МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра компьютерных технологий и систем

ШИБКО Татьяна Александровна

РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ОБЩЕНИЯ С МИКРОКОНТРОЛЛЕРОМ

Дипломная работа

Научный руководитель:
кандидат физико-математических наук
доктор педагогических наук
В.В. Казаченок

Допущена к защите
«__»_____ 2025 г.
Заведующий кафедрой компьютерных технологий и систем
кандидат физико-математических наук,
доктор педагогических наук
В.В. Казаченок

Минск, 2025

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

IoT – Интернет вещей
IIoT – Промышленный интернет вещей
USB – универсальный интерфейс подключения
Wi-Fi – беспроводная связь высокой скорости
ZigBee – протокол для низкопотребляющих IoT-устройств
Z-Wave –протокол для «умного дома»
Mesh – ячеистая сеть
Bluetooth / BLE – беспроводная связь / энергоэффективная версия
GPS-ремни – носимые устройства с GPS
MQTT – протокол передачи данных в IoT
HTTP / HTTPS – протоколы передачи данных / защищённый
JSON – формат обмена данными
API – интерфейс программирования приложений
UI/UX – пользовательский интерфейс и опыт
Flutter – фреймворк кроссплатформенной разработки
React Native – фреймворк мобильной разработки
Kotlin Multiplatform – кроссплатформенная разработка
Dart – язык программирования Flutter
ESP32/ESP8266 – популярные микроконтроллеры IoT
STM32 – семейство микроконтроллеров
ARM Cortex-M – архитектура ядер для STM32
Skia – графический движок Flutter
Postman – инструмент тестирования API
DevTools – набор инструментов для разработчиков
APK/IPA – форматы Android/iOS-приложений
WPAN – беспроводная персональная сеть
NFC – ближняя бесконтактная связь
eSE – встроенный защищённый элемент
BlackBox – автоматическая интеграция устройств
SDK – набор инструментов разработки ПО
IDE – интегрированная среда разработки
TypeScript – типизированный JavaScript
Node-RED – визуальная разработка IoT-приложений
Google Play Store / App Store – магазины приложений
Arduino – платформа для электронных проектов
Speech_to_text – библиотека речевого ввода
TokenManager – менеджер токенов авторизации
AuthService – сервис аутентификации

LedController – контроллер управления светодиодами
HttpService – обработка HTTP-запросов
SpeechService – распознавание речи
VoiceCommandHandler – обработка голосовых команд
HomePage – главная страница приложения
BottomNavigationBar – нижняя панель навигации
SnackBar / Dialogs – всплывающие уведомления / окна
.arb – файлы локализации в Flutter
SVG / flutter_svg – формат изображений / библиотека
LayoutBuilder / MediaQuery – инструменты адаптивности
PlatformIO / Arduino IDE / Xcode / VS Code – среды разработки
STMicroelectronics – производитель STM32
I2C / UART – последовательные шины связи
Espressif – производитель ESP32/ESP8266
Raspberry Pi – одноплатный компьютер
Android / iOS – мобильные операционные системы

РЕФЕРАТ

Дипломная работа 82 с., 10 рис., 12 таблиц, 2 приложения, 30 источников.

Ключевые слова: МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, УПРАВЛЕНИЕ МИКРОКОНТРОЛЛЕРОМ, ESP-32, IOT, ФУНКЦИОНАЛ ПРИЛОЖЕНИЯ, ОПТИМИЗАЦИЯ.

Объект исследования: Разработка мобильного приложения для управления микроконтроллером ESP-32 через Wi-Fi и USB. Исследуются методы взаимодействия мобильных устройств с IoT-устройствами, а также способы повышения удобства и производительности приложения.

Цель исследования: Создание мобильного приложения для управления микроконтроллером, который может быть использован в бытовых и промышленных системах.

Методы исследования: Анализ современных технологий IoT, сравнение способов связи (USB, Wi-Fi, Bluetooth), выбор подходящих инструментов разработки (Flutter), проектирование архитектуры приложения, реализация функционала и тестирование производительности.

Полученные результаты и их новизна: Разработано мобильное приложение с поддержкой голосового управления, динамической локализации (русский, английский, арабский) и системы световых эффектов. Приложение успешно прошло тестирование, показав время отклика в пределах 300-500 миллисекунд.

Область возможного практического применения: Управление IoT-устройствами в умных домах, образовательных проектах и промышленности.

Автор работы подтверждает, что приведенный в ней расчетно-аналитический материал правильно и объективно отражает состояние исследуемого процесса, а все заимствованные из литературных и других источников теоретические, методологические и методические положения и концепции сопровождаются ссылками на их авторов.

РЭФЕРАТ

Дыпломная праца 82 с., 10 мал., 12 табліц, 2 дадатка, 30 крыніц.

Ключавыя словы: МАБІЛЬНАЕ ПРЫКЛАДАННЕ, КІРАВАННЕ МІКРАКАНТРОЛЕРАМ, ESP-32, IOT, ФУНКЦЫЯНАЛ ПРЫКЛАДАННЯ, АПТЫМІЗАЦЫЯ.

Аб'ект даследвання: Распрацоўка мабільнага прыкладання для кіравання мікракантролерам ESP-32 праз Wi-Fi і USB. Даследуюцца метады ўзаемадзеяння мабільных прылад з IoT-прыладамі, а таксама спосабы павышэння зручнасці і прадукцыйнасці прыкладання.

Мэта даследвання: Стварэнне мабільнага прыкладання для кіравання мікракантролерам, якое можа быць выкарыстана ў бытавых і прамысловых сістэмах.

Метады даследвання: Аналіз сучасных тэхналогій IoT, параўнанне спосабаў сувязі (USB, Wi-Fi, Bluetooth), выбар падыходных інструментаў распрацоўкі (Flutter), праектаванне архітэктуры прыкладання, рэалізацыя функцыяналу і тэставанне прадукцыйнасці.

Атрыманыя вынікі і іх новізна: Распрацавана мабільнае прыкладанне з падтрымкай галасавога кіравання, дынамічнай лакалізацыі (руская, англійская, арабская) і сістэмы светлавых эфектаў. Дадатак паспяхова прайшоў тэставанне, паказаўшы час водгуку ў межах 300-500 мілісекунд.

Вобласць магчымага практычнага прымянення: Кіраванне IoT-прыладамі ў разумных дамах, адукацыйных праектах і прамысловасці.

Аўтар працы пацвярджае, што прыведзены ў ёй разлікова-аналітычны матэрыял правільна і аб'ектыўна адлюстроўвае стан доследнага працэсу, а ўсе запазычаныя з літаратурных і іншых крыніц тэарэтычныя, метадалагічныя і метадычныя становішча і канцэпцыі суправаджаюцца спасылкамі на іх аўтараў.

ABSTRACT

Degree paper: 82 pages, 10 img., 12 tables, 2 appendixes, 30 sources.

Keywords: MOBILE APPLICATION, MICROCONTROLLER CONTROL, ESP-32, IOT, APPLICATION FUNCTIONALITY, OPTIMIZATION.

Object of research: Development of a mobile application for controlling the ESP-32 microcontroller via Wi-Fi and USB. The study explores methods of interaction between mobile devices and IoT devices, as well as ways to improve the convenience and performance of the application.

Research objective: Creating a mobile application to control a microcontroller that can be used in domestic and industrial systems.

Research methods: Analysis of modern IoT technologies, comparison of communication methods (USB, Wi-Fi, Bluetooth), selection of appropriate development tools (Flutter), design of the application architecture, implementation of functionality, and performance testing.

Obtained results and their novelty: A mobile application has been developed with support for voice control, dynamic localization (Russian, English, Arabic), and a system of light effects. The application successfully passed testing, showing a response time within 300-500 milliseconds.

Potential area of practical application: Control of IoT devices in smart homes, educational projects, and industry.

The author confirms that the presented calculation and analytical material accurately and objectively reflects the state of the studied process, and all theoretical, methodological, and methodical positions and concepts borrowed from literary and other sources are accompanied by references to their authors.

ВВЕДЕНИЕ

В эпоху стремительного технологического прогресса наш мир наполняется умными устройствами, взаимодействие с которыми становится всё более естественным и доступным. Невозможно представить современную жизнь без микроконтроллеров – миниатюрных вычислительных модулей, которые словно невидимые помощники управляют множеством систем вокруг нас. От простейших светодиодных лент, создающих уютную атмосферу в наших домах, до сложнейших промышленных комплексов – везде присутствуют эти маленькие «мозги» электронных устройств.

К сожалению, несмотря на повсеместное распространение микроконтроллеров, работа с ними до сих пор остаётся прерогативой технических специалистов, владеющих навыками программирования и электроники. Это создаёт значительный разрыв между потенциалом технологий и их доступностью для обычных пользователей. Именно эту проблему призвана решить разработка специализированного мобильного приложения для управления микроконтроллерами через смартфон, которая представлена в данной работе. Такое решение способно существенно снизить входной порог для взаимодействия с электронными устройствами, делая их доступными даже для людей без глубоких технических познаний.

Актуальность данного исследования обусловлена растущей потребностью общества в понятных и удобных инструментах управления электронными системами. Особую значимость эта задача приобретает в контексте бурного развития интернета вещей (IoT), где эффективное взаимодействие между устройствами становится не просто желательным, а необходимым условием для построения по-настоящему интеллектуальных систем, окружающих нас повсюду.

Эта дипломная работа фокусируется на создании полноценного мобильного приложения для управления микроконтроллерами. Чтобы достичь этой цели, нужно последовательно решать связанные задачи.

В первой главе объясняются основные идеи, на которых строится вся работа – в частности, как работает интернет вещей (IoT) и как между собой взаимодействуют устройства без участия человека. Во второй главе рассказывается, какие технологии были выбраны для разработки приложения, почему они подошли, и как проектировалась его структура – то есть как всё должно работать внутри. В третьей главе показаны результаты: что получилось в итоге, как приложение себя показало при тестировании, какие у него сильные стороны, а что можно улучшить в будущем.

Практическая ценность этой работы в том, что разработанное решение можно использовать в разных областях. В образовании оно может быть

полезным инструментом для изучения микроконтроллеров и программирования. Для индивидуальных проектов приложение становится отличным помощником для быстрого прототипирования и отладки. В повседневной жизни оно помогает создавать простые, но эффективные системы умного дома. В промышленности же такое решение может использоваться для оперативного мониторинга и управления несложными автоматизированными системами. Подобная универсальность обеспечивает востребованность предложенного подхода среди широкого спектра потенциальных пользователей – от студентов и энтузиастов до профессионалов различных отраслей.

Каждый раздел исследования завершается содержательными выводами, обобщающими ключевые результаты проделанной работы. Таким образом, представленное исследование нацелено на создание интуитивно понятного и доступного инструмента управления микроконтроллерами, способного найти применение в самых разнообразных аспектах современной жизни.

ГЛАВА 1
ТЕХНИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ВЗАИМОДЕЙСТВИЯ С МИКРОКОНТРОЛЛЕРОМ

1.1 Обзор технологий IoT и их роли в современном мире

Современный мир невозможно представить без концепции Интернета вещей (IoT) – инновационного подхода, предполагающего объединение различных физических устройств, оснащенных сенсорами, программным обеспечением и другими технологиями для обмена данными через глобальную сеть[4]. Суть данной технологии заключается в возможности подключения к интернету широчайшего спектра объектов – от привычных бытовых приборов до сложных промышленных машин, что наделяет их способностью к автономному взаимодействию и интеллектуальному управлению. Достигается подобный эффект посредством применения специализированных сенсоров, непрерывно собирающих информацию об окружающей среде и передающих полученные данные для дальнейшей обработки. Особую роль в этом процессе играет программное обеспечение, использующее передовые алгоритмы машинного обучения и искусственного интеллекта для аналитики поступающей информации, что позволяет системам принимать обоснованные решения, автоматизировать рутинные процессы и существенно оптимизировать работу устройств. Примером такого функционирования может служить современная система управления «умным домом», способная в автоматическом режиме регулировать освещение и отопление на основе показаний различных датчиков, обеспечивая тем самым максимальный комфорт проживания при одновременной экономии энергоресурсов.

Сфера применения технологий IoT поистине обширна и охватывает множество областей человеческой деятельности – от повседневного домашнего быта до промышленного производства и городской инфраструктуры. Технологии умного дома [17] по праву считаются одним из наиболее перспективных направлений развития Интернета вещей. Современное жилое пространство уже сложно представить без множества "умных" устройств, автоматизирующих повседневные задачи: интеллектуальные термостаты и кондиционеры, голосовые помощники в виде умных колонок, адаптивные системы освещения, автоматические кормушки для домашних питомцев – все эти технологии делают жизнь значительно удобнее, комфортнее и энергоэффективнее. К примеру, современные умные термостаты способны не просто поддерживать заданную температуру, но и изучать предпочтения

пользователей, адаптивно регулируя микроклимат в зависимости от времени суток или присутствия людей в помещении.

В аграрном секторе IoT-технологии нашли не менее впечатляющее применение, активно используясь как в растениеводстве, так и в животноводстве. Особенно ярким примером подобной интеграции являются специализированные дроны, существенно упрощающие процесс мониторинга состояния сельскохозяйственных угодий, анализа состава почвы и прогнозирования климатических изменений. Не менее важную роль играют IoT-устройства в контексте отслеживания здоровья сельскохозяйственных животных и оперативного определения местоположения особей с признаками заболеваний, что значительно повышает эффективность работы современных фермерских хозяйств и минимизирует риски массового распространения болезней.

Промышленный сектор также активно внедряет технологии Интернета вещей, где данная концепция получила специальное наименование «промышленный интернет вещей» (IIoT). В производственных процессах эти технологии находят применение для непрерывного мониторинга состояния оборудования, автоматизированного контроля качества выпускаемой продукции и комплексного анализа больших массивов данных. Передовые сенсорные системы и программные комплексы способствуют разработке более точных дизайнерских решений и постоянному совершенствованию производственных циклов [4].

Сфера розничной торговли меняется благодаря внедрению технологий интернета вещей (IoT). Теперь магазины могут точнее настраивать рекламу так, чтобы покупателям показывались именно те товары, которые им сейчас нужны. Это можно делать в нужное время и в удобном месте. Кроме того, IoT помогает лучше организовать поставки товаров и анализировать, что покупают чаще всего. Благодаря этому торговые сети могут работать эффективнее, быстрее реагировать на изменения спроса и улучшать обслуживание клиентов. Наглядными примерами интеграции IoT в розничную торговлю служат системы бесконтактных платежей и специализированные приложения для совершения покупок, существенно упрощающие процесс взаимодействия между продавцом и покупателем.

Здравоохранение, несомненно, является одной из важнейших сфер применения IoT-технологий. Современные решения позволяют медицинским специалистам оказывать квалифицированную помощь пациентам в удаленном режиме, использовать беспилотные летательные аппараты для оперативной доставки жизненно важных медикаментов в труднодоступные районы и совершать настоящие прорывы в области генетических исследований. Технологии Интернета вещей становятся основой для развития

персонализированной медицины, где методики лечения и профилактики заболеваний подбираются индивидуально на основании комплексного анализа состояния здоровья конкретного пациента [7]. Несмотря на объективные сложности, связанные с процессом интеграции инновационных IoT-решений в консервативную медицинскую практику, данное направление продолжает демонстрировать стремительные темпы развития, открывая новые горизонты для повышения качества и доступности медицинской помощи.

Умные автомобили представляют собой, пожалуй, одно из наиболее впечатляющих воплощений концепции IoT [5]. Современные транспортные средства оснащаются множеством высокочувствительных датчиков и постоянно подключены к глобальной сети, что обеспечивает беспрецедентный уровень комфорта для водителей и пассажиров. Перспективным направлением развития в данной области являются полностью автономные автомобили, управляемые продвинутыми системами искусственного интеллекта, способные кардинально изменить существующую транспортную парадигму. Такие транспортные средства смогут не только обеспечивать безопасную и своевременную доставку пассажиров, но и предоставлять широкий спектр дополнительных функций, включая удаленное управление дверьми или регулирование температурного режима в гаражном помещении.

Носимые устройства, такие как фитнес-браслеты, умные часы и GPS-ремни, стали неотъемлемой частью повседневной жизни современного человека [4]. Они предназначены для мониторинга показателей здоровья, включая пульс и уровень физической активности. Многие из них интегрируются с медицинскими IoT-системами, обеспечивая автоматический сбор данных в режиме реального времени. Лидерами на рынке выступают крупные компании – Apple, Samsung и Motorola. Они регулярно совершенствуют свои устройства, расширяя их функциональные возможности и повышая практическую ценность для пользователей. Ещё одной перспективной областью применения IoT [4] является концепция «умного города». Она предполагает использование технологий для повышения безопасности, удобства и энергоэффективности городской среды. Например, внедрение автоматизированного уличного освещения, систем управления парковками и мониторинга уровня шума и экологической обстановки позволяет значительно улучшить качество жизни в крупных городах. К практическим примерам реализации данной концепции можно отнести интеллектуальные парковочные системы, автоматизированные комплексы управления уличным освещением, интерактивные карты шумового загрязнения и многоуровневый мониторинг экологической обстановки. Несмотря на то, что многие из перечисленных решений находятся на различных стадиях разработки и тестирования, их потенциал для улучшения качества

городской среды поистине огромен. Умные города будущего смогут эффективнее контролировать транспортные потоки, существенно снижать уровень вредных выбросов и значительно повышать общее качество жизни городских жителей.

