

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программный комплекс мониторинга и управления
микроконтроллерной системой автополива**

Студент _____ Хмыренков Никита Александрович
(подпись, дата) (ФИО полностью)

Руководитель _____ Балашова И.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

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

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

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

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

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«26» декабря 2025 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Хмыренков Никита Александрович гр. 22ВП1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Программный комплекс мониторинга и управления
микроконтроллерной системой автополива

Тема утверждена приказом ПГУ № _____ от “___” _____ 20__

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Балашова И.Ю.

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

1) *Разработать Программный комплекс мониторинга и управления*
микроконтроллерной системой автополива.

2) *Среда выполнения приложения: операционная система Android 9 и выше*
Среда выполнения сервера: прошивка микроконтроллера NodeMcu lolin v3
Языковые средства разработки: C, UML, HTML, JavaScript, TypeScript
Инструментальные средства разработки: Visual Studio 2022, Visual Paradigm
Arduino IDE

Технологии разработки: RUP, ООП

3) *Основные функции мобильного приложения:*
а) просмотр параметров системы автополива с датчиков в реальном времени;
б) настройка параметров системы автополива;
в) настройка уведомлений об отклонении параметров системы от нормы;
г) просмотр графиков параметров по часам, дням;
д) отправка уведомлений о состоянии системы;
е) мониторинг параметров системы.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 96 листов, 28 рисунков, 17 таблиц, 31 использованный источник, 5 приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ARDUINO, ESP8266, REACT NATIVE, АВТОПОЛИВ, МОНИТОРИНГ, REST API, TYPESCRIPT, LITTLEFS, IOT, UML, RUP

Объектом разработки является программный комплекс мониторинга и управления микроконтроллерной системой автополива.

Цель работы – разработка программного комплекса, включающего прошивку микроконтроллера Arduino для сбора данных с датчиков и управления насосом полива, прошивку Wi-Fi модуля ESP8266, предоставляющую REST API и выполняющую журналирование показаний, а также кроссплатформенное мобильное приложение для мониторинга параметров системы и дистанционного управления режимом полива.

Разработка прошивки микроконтроллера Arduino производилась на языке программирования C в среде Arduino IDE.

Разработка серверной части производилась на языке C в среде Arduino IDE для модуля ESP8266.

Разработка мобильного приложения производилась на языке TypeScript с использованием фреймворка React Native и платформы Expo SDK 54 в среде Visual Studio Code. Приложение предназначено для использования под управлением ОС Android версии 9.0 и выше, а также iOS.

					ПГУ 09.03.04 – 08БР221.23 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Программный комплекс мониторинга и управления микроконтроллерной системой автополива Пояснительная записка	Лит.	Лист	Листов
Разраб.		Хмыренков Н.А.					4	96
Пров.		Балашова И.Ю.						
Н. контр.		Такташкин Д.В.				Группа 22ВП1		
Утверд.		Козлов А.Ю.						

Содержание

Введение	7
1 Анализ предметной области и требований к программному комплексу автополива	9
1.1 Анализ предметной области и постановка задачи автоматизации полива	9
1.2 Анализ программных продуктов для управления системами автополива	10
1.3 Анализ целевой аудитории системы автополива	12
1.4 Анализ функциональных требований к программному комплексу автополива	14
1.5 Анализ нефункциональных требований к программному комплексу автополива	20
1.6 Планирование разработки и оценка бюджета	21
2 Проектирование мобильного приложения	26
2.1 Выбор технологий и инструментальных средств	26
2.2 Проектирование архитектуры программного обеспечения	27
2.3 Проектирование протокола обмена данными	28
2.4 Проектирование конечных точек и форматов данных	29
2.5 Проектирование интерфейса мобильного приложения	31
2.6 Проектирование структуры хранения данных	35
3 Реализация программного обеспечения	37
3.1 Реализация прошивки основного микроконтроллера	37
3.2 Реализация прошивки модуля ESP8266	38
3.3 Реализация клиентской части приложения	40
4 Тестирование программного продукта	43
4.1 Функциональное тестирование черным ящиком	43
4.2 Функциональная диаграмма и таблица решений	44
4.3 Модульное тестирование	46
4.4 Метрики кода	47
Заключение	52

Список использованных источников.....	54
Приложение А (обязательное) Исходный код серверной части.....	57
Приложение Б (обязательное) Исходный код мобильного приложения	67
Приложение В (обязательное) Исходный код библиотеки DS3231 и контроллера Arduino.....	83
Приложение Г (рекомендуемое) Результаты тестирования серверной части.....	86
Приложение Д (рекомендуемое) Исходный код функций тестирования	92

Введение

В условиях стремительного развития цифровых технологий и повсеместного распространения интернета вещей автоматизация бытовых и сельскохозяйственных процессов становится все более доступной и востребованной. Применение микроконтроллерных систем позволяет решать задачи непрерывного мониторинга окружающей среды и управления исполнительными устройствами в режиме реального времени.

Одним из наиболее актуальных направлений домашней автоматизации является управление поливом растений. Ручной полив требует постоянного присутствия человека и не позволяет поддерживать оптимальный режим увлажнения почвы, что негативно сказывается на состоянии растений. Разработка программного комплекса на базе доступных микроконтроллерных платформ, таких как Arduino и ESP8266, обеспечивает непрерывный мониторинг параметров окружающей среды и автоматическое управление насосом полива без постоянного участия человека.

Мобильный телефон с выходом в интернет стал основным устройством взаимодействия пользователя с IoT-системами. По данным аналитических агентств, в России на 2025 год число мобильных подключений превысило 230 миллионов, а доля мобильного трафика продолжает неуклонно расти [1]. Именно мобильное приложение является наиболее удобным интерфейсом управления системой автополива: оно позволяет в любой момент просматривать показания датчиков, изменять параметры системы и получать уведомления об отклонениях, не находясь вблизи объекта.

Вместе с тем готовые коммерческие решения в данной области обладают рядом существенных недостатков: высокой стоимостью оборудования, зависимостью от облачной инфраструктуры производителя и отсутствием возможности локальной работы без доступа к сети интернет. Открытые платформы автоматизации, обеспечивают гибкость настройки, но, как правило,

требуют значительных технических знаний для развертывания и не предоставляют специализированного интерфейса управления поливом.

Изложенное выше определяет актуальность настоящей работы. Целью данной работы является создание программного комплекса мониторинга и управления микроконтроллерной системой автополива, включающего прошивку для микроконтроллера Arduino, прошивку для Wi-Fi модуля ESP8266 и кроссплатформенное мобильное приложение на React Native, ориентированного на преодоление ключевых ограничений рассмотренных аналогов.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области и существующих программных решений для управления системами автополива;
- сформулировать функциональные и нефункциональные требования к разрабатываемому программному комплексу;
- спроектировать архитектуру программного комплекса и протокол обмена данными между его компонентами;
- разработать прошивку микроконтроллера Arduino, обеспечивающую сбор данных с датчиков и управление насосом полива;
- разработать прошивку Wi-Fi модуля ESP8266, реализующую REST API и журналирование показаний датчиков в файловой системе LittleFS;
- разработать кроссплатформенное мобильное приложение на React Native для мониторинга параметров системы и дистанционного управления поливом;
- провести тестирование программного комплекса методом черного ящика, выполнить модульное тестирование и оценить метрики качества кода.

1 Анализ предметной области и требований к программному комплексу автополива

1.1 Анализ предметной области и постановка задачи автоматизации полива

В современных условиях автоматизация бытовых и сельскохозяйственных процессов приобретает все большую актуальность. Одним из направлений такой автоматизации является управление поливом растений. Использование микроконтроллерных систем позволяет перейти от ручного режима к автоматическому: датчики непрерывно отслеживают параметры окружающей среды, а исполнительные устройства реагируют на отклонения без участия человека.

Объектом разработки является программный комплекс мониторинга и управления микроконтроллерной системой автополива. Система предназначена для автоматического поддержания заданного уровня влажности почвы, мониторинга температуры и влажности воздуха, а также дистанционного управления параметрами полива через мобильное приложение.

Разрабатываемый программный комплекс должен обеспечивать выполнение следующих основных функций:

- непрерывное считывание показаний датчика влажности почвы и датчика температуры и влажности воздуха;
- автоматическое включение и отключение насоса полива в зависимости от текущего уровня влажности почвы и заданного порогового значения;
- предоставление REST API для получения текущих показаний датчиков и управления системой;
- журналирование показаний датчиков в файловую систему с привязкой ко времени;
- отображение текущих показаний и истории измерений в мобильном приложении;

– аутентификацию пользователя при выполнении управляющих команд.

Таким образом, разрабатываемый программный комплекс охватывает три уровня: уровень микроконтроллера (сбор данных и управление насосом), уровень Wi-Fi модуля (сетевое взаимодействие, хранение истории, веб-интерфейс) и уровень мобильного приложения (визуализация данных и управление).

1.2 Анализ программных продуктов для управления системами автополива

В настоящее время существует ряд готовых решений для автоматизации полива растений. Рассмотрим наиболее известные из них и проведем их сравнительный анализ.

Gardena Smart System – коммерческое решение для автоматизации полива, включающее датчики влажности почвы, контроллеры клапанов и мобильное приложение [2]. Система управляется через облачный сервер: данные с датчиков передаются на серверы производителя, после чего становятся доступны в приложении.

К достоинствам системы относятся готовность к использованию «из коробки», широкий ассортимент совместимого оборудования и удобное мобильное приложение. К недостаткам – высокая стоимость оборудования, зависимость от облачной инфраструктуры производителя, отсутствие возможности локальной работы без доступа к интернету, а также закрытость программного обеспечения, исключающая возможность доработки под конкретные нужды.

Home Assistant – программное обеспечение домашней автоматизации, поддерживающая интеграцию с микроконтроллерами через фреймворк ESPHome [3]. Позволяет строить локальные системы мониторинга и управления на базе ESP8266/ESP32.

