystemInDarkTheme(),

2744. // Dynamic color is available on Android 12+

2745. dynamicColor: Boolean = true,

2746. content: @Composable () -> Unit

2747. ) {

2748. val colorScheme = when {

2749. dynamicColor && Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.S -> {

2750. val context = LocalContext.current

2751. if (darkTheme) dynamicDarkColorScheme(context) else dynamicLightColorScheme(context)

2752. }

2753.

2754. darkTheme -> DarkColorScheme

2755. else -> LightColorScheme

2756. }

2757.

2758. MaterialTheme(

2759. colorScheme = colorScheme,

2760. typography = Typography,

2761. content = content

2762. )

2763. }

Type.kt

2764. package com.example.iot\_ha.ui.theme

2765.

2766. import androidx.compose.material3.Typography

2767. import androidx.compose.ui.text.TextStyle

2768. import androidx.compose.ui.text.font.FontFamily

2769. import androidx.compose.ui.text.font.FontWeight

2770. import androidx.compose.ui.unit.sp

2771.

2772. // Set of Material typography styles to start with

2773. val Typography = Typography(

2774. bodyLarge = TextStyle(

2775. fontFamily = FontFamily.Default,

2776. fontWeight = FontWeight.Normal,

2777. fontSize = 16.sp,

2778. lineHeight = 24.sp,

2779. letterSpacing = 0.5.sp

2780. )

2781. /\* Other default text styles to override

2782. titleLarge = TextStyle(

2783. fontFamily = FontFamily.Default,

2784. fontWeight = FontWeight.Normal,

2785. fontSize = 22.sp,

2786. lineHeight = 28.sp,

2787. letterSpacing = 0.sp

2788. ),

2789. labelSmall = TextStyle(

2790. fontFamily = FontFamily.Default,

2791. fontWeight = FontWeight.Medium,

2792. fontSize = 11.sp,

2793. lineHeight = 16.sp,

2794. letterSpacing = 0.5.sp

2795. )

2796. \*/

2797. )

AuthorizationViewModel.kt

2798. package com.example.iot\_ha.ui.viewmodels

2799.

2800. import androidx.compose.runtime.State

2801. import androidx.compose.runtime.mutableStateOf

2802. import androidx.lifecycle.ViewModel

2803. import androidx.lifecycle.viewModelScope

2804. import com.example.iot\_ha.data.local.RoomLocalDatabase

2805. import com.example.iot\_ha.data.local.broker.Broker

2806. import com.example.iot\_ha.data.local.broker.BrokerState

2807. import com.example.iot\_ha.data.mqtt.MQTTClient

2808. import com.example.iot\_ha.data.mqtt.domain.MQTTMessageHandler

2809. import com.example.iot\_ha.data.mqtt.util.Topics

2810. import kotlinx.coroutines.Dispatchers

2811. import kotlinx.coroutines.delay

2812. import kotlinx.coroutines.launch

2813.

2814. class AuthorizationViewModel(

2815. private val db: RoomLocalDatabase,

2816. private val messageHandler: MQTTMessageHandler

2817. ) : ViewModel() {

2818.

2819. private val brokerDao = db.brokerDAO()

2820.

2821. private var \_brokers = mutableStateOf<List<Broker>>(emptyList())

2822. val brokers: State<List<Broker>> = \_brokers

2823.

2824. init {

2825. loadBrokers()

2826. }

2827.

2828. private fun loadBrokers() {

2829. viewModelScope.launch {

2830. \_brokers.value = brokerDao.getAllBrokers()

2831. }

2832. }

2833.

2834. fun addBroker(serverUri: String, serverPort: Int, user: String?, password: String?) {

2835. if (serverUri.isBlank()) return

2836. if (serverPort !in 1..65535) return

2837.

2838. viewModelScope.launch {

2839. val broker = Broker(

2840. serverUri = serverUri,

2841. serverPort = serverPort,

2842. user = user,

2843. password = password

2844. )

2845. brokerDao.insert(broker)

2846. loadBrokers()

2847. }

2848. }

2849.

2850.

2851. fun deleteBroker(broker: Broker) {

2852. viewModelScope.launch {

2853. MQTTClient.getInstance().disconnect()

2854. brokerDao.deleteBroker(broker)

2855. loadBrokers()

2856. }

2857. }

2858.

