

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программные средства интеллектуального
управления умным домом**

Студент	_____	Басманова Анастасия Андреевна
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Иванчуков А.Г.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МОиПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«__» _____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Басманова Анастасия Андреевна гр. 19ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства интеллектуального управления умным домом

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ»

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: облегчение процесса управления работы умного дома.

Основные функции системы:

1. возможность регистрации и авторизации пользователей;
2. добавление, изменение и удаление устройств;
3. создание, изменение и удаление комнат;
4. создание, редактирование и удаление сценариев;
5. запуск и остановка сценариев;
6. предложение пользователю выполнения действия.

Технология разработки: объектно-ориентированное программирование.

Язык программирования: Dart, Python.

Среда программирования: Visual Studio Code.

5. Перечень подлежащих разработке вопросов:

1. анализ предметной области и постановка задачи;
2. проектирование программных средств;
3. реализация программных средств;
4. тестирование программных средств;
5. оформление ВКР.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	10%	12.04.2023 – 17.04.2023	
2	Проектирование программных средств	20%	18.04.2023 – 25.04.2023	
3	Реализация программных средств	30%	26.04.2023 – 09.05.2023	
4	Тестирование программных средств	20%	10.05.2023 – 17.05.2023	
5	Оформление ВКР	20%	18.05.2023 – 31.05.2023	

Дата выдачи задания " _____ " _____ 20 _____

Руководитель бакалаврской работы

(подпись, дата)

Задание к исполнению принял студент

(подпись, дата)

Работу к защите допустить

Декан факультета

(подпись, дата)

Реферат

Пояснительная записка содержит 99 листов, 29 рисунков, 42 таблицы, 4 листинга, 27 использованных источников и 3 приложения.

ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА, МАШИННОЕ ОБУЧЕНИЕ, УМНЫЙ ДОМ, МИКРОСЕРВИСЫ, ДИАГРАММЫ UML, MICROSOFT VISUAL STUDIO CODE, DART, PYTHON, FLUTTER, FIREBASE.

Предмет работы: разработка программного средства интеллектуального управления умным домом.

Объект работы: облегчение процесса управления работы умного дома.

Цель работы: разработка программного средства интеллектуального управления умным домом.

Технология разработки: Flutter.

Результаты работы: спроектировано и разработано приложение для управления умным домом.

Область применения: пользователи, обладающие умными устройствами.

					ПГУ 09.03.02 – 08БР191.1 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Программные средства интеллектуального управления умным домом	Лит.	Лист	Листов
Разраб.		Басманова А.А.						
Провер.		Иванчуков А.Г.					4	99
Реценз.						группа 19ВИ1		
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и требований.....	8
1.1 Анализ предметной области и постановка задачи.....	8
1.2 Анализ существующих решений.....	9
1.2.1 Программный продукт Mi Home.....	9
1.2.2 Программный продукт SmartThings.....	9
1.2.3 Программный продукт умный дом с Алисой.....	10
1.3 Сравнительный анализ существующих решений.....	11
1.4 Анализ функциональных требований.....	13
1.5 Функционально-стоимостной анализ.....	22
2 Анализ технологий проектирования и разработки.....	25
2.1 Инструментальные средства проектирования.....	25
2.1.1 Язык моделирования UML.....	25
2.1.2 Выбор средств проектирования.....	25
2.2 Инструментальные средства разработки.....	26
2.2.1 Язык программирования Dart.....	26
2.2.2 Платформа Flutter.....	27
2.2.3 Платформа Firebase.....	27
3 Обучение модели машинного обучения.....	29
4 Проектирование программного средства.....	31
4.1 Проектирование клиентской части приложения.....	31
4.2 Проектирование серверной части.....	35
4.3 Проектирование базы данных.....	37
4.4 Проектирование пользовательского интерфейса.....	41
5 Реализация программного средства.....	44
5.1 Реализация клиентской части приложения.....	44
5.2 Реализация серверной части приложения.....	47
5.3 Разработка пользовательского интерфейса.....	48
6 Тестирование функций взаимодействия с устройствами.....	58

6.1	Тестирование экрана добавления устройства чёрным ящиком.....	58
6.2	Тестирование метрики кода.....	62
6.3	Автоматизированное тестирование функций приложения.....	65
	Заключение	68
	Список использованных источников	69
	Приложение А. Глоссарий	72
	Приложение Б. Исходные коды приложения.....	74
	Приложение В. Результаты тестирования	97

Введение

В настоящее время люди любят облегчать себе жизнь, стремятся автоматизировать повседневную рутину. Одной из наиболее популярных сфер автоматизации является умный дом. Умный дом – это система связанных между собой устройств, которые помогают упростить выполнение различных бытовых задач [1]. Такая система позволяет управлять освещением, менять его цвет и яркость, контролировать температуру в комнатах, дистанционно взаимодействовать с умными приборами, такими как: умный чайник, робот – пылесос, телевизор и т. д. Для простого взаимодействия со всеми элементами умного дома используется специальное программное обеспечение. Такое приложение может быть удобно в использовании даже с мобильного устройства, так как находясь далеко от дома, например, выйдя с работы, к своему возвращению можно включить чайник или запустить кондиционер.

Объектом выпускной квалификационной работы является облегчение процесса управления работы умного дома.

Предметом является программное средство интеллектуального управления умным домом.

Цель данной работы заключается в разработке программного средства интеллектуального управления умным домом.

Для достижения поставленной цели были выделены следующие задачи:

- постановка задачи на разработку системы;
- анализ существующих систем;
- определение требований к разрабатываемой системе;
- проектирование программного средства интеллектуального управления умным домом;
- реализация системы;
- тестирование разработанной системы.

В результате выполнения работы было разработано приложение для интеллектуального управления умным домом.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

Предметная область «Умный дом» представляет собой автоматизированную систему управления всеми приборами в доме, которые объединены в единую экосистему. Система может сама принимать решения и выполнять определенные задачи, без участия человека. Человеку остается лишь управлять дистанционно нажатием кнопок на пульте/гаджетах или голосовыми командами.

Человек может задать алгоритм, в соответствии с которым система будет включать или отключать разные приборы, бытовую технику, освещение, регулировать доступ в дом и выполнять другие важные функции. Иными словами, умный дом подразумевает упрощение нагрузки в быту и делает дом удобным и комфортным.

Самые популярные на сегодняшний день устройства для умного дома используют технологии Wi-Fi, Bluetooth, ZigBee и Z-Wave [2]. Каждая технология имеет свои сильные и слабые стороны, и никто не запрещает их совместное использование для компенсации их недостатков. Однако для разных задач и разных типов интеллектуальных устройств используются разные технологии.

Например, бытовая электроника (телевизоры, холодильники, кофеварки) обычно использует Wi-Fi или Bluetooth, которые есть во всех телефонах. Причина в том, что технологию можно использовать без полноценной системы умного дома. Автоматизация освещения и отопление лучше подходит для встроенных модулей ZigBee или Z-Wave, которые специально разработаны для интеграции с существующим оборудованием освещения и отопления. Но для серьезной работы нужен специальный концентратор [3].

Приложение должно позволять управлять умными устройствами с помощью телефона.

1.2 Анализ существующих решений

Перед разработкой приложения необходимо осуществить анализ уже существующих программных средств. Ниже приведены некоторые из них.

1.2.1 Программный продукт Mi Home

Программный продукт Mi Home, разработанный китайской компанией Xiaomi, является бесплатной системой управления умными устройствами. Особенностью данной программы является то, что она имеет готовый сценарий для работы. То есть пользователь может сразу использовать приложение, без особых усилий, и помимо этого у него есть возможность изменить уже существующий шаблон под себя. В программе можно регулировать свет, подачу воды и работу электроприборов [4].

В качестве примера приведена камера видеонаблюдения. Для того чтобы начать работу с устройством, необходимо скачать приложение Mi Home, далее, надо подключить камеру к сети питания. Последним этапом является подключение устройства к приложению, с помощью сети Wi-Fi. После привязки камеры к системе, пользователь получает удобный доступ и может наблюдать за происходящим, используя свой смартфон.

1.2.2 Программный продукт SmartThings

Программный продукт SmartThings, разработан в американской компании – Samsung. Свою систему умного дома для пользователей предоставили в бесплатной версии. Особенностью данной программы является то, что она имеет три абстракции: комнаты, сцены, сценарии. Все устройства, которые есть дома группируются в комнаты. При появлении новых гаджетов, их можно добавить в одну комнату и дать ей название, например, «Гостиная», «Детская», «Кухня». Также есть сцены, ситуации, которые имеют определенные состояния устройств, например, состояние «игра», «работа», «уборка». Последнее, это сценарии – четкие правила автоматизации, по которым работает программа. По стандарту четыре сценария: «Я вернулся»,

«Спокойной ночи», «Пока», «Доброе утро». Пользователь вправе настроить активацию сценария под себя. Например, когда придет время до уборки в квартире, можно поставить определенную музыку и запустить все устройства для уборки квартиры – робот – пылесос, очиститель воздуха [5]. В программе можно создать свою экосистему начиная от настройки освещения, кондиционеров и роботов – пылесосов до контроля изображения с камер.

Примером устройства может быть стиральная машина с функцией сушки. Добавив технику в приложение на смартфоне, удаленно можно запустить стирку белья к возвращению домой.

1.2.3 Программный продукт умный дом с Алисой

Программный продукт умный дом с Алисой был разработан в российской компании Яндекс. Приложение является бесплатным, но для использования некоторых устройств в полной мере необходима платная подписка. Особенностью данной программы является то, что устройствами, которые подключены к системе, можно управлять с помощью умной колонки отдавая голосовые команды, например, «Алиса, включи свет». В программе можно настроить расписание для работы устройств, например, включать увлажнитель перед сном. Можно также создавать команды для всего, то есть запускать несколько устройств одновременно одной фразой, например, «Алиса, я проснулся». По этой команде включится телевизор, на кухне включится чайник, а робот – пылесос начнет уборку [6].

Также, есть возможность подключения приборов разных производителей, главное обратить на это внимание при покупке, по отметке «работает с Алисой». Можно подключить такие устройства как: колонка, лампочка, телевизор, увлажнитель воздуха, кондиционер, а также различные датчики на открытие, протечки, движения.

1.3 Сравнительный анализ существующих решений

Проанализировав приведенные аналоги, были поставлены критерии, по которым было проведено сравнение аналогов: управление устройствами, комнатами, сценариями, голосом, изменение внешнего вида приложения.

Критерий «Управление устройствами» показывает насколько удобно создавать и взаимодействовать с устройствами. Оценка критерия от 0 до 4. Расшифровка оценок критерия представлена в таблице 1.

Таблица 1 – Расшифровка оценок критерия «Управление устройствами»

Оценка	Описание оценки
0	Нельзя добавлять и управлять
1	Можно добавлять устройства, однако процесс добавления совершенно не понятен. Можно управлять базовыми функциями устройств
2	Можно добавлять устройства, однако процесс добавления сложный, нужны дополнительные инструкции. Можно управлять базовыми функциями устройств
3	Можно добавлять устройства, процесс добавления понятный. Можно управлять базовыми функциями устройств
4	Можно добавлять устройства, процесс добавления понятный. Можно управлять всеми функциями устройств

Критерий «Управление комнатами» показывает насколько удобно создавать комнаты и перемещать устройства между ними. Оценка критерия от 0 до 5. Расшифровка оценок критерия представлена в таблице 2.

Таблица 2 – Расшифровка оценок критерия «Управление комнатами»

Оценка	Описание оценки
0	Нельзя создавать комнаты
1	Можно создавать комнаты, но нельзя перемещать между ними устройства
2	Можно создавать комнаты и перемещать между ними устройства
3	Можно создавать комнаты и перемещать между ними устройства, интерфейс совершенно неудобен
4	Можно создавать комнаты и перемещать между ними устройства, интерфейс в целом удобен, но есть затруднения в пользовании
5	Можно создавать комнаты и перемещать между ними устройства, интерфейс понятен и удобен

Критерий «Управление сценариями» показывает насколько удобно создавать сценарии, насколько удобны условия запуска сценариев. Оценка критерия от 0 до 5. Расшифровка оценок критерия представлена в таблице 3.

Таблица 3 – Расшифровка оценок критерия «Управление сценариями»

Оценка	Описание оценки
0	Нельзя создавать сценарии
1	Можно создавать сценарии, но нет возможностей добавить условия запуска
2	Можно создавать сценарии, есть возможность добавления условий для запуска. Сценарии могут управлять только устройствами
3	Можно создавать сценарии, есть возможность добавления условий для запуска. Сценарии могут управлять устройствами и присылать уведомления
4	Можно создавать сценарии, есть возможность добавления условий для запуска. Сценарии могут управлять устройствами, присылать уведомления, запускать и останавливать другие сценарии
5	Можно создавать сценарии, есть возможность добавления условий для запуска. Сценарии могут управлять устройствами, присылать уведомления, запускать и останавливать другие сценарии, а также отслеживать местоположение пользователя

Критерий «Управление голосом» показывает насколько удобно управлять устройствами голосом. Оценка критерия от 0 до 2. Расшифровка оценок критерия представлена в таблице 4.

Таблица 4 – Расшифровка оценок критерия «Управление голосом»

Оценка	Описание оценки
0	Нельзя управлять голосом
1	Управлять голосом можно, но для этого нужно нажать кнопку
2	Можно управлять голосом, не прикасаясь к телефону

Критерий «Изменение внешнего вида» демонстрирует возможность изменить тему в приложении. Оценка критерия от 0 до 1. Расшифровка оценок критерия представлена в таблице 5.

Таблица 5 – Расшифровка оценок критерия «Изменение внешнего вида»

Оценка	Описание оценки
0	Нельзя изменить внешний вид
1	Есть возможность смены темы

Результаты сравнения программ-аналогов по представленным выше критериям приведены в таблице 6.

Таблица 6 – Сравнительная характеристика программных продуктов

Функционал	Mi Home	Smart Things	Умный дом с Алисой
Управление устройствами	3	4	4
Управление комнатами	4	4	5
Управление сценариями	4	5	4
Управление голосом	1	1	2
Изменение внешнего вида приложения	1	0	1

На основе сравнения аналогов и их критериев были выделены основные функции для разработки системы умного дома.

Проанализировав функциональные возможности рассмотренных приложений, можно сделать вывод, что ни одно из приложений, не имеет функции анализа и прогнозирования действий пользователя. Такая функция позволяет разрабатываемой системе получить конкурентные преимущества на рынке существующих решений.

1.4 Анализ функциональных требований

Система интеллектуального управления умным домом – это система, которая позволяет управлять устройствами в доме с помощью телефона. С помощью такой системы можно настроить работу таких устройств как телевизор, кондиционер, а также освещение в доме. Помимо этого, есть возможность создания сценариев – автоматических действий по расписанию, созданных пользователем. Сценарии могут содержать два работы: действие и ожидание. Действие подразумевает выполнение устройством определённых функций. Ожиданием является приостановка выполнения сценария на заданное время.

В приложении должна быть предусмотрена регистрация и авторизация, для поддержки большого количества пользователей. В случае, если человек забыл свой пароль от аккаунта, у него есть возможность восстановить пароль через электронную почту.

Для удобства управления устройствами, необходимо разделение на комнаты, с возможностью добавления, редактирования и удаления. Удобное разделение на комнаты, позволит пользователю не путаться в устройствах.

Особенностью разрабатываемого приложения будет являться то, что на основе ежедневных действий пользователя будет выполняться прогнозирование этих же действий, которые в будущем будут предложены пользователю для выполнения.

Данная функция основана на машинном обучении, то есть программа самостоятельно обучается, от пользователя всего лишь поступают данные.

Благодаря такой особенности управление умным домом становится более удобным за счет сокращения действий пользователя. В дополнение не требуется покупать различные датчики и концентраторы. Человек меньше тратит денежные средства на обустройство своего дома, а функционал продолжает облегчать жизнь.

В соответствии с проведенным анализом функциональных требований была составлена диаграмма вариантов использования.

Диаграмма вариантов использования – это графический инструмент для изображения функциональных требований к системе, отношения между акторами и вариантами использования (прецедентами). Диаграмма представляет графическое отображение акторов и вариантов использования системы. Актор – это внешняя сущность, которая взаимосвязана с системой. Варианты использования представляют собой эллипсы, которые отображают возможный функционал системы [7].

Построенная диаграмма вариантов использования представлена на рисунке 1.

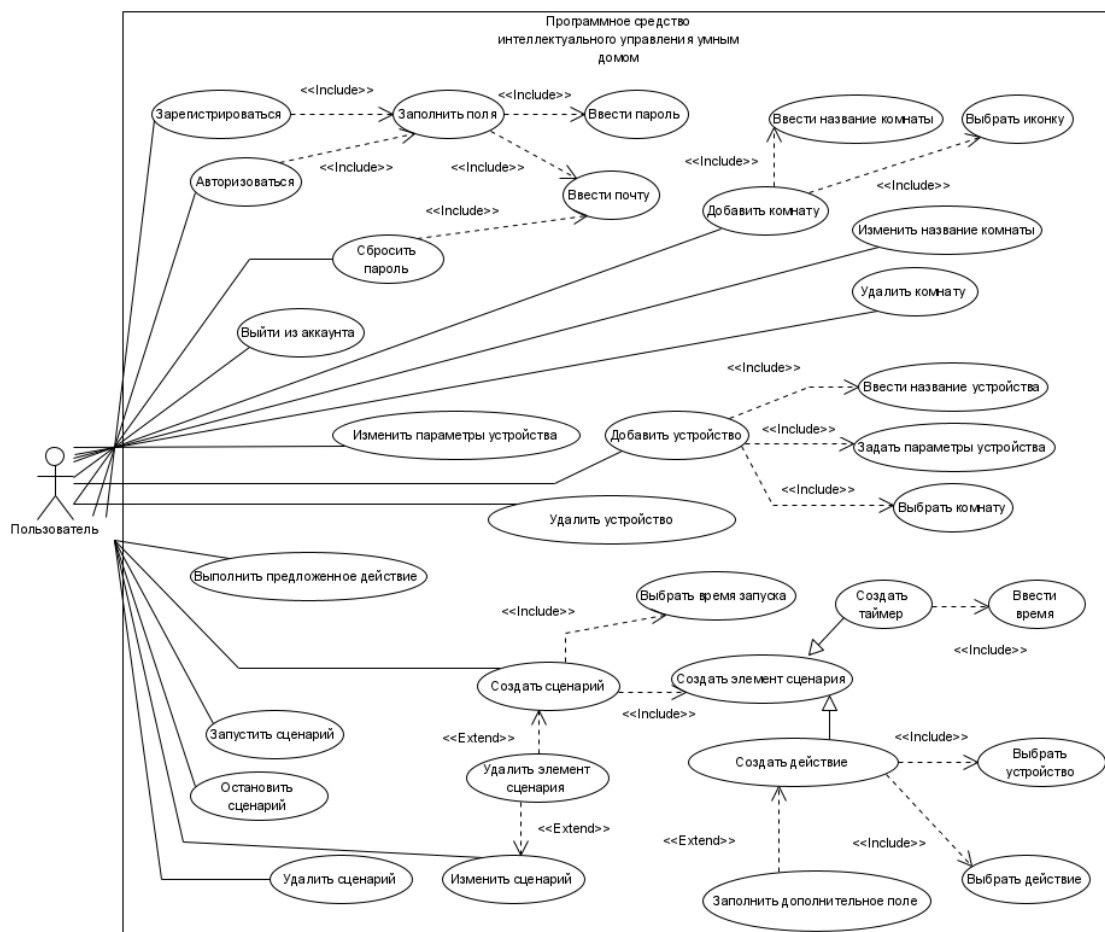


Рисунок 1 – Диаграмма вариантов использования

Был выделен единственный актер – пользователь. Если он впервые вошёл в программу, то он должен зарегистрироваться, для чего необходимо ввести адрес электронной почты и пароль. Описание варианта использования «Зарегистрироваться» представлено в таблице 7.

