­ **Министерство образования и науки Российской Федерации**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Санкт-Петербургский государственный университет**

**промышленных технологий и дизайна»**

Институт информационных технологий и автоматизации

Направление подготовки 09.03.02 Информационные системы и технологии

Выпускающая кафедра интеллектуальных систем и защиты информации

Допустить к защите

Заведующий кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 г.

**ВЫПУСКНАЯ**

**КВАЛИФИКАЦИОННАЯ РАБОТА**

**бакалаврская работа**

на тему “Разработка архитектуры и реализация системы умного дома”

Исполнитель – обучающийся учебной группы 4-МД-16

Федоренко Егор Олегович

Руководитель выпускной квалификационной работы

доцент, к.т.н. Бусыгин Константин Николаевич

Нормоконтролер: старший преподаватель Чалова Екатерина Игорьевна

**Санкт-Петербург**

**2023**

Реферат

Пояснительная записка содержит //75 стр., 63 рис, 9 источников.

Перечень ключевых слов: умный дом, Java, Flutter, разработка архитектуры, лампа с Wi-Fi.

Тема: «Разработка архитектуры и реализация системы умного дома».

Целью работы является разработка системы умного дома с возможностью удаленного управления и настройки через мобильное приложение.

В ходе работы планируется изучение теоретической и практической части по разработке мобильного приложения на базе фреймворка Flutter и серверной части на языке программирования Java.

В первой главе была создана архитектура умного дома. Так же были выбраны технологии и фреймворки, которые обеспечат работоспособность планируемой системы и выбраны языки программирования. Был рассмотрен вопрос безопасности передачи данных между спроектированными системами.

Во второй главе была реализована система умного дома с аргументацией примененных технических решений. Были описаны все возможности системы, как происходит взаимодействие с устройствами, сервером, мобильным приложением и базой данных.

В третьей главе был описан способ внедрения системы умного дома и его первоначальной настройки.

В четвертой главе был выполнен технико-экономический расчет, в котором были подсчитаны все затраты при создании данной системы и ее внедрения в дом.

**ОГЛАВЛЕНИЕ**

[Реферат 0](#_Toc134636281)

[ВВЕДЕНИЕ 3](#_Toc134636282)

[ГЛАВА 1. Концепция умного дома 5](#_Toc134636283)

[1.1 Термин умного дома 5](#_Toc134636284)

[1.2 Историческая справка 5](#_Toc134636285)

[1.3 Производители устройств умного дома 6](#_Toc134636286)

[1.3.1 AJAX 7](#_Toc134636287)

[1.3.2 Aqara 9](#_Toc134636288)

[1.3.3 Яндекс 10](#_Toc134636289)

[1.4 Связь устройств умного дома 12](#_Toc134636290)

[1.4.1 ZigBee 12](#_Toc134636291)

[1.4.2 Wi-Fi 13](#_Toc134636292)

[1.4.3 Bluetooth 14](#_Toc134636293)

[1.4.4 Z-Wave 14](#_Toc134636294)

[1.4.5 Matter 15](#_Toc134636295)

[1.5 Управление умным домом 15](#_Toc134636296)

[ГЛАВА 2. РАЗРАБОТКА АРХИТЕКТУРЫ СИСТЕМЫ УМНОГО ДОМА 18](#_Toc134636297)

[2.1 Задачи, решаемые разработкой архитектуры ПО 18](#_Toc134636298)

[2.2 Разработка требований для умного дома 19](#_Toc134636299)

[2.3 Задачи, решаемые подсистемами умного дома 20](#_Toc134636300)

[2.4 Выбор инструментов для разработки 21](#_Toc134636301)

[2.4.1 Выбор решения для компонента сервера и хаба. 21](#_Toc134636302)

[2.4.2 Выбор базы данных 24](#_Toc134636303)

[2.4.3 Выбор фреймворка для клиентского приложения. 27](#_Toc134636304)

[2.5 Выбор устройств и протокола для работы системы умного дома 29](#_Toc134636305)

[ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ УМНОГО ДОМА НА БАЗЕ ВЫБРАННОЙ АРХИТЕКТУРЫ 31](#_Toc134636306)

[3.1 Разработка мобильного приложения 31](#_Toc134636307)

[3.2 Разработка серверного приложения 47](#_Toc134636308)

[ГЛАВА 4. ВОЗМОЖНОСТИ РАСШИРЕНИЯ УМНОГО ДОМА 48](#_Toc134636309)

[ГЛАВА 5. ТЕХНИКО-ЭКОНОМИЧЕСКИЙ РАСЧЕТ 49](#_Toc134636310)

[ЗАКЛЮЧЕНИЕ 50](#_Toc134636311)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 51](#_Toc134636312)

ВВЕДЕНИЕ

**Актуальность.** С каждым годом технологии становятся более доступными и все больше встраиваются в повседневную жизнь людей, делая ее более комфортной и практичной.

Благодаря быстрому развитию компьютерных систем появилась возможность удаленно контролировать и автоматизировать любые процессы. Например, при помощи автоматизированных систем контроля отпадает нужда в личной сверке данных или личном присутствии человека, следящим за некоторым объектом, так как данные системы способны самостоятельно собирать данные, при необходимости - проверять их, и отображать в любом удобном формате в виде текста, графиков или картинок. В данном случае системы автоматизации и контроля удобнее и проще в использовании, так как при помощи одного управляющего приложения можно следить за состоянием всех критичных для работы объекта систем в одном месте в легком для понимания виде. Также это помогает дополнительно обезопасить объект в случае несанкционированного проникновения. Среди других плюсов, такая система способна самостоятельно отрабатывать ситуации внештатные ситуации, например отключать электрический щит при обнаруженной протечке воды.

Данные системы существенно упрощают процесс обслуживания любого объекта, так как позволяют решить такие рутинные задачи, как:

* Получение состояния отдельных датчиков, например, датчик протечки воды, состояние световых ламп
* Сбор информации о работе устройств
* Отправка уведомления пользователю в случае нештатной ситуации, например при протечке воды или открытии двери при включенной сигнализации
* Удаленное управление отдельными устройствами.

Данные системы могут обеспечивать безопасность и работоспособность не только коммерческих объектов, но и домов и квартир частных лиц. Примером такой системы может являться умный дом.

**Объект исследования:** технология беспроводного управления беспроводными устройствами в системе умного дома.

**Предмет исследования:** внутреннее устройство, безопасность и конфиденциальность систем умного дома.

**Цель работы и задачи исследований.** Целью данной работы является разработка системы умного дома, позволяющая удаленно управлять подключенными устройствами через мобильное приложение. Для достижения поставленной цели необходимо решить следующие задачи:

1. Выбор необходимой архитектуры для умного дома
2. Разработка серверной части системы
3. Написание мобильного приложения – клиента
4. Создание системы домашнего хаба для объединения устройств умного дома
5. Рассмотрение вопроса безопасности передаваемых данных через сеть Интернет
6. Развертывание приложения на рабочем окружении

В рамках проведённых теоретических и экспериментальных исследований все результаты были получены автором самостоятельно.

Выпускная квалификационная работа состоит из введения, пяти глав, заключения и списка используемой литературы. Основное содержание изложено на 75 стр., включая 63 рисунка. Список литературы содержит 9 наименований.

ГЛАВА 1. Концепция умного дома

## **1.1 Термин умного дома**

Умным домом можно назвать систему, которая собирает данные с различных датчиков, установленных в доме пользователя, и удаленно управляет ими при помощи команд из пользовательского интерфейса. Умный дом – это гибко настраиваемая система, с ее помощью можно автоматизировать практически любые домашние процессы.

## **1.2 Историческая справка**

Первые попытки автоматизировать дом были предприняты в середине двадцатого века. Журнал “Popular Mechanincs” в издании за декабрь 1950-года напечатал историю об изобретателе Эмиле Мэтиасе [1]. Он встроил в свой дом в Джексоне, штат Мичиган, США, множество кнопок, которые отвечали за выполнение определенных действий. Например, Эмиль с помощью кнопки мог открыть дверь гаража или включить и выключить радио. На рисунке 1 приведен план его дома с обозначениями установленных устройств. Этот дом получил прозвище «Push-Button Manor», что в вольном переводе означает «кнопочное поместье». Из статьи также можно узнать, что изобретатель спрятал в стенах своего дома больше семи тысяч футов проводки, а управляющая часть этого умного дома находилась в его кладовке. Журналист Артур Рейлтон оценил качество работы изобретателя: в доме нет ни одного висящего провода, который бы выдавал наличие любых вспомогательных систем.

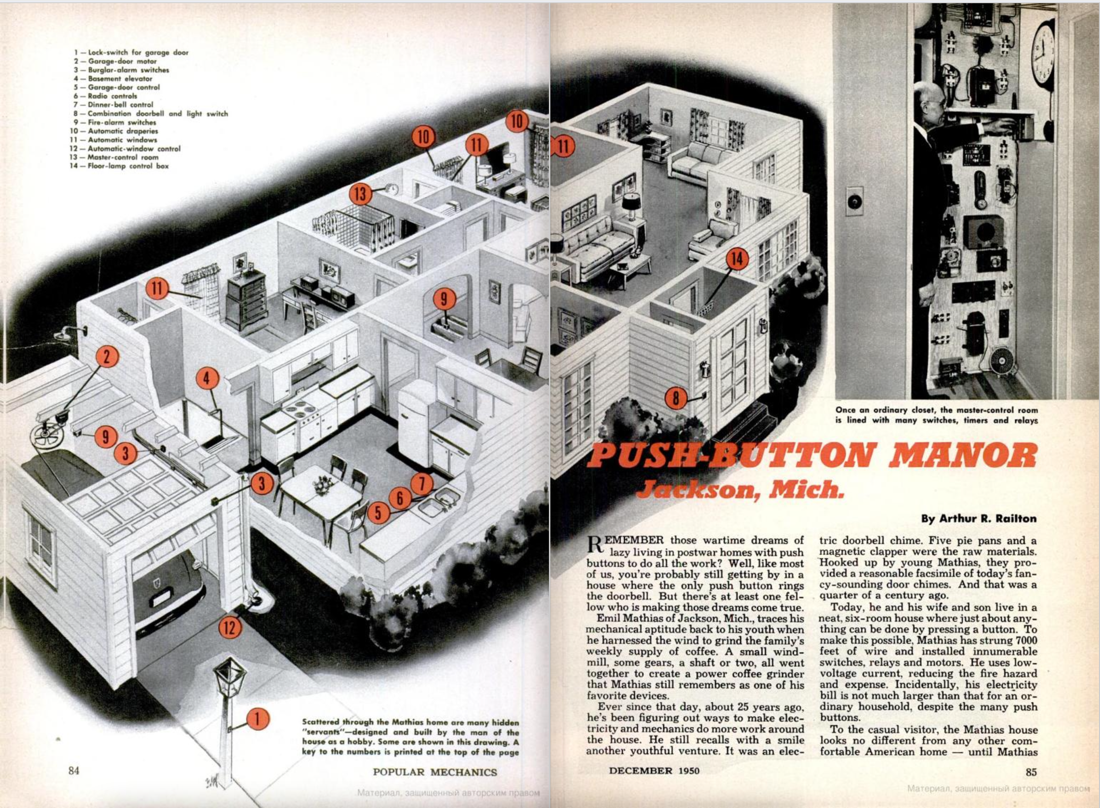


Рисунок 1. Планировка умного дома Эмиля Мэтиаса

Через двадцать пять лет, в 1975-ом году, компания Pico Electronics создала собственный стандарт для связи устройств: X10. Данный стандарт управлял электрическими приборами при помощи силовых линий, позволяя включать и выключать их при помощи пульта дистанционного управления. Это решение популярно в виду дешевизны и простоты в установке.