2859. fun handleLogin(broker: Broker, onSuccess: () -> Unit) {

2860. BrokerState.setBrokerId(broker.id)

2861.

2862. val mqttClient = MQTTClient.reinitialize(broker, messageHandler)

2863. val isSuccess = mqttClient.connect()

2864. if (isSuccess) {

2865. onSuccess()

2866.

2867. viewModelScope.launch(Dispatchers.IO) {

2868. mqttClient.subscribe(Topics.SUBSCRIBE\_DEVICE\_LIST\_TOPIC)

2869. delay(500)

2870. mqttClient.subscribe(Topics.SUBSCRIBE\_DEVICE\_COMMANDS\_TOPIC)

2871. mqttClient.subscribe(Topics.SUBSCRIBE\_DEVICE\_STATE\_TOPIC)

2872.

2873. mqttClient.subscribe(Topics.SUBSCRIBE\_LED\_STATE\_TOPIC)

2874. }

2875. }

2876. }

2877. }

HomeViewModel.kt

2878. package com.example.iot\_ha.ui.viewmodels

2879.

2880. import androidx.lifecycle.ViewModel

2881. import com.example.iot\_ha.data.local.RoomLocalDatabase

2882.

2883.

2884. class HomeViewModel(db: RoomLocalDatabase) : ViewModel()

LEDScreenViewModel.kt

2885. package com.example.iot\_ha.ui.viewmodels

2886.

2887. import androidx.lifecycle.ViewModel

2888. import androidx.lifecycle.viewModelScope

2889. import com.example.iot\_ha.data.local.led.LEDState

2890. import com.example.iot\_ha.data.local.led.LEDStatus

2891. import com.example.iot\_ha.data.mqtt.MQTTClient

2892. import com.example.iot\_ha.data.mqtt.util.Topics

2893. import kotlinx.coroutines.flow.first

2894. import kotlinx.coroutines.launch

2895.

2896. class LEDScreenViewModel : ViewModel() {

2897.

2898. // Отправить статус LED

2899. fun sendLEDStatus() {

2900. viewModelScope.launch {

2901. val brightness = LEDState.brightness.first().toInt()

2902. val red = LEDState.red.first().toInt()

2903. val green = LEDState.green.first().toInt()

2904. val blue = LEDState.blue.first().toInt()

2905.

2906. val ledStatus = LEDStatus(

2907. brightness = brightness,

2908. red = red,

2909. green = green,

2910. blue = blue

2911. )

2912.

2913. MQTTClient.publish(Topics.LED\_SET\_STATE\_TOPIC, ledStatus.toJson())

2914. }

2915. }

2916.

2917. fun setAUTOMode() {

2918. viewModelScope.launch {

2919. // Статус AUTO

2920. val ledAutoStatus = """

2921. {

2922. "state": "OFF",

2923. "brightness": 255,

2924. "color": {

2925. "r": 255,

2926. "g": 255,

2927. "b": 255

2928. },

2929. "color\_mode": "rgb",

2930. "mode": "auto"

2931. }

2932. """.trimIndent()

2933.

2934. MQTTClient.publish(Topics.LED\_SET\_STATE\_TOPIC, ledAutoStatus)

2935. }

2936. }

2937. }

AuthorizationViewModelFactory.kt

2938. package com.example.iot\_ha.ui.viewmodels.factory

2939.

2940. import androidx.lifecycle.ViewModel

2941. import androidx.lifecycle.ViewModelProvider

2942. import com.example.iot\_ha.data.local.RoomLocalDatabase

2943. import com.example.iot\_ha.data.mqtt.domain.MQTTMessageHandler

2944. import com.example.iot\_ha.ui.viewmodels.AuthorizationViewModel

2945.

2946. class AuthorizationViewModelFactory(

2947. private val db: RoomLocalDatabase,

2948. private val messageHandler: MQTTMessageHandler,

2949. ) : ViewModelProvider.Factory {

2950.