Таблица 7 – Вариант использования «Зарегистрироваться»

Вариант использования	Зарегистрироваться
ID	1
Краткое описание	Пользователь регистрируется в приложении, вводя почту и пароль
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	1. Почта корректна; 2. Почта не занята; 3. Пароль корректен.
Основной поток	1. Пользователь вводит почту; 2. Пользователь вводит пароль.
Постусловия	Пользователь зарегистрирован

Если же пользователь уже зарегистрирован, он может пройти авторизацию, введя данные указанные во время регистрации. Описание варианта использования «Авторизоваться» представлено в таблице 8.

Таблица 8 – Вариант использования «Авторизоваться»

Вариант использования	Авторизоваться
ID	2
Краткое описание	Пользователь авторизуется в приложении, вводя почту и пароль
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Нет
Основной поток	1. Пользователь вводит почту; 2. Пользователь вводит пароль.
Постусловия	Пользователь авторизовался

Также пользователь может сбросить пароль или выйти из аккаунта при необходимости. Описания вариантов использования «Сбросить пароль» и «Выйти из аккаунта» представлены в таблицах 9 и 10.

Таблица 9 – Вариант использования «Сбросить пароль»

Вариант использования	Сбросить пароль
ID	3
Краткое описание	Пользователь восстанавливает пароль в приложении, вводя почту
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Нет
Основной поток	1. Пользователь вводит почту; 2. Пользователь переходит по ссылке из письма, которое пришло ему на почту; 3. Пользователь попадает на сайт, где вводит новый пароль.
Постусловия	Пароль сброшен

Таблица 10 – Вариант использования «Выйти из аккаунта»

Вариант использования	Выйти из аккаунта
ID	4
Краткое описание	Пользователь выходит из аккаунта в приложении, нажимая кнопку «Выйти»
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Нет
Основной поток	Пользователь нажимает кнопку «Выйти»
Постусловия	Пользователь вышел из аккаунта

Перед тем, как добавлять устройства, необходимо создать минимум одну комнату. Для этого пользователь вводит название и выбирает иконку из предложенных вариантов. Описание варианта использования «Добавить комнату» представлено в таблице 11.

Таблица 11 – Вариант использования «Добавить комнату»

Вариант использования	Добавить комнату
ID	5
Краткое описание	Пользователь добавляет комнату
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Название комнаты корректно
Основной поток	1. Пользователь добавляет новую комнату нажав на иконку «Плюс» 2. Пользователь вводит название комнаты; 3. Пользователь выбирает иконку.
Постусловия	Добавляется комната

При необходимости можно изменить название уже созданной комнаты или вовсе удалить её. Описания вариантов использования «Изменить название комнаты» и «Удалить комнату» представлены в таблицах 12 и 13.

Таблица 12 – Вариант использования «Изменить название комнаты»

Вариант использования	Изменить название комнаты
ID	6
Краткое описание	Пользователь меняет название комнаты
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Комната существует
Основной поток	1. Пользователь выбирает комнату; 2. Изменяет название комнаты.
Постусловия	Название комнаты изменено

Таблица 13 – Вариант использования «Удалить комнату»

Вариант использования	Удалить комнату
ID	7
Краткое описание	Пользователь удаляет комнату
Главные актёры	Пользователь
Второстепенные актёры	Нет
Предусловия	Комната существует
Основной поток	1. Пользователь удерживает иконку комнаты до появления окна подтверждения; 2. Пользователь подтверждает действие.
Постусловия	Комната удалена

Для добавления устройства, необходимо ввести его название, задать параметры и выбрать комнату из списка. Описание варианта использования «Добавить устройство» представлено в таблице 14.

Таблица 14 – Вариант использования «Добавить устройство»

Вариант использования	Добавить устройство
ID	8
Краткое описание	Пользователь добавляет устройство
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Название устройства корректно
Основной поток	1. Пользователь переходит на страницу добавления; 2. Пользователь нажимает на иконку с нужным типом устройства; 3. Пользователь вводит название устройства; 3. Пользователь задает нужные настройки устройства; 4. Пользователь выбирает комнату для добавления устройства.
Постусловия	Добавляется устройство

Пользователь также может изменить параметры устройства, для этого необходимо открыть его страницу и задать новые параметры. Описание варианта использования «Изменить параметры устройства» представлено в таблице 15.

Таблица 15 – Вариант использования «Изменить параметры устройства»

Вариант использования	Изменить параметры устройства
ID	9
Краткое описание	Пользователь изменяет параметры устройства
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Устройство существует
Основной поток	1. Пользователь выбирает устройство; 2. Пользователь изменяет параметры устройства.
Постусловия	Параметры устройства изменены

После можно удалить устройство. Описание варианта использования «Удалить устройство» представлено в таблице 16.

Таблица 16 – Вариант использования «Удалить устройство»

Вариант использования	Удалить устройство
ID	10
Краткое описание	Пользователь удаляет устройство
Главные актёры	Пользователь
Второстепенные актеры	Нет

Продолжение таблицы 16

Вариант использования	Удалить устройство
Предусловия	Устройство существует
Основной поток	1. Пользователь удерживает иконку устройства до появления окна подтверждения; 2. Пользователь подтверждает действие.
Постусловия	Устройство удалено

Пользователь может создавать сценарии, для чего необходимо выбрать время запуска и добавить элементы сценария. Описание варианта использования «Создать сценарий» представлено в таблице 17.

Таблица 17 – Вариант использования «Создать сценарий»

Вариант использования	Создать сценарий
ID	11
Краткое описание	Пользователь создает сценарий
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Нет
Основной поток	1. Пользователь переходит на страницу «Сценарии»; 2. Пользователь нажимает на иконку «Плюс» для создания сценария; 3. Пользователь задает название для сценария; 4. Пользователь задает время запуска для сценария; 5. Пользователь создает элементы сценария.
Постусловия	Создан сценарий

Элементами могут быть либо действия с устройствами, либо таймер. При действии нужно выбрать устройство, выполняемое им действие и, если необходимо заполнить дополнительное поле, в котором может быть указана, например, температура кондиционера. Описание варианта использования «Создать действие» представлено в таблице 18.

Таблица 18 – Вариант использования «Создать действие»

Вариант использования	Создать действие
ID	12
Краткое описание	Пользователь создает элемент сценария с типом «Действие»
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Нет

Продолжение таблицы 18

Вариант использования	Создать действие
Основной поток	1. Пользователь нажимает на иконку «Плюс» для создания элемента сценария; 2. Пользователь выбирает тип «Действие»; 3. Пользователь выбирает нужное устройство; 4. Пользователь выбирает нужное действие.
Постусловия	Создан элемент сценария с типом «Действие»

Если выбран таймер, то нужно ввести время для ожидания. При необходимости элементы сценария могут быть удалены. Описание варианта использования «Создать таймер» представлено в таблице 19.

Таблица 19 – Вариант использования «Создать таймер»

Вариант использования	Создать таймер
ID	13
Краткое описание	Пользователь создает элемент сценария с типом «Таймер»
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Нет
Основной поток	1. Пользователь нажимает на иконку «Плюс» для создания элемента сценария; 2. Пользователь выбирает тип «Таймер»; 3. Пользователь задает время ожидания.
Постусловия	Создан элемент сценария с типом «Таймер»

Созданный сценарий можно запустить либо остановить. Описания вариантов использования «Запустить сценарий» и «Остановить сценарий» представлены в таблицах 20 и 21.

Таблица 20 – Вариант использования «Запустить сценарий»

Вариант использования	Запустить сценарий
ID	14
Краткое описание	Пользователь запускает сценарий
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Сценарий создан
Основной поток	Пользователь запускает сценарий
Постусловия	Сценарий запущен

Таблица 21 – Вариант использования «Остановить сценарий»

Вариант использования	Остановить сценарий
ID	15
Краткое описание	Пользователь останавливает сценарий
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Сценарий создан и запущен
Основной поток	Пользователь останавливает сценарий
Постусловия	Сценарий остановлен

Сценарии также можно при необходимости изменить или удалить. Описания вариантов использования «Изменить сценарий» и «Удалить сценарий» представлены в таблицах 22 и 23.

Таблица 22 – Вариант использования «Изменить сценарий»

Вариант использования	Изменить сценарий
ID	16
Краткое описание	Пользователь изменяет сценарий
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Сценарий создан
Основной поток	1. Пользователь выбирает сценарий; 2. Пользователь изменяет сценарий.
Постусловия	Сценарий изменен

Таблица 23 – Вариант использования «Удалить сценарий»

Вариант использования	Удалить сценарий
ID	17
Краткое описание	Пользователь удаляет сценарий
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Сценарий создан
Основной поток	1. Пользователь удерживает нужный сценарий для удаления; 2. Пользователь подтверждает действие.
Постусловия	Сценарий удален

Помимо этого, пользователю могут приходить уведомления с предложением выполнить определенное действие, например, включить свет в комнате или изменить температуру кондиционера. Описание варианта использования «Выполнить предложенное действие» представлено в таблице 24.

Таблица 24 – Вариант использования «Выполнить предложенное действие»

Вариант использования	Выполнить предложенное действие
ID	18
Краткое описание	Пользователь подтверждает выполнение предложенного действия
Главные актёры	Пользователь
Второстепенные актеры	Нет
Предусловия	Пришло уведомление с предложением действия
Основной поток	Пользователь подтвердил выполнение предложенного действия
Постусловия	Предложенное действие выполнено

Такие предложения формируются моделью машинного обучения с учителем. Модель использует данные о действиях, которые выполнял пользователь в определенное время каждый день и на основе это предлагает выполнить те же действия в то же время.

1.5 Функционально-стоимостной анализ

Планирование времени и бюджета проекта позволяет определить финансовые ресурсы и сроки выполнения, необходимые для успешной реализации проекта. Для такого планирования выделяются задачи и затрачиваемые для их выполнения ресурсы. Это позволяет определить даты и стоимость выполнения каждого этапа работы над проектом, а также оценить возможные риски и проблемы.

Для планирования времени, необходимого на разработку программного средства интеллектуального управления умным домом были определены задачи и сроки их выполнения, представленные в таблице 25.

Таблица 25 – Задачи проекта

№	Наименование задачи	Дата начала	Дата окончания	Длительность (в часах)
1	Анализ предметной области и требований	12.04.2023	17.04.2023	32
1.1	Анализ предметной области и постановка задач	12.04.2023	12.04.2023	8
1.2	Анализ существующих решений	13.04.2023	14.04.2023	16
1.3.	Анализ функциональных требований	17.04.2023	17.04.2023	8
2	Анализ технологий проектирования и разработки	18.04.2023	18.04.2023	8

Продолжение таблицы 25

3	Обучение модели машинного обучения	19.04.2023	20.04.2023	16
4	Проектирование программного средства	21.04.2023	03.05.2023	72
4.1.	Проектирование клиентской части	21.04.2023	25.04.2023	24
4.2.	Проектирование серверной части	26.04.2023	28.04.2023	24
4.3.	Проектирование базы данных	01.05.2023	01.05.2023	8
4.4.	Проектирование интерфейса	02.05.2023	03.05.2023	16
5	Разработка программного средства	04.05.2023	16.05.2023	72
5.1.	Разработка клиентской части	04.05.2023	09.05.2023	32
5.2.	Разработка серверной части	10.05.2023	12.05.2023	24
5.3.	Разработка пользовательского интерфейса	15.05.2023	16.05.2023	16
6	Тестирование приложения	17.05.2023	24.05.2023	48
6.1.	Тестирование методом черного ящика	17.05.2023	18.05.2023	16
6.2.	Тестирование метрики кода	19.05.2023	22.05.2023	16
6.3.	Автоматизированное тестирование	23.05.2023	24.05.2023	16
7	Оформление пояснительной записки	25.05.2023	31.05.2023	40

Для представления задач и их сроков в удобной форме была составлена диаграмма Ганта. Диаграмма Ганта – это инструмент, позволяющий управлять проектами. С ее помощью можно структурировать задачи, отслеживать прогресс выполнения поставленных целей. Диаграмма состоит из временной шкалы: вертикальной оси, отвечающей за список задач, и горизонтальной оси, которая несет в себе временную шкалу проекта [8].

Диаграмма Ганта приведена на рисунке 2.

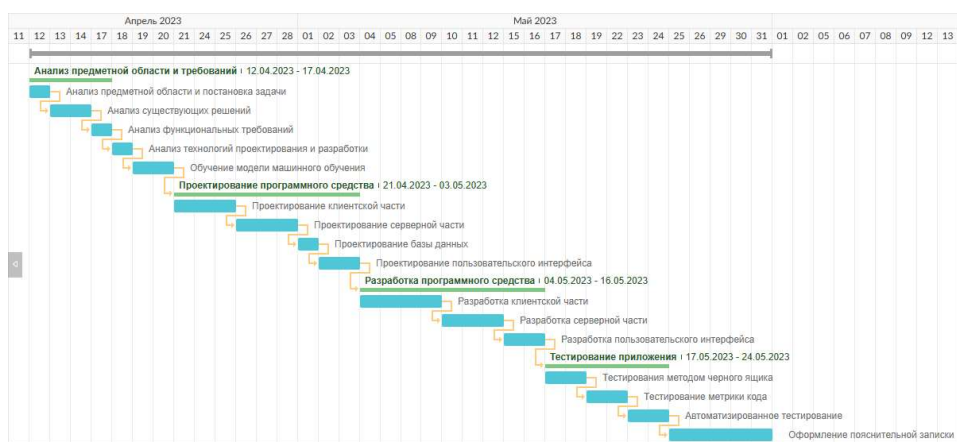


Рисунок 2 – Диаграмма Ганта

Далее были определены ресурсы проекта и подсчитана стоимость выполнения каждого этапа разработки. Результат представлен в таблице 26.

Таблица 26 – Длительность и стоимость выполнения этапов проекта

№	Наименование этапа	Длительность (в часах)	Исполнитель	Зарплата (руб/ч)	Стоимость
1	Анализ предметной области и требований	32	Аналитик	100	3200
2	Анализ технологий проектирования и разработки	8	Аналитик	100	800
3	Обучение модели машинного обучения	16	Программист	150	2400
4	Проектирование программного средства	72	Проектировщик	100	7200
5	Разработка программного средства	72	Программист	150	10800
6	Тестирование программного средства	48	Тестировщик	100	4800
7	Оформление пояснительной записки	40	Технический писатель	100	4000

В результате планирования были определены время выполнения и стоимость каждого этапа разработки. Суммарная стоимость всего проекта составляет 33200 рублей, а длительность выполнения без учета выходных дней составляет 288 часов.

2 Анализ технологий проектирования и разработки

2.1 Инструментальные средства проектирования

2.1.1 Язык моделирования UML

Унифицированный язык моделирования (Unified Modeling Language, UML) – это графический язык для визуализации, спецификации, построения и документирования систем, в которых программное обеспечение играет главную роль. Используя UML, можно описать не только детали системы, системные функции, бизнес-процессы, но и классы, написанные на языках программирования, схемы баз данных и повторно используемое программное обеспечение [9]. Самыми полезными и часто используемыми диаграммами являются: диаграмма классов, последовательности, деятельности, состояний. Схемы UML имеют ряд преимуществ, например, универсальность, языковую независимость, уменьшение времени разработки, улучшение качества продукта.

2.1.2 Выбор средств проектирования

Для проектирования диаграмм UML будет использоваться программа Visual Paradigm. Visual Paradigm – это инструмент для моделирования информационных систем, который позволяет создавать диаграммы классов, диаграммы компонентов, диаграммы последовательности, а также бизнес-процессов, проектирования баз данных и решения многих других задач [10]. Visual Paradigm позволяет создавать приложения с помощью языков UML, SysML, SoaML [11]. Также, в программе есть функция автоматического создания кода на основе диаграмм, возможность импорта и экспорта данных из различных форматов файлов.

2.2 Инструментальные средства разработки

2.2.1 Язык программирования Dart

Dart – это высокоуровневый язык программирования, разработанный компанией Google. Язык в основном применяется для разработки мобильных и веб приложений. При этом для разработки используется фреймворк Flutter, который позволяет легко создавать приложения под все основные платформы.

Язык является С-подобным, поэтому его синтаксис и структура кода напоминает С. При этом Dart не имеет строгих правил касательно типизации. Разработчик может как вручную задавать все типы переменных при статической типизации, так и не указывать их и использовать динамическую типизацию.

Также Dart поддерживает два вида компиляции:

- АОТ (ahead of time) – код программы целиком переводится в машинные коды и после этого запускается;
- JIT (just in time) – код программы компилируется и выполняется частично.

При АОТ компиляции готовое приложение работает довольно быстро, однако время его разработки увеличивается. JIT компиляция ускоряет разработку, однако увеличивает время запуска готового приложения.

Благодаря этому, разработчик может проводить тестирование и отладку отдельных модулей программы используя JIT компиляцию, а на завершающем этапе разработки выполнить АОТ компиляцию, получив в результате и ускоренную разработку, и быстрое приложение [12].