Можно увидеть, что интерес к автоматизации жилых помещений с каждым годом только растет. Например, согласно исследованию о российском рынке умного дома [2], количество хабов умного дома в России в 2022 году составило 176 тысяч штук, “рост продаж в 2 раза год к году”. Это говорит об интересе людей к новшествам в сфере автоматизации умных домов и их открытости к новым технологиям.

## **1.3 Производители устройств умного дома**

На данный момент существует множество реализаций умного дома от разных производителей. Данные решения продаются как отдельными устройствами, так и набором, позволяя пользователю самостоятельно решить, какие системы дома он хочет автоматизировать и иметь возможность управлять удаленно.

Лидирующими производителями устройств для умного дома являются компании AJAX, Aqara и Яндекс (от mobile-review). Рассмотрим каждого из производителей подробнее.

### 1.3.1 AJAX

AJAX – это украинская компания, специализирующаяся на повышении безопасности объектов путем установки датчиков по всему дому. Их устройства созданы для защиты объектов при нештатных ситуациях, таких как:

* Взлом дверей
* Прорыв труб
* Пожар
* Попытка ограбления

Управление происходит через собственное мобильное приложение, позволяющее следить за безопасностью объектов. Для работы системы необходим хаб, который следит за состоянием всех установленных датчиков и передает данные на сервер.

Важной функцией их хабов является дублирование систем энергообеспечения. В случае, если электрическая сеть дома будет отключена, например, из общего щитка в подъезде, то хаб имеет возможность через SIM-карту, вставленную в нее, отправлять уведомления о нарушении электропитания, так как хаб имеет встроенный аккумулятор. Так же и с WI-FI: если хаб потеряет домашнюю сеть WI-FI, то он может «общаться» с серверами систем через мобильную сеть или SMS-сообщения [3] (https://ajax.systems/ru-ua/products/hubs/). Внешний вид хаба отображен на рисунке 2.



Рисунок 2. Хаб компании Ajax

Отличительной чертой данных устройств является возможность подключения к пульту охранной системы. Системы AJAX могут отправлять сигнал тревоги, на который отреагируют сотрудники правоохранительных органов. (https://volkodavufa.ru/article/obzory/ajax-systems-security-system-for-home,-apartment-and-office)

Список возможных к установке датчиков достаточно большой, среди них:

* Датчики протечки
* Датчики движения
* Пожарные датчики
* Датчики открытия
* Датчики разбития стекла
* Сирены
* Ретрансляторы
* Тревожные кнопки
* Устройства автоматизации

1.3.2 Aqara

Другой компанией, поставляющей устройства умного дома, является Aqara. Aqara – это суб-бренд китайской компании Xiaomi, который специализируется на электронике и датчиках умного дома. Выбор устройств у компании достаточно обширен – это:

* Веб-камеры
* Умные замки
* Умные колонки с голосовыми ассистентами
* Шторы с электроприводом
* Лампы
* Реле
* Датчики протечки
* Мониторы качества воздуха
* Датчики открытия дверей
* Датчики движения
* Датчик вибраций

В системе умного дома от Aqara реализована система сценариев. Сценарии – это автоматический старт последовательности действий при совершении указанного пользователем действия. Например, на телефон придёт уведомление, если сработает датчик открытия дверей. Также может начать работать сирена, оповещающая о тревоге. Приведу другой сценарий: при срабатывании датчика вибрации на телефон придет соответствующее уведомление. Эти устройства помогают обезопасить помещение и быстрее отреагировать в случае нештатной ситуации. На рисунке 3 отображен датчик открытия дверей.



Рисунок 3. Датчик открытия дверей

Важной функцией умного дома от Aqara является возможность добавления пользователей в уже существующий умный дом. Например, администратор такого умного дома может открыть доступ к устройствам другим людям (например, членам семьи). Эта функция позволяет делегировать полномочия по использованию устройств и следить за состоянием умного дома и следить за статистикой использования датчиков.

Управлением умным домом также происходит через мобильное приложение, которое соединяется с серверами и передает данные к хабу, который в свою очередь также следит за состоянием установленных в доме датчиков и передает информацию на сервера компании.

1.3.3 Яндекс

Другим большим производителей умных устройств является компания Яндекс. Среди их устройств можно отметить «Яндекс.Станция», умную колонку с голосовым управлением.

У Яндекса набор предметов домашней автоматизации меньше, чем у Aqara:

* Датчики движения и освещения
* Беспроводные колонки
* Датчики температуры и влажности
* Датчики протечки воды
* Wi-Fi лампы
* Хабы
* Музыкальные колонки

На рисунке 4 изображен датчик движения от Яндекса.



Рисунок 4. Датчик движения

Управление умным домом осуществляется при помощи голосового ассистента «Алиса» и мобильного приложения.

Отличительной чертой устройств Яндекса является необязательность установки хаба при наличии Яндекс.Станции. Станция используется одновременно и как хаб для связи с устройствами, и как музыкальная колонка с голосовым ассистентом. Это решение можно назвать практичным и удобным, так как не нужно покупать отдельное устройство для установки связи с умными устройствами.

В умном доме от Яндекса поддерживаются и другие устройства, работающие по протоколу ZigBee.

1.4 Связь устройств умного дома

Устройства умного дома постоянно подключены к сети и обмениваются данными с сервером для быстрой и надежной обработки команд. Разные производители используют разные решения и методы передачи данных в своих решениях. Например, в системах умного дома устройства могут обмениваться командами по протоколу ZigBee, Wi-Fi, Bluetooth и другими. Рассмотрю протоколы для устройств умного дома.

1.4.1 ZigBee

ZigBee - это протокол беспроводной связи, который был разработан для создания низкопотребляющих и недорогих сетей устройств, используемых в умном доме, промышленности и других областях. Протокол ZigBee работает на частоте 2,4 ГГц и имеет дальность передачи данных до 100 метров в открытом пространстве.

Одной из главных особенностей протокола ZigBee является его низкое энергопотребление. Устройства, работающие на протоколе ZigBee, могут работать на батарейках до нескольких лет, что делает его хорошим выбором для использования в умном доме, так как не придется постоянно следить за уровнем батареи в устройстве.

Протокол ZigBee поддерживает множество топологий сетей, включая звезду, сетку и дерево. При типе «сетка» количество переходов при отправке от устройства к устройству неограничено. Также при данной топологии необязательно постоянное подключение к хабу. Максимальное расстояние между устройствами для передачи данных может достигать 20 метров. Максимальное число поддерживаемых устройств – 65000. Протокол также имеет множество функций безопасности, таких как шифрование по стандарту AES-128 и аутентификация, что делает его более надежным и безопасным для использования.

Устройства ZigBee работают на частоте 2.4 ГГц, таком же, как и Wi-Fi, возможны помехи в их работе. В россии – 928 мгц(https://ixsmart.ru/umnyj-dom/zigbee-ili-z-wave-chto-vybrat-dlya-umnogo-doma/)

1.4.2 Wi-Fi

Wi-Fi — это популярный протокол беспроводной связи, который позволяет передавать данные по радиоволнам. В контексте умного дома, Wi-Fi используется для подключения устройств к сети Интернет и обеспечивает их удаленное управление через приложение.

Одним из главных преимуществ Wi-Fi является его широкое распространение и доступность. Большинство домов уже имеют установленную Wi-Fi-сеть, что делает его очень удобным для использования в умном доме. Также Wi-Fi имеет высокую скорость передачи данных и обеспечивает высокую производительность устройств в умном доме. Это позволяет управлять устройствами в режиме реального времени и обеспечивает быстрый доступ к данным и контролю устройств.

Среди недостатков этого протокола можно отметить высокое энергопотребление и не самую высокую надежность для использования в критических системах умного дома. Кроме того, Wi-Fi может столкнуться с проблемами перегруженности сети, особенно если в доме много WI-FI устройств.

1.4.3 Bluetooth

Bluetooth — это протокол беспроводной связи, который используется для передачи данных между устройствами в непосредственной близости друг от друга. В контексте умного дома, Bluetooth редко используется для непосредственной работы системы умного дома. Некоторые производители используют его для первоначальной настройки устройства.

Одной из главных преимуществ протокола Bluetooth является его низкое энергопотребление. Bluetooth устройства могут работать на батарейках до нескольких лет, это делает его неплохим возможным вариантом для использования в системах, рассчитанных на долгое время работы.

Bluetooth имеет невысокую скорость передачи данных (1 Мбит/с) и обеспечивает нормальную передачу данных.

Однако, Bluetooth имеет свои недостатки. У данного протокола небольшой радиус действия, что ограничивает его использование в больших домах. Кроме того, Bluetooth не поддерживает топологию «сетка», не позволяя через себя передавать данные на другие устройства, что является критичным фактором не в пользу данного протокола.

1.4.4 Z-Wave

Z-Wave — это протокол беспроводной связи, который используется для создания сетей устройств в умном доме.

Протокол похож по своим характеристикам на ZigBee: он также использует топологию сети «сетка», устройства также могут передавать через себя команды на другие устройства, имеет такой же стандарт шифрования передаваемых данных AES-128. Отличия заключаются в ограниченности количества переходов между устройствами: «переслать» через себя команду могут максимум 4 устройства. Также ограничено максимальное количество устройств – всего 232.

Протокол функционирует на волне 800–900 МГц, на данном радиодиапазоне…

Для функционирования Z-Wave необходим центральный хаб.

1.4.5 Matter

Matter - это протокол для умного дома, который разрабатывается в рамках сотрудничества между крупными компаниями, такими как Apple, Amazon, Google и др. Он был создан для упрощения и унификации процесса взаимодействия между устройствами в умном доме.

Matter выделяется своей универсальностью. Он позволяет различным устройствам в умном доме взаимодействовать между собой, независимо от их производителя, типа или используемого протокола. Это значит, что устройства, работающие на протоколе Matter, могут легко интегрироваться и работать вместе, что делает управление умным домом более удобным и простым.

Matter позволяет, например, использовать Bluetooth для первоначальной настройки устройства, а Wi-Fi – для передачи данных.

Протокол Matter также обеспечивает высокую безопасность и защиту данных. Он использует шифрование и аутентификацию для защиты данных и устройств в умном доме от взлома и хакерских атак.

Таким образом, Matter – это перспективный стандарт для решения проблемы использования разными устройствами разных протоколов.

1.5 Управление умным домом

В прошлом параграфе было выяснено, что множество производителей оборудования для умного дома используют самые разнообразные протоколы связи, и вместе с этим разрабатывают свое мобильное приложение для оркестрирования устройствами. Это ведет к разрозненному и неудобному управлению умным домом через множество приложений. Для избежания путаницы существует множество фреймворков для объединения устройств в одном приложении. Такими приложениями являются Apple HomeKit, Google Home и Samsung SmartThings.

Данные системы позволяют подключать к себе как уже установленные устройства от других производителей, так и устройства, работающие напрямую с экосистемой Google, Apple и Samsung. Такие решения позволяют бесшовно объединить устройства в одном приложении, добавляя удобства при использовании умного дома. Пример интерфейса Apple HomeKit отображен на рисунке 5.