2951. @Suppress("UNCHECKED\_CAST")

2952. override fun <T : ViewModel> create(modelClass: Class<T>): T {

2953. if (modelClass.isAssignableFrom(AuthorizationViewModel::class.java)) {

2954. return AuthorizationViewModel(db, messageHandler) as T

2955. }

2956. throw IllegalArgumentException("Unknown ViewModel class")

2957. }

2958. }

DevicesViewModelFactory.kt

2959. package com.example.iot\_ha.ui.viewmodels.factory

2960.

2961. import androidx.lifecycle.ViewModel

2962. import androidx.lifecycle.ViewModelProvider

2963. import com.example.iot\_ha.data.local.RoomLocalDatabase

2964. import com.example.iot\_ha.ui.viewmodels.shared.DevicesViewModel

2965.

2966. class DevicesViewModelFactory(private val db: RoomLocalDatabase) : ViewModelProvider.Factory {

2967. override fun <T : ViewModel> create(modelClass: Class<T>): T {

2968. if (modelClass.isAssignableFrom(DevicesViewModel::class.java)) {

2969. @Suppress("UNCHECKED\_CAST")

2970. return DevicesViewModel(db) as T

2971. }

2972. throw IllegalArgumentException("Unknown ViewModel class")

2973. }

2974. }

HomeViewModelFactory.kt

2975. package com.example.iot\_ha.ui.viewmodels.factory

2976.

2977. import androidx.lifecycle.ViewModel

2978. import androidx.lifecycle.ViewModelProvider

2979. import com.example.iot\_ha.data.local.RoomLocalDatabase

2980. import com.example.iot\_ha.ui.viewmodels.HomeViewModel

2981.

2982. class HomeViewModelFactory(private val db: RoomLocalDatabase) : ViewModelProvider.Factory {

2983. override fun <T : ViewModel> create(modelClass: Class<T>): T {

2984. if (modelClass.isAssignableFrom(HomeViewModel::class.java)) {

2985. @Suppress("UNCHECKED\_CAST")

2986. return HomeViewModel(db) as T

2987. }

2988. throw IllegalArgumentException("Unknown ViewModel class")

2989. }

2990. }

LEDScreenViewModelFactory.kt

2991. package com.example.iot\_ha.ui.viewmodels.factory

2992.

2993. import androidx.lifecycle.ViewModel

2994. import androidx.lifecycle.ViewModelProvider

2995. import com.example.iot\_ha.ui.viewmodels.LEDScreenViewModel

2996.

2997. class LEDScreenViewModelFactory : ViewModelProvider.Factory {

2998. override fun <T : ViewModel> create(modelClass: Class<T>): T {

2999. if (modelClass.isAssignableFrom(LEDScreenViewModel::class.java)) {

3000. @Suppress("UNCHECKED\_CAST")

3001. return LEDScreenViewModel() as T

3002. }

3003. throw IllegalArgumentException("Unknown ViewModel class")

3004. }

3005. }

RoomsViewModelFactory.kt

3006. package com.example.iot\_ha.ui.viewmodels.factory

3007.

3008. import androidx.lifecycle.ViewModel

3009. import androidx.lifecycle.ViewModelProvider

3010. import com.example.iot\_ha.data.local.RoomLocalDatabase

3011. import com.example.iot\_ha.ui.viewmodels.shared.RoomsViewModel

3012.

3013. class RoomsViewModelFactory(private val db: RoomLocalDatabase) : ViewModelProvider.Factory {

3014. override fun <T : ViewModel> create(modelClass: Class<T>): T {

3015. if (modelClass.isAssignableFrom(RoomsViewModel::class.java)) {

3016. @Suppress("UNCHECKED\_CAST")

3017. return RoomsViewModel(db) as T

3018. }

3019. throw IllegalArgumentException("Unknown ViewModel class")

3020. }

3021. }

SensorsViewModelFactory.kt

3022. package com.example.iot\_ha.ui.viewmodels.factory

3023.

3024. import androidx.lifecycle.ViewModel

3025. import androidx.lifecycle.ViewModelProvider

3026. import com.example.iot\_ha.ui.viewmodels.shared.SensorsViewModel

3027.

3028. class SensorsViewModelFactory : ViewModelProvider.Factory {

3029. @Suppress("UNCHECKED\_CAST")

3030. override fun <T : ViewModel> create(modelClass: Class<T>): T {

3031. if (modelClass.isAssignableFrom(SensorsViewModel::class.java)) {

3032. return SensorsViewModel() as T

3033. }

3034. throw IllegalArgumentException("Unknown ViewModel class")

3035. }

3036. }

DevicesViewModel.kt

3037. package com.example.iot\_ha.ui.viewmodels.shared

3038.