В сфере логистики внедрение IoT-технологий открывает беспрецедентные возможности для отслеживания товарных потоков в режиме реального времени, оперативного контроля процесса доставки и обеспечения открытого обмена информацией между всеми участниками цепочки поставок [7]. Это позволяет существенно снизить количество требуемого персонала, автоматизировать рутинные операции и значительно уменьшить сопутствующие затраты при одновременном повышении эффективности логистических процессов.

Энергетический сектор также активно внедряет инновационные решения на базе IoT для создания умных электросетей, способных автоматически собирать и анализировать данные о циркуляции электроэнергии. Подобные системы приносят пользу как конечным потребителям, так и поставщикам энергоресурсов, помогая оптимизировать использование доступных мощностей и существенно снижать сопутствующие расходы.

В основе функционирования большинства IoT-устройств лежат микроконтроллеры – миниатюрные вычислительные системы, объединяющие процессор, память и набор периферийных модулей на едином кристалле. Эти высокоинтегрированные схемы спроектированы для выполнения специализированных задач, включая управление разнообразными датчиками, обработку поступающих данных и организацию взаимодействия с другими устройствами посредством различных коммуникационных интерфейсов. Благодаря своей универсальности и компактным размерам, современные микроконтроллеры нашли применение практически во всех сферах – от повседневной бытовой техники до комплексных промышленных систем автоматизации.

Одним из ключевых преимуществ микроконтроллеров является их исключительная энергоэффективность. В подавляющем большинстве IoT-приложений устройства должны функционировать автономно на протяжении длительных периодов времени, зачастую используя в качестве источника питания батареи или другие ограниченные энергоресурсы. Современные микроконтроллеры специально проектируются с учетом оптимизации энергопотребления, что позволяет значительно увеличить время автономной работы устройств и существенно повысить их общую энергоэффективность. Именно это свойство делает микроконтроллеры незаменимыми компонентами таких решений, как автономные сенсорные системы, портативные носимые гаджеты и распределенные комплексы "умного дома".

Компактные габариты микроконтроллеров также представляют собой значительное технологическое преимущество. Множество современных IoT-устройств требуют максимальной миниатюризации для обеспечения возможности их интеграции в ограниченное пространство или встраивания в уже существующие продукты. Современная микроэлектронная промышленность позволяет производить исключительно миниатюрные микроконтроллеры, что открывает перспективы для создания легких и компактных устройств без ущерба для их функциональности. Особую ценность данное свойство представляет для медицинской отрасли, где подобные технологии становятся основой для разработки имплантируемых устройств и других инновационных решений, способных значительно повысить качество диагностики и лечения различных заболеваний.

Еще одним существенным фактором, способствующим широкому распространению микроконтроллеров в IoT-экосистеме, является их доступная стоимость. Относительно низкая цена данных компонентов значительно стимулирует процесс распространения IoT-технологий, позволяя даже небольшим компаниям и инновационным стартапам разрабатывать и выводить на рынок передовые продукты и сервисы. Это, в свою очередь, существенно ускоряет внедрение новых технологических решений и значительно расширяет горизонты возможностей для дальнейшего технологического прогресса во множестве отраслей современной экономики.

1.2 Методы связи мобильных устройств с микроконтроллерами

Современные системы умного дома функционируют на основе различных протоколов связи, каждый из которых обладает своими уникальными характеристиками, определяющими их применимость в экосистеме интернета вещей. Стоит подробнее рассмотреть четыре ключевых протокола – Wi-Fi, Bluetooth, Z-Wave и ZigBee – с точки зрения их практического использования при организации умного дома (см. Таблицу 1.1).

Таблица 1.1 – Итоговый сравнительный анализ методов связи мобильных устройств с микроконтроллерами

Критерий	**Wi-Fi**	**Bluetooth**	**Z-Wave**	**ZigBee**
Дальность передачи	Ограниченная дальностью роутера	До 10 метров	До 100 метров	До 100 метров
Энергопотребление	Высокое	Низкое (BLE)	Низкое	Низкое
Скорость передачи	Высокая	Низкая	Низкая	Низкая

Совместимость	Широкая	Широкая	Ограниченная (по регионам)	Высокая (открытый стандарт)
Безопасность	Умеренная	Умеренная	Высокая	Высокая
Стоимость устройств	Доступная	Доступная	Высокая	Умеренная

Wi-Fi занимает позицию одного из наиболее распространённых стандартов беспроводной связи, находя применение не только для обеспечения доступа в интернет, но и для управления широким спектром умных устройств, включая термостаты, интеллектуальные розетки и системы видеонаблюдения. Основное преимущество данного стандарта заключается в его удобстве использования, поскольку каждое поддерживающее Wi-Fi устройство способно подключаться к сети напрямую, без необходимости задействования дополнительных хабов или шлюзов [18]. Кроме того, Wi-Fi обеспечивает значительную скорость передачи данных, что становится критически важным фактором для устройств, требующих обработки объёмного трафика, таких как IP-камеры высокого разрешения. Повсеместное распространение Wi-Fi превращает его в универсальное решение, поддерживаемое большинством современных электронных устройств. Тем не менее, данный протокол не лишён определённых недостатков. Его высокое энергопотребление существенно сокращает время автономной работы устройств, функционирующих от аккумуляторов или батарей, а большое количество одновременно подключённых девайсов способно привести к перегрузке сети и заметному снижению её производительности. Следует также принимать во внимание потенциальные уязвимости в системе безопасности, особенно если сетевая инфраструктура не защищена должным образом. Дополнительным ограничением выступает дальность действия сигнала, которая напрямую зависит от мощности используемого роутера, и для расширения зоны покрытия зачастую приходится прибегать к использованию репитеров или современных Mesh-систем [19].

Bluetooth представляет собой ещё один широко используемый протокол, предназначенный преимущественно для организации связи на относительно коротких дистанциях, обычно не превышающих 10 метров. Данная технология часто находит применение в носимых устройствах, таких как фитнес-браслеты, а также в умных замках и других гаджетах, не требующих постоянного подключения к глобальной сети. Одним из ключевых преимуществ Bluetooth, особенно его энергоэффективной версии Low Energy (BLE), является минимальное потребление энергии, что позволяет значительно увеличить время автономной работы устройств без подзарядки. Процесс настройки соединения между Bluetooth-устройствами отличается простотой и интуитивной

понятностью, что делает данную технологию доступной даже для пользователей без специальных технических навыков. Важным фактором выступает также то, что Bluetooth [2] поддерживается практически всеми современными смартфонами, планшетами и различными мобильными гаджетами. Однако следует учитывать и существенные ограничения данного протокола. Его сравнительно небольшая дальность действия делает Bluetooth малоприменимым для организации связи в больших помещениях или между удалёнными устройствами, а количество одновременных подключений имеет строгие лимиты. Более того, для создания полноценной умной экосистемы на основе Bluetooth обычно требуется дополнительное оборудование в виде центрального хаба или шлюзового устройства, что неизбежно усложняет общую архитектуру системы.

Z-Wave [2] представляет собой специализированный протокол, разработанный именно для применения в системах умного дома. Данная технология функционирует в низкочастотном диапазоне (800-900 МГц), что обеспечивает высокий уровень стабильности соединения даже в сложных условиях. Надёжность Z-Wave во многом обусловлена использованием принципа Mesh-сети, в которой каждое отдельное устройство способно выступать в качестве ретранслятора сигнала, значительно усиливая общее покрытие сети. Дальность передачи сигнала в данном протоколе может достигать 100 метров на открытом пространстве, а сравнительно низкое энергопотребление позволяет совместимым устройствам функционировать от батарей в течение продолжительного времени без необходимости замены источника питания. Тем не менее, у Z-Wave имеются определённые недостатки. В разных странах мира для данного протокола используются различные частотные диапазоны, что потенциально создаёт проблемы совместимости между устройствами, приобретёнными в разных географических регионах. Кроме того, стоимость Z-Wave-устройств обычно превышает цену аналогичных решений, построенных на основе альтернативных протоколов связи [23].

ZigBee выступает ещё одним популярным протоколом для организации умного дома, ориентированным преимущественно на устройства с минимальным энергопотреблением. Данная технология широко применяется в различных датчиках, интеллектуальных розетках и автоматизированных выключателях. Главным преимуществом ZigBee является его исключительная энергоэффективность: например, типичный датчик движения на основе данного протокола способен непрерывно работать от одной стандартной батарейки на протяжении до полутора лет [27]. Подобно Z-Wave, ZigBee также использует принцип Mesh-сети, что значительно повышает надёжность связи благодаря возможности ретрансляции сигнала между отдельными устройствами системы.

Открытость данного стандарта теоретически обеспечивает высокую совместимость устройств от различных производителей, а встроенные механизмы шифрования и аутентификации делают ZigBee достаточно безопасным решением. Однако у данного протокола существуют и определённые ограничения. Сравнительно низкая пропускная способность делает его неподходящим для передачи мультимедийного контента, такого как видео или высококачественное аудио [27]. Помимо этого, некоторые производители оборудования создают закрытые экосистемы, вследствие чего устройства различных брендов могут оказаться несовместимыми друг с другом, несмотря на формальную поддержку единого стандарта [27].

При построении современной системы умного дома наиболее универсальным подходом является комбинированное использование устройств, поддерживающих Wi-Fi и ZigBee. Wi-Fi обеспечивает удобное подключение и высокую скорость передачи данных, что особенно важно для устройств с высокими требованиями к пропускной способности канала, таких как камеры видеонаблюдения или мультимедийные системы. В то же время ZigBee предоставляет превосходную надёжность, энергоэффективность и совместимость между устройствами различных производителей, что делает его оптимальным выбором для датчиков, выключателей и других малопотребляющих компонентов умного дома. Подобное сочетание технологий позволяет создать сбалансированную и функциональную экосистему, объединяющую преимущества обоих протоколов в рамках единого пространства.

В ситуациях, когда основным требованием к системе выступает энергоэффективность и обеспечение стабильной связи для устройств с минимальным энергопотреблением, предпочтительным выбором становятся протоколы ZigBee или Z-Wave. Эти технологии демонстрируют высокую степень надёжности даже в условиях плотной городской застройки благодаря использованию принципа Mesh-сетей. Для компактных помещений или устройств, функционирующих на небольших расстояниях друг от друга, вполне целесообразным может оказаться применение Bluetooth, особенно в случаях, когда критически важными факторами являются простота первоначальной настройки и минимальное потребление энергии.

Рассмотрим далее особенности практической реализации связи через интерфейсы USB и Wi-Fi (см. Таблицу 1.2).

Таблица 1.2 – Сравнение особенностей реализации через USB и Wi-Fi

Параметр	**USB**	**Wi-Fi**
Тип подключения	Проводное	Беспроводное
Скорость подключения	Высокая	Очень высокая
Энергопотребление	Низкое	Высокое
Настройка подключения	Простая (прямое подключение)	Требует настройки сети
Масштабируемость	Ограничена одним устройством	Поддерживает множество устройств
Безопасность	Физическое подключение снижает риски	Требует шифрования и защиты данных
Примеры использования	Образовательные проекты, прототипирование	Умные дома, облачные системы, промышленность

USB (Universal Serial Bus) представляет собой универсальный проводной интерфейс для подключения различных периферийных устройств, получивший широчайшее распространение благодаря своей практичности, высокой надёжности и впечатляющей скорости передачи данных. Однако при организации связи между мобильным устройством и микроконтроллером посредством USB необходимо учитывать ряд специфических особенностей данного интерфейса.

Современные мобильные устройства, безусловно, поддерживают подключение через USB, однако уровень этой поддержки существенно варьируется в зависимости от используемой операционной системы. Например, в экосистеме Android для обеспечения полноценного взаимодействия с внешними устройствами зачастую требуется применение специализированных программных библиотек, таких как Android Open Accessory Protocol (AOA). В случае с устройствами на базе iOS уровень поддержки USB оказывается ещё более ограниченным и обычно требует использования аксессуаров, официально сертифицированных по программе MFi (Made for iPhone), что накладывает серьёзные дополнительные ограничения на разработчиков соответствующих решений.

Физическое подключение через интерфейс USB характеризуется простотой реализации и обеспечивает стабильное соединение практически без задержек, что делает данный интерфейс особенно удобным для задач, требующих высочайшей надёжности канала связи. При этом следует учитывать, что современные мобильные устройства могут использовать различные типы

физических разъёмов (USB-C или Lightning), что в некоторых ситуациях может потребовать применения дополнительных переходников или адаптеров.

Важным функциональным преимуществом USB является возможность не только передачи данных, но и обеспечения электрического питания подключённых устройств. Эта особенность оказывается исключительно полезной при работе с микроконтроллерами, характеризующимися низким энергопотреблением, поскольку позволяет организовать их питание непосредственно от мобильного устройства без необходимости использования отдельного источника энергии.

Необходимо отметить, что современные мобильные платформы накладывают определённые программные ограничения на использование USB для передачи данных. В частности, разработчикам программного обеспечения для iOS приходится работать с относительно сложным External Accessory Framework, что заметно усложняет процесс разработки по сравнению с другими мобильными платформами.

В качестве наглядного практического примера можно рассмотреть использование популярных плат Arduino с USB-интерфейсом, где специализированное мобильное приложение способно передавать управляющие команды микроконтроллеру через стандартный последовательный порт. Подобный подход нашёл широкое применение в образовательных проектах и при создании функциональных прототипов различных электронных устройств [2].

Wi-Fi как беспроводной протокол связи приобрёл колоссальную популярность благодаря своей способности обеспечивать подключение устройств как к локальным вычислительным сетям, так и непосредственно к глобальной сети интернет. При организации взаимодействия между мобильными устройствами и микроконтроллерами через Wi-Fi необходимо принимать во внимание несколько ключевых технических аспектов.

Одним из основных преимуществ Wi-Fi выступает его впечатляющая скорость передачи данных, что делает данный протокол оптимальным выбором для задач, связанных с обработкой и передачей значительных объёмов информации. Это свойство приобретает особую актуальность для систем видеонаблюдения, потоковой передачи различного мультимедийного контента или управления сложными автоматизированными комплексами с высокими требованиями к пропускной способности канала связи.

Существенным достоинством Wi-Fi является его естественная и бесшовная интеграция с существующей интернет-инфраструктурой. Данная особенность позволяет микроконтроллерам устанавливать прямое подключение к различным облачным сервисам, что открывает поистине широчайшие возможности для

создания распределённых систем сбора, обработки, анализа и долговременного хранения данных.

Однако при проектировании систем на основе Wi-Fi необходимо учитывать его сравнительно высокое энергопотребление по сравнению с другими беспроводными протоколами, такими как Bluetooth или ZigBee. Этот фактор может стать серьёзным ограничением при разработке автономных устройств, функционирующих исключительно от батарейного питания на протяжении длительного времени [9].

Для обеспечения стабильной работы Wi-Fi-устройств требуется предварительная настройка сетевого окружения. Используемый микроконтроллер должен быть корректно сконфигурирован для подключения к определённой беспроводной точке доступа (идентифицируемой по SSID) с применением соответствующих параметров аутентификации. Данный процесс может создавать определённые технические сложности при массовом развёртывании IoT-устройств в реальных производственных условиях.

Wi-Fi [2] демонстрирует превосходную масштабируемость, позволяя подключать к единой сети значительное количество разнообразных устройств. Это свойство делает его подходящим технологическим решением для крупных инфраструктурных систем, таких как комплексные умные дома или промышленные автоматизированные комплексы. Тем не менее, необходимо принимать во внимание, что при большом количестве одновременно активных подключений возможно возникновение перегрузок сети, что может негативно сказываться на общей производительности системы.

Вопросы обеспечения информационной безопасности при использовании Wi-Fi требуют особого внимания со стороны разработчиков. Для надёжной защиты передаваемых данных необходимо последовательно применять современные методы криптографического шифрования (например, WPA3) и реализовывать дополнительные многоуровневые механизмы аутентификации для эффективного предотвращения несанкционированного доступа к системе.

В качестве наглядных примеров успешной практической реализации Wi-Fi-решений можно привести такие популярные микроконтроллерные платформы как ESP8266 и ESP32, предлагающие встроенную аппаратную поддержку Wi-Fi. Эти микроконтроллеры нашли широчайшее применение в различных IoT-проектах и могут быть легко интегрированы с современными мобильными приложениями посредством различных программных API или облачных сервисов, таких как Firebase или AWS IoT [3].

Реализация связи между мобильными устройствами и микроконтроллерами через интерфейсы USB и Wi-Fi имеет свои характерные сильные и слабые стороны. USB идеально подходит для технических задач, где

требуется организация стабильного и высокоскоростного подключения с минимальным энергопотреблением. Wi-Fi, в свою очередь, предлагает непревзойдённую гибкость, отличную масштабируемость и возможность бесшовной интеграции с существующей интернет-инфраструктурой, что делает данный протокол предпочтительным выбором для современных IoT-решений различной сложности. Окончательный выбор между этими технологиями в значительной степени зависит от конкретных технических требований разрабатываемого проекта, таких как допустимый уровень энергопотребления, необходимая дальность связи и потребность в прямом подключении устройств к глобальной сети интернет.

В современной практике разработки IoT-систем [6] существует множество примеров успешной реализации эффективного взаимодействия между мобильными приложениями и микроконтроллерами. Эти инновационные решения наглядно демонстрируют, каким образом различные протоколы связи и технологии могут быть гармонично интегрированы для создания функциональных и удобных систем управления электронными устройствами.