Рисунок 5. Интерфейс Apple HomeKit

Можно сделать вывод, что производители устройств умного дома в своих решениях используют связку хаба и мобильного приложения для управления умным домом как самый надежный и удобный. Хаб в данных решениях отвечает за обработку данных и передачу команд с и от устройств, что существенно облегчает управление устройствами в плане разработки такого решения.

Система умного дома существенно облегчает жизнь людей, позволяя удаленно следить за состоянием жилья через Интернет. Встает вопрос о приватности такого решения: как происходит обработка данных, имеет ли производитель полный доступ к устройствам и насколько эта интеграция устройств в жизнь человека расходится с правами на неприкосновенность частной жизни.

Существуют открытые программные решения для оркестрирования устройствами умного дома, которые пользователь может самостоятельно установить дома, но это предполагает более глубокого понимания устройства сети, а также времени. Такие решения существуют скорее для энтузиастов, а не для обычного потребителя, так как самостоятельная настройка может запутать людей.

Также существуют опасения, что компании-производители могут использовать свои устройства для продажи данных пользователей в маркетинговых целях. Не маловероятно, что устройства могут быть скомпрометированы для слежки за пользователями. Производители не выкладывают ПО в открытый доступ для общественного анализа систем, из-за чего возникает вопрос о приватности и методах обработки данных компаниями. Пользователям остается вслепую доверять производителям, что они не будут использовать или обрабатывать информацию с устройств умного дома в корыстных целях.

Данные опасения побудили меня на создание собственной реализации умного дома.

ГЛАВА 2. РАЗРАБОТКА АРХИТЕКТУРЫ СИСТЕМЫ УМНОГО ДОМА

Архитектура программного обеспечения – это способ связи компонентов системы между собой. Не существует четкого определения, что такое архитектура ПО, это можно назвать «рекомендациями» как написать поддерживаемый и чистый код.

Архитектура приложения определяет, какие функции системы за что отвечают и как они взаимодействуют между собой. Архитектура приложения разрабатывается после анализа требований к системе.

Можно провести параллели между планом здания и программой: то, по каким технологиям сооружено здание, определит его свойства, прочность, долговечность и надежность. Можно создать систему и без продумывания архитектуры и методов взаимосвязи компонентов, но это может усложнить процесс дальнейшего развития ПО в виду хаотичного расположения компонентов и сложной структуры, которая решала только сиюминутные задачи.

2.1 Задачи, решаемые разработкой архитектуры ПО

Разработанная и продуманная архитектура программного обеспечения позволять решить следующие задачи при создании необходимой системы:

* Выделяет внутреннюю структуру программы и описывает, на каких уровнях выполняются необходимые задачи.
* Упорядочивает взаимодействие внутренних элементов при вызове необходимых методов.
* Определяет первостепенные и второстепенные элементы для реализации в планируемой системе.
* Помогает определить необходимый стек технологий для имплементации в проекте.
* Дает представление о масштабируемости ПО и его слабых местах
* Упрощает дальнейшее сопровождение ПО.

Можно представить архитектуру ПО как многоуровневую абстракцию, где на верхнем уровне будет изображена концептуальная схема разрабатываемого приложения, а на нижнем – используемые фреймворки и технологии, с помощью которых компоненты системы будут взаимодействовать друг с другом.

2.2 Разработка требований для умного дома

Выделю требования, которым должна отвечать система умного дома.

* Легкость настройки, добавления и удаления умных устройств
* Безопасность передаваемых данных между компонентами системы
* Быстрота выполнения команд (меньше 2 секунд между отправкой команды и ее выполнения)
* Возможность управления устройствами вне дома
* Простота использования
* Сохранение метрик отдельных умных устройств
* Корректная обработка возможных ошибок
* Поддержка постоянного соединения с устройствами через сеть Интернет

Исходя из требований, система умного дома будет состоять из 3 компонентов: сервер, хаб и мобильное приложение. Данная архитектура используется в решениях от компаний-производителей, анализ которых был проведен в первой главе работы. Подобная архитектура позволяет удобно и удаленно управлять умными устройствами через мобильное приложение. Серверная часть должна принимать все запросы и обрабатывать их, а хаб – поддерживать соединение с сервером и устройствами. Схема решения нарисована на рисунке 6.

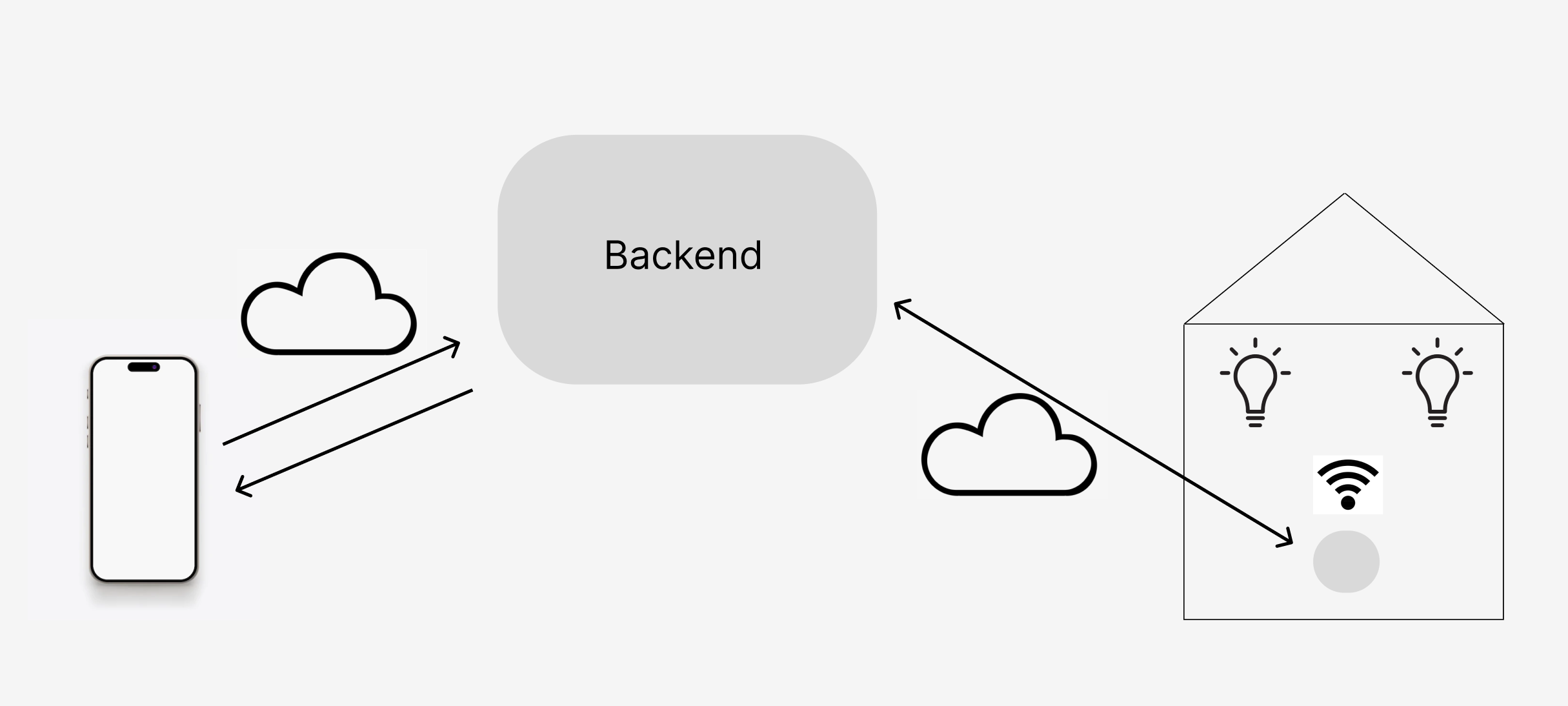


Рисунок 6. Архитектура приложения умного дома

2.3 Задачи, решаемые подсистемами умного дома

Каждая подсистема должна выполнять свои задачи в соответствие с требованиями к системе.

Составлю задачи, которые должны выполнять компоненты системы умного дома.

Задачи серверной части:

* аутентификация и авторизация пользователей
* поддержка протоколов шифрования передаваемых данных
* поддержка постоянного подключения к базе данных
* передача пользователю информацию о подключенных к его умному дому устройствах
* отправка и обработка команд для управления устройствами через собственный API
* сохранение метрик устройств в базе данных
* обработка возможных ошибок

Задачи мобильного приложения:

* корректное и понятное отображение устройств умного дома
* поддержка шифрования подключений
* возможность настройки каждого устройства
* удаление и добавление устройств
* удаление и добавление хабов

Задачи хаба:

* установка и поддержание постоянного защищенного соединения с сервером
* обработка команд от сервера
* передача на сервер ответов от умных устройств

Исходя из требований и задач каждого компонента системы, рассмотрю существующие фреймворки и языки программирования для реализации данных требований в системе умного дома.

2.4 Выбор инструментов для разработки

2.4.1 Выбор решения для компонента сервера и хаба.

Сервер в системе умного дома является головным элементом: он получает запросы от приложения и поддерживает подключение с хабом. Его задача – корректно получать любую информацию, обрабатывать и сохранять.

Серверная часть также должна шифровать данные и проверять пользователя на возможность совершения действий внутри системы.

Рассмотрим существующие фреймворки для разработки серверных решений.

*Spring Boot*

Spring Boot - это фреймворк для разработки приложений на языке Java, который позволяет быстро и легко создавать приложения с минимальной конфигурацией. Spring Boot использует принципы автоматической конфигурации и управлением зависимостями. Данный фреймворк обеспечивает высокую производительность и масштабируемость приложений, что делает его хорошим выбором для разработки как небольших, так и больших проектов.

Помимо Spring Boot, существуют другие фреймворки, входящие в проект Spring для разработки на Java. Рассмотрим их.

* Spring Security – компонент, который отвечает за настройку защиту приложения. Spring Security предоставляет множество функций, таких как защита от атак типа CSRF и XSS, поддержка различных механизмов аутентификации (например, по логину и паролю или OAuth 2), возможность настройки прав доступа на основе ролей и правил, поддержка шифрования паролей и другое. Spring Security является одним из самых популярных фреймворков для обеспечения безопасности приложений на языке Java.
* Spring Data – компонент, отвечающий за работу с базами данных. Он поддерживает как реляционные базы данных, например PostgreSQL, так и нереляционные БД (MongoDB). Spring Data позволяет разработчикам использовать простые интерфейсы для выполнения операций с базами данных, таких как создание, чтение, обновление и удаление данных. Он также предоставляет возможность автоматически создавать запросы на основе методов интерфейса, что упрощает написание кода. Также фреймворк обеспечивает поддержку транзакций, таким образом обеспечивая целостность данных.
* Spring MVC - это модуль для разработки веб-приложений. Он предоставляет возможности для работы с веб-компонентами, такими как HTTP-запросы и ответы, сессии, параметры запросов и другие. Spring MVC использует терн проектирования Model-View-Controller для разделения логики приложения на три компонента: модель, представление и контроллер. Модель представляет данные приложения и обеспечивает доступ к ним. Представление отвечает за отображение данных пользователю. Контроллер обрабатывает запросы пользователя и управляет взаимодействием модели и представления.
* Spring WebSocket - это модуль, который обеспечивает поддержку постоянной двусторонней связи между сервером и клиентом. Он использует протокол WebSocket для обмена данными. Spring WebSocket дает возможность создавать собственные WebSocket-эндпоинтов на сервере и поддерживает подключение к эндпоинтам на стороне клиента. Он также добавляет поддержку аннотаций для обработки сообщений, отправленных через WebSocket-соединение.