3039. import android.util.Log

3040. import androidx.lifecycle.ViewModel

3041. import androidx.lifecycle.viewModelScope

3042. import com.example.iot\_ha.data.local.RoomLocalDatabase

3043. import com.example.iot\_ha.data.local.broker.BrokerState

3044. import com.example.iot\_ha.data.local.command.Command

3045. import com.example.iot\_ha.data.local.device.Device

3046. import com.example.iot\_ha.data.mqtt.MQTTClient

3047. import kotlinx.coroutines.delay

3048. import kotlinx.coroutines.flow.MutableStateFlow

3049. import kotlinx.coroutines.flow.StateFlow

3050. import kotlinx.coroutines.flow.collectLatest

3051. import kotlinx.coroutines.flow.update

3052. import kotlinx.coroutines.launch

3053.

3054. open class DevicesViewModel(private val db: RoomLocalDatabase) : ViewModel() {

3055. private val \_devices = MutableStateFlow<List<Device>>(emptyList())

3056. val devices: StateFlow<List<Device>> = \_devices

3057.

3058. init {

3059. viewModelScope.launch {

3060. BrokerState.brokerId.collectLatest { brokerId ->

3061. if (brokerId != null) {

3062. loadDevices(brokerId)

3063. }

3064. }

3065.

3066. while (true) {

3067. updateDevices()

3068. delay(3000)

3069. }

3070. }

3071. }

3072.

3073.

3074. private suspend fun loadDevices(brokerId: Int) {

3075. db.deviceDAO().getDevicesByBrokerFlow(brokerId)

3076. .collect { deviceList ->

3077. \_devices.value = deviceList

3078. }

3079. }

3080.

3081. private suspend fun updateDevices() {

3082. val devicesList = db.deviceDAO().getAllDevices()

3083. \_devices.value = devicesList

3084. }

3085.

3086.

3087. fun getDeviceIdByIeeeAddr(ieeeAddr: String, callback: (Int?) -> Unit) {

3088. viewModelScope.launch {

3089. val device = db.deviceDAO().getDeviceByIeeeAddr(ieeeAddr)

3090. callback(device?.id)

3091. }

3092. }

3093.

3094.

3095. fun getDevicesByTypeFlow(type: String): StateFlow<List<Device>> {

3096. val resultFlow = MutableStateFlow<List<Device>>(emptyList())

3097.

3098. viewModelScope.launch {

3099. db.commandDAO().getCommandsByTypeFlow(type).collect { commands ->

3100. val deviceIds = commands.map { it.deviceId }.toSet()

3101. val filteredDevices = \_devices.value.filter { it.id in deviceIds }

3102. resultFlow.value = filteredDevices

3103. }

3104. }

3105.

3106. return resultFlow

3107. }

3108.

3109. fun getDevicesWithoutCommandsFlow(): StateFlow<List<Device>> {

3110. val resultFlow = MutableStateFlow<List<Device>>(emptyList())

3111.

3112. viewModelScope.launch {

3113. db.deviceDAO().getAllDevicesFlow().collect { devices ->

3114. db.commandDAO().getAllCommandsFlow().collect { commands ->

3115. val deviceIdsWithCommands = commands.map { it.deviceId }.toSet()

3116. val devicesWithoutCommands = devices.filter { it.id !in deviceIdsWithCommands }

3117. resultFlow.value = devicesWithoutCommands

3118. }

3119. }

3120. }

3121.

3122. return resultFlow

3123. }

3124.

3125.

3126. fun getDevicesByRoomIdFlow(roomId: Int): StateFlow<List<Device>> {

3127. val resultFlow = MutableStateFlow<List<Device>>(emptyList())

3128.

3129. viewModelScope.launch {

3130. db.deviceDAO().getDevicesByRoomIdFlow(roomId).collect { devices ->

3131. val deviceIds = devices.map { it.id }.toSet()

3132. val filteredDevices = \_devices.value.filter { it.id in deviceIds }

3133. resultFlow.value = filteredDevices

3134. }

3135. }

3136.