Одним из наиболее ярких технологических примеров выступает использование MQTT-протокола для организации управления устройствами на базе микроконтроллера ESP32 [5]. Данный протокол получил широчайшее распространение в современных IoT-проектах благодаря своей исключительной лёгкости и эффективности при передаче данных через интернет. В рамках одного из профильных исследований была разработана комплексная система удалённого управления, в которой специализированное мобильное приложение взаимодействовало с микроконтроллером ESP32 через MQTT, обеспечивая стабильное сетевое подключение и минимальные временные задержки при передаче управляющих команд. Подобные высокоэффективные решения приобрели особую популярность в современных системах умного дома, где требуется централизованное управление освещением, климатическими системами или комплексами обеспечения безопасности.

Заслуживающим внимания технологическим подходом является также TAC-технология, позволяющая организовать практически незаметное для конечного пользователя взаимодействие между разнообразными электронными устройствами, включая персональные компьютеры и современные смартфоны. Данная технология демонстрирует исключительную эффективность в случаях, когда необходима глубокая интеграция множества разнородных устройств в единую функциональную экосистему без необходимости выполнения сложной технической настройки со стороны пользователя. Например, специализированная система BlackBox может успешно применяться для

комплексной автоматизации различных бизнес-процессов в интеллектуальных офисных пространствах или промышленных системах управления.

В современной автомобилестроительной индустрии всё шире применяются передовые технологии, такие как NFC и eSE (встраиваемые защищённые элементы). Они обеспечивают безопасный обмен данными между смартфоном пользователя и бортовыми системами автомобиля. Указанные технологии позволяют использовать мобильное устройство в качестве цифрового ключа для бесключевого доступа, запуска двигателя, а также для дистанционного управления функциями, такими как климат-контроль и мультимедийная система. Микроконтроллеры [4] играют ключевую роль в обеспечении устойчивого соединения и многоуровневой защиты конфиденциальной информации. Это повышает как удобство, так и безопасность эксплуатации транспортного средства.

Протокол ZigBee [8] также представляет собой важнейший инструмент для организации эффективного взаимодействия между мобильными приложениями и микроконтроллерами, особенно в комплексных системах умного дома. Несмотря на то, что интеграция ZigBee объективно увеличивает общую стоимость разрабатываемой системы, данный подход позволяет максимально эффективно использовать ограниченные вычислительные ресурсы микроконтроллера для выполнения разнообразных задач по управлению подключёнными устройствами. Данное технологическое решение часто применяется в проектах, где приоритетными требованиями выступают высочайшая надёжность и безупречная стабильность беспроводной связи на протяжении всего жизненного цикла системы [8].

Среди программных решений особого внимания заслуживает использование различных интегрированных сред разработки, таких как Arduino IDE, PlatformIO и Keil uVision. Эти мощные инструменты предоставляют профессиональным разработчикам исключительно широкие возможности для создания оптимизированного программного обеспечения, эффективно управляющего процессами взаимодействия между мобильными приложениями и различными микроконтроллерами. Например, популярная среда Arduino IDE активно используется для быстрого прототипирования и разработки функциональных IoT-устройств различной сложности, что делает её чрезвычайно востребованным выбором среди студентов технических специальностей и начинающих разработчиков встраиваемых систем.

Особое внимание в современной индустрии уделяется перспективному применению нейросетевых решений на микроконтроллерах ARM-архитектуры. Эти инновационные системы позволяют выполнять сложнейшие вычислительные задачи непосредственно на конечном устройстве, существенно

минимизируя временные задержки и вычислительную нагрузку на удалённые облачные серверы. Мобильные приложения в подобных технологически продвинутых проектах выступают в качестве удобного пользовательского интерфейса для управления системой и многостороннего анализа данных, получаемых от микроконтроллера в режиме реального времени. Например, такие высокоинтеллектуальные решения могут эффективно применяться в современных системах компьютерного зрения или комплексах обработки естественного языка.

Наконец, микроконтроллеры нашли широчайшее применение в передовой мобильной робототехнике, где они выступают в роли критически важного связующего звена между управляющим мобильным приложением и исполнительными механизмами автономных роботов. Это направление приобретает особую значимость в образовательных проектах, где студенты получают возможность практического создания автономных электронных устройств, дистанционно управляемых через обычный смартфон.

1.3 Анализ популярных микроконтроллеров для IoT

Микроконтроллеры ESP8266 и ESP32 [4], сделанные компанией Espressif, сильно изменили мир интернета вещей (IoT). ESP8266 был одним из первых недорогих устройств с поддержкой Wi-Fi. Благодаря низкой цене, маленькому размеру и экономному расходу энергии, он быстро стал популярным среди любителей электроники и опытных разработчиков. Особенно хорошо он зарекомендовал себя в создании несложных устройств вроде датчиков или элементов умного дома [15].

Более современный ESP32 расширяет горизонты возможностей. Помимо Wi-Fi, он оснащен поддержкой Bluetooth, что делает его универсальным решением для проектов с беспроводной коммуникацией [4]. Впечатляет и техническое оснащение ESP32: двухъядерный процессор с тактовой частотой до 240 МГц и увеличенный объем оперативной памяти в 520 КБ позволяют справляться с более сложными задачами – от комплексного управления умным домом до обработки данных носимых устройств [11].

Нельзя не отметить значимость активного сообщества разработчиков, поддерживающих эти платформы. Обилие готовых библиотек, примеров кода и учебных материалов существенно облегчает процесс разработки. Такая поддержка особенно ценна для студентов и начинающих специалистов, делающих первые шаги в создании IoT-устройств.

Микроконтроллеры семейства STM32 от STMicroelectronics славятся своей надежностью и выдающейся производительностью. Основанные на архитектуре ARM Cortex-M, они представлены широчайшим спектром моделей

– от базовых до высокопроизводительных вариантов. Главное преимущество STM32 заключается в способности обрабатывать внушительные объемы данных и выполнять комплексные вычисления в режиме реального времени. Разнообразие поддерживаемых интерфейсов, включая SPI, I2C и UART, обеспечивает гибкость и многофункциональность этих микроконтроллеров. Особенно важным качеством становится экономичное энергопотребление, критичное для автономных устройств на батарейном питании – например, систем экологического мониторинга или датчиков окружающей среды.

STMicroelectronics обеспечивает разработчиков исчерпывающей документацией, готовыми примерами кода и специализированной средой разработки STM32Cube, значительно упрощающей создание и отладку приложений. Благодаря своей надежности и широким возможностям конфигурации, семейство STM32 приобрело популярность в профессиональных и коммерческих проектах.

Говоря о популярных платформах для интернета вещей, нельзя обойти вниманием Raspberry Pi, занимающую особую нишу среди IoT-решений. В отличие от ESP8266/ESP32 и STM32, Raspberry Pi представляет собой полноценный одноплатный компьютер с операционной системой, что открывает возможности для запуска сложных приложений и работы с графическим интерфейсом. Однако эти преимущества сопровождаются повышенным энергопотреблением и более высокой стоимостью. Raspberry Pi обычно применяется в проектах, требующих выполнения задач высокого уровня – например, серверных решениях или системах периферийного машинного обучения. Для простых устройств, таких как датчики или исполнительные механизмы, Raspberry Pi часто оказывается избыточной как по цене, так и по энергозатратам [4].

ESP8266/ESP32 и STM32 входят в число наиболее популярных микроконтроллеров в сфере разработки IoT-устройств. ESP8266 и ESP32 привлекают встроенной поддержкой беспроводных технологий, что делает их идеальными для проектов с подключением к интернету или коммуникацией через Bluetooth. STM32, в свою очередь, выделяется высокой производительностью и универсальностью, позволяющими использовать его в сложных и требовательных задачах [25].

Наличие большого сообщества, доступных библиотек и примеров кода существенно облегчает процесс разработки и внедрения IoT-приложений. Для студентов, работающих над дипломными проектами, эти платформы предоставляют прекрасную возможность воплотить свои идеи без столкновения с серьезными техническими трудностями на этапе обучения.

Каждый из рассмотренных микроконтроллеров обладает уникальными преимуществами, определяющими области их применения. ESP8266 [25] великолепно подходит для новичков и несложных проектов с базовой функциональностью – его часто можно встретить в датчиках, умных розетках и других бюджетных устройствах. Более мощный и универсальный ESP32 отлично справляется с задачами, требующими поддержки Wi-Fi и Bluetooth, а также хорошо зарекомендовал себя в носимых устройствах и системах умного дома. STM32 с его выдающейся производительностью и надежностью становится незаменимым в профессиональных проектах – от промышленной автоматизации до медицинского оборудования и систем мониторинга.

Raspberry Pi, хотя и не является классическим микроконтроллером, занимает особое место среди решений для интернета вещей. Возможности этого одноплатного компьютера выходят далеко за рамки типичных задач для микроконтроллеров, однако повышенное энергопотребление и стоимость ограничивают его применение в автономных и бюджетных проектах. Тем не менее, для задач, требующих запуска полноценной операционной системы или выполнения сложных алгоритмов, Raspberry Pi остается предпочтительным выбором.

1.4 Исследование и сравнительный анализ технологий

1.4.1 Сравнительный анализ микроконтроллеров

Микроконтроллеры являются ключевым элементом любой IoT-системы, обеспечивающим выполнение логики устройства и взаимодействие с внешними компонентами. Для выбора наиболее подходящего микроконтроллера были рассмотрены три популярные платформы: Arduino, ESP8266/ESP32 и STM32. Ниже представлен сравнительный анализ их характеристик (см. Таблицу 1.3).

Таблица 1.3 – Сравнение характеристик микроконтроллеров

Характеристика	**Arduino (UNO)**	**ESP266/ESP32**	**STM32**
Процессор	Atmega328P, 16 МГц	Tensilica LX6, до 240 МГц	ARM Cortex-M, до 72 МГц
Память (RAM)	2 КБ	520 КБ	До 192 КБ
Флеш-память	32 КБ	4 МБ	До 2 МБ
Wi-Fi/Bluetooth	Нет	Да	Опционально
Энергопотребление	Среднее	Низкое	Очень низкое
Сложность освоения	Низкая	Средняя	Высокая
Сообщество поддержки	Обширное	Широкое	Профессиональное

Arduino, в частности модель UNO, выделяется среди микроконтроллеров благодаря своей простоте и доступности, что делает его отличным выбором для новичков и образовательных целей [25]. Легкость освоения платформы обусловлена активным сообществом и обилием обучающих материалов. Однако, скромные технические характеристики, такие как процессор с частотой 16 МГц и всего 2 КБ ОЗУ, а также отсутствие встроенной беспроводной связи, ограничивают его применение сложными IoT-проектами, требующими интенсивной обработки данных или беспроводного обмена информацией.

Семейства микроконтроллеров ESP8266 и ESP32 представляют собой более современные и технологически продвинутые решения. Важным преимуществом является наличие встроенных модулей Wi-Fi и Bluetooth, избавляющих от необходимости использования дополнительных компонентов для беспроводной связи. Более высокая тактовая частота (до 240 МГц) и значительный объем памяти (520 КБ ОЗУ и до 4 МБ флеш-памяти) позволяют эффективно решать задачи, требующие обработки данных в реальном времени. Энергоэффективность этих микроконтроллеров делает их популярными при создании автономных устройств [6]. ESP32, будучи более продвинутой версией, используется в проектах, где критичны производительность и функциональность.

STM32 представляет собой профессиональное решение на базе архитектуры ARM Cortex-M. Эти микроконтроллеры характеризуются высокой производительностью (до 72 МГц) при минимальном энергопотреблении, что делает их подходящим выбором для энергоэффективных решений. Поддержка различных интерфейсов (SPI, I2C, UART) обеспечивает гибкость при интеграции с различными периферийными устройствами [22]. Однако сложность в освоении и необходимость глубоких технических знаний могут стать препятствием для начинающих, ограничивая его использование в основном профессиональными инженерами.

Области применения этих микроконтроллеров различаются в зависимости от их характеристик. Arduino идеально подходит для простых устройств, таких как датчики температуры, системы сигнализации или для образовательных целей [5]. ESP8266/ESP32 широко используются в системах умного дома, носимых устройствах и различных решениях для мониторинга, где необходима беспроводная связь с интернетом. STM32, в свою очередь, применяется в промышленных системах управления, медицинском оборудовании и других критически важных приложениях, где важны высокая производительность и надежность.

1.4.2 Сравнительный анализ фреймворков для мобильной разработки

Для реализации мобильного приложения были рассмотрены несколько популярных фреймворков: Flutter, React Native, Kotlin Multiplatform и нативная разработка. Каждый из них имеет свои преимущества и недостатки, которые необходимо учитывать при выборе (см. Таблицу 1.4).

Таблица 1.4 – Сравнительный анализ фреймворков

Фреймворк	**Язык программирования**	**Производительность**	**Кроссплатформенность**	**Локалиция/ Гибкость UI**
Flutter	Dart	Высокая	Полная	Высокая
React Native	JavaScript	Средняя	Полная	Средняя
Kotlin Multiplatform	Kotlin	Высокая (только Android)	Частичная	Средняя
Native Android/iOS	Java/Swift/Kotlin	Максимальная	Нет	Высокая

Flutter показывает отличную производительность за счет встроенного графического движка Skia и большого количества готовых виджетов [29]. Эти возможности делают его прекрасным решением для создания сложных пользовательских интерфейсов. Flutter также предлагает разработчикам удобные инструменты для локализации приложений и интеграции с внешними библиотеками, что заметно упрощает весь процесс разработки.

React Native [29] завоевал популярность среди разработчиков благодаря простоте изучения и активному сообществу, которое его поддерживает. Однако у этой платформы есть определенные ограничения при работе с нативными API операционных систем, особенно с Bluetooth и Wi-Fi интерфейсами. Такие ограничения могут создавать трудности при реализации некоторых функций в мобильных приложениях.

Kotlin Multiplatform изначально создавался с ориентацией на Android-разработку и имеет довольно ограниченную поддержку iOS. Эта особенность делает его менее привлекательным выбором для полноценной кроссплатформенной разработки мобильных приложений.

1.4.3 Сравнительный анализ способов связи

Для взаимодействия между мобильным приложением и микроконтроллером были рассмотрены различные способы связи: USB, Wi-Fi,

Bluetooth и Zigbee [9]. Каждый из них имеет свои особенности, которые определяют его применимость в различных сценариях (см. Таблицу 1.5).

Таблица 1.5 – Сравнение способов связи

Технология	**Скорость передачи данных**	**Дальность**	**Энергопотребление**	**Применимость**
USB	До 5 Гбит/с (USB 3.0)	Проводное	Среднее	Настройка и отладка устройств
Wi-Fi	Высокая	До 100 метров	Высокое	Умный дом, промышленность
Bluetooth BLE	До 2 Мбит/с	До 100 метров	Низкое	Носимые устройства, умный дом
ZigBee	До 250 Кбит/с	До 100 метров	Очень низкое	Автоматизация, IoT-сети

USB-соединение обеспечивает высокую скорость передачи данных – до 5 Гбит/с в версии 3.0, что делает его эффективным для операций с большими объёмами информации. Однако данная технология требует физического подключения, что ограничивает её применение в мобильных и масштабных системах. Поэтому USB чаще всего используется на этапах настройки и тестирования оборудования, где критичны стабильность соединения и минимальные задержки при передаче данных. Wi-Fi технология благодаря своей высокой скорости передачи данных и значительной дальности действия (до 100 метров) стала основой для большинства современных решений в сфере умного дома и промышленной автоматизации. Вместе с тем, её высокое энергопотребление делает Wi-Fi не самым подходящим вариантом для автономных устройств, работающих от батареек. Также стоит учитывать, что для нормального функционирования требуется наличие стабильной сетевой инфраструктуры, что может быть сложно реализовать в удалённых или труднодоступных местах. Тем не менее, Wi-Fi активно применяется в системах видеонаблюдения, комплексных решениях по управлению зданиями и других сценариях, где важна интеграция с уже существующей IT-инфраструктурой.

Bluetooth Low Energy (BLE), несмотря на относительно невысокую скорость передачи данных (до 2 Мбит/с) и ограниченную дальность действия, остаётся оптимальным выбором для устройств, где главной задачей является минимизация энергопотребления. Благодаря этой особенности BLE получил

широкое распространение в носимых гаджетах, таких как фитнес-трекеры и умные часы. Возможность длительной автономной работы делает эту технологию особенно актуальной для персональных устройств. В системах умного дома BLE чаще всего используется в компактных решениях, которым не требуется постоянное подключение к сети.

ZigBee, в свою очередь, демонстрирует исключительно низкое энергопотребление и при этом обеспечивает дальность действия до 100 метров, что сделало её стандартом для построения масштабируемых IoT-сетей. Однако невысокая скорость передачи данных (до 250 Кбит/с) и необходимость использования специализированного оборудования ограничивают область применения данной технологии. На практике ZigBee наиболее востребована в системах автоматизации зданий, где важно обеспечение надёжной работы множества взаимосвязанных датчиков и исполнительных механизмов при минимальном потреблении энергии.

ГЛАВА 2
ВЫБОР ТЕХНОЛОГИЙ И РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

2.1 Анализ существующих технологий и инструментов разработки

В современной цифровой среде мобильные приложения играют одну из ключевых ролей, становясь неотъемлемой частью повседневной жизни. Они затрагивают практически все аспекты – от развлечений и социальных сетей до бизнес-приложений и даже промышленных решений. При этом выбор платформы для разработки играет решающее значение, поскольку напрямую влияет на функциональность, производительность и удобство конечного продукта. На сегодняшний день в этой области выделяются два основных направления: нативная разработка под Android и iOS, а также кроссплатформенные подходы, представленные такими фреймворками, как Flutter и React Native. Каждый из этих вариантов обладает своими особенностями, которые необходимо учитывать при планировании проекта (см. Таблицу 2.1).