*Node.JS*

Данный фреймворк создан для запуска JavaScript кода в серверных приложениях. Одним из основных преимуществ Node.js является его асинхронная модель выполнения кода. Вместо того, чтобы блокировать выполнение кода до завершения операции ввода-вывода, Node.js использует неблокирующие операции ввода-вывода, что позволяет приложению продолжать работу без ожидания завершения операции ввода-вывода. Минимальная конфигурация Node.JS не позволяет написать приложение, однако большое количество пакетов доступно для скачивания, что добавляет гибкости и расширяемости в проект.

*Rails on Ruby*

Rails on Ruby - это фреймворк для веб-разработки, написанный на языке Ruby. Он был создан в 2004 году Дэвидом Хейнемеер Ханссоном и с тех пор стал одним из самых популярных инструментов для создания веб-приложений.

Rails предоставляет множество готовых решений для различных задач, таких как работа с базами данных, авторизация и аутентификация пользователей, обработка форм и многое другое. Это позволяет разработчикам сосредоточиться на создании бизнес-логики приложения, не тратя много времени на написание базового функционала.

Важной особенностью данного фреймворка является следование принципам конвенции над конфигурацией. Это означает, что изменять конфигурацию нужно только тогда, когда она не удовлетворяет требованиям. Это позволяет сосредоточиться на разработке, а не настройке программы.

При выборе фреймворка для написания системы стоит в первую очередь смотреть на язык программирования, который используется для создания приложения, скорость работы, уровень сложности и стоимость разработки. Все вышеперечисленные фреймворки способны выполнить серверные задачи, однако в данной работе будет использоваться фреймворк Spring и язык программирования Java. Данный выбор продиктован опытом разработки автором серверных приложений, а также удобной конфигурацией, позволяющей быстро создавать необходимые компоненты для нужд системы. Также важной составляющей при выборе является безопасность, так как Spring Security, входящий в проект Spring, поддерживает необходимые функции по авторизации и аутентификации. Функции шифрования передаваемых данных будут вынесены в веб-сервер nginx для разделения ответственности и удобства настройки бесплатных сертификатов шифрования Let’s Encrypt.

Список фреймворков и языков программирования для создания хаба умного дома совпадает со списком для серверной части системы. Для нужд хаба в этой работе выбран фреймворк Spring, в частности – Spring WebSocket, который позволит инициализировать подключение к серверу умного дома. Выбор данного фреймворка также продиктован удобством разработки и масштабируемостью.

2.4.2 Выбор базы данных

База данных – важнейшая часть любой системы. В ней будут хранится все данные обо всех устройствах, подключенных к умному дому пользователей. Помимо этого, она будет хранить данные о пользователях, статистику о каждом устройстве и другую важную информацию.

Существует множество систем управления базами данных, например: MySQL, PostgreSQL и MongoDB. Рассмотрим их подробнее.

MySQL — это система управления реляционными базами данных, которая используется для хранения, управления и доступа к данных. MySQL является одной из самых популярных СУБД в мире и используется в различных приложениях, включая веб-сайты, блоги, форумы, электронную коммерцию и многие другие. На рисунке 7 приведен список самых популярных СУБД.



Рисунок 7. Популярные базы данных

MySQL был разработан в 1995 году и с тех пор был приобретен компанией Oracle. Данная СУБД поддерживает множество функций, включая транзакции, хранимые процедуры, триггеры, индексы и многие другие. MySQL также поддерживает множество языков программирования, включая PHP, Java, Python и другие.

СУБД MySQL ориентирована на быстроту выполнения операций: в ней поддерживается стандартные типы данных, но, например, отсутствует тип данных jsonb, который есть в PostgreSQL.

Вторая для рассмотрения СУБД – PostgreSQL. Это система управления реляционными базами данных с открытым исходным кодом, которая используется для хранения, управления и доступа к большим объемам данных. PostgreSQL был разработан в 1986 году и с тех пор получил широкое распространение в различных приложениях, включая веб-сайты, приложения для анализа данных, финансовые системы и многие другие.

Одной из основных причин популярности PostgreSQL является его способность к обработке сложных запросов и поддержке расширенных функций, таких как триггеры, хранимые процедуры и транзакции.

PostgreSQL имеет множество функций, которые делают его более гибким и мощным, чем другие СУБД. Например, PostgreSQL поддерживает полнотекстовый поиск, географические данные и JSON-данные. Он также имеет множество инструментов и ресурсов для управления базами данных, таких как pgAdmin, phpPgAdmin и другие.

PostgreSQL имеет открытый исходный код и доступен для загрузки и использования бесплатно.

PostgreSQL является надежной и мощной СУБД, которая может обрабатывать большие объемы данных и поддерживать вложенные функции. Он также может быть легко масштабирован для управления большими объемами данных и высокой нагрузки.

Третья рассматриваемая база данных - MongoDB - документ-ориентированная база данных, которая хранит данные в виде документов, а не таблиц. Она позволяет гибко изменять структуру данных в процессе работы, не придерживаясь фиксированной схемы данных. MongoDB поддерживает горизонтальное масштабирование, что позволяет распределить данные на несколько серверов и обеспечить высокую доступность.

Данная СУБД имеет мощный язык запросов, который позволяет выполнять сложные запросы и агрегации данных. MongoDB также поддерживает индексы, что повышает производительность при выполнении запросов на больших объемах данных.

MongoDB имеет встроенную поддержку репликации и шардинга, что обеспечивает высокую доступность и масштабируемость. Она поддерживает различные форматы хранения данных, включая JSON, BSON и CSV.

Данная СУБД хорошо подходит для кеширования данных или для проектов, в которых структура данных может со временем измениться.

Исходя из перечисленных СУБД, в данной работе будет использоваться PostgreSQL. Данный выбор продиктован опытом автора в разработке баз данных и общей надежностью БД. В ней поддерживаются все принципы ACID, то есть атомарность, согласованность, изолированность и долговечность, что важно при хранении данных.

2.4.3 Выбор фреймворка для клиентского приложения.

Мобильное приложение является управляющим элементом системы умного дома. Через него пользователь будет взаимодействовать с умными устройствами.

Мобильное приложение должно без зависимости от местоположения пользователя отправлять запросы на сервер и получать от него ответ.

Существует множество фреймворков и языков программирования для создания мобильных приложений, среди самых популярных можно отметить:

* Фреймворк Flutter от Google
* Swift – для Apple устройств
* Kotlin – для Андроид устройств

Рассмотрю данные фреймворки.

Flutter - это фреймворк для разработки мобильных приложений, который был разработан компанией Google. Flutter использует язык программирования Dart и позволяет разработчикам создавать мобильные приложения для Android и iOS. Отличительная черта данного фреймворка – возможность создания приложения с использованием одинакового кода для нескольких платформ, например Android, iOS и Web. Такие приложения называются кроссплатформенные.

Flutter использует в себе концепцию виджетов – это отдельные элементы интерфейса, с которыми взаимодействует пользователь. Во всем приложении выстраивается иерархия виджетов, которые можно настроить согласно требованиям. Этими виджетами являются как иконки и кнопки, так и тулбары и невидимые пользователю элементы.

Flutter имеет множество преимуществ, которые делают его привлекательным для выбора. Одним из главных преимуществ является его быстродействие и производительность. Flutter использует свой собственный движок рендеринга, который позволяет создавать красивые и плавные пользовательские интерфейсы. Скорость рисовки интерфейса достигает 60 кадров в секунду, согласно заверениям Google.

Flutter также имеет мощную систему горячей перезагрузки, которая позволяет разработчикам мгновенно видеть изменения в приложении при внесении изменений в код. Это значительно ускоряет процесс разработки и упрощает тестирование приложений.

Следующий рассматриваемый объект – это язык программирования Swift, разработанный компанией Apple в 2014 году для создания приложений на iOS, macOS, watchOS и tvOS. Swift является мощным и гибким языком программирования, который может быть использован для создания различных типов приложений, включая мобильные приложения, настольные приложения и серверные приложения. Можно назвать Swift единственным выбором при написании нативного приложения для платформы Apple.

Swift также имеет мощную систему типов данных, которая позволяет разработчикам создавать безопасный и надежный код. В Swift имеется множество функций, которые делают его более гибким и мощным, таких как опциональные типы данных, функции высшего порядка, замыкания и другие.

Swift содержит множество инструментов и ресурсов для управления памятью, таких как автоматическое управление памятью и ARC (Automatic Reference Counting). Это позволяет разработчикам создавать приложения с высокой производительностью и эффективностью.

В Swift имеется множество инструментов и библиотек для создания приложений, таких как UIKit, SwiftUI, Core Data и другие. Он также имеет активное сообщество разработчиков, которые создают и поддерживают множество пакетов и библиотек для Swift.

Третий рассматриваемый элемент - Kotlin - это статически типизированный язык программирования, который работает на виртуальной машине Java. Разработан компанией JetBrains. Этот язык программирования сочетает в себе функциональные и объектно-ориентированные возможности, что делает его очень гибким и удобным для разработки широкого спектра приложений.

Kotlin также предоставляет множество возможностей для безопасности, сокращения количества ошибок и повышения производительности приложений.

Kotlin стал официальным языком разработки для Android, что привело к резкому росту его популярности в последние годы.

Для разработки мобильного приложения будет использоваться фреймворк Flutter. Данный выбор обоснован кроссплатформенностью, а значит и экономичностью разработки, так как приложение может быть выпущено под несколько платформ. Также для мобильного приложения не требуются специфичные для платформы функции, соответственно Flutter является хорошим выбором для написания приложения. Дополнительным плюсом в сторону выбора Flutter является отсутствие сложной логики в мобильном приложении: ему достаточно передавать и отправлять данные, а также корректно отрисовывать интерфейс.

2.5 Выбор устройств и протокола для работы системы умного дома

В первой главе были рассмотрены умные устройства, доступные потребителю для установки в своем жилье. Для данной работы будут использованы две Wi-Fi лампы Sonoff, которые будут установлены в цоколь и подключены в режиме DIY-mode. Данный режим позволять использовать API устройства для управления им в домашней сети.

Транспортом для передачи команд будет выступать Wi-Fi в виду простоты настройки и передачи данных.

Таким образом, в данной главе было выяснено, что для системы умного дома будут использоваться следующие фреймворки, программы и устройства:

* Spring Framework для серверной части и хаба умного дома
* Фреймворк Flutter для разработки мобильного приложения
* PostgreSQL для хранения данных
* nginx для обеспечения шифрования передаваемых данных через сеть
* Умные лампы Sonoff

ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ УМНОГО ДОМА НА БАЗЕ ВЫБРАННОЙ АРХИТЕКТУРЫ

В предыдущей главе было решено, что система умного дома будет написана на языках программирования Dart (для мобильного приложения) и Java (для сервера и домашнего хаба).

Базовый принцип работы системы умного дома следующий: пользователь из мобильного приложения отправляет запрос на сервер. Сервер авторизовывает пользователя, выполняет команду и отправляет ответ обратно пользователю. Таким образом будет происходить коммуникация систем при отправке команд из приложения.

В следующих параграфах опишем процесс разработки приложения.