К достоинствам относятся открытый исходный код, возможность полностью локальной работы, гибкость настройки и большое сообщество. К недостаткам – высокий порог вхождения, необходимость развертывания отдельного сервера, избыточная сложность для простых задач автополива, а также отсутствие специализированного мобильного приложения для управления поливом.

Blynk – платформа для создания мобильных приложений управления IoT-устройствами [4]. Предоставляет готовые виджеты для отображения данных с датчиков и управления исполнительными устройствами через облачный брокер.

К достоинствам относятся простота создания интерфейса, поддержка Arduino и ESP8266, наличие мобильного приложения для iOS и Android. К недостаткам – зависимость от облачного сервера Blynk, ограничения бесплатного тарифа, невозможность работы в локальной сети без подписки, а также отсутствие встроенного журналирования данных на устройстве.

По результатам анализа была составлена сравнительная таблица рассмотренных решений (таблица 1).

Таблица 1 – Сравнение аналогов

Критерий	Gardena Smart System	Home Assistant + ESPHome	Blynk	Разрабатываемый комплекс
Локальная работа без интернета	Нет	Да	Нет	Да
Открытый исходный код	Нет	Да	Нет	Да
Специализированное мобильное приложение	Да	Нет	Да	Да
Журналирование на устройстве	Нет	Да	Нет	Да
Простота развертывания	Высокая	Низкая	Средняя	Высокая
Стоимость	Высокая	Низкая	Средняя	Низкая

Таким образом, в результате анализа аналогов было принято решение о разработке собственного программного комплекса, лишенного выявленных

недостатков: зависимости от облачной инфраструктуры, высокой стоимости и избыточной сложности развертывания.

1.3 Анализ целевой аудитории системы автополива

Целевой аудиторией разрабатываемого программного комплекса являются владельцы домашних растений, дачных участков и небольших теплиц, заинтересованные в автоматизации полива без привлечения дорогостоящих коммерческих решений.

Согласно данным исследований рынка умного дома, интерес к IoT-устройствам для автоматизации быта в России стабильно растет. По данным аналитического агентства J'son & Partners Consulting, объем российского рынка умного дома ежегодно увеличивается на 15–20% [5]. При этом значительная часть пользователей предпочитает самостоятельно собирать устройства на базе доступных микроконтроллерных платформ, таких как Arduino и ESP8266.

Поскольку мобильное приложение является основным интерфейсом взаимодействия с системой, важно учитывать распределение мобильных операционных систем среди потенциальных пользователей.

По данным Statcounter, мировой рынок мобильных операционных систем распределяется следующим образом [6]:

- Android – 70,36%;
- iOS – 29,25%;
- прочие (Samsung, Unknown, KaiOS, Linux и др.) – менее 0,4%.

Таким образом, большинство пользователей используют операционную систему Android, вторым по численности является iOS. Совокупно эти две платформы охватывают более 99% рынка мобильных устройств.

На рисунке 1 представлены доли мобильных операционных систем на российском рынке [7].

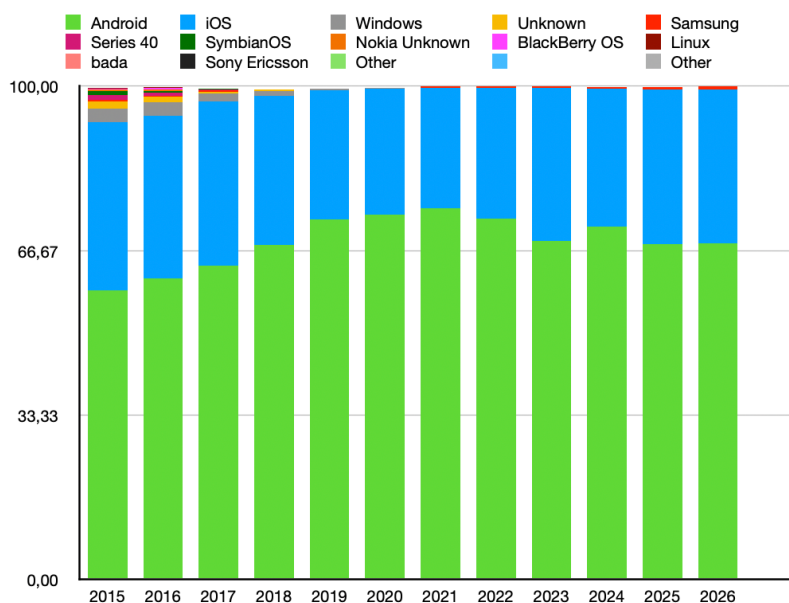


Рисунок 1 – Доли мобильных операционных систем на российском рынке

Несмотря на постепенное снижение доли Android с 2021 года, данная операционная система занимает лидирующее положение. Также в России развиваются отечественные мобильные операционные системы: «Аврора», «Astra Linux Mobile» и «Ред ОС М», однако их совокупная доля остается незначительной, и они ориентированы преимущественно на корпоративный сектор.

Таким образом, подавляющее большинство потенциальных пользователей системы автополива работают на Android или iOS. Отечественные платформы и прочие системы занимают пренебрежимо малую долю и не являются приоритетными для разработки. Следовательно, для охвата максимальной доли целевой аудитории разрабатываемое мобильное приложение должно поддерживать обе ведущие платформы, что обуславливает выбор кроссплатформенного подхода к разработке.

Пользователь системы должен иметь доступ к текущим показаниям датчиков (температура воздуха, влажность воздуха, влажность почвы), просмотру истории измерений в виде графиков, а также возможность устанавливать пороговое значение влажности почвы, при достижении которого система автоматически активирует насос полива.

Мобильное приложение разрабатывается на платформе React Native (Expo), что обеспечивает его работу как на устройствах под управлением Android, так и iOS, охватывая тем самым наибольшую долю рынка мобильных операционных систем.

1.4 Анализ функциональных требований к программному комплексу автополива

На основе анализа предметной области были определены следующие функциональные требования к разрабатываемому программному комплексу:

- просмотр текущих показаний влажности почвы, влажности воздуха и температуры воздуха в режиме реального времени с автоматическим обновлением данных и отображением единиц измерения для каждого параметра;
- автоматический полив при достижении заданного порогового значения влажности почвы;
- установка порогового значения влажности почвы, при котором активируется насос полива, с возможностью сохранения настроек на устройстве и их применения без перезапуска системы;
- просмотр истории изменения параметров в виде графиков с выбором даты и возможностью навигации по временным интервалам для анализа динамики показателей за выбранный период;
- авторизация в системе для получения доступа к управляющим функциям с защитой от несанкционированного изменения параметров работы системы автополива;
- настройка адреса устройства для подключения мобильного приложения к системе с возможностью сохранения введённых параметров подключения между сеансами использования приложения.

На основе перечисленных требований была построена диаграмма вариантов использования, представленная на рисунке 2.



Рисунок 2 – Диаграмма вариантов использования

На основе диаграммы вариантов использования составим спецификации для основных вариантов использования.

На рисунке 3 представлена спецификация «Просмотр текущих показаний»

Параметр	Значение
Наименование	Просмотр текущих показаний
ID	UC01
Краткое описание	Пользователь запрашивает актуальные данные с датчиков системы автополива, система отображает текущие показания
Действующие лица	Пользователь
Предусловия	Пользователь авторизован в системе; микроконтроллерное устройство подключено и передает данные
Основной поток	1. Пользователь открывает раздел текущих показаний 2. Система обращается к устройству за актуальными данными 3. Система отображает показания датчиков в реальном времени
Постусловия	На экране отображены актуальные значения параметров системы автополива

Рисунок 3 – Спецификация варианта использования «Просмотр текущих показаний»

На рисунке 4 представлена спецификация «Авторизация в системе»

Параметр	Значение
Наименование	Авторизация в системе
ID	UC02
Краткое описание	Пользователь вводит учетные данные, система проверяет их и предоставляет доступ к функциям комплекса
Действующие лица	Пользователь
Предусловия	Пользователь не авторизован; учетная запись существует в системе
Основной поток	1. Пользователь открывает форму входа 2. Ввод логина и пароля 3. Система проверяет корректность учетных данных 4. Система создает сессию и перенаправляет пользователя на главную страницу
Альтернативный поток	3а. Учетные данные не найдены – система выводит сообщение об ошибке, возврат к шагу 2
Постусловия	Пользователь авторизован и имеет доступ к функциям системы

Рисунок 4 – Спецификация варианта использования «Авторизация в системе»

На рисунке 5 представлена спецификация «Установка порогового значения влажности почвы»

Параметр	Значение
Наименование	Установка порогового значения влажности почвы
ID	UC03
Краткое описание	Администратор задает пороговое значение влажности, при достижении которого система автоматически активирует или деактивирует полив
Действующие лица	Администратор
Предусловия	Администратор авторизован в системе; устройство подключено
Основной поток	1. Администратор открывает раздел настройки параметров 2. Выбирает параметр «Пороговое значение влажности» 3. Вводит новое числовое значение 4. Система валидирует введенное значение 5. Система сохраняет значение и передает его на микроконтроллер 6. Система подтверждает успешное сохранение
Альтернативный поток	4а. Введено недопустимое значение – система выводит сообщение об ошибке, возврат к шагу 3
Постусловия	Новое пороговое значение влажности сохранено и применено на устройстве

Рисунок 5 – Спецификация варианта использования «Установка порогового значения влажности почвы»

На рисунке 6 представлена спецификация «Просмотр истории параметров.