Операционная система Android занимает одно из лидирующих мест по распространённости, что связано с её использованием на миллиардах устройств по всему миру. Такая популярность делает её особенно привлекательной для разработчиков, желающих достичь максимального охвата аудитории. В качестве основных языков программирования здесь применяются Java и Kotlin, причём последний стал официальным языком разработки благодаря своей лаконичности, безопасности и простоте использования. Android SDK предоставляет широкий набор инструментов, позволяющий реализовать сложные функциональные возможности, включая работу с различными интерфейсами связи, такими как USB и Wi-Fi.

Несмотря на массовое распространение Android, система iOS представляет собой принципиально иной подход. Данная закрытая платформа, ориентированная исключительно на устройства Apple, что определяет особенности её архитектуры и процесса разработки. Для создания приложений под iOS чаще всего используются языки Swift или Objective-C, однако предпочтение всё чаще отдается Swift за счёт его современного синтаксиса и высокой читаемости кода. Разработка под iOS требует работы в специализированной среде Xcode, а публикация приложений возможна только через App Store после прохождения строгого ревью. Также стоит отметить необходимость покупки лицензии для размещения приложений, что может быть ограничением для начинающих разработчиков.

Вместе с тем, всё большее внимание уделяется кроссплатформенным решениям, которые позволяют создавать приложения сразу под несколько операционных систем, используя единую кодовую базу. Среди них выделяются такие фреймворки, как Flutter и React Native. Flutter, разработанный компанией Google, предлагает возможность одновременной разработки под Android и iOS с помощью одного языка – Dart. Хотя он менее распространён, чем JavaScript, его синтаксис считается понятным и логичным. Одним из главных преимуществ Flutter является наличие богатого набора готовых виджетов, что позволяет создавать визуально привлекательные и функциональные интерфейсы без необходимости глубокого дизайнерского программирования. Кроме того, активное развитие проекта и растущее сообщество делают его перспективным решением для долгосрочных проектов.

React Native, в свою очередь, был создан Facebook и основан на популярной библиотеке React, использующей язык JavaScript. Это делает его особенно привлекательным для веб-разработчиков, которые уже знакомы с данными технологиями и могут быстро освоить мобильную разработку. Приложения, созданные с помощью React Native, демонстрируют производительность, близкую к нативной, а их внешний вид хорошо адаптирован под стандарты каждой из платформ. Однако для реализации сложных функций, связанных с работой на уровне ОС или аппаратных компонентов, может потребоваться использование нативного кода, что частично снижает преимущество кроссплатформенности.

Таблица 2.1 – Сравнительный анализ платформ разработки

Параметр	Android	iOS	Flutter	React Native
Язык программирования	Java, Kotlin	Swift, Objective-C	Dart	JavaScript, TypeScript
Производительность	Высокая	Очень высокая	Близкая к нативной	Близкая к нативной
Скорость разработки	Средняя	Средняя	Высокая	Высокая
Доступность инструментов	Android Studio	Xcode	Visual Studio Code, IntelliJ, Android Studio	Visual Studio Code
Целевая аудитория	Широкая	Устройства Apple	Мультиплатформенная	Мультиплатформенная
Сложность обучения	Средняя	Средняя	Низкая	Низкая

Выбор языка программирования и среды разработки играет важную роль при создании мобильного приложения, поскольку от этого зависит не только скорость и удобство процесса разработки, но и качество конечного продукта. Например, в случае с Android использование Kotlin вместо Java может положительно сказаться на производительности команды – за счёт более лаконичного и безопасного кода уменьшается вероятность ошибок, а также ускоряется сам процесс написания приложения. Для платформы iOS предпочтение стоит отдавать Swift, поскольку этот язык сочетает в себе простоту использования и высокую производительность, что делает его оптимальным выбором для современных задач.

Если речь идёт о кроссплатформенной разработке, то выбор языка зависит не только от индивидуальных предпочтений разработчиков, но и от конкретных требований проекта, наличия специалистов в команде и перспектив дальнейшего развития приложения. В этом случае важно учитывать не только технические характеристики языка, но и экосистему вокруг него, доступность документации и поддержку сообщества.

Не менее важное значение имеет и среда разработки, поскольку она обеспечивает необходимые инструменты для тестирования, отладки и оптимизации приложений. Для Android-разработки стандартом является Android Studio – мощная IDE, предоставляющая широкий функционал для всех этапов создания приложения. Для iOS такой средой выступает Xcode, которая

интегрирована в экосистему Apple и предоставляет разработчикам всё необходимое для реализации сложных решений. Что касается кроссплатформенных фреймворков, таких как Flutter и React Native, то здесь разработчики могут использовать уже знакомые им редакторы, например Visual Studio Code или IntelliJ IDEA, расширяя их функционал с помощью специализированных плагинов. Такой подход позволяет сохранить привычный рабочий процесс и повысить общую эффективность.

Особое внимание необходимо уделить реализации взаимодействия мобильного приложения с микроконтроллерами через USB и Wi-Fi. При работе с Android можно воспользоваться Android USB Host API, который предоставляет возможности для обнаружения и управления подключенными устройствами. Также существуют сторонние библиотеки, такие как usb-serial-for-android, которые упрощают работу с последовательными портами и позволяют быстрее интегрировать необходимую функциональность. В случае организации сетевого взаимодействия по Wi-Fi применяются различные библиотеки, например Retrofit и OkHttp, обеспечивающие удобную реализацию HTTP-запросов и управление сетевыми соединениями между мобильным приложением и микроконтроллером.

Для разработки приложений под iOS используется Network Framework, поддерживающий современные протоколы связи и обеспечивающий стабильное сетевое взаимодействие. Для задач IoT-направления особенно актуальны библиотеки, реализующие протоколы MQTT, CoAP и HTTP, среди которых можно выделить Eclipse Paho и Mosquitto. Эти решения помогают организовать передачу данных в реальном времени, что особенно важно для систем, где требуется оперативная реакция на изменение параметров (см. Таблицу 2.2).

Таблица 2.2 - Сравнение библиотек для работы с USB и Wi-Fi

Библиотека	**Поддержка платформ**	**Протоколы**	**Уровень сложности**
Android USB Host API	Android	USB	Средний
usb-serial-for-android	Android	USB	Низкий
Retrofit	Android, iOS	HTTP	Средний
OkHttp	Android, iOS	HTTP	Средний
Network Framework	iOS	Wi-Fi, Bluetooth	Низкий
Eclipse Paho	Кроссплатформенный	MQTT	Средний

В сфере IoT-разработки доступно множество современных фреймворков, которые упрощают процесс создания приложений для взаимодействия с микроконтроллерами. Одним из самых известных и доступных решений является Arduino IDE – эта среда предоставляет удобный интерфейс и базовые инструменты для программирования микроконтроллеров, включая поддержку USB и Wi-Fi. Простота использования делает её особенно популярной среди начинающих разработчиков, которые только осваивают работу с аппаратными устройствами.

Для более опытных пользователей подходит PlatformIO – это продвинутый инструмент, поддерживающий широкий спектр аппаратных платформ, включая ESP8266, ESP32 и STM32. Платформа предлагает гибкие настройки и возможность интеграции с такими популярными редакторами кода, как Visual Studio Code, что позволяет использовать привычные рабочие инструменты при разработке IoT-устройств.

Ещё одним интересным решением в этой области является Node-RED – визуальный инструмент для проектирования IoT-приложений, который позволяет соединять устройства и обрабатывать данные в виде графических потоков. Такой подход значительно упрощает разработку, поскольку не требует глубокого знания программирования. Node-RED широко используется для быстрого прототипирования и тестирования решений, особенно на ранних этапах проекта, когда важно оперативно проверить концепцию и собрать рабочую модель.

2.2 Обоснование выбора технологий

Выбор технологий для реализации проекта является ключевым этапом, определяющим успешность разработки и дальнейшее функционирование системы. В данном разделе представлено обоснование выбора микроконтроллера, фреймворка для мобильной разработки и способов связи между мобильным приложением и устройством.

2.2.1 Выбор микроконтроллера

В качестве аппаратной основы проекта был выбран микроконтроллер ESP32 производства компании Espressif Systems. Данное решение обладает комплексом технических характеристик, которые делают его оптимальным выбором для разработки IoT-устройств.

Основным преимуществом ESP32 является его высокая вычислительная мощность, обеспечиваемая двухъядерным процессором Tensilica LX6 с тактовой частотой до 240 МГц. Такая производительность позволяет эффективно обрабатывать значительные объемы данных в режиме реального времени.

Важным аспектом выбора стало наличие в микроконтроллере встроенных модулей беспроводной связи Wi-Fi и Bluetooth. Эта особенность устраняет необходимость использования дополнительных компонентов для организации беспроводного взаимодействия, что упрощает конструкцию устройства и снижает его себестоимость.

Особого внимания заслуживает энергоэффективность ESP32, обеспечиваемая поддержкой различных режимов энергосбережения. Данная характеристика делает микроконтроллер пригодным для работы в автономных системах с ограниченными источниками питания.

Дополнительным аргументом в пользу выбора ESP32 стала его экономическая доступность в сочетании с развитой экосистемой поддержки. Активное сообщество разработчиков обеспечивает постоянное обновление документации, создание специализированных библиотек и накопление базы готовых решений, что существенно ускоряет процесс разработки.

2.2.2 Выбор фреймворка для мобильной разработки

При разработке мобильного приложения выбор пал на фреймворк Flutter, что стало результатом сравнительного анализа современных кроссплатформенных решений, включая React Native и Kotlin Multiplatform. Ключевым аргументом в пользу Flutter выступила его высокая производительность, достигаемая благодаря использованию графического движка Skia, который гарантирует плавную анимацию и мгновенный отклик пользовательского интерфейса.

Существенным преимуществом Flutter является его кроссплатформенная природа, позволяющая создавать приложения, одинаково эффективно работающие на платформах Android и iOS. Данное качество не только сокращает временные затраты на разработку, но и существенно уменьшает финансовые вложения в проект. Особого внимания заслуживает встроенная система локализации, реализованная через arb-файлы, которая значительно упрощает процесс адаптации интерфейса для различных языковых рынков.

Важным фактором выбора стало активное развитие экосистемы Flutter, поддерживаемой компанией Google. Многочисленное сообщество разработчиков постоянно пополняет коллекцию доступных плагинов и библиотек, ускоряя процесс создания приложений. Отдельно стоит отметить прекрасную совместимость Flutter с технологиями работы с Wi-Fi и USB, что делает его оптимальным решением для разработки приложений, взаимодействующих с микроконтроллерами ESP32.

2.2.3 Выбор способов связи

Для обеспечения взаимодействия между мобильным приложением и микроконтроллером в проекте были выбраны два способа связи – Wi-Fi и USB. Основным протоколом стал Wi-Fi, поскольку он обладает рядом важных преимуществ. Высокая скорость передачи данных позволяет оперативно обрабатывать команды и получать обратную связь, что особенно критично для приложений, работающих в режиме реального времени. Возможность подключения нескольких устройств к одной сети делает Wi-Fi подходящим решением для масштабных систем, таких как умный дом или автоматизированные промышленные комплексы. Также важно, что через Wi-Fi можно управлять устройствами удалённо, из любой точки мира, где есть интернет. Так значительно повышается удобство использования системы и расширяет её функциональные возможности.

Однако на этапе разработки и тестирования также была реализована поддержка USB. Этот способ используется только тогда, когда беспроводное соединение недоступно или требуется более стабильное физическое подключение. USB оказался удобным при начальной настройке устройства, так как не требует дополнительной конфигурации сетевых параметров и обеспечивает надёжное соединение. После завершения отладки этот метод больше не применяется, поскольку вся основная работа системы осуществляется через Wi-Fi.

Bluetooth, несмотря на то, что он поддерживается микроконтроллером ESP32, не был выбран в качестве одного из способов связи. Среди основных причин – ограниченный радиус действия (до 10 метров), что делает его малопригодным для использования в больших помещениях или на открытых территориях. Ещё одна проблема – нестабильная работа при подключении множества устройств одновременно. Кроме того, для полноценного функционирования Bluetooth в составе умного дома часто требуется использование дополнительного шлюза или хаба, что усложняет архитектуру и увеличивает стоимость решения. Всё это стало основанием для отказа от Bluetooth в пользу более универсальных и надёжных вариантов.

2.3 Проектирование архитектуры приложения

2.3.1 Описание архитектуры клиент-серверного взаимодействия

Разрабатываемая система построена на основе клиент-серверной архитектуры, которая обеспечивает взаимодействие между мобильным приложением и микроконтроллером ESP32 с использованием HTTP-протокола

(см. Таблицу 2.3). В такой организации логика работы системы разделена: клиентская часть реализована в мобильном приложении, а серверная – на стороне микроконтроллера. Это позволяет повысить надёжность всей системы и даёт возможность её дальнейшего масштабирования [1]. Для защиты от несанкционированного доступа в системе реализована авторизация с использованием токенов, что добавляет дополнительный уровень безопасности.

Как показано в Таблице 2.3, обмен данными между компонентами организован следующим образом: мобильное приложение отправляет управляющие команды на микроконтроллер. Эти команды могут быть разными – например, включение или выключение светодиодной ленты, изменение цвета или уровня яркости. Микроконтроллер ESP32 принимает эти запросы, проверяет наличие действительного токена авторизации, выполняет указанную команду и формирует ответ, в котором указывается статус выполнения операции.

Для связи между мобильным приложением и микроконтроллером используется беспроводное соединение по Wi-Fi. Применение стандартного HTTP-протокола делает систему более универсальной и совместимой с различными платформами, что упрощает как разработку, так и последующую отладку. Также предусмотрена поддержка проводного USB-подключения, которое применяется в основном на этапе начальной настройки и тестирования системы. Такое сочетание беспроводной и проводной передачи данных позволяет создать устойчивую и надёжную систему взаимодействия между мобильным приложением и микроконтроллером, используя лучшие стороны каждой из технологий.

Таблица 2.3 – Принципы взаимодействия компонентов

Компонент	Роль	Ключевые функции
Мобильное приложение	Клиент	Отправление команд, обработка ответов, отображение состояния устройств
Микроконтроллер ESP32	Сервер	Приём запросов, проверка токенов, выполнение команд, возврат статуса операции
Wi-Fi	Основной канал связи	Обеспечение беспроводного взаимодействия
USB	Вспомогательный канал связи	Обеспечение стабильности соединения для настройки и отладки

2.3.2 Разработка структуры приложения: модули, классы, функции

Архитектура приложения была разработана с соблюдением принципов модульности и чистого кода, что способствует повышению удобства разработки, упрощает процесс тестирования и создает основу для будущего расширения функциональных возможностей. Приложение организовано в виде набора взаимосвязанных модулей, каждый из которых выполняет строго определенный набор функций (см. Таблицу 2.4).

Модуль авторизации отвечает за аутентификацию пользователей и процедуру генерации уникальных токенов доступа. В его задачи входит безопасное хранение и извлечение токенов из локального хранилища устройства, что позволяет снизить вероятность утечки конфиденциальных данных и упрощает процесс повторной аутентификации при последующих обращениях к микроконтроллеру.

Функциональный модуль управления устройством обеспечивает двустороннюю коммуникацию с микроконтроллером, включая отправку управляющих команд и обработку поступающих ответов. Этот компонент системы реализует базовые операции управления, такие как включение и

выключение устройства, регулировка параметров освещения (цвет, яркость), а также активацию различных световых эффектов.

Модуль голосового управления предоставляет пользователям возможность взаимодействовать с системой через голосовые команды (см. Приложение А). Он включает механизмы распознавания речи и преобразования устных команд в конкретные действия, что позволяет управлять устройством без необходимости использования сенсорного интерфейса.

Пользовательский интерфейс, реализованный в виде отдельного модуля, объединяет все экранные формы и элементы управления приложения. Этот компонент разработан с учетом принципов юзабилити и обеспечивает интуитивно понятное взаимодействие с системой через различные элементы управления, включая кнопки, ползунки и другие интерактивные компоненты.

Таблица 2.4 – Структура приложения

Модуль	**Функционал**	**Ключевые компоненты**
Авторизация	Управление токенами, хранение данных	TokenManager, AuthService
Управление устройством	Отправка команд, обработка ответов	LedController, HttpService
Голосовое управление	Распознавание речи, выполнение команд	SpeechService, VoiceCommandHandler
Пользовательский интерфейс	Экраны, виджеты, навигация	HomePage, main

2.3.3 Создание пользовательского интерфейса (UI/UX-дизайн)

Проектирование пользовательского интерфейса осуществлялось с учетом современных стандартов UX/UI-дизайна, основной целью которых является создание удобного и интуитивно понятного взаимодействия пользователя с приложением. Особое внимание уделялось не только эстетической составляющей интерфейса, но и его функциональности, чтобы обеспечить простоту управления устройством без необходимости специальной подготовки.

В основе проектирования лежали ключевые принципы организации пользовательского интерфейса (см. Таблицу 2.5). Навигационная система была максимально упрощена за счет использования нижней панели навигации, содержащей все основные функции приложения. Такой подход позволяет пользователю быстро находить нужные элементы управления, минимизируя количество требуемых действий.