3.1 Разработка мобильного приложения

Мобильное приложение в системе умного дома – главный метод управления системой. Оно должно отображать состояние подключенных устройств, управлять ими и уметь настраивать другие устройства.

Пользовательский интерфейс состоит из трех основных компонентов: авторизация, управление подключенными устройствами и настройки умного дома. Подробно опишем, как создавались эти компоненты.

Для компонента авторизации пользователя в мобильном приложении используется библиотека “flutter-bloc” для реализации авторизации пользователя вместе с методами для загрузки и обработки данных.

Данная библиотека реализует шаблон BLoC – это акроним от словосочетания «Business Logic Component» - это архитектурный паттерн для разработки мобильных и веб-приложений, который разделяет приложение на три основных компонента: слои представления, бизнес-логики и данных.

Слой представления отвечает за отображение данных пользователю и взаимодействие с ним. Он содержит пользовательский интерфейс и обрабатывает пользовательские действия, такие как клики на кнопки или ввод текста.

Слой бизнес-логики содержит код приложения, который обрабатывает данные, полученные от уровня представления, и взаимодействует с уровнем данных для получения и сохранения данных. Слой бизнес-логики должен быть независим от Presentation Layer и Data Layer, чтобы обеспечить гибкость и возможность переиспользования кода.

Слой данных отвечает за хранение и получение данных в приложении. Он может использовать различные источники данных, такие как базы данных, файлы или внешние API.

BLOC архитектура позволяет разделить приложение на независимые компоненты, что упрощает разработку, тестирование и поддержку приложения. Она также обеспечивает гибкость и возможность переиспользования кода, что позволяет быстро адаптировать приложение к изменяющимся требованиям.

Последовательность работы BLoC компонентов имеет следующие шаги:

* Пользователь нажимает на элемент интерфейса, который запускает событие, необходимое для обработки.
* BLoC компонент определяет событие и выполняет необходимое для данного события заданную бизнес-логику.
* После завершения выполнения бизнес-логики BLoC генерирует состояние, которое реализуется в пользовательском интерфейсе.

На листинге 3.1 указан код, который инициализирует состояния и Bloc-компонент.

Widget build(BuildContext context) {  
 return BlocProvider(  
 create: (context) => LoginBloc(authAPI: AuthAPI()),  
 child: BlocBuilder<LoginBloc, LoginState>(  
 builder: (context, state) {  
 return LoginView();  
 },  
 ),  
 );  
}

Листинг 3.1. Инициализация BLoC компонента авторизации

Данный код получает экземпляр класса AuthAPI, который содержит в себе метод для отправки данных на сервер с логином и паролем пользователя.

BLOC компонент имеет 4 состояния: LoginInitial, LoginInProgress, LoginSuccess и LoginFailure. Они соответственно отображают следующие состояния:

* LoginInitial – начальное состояние, которое включается при старте приложения.
* LoginInProgress – состояние, когда пользователь вводит или ввел свои данные и нажал кнопку «войти».
* LoginSuccess – состояние, когда пользователь успешно вошел в систему
* LoginFailure – пользователю не удалось войти в систему

 LoginBloc – это класс, в котором реализуется бизнес-логика входа в приложение. При нажатии на кнопку «Login» вызывается метод \_handleLoginWithEmailAndPasswordEvent, который в свою очередь вызывает метод login в AuthAPI и принимает токен, идентифицирующий пользователя в данной сессии при успешном входе. При неудачном входе в нижней части экрана появляется предупреждение об ошибке. На рисунке 3.2 отображен экран приложения с предупреждением о неудачном входе в систему умного дома.

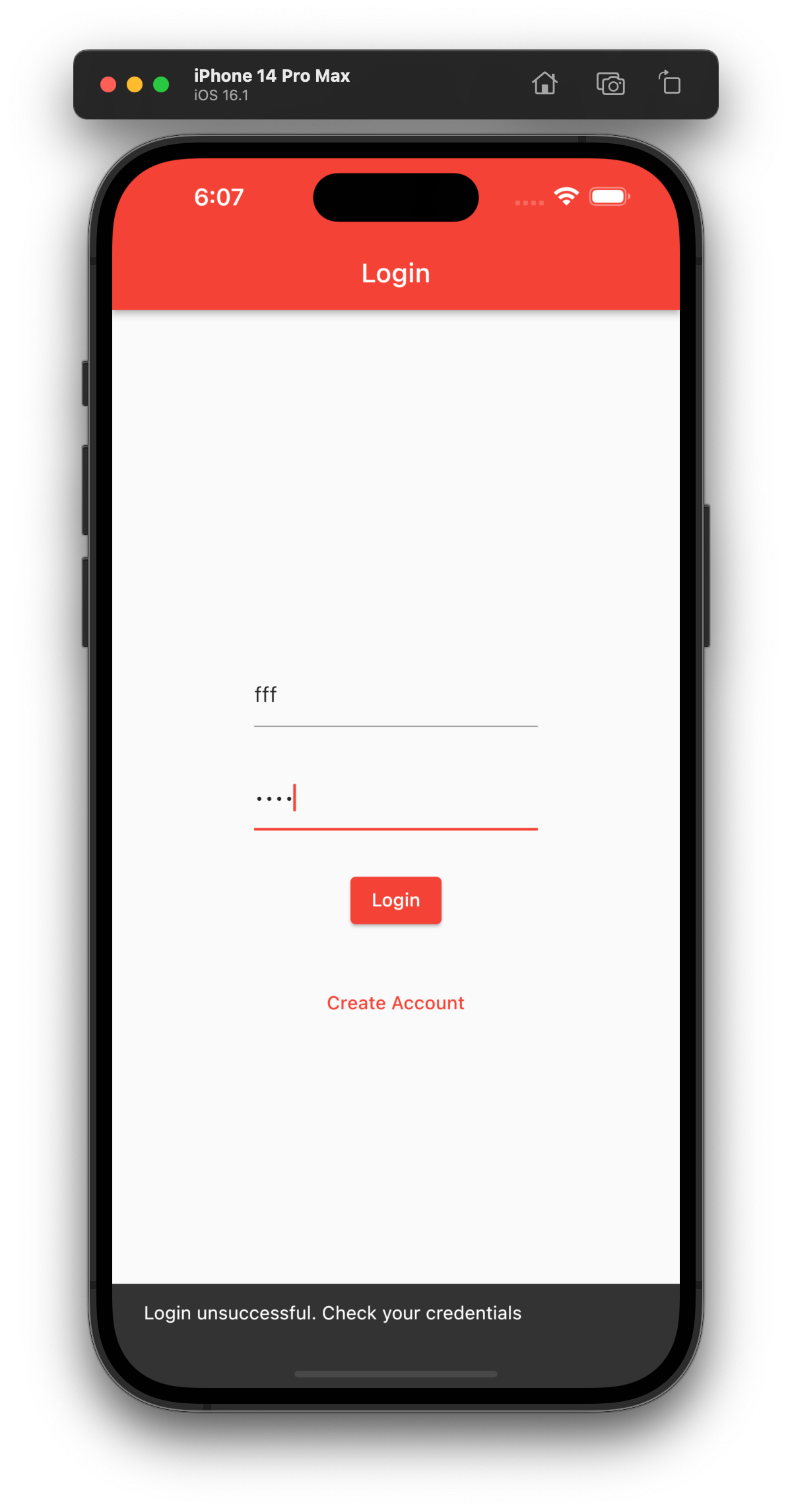


Рисунок 3.2 Уведомление о неудачном входе в приложение

После успешного входа пользователю открывается экран со всеми подключенными устройствами. Данный экран использует виджет FutureBuillder, отображенный на листинге 3.2

Widget build(BuildContext context) {

**return** FutureBuilder(

**future:** \_getListOfDevices(),

**builder:** (context, projectSnap) {

List<Widget> children = [];

**if** (!projectSnap.hasData) {

children = <Widget>[**const** Center(**child:** CircularProgressIndicator())];

}

**if** (projectSnap.hasData) {

children = <Widget>[

ListView.builder(

**itemCount:** projectSnap.data?.length,

**scrollDirection:** Axis.vertical,

**shrinkWrap:** **true**,

**itemBuilder:** (context, index) {

**return** DeviceListItem(projectSnap.data![index]);

},

)

];

}

**return** Column(

**children:** children,

);

},

);

}

Листинг 3.2

FutureBuilder – это виджет, который зависит от асинхронных методов загрузки данных и возвращает виджеты в зависимости от статуса и наличия этих данных. В листинге указано, что FutureBuilder принимает метод \_getListOfDevices в качестве асинхронного метода, который в свою очередь возвращает данные от getListOfDevices в классе AuthAPI, отправляющий запрос на сервер для получения списка всех доступных пользователю устройств. При наличии данных FutureBuilder через поле projectSnap передает их в виджет ListView. Виджет ListView позволяет построить листаемый на экране пользователя массив виджетов.

Для управления каждым отдельным устройством был создан класс DeviceListItem, который принимает в себя объект устройства умного дома и отображает его виджет на главном экране. Это позволяет идентифицировать каждое отдельное устройство, так как виджет ListView лишь занимается созданием списка отображения. Экран приложения, отображающий список всех устройств, приведен на рисунке 3.3.