Параметр	Значение
Наименование	Просмотр истории параметров
ID	UC04
Краткое описание	Пользователь просматривает исторические данные показаний датчиков системы автополива за выбранный период
Действующие лица	Пользователь
Предусловия	Пользователь авторизован; в базе данных имеются ранее сохраненные показания
Основной поток	1. Пользователь открывает раздел истории параметров 2. Выбирает временной диапазон для отображения 3. Система выбирает данные из хранилища за указанный период 4. Система отображает историю параметров в виде таблицы или графика
Альтернативный поток	3а. Данных за выбранный период нет – система выводит соответствующее уведомление
Постусловия	На экране отображена история показаний датчиков за выбранный период

Рисунок 6 – Спецификация варианта использования «Просмотр истории параметров»

На рисунке 7 представлена спецификация «Настройка адреса устройства»

Параметр	Значение
Наименование	Настройка адреса устройства
ID	UC05
Краткое описание	Пользователь указывает сетевой адрес (IP/порт) микроконтроллерного устройства для установления соединения с системой мониторинга
Действующие лица	Пользователь
Предусловия	Пользователь авторизован в системе
Основной поток	1. Пользователь открывает раздел настройки подключения 2. Вводит адрес устройства 3. Система проверяет формат введенных данных 4. Система сохраняет адрес и пытается установить соединение с устройством 5. Система подтверждает успешное подключение
Альтернативный поток	4а. Устройство недоступно по указанному адресу – система выводит сообщение об ошибке соединения, возврат к шагу 2
Постусловия	Адрес устройства сохранен; соединение с микроконтроллером установлено

Рисунок 7 – Спецификация варианта использования «Настройка адреса устройства»

Для вариантов использования «Установка порога полива» и «Авторизация в системе» составлены диаграммы деятельности. Диаграмма деятельности варианта использования «Установка порога полива» представлена на рисунке 8.

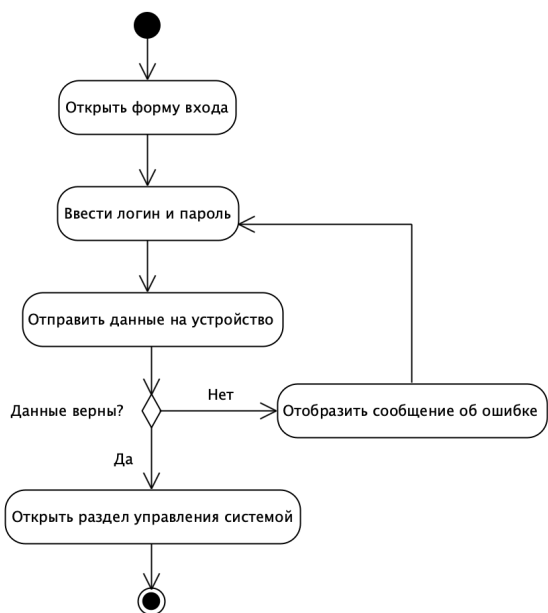


Рисунок 8 – Диаграмма деятельности варианта использования «Установка порога полива»

Диаграмма деятельности варианта использования «Авторизация в системе» представлена на рисунке 9.

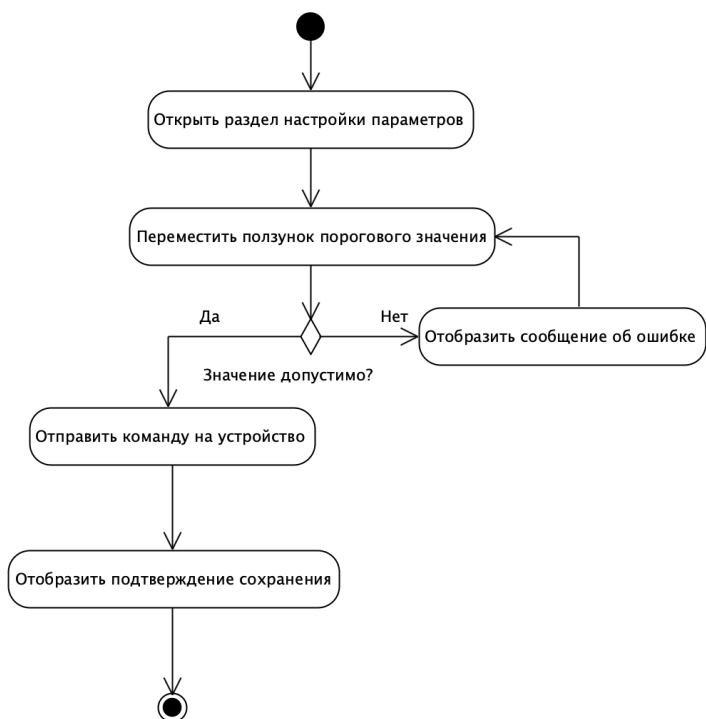


Рисунок 9 – Диаграмма деятельности варианта использования «Авторизация в системе»

Также для вариантов использования «Просмотр текущих параметров» и «Просмотр истории параметров» составлены диаграммы последовательности. Диаграмма последовательности варианта использования «Просмотр текущих параметров» представлена на рисунке 10.

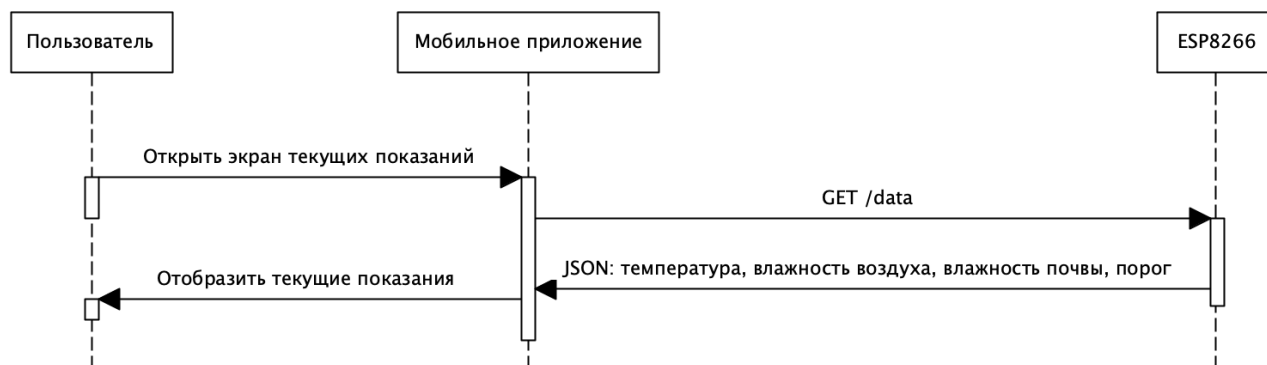


Рисунок 10 – Диаграмма последовательности варианта использования «Просмотр текущих параметров»

Диаграмма последовательности варианта использования «Просмотр истории параметров» представлена на рисунке 11.

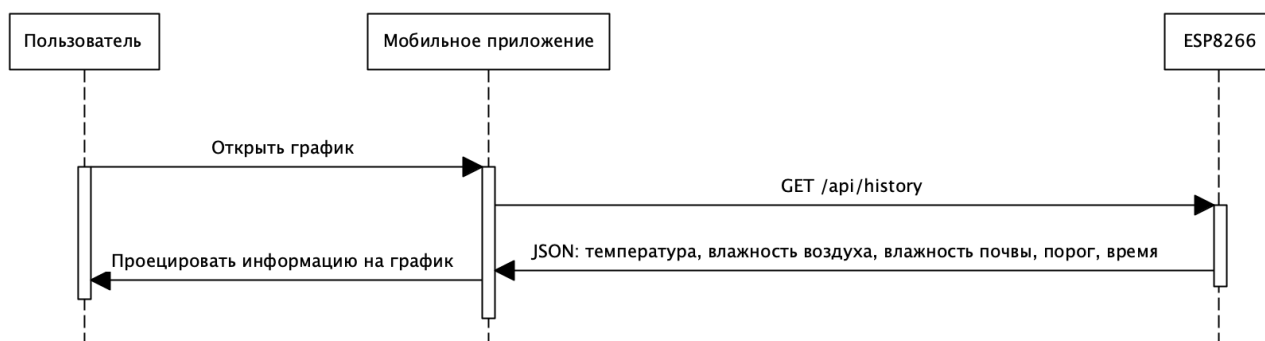


Рисунок 11 – Диаграмма последовательности варианта использования «Просмотр истории параметров»

На текущем этапе были определены ключевые функции разрабатываемого мобильного приложения. В дополнение к этому были построены диаграммы вариантов использования, деятельности и последовательности, а также подготовлены спецификации для отдельных вариантов использования.

1.5 Анализ нефункциональных требований к программному комплексу автополива

На данном этапе были определены нефункциональные требования к разрабатываемому программному комплексу. В качестве методологии разработки используется технология RUP. В качестве технологии разработки используется объектно-ориентированное программирование (ООП) [8].

Прошивка основного микроконтроллера будет разработана на языке программирования C++ в среде Arduino IDE для платформы Arduino Uno/Nano. Опрос датчиков и передача данных должны выполняться с периодом не более 1000 мс. Пороговое значение влажности почвы должно сохраняться в энергонезависимой памяти и восстанавливаться после перезапуска устройства [9].

Прошивка Wi-Fi модуля будет разработана на языке программирования C++ в среде Arduino IDE для платформы NodeMCU v3 (ESP8266). Модуль должен обеспечивать подключение к локальной сети Wi-Fi, синхронизацию времени по протоколу NTP и предоставление REST API. Время отклика REST API не должно превышать 500 мс при нормальных условиях работы сети. Журналирование показаний датчиков должно выполняться с периодом не более 30 минут [10].

Мобильное приложение будет разработано на языке программирования TypeScript с использованием фреймворка React Native в среде Visual Studio Code [11]. Оно должно представлять собой кроссплатформенное мобильное решение с удобным и интуитивно понятным пользовательским интерфейсом. Мобильное приложение должно корректно функционировать на устройствах под управлением операционной системы Android версии 10.0 и выше, а также iOS версии 14.0 и выше. Обновление показаний датчиков в приложении должно выполняться с периодом не более 2000 мс.

Управляющие команды должны быть доступны только авторизованным пользователям. Просмотр текущих показаний датчиков должен быть доступен без авторизации.

1.6 Планирование разработки и оценка бюджета

Эффективная реализация любого проекта в сфере информационных технологий невозможна без тщательного планирования. Оно служит основой для организации всех этапов работы таким образом, чтобы достичь поставленных целей в оптимальные сроки, с минимальными затратами и на должном уровне качества. Грамотное планирование способствует снижению рисков, повышению управляемости проекта и позволяет гибко реагировать на изменения, возникающие в ходе его реализации.