Все интерактивные элементы интерфейса, включая кнопки, ползунки и другие элементы управления, были выполнены в увеличенном размере и снабжены текстовыми подписями. Это решение значительно повышает удобство взаимодействия с приложением, особенно при использовании на мобильных устройствах. Основные функции управления устройством, такие как включение/выключение, регулировка цвета и яркости, были вынесены на главный экран для обеспечения максимально быстрого доступа.

Особое внимание было уделено адаптивности интерфейса. С помощью технологий MediaQuery и LayoutBuilder было обеспечено корректное отображение на устройствах с различными размерами экранов и разрешениями. Для визуального представления языковых опций были использованы SVG-изображения флагов, интегрированные в приложение посредством библиотеки flutter_svg, что позволило сохранить четкость графики на любых устройствах.

Таблица 2.5 – Особенности пользовательского интерфейса

Характеристика	**Описание**	**Инструменты реализации**
Навигация	Удобная панель навигации с минимальным количеством действий	BottomNavigationBar
Адаптивность	Корректное отображение на устройствах с разных разрешением	MediaQuery, LayoutBuilder
Локализация	Поддержка трёх языков: русский, английский, арабский	.arb-файлы, flutter_localization
Визуальная обратная связь	Уведомление об успешности выполнения команды или возможных ошибках	SnackBar, Dialogs

2.3.4 Алгоритмы обработки данных, передаваемых между мобильным приложением и микроконтроллером

Обработка данных между мобильным приложением и микроконтроллером реализована с использованием JSON-формата для передачи команд. Это

позволяет обеспечить универсальность и совместимость с различными платформами.

Для обратной связи используется механизм потокового обновления интерфейса, который позволяет отображать текущее состояние устройства в реальном времени. Например, если пользователь меняет яркость, интерфейс автоматически обновляется, отражая изменения (см. Таблицу 2.6).

Таблица 2.6 – Этапы обработки данных

Этап	**Описание**	**Технологии реализации**
Формирование команды	Создание JSON-объекта с данными для отправки	Dart, Flutter
Отправка запроса	Передача команды на микроконтроллер через HTTP-запрос	http.post
Обработка ответа	Анализ статуса выполнения операции и отображения результатов	StreamBuilder, State Managment
Обновление интерфейса	Автоматическое отображения изменений в реальном времени	setState, Provider

2.3.5 Сравнительный анализ разрабатываемого приложения с аналогами

Для сравнения разрабатываемого мобильного приложения были выбраны два аналога: LED Light Controller Remote + (см. рисунок 2.1) и HappyLighting (см. рисунок 2.2). Эти приложения широко используются для управления светодиодными лентами через микроконтроллеры и имеют схожие функциональные возможности. Оба поддерживают управление цветом и яркостью, работают на Android и iOS, а также позволяют управлять устройствами по Wi-Fi или Bluetooth.

Рисунок 2.1 – Интерфейс приложения LED Light Controller Remote +

Рисунок 2.2 – Интерфейс приложения HappyLighting

Сравнение проводилось по следующим параметрам: кроссплатформенность, поддержка языков интерфейса, наличие голосового управления, возможность настройки цвета и яркости, способы связи с устройством и уровень адаптации под разные экраны (см. Таблицу 2.7).

Таблица 2.7 – Сравнительный анализ разрабатываемого приложения с аналогами

Параметр	**Разрабатываемое приложение**	**LED Ligth Controller Remote+**	**HappyLighting**
Кроссплатформенность	Да (Flutter)	iOS	Android, iOS
Поддержка языков интерфейса	Русский, английский, арабский	Английский	Подстраивается под системные настройки
Голосовое управление	Есть, с поддержкой нескольких языков	Нет	Частично, может синхронизироваться с музыкой
Настройка цвета и яркости	Ползунки, палитра, эффекты, настройки пикселей	Базовые настройки	Базовые настройки
Способы связи с устройством	Wi-Fi, USB	Wi-Fi, Bluetooth	Wi-Fi, Bluetooth
Адаптации под экраны	Полная	Частичная	Частичная

Кроссплатформенность была реализована с помощью фреймворка Flutter. Это позволило создать единый интерфейс, который одинаково хорошо работает как на Android, так и на iOS. Из изученных аналогов только HappyLight поддерживается на обеих платформах. LED Light Controller Remote+ сделан под операционную систему iOS.

Поддержка нескольких языков стала важной особенностью разработанного приложения. В отличие от других решений, оно предоставляет пользователю возможность переключать язык прямо в настройках без перезагрузки. В разрабатываемом приложении доступны 3 языка: английский, русский и арабский, в то время как в приложении LED Light Controller Remote+ доступен только английский. HappyLight способен подстраиваться под системные настройки мобильного устройства.

Голосовое управление было внедрено с помощью библиотеки speech_to_text. Эта функция позволяет управлять освещением без использования рук. Такой подход особенно полезен для людей с ограниченными возможностями или в ситуациях, когда удобнее говорить, чем нажимать кнопки. У Light Controller Remote+ эта функция отсутствует, но у HappyLight

присутствует функция синхронизации работы светодиодной ленты с музыкой из приложения.

Настройка цвета и яркости в разрабатываемом приложении выполнена более гибко. Доступны ползунки, палитра цветов и световые эффекты, а также возможность настраивать отдельные пиксели, тем самым создавая освещение под себя. В аналогах настройки ограничены базовыми функциями, что снижает уровень контроля над освещением.

Связь с устройством осуществляется через Wi-Fi и USB. Это расширяет возможности использования: Wi-Fi применяется для удалённого управления, а USB обеспечивает стабильное соединение при настройке и отладке. Конкуренты в основном используют Wi-Fi и Bluetooth, что не всегда гарантирует надёжную связь.

Адаптация под экраны была реализована с использованием современных инструментов Flutter. Это позволило корректно отображать интерфейс на устройствах с любыми диагоналями и разрешениями. В аналогах подобная адаптивность встречается частично, из-за чего могут возникнуть проблемы с отображением на некоторых устройствах.

В результате можно сделать вывод, что разрабатываемое приложение превосходит аналоги по таким параметрам, как кроссплатформенность, поддержка нескольких языков, наличие голосового управления, гибкость настройки цвета и яркости, а также уровень адаптации под разные экраны. Оно предлагает пользователям современный и удобный инструмент для управления IoT-устройствами. При этом аналоги показывают схожую эффективность в базовых функциях управления освещением, однако их возможности ограничены в вопросах удобства и дополнительных функций.

2.4 Интеграция протоколов связи в приложение

Интеграция протоколов связи является важным этапом разработки мобильного приложения, так как от неё зависит стабильное взаимодействие с микроконтроллером ESP32. В рамках проекта были реализованы поддержка USB-подключения и настройка Wi-Fi-соединения, а также проведено тестирование их устойчивости и производительности.

USB-подключение используется в первую очередь для начальной настройки и отладки устройства, поскольку обеспечивает надёжное физическое соединение. Для реализации этой функции в мобильном приложении применялись стандартные инструменты, такие как Android USB Host API, позволяющий обнаруживать и взаимодействовать с подключёнными устройствами. Благодаря высокой скорости передачи данных (до 5 Гбит/с в версии USB 3.0), USB хорошо подходит для задач, где критична минимальная

задержка. Также стоит отметить, что через USB можно передавать не только данные, но и питание, что особенно полезно для маломощных устройств. Однако на iOS интеграция осложняется необходимостью использования специального протокола External Accessory Framework, в отличие от более гибкой реализации в Android. Для упрощения работы с последовательным портом была использована библиотека usb-serial-for-android, которая предоставляет удобный интерфейс для отправки и получения данных. Примером может служить использование плат Arduino [6] с USB-интерфейсом, где мобильное приложение отправляет команды микроконтроллеру по последовательному каналу.

Настройка Wi-Fi-соединения стала основным способом взаимодействия между приложением и микроконтроллером из-за его беспроводной природы и возможности удалённого управления. Для обмена данными использовался HTTP-протокол, который обеспечивает простоту реализации и совместимость с большинством систем. Микроконтроллер ESP32 мог работать как в режиме точки доступа, так и подключаться к уже существующей Wi-Fi сети. Для защиты передаваемых данных был внедрён механизм токенов, которые добавлялись в заголовок каждого запроса. Команды формируются в формате JSON и отправляются через POST-запросы. Сервер на стороне микроконтроллера принимает и обрабатывает эти команды, после чего возвращает ответ с информацией о результате выполнения. Основными преимуществами Wi-Fi являются высокая скорость передачи и возможность интеграции с интернетом, что позволяет строить облачные решения. Однако этот протокол потребляет больше энергии по сравнению с другими технологиями, что может быть ограничивающим фактором для автономных устройств.

Для проверки стабильности работы протоколов USB и Wi-Fi были проведены комплексные тесты. Измерялось время выполнения типовых команд – таких как включение или выключение устройства, изменение цвета светодиодной ленты и регулировка яркости. Также оценивалось время отклика системы в разных условиях эксплуатации и проверялась работа приложения при временных потерях соединения и его восстановлении. По результатам тестирования было установлено, что Wi-Fi обеспечивает хорошую скорость и удобство использования, особенно при удалённом управлении, тогда как USB демонстрирует высокую надёжность на этапах настройки и отладки. Таким образом, выбор между USB и Wi-Fi зависит от конкретных требований проекта, включая необходимость беспроводного доступа, уровень энергопотребления и условия использования устройства.

2.5 Добавление функций и улучшений

При модернизации мобильного приложения особое внимание было уделено внедрению новых функций, значительно расширяющих возможности пользователя по управлению светодиодной лентой. Ключевыми улучшениями стали реализация голосового управления, многоязычная поддержка интерфейса и усовершенствование системы контроля параметров освещения.

Голосовое управление в разработанном приложении реализовано с учётом требований к удобству и точности. Оно позволяет пользователю управлять светодиодной лентой без использования рук, что особенно удобно в бытовых условиях. Система поддерживает распознавание команд на нескольких языках. Точность распознавания команд в тихой обстановке находится на высоком уровне. Однако в шумной среде эффективность системы снижается, что говорит о необходимости дальнейшей оптимизации.

Многоязычный интерфейс был реализован с использованием стандартных инструментов Flutter: flutter_localizations и intl, работающих на основе .arb-файлов. Приложение поддерживает три языковые версии (см. рисунок 2.3): английскую (как основную), русскую (для пользователей из стран СНГ) и арабскую (демонстрирующую корректную работу с правосторонним письмом). Особенностью реализации стала возможность мгновенного переключения языка в настройках приложения без необходимости его перезагрузки, что существенно повышает удобство использования. Каждый языковой вариант включает полную локализацию всех элементов интерфейса с сохранением единого стиля оформления.

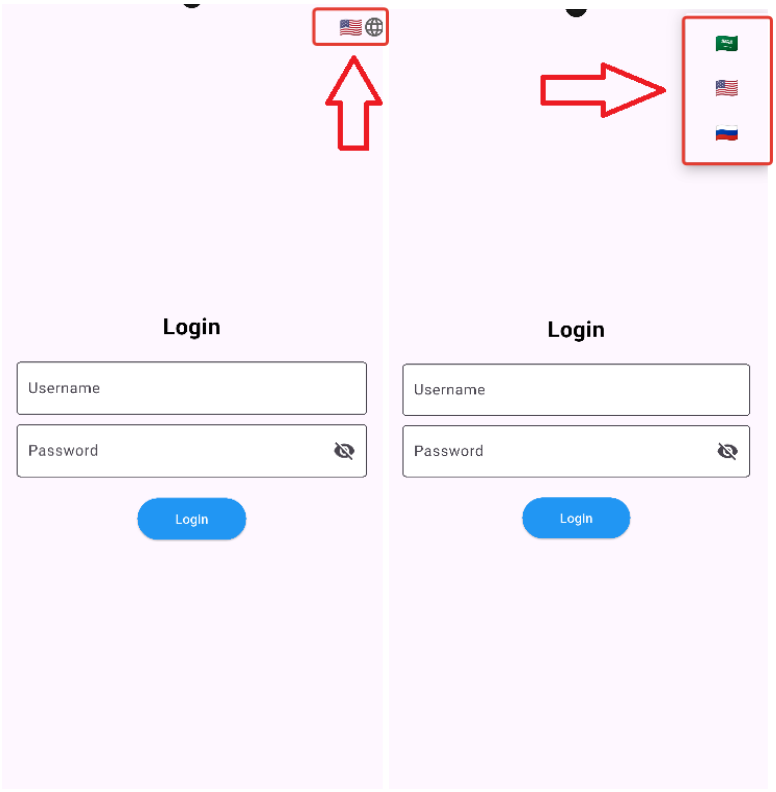

Рисунок 2.3 – Выбор языка приложения

Для повышения функциональности приложения были внедрены дополнительные возможности, направленные на улучшение пользовательского опыта. Одним из ключевых аспектов стала реализация управления цветом, что позволяет пользователю выбирать цвет из предложенной палитры или использовать заранее настроенные варианты, такие как красный, зелёный и синий. Для удобства добавлен ползунок, обеспечивающий плавную регулировку оттенков. Это делает процесс настройки более интуитивным и гибким.

Важной особенностью приложения стало управление световыми эффектами (см. Приложение Б). Было реализовано несколько популярных эффектов, таких как мерцание, плавное изменение цвета и радужная подсветка. Пользователь может переключаться между этими режимами как через кнопки интерфейса, так и с помощью голосовых команд, что значительно расширяет спектр возможностей для взаимодействия с системой. Такая функция особенно актуальна в условиях современного стремления к удобству и автоматизации.

Кроме того, была добавлена возможность регулировки яркости освещения. Для этого используется ползунок, позволяющий изменять уровень яркости в диапазоне от 0 до 100%. Стоит отметить, что значение яркости сохраняется после перезапуска приложения, что обеспечивает непрерывность работы и исключает необходимость повторной настройки параметров. Это решение значительно упрощает использование системы, особенно в случаях, когда пользователь предпочитает фиксированный уровень яркости для определённых сценариев.

ГЛАВА 3
РЕЗУЛЬТАТЫ РАЗРАБОТКИ И ТЕСТИРОВАНИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

3.1 Реализация функционала приложения

Разработка мобильного приложения для управления светодиодной лентой с использованием микроконтроллера ESP-32 являлась ключевым этапом проекта, в рамках которого были реализованы различные функциональные задачи. Этот процесс начался с анализа требований к функционалу и выбора технологий, а затем перешел к реализации программного кода, интеграции различных способов связи (USB и Wi-Fi), внедрению базовых операций и добавлению дополнительных функций, которые делают взаимодействие с устройством максимально удобным.

Процесс создания приложения был начат с глубокого погружения в его будущие возможности. Было решено использовать Flutter – современный фреймворк, который позволяет создавать кроссплатформенные приложения, совместимые как с Android, так и с iOS. Этот выбор был обусловлен не только универсальностью Flutter, но и широким набором готовых виджетов, что значительно упрощает процесс разработки пользовательского интерфейса.

Для взаимодействия с микроконтроллером были выбраны HTTP-запросы для Wi-Fi и сторонние библиотеки (usb_serial или flutter_libserialport) для USB. Все команды передаются в формате JSON, что обеспечивает универсальность системы и возможность её легкого расширения. Например, команда для изменения цвета светодиодной ленты выглядит следующим образом (см. рисунок 3.1).

Рисунок 3.1 – Содержание POST-запроса для изменения цвета

Такая организация данных позволяет легко добавлять новые параметры без необходимости кардинальных изменений в архитектуре системы.

На этапе написания кода особое внимание уделялось деталям. Каждая команда тщательно формировалась и проверялась на корректность. Для

отслеживания работы системы на каждом этапе были добавлены команды print(), которые позволяли фиксировать поток вызовов функций, перехватывать внутренние сообщения и выявлять возможные ошибки. Это помогло создать надёжное и стабильное приложение, способное работать в реальных условиях.

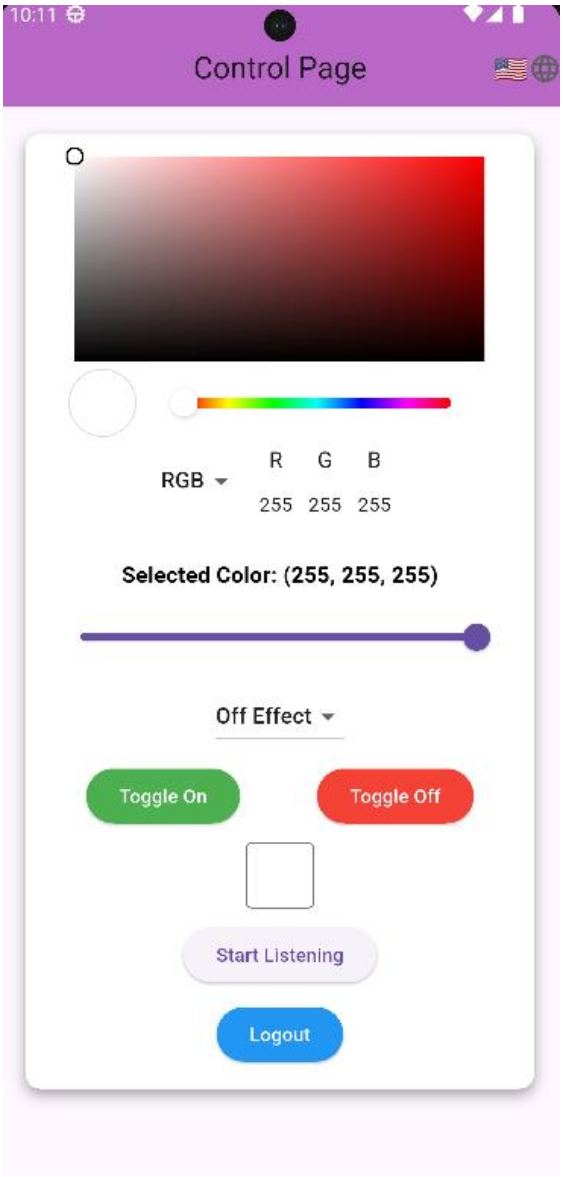

Рисунок 3.2 – Главный экран управления приложения

Один из важнейших аспектов проекта – обеспечение гибкости взаимодействия с микроконтроллером. Для этого были реализованы два режима связи: Wi-Fi и USB.