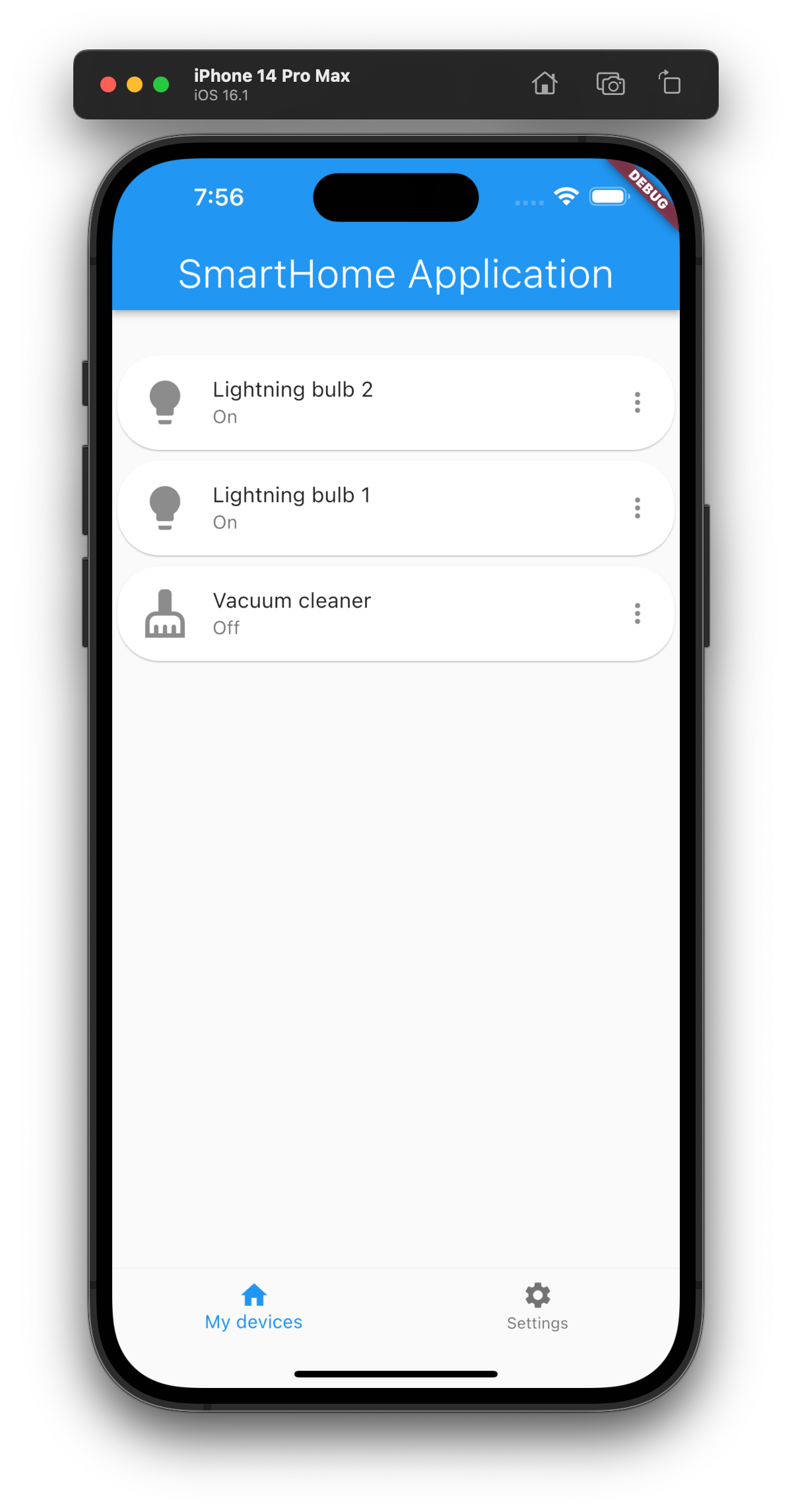


Рисунок 3.3. Главный экран приложения

Класс DeviceListItem содержит в себе методы для корректного отображения имени устройства, его типа и состояния. Для этого используется несколько виджетов: Card и ListTile.

Виджет ListTile содержит в себе несколько важных методов: onTap() и onLongPress(). Первый позволяет при нажатии на виджет запустить определенный метод. В мобильном приложении при нажатии на виджет устройства реализовано включение или выключение умного устройства. Метод onTap() запускает выполнение функции switchStateOfDevice() в классе AuthAPI и передает ему уникальный id устройства вместе с параметром isOn, отвечающий за состояние новое устройства. Код метода отображен на листинге 3.3

**onTap:** () {

AuthAPI.switchStateOfDevice(device.deviceId, !isOn);

setState(() {

widget.device.details["isOn"] = (!isOn).toString();

isOn = !isOn;

isOnString;

});

},

Листинг 3.3. Функция для отправки команды включения или выключения умного устройства.

При долгом нажатии на виджет ListTile реализуется метод onLongPress(). Данный метод в зависимости от типа устройства переводит на страницу с детальной информацией об устройстве. Реализовано данное переключение между экранами через статический метод Navigator.push().

Экран настроек содержит в себе несколько пунктов:

* Добавить устройство
* Добавить хаб
* Удалить устройство
* Удалить хаб

Внешний вид экрана настроек приведен на рисунке 3.4.



Рисунок 3.4. Внешний вид экрана настроек

Эти виджеты отображаются при помощи метода returnCardSettings(), созданный как шаблон для удобного создания параметров. В нем указывается, какая иконка будет отображаться слева от параметра, само название параметра и по нажатию на виджет будет совершен переход в желаемую настройку. В листинге 3.4 отображен метод для создания карточки настройки.

Card returnSettingCard(**String** settingName, StatefulWidget widget) {

**return** Card(

**shape:** **const** RoundedRectangleBorder(

**borderRadius:** BorderRadius.all(Radius.circular(**32**))),

**child:** ListTile(

**leading:** **const** FlutterLogo(**size:** **56.0**),

**title:** Text(settingName),

**trailing:** **const** Icon(Icons.add),

**onTap:** () {

Navigator.push(context,

MaterialPageRoute<Widget>(**builder:** (BuildContext context) {

**return** widget;

}));

},

),

);

}

Листинг 3.4. Создание карточки настроек

Рассмотрим создание каждой настройки.

Добавление нового устройства в систему умного дома происходит путем нажатия пользователем в настройках пункта «Add new device». На данном экране представлено несколько полей, которые пользователь должен заполнить, чтобы добавить устройство в свой умный дом. Экран с добавлением устройства представлен на рисунке 3.5.

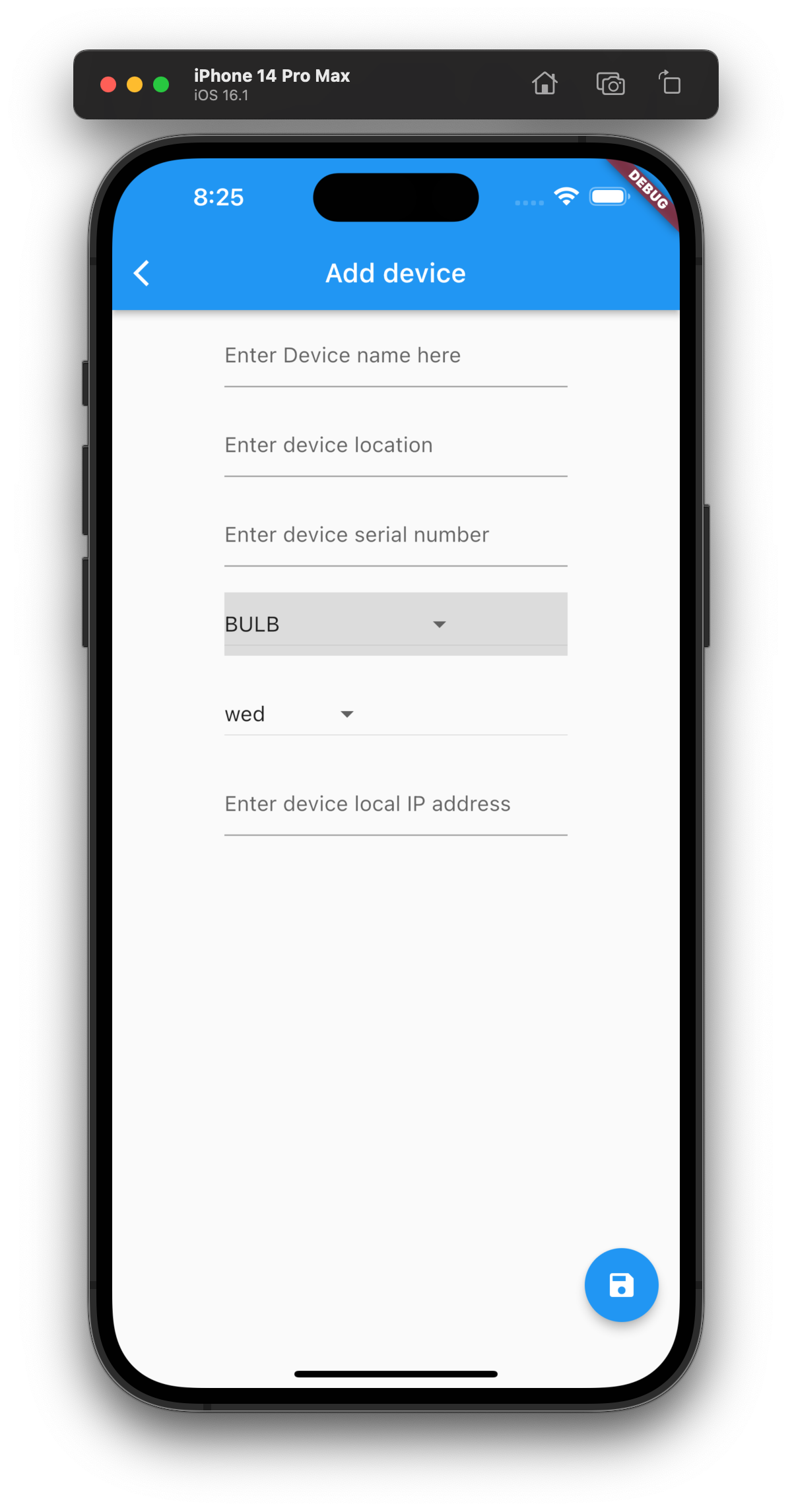


Рисунок 3.5. Экран добавления умного устройства

На данном экране предлагается ввести имя устройства, его местоположение, серийный номер, тип устройства, хаб, к которому устройство будет привязано и IP-адрес устройства в сети. Данные настройки вводятся при помощи виджетов TextField.

Для работы виджетов TextField необходим контроллер ввода текста TextEditingController. Он позволяет отслеживать набираемый пользователем текст и, когда пользователь хочет сохранить данные, данный контролер текста передает пользовательский текст в методы сохранения нового устройства.

Помимо текста, на данном экране присутствуют два выпадающих меню: для выбора типа устройства и хаба, к которому привязывается устройство. Для реализации использовались два класса: DropdownButton и DropdownMenuItem. Первый класс позволяет создать само выпадающее меню, настроить функции при нажатии на него, выбрать стили. Второй класс, DropdownMenuItem, можно назвать оберткой для данных, которые пользователь видит на экране. Класс DropdownMenuItem необходим для корректного отображения списка опций и содержит в себе два поля: value и child. В поле value передается один из выбираемых объектов, а в child – класс, который будет отвечать за корректное отображение на экране этого объекта. В данной работе в value передается тип возможного девайса как объект типа String, а в child устанавливается этот же объект, но в обертке Text() для корректного отображения.

Такой же подход используется и для выпадающего меню с выбором хаба, к которому будет привязано новое устройство. Класс DropdownButton отвечает за создание самого виджета выпадающего меню, а класс DropdownMenuItem – за каждый объект, который можно выбрать. В случае с выбором хаба, у класса DropdownMenuItem в поле value устанавливается объект типа HubDTO, который инкапсулирует в себе один из хабов, привязанных к пользователю, а в поле child – виджет Text, который отображает в меню выбора хаба название привязываемого хаба.

Чтобы иметь возможность выбрать тип устройства и хаб, к которому привязывается устройство, необходимо заранее получить список возможных для выбора вариантов. Чтобы реализовать данное требование, в классе NewDevice, который полностью отвечает за экран создания нового устройства, используется виджет FutureBuilder, который отвечает за построение виджетов в зависимости от асинхронного метода загрузки данных, то есть при статусе загрузки данных будут отображаться различные виджеты. Данный виджет уже был упомянут в данной работе, он используется в данном классе схожим образом.

Асинхронный метод \_getListOfHubs(), передаваемый в поле future класса FutureBuilder, при помощи статических методов getListOfHubs() и getDeviceTypes() класса AuthAPI запрашивает у сервера данные о хабах пользователя и типах девайсах, которые возможны в системе умного дома. После получения данных, они сохраняются в полях listOfHubs и listOfDeviceTypes, представляющие из себя массивы. После получения данных, поле snapshot.hasData становится true, что говорит о полном получении данных с сервера. Виджет перестраивается, и массивы с выбираемыми типами устройств и хабов преобразовываются в необходимые для работы с DropdownButton классы DropdownMenuItem. Таким образом реализован выбор хаба и типа устройства.

Сохранение данных происходит при нажатии пользователем на кнопку сохранения в правом нижнем углу экрана. Этот виджет реализован при помощи класса FloatingActionButton. Данный класс передается в поле floatingActionButton виджета Scaffold в классе NewDevice. В нем возможно настроить отображаемую иконку путем передачи класса Icon в поле child, а также настроить метод onPressed(), который будет запускаться при нажатии на кнопку сохранения данных.