Задачи, представляющие собой этапы работы над проектом, представлены в таблице 2.

Таблица 2 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	–	–
2	Анализ требований программного комплекса автополива	–	–
3	Анализ предметной области и постановка задачи	2 дня	1
4	Анализ программных продуктов аналогичного назначения	2 дня	1
5	Анализ функциональных требований	3 дня	3; 4
6	Анализ нефункциональных требований	3 дня	5
7	Анализ целевой аудитории	1 день	6
8	Анализ требований завершен	–	7
9	Выбор технологий разработки программного комплекса	–	–
10	Анализ технологий разработки встроенного ПО	3 дня	8
11	Выбор технологий разработки ПО	2 дня	10
12	Выбор технологий завершен	–	11
13	Проектирование программных средств программного комплекса	–	–
14	Проектирование структуры хранения данных	2 дня	8; 12
15	Проектирование архитектуры прошивки микроконтроллера	3 дня	14

Продолжение таблицы 2

№	Название задачи	Длительность	Предшественники
16	Проектирование архитектуры Wi-Fi модуля	6 дня	14
17	Проектирование архитектуры мобильного приложения	5 дней	12
18	Проектирование завершено	–	15; 16; 17
19	Реализация программных средств программного комплекса	–	–
20	Реализация прошивки микроконтроллера	5 дней	15
21	Реализация прошивки Wi-Fi модуля	7 дней	16
22	Реализация мобильного приложения	14 дней	17
23	Реализация завершена	–	20; 21; 22
24	Тестирование программного комплекса	–	–
25	Функциональное тестирование	3 дня	23
26	Интеграционное тестирование	2 дня	23
27	Тестирование завершено	–	25; 26
28	Составление и оформление документации	7 дней	27
29	Конец проекта	–	28

Таким образом, для реализации проекта необходимо выполнить 29 задач.

В таблице 3 представлено описание затрачиваемых ресурсов.

Таблица 3 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист 1	Трудовой	600 Р/ч	800 Р/ч
Программист 2	Трудовой	600 Р/ч	800 Р/ч
Аналитик	Трудовой	500 Р/ч	600 Р/ч
Тестировщик	Трудовой	300 Р/ч	350 Р/ч
Интернет	Затраты	2 500 Р/месяц	
Компьютер	Материальный	500 Р	
Микроконтроллерный набор	Материальный	1 500 Р	
Бумага	Материальный	500 Р	

Таким образом, в общей сложности потребуется 8 видов ресурсов.

Для проверки корректности расчетов и вычисления бюджета, необходимого для создания программного комплекса, использована система управления проектами Microsoft Project, которая позволяет автоматически производить все вычисления на основе данных пользователя. В качестве даты начала проекта принята дата 07.04.2026 (рисунок 12).

	Название задачи ▼	Длительность ▼	Предшественник ▼	Начало ▼	Окончание ▼
1	Начало проекта	0 дней		Вт 07.04.26	Вт 07.04.26
2	▲ Анализ требований программного комплекса автополива	9 дней		Вт 07.04.26	Ср 15.04.26
3	Анализ предметной области и постановка задачи	2 дней	1	Вт 07.04.26	Ср 08.04.26
4	Анализ программных продуктов аналогичного назначения	2 дней	1	Вт 07.04.26	Ср 08.04.26
5	Анализ функциональных требований	3 дней	4;3	Чт 09.04.26	Сб 11.04.26
6	Анализ нефункциональных требований	3 дней	5	Вс 12.04.26	Вт 14.04.26
7	Анализ целевой аудитории	1 день	6	Ср 15.04.26	Ср 15.04.26
8	Анализ требований завершен	0 дней	7	Ср 15.04.26	Ср 15.04.26
9	▲ Выбор технологий разработки программного комплекса	5 дней		Чт 16.04.26	Пн 20.04.26
10	Анализ технологий разработки программного комплекса	3 дней	8	Чт 16.04.26	Сб 18.04.26
11	Выбор технологии разработки ПО	2 дней	10	Вс 19.04.26	Пн 20.04.26
12	Выбор технологий завершен	0 дней	11	Пн 20.04.26	Пн 20.04.26
13	▲ Проектирование программных средств программного комплекса	10 дней		Пн 20.04.26	Ср 29.04.26
14	Проектирование структуры хранения данных	2 дней	8;12	Пн 20.04.26	Ср 22.04.26
15	Проектирование архитектуры прошивки микроконтроллера	3 дней	14	Ср 22.04.26	Сб 25.04.26
16	Проектирование архитектуры Wi-Fi модуля	6 дней	14	Чт 23.04.26	Вт 28.04.26
17	Проектирование архитектуры мобильного приложения	5 дней	12	Вт 21.04.26	Сб 25.04.26
18	Проектирование приложения завершенно	0 дней	16;15;17	Вт 28.04.26	Вт 28.04.26
19	▲ Реализация программных средств программного комплекса	16 дней		Сб 25.04.26	Вс 10.05.26
20	Реализация прошивки микроконтроллера	5 дней	15	Вс 26.04.26	Чт 30.04.26
21	Реализация прошивки Wi-Fi модуля	7 дней	16	Ср 29.04.26	Вт 05.05.26
22	Реализация мобильного приложения	14 дней	17	Вс 26.04.26	Сб 09.05.26
23	Реализация завершенна	0 дней	21;22;20	Сб 09.05.26	Сб 09.05.26
24	▲ Тестирование программного комплекса	3 дней		Вс 10.05.26	Вт 12.05.26
25	Функциональное тестирование	3 дней	23	Вс 10.05.26	Вт 12.05.26
26	Интеграционное тестирование	2 дней	23	Вс 10.05.26	Пн 11.05.26
27	Тестирование завершенно	0 дней	25;26	Вт 12.05.26	Вт 12.05.26
28	Составление и оформление документации	7 дней	27	Ср 13.05.26	Вт 19.05.26
29	Окончание проекта	0 дней	28	Вт 19.05.26	Вт 19.05.26

Рисунок 12 – Задачи проекта в среде Microsoft Project

Диаграмма Ганта, отображающая распределение задач по срокам выполнения, представлена на рисунке 13.

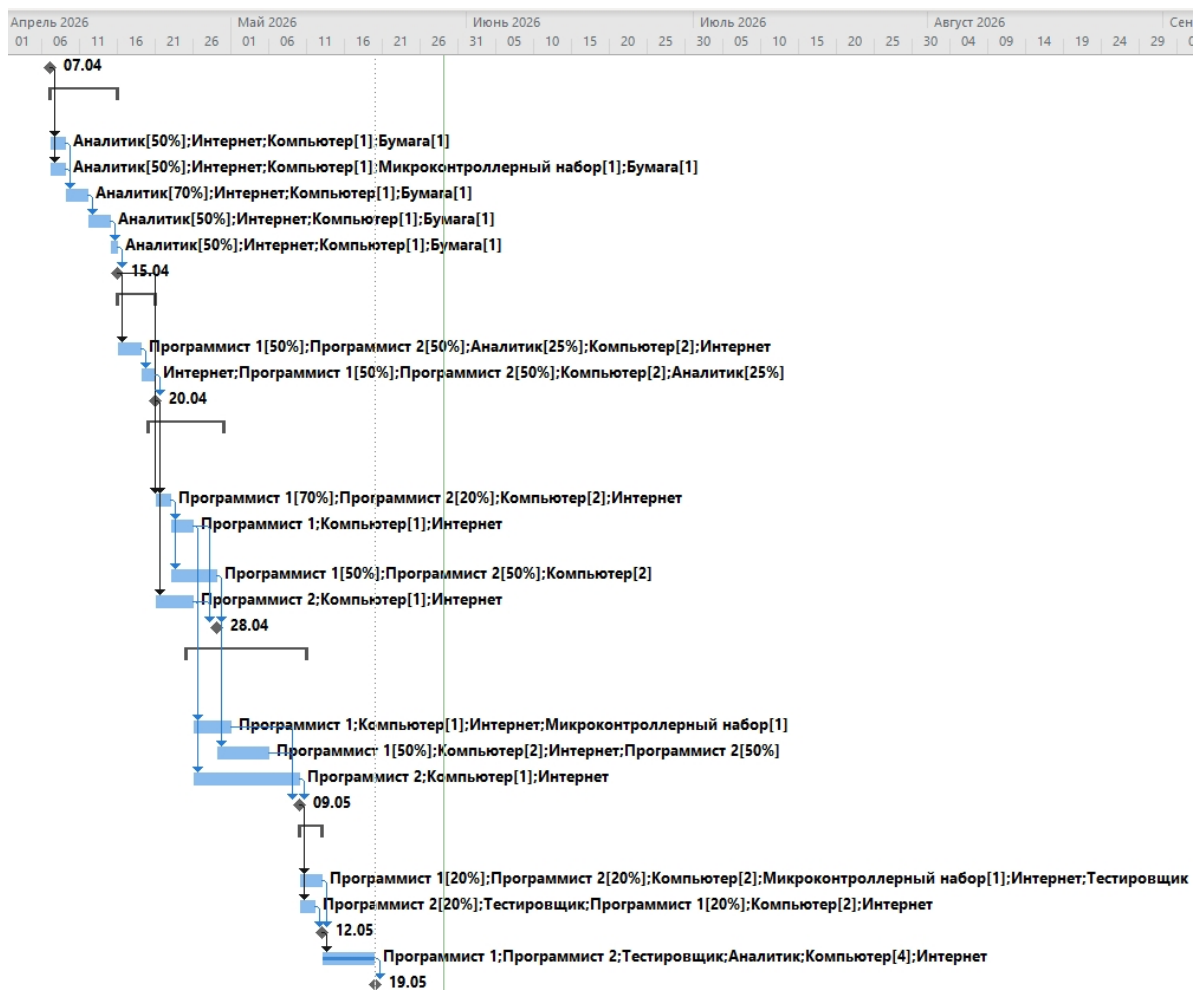


Рисунок 13 – Диаграмма Ганта

Обзорный отчет проекта, автоматически сформированный средствами Microsoft Project, представлен на рисунке 14.