3137. return resultFlow

3138. }

3139.

3140. fun addDeviceIfNotExists(device: Device) {

3141. viewModelScope.launch {

3142. val existingDevice = db.deviceDAO().getDeviceByIeeeAddr(device.ieeeAddr)

3143.

3144. Log.i("DEVLIST", devices.value.toString())

3145.

3146. if (existingDevice == null) {

3147. db.deviceDAO().insertDevice(device)

3148. Log.i("DevicesViewModel", "Device added: $device")

3149. } else {

3150. Log.i("DevicesViewModel", "Device already exists: $device")

3151. }

3152. }

3153. }

3154.

3155. fun updateDeviceName(deviceId: Int, newName: String) {

3156. viewModelScope.launch {

3157. val device = db.deviceDAO().getDeviceById(deviceId)

3158.

3159. val updatedDevice = device.copy(friendlyName = newName)

3160.

3161. db.deviceDAO().updateDevice(updatedDevice)

3162. \_devices.update { list ->

3163. list.map {

3164. if (it.id == deviceId) updatedDevice else it

3165. }

3166. }

3167. }

3168. }

3169.

3170.

3171. fun addCommandIfNotExists(command: Command) {

3172. Log.i("addCommandIfNotExists", command.toString())

3173.

3174. viewModelScope.launch {

3175. val existingCommand = db.commandDAO().getCommandByCommandTopic(command.commandTopic)

3176.

3177. if (existingCommand == null) {

3178. db.commandDAO().insertCommand(command)

3179. } else {

3180. Log.i("DevicesViewModel", "Command already exists: $existingCommand")

3181. }

3182. }

3183. }

3184.

3185. fun onToggle(deviceId: Int, state: Boolean) {

3186. viewModelScope.launch {

3187. val cmd = db.commandDAO().getSwitchCommandByDeviceId(deviceId)

3188.

3189. val newState = if (!state) cmd?.payloadOff else cmd?.payloadOn

3190. cmd?.let {

3191. if (newState != null) {

3192. sendCommandToMqtt(it.commandTopic, newState)

3193. }

3194. }

3195. }

3196. }

3197.

3198. fun onSelectChange(deviceId: Int, option: String) {

3199. viewModelScope.launch {

3200. sendCommandToMqtt("command", "")

3201. }

3202. }

3203.

3204. fun onValueChange(deviceId: Int, value: Int) {

3205. viewModelScope.launch {

3206. sendCommandToMqtt("command", "")

3207. }

3208. }

3209.

3210. private fun sendCommandToMqtt(topic: String, command: String) {

3211. val mqttClient = MQTTClient.getInstance()

3212. mqttClient.publish(topic, command)

3213. println("Отправка в MQTT: $command")

3214. }

3215.

3216. fun assignRoomToDevice(deviceId: Int, roomId: Long?) {

3217. viewModelScope.launch {

3218. val device = db.deviceDAO().getDeviceById(deviceId)

3219. val updatedDevice = device.copy(roomId = roomId)

3220. db.deviceDAO().updateDevice(updatedDevice)

3221. }

3222. }

3223.

3224. }

3225.

RoomsViewModel.kt

3226. package com.example.iot\_ha.ui.viewmodels.shared

3227.

3228. import androidx.lifecycle.ViewModel

3229. import androidx.lifecycle.viewModelScope

3230. import com.example.iot\_ha.data.local.RoomLocalDatabase

3231. import com.example.iot\_ha.data.local.room.RoomEntity

3232. import kotlinx.coroutines.flow.MutableStateFlow

3233. import kotlinx.coroutines.flow.StateFlow

3234. import kotlinx.coroutines.launch

3235.

3236. class RoomsViewModel(private val db: RoomLocalDatabase) : ViewModel() {

3237. private val \_rooms = MutableStateFlow<List<RoomEntity>>(emptyList())

3238. val rooms: StateFlow<List<RoomEntity>> = \_rooms

3239.

3240. init {

3241. viewModelScope.launch {

3242. db.roomEntityDAO().getAllRooms().collect { \_rooms.value = it }

3243. }

3244. }

3245.

3246. fun addRoom(name: String) {

3247. viewModelScope.launch {

3248. db.roomEntityDAO().insertRoom(RoomEntity(name = name))

3249. }

3250. }

3251.