В дополнение ко всему, Dart имеет множество встроенных библиотек, позволяющих использовать кодировщики и декодеры данных, математические функции и константы, асинхронное программирование и другие [13]. Так же существует множество сторонних библиотек, позволяющих добавить практически любые необходимые функции.

2.2.2 Платформа Flutter

Flutter – это бесплатная платформа с открытым исходным кодом, разработанная компанией Google. Фреймворк в основном используется для разработки мобильных приложений под Android и IOS, однако существует и поддержка разработки для Windows, MacOS и Linux, а вместе с тем и веб-приложений.

Flutter включает в себя набор инструментов для разработки, а также библиотеку готовых элементов пользовательского интерфейса – виджетов. Разработка происходит на языке программирования Dart, который компилируется в машинный код, что обеспечивает быструю работу. При этом достаточно написать только один код, который будет использован для создания приложений на нескольких платформах.

Приятной особенностью Flutter является «горячая перезагрузка». Она позволяет менять код и сразу же видеть результат изменений. Это позволяет значительно ускорить разработку интерфейса приложения.

Помимо этого, платформа имеет подробную документацию с простыми примерами. Поэтому разработчики могут легко найти информацию на возникающие во время разработки вопросы [14].

2.2.3 Платформа Firebase

Firebase – это облачная платформа разработки мобильных приложений, созданная компанией Google. Платформа предоставляет разработчикам такие инструменты как:

- база данных Cloud Firestore;
- аутентификации пользователей;
- хранение файлов;
- аналитика;
- хостинг веб приложений;
- сервис машинного обучения.

Использование Firebase позволяет повысить скорость разработки за счет наличия множества готовых сервисов, а также дает возможность разработчикам сосредоточить внимание на клиентской части приложения. Так как платформа работает на базе Google, то позволяет взаимодействовать с другими сервисами этой компании [15].

Cloud Firestore является частью платформы Firebase. База данных размещается в облачном хранилище и предоставляет доступ к себе через собственные SDK [16].

База данных является документно-ориентированной, то есть в ней нет таблиц или строк. Поэтому все данные хранятся в документах. Документ – это запись, содержащая поля по типу ключ-значение. Документы находятся в контейнерах, которые называются коллекциями. Таким образом, база данных представляет собой множество коллекций и документов в них.

Cloud Firestore является NoSQL базой данных. Это означает, что она не имеет схемы и в каждом документе одной коллекции могут находиться совершенно разные поля [17].

Дерево представляет собой набор вопросов к данным. Узлы дерева, где пишется вопрос, называются вершинами. Узлы, где нет вопросов, называются листьями. Стрелка влево из вершины означает истинность условия, вправо – ложность. Из дерева видно, что при значении времени меньше 5,5 часов (самая верхняя вершина), целевая переменная имеет класс «off», то есть устройство выключено. А если время больше 5,5 часов, но меньше 6,5 часов, то целевая переменная принимает класс «on», то есть устройство включено. Таким образом, дерево решений задает вопросы о времени и получает результат – включено устройство или нет.

Была подсчитана точность обученной модели с помощью метода score. В качестве параметров использовались ранее отобранные тренировочные данные. Точность составила 79%, что говорит о корректной работе модели.

Однако такая модель имеет весомый недостаток. Если пользователь не будет выполнять действия на протяжении долгого времени, модель всё ещё продолжит предлагать их выполнять. Это происходит, потому что модель использует данные, которые когда-то были актуальны, однако сейчас уже утратившие её. Для исправления данной проблемы был создан механизм «забывания» старых данных, который заключается в добавлении записей о выключенном состоянии устройств в конце каждой недели. Таким образом, через одну-две недели модель перестанет выполнять предложения.

4 Проектирование программного средства

4.1 Проектирование клиентской части приложения

Клиентская часть приложения реализована с использованием фреймворка Flutter. Структура приложения представляет собой совокупность страниц для создания устройств, комнат и сценариев, а также страниц для взаимодействия с ними. Для визуализации основных классов и связей между ними была разработана диаграмма классов.

Диаграмма классов – это графическое представление классов, интерфейсов, атрибутов и методов программного кода. Она используется для описания структуры и взаимодействия компонентов программы в объектно-ориентированном программировании [19]. Разработанная диаграмма классов представлена на рисунке 4.

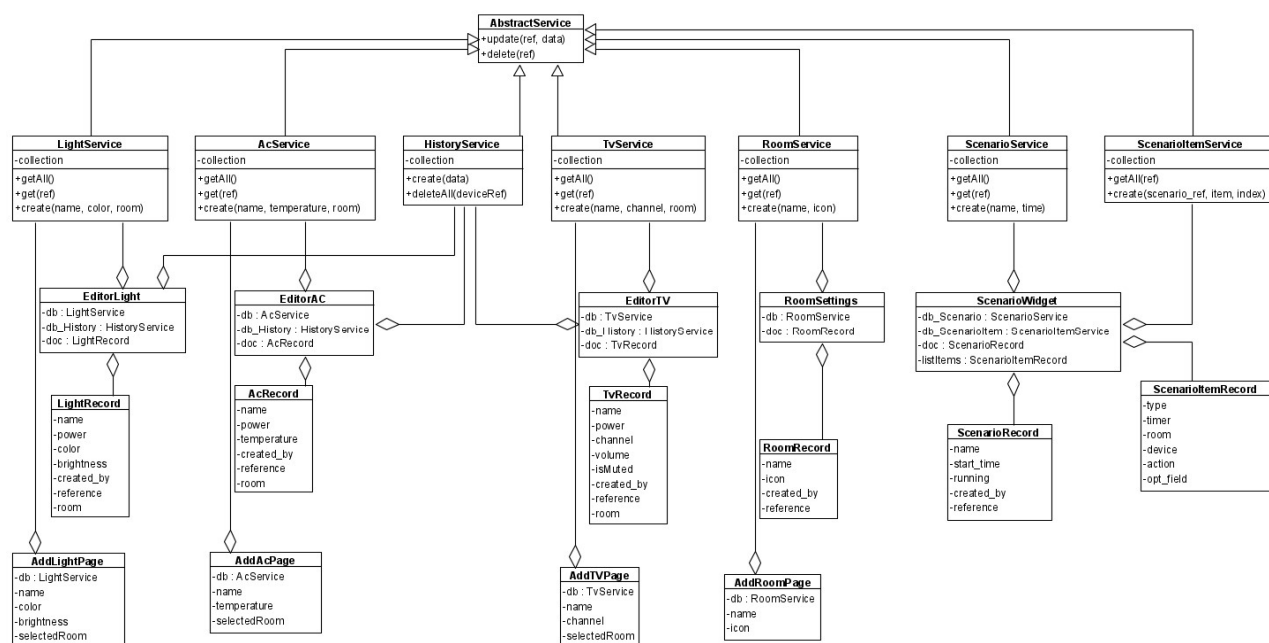


Рисунок 4 – Диаграмма классов

Классы LightService, AcService, TvService, HistoryService, RoomService, ScenarioService и ScenarioItemService предназначены для взаимодействия с базой данных. Они наследуются от класса AbstractService, который имеет следующие методы:

- update – обновляет указанные в объекте поля документа БД;

- delete – удаляет указанный документ.

При этом последние методы update и delete уже имеют реализацию, так как для любого объекта базы данных они одинаковые. Поэтому эти методы не нуждаются в реализации в классах наследниках.

Все классы содержат поле collection, которое является объектом коллекции базы данных Firestore, к которой выполняются запросы.

Классы RoomService, TvService, AcService, LightService и ScenarioService предназначены для получения и создания устройств, комнат и сценариев и имеют следующие методы:

- getAll – получает список всех объектов из базы данных;
- get – получает объект из БД по указанной ссылке;
- create – создает документ в БД с указанными параметрами.

Класс ScenarioItemService предназначен для получения и создания элементов сценария.

Класс HistoryService предназначен для добавления и удаления записей о действиях пользователей. Он содержит метод для создания одной и удаления всех записей, которые связаны с определенным устройством.

Класс LightRecord предназначен для хранения данных об устройстве освещения. Он содержит следующие поля:

- name – имя устройства;
- power – состояние устройства(включено или выключен);
- color – цвет освещения;
- brightness – яркость освещения;
- created_by – ссылка на документ пользователя, который создал устройство;
- reference – ссылка на документ устройства в базе данных;
- room – ссылка на документ комнаты, где находится устройство.

Класс AcRecord предназначен для хранения данных о кондиционере. Он содержит следующие поля:

- name – имя устройства;

- power – состояние устройства(включено или выключен);
- temperature – температура;
- created_by – ссылка на документ пользователя, который создал устройство;

- reference – ссылка на документ устройства в базе данных;
- room – ссылка на документ комнаты, где находится устройство.

Класс TvRecord предназначен для хранения данных о телевизоре. Он содержит следующие поля:

- name – имя устройства;
- power – состояние устройства(включено или выключен);
- channel – канал;
- volume – громкость;
- isMuted – состояние звука (заглушен или нет);
- created_by – ссылка на документ пользователя, который создал устройство;

- reference – ссылка на документ устройства в базе данных;
- room – ссылка на документ комнаты, где находится устройство.

Класс RoomRecord предназначен для хранения данных о комнате. Он содержит следующие поля:

- name – имя комнаты;
- icon – иконка комнаты;
- created_by – ссылка на документ пользователя, который создал комнату;
- reference – ссылка на документ комнаты в базе данных.

Класс ScenarioRecord предназначен для хранения данных о сценарии. Он содержит следующие поля:

- name – имя сценария;
- start_time – время запуска сценария;
- running – состояние сценария (запущен или нет);

- `created_by` – ссылка на документ пользователя, который создал сценарий;
- `reference` – ссылка на документ сценария в базе данных.

Класс `ScenarioItemRecord` предназначен для хранения данных об элементе сценария. Он содержит следующие поля:

- `type` – тип элемента сценария (действие или ожидание);
- `timer` – время ожидания (пустое, если тип - действие);
- `room` – ссылка на документ комнаты в базе данных (пустое, если тип - ожидание);
- `device` – ссылка на документ устройства в базе данных (пустое, если тип - ожидание);
- `action` – выполняемое с устройством действие (пустое, если тип - ожидание);
- `opt_field` – дополнительное поле, применяемое для ввода значений, таких как номер канала, температура и т. д. (пустое, если тип - ожидание).

Класс `EditorLight` является страницей для изменения настроек освещения. Он содержит поля для доступа к базе данных коллекции освещения (поле `db`) и истории действий (поле `db_History`). Также имеется поле `doc`, которое является объектом класса `LightRecord`, содержащее данные о устройстве.

Классы `EditorTv` и `EditorAc` предназначены для аналогичных целей для телевизора и кондиционера соответственно.

Класс `RoomSettings` является страницей для просмотра содержимого комнаты и изменения её названия. Класс содержит поле `db` для доступа к базе данных коллекции комнат, а также поле `doc`, которое является объектом класса `RoomRecord`, содержащее данные о комнате.

Класс `ScenarioWidget` является страницей сценария, где можно изменить название сценария, выбрать время запуска и добавить элементы сценария. Класс содержит следующие поля:

- `db_Scenario` – предоставляет доступ к базе данных коллекции сценариев;

- `db_ScenarioItem` – предоставляет доступ к базе данных коллекции элементов сценария;
- `doc` – содержит объект класса `ScenarioRecord`, содержащий данные о сценарии;
- `listItems` – содержит список объектов класса `ScenarioItemRecord`, содержащими данные об элементах сценария.

Классы `AddLightPage`, `AddAcPage` и `AddTvPage` являются страницами добавления освещения, кондиционера и телевизора соответственно. Они содержат поля для доступа к своим коллекциям в базе данных и поля настроек устройств.

Класс `AddRoomPage` является страницей создания комнаты. Он содержит поле для доступа к коллекции комнат в базе данных, а также поля имени комнаты и её иконки.

4.2 Проектирование серверной части

Серверная часть приложения предназначена для запуска и остановки сценариев, а также для отправки пользователю предложений для включения устройств. Сервер реализован на языке Python с использованием Фреймворка Flask.

Сервер содержит методы для получения POST запросов, предназначенных для запуска и остановки выполнения сценариев. При получении запроса по адресу «`/start`» сервер создает повторяющееся задание, которое будет запускать указанный сценарий каждый день в заданное время. При получении запроса по адресу «`/stop`» задание удаляется, и выполнение сценария больше запущено не будет.

Выполнение сценариев по расписанию реализуется при помощи библиотеки `APScheduler`, которая позволяет создавать задания, выполняемые одновременно или же периодически. Так при получении запроса на запуск сценария создается задание, которое выполняет функцию `start_scenario` один раз

в сутки в указанное время. Алгоритм функции выполнения сценария представлен на диаграмме деятельности на рисунке 5.

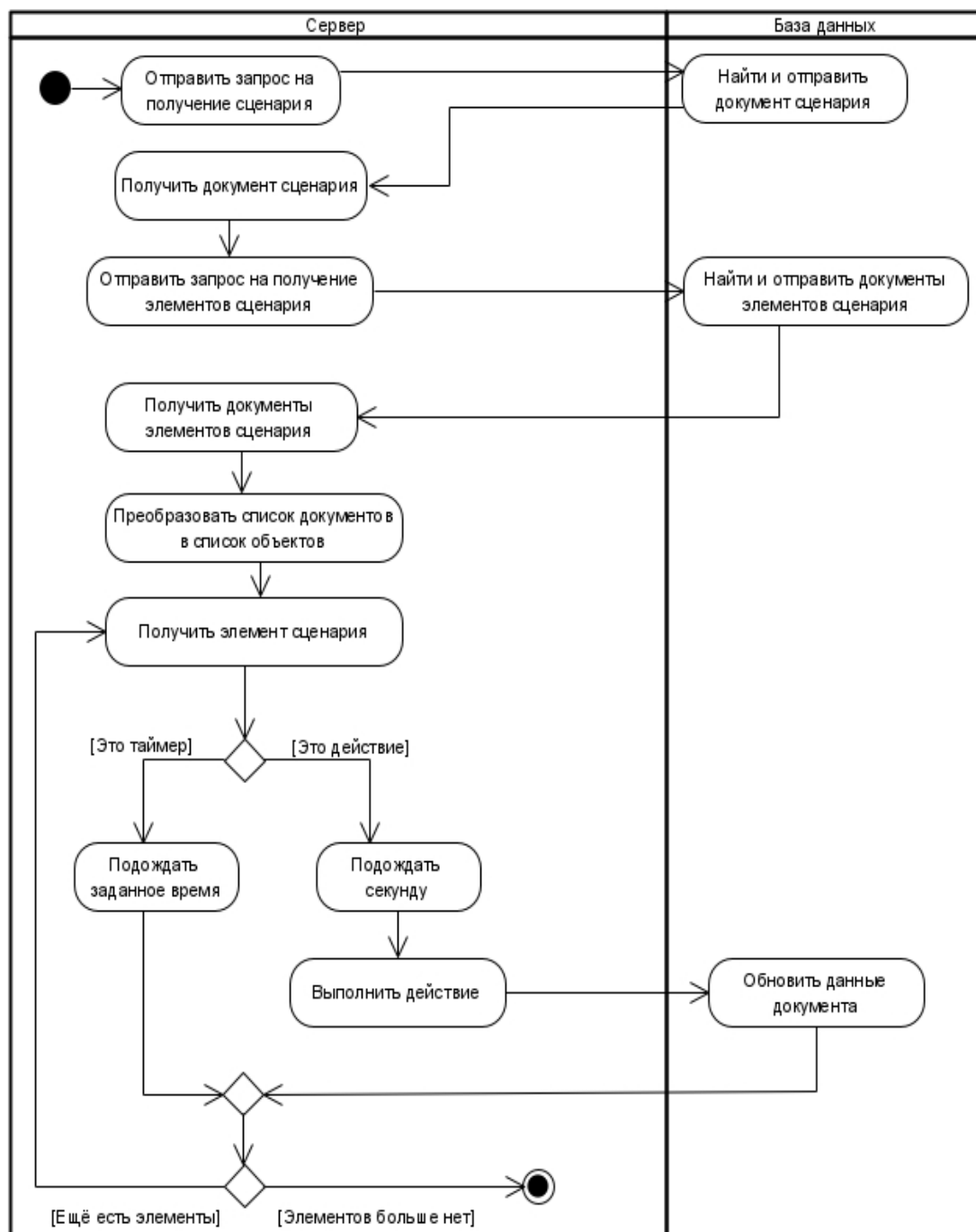


Рисунок 5 – Диаграмма деятельности

Чтобы выполнить сценарий сервер отправляет запрос к базе данных для получения документа сценария по указанному идентификатору. После этого отправляется запрос на получение элементов этого сценария. Полученный список документов преобразуется в список объектов. Далее начинается цикл по каждому элементу. Если элемент является таймером, то программа

приостанавливается на заданное время. Если элемент является действием, то после секундного ожидания оно выполняется путем обновления указанных полей документа в базе данных. После этого, если в списке ещё есть элементы, то для каждого выполняются аналогичный алгоритм. В противном случае выполнение сценария считается завершённым.

Помимо этого, сервер каждый час выполняет проверку о необходимости предложения действия пользователю. Если в это время пользователь часто выполняет определенное действие, то сервер отправит уведомление с предложением повторить это действие.

4.3 Проектирование базы данных

Для выполнения ранее описанных функциональных требований была построена логическая модель базы данных в нотации «сущность-связь».

Диаграмма «сущность – связь» позволяет разработать модель данных предметной области.

Основными элементами диаграммы являются:

- сущность;
- атрибуты сущности;
- связь между сущностями;
- степень связи;
- класс принадлежности экземпляров сущности.

Сущности – это объекты, информация о которых хранится в базе данных. Экземпляры сущностей различны и могут быть однозначно распознаны. Имена сущностей обычно являются существительными.

Атрибуты – это свойства сущности.

Ключ сущности – это атрибут или набор атрибутов, используемых для идентификации экземпляра сущности.

Связь между сущностями – означает, что атрибуты одной сущности зависят от атрибутов другой. Такая связь называется отношением и обычно представляется глаголом [20].

Логическая модель базы данных представлена на рисунке 6.



Рисунок 6 – Логическая модель базы данных

Созданная модель позволяет хранить данные о следующих сущностях:

- пользователи;
- комнаты;
- телевизоры;
- кондиционеры;
- освещение;
- сценарии;
- элементы сценария;
- история действий.