Статистика проекта для 'диплом нов.mpr'			
	Начало	Окончание	
Текущее	Пн 01.06.26	Ср 29.07.26	
Базовое	НД	НД	
Фактическое	НД	НД	
Отклонение	0д	0д	
	Длительность	Трудозатраты	Затраты
Текущие	43д	697,2ч	398 540,00 Р
Базовые	0д	0ч	0,00 Р
Фактические	0д	0ч	0,00 Р
Оставшиеся	43д	697,2ч	398 540,00 Р
Процент завершения			
Длительность: 0%		Трудозатраты: 0%	
			Заккрыть

Рисунок 14 – Обзорный отчет проекта

На основе произведенных оценок стоимости и сроков проекта можно сделать вывод о его экономической целесообразности. Общая длительность проекта составляет 43 рабочих дня, что совпадает со значением, вычисленным вручную, а предполагаемые затраты – 398 549 руб.

Структура затрат проекта включает расходы на оплату труда разработчика, приобретение аппаратных компонентов, а также амортизацию программного обеспечения и оборудования рабочего места.

Критический путь проекта проходит через этапы проектирования архитектуры, разработки прошивки микроконтроллера и реализации мобильного приложения, что определяет минимально возможную длительность проекта. Запас времени по некритическим задачам (написание документации, подготовка тестовых сценариев) составляет от 2 до 5 рабочих дней, что обеспечивает устойчивость плана к незначительным отклонениям.

Анализ рисков проекта выявил следующие основные угрозы: несовместимость аппаратных компонентов, нестабильность Wi-Fi соединения в условиях эксплуатации и превышение сроков разработки мобильного приложения. Для минимизации рисков предусмотрено предварительное макетирование аппаратной части на этапе проектирования, использование механизма автоматического переподключения в прошивке ESP8266, а также применение кроссплатформенного фреймворка React Native, сокращающего трудозатраты на поддержку нескольких операционных систем.

Таким образом, проведенное планирование подтверждает реализуемость проекта в установленные сроки при имеющихся ресурсах.

2 Проектирование мобильного приложения

2.1 Выбор технологий и инструментальных средств

На основе анализа требований, проведенного в первом разделе, были определены технологии и инструментальные средства для реализации каждого компонента программного комплекса.

В качестве методологии разработки используется технология RUP (Rational Unified Process), обеспечивающая итеративный подход к проектированию и реализации программного обеспечения. Для моделирования архитектуры системы и построения UML-диаграмм используется среда Visual Paradigm Community Edition [12]. Планирование проекта выполняется в Microsoft Project.

Для разработки прошивки основного микроконтроллера выбрана платформа Arduino Uno на базе микроконтроллера ATmega328P. Программирование осуществляется на языке C++ в среде Arduino IDE. Выбор данной платформы обусловлен ее широкой распространенностью, наличием большого количества совместимых датчиков и исполнительных устройств, а также низким порогом вхождения. Для считывания температуры и влажности воздуха используется датчик DHT11 [13], для измерения влажности почвы – аналоговый емкостный датчик, подключенный к входу АЦП.

Для реализации сетевого уровня системы выбран модуль NodeMCU v3 на базе микроконтроллера ESP8266. Данный модуль обладает встроенным Wi-Fi интерфейсом, достаточным объемом оперативной и флеш-памяти, а также поддерживает файловую систему LittleFS для хранения журналов [14]. Программирование модуля осуществляется на языке C++ в среде Arduino IDE с использованием библиотек ESP8266WiFi, ESP8266WebServer и ArduinoJson.

Для разработки мобильного приложения выбран фреймворк React Native с использованием платформы Expo (SDK 54) [15]. Язык программирования – TypeScript. Данный выбор обусловлен возможностью создания

кроссплатформенного приложения (Android и iOS) из единой кодовой базы, наличием развитой экосистемы библиотек и инструментов, а также высокой скоростью разработки. Среда разработки – Visual Studio Code.

2.2 Проектирование архитектуры программного обеспечения

Архитектура разрабатываемого программного комплекса представляет собой трёхуровневую систему, в которой каждый уровень выполняет строго определённые функции и взаимодействует с соседними уровнями по заданным протоколам. Такое разделение обеспечивает независимость компонентов, упрощает сопровождение системы и позволяет масштабировать или заменять отдельные уровни без изменения остальных.

Первый уровень – уровень сбора данных и управления. На данном уровне функционирует основной микроконтроллер Arduino, который выполняет циклический опрос датчиков влажности почвы, влажности воздуха и температуры, принимает решение о включении или отключении насоса полива на основе сравнения текущей влажности почвы с пороговым значением. Логика управления реализована непосредственно на микроконтроллере, что обеспечивает автономную работу системы полива даже при временной недоступности вышестоящих уровней.

Второй уровень – уровень сетевого взаимодействия и хранения данных. На данном уровне функционирует модуль ESP8266, который принимает данные от Arduino по UART, предоставляет REST API для внешних клиентов, выполняет журналирование показаний датчиков в файловую систему LittleFS с привязкой ко времени, а также обеспечивает веб-интерфейс для управления системой.

Третий уровень – уровень представления. На данном уровне функционирует мобильное приложение, которое взаимодействует с ESP8266 по протоколу HTTP, отображает текущие показания датчиков, историю измерений в виде графиков и предоставляет интерфейс для установки порогового значения влажности почвы.

На рисунке 15 представлена диаграмма компонентов системы.

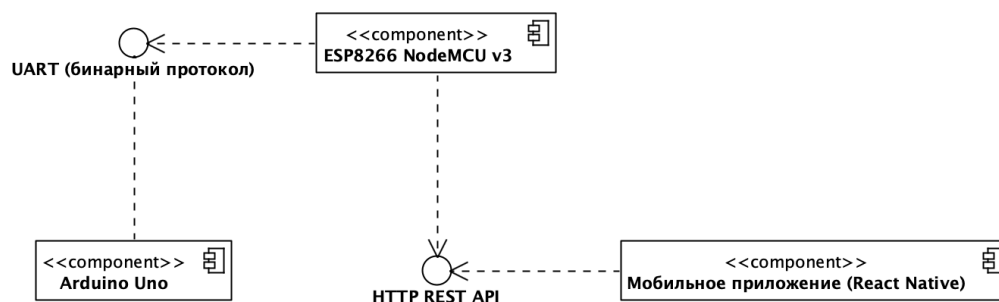


Рисунок 15 – Диаграмма компонентов программного комплекса

Взаимодействие между уровнями осуществляется следующим образом:

- между Arduino и ESP8266 – по бинарному UART-протоколу на скорости 9600 бод;
- между ESP8266 и мобильным приложением – по протоколу HTTP в локальной Wi-Fi сети.

Такая архитектура обеспечивает модульность системы: каждый компонент может быть разработан, протестирован и заменен независимо от остальных, при условии соблюдения заданных интерфейсов взаимодействия.

2.3 Проектирование протокола обмена данными

Для обмена данными между основным микроконтроллером Arduino и модулем ESP8266 разработан бинарный протокол, функционирующий поверх интерфейса UART на скорости 9600 бод [16].

Формат пакета протокола представлен в таблице 4.

Таблица 4 – Формат пакета бинарного протокола

Стартовый байт	Код команды	Данные	Конечный байт
0xAA	cmd	data	0x55

Стартовый байт (0xAA) служит для синхронизации приемника и обнаружения начала пакета. Конечный байт (0x55) обозначает завершение пакета. Код команды определяет тип передаваемых данных. Поле данных содержит значение параметра в виде одного байта (диапазон 0–255).

В таблице 5 представлены коды команд протокола.

Таблица 5 – Коды команд бинарного протокола

Код команды	Назначение
's'(0x73)	Влажность почвы (0–100%)
'h'(0x68)	Влажность воздуха (0–100%)
't'(0x74)	Температура воздуха, целая часть (°C)
'y'(0x79)	Температура воздуха, дробная часть (сотые доли °C)
'r'(0x72)	Пороговое значение влажности почвы (0–100%)
'a'(0x61)	Подтверждение приема (ACK)

Направление передачи данных:

- от Arduino к ESP8266: команды 's', 'h', 't', 'y', 'r' – передача текущих показаний датчиков и порогового значения с периодом 1000 мс;
- от ESP8266 к Arduino: команда 'r' – установка нового порогового значения влажности почвы по запросу пользователя.

При приеме пакета получатель проверяет наличие стартового байта, считывает код команды и данные до обнаружения конечного байта. Пакеты с некорректной структурой отбрасываются. Новое пороговое значение, полученное Arduino от ESP8266, сохраняется в энергонезависимой памяти EEPROM с контрольной суммой для защиты от повреждения данных при сбоях питания [9].

2.4 Проектирование конечных точек и форматов данных

На этапе проектирования определен набор конечных точек, через которые мобильное приложение и веб-интерфейс будут взаимодействовать с модулем ESP8266. Обмен данными предполагается по протоколу HTTP на порту 80 в локальной Wi-Fi сети [17].

В таблице 6 представлено описание спроектированных конечных точек. Таблица 6 – Конечные точки системы

URL	Метод	Описание	Авторизация
/data	GET	Текущие показания датчиков	Нет
/api/history?date=DD-MM	GET	История показаний за указанную дату	Нет
/api/history/latest?count=N	GET	Последние N записей за текущий день	Нет

Продолжение таблицы 6

URL	Метод	Описание	Авторизация
/api/files	GET	Список доступных файлов журнала	Нет
/loginps	POST	Аутентификация (логин, пароль → сессионный токен)	Нет
/logout	GET	Завершение сессии	Нет
/sendCommand	POST	Установка порогового значения влажности почвы	Нет

Представлен формат ответа конечной точки /data в листинге 1, возвращающей текущие показания датчиков.