3252. fun getDeviceCount(roomId: Long, onResult: (Int) -> Unit) {

3253. viewModelScope.launch {

3254. val count = db.deviceDAO().getDeviceCountForRoom(roomId)

3255. onResult(count)

3256. }

3257. }

3258.

3259. fun deleteRoom(roomId: Int) {

3260. viewModelScope.launch {

3261. db.roomEntityDAO().deleteRoom(roomId)

3262. }

3263. }

3264. }

SensorsViewModel.kt

3265. package com.example.iot\_ha.ui.viewmodels.shared

3266.

3267. import androidx.lifecycle.ViewModel

3268. import com.example.iot\_ha.ui.viewmodels.shared.data.SensorsData

3269. import kotlinx.coroutines.flow.MutableStateFlow

3270. import kotlinx.coroutines.flow.StateFlow

3271.

3272. class SensorsViewModel : ViewModel() {

3273. private val \_sensorData = MutableStateFlow<Map<String, SensorsData>>(emptyMap())

3274. val sensorData: StateFlow<Map<String, SensorsData>> = \_sensorData

3275.

3276. fun updateSensorData(topic: String, payload: String) {

3277. \_sensorData.value += (topic to SensorsData(topic, payload))

3278. }

3279. }

SensorsData.kt

3280. package com.example.iot\_ha.ui.viewmodels.shared.data

3281.

3282. data class SensorsData(

3283. val topic: String,

3284. val payload: String,

3285. val timestamp: Long = System.currentTimeMillis()

3286. )

Constants.kt

3287. package com.example.iot\_ha.utils

3288.

3289. object Constants {

3290. const val SWITCH\_TYPE = "switch"

3291. const val SELECT\_TYPE = "select"

3292. const val DIMMER\_TYPE = "dimmer"

3293.

3294. val TABS\_LIST = listOf("Devices", "Rooms", "Settings")

3295.

3296. const val DISCOVERY\_ENABLE = "true"

3297. const val DISCOVERY\_DISABLE = "false"

3298. const val DISCOVERY\_TIME = 255\_000L

3299. }

DeviceActionBuilder.kt

3300. class DeviceActionBuilder(private val deviceId: Int) {

3301. fun buildOnToggle(): (Int, Boolean) -> Unit = { id, state ->

3302. if (id == deviceId) {

3303. println("[$deviceId] Toggle changed: $state")

3304. }

3305. }

3306.

3307. fun buildOnSliderChange(): (Int, Float) -> Unit = { id, value ->

3308. if (id == deviceId) {

3309. println("[$deviceId] Slider value: $value")

3310. }

3311. }

3312.

3313. fun buildOnSelectChange(): (Int, String) -> Unit = { id, option ->

3314. if (id == deviceId) {

3315. println("[$deviceId] Selected: $option")

3316. }

3317. }

3318. }

Extensions.kt

3319. package com.example.iot\_ha.utils

3320.

3321. fun String.toBooleanState(): Boolean = when (this) {

3322. "ON", "true", "1" -> true

3323. else -> false

3324. }

Logger.kt

3325. package com.example.iot\_ha.utils.logging

3326.

3327. import android.util.Log

3328. import kotlin.reflect.KClass

3329.

3330. object Logger {

3331. fun log(className: KClass<\*>, message: String) {

3332. Log.i(className.simpleName, message)

3333. }

3334.

3335. fun log(tag: String, message: String) {

3336. Log.i(tag, message)

3337. }

3338. }

ExampleUnitTest.kt

3339. package com.example.iot\_ha

3340.

3341. import org.junit.Assert.assertEquals

3342. import org.junit.Test

3343.

3344. /\*\*

3345. \* Example local unit test, which will execute on the development machine (host).

3346. \*

3347. \* See [testing documentation](http://d.android.com/tools/testing).

3348. \*/

3349. class ExampleUnitTest {

3350. @Test

3351. fun addition\_isCorrect() {

3352. assertEquals(4, 2 + 2)

3353. }

3354. }

ПРИЛОЖЕНИЕ Б

(обязательное)

Спецификация

ПРИЛОЖЕНИЕ В

(обязательное)

Ведомость документов