Сущность «Пользователи» предназначена для хранения информации о пользователях. Описание атрибутов сущности представлено в таблице 27.

Таблица 27 – Атрибуты сущности «Пользователи»

Название	Роль	Описание
Идентификатор	Первичный ключ	Уникальный идентификатор пользователя
Почта		Адрес электронной почты пользователя
Пароль		Пароль пользователя

Сущность «Комнаты» предназначена для хранения информации о комнатах. Описание атрибутов сущности представлено в таблице 28.

Таблица 28 – Атрибуты сущности «Комнаты»

Название	Роль	Описание
Идентификатор	Первичный ключ	Уникальный идентификатор комнаты
Название		Название комнаты
Иконка		Иконка комнаты
Владелец	Внешний ключ	Владелец комнаты

Сущность «Телевизоры» предназначена для хранения информации о комнатах. Описание атрибутов сущности представлено в таблице 29.

Таблица 29 – Атрибуты сущности «Телевизоры»

Название	Роль	Описание
Идентификатор	Первичный ключ	Уникальный идентификатор телевизора
Название		Название телевизора
Состояние		Состояние телевизора (вкл/выкл)
Владелец	Внешний ключ	Владелец телевизора
Канал		Текущий канал
Громкость		Текущая громкость
Звук выключен		Состояние звука (заглушен или нет)

Сущность «Кондиционеры» предназначена для хранения информации о комнатах. Описание атрибутов сущности представлено в таблице 30.

Таблица 30 – Атрибуты сущности «Кондиционеры»

Название	Роль	Описание
Идентификатор	Первичный ключ	Уникальный идентификатор кондиционера
Название		Название кондиционера
Состояние		Состояние кондиционера (вкл/выкл)
Владелец	Внешний ключ	Владелец кондиционера

Сущность «Освещение» предназначена для хранения информации о комнатах. Описание атрибутов сущности представлено в таблице 31.

Таблица 31 – Атрибуты сущности «Освещение»

Название	Роль	Описание
Идентификатор	Первичный ключ	Уникальный идентификатор освещения
Название		Название освещения
Состояние		Состояние освещения (вкл/выкл)
Владелец	Внешний ключ	Владелец освещения
Цвет		Выбранный цвет света
Яркость		Выбранная яркость света

Сущность «Сценарии» предназначена для хранения информации о сценариях. Описание атрибутов сущности представлено в таблице 32.

Таблица 32 – Атрибуты сущности «Сценарии»

Название	Роль	Описание
Идентификатор	Первичный ключ	Уникальный идентификатор сценария
Название		Название сценария
Время запуска		Время запуска сценария
Владелец	Внешний ключ	Владелец освещения
Состояние		Состояние сценария (запущен/не запущен)

Сущность «Элементы сценария» предназначена для хранения информации об элементах внутри сценариев. Описание атрибутов сущности представлено в таблице 33.

Таблица 33 – Атрибуты сущности «Элементы сценария»

Название	Роль	Описание
Идентификатор	Первичный ключ	Уникальный идентификатор сценария
Тип		Тип элемента (таймер/действие)
Сценарий	Внешний ключ	Сценарий, в состав которого входит элемент
Номер		Порядковый номер элемента внутри сценария
Параметры элемента		

Сущность «История действий» предназначена для хранения информации действиях пользователей с устройствами. Описание атрибутов сущности представлено в таблице 34.

Таблица 34 – Атрибуты сущности «История действий»

Название	Роль	Описание
Идентификатор	Первичный ключ	Уникальный идентификатор записи истории
Устройство	Внешний ключ	Устройство, с которым произошло событие
Действие		Действие, выполненное пользователем с устройством
Время		Время выполнения действия
Пользователь	Внешний ключ	Пользователь, выполнивший действие

Таким образом, спроектированная база данных позволит выполнять все необходимые функции.

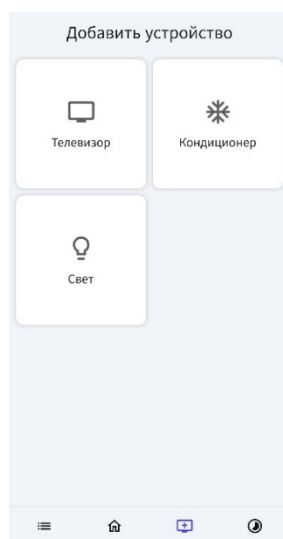
4.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс – это визуальная часть программы, которая определяет, как отображается информация и как с ней взаимодействовать. Качественный интерфейс позволяет сделать взаимодействие с программой простым, понятным и что не менее важно эффективным [21].

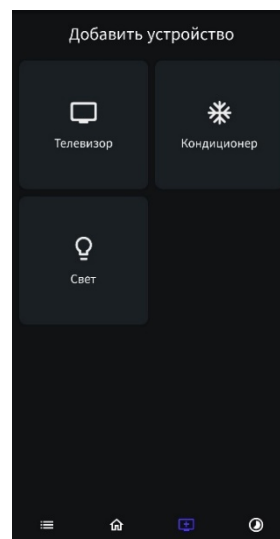
В разрабатываемом приложении интерфейс должен позволять пользователю просматривать список своих комнат и устройств, создавать, редактировать и удалять их, а также предоставлять удобный способ управления устройствами. Основные функции должны быть расположены в нижней части экрана, чтобы пользователь имел к ним быстрый доступ.

Помимо этого, в приложении должна быть возможность переключения темы на темную. Она придает программе приятный внешний вид, позволяет экономить заряд батареи телефона с определенным типом матрицы, снимает напряжение с глаз, а также увеличивает комфорт пользования в темное время суток [22].

В качестве примера внешнего вида приложения приведен экран «Добавить устройство» в светлой (рисунок 7, а) и темной теме (рисунок 7, б).



а) светлая тема



б) темная тема

Рисунок 7 – Внешний вид экрана «Добавить устройство»

Для визуализации всех страниц приложения была составлена карта экранов, представленная на рисунке 8.

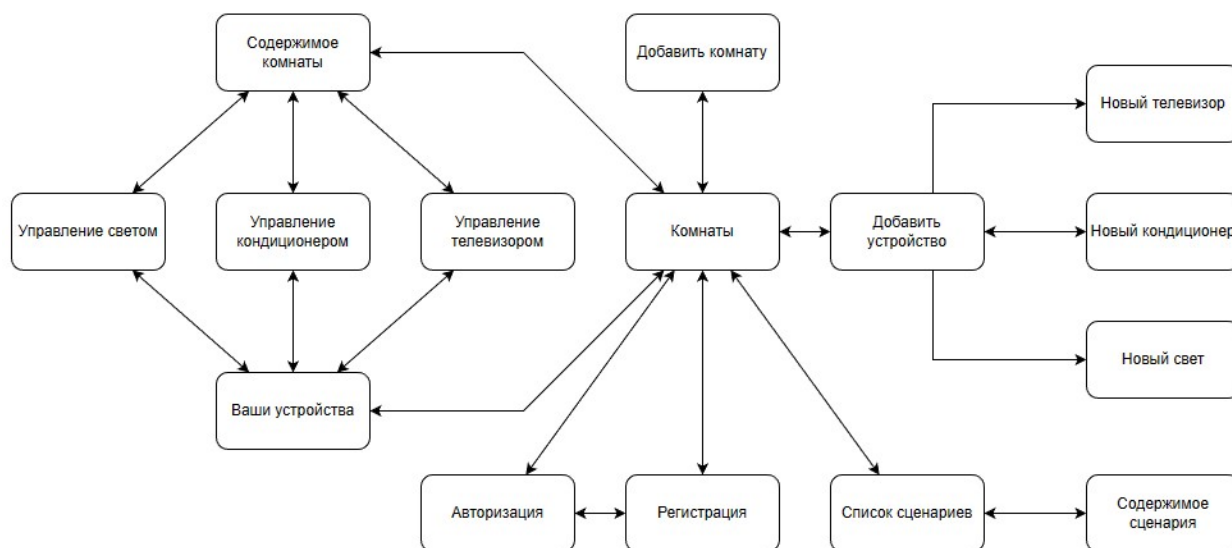


Рисунок 8 – Карта экранов

При первом запуске приложения пользователь попадает на экран авторизации. Он может либо войти в аккаунт и попасть на страницу «Комнаты», где отображаются все комнаты. Если пользователь не имеет своей учетной записи, он может перейти на страницу регистрации, из которой после создания аккаунта перейдёт на экран «Комнаты».

Пользователь может переключаться между основными страницами «Комнаты», «Ваши устройства» и «Добавить устройство» и «Сценарии» с помощью вкладок в нижней части экрана.

Со страницы «Комнаты» можно перейти к экранам «Содержимое комнаты» и «Ваши устройства», где будет возможность просматривать список устройств, а также перейти на страницу управления ими. Помимо этого, можно перейти на экран «Добавить комнату», где можно выполнить одноименное действие.

На странице «Добавить устройство» пользователю дается выбор типа устройства для добавления. Нажав на интересующий тип, откроется экран создания устройства.

На странице «Сценарии» отображаются созданные сценарии, время их запуска и состояние. Отсюда можно перейти на страницу создания сценария, нажав кнопку «+». Также при нажатии на сценарий откроется экран редактирования, где можно задать действия для выполнения.

Таким образом, пользовательский интерфейс позволит выполнять все необходимые функции.

5 Реализация программного средства

5.1 Реализация клиентской части приложения

Клиентская часть приложения реализована на языке Dart. Для демонстрации основных элементов приложения была разработана диаграмма компонентов. Разработанная диаграмма компонентов приведена на рисунке 9.

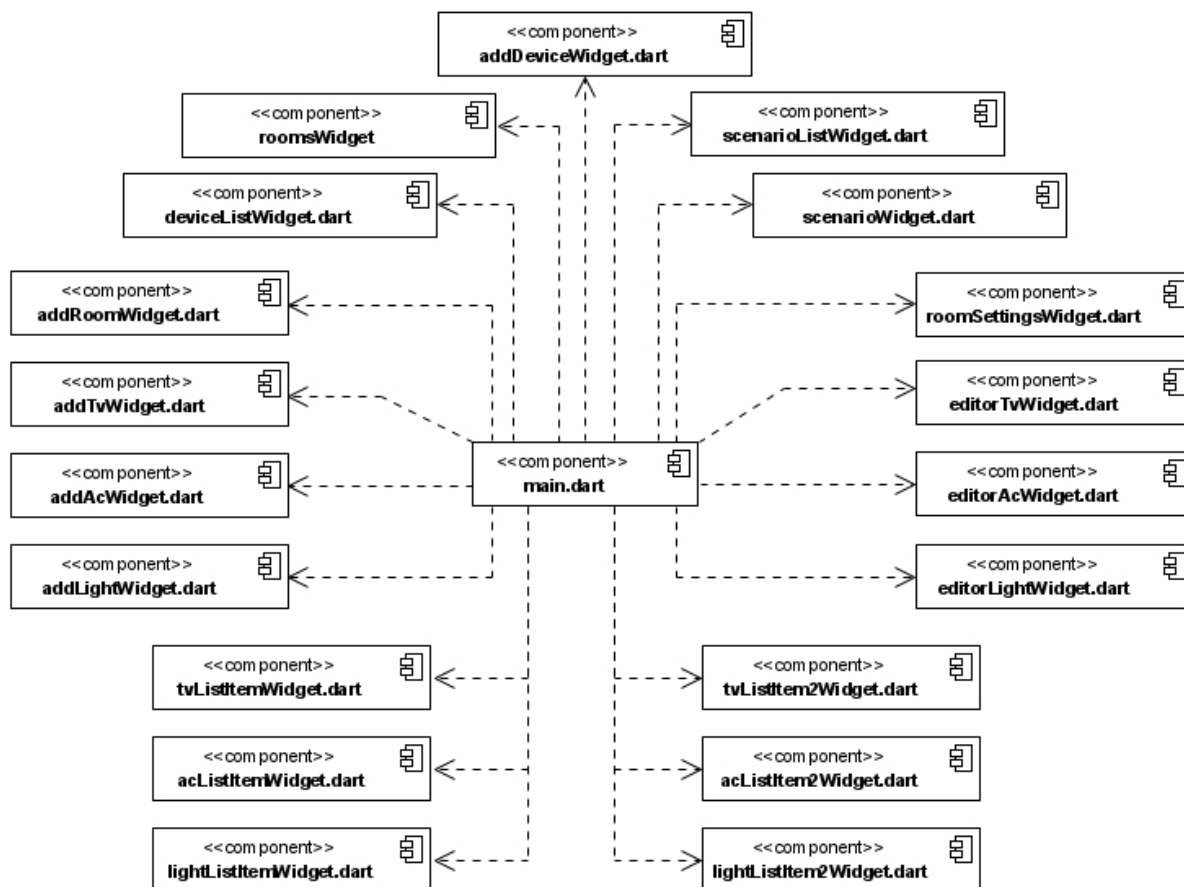


Рисунок 9 – Диаграмма компонентов клиентской части

Главным компонентом является файл `main.dart`. Он содержит в себе панель навигации, из которой можно перейти на другие страницы. Назначение остальных компонентов представлено в таблице 35.

Таблица 35 – Назначение компонентов

№	Название компонента	Назначение
1	<code>deviceListWidget.dart</code>	Страница со списком устройств. Без привязки к комнатам
2	<code>roomsWidget.dart</code>	Страница со списком комнат
3	<code>addDeviceWidget.dart</code>	Страница с выбором типа устройства для добавления

Продолжение таблицы 35

№	Название компонента	Назначение
4	addRoomWidget.dart	Страница добавления комнаты
5	scenarioListWidget.dart	Страница со списком сценариев
6	scenarioWidget.dart	Страница редактирования и запуска сценария
7	addTvWidget.dart	Страница добавления телевизора
8	addAcWidget.dart	Страница добавления кондиционера
9	addLightWidget.dart	Страница добавления освещения
10	roomSettingsWidget.dart	Страница со списком устройств внутри комнаты
11	editorTvWidget.dart	Страница настроек телевизора
12	editorAcWidget.dart	Страница настроек кондиционера
13	editorLightWidget.dart	Страница настроек освещения
14	tvListItemWidget.dart	Виджет, отображающий название телевизора, его комнату и состояние. Отображается в общем списке устройств
15	acListItemWidget.dart	Виджет, отображающий название кондиционера, его комнату и состояние. Отображается в общем списке устройств
16	lightListItemWidget.dart	Виджет, отображающий название освещения, его комнату, цвет и состояние. Отображается в общем списке устройств
17	tvListItem2Widget.dart	Виджет, отображающий название телевизора, его комнату и состояние. Отображается в списке устройств комнаты
18	acListItem2Widget.dart	Виджет, отображающий название кондиционера, его комнату, температуру и состояние. Отображается в списке устройств комнаты
19	lightListItem2Widget.dart	Виджет, отображающий название освещения, его комнату, цвет и состояние. Отображается в списке устройств комнаты

В качестве примера кода приведен класс для взаимодействия с базой данных, а именно с документами коллекции «Rooms». Код представлен в листинге 1.

Листинг 1 – Код класса RoomService

```
class RoomService extends AbstractService {
  static CollectionReference get collection =>
    FirebaseFirestore.instance.collection('Rooms');

  Future<List<RoomsRecord>> getAll() async {
    QuerySnapshot snapshot = await collection.where('created_by', isEqualTo:
      currentUserReference).get();

    List<RoomsRecord> list = [];

    for (final doc in snapshot.docs) {
      list.add(serializers.deserializeWith(RoomsRecord.serializer,
        serializedData(doc)));
    }
  }
}
```

```

    }

    return list;
}

Future<RoomsRecord?> get(DocumentReference ref) async {
    List<RoomsRecord> list = await this.getAll();

    for (var room in list) {
        if (room.reference.id == ref.id) return room;
    }
    return null;
}

Future<DocumentReference<Object?>> create(String name, String icon) {
    final data = createRoomsRecordData(
        name: name,
        icon: icon,
        createdBy: currentUserReference,
    );

    return collection.add(data);
}
}

```

Данный класс содержит статическое поле «collection», которое является объектом коллекции «Rooms» в базе данных Firestore. Через это поле осуществляются все функции класса.

Метод `getAll` предназначен для получения всех документов коллекции из базы данных и конвертации их в объекты класса `RoomsRecord`. Для получения данных сначала используется условие «where», чтобы получить данные только активного пользователя и после этого методом `get` данные непосредственно загружаются из базы данных. Важно отметить, что метод `get` является асинхронным, то есть выполняется параллельно основной программе и поэтому вся функция `getAll` также является асинхронной.

Метод `get` предназначен для получения одного объекта класса `RoomsRecord`. Он вызывает метод `getAll` и ищет объект с идентификатором, указанным в параметре. Если такого объекта нет, то возвращается `null`.

Метод `create` предназначен для создания документа в базе данных. Он создает объект, где указывается название и иконка комнаты и с помощью метода `add` создает документ в БД. В качестве результата возвращает созданный документ.

Аналогичным образом были реализованы и другие сервисы.

Исходные коды приложения представлены в приложении Б.

5.2 Реализация серверной части приложения

Серверная часть приложения была разработана на языке Python и включает в себя следующие компоненты:

- server.py;
- machine.py;
- key.json;
- Firestore.

Диаграмма компонентов серверной части представлена на рисунке 10.

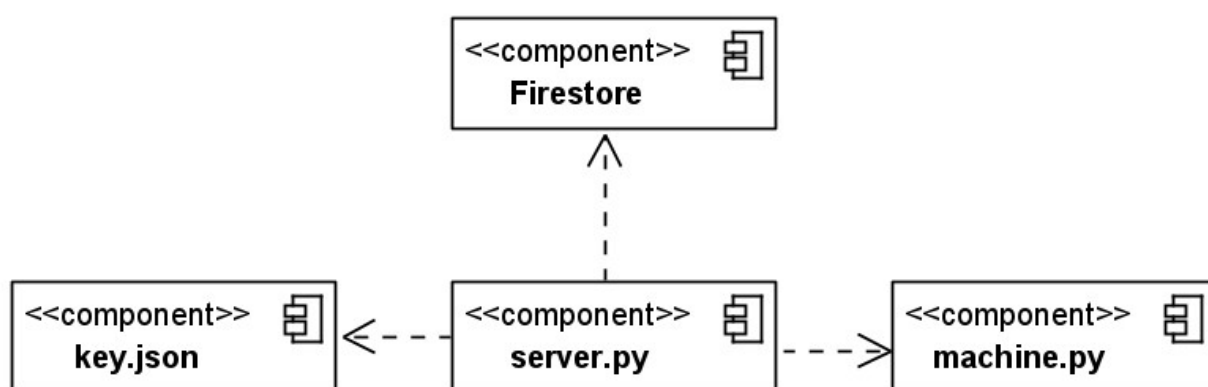


Рисунок 10 – Диаграмма компонентов серверной части

Файл server.py является сервером, который принимает запросы с клиента и выполняет функции по запуску и остановке сценариев, а также прогнозирования выполнения действий пользователем. Для выполнения этих функций серверу необходимо подключение к базе данных Firestore. Настройки для подключения хранятся в файле key.json, который был создан на сайте Firebase. Сервер принимает POST запросы с клиента с помощью функций, представленных в листинге 2.