Листинг 1 – Формат ответа конечной точки /data

```
{
  "temp": 23.45,
  "humidity": 62,
  "soilmoisture": 48,
  "req_soilmoisture": 50
}
```

В листинге 2 представлен формат ответа конечной точки /api/history, возвращающей массив записей журнала за указанную дату.

Листинг 2 – Формат ответа конечной точки /api/history

```
[
  {"time": "10:30:00", "temp": 23.5, "humidity": 60, "soil": 45},
  {"time": "11:00:00", "temp": 24.0, "humidity": 58, "soil": 42}
]
```

Для разграничения доступа спроектирован сессионный механизм аутентификации [18]. При успешной аутентификации через конечную точку /loginps сервер должен генерировать уникальный сессионный токен и возвращать его клиенту в заголовке Set-Cookie. Для выполнения управляющих команд клиент передает токен в параметре запроса session. Просмотр текущих показаний и истории измерений не требует аутентификации.

Журналирование показаний датчиков предполагается с периодом 30 минут. Каждая запись сохраняется в файл с именем, соответствующим текущей дате (формат DD-MM.txt), в файловой системе LittleFS. Синхронизация времени осуществляется по протоколу NTP при запуске устройства [19].

2.5 Проектирование интерфейса мобильного приложения

На этапе проектирования определена структура пользовательского интерфейса мобильного приложения. Навигация организована с использованием нижней панели вкладок [20]. Приложение должно содержать следующие экраны: экран текущих показаний (главный), экран истории измерений, экран настроек и экран авторизации.

На рисунке 16 представлен макет экрана текущих показаний.

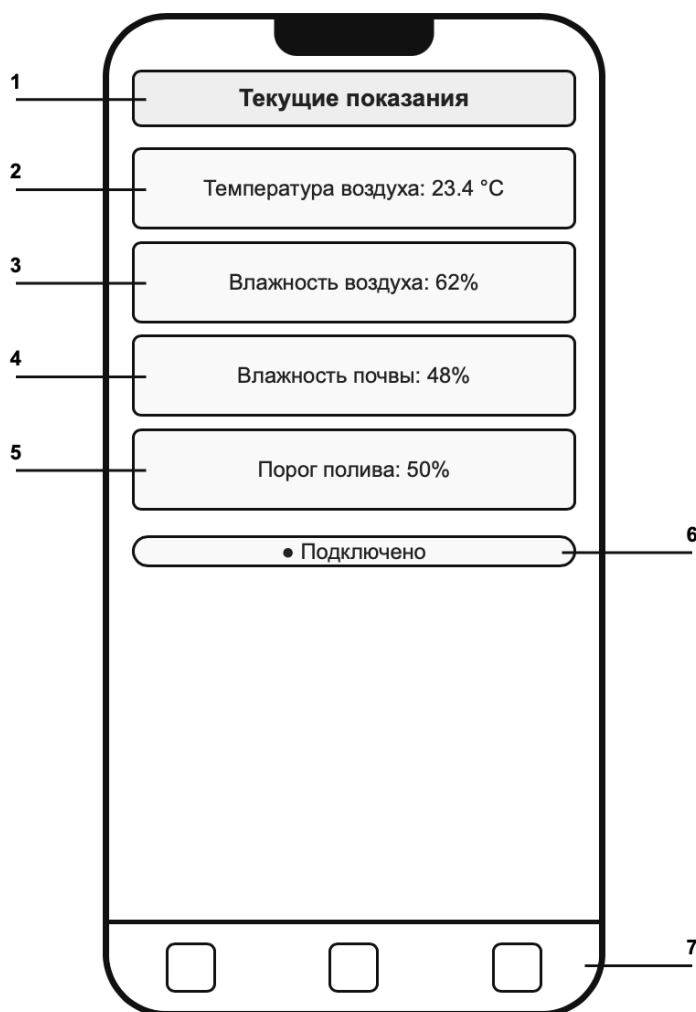


Рисунок 16 – Макет экрана текущих показаний

На экране текущих показаний расположены следующие элементы:

- 1 – заголовок экрана;
- 2 – карточка температуры воздуха (°C);
- 3 – карточка влажности воздуха (%);

4 – карточка влажности почвы (%);

5 – индикатор порогового значения влажности почвы;

6 – индикатор состояния подключения к устройству;

7 – нижняя панель навигации (Tab Bar).

Обновление показаний должно выполняться автоматически каждые 2 секунды.

На рисунке 17 представлен макет экрана истории измерений.



Рисунок 17 – Макет экрана истории измерений

На экране истории измерений расположены следующие элементы:

1 – заголовок экрана;

2 – компонент выбора даты;

3 – график температуры воздуха;

4 – график влажности воздуха;

5 – график влажности почвы;

6 – нижняя панель навигации (Tab Bar).

На рисунке 18 представлен макет экрана настроек.

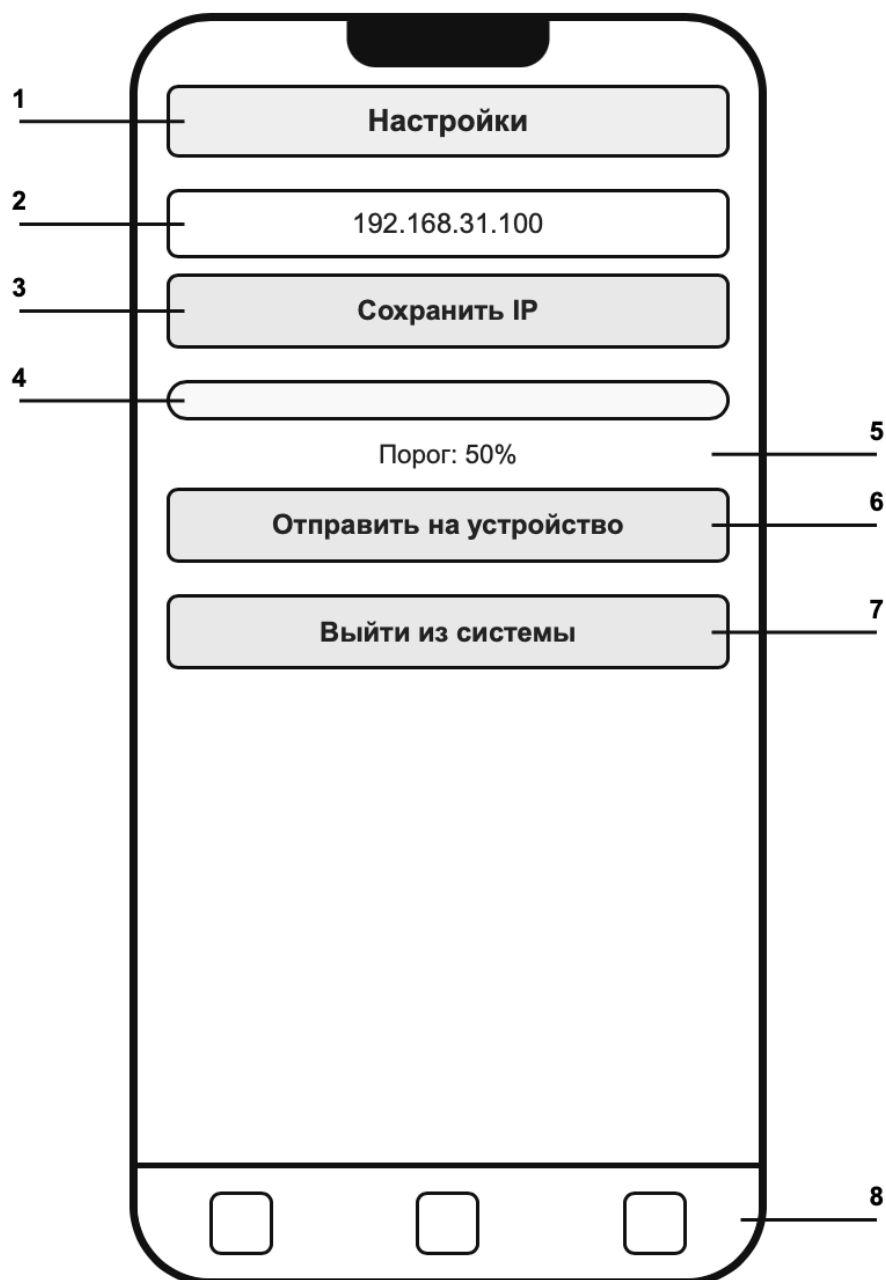


Рисунок 18 – Макет экрана настроек

На экране настроек расположены следующие элементы:

1 – заголовок экрана;

- 2 – поле ввода IP-адреса устройства;
- 3 – кнопка сохранения IP-адреса;
- 4 – ползунок установки порогового значения влажности почвы;
- 5 – текущее значение ползунка;
- 6 – кнопка отправки порогового значения на устройство;
- 7 – кнопка входа/выхода из системы;
- 8 – нижняя панель навигации (Tab Bar).

На рисунке 19 представлен макет экрана авторизации.