Wi-Fi стал основным способом связи благодаря своей универсальности и возможности удалённого управления устройством. Используются стандартные HTTP-запросы POST к заранее известному IP-адресу ESP-32. Команды отправляются в формате JSON, что обеспечивает удобство обработки данных на

стороне сервера. Серверная часть микроконтроллера выполняет парсинг данных и выполнение соответствующих действий.

Примеры команд (см. рисунок 3.3 и рисунок 3.4):

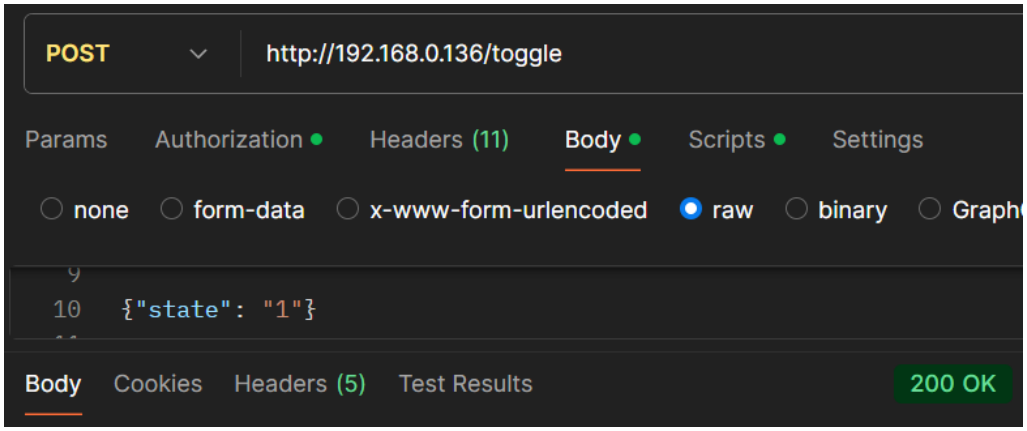

Рисунок 3.3 – POST-запрос на включение устройства

Рисунок 3.4 – POST-запрос на изменение цвета пикселя

USB-подключение играет значительную роль в проекте, особенно для отладки и настройки устройства. Оно реализовано через библиотеки usb_serial или flutter_libserialport. Данный режим обеспечивает стабильное соединение и минимальные задержки, что особенно полезно при первоначальной настройке системы.

Интеграция обоих режимов была протестирована на различных устройствах, что подтвердило их надёжность и совместимость.

Базовые операции включают управление состоянием светодиодной ленты: включение/выключение, изменение цвета и регулировку яркости. Эти функции были реализованы с использованием интуитивно понятного интерфейса, который минимизирует количество действий пользователя.

Например, для включения/выключения устройства используется кнопка на главном экране. При этом команда отправляется на микроконтроллер, который изменяет состояние светодиодов. Изменение цвета осуществляется через экран с палитрой цветов и предустановленными вариантами (красный, зелёный, синий и т.д.). Пользователь может выбрать цвет из палитры или установить его вручную через RGB-слайдеры.

Регулировка яркости реализована через ползунок, который позволяет изменять яркость в диапазоне от 0 до 100%. Значение сохраняется после перезапуска приложения, что делает взаимодействие с устройством ещё более удобным (см. рисунок 3.5).

Кроме того, был создан отдельный интерфейс для детального управления цветом каждого отдельного пикселя на ленте. Это особенно актуально для создания пользовательских эффектов и сценариев.

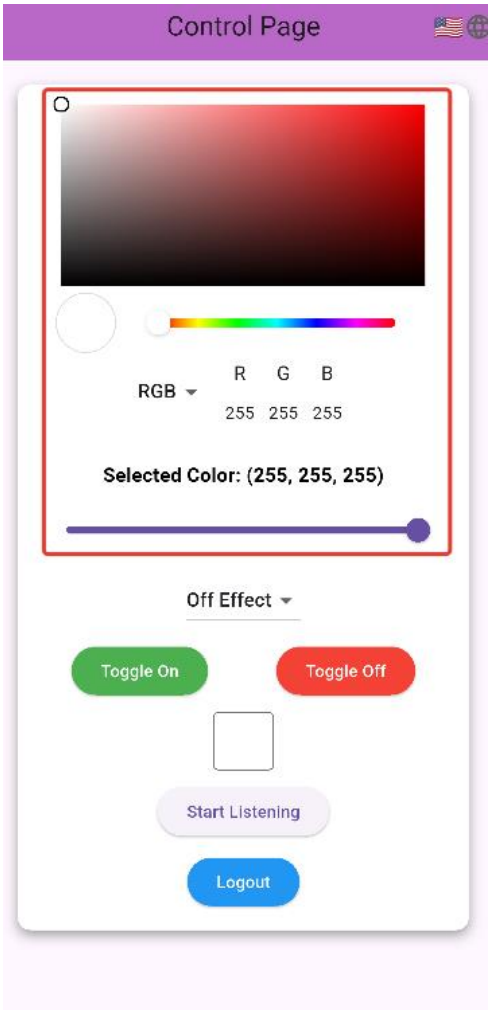

Рисунок 3.5 – Управление цветом и яркостью

В ходе совершенствования приложения были реализованы значимые функциональные улучшения, направленные на повышение удобства и

универсальности использования. Центральным нововведением стала система голосового управления, разработанная с применением библиотеки speech_to_text. Данная технология открывает принципиально новые возможности взаимодействия, позволяя управлять устройством без необходимости физического контакта с экраном. Это решение особенно актуально для ситуаций, когда пользователь занят другими делами, а также значительно расширяет доступность приложения для людей с ограниченными возможностями. Система распознает широкий набор команд, включающих базовые операции управления питанием («turn on»/«turn off»), выбор цветовых режимов («red», «green», «blue») и активацию различных световых эффектов («effect»). Для обеспечения прозрачности процесса взаимодействия реализована комплексная визуальная обратная связь: динамически изменяющаяся иконка микрофона, отображение распознанного текста в реальном времени и система уведомлений о возможных ошибках распознавания.

Особое внимание было уделено вопросам локализации интерфейса, которая выходит за рамки простого перевода текстовых элементов. Реализованный подход предполагает глубокую культурную адаптацию приложения, учитывающую языковые особенности, традиции восприятия информации и повседневные практики пользователей из разных стран. Это позволяет создать у международной аудитории ощущение естественности и комфорта при работе с приложением. Важной особенностью стало внедрение механизма динамического переключения языков, дающего возможность изменять язык интерфейса непосредственно в настройках приложения без необходимости его перезапуска.

Значительно расширены возможности управления световыми эффектами (см. рисунок 3.6). В приложение добавлен набор предустановленных режимов работы светодиодной ленты, включая различные варианты мерцания, радужную подсветку и другие визуальные эффекты. Пользователь может выбирать нужные режимы как традиционным способом через интерфейс приложения, так и с помощью голосовых команд, что обеспечивает дополнительную гибкость взаимодействия. Все новые функции были интегрированы в существующую архитектуру приложения с сохранением стабильности работы и отзывчивости интерфейса.

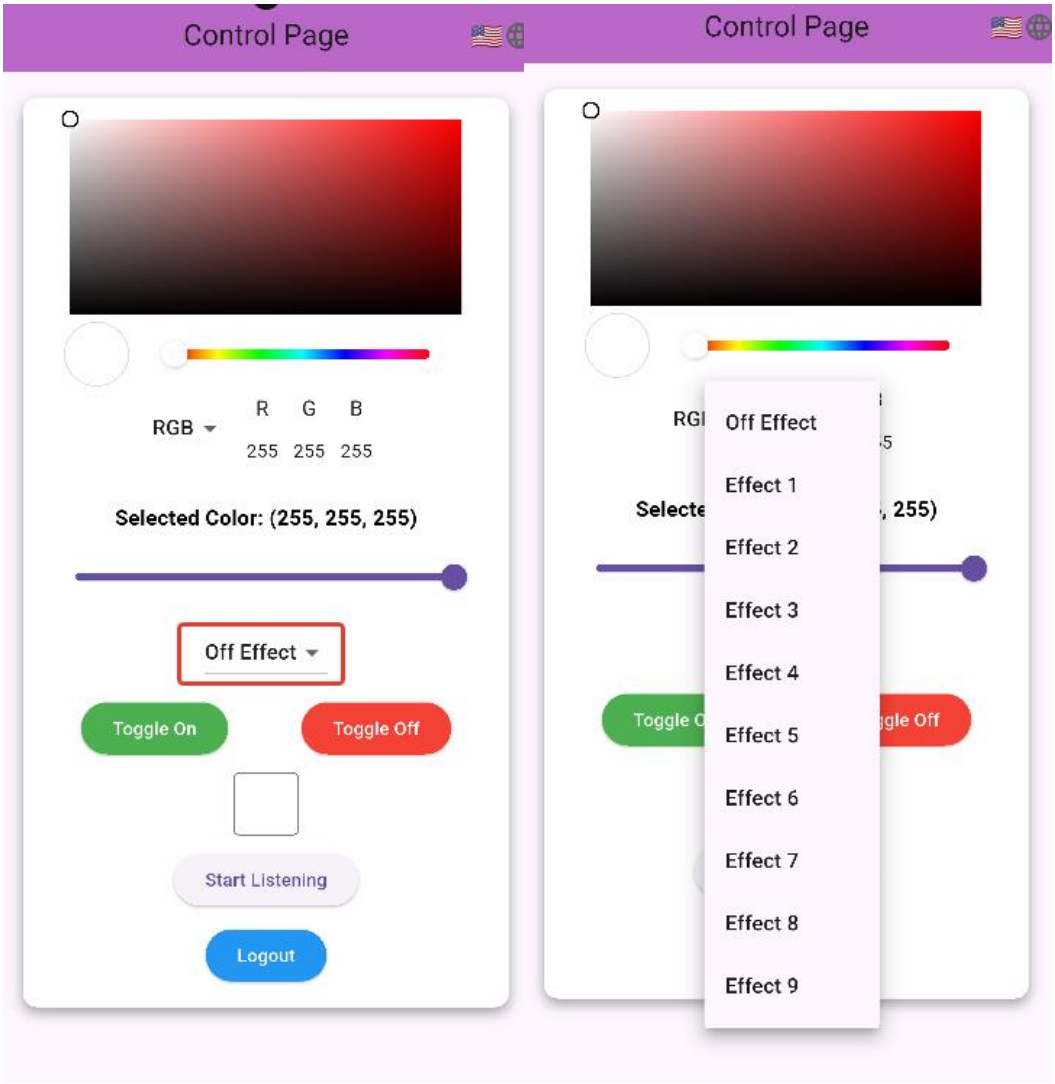

Рисунок 3.6 – Выпадающий список с выбором доступных эффектов

Особое внимание в процессе разработки уделялось адаптации интерфейса для различных устройств и культурных особенностей пользователей, что является важным аспектом создания удобного и интуитивно понятного приложения. Для достижения этой цели был внедрен адаптивный дизайн с использованием инструментов MediaQuery, которые позволяют получать информацию о размерах экрана, плотности пикселей и ориентации устройства. На маленьких экранах блоки управления автоматически группируются вертикально, при этом особый акцент сделан на наиболее часто используемые функции, такие как включение и выключение устройства, регулировка яркости и выбор цвета. На широких экранах применяется горизонтальная компоновка элементов. Каждый блок занимает отведенную ему зону и не перекрывает соседние элементы. Такое решение делает интерфейс удобным для использования на устройствах с различной диагональю экрана и разрешением.

Интерфейс создавался с упором на минималистичный подход. Это позволяет избежать перегрузки пользователя лишней информацией. Каждый

элемент выполняет конкретную функцию и органично вписывается в общую структуру приложения.

Интерфейс поделен на отдельные визуальные секции. Каждая секция отвечает за определенную группу функций. Одна секция управляет освещением, другая позволяет выбирать цвет, третья регулирует яркость, а четвертая настраивает различные эффекты. Такое разделение упрощает восприятие интерфейса пользователями. Оно также помогает людям быстрее разобраться с приложением, даже при первом знакомстве с ним.

Лаконичный дизайн и продуманная организация элементов делают интерфейс интуитивно понятным. Пользователи могут легко взаимодействовать с приложением благодаря такому подходу к проектированию.

3.2 Тестирование приложения

Тестирование – это не просто формальность, а важнейший этап, который позволяет убедиться в надёжности и удобстве созданного продукта. В процессе разработки мобильного приложения для управления светодиодной лентой через микроконтроллер ESP-32 было уделено особое внимание проверке его функциональности, производительности и безопасности. Результаты тестирования показали, что приложение способно работать стабильно даже в условиях реальной эксплуатации, однако некоторые аспекты всё же требуют доработки.

Для оценки качества приложения были применены несколько методов тестирования.

Функциональное тестирование стало основой проверки. Тщательно тестировались все ключевые операции: авторизацию пользователя, управление светом (включение/выключение, изменение цвета, регулировка яркости), переключение эффектов и работу с голосовыми командами. Каждая функция проверялась на эмуляторе и реальных устройствах, чтобы исключить возможные ошибки. Например, команда «Яркость 50» должна была точно установить яркость на 50%, а голосовая команда «Красный» – мгновенно изменить цвет подсветки на красный.

Нагрузочное тестирование позволило оценить, как приложение справляется с интенсивным использованием. Мы отправляли множество запросов на микроконтроллер через Wi-Fi и USB, чтобы проверить скорость обработки данных и стабильность соединения. Результаты показали, что время отклика составляет менее 500 миллисекунд, что делает взаимодействие с устройством практически мгновенным.

Юзабилити-тестирование помогло понять, насколько удобен интерфейс для конечного пользователя. Было обращено внимание на такие детали, как

размер кнопок, чёткость текстовых подписей и логичность расположения элементов. Пользователям понравилась возможность быстро менять язык интерфейса в один клик или настраивать цветовую палитру без лишних переходов. Особенно высоко оценили минималистичный дизайн, где каждый элемент имеет своё назначение и не перегружает экран избыточной информацией.

В процессе тестирования особое внимание было уделено анализу производительности системы при взаимодействии мобильного приложения с микроконтроллером ESP-32. Проведенные измерения позволили получить важные данные о рабочих характеристиках системы.

Скорость передачи данных была измерена в условиях реального использования приложения. HTTP-запросы, содержащие управляющие команды, обрабатываются в среднем за 300-500 миллисекунд, что является удовлетворительным показателем для задач управления освещением в реальном времени. Такой временной интервал обеспечивает комфортное взаимодействие пользователя с системой без заметных задержек.

Надежность соединения была подтверждена в ходе продолжительных испытаний. Беспроводное Wi-Fi подключение продемонстрировало стабильную работу без сбоев и потерь данных на протяжении всего периода тестирования. Проводное USB соединение использовалось главным образом на этапе отладки. Оно продемонстрировало высокую надежность. Это качество особенно ценно при диагностике и настройке системы.

Важной частью исследования стало измерение временных задержек при выполнении команд. Полученные данные показали, что система обеспечивает минимальные задержки при обработке управляющих сигналов. Это хорошо заметно при переключении между различными световыми эффектами. Переходы происходят плавно и без видимых прерываний в работе светодиодной ленты.

Для комплексного анализа производительности применялись специализированные инструменты. Использовались Flutter DevTools и Postman. Эти средства помогли своевременно находить и устранять потенциальные проблемы на этапе разработки. В приложении была реализована система логирования. Она обеспечила возможность детального отслеживания выполнения каждой команды (см. рисунок 3.7). Запись всех операций в лог-файлы заметно упростила процесс анализа работы системы. Также это помогло выявлять возможные узкие места в производительности.

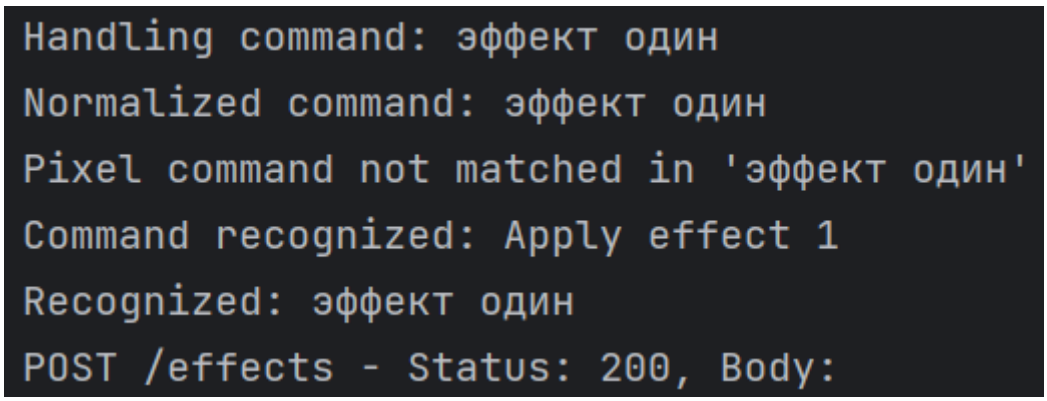

Рисунок 3.7 – Лог голосовой команды

Результаты тестирования показали успешное выполнение всех заявленных функций приложения. При детальном анализе логов и изучении обратной связи от пользователей было выделено несколько направлений для дальнейших улучшений. Ключевым аспектом стала необходимость оптимизации работы голосового управления. Распознавание голосовых команд в целом функционирует правильно. Однако его точность снижается при высоком уровне шума в окружающей среде. Для повышения надежности системы планируется внедрение дополнительных алгоритмов. Они позволят уменьшить влияние внешних факторов на качество распознавания.