Сохранение данных происходит в методе submitData(), который запускается после нажатия на кнопку сохранения. В данном методе вся информация забирается из полей, отвечающих за каждую характеристику умного устройства, и отправляются на сервер через метод addNewDevice() класса AuthAPI.

Таким образом реализован экран добавления нового умного устройства.

Следующим экраном настроек является экран добавления хаба. В мобильном приложении он именуется «Add new Hub».

За создание виджетов при добавлении хаба на экране пользователя отвечает класс NewHub. Пример экрана добавления хаба проиллюстрирован на рисунке 3.6.

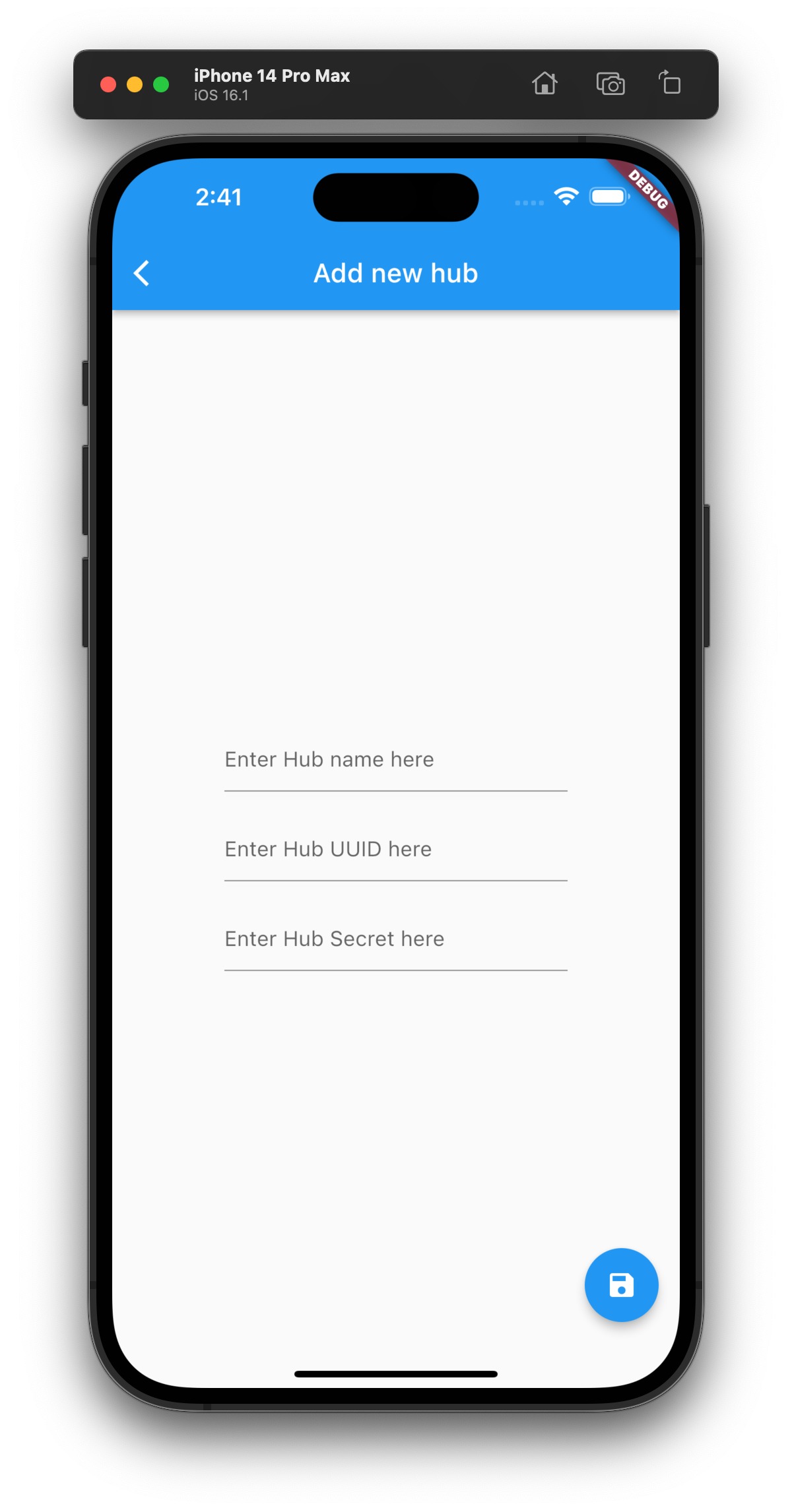


Рисунок 3.6. Экран добавления нового устройства

На данном экране используется несколько важных виджетов: FloatingActionButton и TextField. Первый виджет, аналогично предыдущему экрану добавления устройства, создает кнопку сохранения данных в правом нижнем углу и добавляет метод onTap(), который запускает необходимые действия по нажатии на кнопку. Второй виджет, TextField, отвечает за набранный пользователем текст. Таких виджетов представлено три: для ввода имени хаба, уникального UUID хаба и секретного ключа. Их использование я освещу далее в этой работе.

При нажатии на кнопку сохранения запускается метод submitData(), который собирает введённый пользователем текст из контроллеров набранного текста и передает функции addHub() класса AuthAPI для отправки на сервер.

Третьим экраном настроек является экран удаления устройства. Его внешний вид приведен на рисунке 3.7.

На данном экране пользователь может удалить устройство умного дома, выбрав его и нажав на кнопку удаления в правом нижнем углу. Данное действие реализовано при помощи виджетов FutureBuilder, DropdownMenu и FloatingActionButton.

В классе DeleteDevice, в котором используются вышеперечисленные виджеты, FutureBuilder, аналогично другим экранам, получает в поле future асинхронный метод \_getDevicesList(), который в свою очередь вызывает метод getListOfDevices класса AuthAPI. При получении данных отображается виджет DropdownButton, в котором массив устройств, инкапсулированных в классе Device, преобразуется в массив DropdownMenuItem, необходимым для корректной работы виджета DropdownButton. При выборе устройства объект класса Device записывается в переменную selectedDevice.

При нажатии на кнопку удаления в правом нижнем углу метод onTap  
(), расположенный в виджете FloatingActionButton, вызывает метод submitData(), который передает методу deleteDevice в классе AuthAPI уникальный id выбранного для удаления устройства умного дома.

После выполнения данных действий устройство удаляется и перестает отображаться на главном экране приложения и в списке устройств, доступных для удаления.



Рисунок 3.7. Экран удаления устройства

Четвертым по счету экраном настроек умного дома является удаление хаба, в настройках он отображается как «Delete Hub». На рисунке 3.8 приведен пример данного экрана.

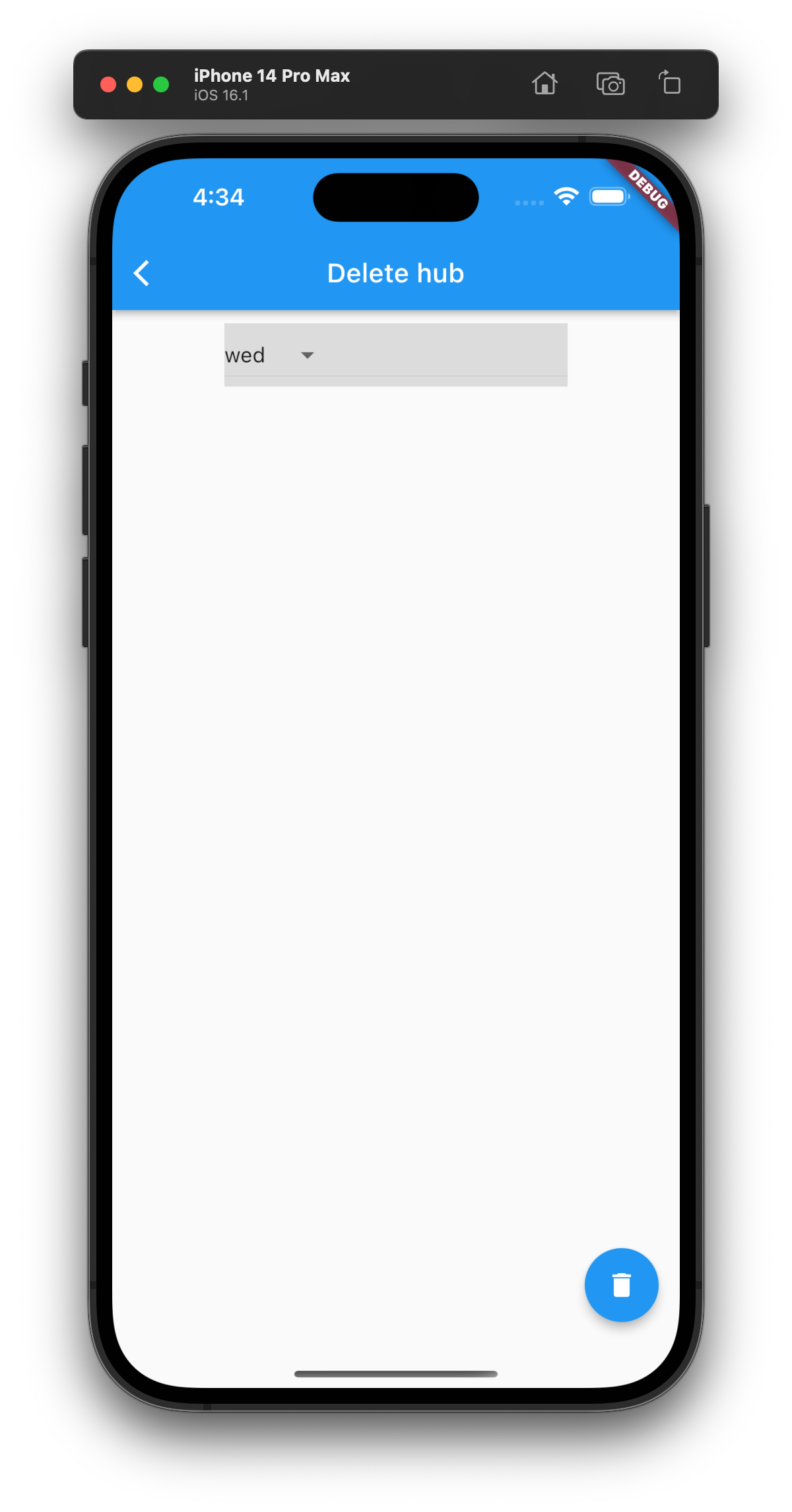
****

Рисунок 3.8. Экран удаления хаба

На данном экране пользователь может удалить хаб, привязанный к его аккаунту. Данное действие реализовано виджетами FutureBuilder, FloatingActionButton и DropdownButton.

В классе DeleteHub, в котором используются вышеперечисленные виджеты, FutureBuilder, как и на других экранах, получает в поле future асинхронный метод \_getListOfHubs(), который в свою очередь вызывает метод getListOfHubs класса AuthAPI. При получении данных отображается виджет DropdownButton, в котором массив хабов, инкапсулированных в классе HubDTO, преобразуется в массив DropdownMenuItem, необходимым для корректной работы виджета DropdownButton. При выборе хаба объект класса HubDTO записывается в переменную selectedHub.