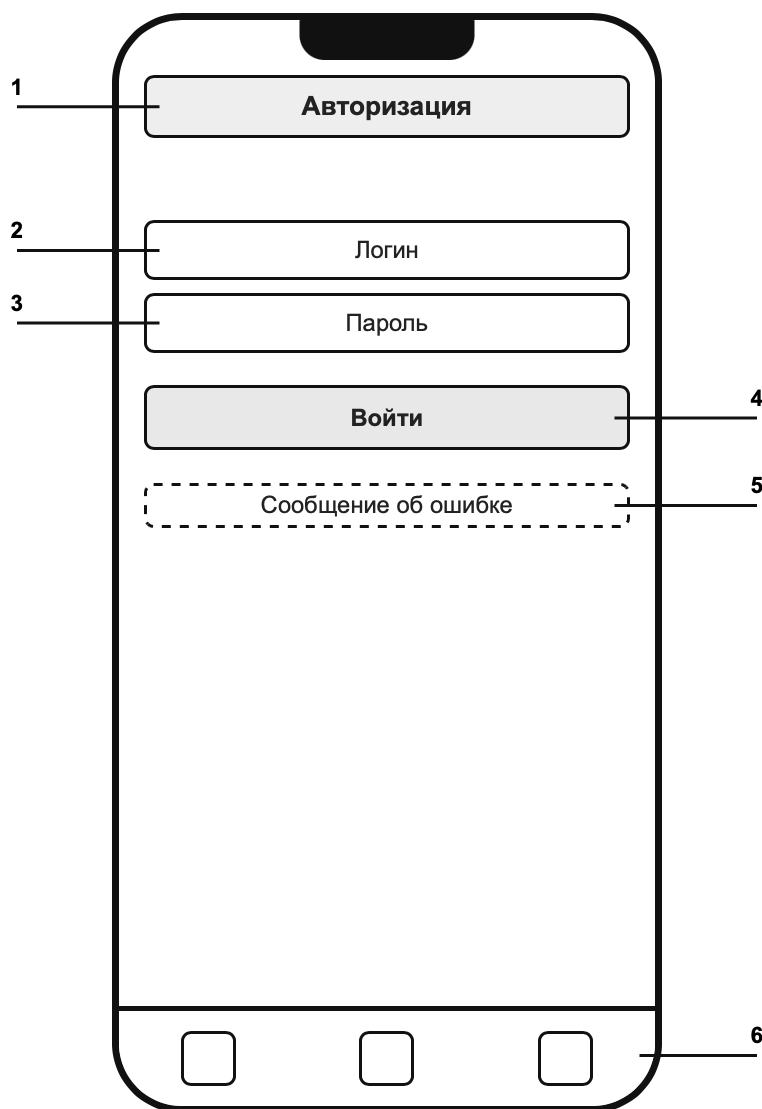


Рисунок 19 – Макет экрана авторизации

На экране авторизации расположены следующие элементы:

- 1 – заголовок экрана;

- 2 – поле ввода логина;
- 3 – поле ввода пароля;
- 4 – кнопка «Войти»;
- 5 – область отображения сообщения об ошибке.

Таким образом, спроектированный интерфейс мобильного приложения обеспечивает удобный доступ ко всем функциям системы. Экран текущих показаний предоставляет оперативную информацию о состоянии микроклимата, экран истории измерений позволяет анализировать динамику параметров за выбранный период, а экран настроек обеспечивает конфигурирование подключения и управление пороговым значением полива. Разделение функциональности по экранам с навигацией через нижнюю панель вкладок соответствует общепринятым паттернам проектирования мобильных интерфейсов и обеспечивает интуитивно понятное взаимодействие пользователя с системой.

2.6 Проектирование структуры хранения данных

На этапе проектирования определена структура хранения данных для каждого уровня программного комплекса.

Для хранения журнала показаний датчиков на модуле ESP8266 выбрана файловая система LittleFS [14]. Данная файловая система специально разработана для микроконтроллеров с ограниченным объемом флеш-памяти и обеспечивает устойчивость к сбоям питания, что является критически важным требованием для автономно работающего устройства.

Спроектирована следующая схема организации хранения данных:

- для каждого дня должен создаваться отдельный файл с именем в формате DD-MM.txt;
- каждая запись в файле должна представлять собой JSON-объект, содержащий время записи, температуру воздуха, влажность воздуха и влажность почвы;
- записи должны разделяться символом переноса строки;

– добавление новых записей должно выполняться в конец файла с периодом 30 минут.

Спроектированный формат одной записи журнала:

```
{"time":"10:30:00","temp":23.45,"humidity":62,"soil":48}.
```

Такая организация хранения обеспечит эффективное извлечение данных за конкретную дату без необходимости чтения всего объема накопленной информации. Конечная точка `/api/history` должна открывать файл, соответствующий запрошенной дате, и формировать JSON-массив из содержащихся в нем записей.

Для хранения порогового значения влажности почвы на стороне основного микроконтроллера Arduino предусмотрено использование энергонезависимой памяти EEPROM. Для защиты от повреждения данных при сбоях питания спроектирован механизм контрольной суммы (XOR всех байтов значения). При запуске микроконтроллер должен считывать сохраненное значение и проверять его целостность; в случае несовпадения контрольной суммы должно использоваться значение по умолчанию (70%).

На стороне мобильного приложения для хранения пользовательских настроек предусмотрено использование AsyncStorage [21] – асинхронного локального хранилища, предоставляемого платформой React Native. Данное решение обеспечивает сохранение настроек между сеансами работы приложения без необходимости обращения к внешним серверам.

3 Реализация программного обеспечения

3.1 Реализация прошивки основного микроконтроллера

Прошивка основного микроконтроллера реализована в виде одного файла `testSystem.ino` на языке C++ для платформы Arduino [9]. Программа выполняет циклический опрос датчиков, управление насосом полива и обмен данными с модулем ESP8266 по бинарному UART-протоколу.

При инициализации системы (функция `setup`) выполняются следующие действия: настройка последовательного порта на скорость 9600 бод, инициализация датчика DHT11, настройка вывода управления насосом в режим выхода, а также чтение сохраненного порогового значения влажности почвы из EEPROM с проверкой контрольной суммы. В случае несовпадения контрольной суммы устанавливается значение по умолчанию – 70%.

Основной цикл программы (функция `loop`) выполняет следующие операции:

- обработку входящих UART-пакетов от ESP8266 (функция `handleSerial`);
- считывание показаний датчика влажности почвы с преобразованием аналогового значения в проценты (функция `readSoilMoisture`);
- считывание температуры и влажности воздуха с датчика DHT11;
- сравнение текущей влажности почвы с пороговым значением и управление насосом;
- периодическую отправку показаний датчиков на ESP8266 с интервалом 1000 мс.

Для работы с энергонезависимой памятью реализованы шаблонные функции `eepromWrite` и `eepromRead`, обеспечивающие запись и чтение произвольных типов данных с контрольной суммой XOR [9]. Контрольная сумма вычисляется как результат операции исключающего ИЛИ над всеми байтами сохраняемого значения и записывается в следующий за данными байт EEPROM.

Преобразование показаний аналогового датчика влажности почвы выполняется с использованием калибровочных констант: значение АЦП 220

соответствует полностью влажной почве (100%), значение 1023 – полностью сухой (0%). Функция `constrain` ограничивает входное значение указанным диапазоном, после чего функция `map` выполняет линейное преобразование в проценты.

Отправка данных по UART реализована функцией `sendByte`, которая формирует пакет из стартового байта, кода команды, одного байта данных и конечного байта. За один цикл передачи отправляются пять пакетов: влажность почвы, влажность воздуха, целая часть температуры, дробная часть температуры и текущее пороговое значение.

Прием команд от ESP8266 реализован в функции `handleSerial`. При обнаружении стартового байта считываются код команды и данные до конечного байта. Поддерживается команда 'r' – установка нового порогового значения влажности почвы, которое сохраняется в EEPROM.

Фрагмент кода программы контроллера Arduino и разработанной библиотеки для него представлен в приложении В. Полный код расположен на сервисе git [22].

3.2 Реализация прошивки модуля ESP8266

Прошивка сетевого модуля реализована в файле `rest.ino` на языке C++ для платформы NodeMCU v3 (ESP8266) [10]. Программа обеспечивает прием данных от Arduino по UART, предоставление REST API и веб-интерфейса по протоколу HTTP, а также журналирование показаний датчиков в файловую систему LittleFS.

При инициализации выполняются следующие действия:

- настройка последовательного порта на скорость 9600 бод;
- подключение к Wi-Fi сети со статическим IP-адресом (192.168.31.100);
- синхронизация времени по протоколу NTP с сервером `pool.ntp.org`;
- монтирование файловой системы LittleFS;
- выполнение первой записи журнала;

- регистрация обработчиков HTTP-маршрутов и запуск веб-сервера на порту 80.

Основной цикл программы выполняет три операции: обработку входящих UART-пакетов (`handleUART`), обработку HTTP-запросов (`server.handleClient`) и периодическое журналирование данных с интервалом 30 минут.

Прием данных от Arduino реализован в функции `handleUART`. Функция использует конечный автомат: при обнаружении стартового байта `0xAA` считываются код команды и байты данных до обнаружения конечного байта `0x55`. В зависимости от кода команды обновляются соответствующие глобальные переменные: `soilMoisture`, `humidity`, `tempInt`, `tempFrac`, `requiredMoisture`.

Журналирование реализовано в функции `logData`. Каждая запись представляет собой JSON-объект с полями `time`, `temp`, `humidity` и `soil` [23]. Записи сохраняются в файл с именем, соответствующим текущей дате (формат `/DD-MM.txt`). Перед записью выполняется проверка на нулевые значения всех датчиков для исключения записи некорректных данных до получения первого пакета от Arduino.

Для получения текущего времени используются функции `getTimeStr` и `getTodayFileName`, которые обращаются к системным часам, синхронизированным по NTP при запуске устройства.

Механизм аутентификации реализован на основе сессионного токена [18]. При успешной аутентификации через конечную точку `/loginps` генерируется уникальный токен (комбинация значения `millis()` и случайного числа), который сохраняется в переменной `activeSession` и передается клиенту через заголовок `Set-Cookie` и параметр URL. Функция `checkSession` проверяет наличие валидного токена в параметре запроса `session` или в заголовке `Cookie`.

REST API реализован с использованием библиотеки `ESP8266WebServer` [10]. Конечная точка `/data` возвращает текущие показания датчиков в формате JSON. Конечная точка `/api/history` принимает параметр `date` и возвращает массив записей из соответствующего файла журнала. Конечная

точка `/api/history/latest` возвращает последние N записей за текущий день с использованием кольцевого буфера. Конечная точка `/api/files` возвращает список доступных файлов журнала. Конечная точка `/sendCommand` принимает новое пороговое значение и передает его Arduino по UART.

Веб-интерфейс генерируется функцией `buildHTML` и представляет собой одностраничное приложение с автоматическим обновлением показаний каждые 2 секунды через JavaScript `fetch API` [24]. Для авторизованных пользователей дополнительно отображается элемент управления пороговым значением.