Листинг 2 – Функции обработки POST запросов

```

@app.route("/start", methods=['POST'])
def start():
    job_id = request.form['id']
    time = request.form['time']
    hours, minutes = map(int, time.split(':'))
    scheduler.add_job(func=lambda: start_scenario(job_id), trigger='cron',
hour=hours, minute=minutes, id=job_id)
  
```

```

return "200"

@app.route("/stop", methods=['POST'])
def stop():
    job_id = request.form['id']
    jobs = scheduler.get_jobs()
    for job in jobs:
        if job.id == job_id:
            scheduler.remove_job(job_id)

return "200"

```

Функция `start` предназначена для запуска сценария. Она имеет декоратор `route`, чтобы указать серверу по какому URL адресу должна выполняться данная функция. В данном случае это «`/start`». При получении такого запроса из тела запроса получаются время и идентификатор сценария, который используется и в качестве идентификатора задачи. Из строкового представления времени считываются значения часов и минут и создается повторяемая задача для запуска сценария, которая выполняется раз в сутки в указанные часы и минуты.

Функция `stop` предназначена для остановки сценария. Она также имеет декоратор `route` с URL адресом «`/stop`». После получения запроса выполняется поиск задачи с указанным идентификатором и если он находится, то задача удаляется.

Файл `machine.py` содержит функции для обучения модели прогнозирования действия пользователя. Обученная модель сохраняется в бинарный файл и загружается на сервер при необходимости.

5.3 Разработка пользовательского интерфейса

При первом входе в приложение пользователь попадает на экран авторизации (рисунок 11), где должен заполнить поля для входа. Если у него нет аккаунта, он может нажать на кнопку «Регистрация» и перейти на страницу регистрации (рисунок 12), где необходимо ввести адрес электронной почты и пароль.

Авторизация

Email
Введите адрес электронной почты

Пароль

Войти

Забыли пароль?

Регистрация

Рисунок 11 – Экран «Авторизация»

Регистрация

Email
Введите адрес электронной почты

Пароль

Зарегистрироваться

Назад

Рисунок 12 – Экран «Регистрация»

В нижней части экрана располагается панель вкладок, откуда можно перейти на три основные страницы: «Ваши устройства», «Комнаты» и

«Добавить устройство» и «Сценарии». После входа в приложение пользователь попадает на страницу «Комнаты», изображенную на рисунке 13.

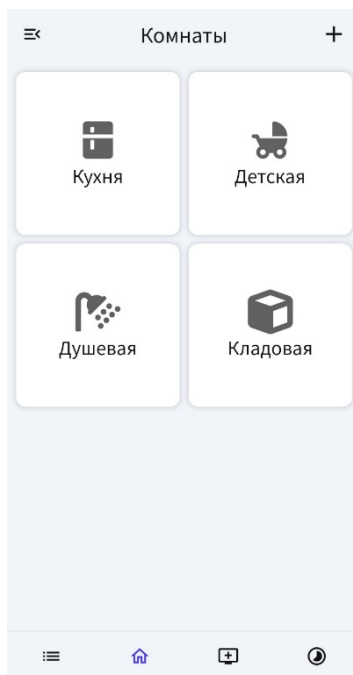


Рисунок 13 – Экран «Комнаты»

В верхнем левом углу находится кнопка для открытия бокового меню, откуда можно изменить тему приложения или выйти из аккаунта. Меню представлено на рисунке 14.



Рисунок 14 – Боковое меню приложения

В верхнем правом углу страницы «Комнаты» расположена кнопка для добавления новой комнаты. Экран добавления комнаты представлен на рисунке 15.

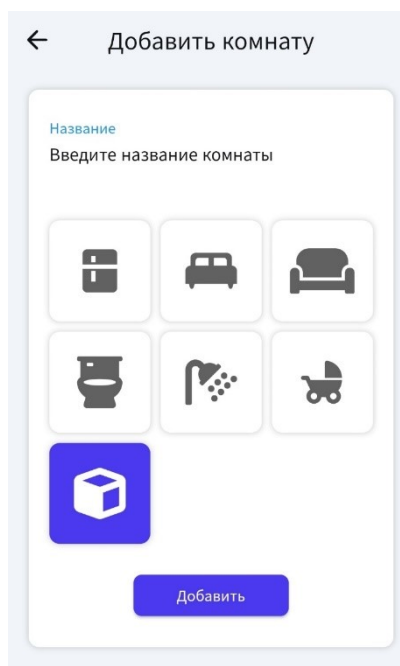


Рисунок 15 – Экран «Добавить комнату»

Нажав на комнату на экране «Комнаты», пользователь переходит на страницу с устройствами выбранной комнаты. Тут отображаются все устройства, находящиеся в данной комнате. Пользователь может изменить название комнаты, нажав на карандаш в правом верхнем углу экрана. Страница комнаты представлена на рисунке 16.

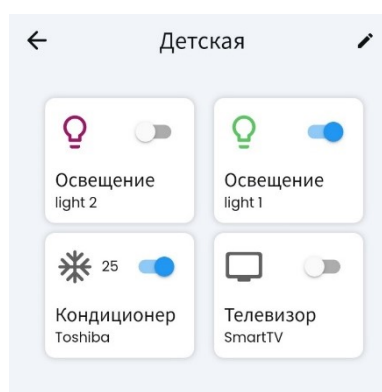


Рисунок 16 – Страница комнаты

На странице «Ваши устройства» отображается список всех добавленных устройств. Отсюда можно быстро включать или выключать устройства, переходить на страницу их настройки, а также удалять их, удерживая палец на выбранном устройстве. Экран представлен на рисунке 17.

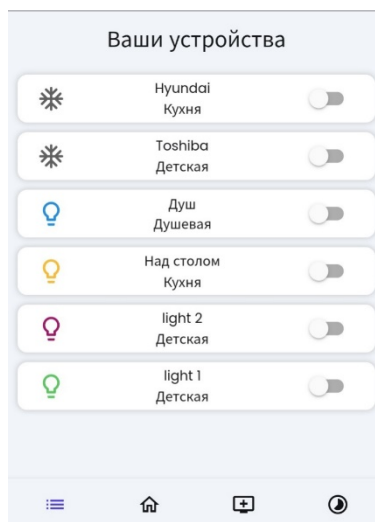


Рисунок 17 – Экран «Ваши устройства»

На странице «Добавить устройство» пользователь выбирает тип устройства для добавления. Экран представлен на рисунке 18.

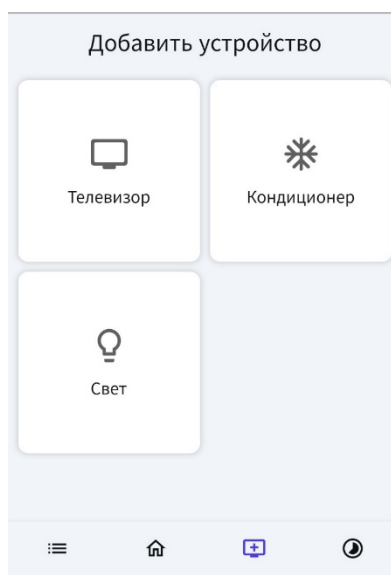


Рисунок 18 – Экран «Добавить устройство»

Выбрав для добавления телевизор, откроется страница, представленная на рисунке 19. Здесь нужно ввести название, канал и выбрать комнату.

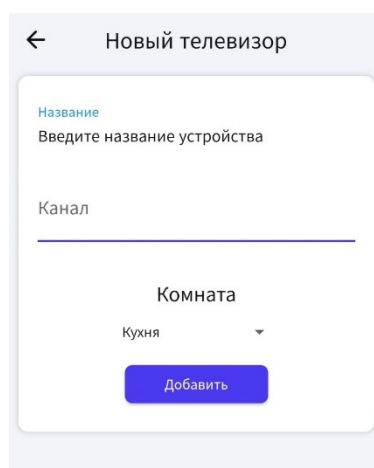


Рисунок 19 – Экран «Новый телевизор»

Выбрав для добавления кондиционер, откроется страница, представленная на рисунке 20. Здесь нужно ввести название, выбрать температуру и комнату.

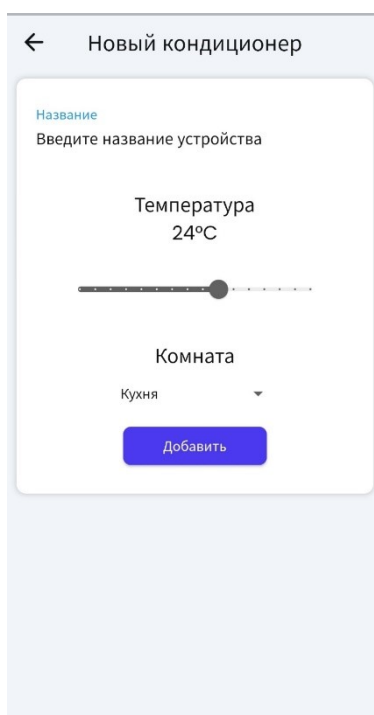


Рисунок 20 – Экран «Новый кондиционер»

Выбрав для добавления свет, откроется страница, представленная на рисунке 21. Здесь нужно ввести название, выбрать цвет, яркость и комнату.

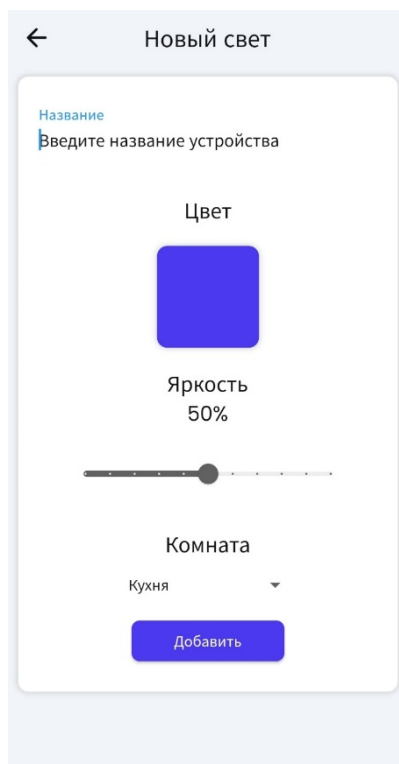


Рисунок 21 – Экран «Новый свет»

Изменить настройки устройств можно нажав на выбранное устройство на страницах «Ваши устройства» и странице комнаты.

На странице изменения телевизора пользователь может включать и выключать устройство, изменять его канал и громкость, заглушать звук, а также менять комнату. Экран настроек телевизора представлен на рисунке 22.

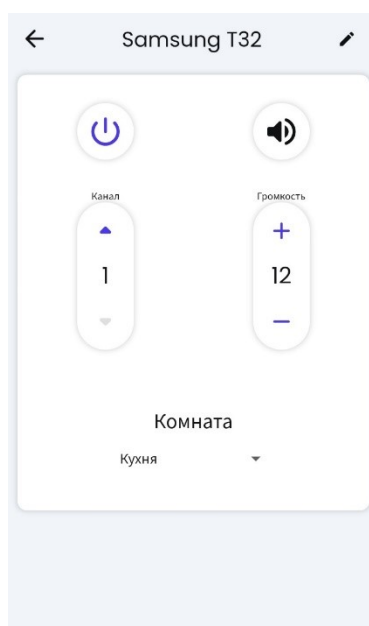


Рисунок 22 – Экран настройки телевизора

На странице изменения кондиционера пользователь может включать и выключать устройство, изменять его температуру, а также менять комнату. Экран настроек кондиционера представлен на рисунке 23.

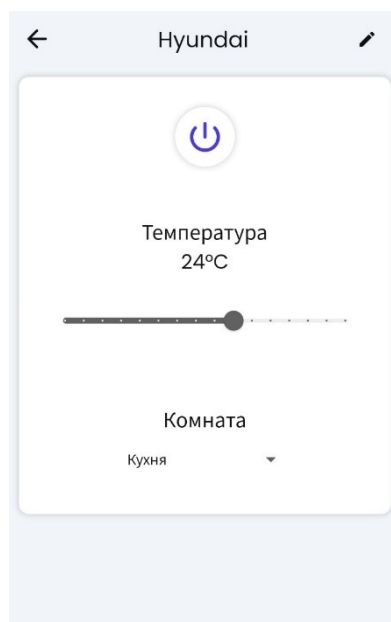


Рисунок 23 – Экран настройки кондиционера

На странице изменения света пользователь может включать и выключать устройство, изменять его цвет и яркость, а также менять комнату. Экран настроек света представлен на рисунке 24.

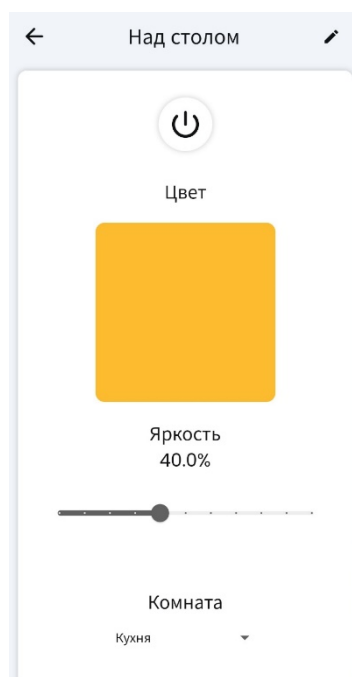


Рисунок 24 – Экран настройки света

Также на всех страницах настроек пользователь может менять названия устройств, нажав на иконку карандаша.

На странице «Сценарии» отображаются все созданные сценарии. Страница представлена на рисунке 25.

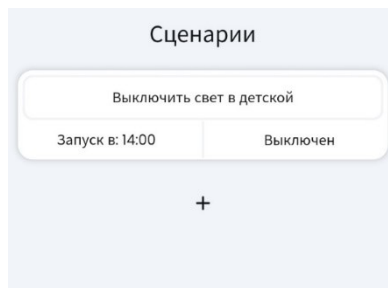


Рисунок 25 – Страница «Сценарии»

Нажав на сценарий, откроется экран со списком действий этого сценария (рисунок 26). Внизу располагаются кнопки для сохранения, запуска и остановки выполнения сценария. Такой же экран открывается и при создании сценария, за исключением того, что в нем не будет кнопок запуска и остановки. Нажав «+» на странице сценария откроется окно добавления действия, где можно выбрать либо ожидание, либо действие (рисунок 27).

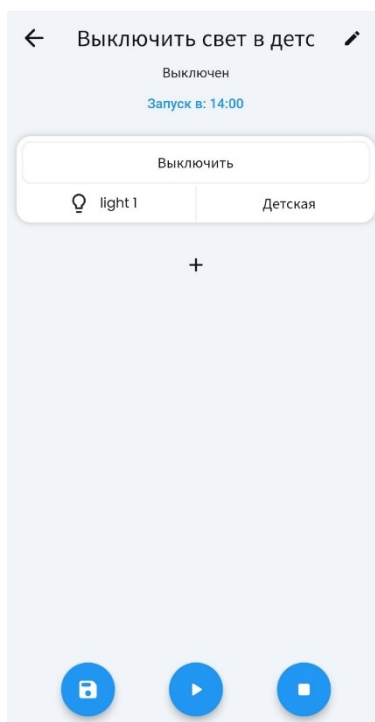


Рисунок 26 – Экран сценария

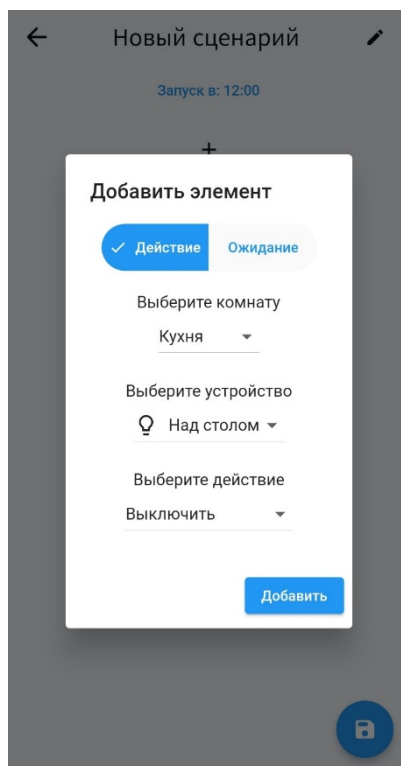


Рисунок 27 – Окно добавления действия

Таким образом, был разработан пользовательский интерфейс приложения, соответствующий всем поставленным требованиям.

6 Тестирование функций взаимодействия с устройствами

Наиболее подходящими вариантами для тестирования функций взаимодействия с устройствами были выбраны: тестирование методом чёрного ящика, тестирование метрики кода и автоматизированное тестирование.

6.1 Тестирование экрана добавления устройства чёрным ящиком

Тестирование методом чёрного ящика – это вид функционального тестирования, при котором система тестируется без информации о её структуре и работы алгоритма. Стратегия чёрного ящика позволяет проверить компоненты системы, интерфейс программы и её работоспособность путём имитации действий пользователя [23].

В тестировании методом чёрного ящика будут проверяться классы эквивалентности, функциональная диаграмма и таблица решений.

Классы эквивалентности позволяют разделить входные тестовые данные на диапазоны или эквивалентные классы. Любое значение из выбранного диапазона должно приводить к одному и тому же результату [24].

Было проведено тестирование классов эквивалентности на примере поля для ввода названия устройства.

Название устройства должно иметь длину от 3 до 20 символов, начинаться с буквы и не содержать специальных символов. Табличное представление классов эквивалентности представлено в таблице 36.

Таблица 36 – Табличное представление классов эквивалентности

	Правильные КЭ	Неправильные КЭ
Длина названия	3–20 (1)	0-2 (2) 21- ∞ (3)
Структура	Начинается с буквы(4)	Начинается не с буквы (5)
Содержание	Не содержит спец. символы (6)	Содержит спец. символы (7)

Графическое представление этих классов эквивалентности представлено на рисунке 28.