При нажатии на кнопку удаления в правом нижнем углу метод onTap  
(), расположенный в виджете FloatingActionButton, вызывает метод submitData(), который передает методу deleteHub в классе AuthAPI уникальный UUID выбранного для удаления хаба умного дома.

После выполнения данных действий хаб удаляется и перестает отображаться на главном экране приложения и в списке устройств, доступных для удаления.

Таким образом, было реализовано мобильное приложение для управления умным домом и описано, как каждый компонент системы взаимодействует с пользователем.

3.2 Разработка серверного приложения

В предыдущей главе было установлено, что серверная часть системы умного дома будет работать на языке программирования Java с использованием фреймворка Spring. Все данные будут сохраняться в СУБД PostgreSQL, а для связи между хабами будет использоваться WebSocket.

В данной части работы будут рассмотрены вопросы работы системы, ее безопасность, принципы хранения данных.

Любой запрос к системе должен быть авторизован. Для этого необходимо определить, кто отправляет запрос и имеет ли он разрешения на совершение запрашиваемых действий. Для этого был настроен фреймворк Spring Security, который отвечает за безопасность системы.

Его настройка происходит путем создания бинов (это объекты и методы, которые управляются фреймворком Spring) в классах, помеченных аннотацией @Configuration. Пример конфигурации приведен на листинге 3.5.

**@Bean**

**public** SecurityFilterChain **filterChain**(HttpSecurity http) **throws** Exception {

http.cors().and().csrf().disable()

.sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.authorizeRequests()

.antMatchers("/api/auth/\*\*").anonymous()

.antMatchers("/api/test/\*\*").anonymous()

.antMatchers("/hello/\*").authenticated()

.antMatchers("/resp/\*").authenticated()

.anyRequest().authenticated();

http.addFilterAfter(authTokenFilter, UsernamePasswordAuthenticationFilter.class);

**return** http.build();

}

Листинг 3.5. Настройка бина безопасности

В листинге 3.5 указывается, что для эндпоинтов авторизации не требуется токенов или других подтверждения авторизации, что логично, так как именно через них пользователь может авторизоваться. Все остальные эндпоинты требуют авторизации, и без ключа или токена ими не получится воспользоваться.

Помимо настроек безопасности эндпоинтов, в листинге указан метод addFilterAfter, в которй передается объект authTokenFilter. Данный объект класса AuthTokenFilter позволяет определить пользователя по токену, который был передан в заголовке запроса. Данный класс занимается расшифровкой токенов, проверкой логина и пароля, который был зашифрован, а также проверяет, является ли токен просроченным или валидным. Если все условия проверки были пройдены, то Spring Security допускает запрос к дальнейшей работе. Если, например, у токена истек срок годности, то Spring запретит продолжение работы и вернет 403 код ответа, что означает «Доступ запрещен».

Создание токена происходит в классе JWTUtils в методе createJWT. Данный метод является перегруженным, что означает, что методы с одним названием принимают разные параметры. Перегрузка метода нужна для ссоздания токена для хаба и пользоваетелей.

Оба метода работают по одному принципу: для создания токена требуется время срока действия токена, ключ шифрования и два поля “id” и “sub”, отвечающих за логин и пароль соответственно. В данной системе токен действует один час, после чего его нужно получить заново, отправив новый запрос на эндпоинт авторизации.

Сам токен является строчкой, разделенной на три части через точки. Первая часть токена является заголовком, вторая – зашифрованные данные, третья – подписью.

В заголовке содержится информация о типе токена и алгоритме шифрования, в системе умного дома используется тип JSON Web Token и шифрование HMAC-SHA256.

Зашифрованными данными являются поля id и sub, которые являются логином и паролем пользователя. Список полей может быть производным.

В третьей части, подписи, используются 2 предыдущих части токена, которые кодируются при помощи преобразования в строку типа base64url и «складываются» с секретным ключом, определенным в файле application.yaml проекта умного дома, при помощи алгоритма HMAC-SHA256. Таким образом, подпись позволяет проверить, были ли модифицированы данные заголовка и блока данных.

После успешной аутентификации можно вызывать заранее определенные эндпоинты. В случае с системой умного дома, эндпоинты разбиты по группам обязанностей такими как: эндпоинт авторизации, эндпоинт для управления умными устройствами, эндпоинт регистрации и эндпоинт управления хабами. Рассмотрим их подробнее.

Эндпоинт авторизации является классом AuthController, помеченный аннтоациями @RestController, @RequestMapping(“/api/auth”) и @RequiredArgsConstructor. Эти аннотации необходимы для определения Spring фреймворком, какой метод запустить при вызове эндпоинта. В классе AuthController написано два метода, login() и loginHub(), отвечающие за авторизацию пользователя и хаба, который хочет соединиться с сервером. Процесс работы у двух методов похожий: сервер принимает JSON-объекты, которые преобразовываются Spring фреймворком при помощи десериализации в Java-объекты: поля из JSON преобразовываются в поля Java-объектов. Эти объекты передаются в класс AuthServiceImpl, который в свою очередь передает данные на проверку в методах класса AuthProvider. Если проверка прошла успешно, то есть логин и пароль совпали с данными из базы данных, то на основе данных пользователя формируется токен. Данный токен передается в класс AuthenticatedUserDTO или AuthenticatedHubDTO, в зависимости от клиента, вызвавшего метод авторизации, и отправляется как ответ с кодом 200.

Класс для управления умными девайсами DeviceController обширней: в нем находятся методы, позволяющие:

* Получить список всех устройств, которые привязаны к аккаунту пользователя.
* Включать или выключать устройство.
* Получить статус устройства.
* Получить информацию об устройстве.
* Сменить цвет устройства (в случае с Wi-FI лампами).
* Добавить новое устройство.
* Удалить устройство.
* Получить список всех доступных типов устройств.

Все методы, определенные в этом классе, имеют аннотацию @RequestMapping вместе с уникальным URI, позволяющую привязать вызываемую «ссылку» к определенному методу. Также в этой аннотации обозначены типы запросов, которые требуются для данного метода. Например, чтобы получить список всех устройств, необходимо отправить GET-запрос на адрес сервера вместе с URI «/api/devices/list».

Говоря о запросах, нужно отметить, что POST, GET, PUT и DELETE – запросы могут возвращать ответы в виде объектов, однако использование отдельных методов позволяет лучше понять API системы. Например, тип запроса означает «получить» или «обновить», а URI – та информация, которую мы передаем или желаем получить. Для каждого действия должен быть свой эндпоинт со своим типом запроса – это принцип REST-архитектуры.

В классе эндпоинтов для управления умными устройствами инкапсулирован объект service класса DeviceServiceImpl. Он отвечает за бизнес-логику при вызове эндпоинтов: все данные, принятые в эндпоинтах, переходят в сервис.

Данный сервис позволяет выполнять логику, которую запрашивает пользователь. В нем описаны методы взаимодействия с базой данных и веб-сокетами, подключенными в данный момент к серверу.

Например, если пользователь отправит PUT-запрос по URI «/api/devices/{deviceId}/state» вместе с уникальным идентификатором умного устройства и булевой переменной isON, отвечающей за включение или выключение устройства, то в методе switchDeviceState() класса DeviceServiceImpl будет произведены следующие операции:

1. Из базы данных будет получена сущность умного устройства в виде объекта DeviceEntity. В нем содержится вся информация об устройстве, например его тип, IP-адрес, название и другое.
2. В этой сущности будет изменен параметр статистики, в котором сохраняется новое значение isOn в виде true или false.
3. Составляется объект, который будет отправлен по веб-сокету на хаб
4. Этот объект отправляется на хаб.

В данном классе описана бизнес-логика для следующий действий:

* Получить список всех устройств пользователя
* Удалить устройство
* Включить или выключит устройство
* Добавить устройство
* Получить информацию об одном устройстве
* Поменять цвет лампы

Здесь важно упомянуть, какие данные и с какими характеристиками сохраняются в базе данных.

В базе данных хранится несколько стержневых сущностей: пользователь, хаб и устройство. Рассмотрю их подробней.

В таблице пользователь, названной «users», хранятся имя и фамилия пользователя, логин и пароль от его аккаунта и уникальный id пользователя. Данная таблица была инициализирована при помощи SQL-кода, отображенном в листинге 3.6.

**CREATE** **TABLE** users (

user\_id serial **NOT** **NULL**,

user\_name text **NOT** **NULL**,

user\_surname text **NOT** **NULL**,

user\_login text **NOT** **NULL**,

user\_password text **NOT** **NULL**,

**PRIMARY** **KEY** (user\_id)

);

Листинг 3.6. Инициализация таблицы users.

Таблица устройств, именованная в базе данных «devices», хранит в себе много важной информации:

* device\_id – уникальный id отдельно взятого умного устройства
* device\_name – имя устройства
* device\_location – местоположение устройства
* device\_serial – серийный номер устройства
* local\_ip\_address – локальный IP-адрес устройства
* device\_owner – ссылка на владельца устройства
* hub\_id – ссылка на привязанный хаб
* device\_type – тип устройства
* json\_properties – тип данных json, который содержит в себе информацию о состоянии устройства

Поле device\_type заполняется текстовым значением из enum-класса DeviceType, который отвечает за поддержку выбора типа устройства. В данной системе пока реализован тип устройства «лампа», отмеченная как «BULB».

Третьей таблицей является таблица хабов, названная «hubs». В ней содержатся следующие поля:

* hub\_id – уникальный id отдельно взятого хаба
* hub\_owner – ссылка на таблицу «users» для определения владельца хаба
* hub\_name – название хаба
* hub\_uuid – уникальный uuid для верификации хаба
* hub\_secret – секретный ключ хаба

SQL-код для создания таблицы приведен в листинге 3.7.

**CREATE** **TABLE** hubs (

hub\_id serial **NOT** **NULL**,

hub\_owner serial **NOT** **NULL**,

hub\_name text **NOT** **NULL**,

hub\_uuid text **NOT** **NULL**,

hub\_secret text **NOT** **NULL**,

**PRIMARY** **KEY** (hub\_id),

**CONSTRAINT** hubs\_hub\_owner\_users\_user\_id\_foreign **FOREIGN** **KEY** (hub\_owner) **REFERENCES** users (user\_id)

);

Листинг 3.7. SQL-код создания таблицы хабов.

Для связи базы данных с серверным приложением в данном проекте используется фреймворк Spring Data JPA и Hibernate 6. Он позволяет при помощи Java-объектов выполнять CRUD запросы к базе данных и получать ответы. Для этого в коде реализованы Java-объекты сущностей базы данных.

Например, для

ГЛАВА 4. ВОЗМОЖНОСТИ РАСШИРЕНИЯ УМНОГО ДОМА

ГЛАВА 5. ТЕХНИКО-ЭКОНОМИЧЕСКИЙ РАСЧЕТ

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Push-Button Manor | Popular Mechanics, стр.85 [Электронный ресурс] / <https://books.google.ru> [сайт]. URL: https://books.google.ru/books?id=jNgDAAAAMBAJ&printsec=frontcover&hl=ru#v=twopage&q&f=false (Дата обращения 27.04.2023)
2. Российский рынок умного дома в 2022 году. Продажи, игроки, перспективы | Эльдар Муртазин, [Электронный ресурс] / <https://mobile-review.com> [сайт]. URL: <https://mobile-review.com/all/articles/analytics/rossijskij-rynok-umnogo-doma-v-2022-godu-prodazhi-igroki-perspektivy/> (Дата обращения 01.05.2023)