Важным направлением станет добавление автоматических тестов. На текущем этапе большая часть проверок выполнялась вручную. Это не всегда гарантирует полное покрытие всех возможных сценариев использования, несмотря на тщательность проверок. В будущем планируется реализация автоматизированных тестов для проверки критически важных функций. Это касается авторизации пользователей и отправки команд микроконтроллеру. Автоматические тесты не только сократят время на тестирование, но и повысят его эффективность. Они могут быстро выявлять ошибки на ранних этапах разработки.

Работа по адаптации интерфейса под различные устройства уже проделана. Однако остаются возможности для дальнейшей оптимизации. Особенно это касается устройств с небольшим разрешением экрана. На таких устройствах некоторые элементы интерфейса могут быть менее удобны для взаимодействия. Планируется уделить особое внимание улучшению отображения компонентов управления на таких устройствах. Это сделает приложение более доступным и удобным для широкого круга пользователей.

Безопасность передачи данных между приложением и микроконтроллером является одной из главных задач. Это особенно важно для IoT-устройств. Для повышения уровня безопасности был внедрен механизм авторизации на основе

токенов. Это позволяет исключить несанкционированный доступ к устройству и обеспечить правильную работу системы.

Для повышения безопасности системы были реализованы дополнительные меры защиты данных и предотвращения несанкционированного доступа. Для шифрования данных при передаче через Wi-Fi было внедрено использование протокола HTTPS. Он обеспечивает надежную защиту информации от перехвата злоумышленниками. Такой подход гарантирует конфиденциальность и защищенность всех данных, передаваемых между мобильным приложением и микроконтроллером.

Была реализована проверка валидности токенов на стороне сервера перед выполнением команд. Это исключило возможность выполнения неавторизованных запросов. Каждое действие теперь требует подтверждения подлинности пользователя через уникальный токен. Такой механизм значительно снижает риски атак, связанных с подделкой запросов или использованием украденных данных.

Особое внимание уделялось мониторингу активности пользователей. Для этого было налажено логирование всех попыток входа в систему. Это позволяет своевременно выявлять подозрительные действия и принимать меры по их предотвращению. Логи содержат информацию о времени, IP-адресах и результатах попыток авторизации. Это делает возможным детальный анализ в случае инцидентов.

3.3 Оптимизация и улучшение приложения

После реализации базового функционала мобильного приложения для управления светодиодной лентой через ESP-32 [5] начнётся этап его улучшения и оптимизации. Целью является повышение производительности, удобства использования и подготовки к публикации в магазинах приложений, таких как Google Play и App Store. Данный этап позволит устранить выявленные недочёты и расширить функционал на основе пользовательской обратной связи.

На стадии тестирования могут быть обнаружены проблемы, связанные с задержками интерфейса при работе с большими объёмами данных, например, при управлении отдельными пикселями ленты. Для диагностики проблемных участков кода планируется использовать инструмент Flutter DevTools [16].

Замедления при перерисовке экранов с множеством элементов будут снижены за счёт минимизации операций обновления и внедрения кэширования часто используемых компонентов. Ожидается увеличение потребления памяти при частом переходе между экранами и отправке серии команд микроконтроллеру. Эта проблема будет решена путём более эффективного

освобождения ресурсов и уменьшения объёма одновременно загружаемых данных.

Будут исправлены ошибки визуального оформления интерфейса такие, как некорректные размеры кнопок и отступы между элементами, что повысит удобство использования.

Особое внимание будет уделено времени отклика приложения на команды пользователя. За счёт оптимизации сетевых запросов и уменьшения объёма передаваемых данных ожидается достичь времени реакции в диапазоне 300-500 миллисекунд.

Планируется улучшить распознавание голосовых команд за счёт внедрения аудиофильтров для работы в шумной среде. Плавные переходы между экранами будут реализованы с использованием стандартных механизмов Flutter [16], включая Hero-анимации и предварительную загрузку данных.

Все изменения будут фиксироваться в виде коммитов с подробным описанием, что обеспечит прозрачность и структурированность процесса доработки.

Обратная связь пользователей станет важным источником информации для дальнейшего развития приложения. На её основе предполагается добавление новых возможностей, таких как дополнительные световые эффекты – динамическая радуга, плавное затухание и другие режимы, доступные для настройки через интерфейс.

Планируется добавить возможность сохранения пользовательских настроек яркости, которые будут автоматически применяться при запуске приложения. Поддержка дополнительных языков, таких как испанский и французский, значительно расширит аудиторию приложения. Особенно пользователи смогут высоко оценить возможность переключения языков интерфейса в один клик. Это сделает приложение доступным для международной аудитории.

Финальным этапом станет подготовка приложения к выпуску в магазины приложений, такие как Google Play и App Store. Этот процесс будет включать несколько важных шагов. Планируется провести анализ и оптимизацию размера APK/IPA файлов.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы было разработано мобильное приложение для управления микроконтроллером ESP-32. Основной задачей стало создание удобного и функционального инструмента, позволяющего управлять светодиодной лентой через Wi-Fi и USB. Были изучены современные технологии взаимодействия мобильных устройств с IoT-устройствами, проведён анализ существующих решений и выбраны подходящие технологии.

Для реализации проекта был выбран микроконтроллер ESP-32, обладающий поддержкой Wi-Fi и Bluetooth, низким энергопотреблением и высокой производительностью. В качестве инструмента для мобильной разработки использовался фреймворк Flutter, обеспечивающий кроссплатформенную совместимость и поддержку многоязычного интерфейса.

Взаимодействие между приложением и микроконтроллером реализовано по модели клиент-сервер с использованием протокола HTTP в формате JSON. Для обеспечения безопасности внедрена авторизация на основе токенов, исключающая несанкционированный доступ к устройству.

Мобильное приложение предоставляет функции управления светодиодной лентой: включение/выключение, изменение цвета и яркости, выбор световых эффектов. Реализована поддержка голосового управления и нескольких языковых версий, что расширяет целевую аудиторию приложения.

Интерфейс разрабатывался с учётом принципов простоты и удобства использования. Все элементы интерфейса интуитивно понятны, имеют оптимальные размеры и контрастность, что обеспечивает комфортную работу даже на устройствах с маленьким экраном. На основании пользовательского тестирования были внесены улучшения, повышающие качество взаимодействия с приложением.

Результаты тестирования подтвердили стабильную работу приложения. Время отклика составило менее 500 миллисекунд, соединение через Wi-Fi оказалось устойчивым. Выявлены отдельные недостатки, связанные с масштабируемостью системы и отзывами пользователей.

При сравнении с аналогами – LED Light Controller Remote+ и HappyLighting – разработанное приложение показало преимущество по ключевым параметрам: наличие голосового управления, гибкая настройка цвета и яркости, поддержка нескольких языков и полная кроссплатформенность.

По сравнению с популярными решениями, такими как LED Light Controller Remote+ и HappyLighting, оно предлагает более широкую поддержку языков, наличие голосового управления и гибкие настройки цвета и яркости. Также отмечается полная кроссплатформенность и высокая адаптивность интерфейса под разные устройства.

На выходе мы имеем полноценное мобильное приложение, предназначенное для управления светодиодной лентой через микроконтроллер ESP-32. Оно может быть использовано как в бытовых, так и в образовательных целях. Полученный результат демонстрирует актуальность разработки в области технологий Интернета вещей и открывает возможности для дальнейшего развития проекта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. БИНОМТЕХ [Электронный ресурс]. – Режим доступа: https://binomtech.com/services/mobile. – Дата доступа: 21.05.2025.

2. Всёсмарт [Электронный ресурс]. – Режим доступа: https://vsesmart.ru/blog/shlyuzy-umnogo-doma-kak-vystroit-edinuyu-sistemu/. – Дата доступа: 21.05.2025.

3. Джейд Картер IoT Интернет вещей [Текст] / Джейд Картер: 2024 – 180 с.

4. Мир микроконтроллеров [Электронный ресурс]. – Режим доступа: https://microkontroller.ru/esp32-projects/esp32-protiv-esp8266-kakoj-mikrokontroller-vybrat/. – Дата доступа: 21.05.2025.

5. Папуловская, Н.В. Основы интернета вещей / Н.В. Папуловская. – Екатеринбург: Уральский федеральный университет, 2022. – 104 с.

6. Проконтакт [Электронный ресурс]. – Режим доступа: https://procontact74.ru/blog/17-radiodetali-dlya-iot-kakie-komponenty-delayut-internet-veshchey-vozmozhnym/. – Дата доступа: 21.05.2025.

7. Русбейс [Электронный ресурс]. – Режим доступа: https://rb.ru/story/iot-irl/. – Дата доступа: 21.05.2025.

8. Софиот [Электронный ресурс]. – Режим доступа: https://sofiot.ru/blog/voprosy-posetiteley-sayta/standarty-interfeysa-umnogo-doma/. – Дата доступа: 21.05.2025.

9. Alotceriot [Электронный ресурс]. – Режим доступа: https://www.alotceriot.com/ru/zigbee-%D0%BF%D1%80%D0%BE%D1%82%D0%B8%D0%B2-wi-fi-%D0%B4%D0%B5%D0%BC%D0%B8%D1%81%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F-%D0%BF%D1%80%D0%BE%D1%82%D0%B8%D0%B2%D0%BE%D1%81%D1%82%D0%BE/. – Дата доступа: 21.05.2025.

10. Arduino docs [Электронный ресурс]. – Режим доступа: https://docs.arduino.cc/language-reference/?spm=a2ty_o01.29997173.0.0.5b29c921nDwjgq. – Дата доступа: 15.05.2025.

11. Artificer [Электронный ресурс]. – Режим доступа: https://artificer.com.ua/esp32-vs-esp8266-sravnenie-kontrollerov/. – Дата доступа: 21.05.2025.

12. Bailey, A. Kotlin Programming: The Big Nerd Ranch Guide, 2nd edition / A. Bailey, D. Greenhalgh, J. Skeen. – Atlanta: Big Nerd Ranch, LLC, 2021. – 444 с.

13. Biessek, A. Flutter for Begginers: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2 / A. Biessek. – Birmingham: Packt Publishing Ltd., 2019. – 468 с.

14. Dart [Электронный ресурс]. – Режим доступа: https://dart.dev/docs?spm=a2ty_o01.29997173.0.0.24aac921MzbqHO. – Дата доступа: 15.05.2025.

15. DigitalWeed [Электронный ресурс]. – Режим доступа: https://digitalweed.ru/%D0%BE%D1%82%D0%BB%D0%B8%D1%87%D0%B8%D0%B5-esp32-%D0%BE%D1%82-esp8266-%D0%B8-arduino/. – Дата доступа: 21.05.2025.

16. Flutter Docs [Электронный ресурс]. – Режим доступа: https://docs.flutter.dev/ui/accessibility-and-internationalization/internationalization?spm=a2ty_o01.29997173.0.0.24aac921MzbqHO. – Дата доступа: 15.05.2025.

17. iTech [Электронный ресурс]. – Режим доступа: https://itechinfo.ru/content/%D1%81%D1%84%D0%B5%D1%80%D1%8B-%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F-iot. – Дата доступа: 21.05.2025.

18. IXBT Live [Электронный ресурс]. – Режим доступа: https://www.ixbt.com/live/chome/wi-fi-bluetooth-z-wave-ili-zigbee-kakoy-protokol-umnogo-doma-vybrat.html. – Дата доступа: 21.05.2025.

19. KNX24.com [Электронный ресурс]. – Режим доступа: https://knx24.com/news/base/smart_home_protocols/?srsltid=AfmBOoqQkuay8vufW7u0pvvrVXYNhGaRFPNtNXChpmaY_B6lYX9xBXt3. – Дата доступа: 21.05.2025.

20. Kotlinlang [Электронный ресурс]. – Режим доступа: https://kotlinlang.org/docs/multiplatform.html?spm=a2ty_o01.29997173.0.0.5b29c921nDwjgq. – Дата доступа: 15.05.2025.

21. Mathias, A. Swift Programming: The Big Nerd Ranch Guide, 2nd edition / A. Mathias, D. Galagher, J. Skeen. – Atlanta: Big Nerd Ranch, LLC, 2016. – 676 с.

22. Openarchive [Электронный ресурс]. – Режим доступа: https://openarchive.nure.ua/server/api/core/bitstreams/c3016065-6764-47df-adb5-16544ba0b30e/content. – Дата доступа: 21.05.2025.

23. PROlum [Электронный ресурс]. – Режим доступа: https://prolum.com.ua/ru/6-luchshykh-protokolov-dlia-umnoho-doma-kak-sdelat-pravylnyi-vybor/?srsltid=AfmBOor7_BrdQTG6mUfrT9cxIap8Afwc8lLsoueybAedpC4DNJ2WWeeo. – Дата доступа: 21.05.2025.

24. Raspberry Pi [Электронный ресурс]. – Режим доступа: https://www.raspberrypi.com/documentation/. – Дата доступа: 15.05.2025.

25. Razumdom [Электронный ресурс]. – Режим доступа: https://razumdom.ru/lectures/MCU.pdf. – Дата доступа: 21.05.2025.

26. React Native [Электронный ресурс]. – Режим доступа: https://reactnative.dev/docs/getting-started?spm=a2ty_o01.29997173.0.0.5b29c921nDwjgq. – Дата доступа: 15.05.2025.

27. Rusmarta [Электронный ресурс]. – Режим доступа: https://rusmarta.ru/reviews/umnyy-dom-na-baze-zigbee/. – Дата доступа: 21.05.2025.

28. Softarc [Электронный ресурс]. – Режим доступа: https://softarc.online/zigbee-knx-wifi-networks/. – Дата доступа: 21.05.2025.

29. Sostav [Электронный ресурс]. – Режим доступа: https://www.sostav.ru/blogs/273358/59906. – Дата доступа: 21.05.2025.

30. STM Espressif [Электронный ресурс]. – Режим доступа: https://www.espressif.com/en/support/documents/technical-documents?spm=a2ty_o01.29997173.0.0.5b29c921nDwjgq. – Дата доступа: 15.05.2025.