Продолжение таблицы 38

Следствия	
C2	Сообщение: «Название должно иметь длину от 3 до 20 символов»
C3	Сообщение: «Введите номер канала»
C4	Сообщение: «Номер канала должен быть больше нуля»
C5	Телевизор добавлен

Для визуализации таблицы причин и следствий, была построена функциональная диаграмма, которая отражает взаимосвязь между входными данными и полученными результатами. Функциональная диаграмма представлена на рисунке 29.

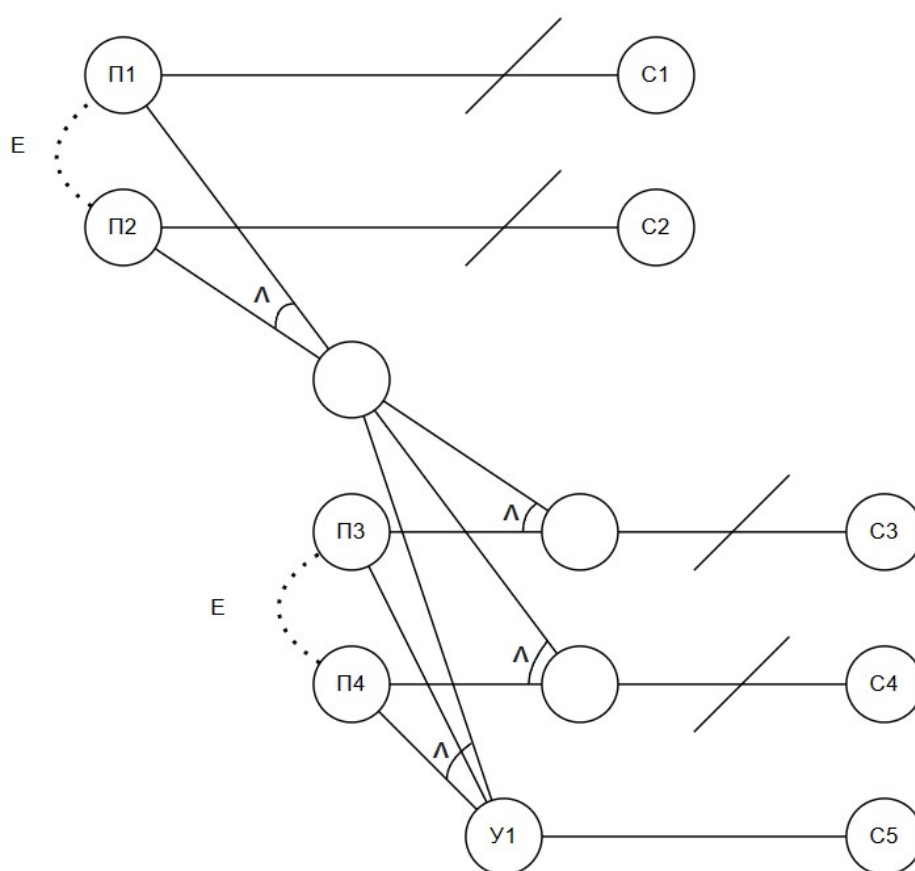


Рисунок 29 – Функциональная диаграмма причин и следствий

Если первая причина не выполняется, то вторая причина выполниться не может, так как нельзя проверить название, если оно не введено. Аналогично четвертая причина не может быть выполнена без выполнения третьей. Также причины проверяются последовательно, то есть сначала проверяется название, а после канал. Поэтому причины 3 и 4 могут выполняться только при выполнении причин 1 и 2. Учитывая всё это, по таблице причин и следствий

была построена таблица решений, которая позволяет проверить поведение системы на разных комбинациях входных данных [25]. Таблица решений представлена в таблице 39.

Таблица 39 – Таблица решений

	1	2	3	4	5	6	7	8	9
П1	0	1	1	0	1	1	0	1	1
П2	0	0	1	0	0	1	0	0	1
П3	0	0	0	1	1	1	1	1	1
П4	0	0	0	0	0	0	1	1	1
С1	1	0	0	1	0	0	1	0	0
С2	0	1	0	0	1	0	0	1	0
С3	0	0	1	0	0	0	0	0	0
С4	0	0	0	0	0	1	0	0	0
С5	0	0	0	0	0	0	0	0	1

По составленной таблице решений были написаны тесты, представленные в таблице 40.

Таблица 40 – Тесты таблицы решений

№	Входные данные	Выходные данные	Результат
1	Название: Пустая строка Канал: Пустая строка	Сообщение: «Введите название»	Тест пройден (рисунок В.1)
2	Название: Sa Канал: Пустая строка	Сообщение: «Название должно иметь длину от 3 до 20 символов»	Тест пройден (рисунок В.2)
3	Название: Samsung Канал: Пустая строка	Сообщение: «Введите номер канала»	Тест пройден (рисунок В.3)
4	Название: Пустая строка Канал: 0	Сообщение: «Введите название»	Тест пройден
5	Название: Sa Канал: 0	Сообщение: «Название должно иметь длину от 3 до 20 символов»	Тест пройден
6	Название: Samsung Канал: 0	Сообщение: «Номер канала должен быть больше нуля»	Тест пройден (рисунок В.4)
7	Название: Пустая строка Канал: 1	Сообщение: «Введите название»	Тест пройден
8	Название: Sa Канал: 1	Сообщение: «Название должно иметь длину от 3 до 20 символов»	Тест пройден
9	Название: Samsung Канал: 1	Телевизор добавлен	Тест пройден

Тестирование методом чёрного ящика показало, что страница добавления телевизора работает правильно и при неверных данных выдает сообщение об ошибке.

6.2 Тестирование метрики кода

В качестве метрики тестирования программного кода была использована метрика Холстеда. Она позволяет оценить сложность программы, исходя из количества операторов и операндов [26].

В качестве примера была подсчитана сложность класса «TvService». Код класса приведен в листинге 3.

Листинг 3 – Код класса «TvService»

```
static CollectionReference get collection =>
FirestoreFirestore.instance.collection('TV');

Future<List<TvRecord>> getAll() async {
  QuerySnapshot snapshot = await collection.where('created_by', isEqualTo:
currentUserReference).get();

  List<TvRecord> list = [];

  for (final doc in snapshot.docs) {
    list.add(serializers.deserializeWith(TvRecord.serializer,
serializedData(doc)));
  }

  return list;
}

Future<TvRecord?> get(DocumentReference ref) async {
  List<TvRecord> list = await this.getAll();

  for (var tv in list) {
    if (tv.reference.id == ref.id) return tv;
  }
  return null;
}

Future<DocumentReference<Object?>> create(String name, int channel,
DocumentReference room) {
  final data = createTvRecordData(
    name: name,
    power: false,
    channel: channel,
    volume: 25,
    isMuted: false,
    room: room,
    createdBy: currentUserReference,
  );
  return collection.add(data);
}
```

Для оценки метрики Холстеда, необходимо подсчитать количество операторов и операндов в коде. Результат подсчета приведен в таблицах 41, 42.

Таблица 41 – Операторы класса

Операторы	Число вхождений
=>	1
{}	5
()	15
;	10
=	1
==	1
if	1
return	4
await	2
for	2
.	12
getAll	2
collection	1
where	1
get	1
add	2
serializers.deserializeWith	1
serializedData	1
createTvRecordData	1
$n_1 = 19$	$N_1 = 64$

Таблица 42 – Операнды класса

Операнды	Число вхождений
collection	4
FirebaseFirestore.instance.collection	1
TvRecord.serializer	1
snapshot	1
snapshot.docs	1
"TV"	2
'created_by'	1
currentUserReference	2
list	5
[]	1
doc	2
tv	2
tv.reference.id	1
ref.id	1
ref	1
name	2
channel	2
room	2

Продолжение таблицы 42

Операнды	Число вхождений
data	2
false	2
25	1
$n_2 = 21$	$N_2 = 37$

На основе этих данных можно рассчитать длину, словарь, объем и сложность программы.

Для определения длины программы, необходимо знать количество использований операторов и операндов. Длина программы рассчитывается по формуле:

$$N = N_1 + N_2,$$

где N_1 – количество использований операторов;

N_2 – количество использований операндов;

N – длина программы.

Таким образом, длина класса равна:

$$N = 64 + 37 = 101.$$

Для определения словаря, необходимо знать количество операторов и операндов. Словарь программы рассчитывается по формуле:

$$n = n_1 + n_2,$$

где n_1 – количество операторов;

n_2 – количество операндов;

n – словарь программы.

Таким образом, словарь класса равен:

$$n = 19 + 21 = 40.$$

Зная длину и словарь программы можно определить её объем. Объем программы рассчитывается по формуле:

$$HV = N * \log_2 n,$$

где N – длина программы;

n – словарь программы;

HV – объем программы.


```

        '3Samsung': false,
        'Sam%sung': false
    };

    map.forEach((value, expected) {
        expect(checkDeviceName(value), expected);
    });
});

test("Тест корректности пароля", () {
    Map<String, bool> map = {
        '2anI-f': true,
        'tT54ki': true,
        'ffffF6': true,
        'aaaAAaa': false,
        'blA': false,
        ' Agas': false,
        '123456': false
    };

    map.forEach((value, expected) {
        expect(checkPassword(value), expected);
    });
});

test("Тест конвертации цвета в HEX строку", () {
    Map<String, Color> map = {
        "#ffffff": Colors.white,
        "#000000": Colors.black,
        "#ff0000": Color.fromARGB(255, 255, 0, 0),
        "#00ff00": Color.fromARGB(255, 0, 255, 0),
        "#0000ff": Color.fromARGB(255, 0, 0, 255),
    };

    map.forEach((expected, value) {
        expect(colorToStringHEX(value), expected);
    });
});

test("Тест конвертации HEX строки в цвет", () {
    Map<String, Color> map = {
        "#ffffff": Colors.white,
        "#000000": Colors.black,
        "#ff0000": Color.fromARGB(255, 255, 0, 0),
        "#00ff00": Color.fromARGB(255, 0, 255, 0),
        "#0000ff": Color.fromARGB(255, 0, 0, 255),
    };

    map.forEach((value, expected) {
        expect(stringToColor(value), expected);
    });
});

```

Во всех тестах входными параметрами являются ключи словаря, а выходными — значения. Входные параметры передаются проверяемым функциям как аргументы, а выходные параметры это ожидаемый результат выполнения функции.

«Тест корректности имени» проверяет функцию «checkDeviceName». Входным параметром является строка с именем.

«Тест корректности пароля» проверяет функцию «checkPassword». Входным параметром является строка с паролем.

«Тест конвертации цвета в HEX строку» проверяет функцию «colorToStringHEX». Входным параметром является цвет с типом Color.

«Тест конвертации HEX строки в цвет» проверяет функцию «stringToColor». Входным параметром является строка, представляющая собой цвет в формате HEX.

Результат выполнения тестов представлен на рисунке 30.

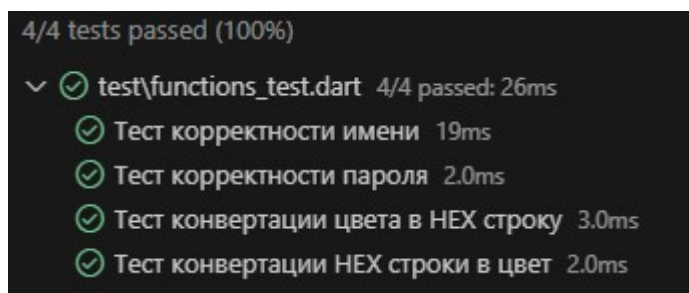


Рисунок 30 – Результат выполнения Unit тестов

Автоматизированное тестирование продемонстрировало, что проверяемые функции приложения работают корректно.

Заключение

В результате выполнения выпускной квалификационной работы было разработано программное средство интеллектуального управления умным домом.

Был проведен анализ предметной и поставлена задача на разработку, а также проведен анализ существующих решений, на основе которых в дальнейшем были определены функциональные требования. Помимо этого, был построен график выполнения работы с помощью диаграммы Ганта, а также посчитана стоимость и сроки разработки проекта.

Также был проведен анализ технологий проектирования и разработки, на основе которого было спроектировано и разработано программное средство. Была описана и обучена модель машинного обучения.

Далее была спроектирована клиентская и серверная часть приложения, база данных и пользовательский интерфейс.

На этапе разработки была разработана клиентская и серверная часть приложения, а также пользовательский интерфейс.

После этого было проведено тестирование функций программы методом черного ящика и метрикой Холстеда, а также выполнено автоматизированное тестирование с помощью Unit-тестов.

Список использованных источников

1. Система «умный дом»: из чего состоит, как работает и чем может быть полезна // VC.RU. [Электронный ресурс]. URL: <https://vc.ru/u/896657-dverprof/496967-sistema-umnyy-dom-iz-chego-sostoit-kak-rabotaet-i-chem-mozhet-byt-polezna> (дата обращения: 12.04.2023).
2. Выбор протокола для Умного Дома. Wi-Fi, Bluetooth, ZigBee или Z-Wave // smarthomegadget. [Электронный ресурс]. URL: <https://smarthomegadget.ru/vybor-protokola-dlya-umnogo-doma-wi-fi-bluetooth-zigbee-ili-z-wave/> (дата обращения: 12.04.2023).
3. Из чего собрать умный дом в 2020 году: от хаба и до лампочки // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/companies/mvideo/articles/499706/> (дата обращения: 12.04.2023).
4. 8 лучших приложений для смартфона по управлению Умным домом // amperika. [Электронный ресурс]. URL: <https://amperika.com/prilozhenie-umnyj-dom/> (дата обращения: 13.04.2023).
5. Встраиваем своё устройство «Умного дома» в экосистему SmartThings // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/companies/samsung/articles/489834/> (дата обращения: 13.04.2023).
6. Умный дом с Алисой // yandex. [Электронный ресурс]. URL: <https://yandex.uz/alice/smart-home> (дата обращения: 13.04.2023).
7. Диаграмма вариантов использования (Use Case Diagram) // itonboard. [Электронный ресурс]. URL: https://itonboard.ru/analysis/629-diagramma_variantov_ispolzovanija_use_case_diagram/ (дата обращения: 17.04.2023).
8. Все о Диаграмме Ганта: Как Успешно и Вовремя Завершать Проекты // ganttpro. [Электронный ресурс]. URL:

<https://blog.ganttpro.com/ru/diagramma-gantta-gantt-chart/#preimuschestva-diagrammy-ganta> (дата обращения: 17.04.2023).

9. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – 2-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2006. – 496 с.

10. Visual Paradigm for UML 8.0 // analyst. [Электронный ресурс]. URL: <http://analyst.by/articles/vpforuml8> (дата обращения: 18.04.2023).

11. Visual Paradigm (бессрочная лицензия), Enterprise Floating в Пензе // softline. [Электронный ресурс]. URL: <https://store.softline.ru/visual-paradigm/-41171/#description> (дата обращения: 18.04.2023).

12. Dart // skillfactory. [Электронный ресурс]. URL: <https://blog.skillfactory.ru/glossary/dart/> (дата обращения: 18.04.2023);

13. Dart overview // dart. [Электронный ресурс]. URL: <https://dart.dev/overview> (дата обращения: 18.04.2023).

14. Что такое Flutter и зачем его изучать? // medium. [Электронный ресурс]. URL: <https://medium.com/nuances-of-programming/что-такое-flutter-и-зачем-его-изучать-8d72475b1ba2> (дата обращения: 18.04.2023).

15. Что такое Firebase от Google? // back4app. [Электронный ресурс]. URL: https://blog.back4app.com/ru/что-такое-firebase/#_FirebaseGoogle (дата обращения: 18.04.2023).

16. Облако Firestore // firebase. [Электронный ресурс]. URL: <https://firebase.google.com/docs/firestore?hl=ru> (дата обращения: 18.04.2023).

17. Модель данных Cloud Firestore // firebase. [Электронный ресурс]. URL: <https://firebase.google.com/docs/firestore/data-model?hl=ru%5D>. (дата обращения: 18.04.2023).

18. Руководство для начинающих по архитектуре Flutter // morioh. [Электронный ресурс]. URL: <https://morioh.com/p/13cd8ebbd34a> (дата обращения: 21.04.2023).

19. Новиков, Ф. А. Учебно-методическое пособие по дисциплине «Анализ и проектирование на UML» / Ф.А. Новиков. – СПб.: Изд-во УИТМО, 2007. – 286 с.

20. Логические модели данных // myfilology. [Электронный ресурс]. URL: <https://myfilology.ru/165/bazy-dannyx/logicheskie-modeli-dannyx/> (дата обращения: 01.05.2023).
21. Почему дизайн интерфейсов (UI) так важен при создании интернет-магазинов? // experrto. [Электронный ресурс]. URL: <https://ru.experrto.io/blog/2020/09/30/pochemu-dizajn-interfejsov-ui-tak-vazhen-pri-sozdanii-internet-magazinov/#vazhnost-ui-dizajna-v-sfere-ehlektronnoj-kommercii> (дата обращения: 02.05.2023).
22. Темная тема – просто тренд или реальный инструмент для повышения вовлеченности пользователей // vc. [Электронный ресурс]. URL: <https://vc.ru/services/439748-temnaya-tema-prosto-trend-ili-realnyy-instrument-dlya-povysheniya-vovlechenosti-polzovateley> (дата обращения: 03.05.2023).
23. Copeland, L. A Practitioner's Guide to Software Test Design / Copeland Lee. – Artech House Publishers London, 2004. – 355с.
24. Что такое классы эквивалентности? // testengineer [Электронный ресурс]. / URL: <https://testengineer.ru/equivalence-classes-guide/?ysclid=lfmvkwkw5d255751647#equivalence-classes> (дата обращения: 17.05.2023).
25. Тестирование таблицы решений// CoderLessons.com. [Электронный ресурс]. / URL: <https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/ruchnoe-testirovanie/testirovanie-tablitsy-reshenii?ysclid=lfmwniexgu320648044> (дата обращения: 18.05.2023).
26. Введение в сложность Холстеда // Русские Блоги [Электронный ресурс]. / URL: <https://russianblogs.com/article/421210007/> (дата обращения: 20.05.2023).
27. Хориков, В. Принципы юнит-тестирования / В. Хориков. – СПб.: Питер, 2021 – 320 с.: ил. – (Серия «Для профессионалов»).