ПРИЛОЖЕНИЕ А

ОБРАБОТКА ГОЛОСОВЫХ КОМАНД

```
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:rgb_tape/service/api_service.dart';
import 'package:rgb_tape/voice/voice_api.dart';
import 'package:rgb_tape/l10n/l10n.dart';

class VoiceCommandHandler {
  final ApiService apiService;
  final VoiceApi voiceApi;
  final void Function() onEnable;
  final void Function() onDisable;
  final void Function(Color) onColorChange;
  final void Function(double) onBrightnessChange;
  final void Function(int) onEffectChange;
  final void Function(int, Color) onPixelChange;
  final VoidCallback onLogout;

  bool isListening = false;

  VoiceCommandHandler({
    required this.apiService,
    required this.voiceApi,
    required this.onEnable,
    required this.onDisable,
    required this.onColorChange,
    required this.onBrightnessChange,
    required this.onEffectChange,
    required this.onPixelChange,
    required this.onLogout,
  });

  void toggleListening(BuildContext context) {
    if (isListening) {
      stopListening();
    } else {
      startListening(context);
```

```
  }
}

void startListening(BuildContext context) {
  isListening = true;
  onEnable();

  if (kDebugMode) print("Voice control started");

  voiceApi.startListening(
        (command) async {
      if (kDebugMode) print("Received voice command: $command");

      stopListening();

      await handleVoiceCommand(command.toLowerCase(), context);

      Future.delayed(const Duration(milliseconds: 500), () {
        if (isListening) startListening(context);
      });
    },
    localeId: Localizations.localeOf(context).languageCode,
  );

  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text(context.l10n.voiceControlEnabled)),
  );
}


void stopListening() {
  isListening = false;
  onDisable();
  voiceApi.stopListening();
  if (kDebugMode) {
    print("Voice control stopped");
  }
}
```

```
  Future<void> handleVoiceCommand(String command, BuildContext
context) async {
    command = command.toLowerCase();
    if (kDebugMode) {
      print("Handling command: $command");
    }

    if (command == context.l10n.voiceTurnOn) {
      if (kDebugMode) {
        print("Command recognized: Turn on");
      }
      apiService.togglePower(1);
      return;
    } else if (command == context.l10n.voiceTurnOff) {
      if (kDebugMode) {
        print("Command recognized: Turn off");
      }
      apiService.togglePower(0);
      return;
    }

    final colorMap = {
      context.l10n.voiceRed.toLowerCase(): Colors.red,
      context.l10n.voiceGreen.toLowerCase(): Colors.green,
      context.l10n.voiceBlue.toLowerCase(): Colors.blue,
      context.l10n.voiceYellow.toLowerCase(): Colors.yellow,
      context.l10n.voiceWhite.toLowerCase(): Colors.white,
    };

    final numberWords = {
      context.l10n.numFirst.toLowerCase(): 1,
      context.l10n.numSecond.toLowerCase(): 2,
      context.l10n.numThird.toLowerCase(): 3,
      context.l10n.numFourth.toLowerCase(): 4,
      context.l10n.numFifth.toLowerCase(): 5,
      context.l10n.numSixth.toLowerCase(): 6,
      context.l10n.numSeventh.toLowerCase(): 7,
      context.l10n.numEighth.toLowerCase(): 8,
      context.l10n.numNinth.toLowerCase(): 9,
```

```
      context.l10n.numTenth.toLowerCase(): 10,
      context.l10n.numEleventh.toLowerCase(): 11,
      context.l10n.numTwelfth.toLowerCase(): 12,
      context.l10n.numThirteenth.toLowerCase(): 13,
      context.l10n.numFourteenth.toLowerCase(): 14,
      context.l10n.numFifteenth.toLowerCase(): 15,
      context.l10n.numSixteenth.toLowerCase(): 16,
      context.l10n.numSeventeenth.toLowerCase(): 17,
      context.l10n.numEighteenth.toLowerCase(): 18,
      context.l10n.numNineteenth.toLowerCase(): 19,
      context.l10n.numTwentieth.toLowerCase(): 20,
      context.l10n.numTwentyFirst.toLowerCase(): 21,
      context.l10n.numTwentySecond.toLowerCase(): 22,
    };

    String normalizeCommand(String command) {
      return command.replaceAll('ё', 'е');
    }

    String normalizedCommand = normalizeCommand(command);

    if (kDebugMode) {
      print("Normalized command: $normalizedCommand");
    }

    RegExp pixelPattern = RegExp(
      r'(' + numberWords.keys.join(r'|') +
r')\s*(пиксель|pixel)\s*(' +

r'(красный|зелёный|синий|жёлтый|белый|red|green|blue|yellow|white)
' +
          r')',
    );

    if (pixelPattern.hasMatch(normalizedCommand)) {
      final pixelMatch =
pixelPattern.firstMatch(normalizedCommand);
      if (pixelMatch != null) {
        String pixelWord = pixelMatch.group(1)!.toLowerCase();
```

```
        int pixelNumber = numberWords[pixelWord]!;
        String color = pixelMatch.group(3)!.toLowerCase();

        if (colorMap.containsKey(color)) {
          if (kDebugMode) {
            print("Command recognized: Set pixel $pixelNumber to
${colorMap[color]}");
          }
          onPixelChange(pixelNumber, colorMap[color]!);
          return;
        } else {
          if (kDebugMode) {
            print("Unrecognized color: $color");
          }
        }
      }
    }

    if (kDebugMode) {
      print("Pixel command not matched in '$command'");
    }


    for (var entry in colorMap.entries) {
      if (command.contains(entry.key)) {
        if (kDebugMode) {
          print("Command recognized: Change color to
${entry.key}");
        }
        onColorChange(entry.value);
        return;
      }
    }

    final Map<String, int> effectWords = {
      context.l10n.numOne: 1,
      context.l10n.numTwo: 2,
      context.l10n.numThree: 3,
      context.l10n.numFour: 4,
```

```
      context.l10n.numFive: 5,
      context.l10n.numSix: 6,
      context.l10n.numSeven: 7,
      context.l10n.numEight: 8,
      context.l10n.numNine: 9,
      context.l10n.voiceDisableEffect: 0
    };

    for (var entry in effectWords.entries) {
      if (command.contains(entry.key)) {
        if (kDebugMode) {
          print("Command recognized: Apply effect
${entry.value}");
        }
        onEffectChange(entry.value);
        return;
      }
    }

    if (kDebugMode) {
      print("Effect command not matched in '$command'");
    }

    final Map<String, int> brightnessWords = {
      context.l10n.numOne: 1,
      context.l10n.numTwo: 2,
      context.l10n.numThree: 3,
      context.l10n.numFour: 4,
      context.l10n.numFive: 5,
      context.l10n.numSix: 6,
      context.l10n.numSeven: 7,
      context.l10n.numEight: 8,
      context.l10n.numNine: 9,
      context.l10n.numTen: 10,
      context.l10n.numEleven: 11,
      context.l10n.numTwelve: 12,
      context.l10n.numThirteen: 13,
      context.l10n.numFourteen: 14,
      context.l10n.numFifteen: 15,
```

```
context.l10n.numSixteen: 16,
context.l10n.numSeventeen: 17,
context.l10n.numEighteen: 18,
context.l10n.numNineteen: 19,
context.l10n.numTwenty: 20,
context.l10n.numTwentyOne: 21,
context.l10n.numTwentyTwo: 22,
context.l10n.numTwentyThree: 23,
context.l10n.numTwentyFour: 24,
context.l10n.numTwentyFive: 25,
context.l10n.numTwentySix: 26,
context.l10n.numTwentySeven: 27,
context.l10n.numTwentyEight: 28,
context.l10n.numTwentyNine: 29,
context.l10n.numThirty: 30,
context.l10n.numThirtyOne: 31,
context.l10n.numThirtyTwo: 32,
context.l10n.numThirtyThree: 33,
context.l10n.numThirtyFour: 34,
context.l10n.numThirtyFive: 35,
context.l10n.numThirtySix: 36,
context.l10n.numThirtySeven: 37,
context.l10n.numThirtyEight: 38,
context.l10n.numThirtyNine: 39,
context.l10n.numForty: 40,
context.l10n.numFortyOne: 41,
context.l10n.numFortyTwo: 42,
context.l10n.numFortyThree: 43,
context.l10n.numFortyFour: 44,
context.l10n.numFortyFive: 45,
context.l10n.numFortySix: 46,
context.l10n.numFortySeven: 47,
context.l10n.numFortyEight: 48,
context.l10n.numFortyNine: 49,
context.l10n.numFifty: 50,
context.l10n.numFiftyOne: 51,
context.l10n.numFiftyTwo: 52,
context.l10n.numFiftyThree: 53,
context.l10n.numFiftyFour: 54,
```

```
context.l10n.numFiftyFive: 55,
context.l10n.numFiftySix: 56,
context.l10n.numFiftySeven: 57,
context.l10n.numFiftyEight: 58,
context.l10n.numFiftyNine: 59,
context.l10n.numSixty: 60,
context.l10n.numSixtyOne: 61,
context.l10n.numSixtyTwo: 62,
context.l10n.numSixtyThree: 63,
context.l10n.numSixtyFour: 64,
context.l10n.numSixtyFive: 65,
context.l10n.numSixtySix: 66,
context.l10n.numSixtySeven: 67,
context.l10n.numSixtyEight: 68,
context.l10n.numSixtyNine: 69,
context.l10n.numSeventy: 70,
context.l10n.numSeventyOne: 71,
context.l10n.numSeventyTwo: 72,
context.l10n.numSeventyThree: 73,
context.l10n.numSeventyFour: 74,
context.l10n.numSeventyFive: 75,
context.l10n.numSeventySix: 76,
context.l10n.numSeventySeven: 77,
context.l10n.numSeventyEight: 78,
context.l10n.numSeventyNine: 79,
context.l10n.numEighty: 80,
context.l10n.numEightyOne: 81,
context.l10n.numEightyTwo: 82,
context.l10n.numEightyThree: 83,
context.l10n.numEightyFour: 84,
context.l10n.numEightyFive: 85,
context.l10n.numEightySix: 86,
context.l10n.numEightySeven: 87,
context.l10n.numEightyEight: 88,
context.l10n.numEightyNine: 89,
context.l10n.numNinety: 90,
context.l10n.numNinetyOne: 91,
context.l10n.numNinetyTwo: 92,
context.l10n.numNinetyThree: 93,
```

```
      context.l10n.numNinetyFour: 94,
      context.l10n.numNinetyFive: 95,
      context.l10n.numNinetySix: 96,
      context.l10n.numNinetySeven: 97,
      context.l10n.numNinetyEight: 98,
      context.l10n.numNinetyNine: 99,
      context.l10n.numOneHundred: 100
    };

    for (var entry in brightnessWords.entries) {
      if (command.contains(entry.key)) {
        if (kDebugMode) {
          print("Command recognized: Set brightness to
${entry.value}");
        }
        onBrightnessChange(entry.value.toDouble());
        return;
      }
    }

    RegExp numberPattern = RegExp(r'(\d+)');
    Match? match = numberPattern.firstMatch(command);
    if (match != null) {
      int number = int.parse(match.group(1)!);
      if (number >= 1 && number <= 100) {
        if (kDebugMode) {
          print("Command recognized: Set brightness to $number");
        }
        onBrightnessChange(number.toDouble());
        return;
      }
    }

    if (kDebugMode) {
      print("Brightness command not matched in '$command'");
    }

    void dispose() {
      stopListening();
```

```
      if (kDebugMode) print("VoiceCommandHandler disposed");
    }

    if (kDebugMode) {
      print("Unknown command: '$command'");
    }

    if (command == context.l10n.voiceExit) {
      if (kDebugMode) {
        print("Command recognized: Logout");
      }
      onLogout();
    } else {
      if (kDebugMode) {
        print("Unknown command: '$command'");
      }
    }
  }
}
```

ПРИЛОЖЕНИЕ Б

СТРАНИЦА УПРАВЛЕНИЯ СВЕТОДИОДНОЙ ЛЕНТОЙ

```
import 'package:flutter/material.dart';
import 'package:flutter_colorpicker/flutter_colorpicker.dart';
import 'package:rgb_tape/api_methods/auth_service.dart';
import 'package:rgb_tape/api_methods/brightness_api.dart';
import 'package:rgb_tape/api_methods/color_api.dart';
import 'package:rgb_tape/api_methods/effects_api.dart';
import 'package:rgb_tape/api_methods/pixel_api.dart';
import 'package:rgb_tape/api_methods/toggle_api.dart';
import 'package:rgb_tape/l10n/l10n.dart';
import 'package:rgb_tape/localization/language_switcher.dart';
import 'package:rgb_tape/service/api_service.dart';
import 'package:rgb_tape/voice/voice_api.dart';
import 'package:rgb_tape/voice/voice_command_handler.dart';

import '../api_methods/client_api.dart';

class MainControlScreen extends StatefulWidget {
  const MainControlScreen({super.key});

  @override
  State<MainControlScreen> createState() =>
_MainControlScreenState();
}

class _MainControlScreenState extends State<MainControlScreen> {
  late final ApiService apiService;
  late final VoiceApi voiceApi;
  late final VoiceCommandHandler voiceCommandHandler;
  bool isVoiceControlEnabled = false;
  bool isLoading = false;
  Color selectedColor = Colors.white;
  double brightness = 100;
  int selectedEffect = 0;
  final TextEditingController _pixelController =
TextEditingController();
```

```
  @override
  void initState() {
    super.initState();
    final client = ClientApi();

    apiService = ApiService(
      colorApi: ColorApi(apiClient: client),
      brightnessApi: BrightnessApi(apiClient: client),
      effectsApi: EffectsApi(apiClient: client),
      pixelApi: PixelApi(apiClient: client),
      toggleApi: ToggleApi(apiClient: client),
      loginApi: AuthService(apiClient: client),
    );

    voiceApi = VoiceApi();
    voiceCommandHandler = VoiceCommandHandler(
      apiService: apiService,
      voiceApi: voiceApi,
      onEnable: _enableVoiceControl,
      onDisable: _disableVoiceControl,
      onColorChange: _handleColorChange,
      onBrightnessChange: _handleBrightnessChange,
      onEffectChange: _handleEffectChange,
      onPixelChange: _handlePixelColorChange,
      onLogout: _logout,
    );

    voiceApi.init();
  }

  void _enableVoiceControl() {
    setState(() {
      isVoiceControlEnabled = true;
    });

    voiceApi.startListening((command) {

voiceCommandHandler.handleVoiceCommand(command.toLowerCase(),
context);
```

```
    });

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(context.l10n.voiceControlEnabled)),
    );
  }

  void _disableVoiceControl() {
    setState(() {
      isVoiceControlEnabled = false;
    });

    voiceApi.stopListening();
  }

  void _logout() {
    apiService.loginApi.logout();
    Navigator.pushReplacementNamed(context, '/login');
  }

  void _handleColorChange(Color color) {
    setState(() {
      selectedColor = color;
    });
    apiService.changeColor(color.red, color.green, color.blue);
  }

  void _handleBrightnessChange(double value) {
    setState(() {
      brightness = value;
    });
    apiService.changeBrightness(brightness.toInt());
  }

  void _handleEffectChange(int effect) {
    setState(() {
      selectedEffect = effect;
    });
    apiService.applyEffect(selectedEffect);
```

```
}

void _handlePixelColorChange(int pixelNumber, Color color) {
  setState(() {
  });
  apiService.pixelApi.changePixelColor(
    pixelNumber,
    color.red,
    color.green,
    color.blue,
  );
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(
        context.l10n.mainPage,
        textAlign: TextAlign.center,
      ),
      centerTitle: true,
      backgroundColor: Colors.purple[300],
      actions: const [
        LanguageSwitcher(),
      ],
    ),
    body: SingleChildScrollView(
      padding: const EdgeInsets.all(20),
      child: Column(
        children: [
          Container(
            padding: const EdgeInsets.all(16),
            decoration: BoxDecoration(
              color: Colors.white,
              borderRadius: BorderRadius.circular(10),
              boxShadow: const [
                BoxShadow(
                  color: Colors.black38,
```

```
                    blurRadius: 8,
                    offset: Offset(0, 4),
                  ),
                ],
              ),
              child: Column(
                children: [
                  SizedBox(
                    width: double.infinity,
                    child: ColorPicker(
                      pickerColor: selectedColor,
                      onColorChanged: _handleColorChange,
                      showLabel: true,
                      pickerAreaHeightPercent: 0.5,
                      enableAlpha: false,
                    ),
                  ),
                  const SizedBox(height: 10),
                  Text(
                    '${context.l10n.selectedColor}:
(${selectedColor.red}, ${selectedColor.green},
${selectedColor.blue})',
                    style: const TextStyle(fontSize: 16, color:
Colors.black, fontWeight: FontWeight.bold),
                  ),
                  const SizedBox(height: 10),
                  Slider(
                    value: brightness,
                    min: 0,
                    max: 100,
                    divisions: 100,
                    label: brightness.round().toString(),
                    onChanged: (value) {
                      _handleBrightnessChange(value);
                    },
                  ),
                  const SizedBox(height: 10),
                  DropdownButton<int>(
                    value: selectedEffect,
```

```
                    onChanged: (value) {
                      if (value != null) {
                        _handleEffectChange(value);
                      }
                    },
                    items: List.generate(10, (index) {
                      return DropdownMenuItem(
                        value: index,
                        child: Text(index == 0 ?
context.l10n.offEffect : '${context.l10n.effectChange} $index'),
                      );
                    }).toList(),
                  ),
                  const SizedBox(height: 10),
                  Row(
                    mainAxisAlignment:
MainAxisAlignment.spaceAround,
                    children: [
                      ElevatedButton(
                        onPressed: () =>
apiService.togglePower(1),
                        style: ElevatedButton.styleFrom(
                          foregroundColor: Colors.white,
                          backgroundColor: Colors.green,
                        ),
                        child: Text(context.l10n.toggleOn),
                      ),
                      ElevatedButton(
                        onPressed: () =>
apiService.togglePower(0),
                        style: ElevatedButton.styleFrom(
                          foregroundColor: Colors.white,
                          backgroundColor: Colors.red,
                        ),
                        child: Text(context.l10n.toggleOff),
                      ),
                    ],
                  ),
                  const SizedBox(height: 10),
```

```
                    Center(
                      child: SizedBox(
                        width: 50,
                        child: TextFormField(
                          controller: _pixelController,
                          keyboardType: TextInputType.number,
                          textAlign: TextAlign.center,
                          decoration: const InputDecoration(
                            border: OutlineInputBorder(),
                            contentPadding:
EdgeInsets.symmetric(vertical: 10),
                          ),
                          onFieldSubmitted: (value) async {
                            final pixelNumber = int.tryParse(value);
                            if (pixelNumber != null && pixelNumber
>= 1 && pixelNumber <= 22) {
                              await
apiService.pixelApi.changePixelColor(
                                pixelNumber,
                                selectedColor.red,
                                selectedColor.green,
                                selectedColor.blue,
                              );
                              _pixelController.clear();
                            } else {

ScaffoldMessenger.of(context).showSnackBar(
                                SnackBar(content:
Text(context.l10n.warningPixel)),
                              );
                            }
                          },
                        ),
                      ),
                    ),
                    const SizedBox(height: 10),
                    ElevatedButton(
                      onPressed: () {
                        if (voiceCommandHandler.isListening) {
```

```
                        voiceCommandHandler.stopListening();
                      } else {

voiceCommandHandler.startListening(context);
                      }
                    },
                    child: Text(voiceCommandHandler.isListening ?
context.l10n.voiceControlOff : context.l10n.voiceControl),
                  ),
                  const SizedBox(height: 10),
                  ElevatedButton(
                    onPressed: _logout,
                    style: ElevatedButton.styleFrom(
                      foregroundColor: Colors.white,
                      backgroundColor: Colors.blue,
                    ),
                    child: Text(context.l10n.logout),
                  ),
                ],
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```