ПРИЛОЖЕНИЕ А
ГЛОССАРИЙ
(справочное)

Wi-Fi – технология беспроводной локальной сети.

Bluetooth – это беспроводная технология передачи данных на короткие расстояния между устройствами.

ZigBee – это беспроводной протокол связи для сетей малой зоны, используемый в системах управления освещением, умных домах, промышленности и т.д.

Z-Wave является запатентованным беспроводным протоколом связи, разработанным для домашней автоматизации, в частности для контроля и управления в жилых и коммерческих объектах.

ПО – программное обеспечение.

IT – информационные технологии.

Фреймворк – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

SDK – это набор инструментов для разработки программного обеспечения в одном устанавливаемом пакете.

NoSQL – это подход к хранению и обработке данных, который не использует реляционную модель и SQL-запросы.

HEX – шестнадцатеричная система представления цвета, которая использует арабские десятичные цифры от 0 до 9 и латинские буквы от A до F.

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЕ КОДЫ ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг Б.1 – Файл main.dart

```

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await initFirebase();

  await FlutterTheme.initialize();

  final appState = FFAppState();

  runApp(ChangeNotifierProvider(
    create: (context) => appState,
    child: MyApp(),
  ));
}

class MyApp extends StatefulWidget {
  @override
  State<MyApp> createState() => _MyAppState();

  static _MyAppState of(BuildContext context) =>
  context.findAncestorStateOfType<_MyAppState>()!;
}

class _MyAppState extends State<MyApp> {
  ThemeMode _themeMode = FlutterTheme.themeMode;

  late Stream<BaseAuthUser> userStream;

  late AppStateNotifier _appStateNotifier;
  late GoRouter _router;

  final authUserSub = authenticatedUserStream.listen((_) {});

  @override
  void initState() {
    super.initState();
    _appStateNotifier = AppStateNotifier();
    _router = createRouter(_appStateNotifier);
    userStream = smartHomeFirebaseUserStream()..listen((user) =>
    _appStateNotifier.update(user));
    jwtTokenStream.listen((_) {});
    Future.delayed(
      Duration(seconds: 1),
      () => _appStateNotifier.stopShowingSplashImage(),
    );
  }

  @override
  void dispose() {
    authUserSub.cancel();
    super.dispose();
  }

  void setThemeMode(ThemeMode mode) => setState(() {
    _themeMode = mode;
    FlutterTheme.saveThemeMode(mode);
  });

  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(

```

```

        title: 'SmartHome',
        localizationsDelegates: [
            FFLocalizationsDelegate(),
            GlobalMaterialLocalizations.delegate,
            GlobalWidgetsLocalizations.delegate,
            GlobalCupertinoLocalizations.delegate,
        ],
        supportedLocales: const [Locale('en', '')],
        theme: ThemeData(brightness: Brightness.light),
        darkTheme: ThemeData(brightness: Brightness.dark),
        themeMode: _themeMode,
        routeInformationParser: _router.routeInformationParser,
        routerDelegate: _router.routerDelegate,
    );
}
}

class NavBarPage extends StatefulWidget {
  NavBarPage({Key? key, this.initialPage, this.page}) : super(key: key);

  final String? initialPage;
  final Widget? page;

  @override
  _NavBarPageState createState() => _NavBarPageState();
}

class _NavBarPageState extends State<NavBarPage> {
  String _currentPageName = 'Rooms';
  late Widget? _currentPage;

  @override
  void initState() {
    super.initState();
    _currentPageName = widget.initialPage ?? _currentPageName;
    _currentPage = widget.page;
  }

  @override
  Widget build(BuildContext context) {
    final tabs = {
      'DeviceList': DeviceListWidget(),
      'Rooms': RoomsWidget(),
      'SelectDeviceForAdd': SelectDeviceForAddWidget()
    };
    final currentIndex = tabs.keys.toList().indexOf(_currentPageName);
    return Scaffold(
      body: _currentPage ?? tabs[_currentPageName],
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: currentIndex,
        onTap: (i) => setState(() {
          _currentPage = null;
          _currentPageName = tabs.keys.toList()[i];
        }),
        backgroundColor: FlutterTheme.of(context).primaryBackground,
        selectedItemColor: FlutterTheme.of(context).primary,
        unselectedItemColor: FlutterTheme.of(context).primaryText,
        showSelectedLabels: false,
        showUnselectedLabels: false,
        type: BottomNavigationBarType.fixed,
        items: <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(
              Icons.list,

```

```

        size: 24.0,
      ),
      label: 'Устройства',
    ),
    BottomNavigationBarItem(
      icon: Icon(
        Icons.home_outlined,
        size: 24.0,
      ),
      label: 'Home',
    ),
    BottomNavigationBarItem(
      icon: Icon(
        Icons.add_to_queue,
        size: 24.0,
      ),
      label: 'Добавить',
    ),
  ],
),
);
}
}

```

Листинг Б.2 – Файл rooms_model.dart

```

class RoomsModel extends Model {
  bool? checkRoom;
  late SideMenuModel sideMenuModel;

  void initState(BuildContext context) {
    sideMenuModel = createModel(context, () => SideMenuModel());
  }

  void dispose() {
    sideMenuModel.dispose();
  }
}

```

Листинг Б.3 – Файл rooms_widget.dart

```

class RoomsWidget extends StatefulWidget {
  const RoomsWidget({Key? key}) : super(key: key);

  @override
  _RoomsWidgetState createState() => _RoomsWidgetState();
}

class _RoomsWidgetState extends State<RoomsWidget> {
  late RoomsModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();
  final _unfocusNode = FocusNode();

  @override
  void initState() {
    super.initState();
    _model = createModel(context, () => RoomsModel());
  }
}

```

```

@override
void dispose() {
  _model.dispose();

  _unfocusNode.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  context.watch<FFAppState>();

  return GestureDetector(
    onTap: () => FocusScope.of(context).requestFocus(_unfocusNode),
    child: Scaffold(
      key: scaffoldKey,
      backgroundColor: FlutterTheme.of(context).primaryBackground,
      drawer: Drawer(
        elevation: 16.0,
        child: wrapWithModel(
          model: _model.sideMenuModel,
          updateCallback: () => setState(() {}),
          child: SideMenuWidget(),
        ),
      ),
      appBar: PreferredSize(
        preferredSize: Size.fromHeight(60.0),
        child: AppBar(
          backgroundColor: Color(0x004B39EF),
          automaticallyImplyLeading: false,
          leading: InkWell(
            splashColor: Colors.transparent,
            focusColor: Colors.transparent,
            hoverColor: Colors.transparent,
            highlightColor: Colors.transparent,
            onTap: () async {
              scaffoldKey.currentState!.openDrawer();
            },
            child: Icon(
              Icons.menu_open_rounded,
              color: FlutterTheme.of(context).primaryText,
              size: 24.0,
            ),
          ),
          title: Text(
            'Комнаты',
            style: FlutterTheme.of(context).headlineMedium.override(
              fontFamily: 'Poppins',
              color: FlutterTheme.of(context).primaryText,
              fontSize: 22.0,
            ),
          ),
          actions: [
            CustomIconButton(
              borderColor: Colors.transparent,
              borderRadius: 30.0,
              borderWidth: 1.0,
              buttonSize: 60.0,
              icon: Icon(
                Icons.add,
                color: FlutterTheme.of(context).primaryText,
                size: 30.0,
              ),
              onPressed: () async {

```

```

        context.pushNamed('addRoom');
      },
    ),
  ],
  centerTitle: true,
  elevation: 0.0,
),
),
body: SafeArea(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.max,
    children: [
      Expanded(
        child: Padding(
          padding: EdgeInsetsDirectional.fromSTEB(10.0, 10.0, 10.0,
10.0),
          child: StreamBuilder<List<RoomsRecord>>(
            stream: queryRoomsRecord(
              queryBuilder: (roomsRecord) =>
roomsRecord.where('created_by', isEqualTo: currentUserReference),
            ),
            builder: (context, snapshot) {
              if (!snapshot.hasData) {
                return LoadingWidget();
              }
              List<RoomsRecord> gridViewRoomsRecordList =
snapshot.data!;
              return GridView.builder(
                padding: EdgeInsets.zero,
                gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                  crossAxisCount: 2,
                  crossAxisSpacing: 10.0,
                  mainAxisSpacing: 10.0,
                  childAspectRatio: 1.0,
                ),
                scrollDirection: Axis.vertical,
                itemCount: gridViewRoomsRecordList.length,
                itemBuilder: (context, gridViewIndex) {
                  final gridViewRoomsRecord =
gridViewRoomsRecordList[gridViewIndex];
                  return InkWell(
                    onTap: () async {
                      context.pushNamed(
                        'RoomSettings',
                        queryParams: {
                          'room': serializeParam(
                            gridViewRoomsRecord,
                            ParamType.Document,
                          ),
                        }.withoutNulls,
                      extra: <String, dynamic>{
                        'room': gridViewRoomsRecord,
                      },
                    );
                  },
                    onLongPress: () async {
                      var confirmDialogResponse = await
showDialog<bool>(
                        context: context,
                        builder: (alertDialogContext) {
                          return AlertDialog(
                            title: Text('Удалить комнату?'),
                            content:
Text(gridViewRoomsRecord.name!),

```

```

        actions: [
          TextButton(
            onPressed: () =>
Navigator.pop(alertDialogContext, false),
            child: Text('Нет'),
          ),
          TextButton(
            onPressed: () =>
Navigator.pop(alertDialogContext, true),
            child: Text('Да'),
          ),
        ],
      );
    },
  ) ??
  false;
  if (confirmDialogResponse) {
    _model.checkRoom = await checkRoomBool(
      gridViewRoomsRecord.reference,
    );
    if (_model.checkRoom == true) {
      await gridViewRoomsRecord.reference.delete();
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(
            'Комната удалена',
            style: TextStyle(
              color:
FlutterTheme.of(context).primaryText,
            ),
          ),
          duration: Duration(milliseconds: 4000),
          backgroundColor:
FlutterTheme.of(context).secondary,
        ),
      );
    } else {
      await showDialog(
        context: context,
        builder: (alertDialogContext) {
          return AlertDialog(
            title: Text('Невозможно удалить
комнату'),
            content: Text(
              'Комната не пустая. Для удаления
переместите устройства и повторите попытку'),
            actions: [
              TextButton(
                onPressed: () =>
Navigator.pop(alertDialogContext),
                child: Text('Ок'),
              ),
            ],
          );
        },
      );
    }
  }

  setState(() {});
},
child: SelectRoomWidget(
  key:
Key('Key31d_${gridViewIndex}_of_${gridViewRoomsRecordList.length}'),

```

```

        icon: gridViewRoomsRecord.icon,
        name: gridViewRoomsRecord.name,
      ),
    );
  },
);
),
),
),
],
),
),
);
}
}

```

Листинг Б.4 – Файл room_settings_model.dart

```

class RoomSettingsModel extends Model {
  late DeviceNameFieldModel deviceNameFieldModel;

  void initState(BuildContext context) {
    deviceNameFieldModel = createModel(context, () => DeviceNameFieldModel());
  }

  void dispose() {
    deviceNameFieldModel.dispose();
  }
}

```

Листинг Б.5 – Файл room_settings_widget.dart

```

class RoomSettingsWidget extends StatefulWidget {
  const RoomSettingsWidget({
    Key? key,
    this.room,
  }) : super(key: key);

  final RoomsRecord? room;

  @override
  _RoomSettingsWidgetState createState() => _RoomSettingsWidgetState();
}

class _RoomSettingsWidgetState extends State<RoomSettingsWidget> {
  late RoomSettingsModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();
  final _unfocusNode = FocusNode();

  @override
  void initState() {
    super.initState();
    _model = createModel(context, () => RoomSettingsModel());

    SchedulerBinding.instance.addPostFrameCallback((_) async {
      setState(() {
        FFAppState().editActive = false;
      });
    });
  }
}

```

```

    });
}

@override
void dispose() {
  _model.dispose();

  _unfocusNode.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  context.watch<FFAppState>();

  return StreamBuilder<RoomsRecord>(
    stream: RoomsRecord.getDocument(widget.room!.reference),
    builder: (context, snapshot) {
      if (!snapshot.hasData) {
        return LoadingWidget();
      }
      final roomSettingsRoomsRecord = snapshot.data!;
      return GestureDetector(
        onTap: () => FocusScope.of(context).requestFocus(_unfocusNode),
        child: WillPopScope(
          onWillPop: () async => false,
          child: Scaffold(
            key: scaffoldKey,
            backgroundColor: FlutterTheme.of(context).primaryBackground,
            appBar: AppBar(
              backgroundColor: Color(0x004B39EF),
              automaticallyImplyLeading: false,
              leading: CustomIconButton(
                borderColor: Colors.transparent,
                borderRadius: 30.0,
                borderWidth: 1.0,
                buttonSize: 60.0,
                icon: Icon(
                  Icons.arrow_back_rounded,
                  color: FlutterTheme.of(context).primaryText,
                  size: 30.0,
                ),
                onPressed: () async {
                  context.pop();
                },
              ),
              title: wrapWithModel(
                model: _model.deviceNameFieldModel,
                updateCallback: () => setState(() {}),
                child: DeviceNameFieldWidget(
                  name: roomSettingsRoomsRecord.name,
                ),
              ),
              actions: [
                Padding(
                  padding: EdgeInsetsDirectional.fromSTEB(0.0, 0.0, 10.0,
0.0),
                  child: ToggleIcon(
                    onPressed: () async {
                      setState(() => FFAppState().editActive =
!FFAppState().editActive);
                      if (!FFAppState().editActive) {
                        if
(functions.checkDeviceName(_model.deviceNameFieldModel.textController.text)) {

```

```

        final roomsUpdateData = createRoomsRecordData(
            name:
_model.deviceNameFieldModel.textController.text,
        );
        await
roomSettingsRoomsRecord.reference.update(roomsUpdateData);
    } else {
        ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(
                content: Text(
                    FFAppState().nameCheck,
                    style: TextStyle(
                        color: Colors.white,
                    ),
                ),
                duration: Duration(milliseconds: 4000),
                backgroundColor: FlutterTheme.of(context).error,
            ),
        );
        setState(() {
            FFAppState().editActive = true;
        });
    }
},
value: FFAppState().editActive,
onIcon: Icon(
    Icons.done,
    color: FlutterTheme.of(context).primaryText,
    size: 25.0,
),
offIcon: Icon(
    Icons.mode_edit,
    color: FlutterTheme.of(context).primaryText,
    size: 20.0,
),
),
),
],
centerTitle: true,
elevation: 0.0,
),
body: SafeArea(
    child: Padding(
        padding: EdgeInsetsDirectional.fromSTEB(20.0, 20.0, 20.0,
20.0),
        child: Column(
            mainAxisAlignment: MainAxisAlignment.max,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
                Expanded(
                    child: StreamBuilder<RoomsRecord>(
                        stream:
RoomsRecord.getDocument(widget.room!.reference),
                        builder: (context, snapshot) {
                            if (!snapshot.hasData) {
                                return LoadingWidget();
                            }
                        },
                    ),
                ),
                return SingleChildScrollView(
                    child: Column(
                        mainAxisAlignment: MainAxisAlignment.max,
                        children: [
                            Align(
                                alignment: AlignmentDirectional(0.0, 0.0),

```

```

child: Wrap(
  spacing: 0.0,
  runSpacing: 0.0,
  alignment: WrapAlignment.start,
  crossAxisAlignment:
WrapCrossAlignment.start,

  direction: Axis.horizontal,
  runAlignment: WrapAlignment.start,
  verticalDirection: VerticalDirection.down,
  clipBehavior: Clip.none,
  children: [
    StreamBuilder<List<LightRecord>>(
      stream: queryLightRecord(
        queryBuilder: (lightRecord) =>
lightRecord
        .where('room', isEqualTo:
widget.room!.reference)
        .where('created_by', isEqualTo:
currentUserReference),
      ),
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return LoadingWidget();
        }
        List<LightRecord>
wrapLightRecordList = snapshot.data!;
        return Wrap(
          children:
List.generate(wrapLightRecordList.length, (wrapIndex) {
          final wrapLightRecord =
wrapLightRecordList[wrapIndex];
          return LightListItem2Widget(
            key:
Key('Keypnl_${wrapIndex}_of_${wrapLightRecordList.length}'),
            ref: wrapLightRecord,
          );
        })),
      );
    ),
    StreamBuilder<List<AcRecord>>(
      stream: queryAcRecord(
        queryBuilder: (acRecord) => acRecord
        .where('room', isEqualTo:
widget.room!.reference)
        .where('created_by', isEqualTo:
currentUserReference),
      ),
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return LoadingWidget();
        }
        List<AcRecord> wrapAcRecordList =
snapshot.data!;
        return Wrap(
          children:
List.generate(wrapAcRecordList.length, (wrapIndex) {
          final wrapAcRecord =
wrapAcRecordList[wrapIndex];
          return ACLListItem2Widget(
            key:
Key('Key5ox_${wrapIndex}_of_${wrapAcRecordList.length}'),
            ref: wrapAcRecord,
          );

```



```

        passwordVisibility = false;
    }

    void dispose() {
        emailController?.dispose();
        passwordController?.dispose();
    }
}

```

Листинг Б.7 – Файл register_page_widget.dart

```

class RegisterPageWidget extends StatefulWidget {
    const RegisterPageWidget({Key? key}) : super(key: key);

    @override
    _RegisterPageWidgetState createState() => _RegisterPageWidgetState();
}

class _RegisterPageWidgetState extends State<RegisterPageWidget> {
    late RegisterPageModel _model;

    final scaffoldKey = GlobalKey<ScaffoldState>();
    final _unfocusNode = FocusNode();

    @override
    void initState() {
        super.initState();
        _model = createModel(context, () => RegisterPageModel());

        _model.emailController ??= TextEditingController();
        _model.passwordController ??= TextEditingController();
    }

    @override
    void dispose() {
        _model.dispose();

        _unfocusNode.dispose();
        super.dispose();
    }

    @override
    Widget build(BuildContext context) {
        context.watch<FFAppState>();

        return GestureDetector(
            onTap: () => FocusScope.of(context).requestFocus(_unfocusNode),
            child: WillPopScope(
                onWillPop: () async => false,
                child: Scaffold(
                    key: scaffoldKey,
                    backgroundColor: FlutterTheme.of(context).primaryBackground,
                    body: SafeArea(
                        child: Padding(
                            padding: EdgeInsetsDirectional.fromSTEB(30.0, 30.0, 30.0, 30.0),
                            child: Column(
                                mainAxisAlignment: MainAxisAlignment.max,
                                mainAxisSize: MainAxisSize.max,
                                mainAxisAlignment: MainAxisAlignment.center,
                                children: [
                                    Padding(
                                        padding: EdgeInsetsDirectional.fromSTEB(0.0, 0.0, 0.0,
30.0),

```

```

        child: Text(
          'Регистрация',
          style: FlutterTheme.of(context).headlineMedium.override(
            fontFamily: 'Poppins',
            color: FlutterTheme.of(context).primaryText,
            fontSize: 32.0,
          ),
        ),
      ),
    ),
    Padding(
      padding: EdgeInsetsDirectional.fromSTEB(0.0, 0.0, 0.0,
20.0),
      child: TextFormField(
        controller: _model.emailController,
        autofocus: true,
        obscureText: false,
        decoration: InputDecoration(
          labelText: 'Email',
          hintText: 'Введите адрес электронной почты',
          hintStyle: FlutterTheme.of(context).bodySmall,
        ),
        style: FlutterTheme.of(context).bodyMedium,
        keyboardType: TextInputType.emailAddress,
        validator:
_model.emailControllerValidator.asValidator(context),
      ),
    ),
    TextFormField(
      controller: _model.passwordController,
      autofocus: true,
      obscureText: !_model.passwordVisibility,
      decoration: InputDecoration(
        labelText: 'Пароль',
        hintText: 'Введите пароль',
        hintStyle: FlutterTheme.of(context).bodySmall,
        suffixIcon: InkWell(
          onTap: () => setState(
            () => _model.passwordVisibility =
!_model.passwordVisibility,
          ),
          focusNode: FocusNode(skipTraversal: true),
          child: Icon(
            _model.passwordVisibility ? Icons.visibility_outlined
: Icons.visibility_off_outlined,
            color: FlutterTheme.of(context).accent1,
            size: 22,
          ),
        ),
      ),
      style: FlutterTheme.of(context).bodyMedium,
      keyboardType: TextInputType.visiblePassword,
      validator:
_model.passwordControllerValidator.asValidator(context),
    ),
    Padding(
      padding: EdgeInsetsDirectional.fromSTEB(0.0, 30.0, 0.0,
0.0),
      child: FFButtonWidget(
        onPressed: () async {
          if (checkPassword(_model.passwordController.text)) {
            GoRouter.of(context).prepareAuthEvent();

            final user = await authManager.createAccountWithEmail(
              context,

```

```

        _model.emailController.text,
        _model.passwordController.text,
    );
    if (user == null) {
        return;
    }

    context.goNamedAuth('Rooms', mounted);
} else
    ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
            content: Text(
                "Пароль должен иметь минимум 6 символов, 1
цифру, 1 символ верхнего и нижнего регистра",
                style: TextStyle(
                    color: Colors.white,
                ),
            ),
            duration: Duration(milliseconds: 4000),
            backgroundColor: FlutterTheme.of(context).error,
        ),
    );
},
text: 'Зарегистрироваться',
options: FFButtonOptions(
    width: double.infinity,
    height: 40.0,
    padding: EdgeInsetsDirectional.fromSTEB(0.0, 0.0, 0.0,
0.0),
    iconPadding: EdgeInsetsDirectional.fromSTEB(0.0, 0.0,
0.0, 0.0),

    color: FlutterTheme.of(context).primary,
    textStyle: FlutterTheme.of(context).titleSmall.override(
        fontFamily: 'Poppins',
        color: Colors.white,
    ),
    borderSide: BorderSide(
        color: Colors.transparent,
        width: 1.0,
    ),
    borderRadius: BorderRadius.circular(8.0),
),
),
),
Padding(
    padding: EdgeInsetsDirectional.fromSTEB(30.0, 30.0, 30.0,
30.0),

    child: InkWell(
        splashColor: Colors.transparent,
        focusColor: Colors.transparent,
        hoverColor: Colors.transparent,
        highlightColor: Colors.transparent,
        onTap: () async {
            context.safePop();
        },
        child: Text(
            'Назад',
            style: FlutterTheme.of(context).bodyMedium,
        ),
    ),
),
],
),
),
),

```

```

        ),
      ),
    ),
  );
}
}

```

Листинг Б.8 – Файл `device_list_model.dart`

```

class DeviceListModel extends Model {
  late SideMenuModel sideMenuModel;

  void initState(BuildContext context) {
    sideMenuModel = createModel(context, () => SideMenuModel());
  }

  void dispose() {
    sideMenuModel.dispose();
  }
}

```

Листинг Б.9 – Файл `device_list_widget.dart`

```

class DeviceListWidget extends StatefulWidget {
  const DeviceListWidget({Key? key}) : super(key: key);

  @override
  _DeviceListWidgetState createState() => _DeviceListWidgetState();
}

class _DeviceListWidgetState extends State<DeviceListWidget> {
  late DeviceListModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();
  final _unfocusNode = FocusNode();

  @override
  void initState() {
    super.initState();
    _model = createModel(context, () => DeviceListModel());
    SchedulerBinding.instance.addPostFrameCallback((_) async {
      setState(() {
        FFAppState().editActive = false;
      });
    });
  }

  @override
  void dispose() {
    _model.dispose();

    _unfocusNode.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    context.watch<FFAppState>();
    return GestureDetector(
      onTap: () => FocusScope.of(context).requestFocus(_unfocusNode),
      child: Scaffold(

```

```

key: scaffoldKey,
backgroundColor: FlutterTheme.of(context).primaryBackground,
drawer: Drawer(
  elevation: 16.0,
  child: wrapWithModel(
    model: _model.sideMenuModel,
    updateCallback: () => setState(() {}),
    child: SideMenuWidget(),
  ),
),
appBar: AppBar(
  backgroundColor: Color(0x00FFFFFF),
  automaticallyImplyLeading: false,
  title: Align(
    alignment: AlignmentDirectional(0.0, 0.0),
    child: Text(
      'Ваши устройства',
      style: FlutterTheme.of(context).headlineMedium.override(
        fontFamily: 'Poppins',
        color: FlutterTheme.of(context).primaryText,
        fontSize: 22.0,
      ),
    ),
  ),
  actions: [],
  centerTitle: false,
  elevation: 0.0,
),
body: SafeArea(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.max,
    children: [
      Expanded(
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.max,
            children: [
              StreamBuilder<List<TvRecord>>(
                stream: queryTvRecord(
                  queryBuilder: (tvRecord) =>
tvRecord.where('created_by', isEqualTo: currentUserReference),
                ),
                builder: (context, snapshot) {
                  if (!snapshot.hasData) {
                    return LoadingWidget();
                  }
                  List<TvRecord> listViewTVTvRecordList =
snapshot.data!;

                  return ListView.builder(
                    padding: EdgeInsets.zero,
                    primary: false,
                    shrinkWrap: true,
                    scrollDirection: Axis.vertical,
                    itemCount: listViewTVTvRecordList.length,
                    itemBuilder: (context, listViewTVIndex) {
                      final listViewTVTvRecord =
listViewTVTvRecordList[listViewTVIndex];
                      return TVListItemWidget(
                        key:
Key('Keyhj7_${listViewTVIndex}_of_${listViewTVTvRecordList.length}'),
                        ref: listViewTVTvRecord,
                      );
                    },
                  );
                );
            ],
          );
        );
      );
    ],
  );

```

```

        },
    ),
    StreamBuilder<List<AcRecord>>(
        stream: queryAcRecord(
            queryBuilder: (acRecord) =>
acRecord.where('created_by', isEqualTo: currentUserReference),
        ),
        builder: (context, snapshot) {
            if (!snapshot.hasData) {
                return LoadingWidget();
            }
            List<AcRecord> listViewACAcRecordList =
snapshot.data!;

            return ListView.builder(
                padding: EdgeInsets.zero,
                primary: false,
                shrinkWrap: true,
                scrollDirection: Axis.vertical,
                itemCount: listViewACAcRecordList.length,
                itemBuilder: (context, listViewACIndex) {
                    final listViewACAcRecord =
listViewACAcRecordList[listViewACIndex];
                    return ACListItemWidget(
                        key:
Key('Key49z_${listViewACIndex}_of_${listViewACAcRecordList.length}'),
                        ref: listViewACAcRecord,
                    );
                },
            );
        },
    ),
    StreamBuilder<List<LightRecord>>(
        stream: queryLightRecord(
            queryBuilder: (lightRecord) =>
lightRecord.where('created_by', isEqualTo:
currentUserReference),
        ),
        builder: (context, snapshot) {
            if (!snapshot.hasData) {
                return LoadingWidget();
            }
            List<LightRecord> listViewLightLightRecordList =
snapshot.data!;

            return ListView.builder(
                padding: EdgeInsets.zero,
                primary: false,
                shrinkWrap: true,
                scrollDirection: Axis.vertical,
                itemCount: listViewLightLightRecordList.length,
                itemBuilder: (context, listViewLightIndex) {
                    final listViewLightLightRecord =
listViewLightLightRecordList[listViewLightIndex];
                    return LightListItemWidget(
                        key:
Key('Key6wa_${listViewLightIndex}_of_${listViewLightLightRecordList.length}'),
                        ref: listViewLightLightRecord,
                    );
                },
            );
        },
    ),
],
),
),

```

```

        ),
      ],
    ),
  ),
);
}
}

```

Листинг Б.10 – Файл select_device_for_add_model.dart

```

class SelectDeviceForAddModel extends Model {
  late SideMenuModel sideMenuModel;

  void initState(BuildContext context) {
    sideMenuModel = createModel(context, () => SideMenuModel());
  }

  void dispose() {
    sideMenuModel.dispose();
  }
}

```

Листинг Б.11 – Файл log_in_page_model.dart

```

class LogInPageModel extends Model {
  TextEditingController? emailController;
  String? Function(BuildContext, String?)? emailControllerValidator;

  TextEditingController? passwordController;
  late bool passwordVisibility;
  String? Function(BuildContext, String?)? passwordControllerValidator;

  void initState(BuildContext context) {
    passwordVisibility = false;
  }

  void dispose() {
    emailController?.dispose();
    passwordController?.dispose();
  }
}

```

Листинг Б.12 – Файл log_in_page_widget.dart

```

class LogInPageWidget extends StatefulWidget {
  const LogInPageWidget({Key? key}) : super(key: key);

  @override
  _LogInPageWidgetState createState() => _LogInPageWidgetState();
}

class _LogInPageWidgetState extends State<LogInPageWidget> {
  late LogInPageModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();
  final _unfocusNode = FocusNode();

  @override
  void initState() {

```



```

        ),
      ),
    ),
    style: FlutterTheme.of(context).bodyMedium,
    keyboardType: TextInputType.emailAddress,
    validator:
      _model.emailControllerValidator.asValidator(context),
  ),
),
TextFormField(
  controller: _model.passwordController,
  autofocus: true,
  obscureText: !_model.passwordVisibility,
  decoration: InputDecoration(
    labelText: 'Пароль',
    hintText: 'Введите пароль',
    hintStyle: FlutterTheme.of(context).bodySmall,
    enabledBorder: UnderlineInputBorder(
      borderSide: BorderSide(
        color: Color(0x00000000),
        width: 1.0,
      ),
      borderRadius: const BorderRadius.only(
        topLeft: Radius.circular(4.0),
        topRight: Radius.circular(4.0),
      ),
    ),
  ),
  suffixIcon: InkWell(
    onTap: () => setState(
      () => _model.passwordVisibility =
        !_model.passwordVisibility,
    ),
    focusNode: FocusNode(skipTraversal: true),
    child: Icon(
      _model.passwordVisibility ? Icons.visibility_outlined
      : Icons.visibility_off_outlined,
      color: FlutterTheme.of(context).accent1,
      size: 22,
    ),
  ),
),
style: FlutterTheme.of(context).bodyMedium,
keyboardType: TextInputType.visiblePassword,
validator:
  _model.passwordControllerValidator.asValidator(context),
),
Padding(
  padding: EdgeInsetsDirectional.fromSTEB(0.0, 30.0, 0.0,
0.0),
  child: FFButtonWidget(
    onPressed: () async {
      GoRouter.of(context).prepareAuthEvent();
      final user = await authManager.signInWithEmail(
        context,
        _model.emailController.text,
        _model.passwordController.text,
      );
      if (user == null) return;
      context.goNamedAuth('Rooms', mounted);
    },
    text: 'Войти',
    options: FFButtonOptions(
      width: double.infinity,
      height: 40.0,

```

```

padding: EdgeInsetsDirectional.fromSTEB(0.0, 0.0, 0.0,
0.0),
padding: EdgeInsetsDirectional.fromSTEB(0.0, 0.0,
0.0, 0.0),
color: FlutterTheme.of(context).primary,
textStyle: FlutterTheme.of(context).titleSmall.override(
  fontFamily: 'Poppins',
  color: Colors.white,
),
borderSide: BorderSide(color: Colors.transparent, width:
1.0),
borderRadius: BorderRadius.circular(8.0),
),
),
),
Padding(
padding: EdgeInsetsDirectional.fromSTEB(0.0, 30.0, 0.0,
0.0),
child: FFButtonWidget(
  onPressed: () async {
    context.pushNamed('RegisterPage');
  },
  text: 'Регистрация',
  options: FFButtonOptions(
    width: double.infinity,
    height: 40.0,
    padding: EdgeInsetsDirectional.fromSTEB(0.0, 0.0, 0.0,
0.0),
    iconPadding: EdgeInsetsDirectional.fromSTEB(0.0, 0.0,
0.0, 0.0),
    color: FlutterTheme.of(context).primary,
    textStyle: FlutterTheme.of(context).titleSmall.override(
      fontFamily: 'Poppins',
      color: Colors.white,
    ),
    borderSide: BorderSide(
      color: Colors.transparent,
      width: 1.0,
    ),
    borderRadius: BorderRadius.circular(8.0),
  ),
),
),
),
),
),
),
),
),
),
),
);
}
}

```

Листинг Б.13 – Файл functions.dart

```

bool checkDeviceName(String name) {
  final reg = RegExp(r'^[a-zA-Za-яА-ЯёЁ]+[a-zA-Za-яА-ЯёЁ0-9 ]+$');
  return name.length >= 3 && name.length <= 20 && reg.hasMatch(name);
}

bool checkPassword(String password) {
  final nonSpaceAndLength = RegExp(r'^(\S+){6}$');

```

```

    final lowerCase = RegExp(r'[a-z]');
    final upperCase = RegExp(r'[A-Z]');
    final digits = RegExp(r'\d');
    return nonSpaceAndLength.hasMatch(password) &&
        lowerCase.hasMatch(password) &&
        upperCase.hasMatch(password) &&
        digits.hasMatch(password);
}

int stringToInt(String str) {
    return double.parse(str).round();
}

Color stringToColor(String str) {
    return Color(int.parse(str.substring(1, 7), radix: 16) + 0xFF000000);
}

String colorToStringHEX(Color color) {
    String str = '${color.value.toRadixString(16)}';
    return '#' + str.substring(2, 8);
}

Future<bool> checkRoomBool(DocumentReference roomRef) async {
    int count = 0;
    count += await checkDevicesInRoom("TV", roomRef);
    count += await checkDevicesInRoom("AC", roomRef);
    count += await checkDevicesInRoom("Light", roomRef);

    if (count != 0) return false;

    return true;
}

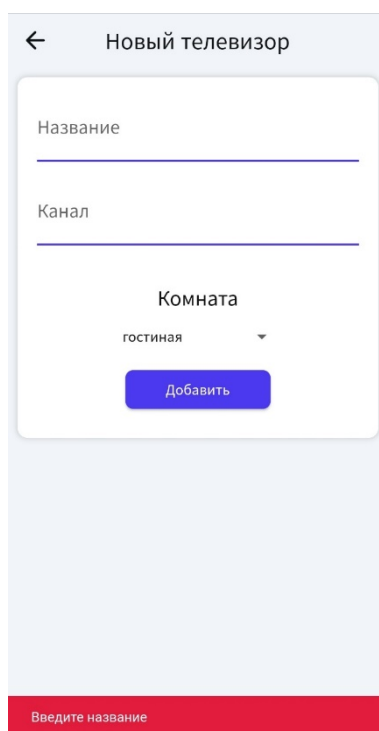
Future<int> checkDevicesInRoom(
    String roomName, DocumentReference roomRef) async {
    FirebaseFirestore db = FirebaseFirestore.instance;

    QuerySnapshot snapshot =
        await db.collection(roomName).where("room", isEqualTo: roomRef).get();

    return snapshot.size;
}

```

ПРИЛОЖЕНИЕ В
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)



← Новый телевизор

Название

Канал

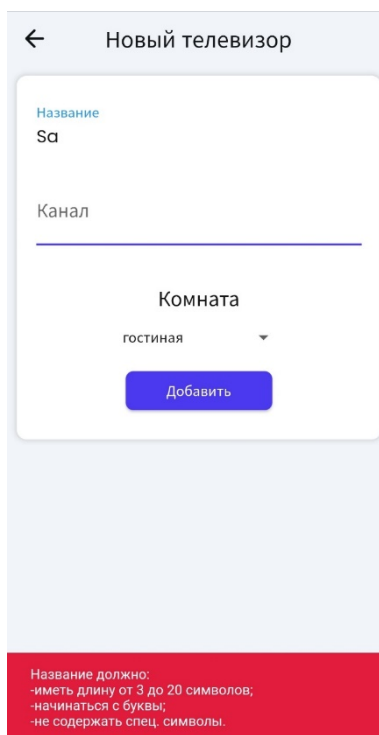
Комната

гостиная

Добавить

Введите название

Рисунок В.1 – Тест добавления телевизора (пустое название, пустой канал)



← Новый телевизор

Название

Sa

Канал

Комната

гостиная

Добавить

Название должно:
-иметь длину от 3 до 20 символов;
-начинаться с буквы;
-не содержать спец. символы.

Рисунок В.2 – Тест добавления телевизора (некорректное название, пустой канал)

← Новый телевизор

Название
Samsung

Канал

Комната
гостиная

Добавить

Введите номер канала

Рисунок В.3 – Тест добавления телевизора (корректное название, пустой канал)

← Новый телевизор

Название
Samsung

Канал
0

Комната
гостиная

Добавить

Номер канала должен быть больше нуля

Рисунок В.4 – Тест добавления телевизора (корректное название, некорректный канал)