Фрагмент кода серверной части программы представлен в приложении А. Полный код расположен на сервисе `git` [25].

3.3 Реализация клиентской части приложения

Разработка мобильного приложения производилась на языке TypeScript с использованием фреймворка React Native и платформы Expo SDK 54 [15]. В качестве среды программирования использовался Visual Studio Code [26].

Диаграмма компонентов клиентской части приложения представлена на рисунке 20.

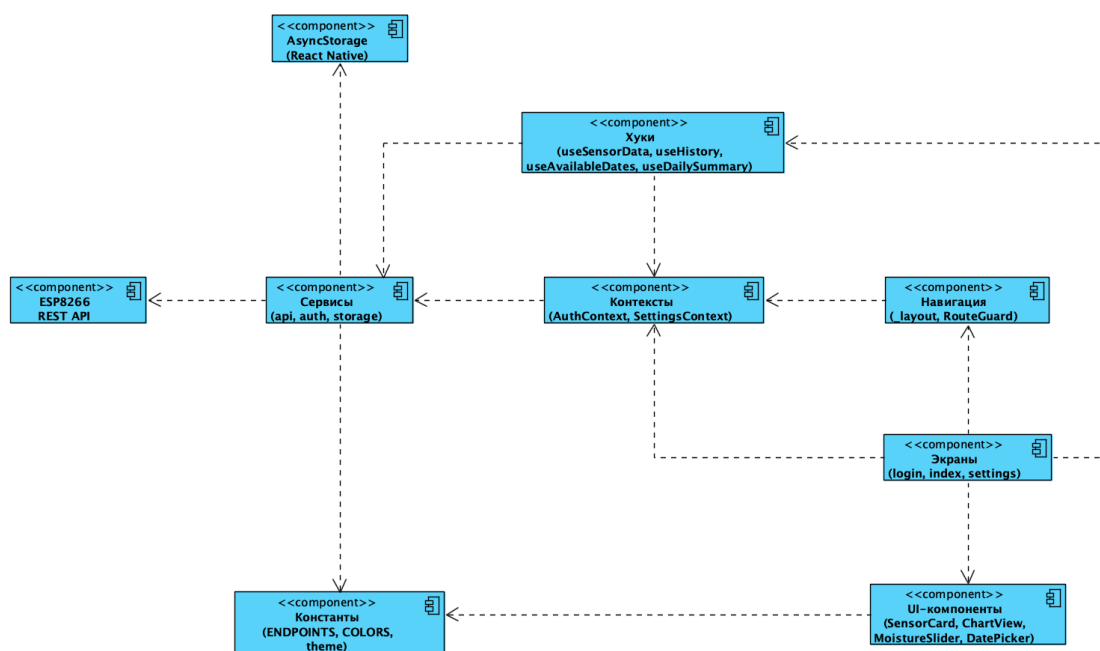


Рисунок 20 – Диаграмма компонентов мобильного приложения

Описание компонентов представлено в таблице 7.

Таблица 7 – Компоненты клиентского приложения

Компонент	Состав	Назначение
Навигация	app/_layout.tsx, app/(tabs)/_layout.tsx	Является точкой входа приложения; оборачивает дерево компонентов в провайдеры SettingsProvider и AuthProvider; компонент RouteGuard реализует защиту маршрутов – перенаправляет пользователя на экран настроек при отсутствии IP-адреса или на главный экран при успешной аутентификации; (tabs)/_layout.tsx определяет структуру нижней панели вкладок и параметры ее отображения
Экраны	app/login.tsx, app/(tabs)/index.tsx, app/(tabs)/settings.tsx	login.tsx содержит форму авторизации и вызывает функцию login из AuthContext; index.tsx отображает текущие показания датчиков в виде карточек и графики истории в режимах «За день» и «По дням»; settings.tsx обеспечивает ввод IP-адреса, установку целевых значений параметров, настройку уведомлений, вход и выход из системы
Контексты	context/AuthContext.tsx, context/SettingsContext.tsx	SettingsContext хранит IP-адрес устройства, загружает его из AsyncStorage при запуске и предоставляет функцию для его обновления; AuthContext управляет сессионным токеном, реализует функции login и logout, при входе выполняет HTTP-запрос к конечной точке /loginps и сохраняет токен в постоянном хранилище
Хуки	hooks/useSensorData.ts, hooks/useHistory.ts, hooks/useAvailableDates.ts, hooks/useDailySummary.ts	useSensorData выполняет периодический опрос конечной точки /data с интервалом 2000 мс; useHistory загружает историю показаний за указанную дату и фильтрует нулевые записи; useAvailableDates получает список файлов журнала и преобразует их в даты; useDailySummary вычисляет средние значения параметров по дням; все хуки переподключаются при изменении IP-адреса или токена
UI-компоненты	components/SensorCard.tsx, components/ChartView.tsx, components/MoistureSlider.ts, components/DatePicker.tsx	SensorCard отображает название, значение и единицу измерения параметра; ChartView строит линейный график с заливкой области, пунктирной линией целевого значения и всплывающей подсказкой при касании; MoistureSlider предоставляет ползунок установки порога полива и доступен только авторизованным пользователям; DatePicker реализован в виде горизонтального списка дат-чипов
Сервисы	services/api.ts, services/auth.ts, services/storage.ts	api.ts реализует фабричную функцию createApiClient с методами getSensorData, getHistory, getLatest, getFiles и setThreshold; сессионный токен автоматически добавляется ко всем запросам; auth.ts реализует вход и выход, извлекая токен из ответа сервера; storage.ts предоставляет типизированную обертку над AsyncStorage для хранения IP-адреса, токена, настроек уведомлений и целевых значений параметров
Константы	constants/api.ts, constants/ui.ts, constants/theme.ts	api.ts содержит адреса конечных точек REST API устройства: /data, /api/history, /api/history/latest, /api/files, /loginps, /logout, /sendCommand; ui.ts содержит цветовую палитру приложения; theme.ts хранит расширенные настройки цветовой схемы и шрифтов для светлой и темной тем

Как показано на диаграмме, приложение разделено на семь внутренних компонентов и взаимодействует с двумя внешними. Компоненты экранов и

навигации формируют пользовательский интерфейс и опираются на контексты для получения глобального состояния, а на хуки – для получения данных с устройства. Хуки и контексты обращаются к сервисному слою, который, в свою очередь, взаимодействует с AsyncStorage для персистентного хранения настроек и с REST API устройства ESP8266 по протоколу HTTP для получения показаний датчиков и передачи команд управления.

Диаграмма развертывания, описывающая физическое расположение компонентов системы, представлена на рисунке 21.

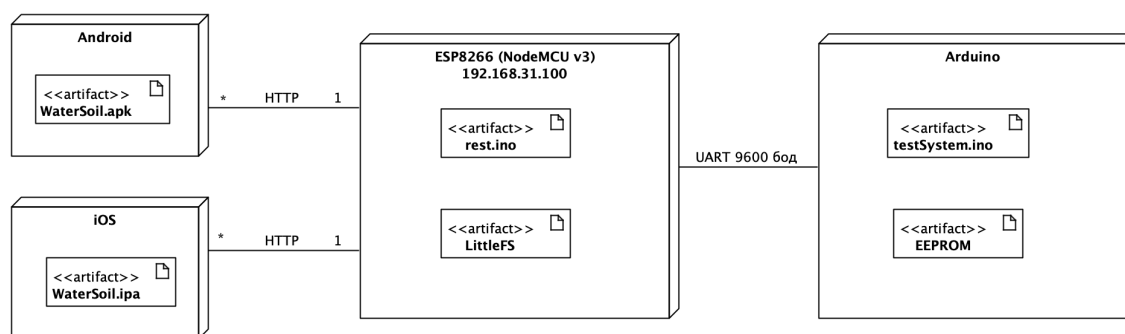


Рисунок 21 – Диаграмма развертывания системы

Как показано на диаграмме, мобильное приложение существует в двух исполняемых формах: WaterSoil.ipa для устройств под управлением iOS и WaterSoil.apk для устройств под управлением Android. Приложение подключается к локальной Wi-Fi сети и взаимодействует с модулем ESP8266 по протоколу HTTP на порту 80; модуль имеет статический IP-адрес 192.168.31.100. На ESP8266 разворачивается прошивка rest.ino, реализующая HTTP-сервер и ведущая журналы показаний в файловой системе LittleFS в файлах формата /DD-MM.txt. Модуль ESP8266 соединен с микроконтроллером Arduino по протоколу UART со скоростью 9600 бод. На Arduino разворачивается прошивка testSystem.ino, обеспечивающая опрос датчиков DHT11 и аналогового датчика влажности почвы, управление насосом полива через реле, а также хранение порогового значения влажности в энергонезависимой памяти EEPROM.

Фрагмент кода клиентской части программы представлен в приложении Б. Полный код расположен на сервисе git [27].

4 Тестирование программного продукта

4.1 Функциональное тестирование черным ящиком

Тестирование методом черного ящика – это вид тестирования, при котором при проверке не учитывается внутренняя структура приложения [28]. На вход программы подаются наборы входных данных и полученные выходные данные сравниваются с ожидаемыми.

Рассмотрим данный вид тестирования на примере вводимого поля температуры. Ниже в таблице 8 определены классы эквивалентности.

Таблица 8 – Классы эквивалентности

Название параметра	Правильные КЭ	Неправильные КЭ
Длина температуры	3-5(1)	0-2(2) 6-∞ (3)
Структура	Перед числом может быть дефис(4) Температура должна разделяться запятой(5) После запятой должна быть одна цифра(6)	Дефис не стоит на первом месте(7) Температура не содержит дробной части(8) Более одной цифры после запятой(9)
Содержание	Дефис(10) Цифры(11) Запятая(12)	Остальное(13)

В таблице 9 отображены граничные условия для поля температуры

Таблица 9 – Граничные условия

Границы	Значение
Физические	Число отрицательной длины
Логические	Ограничение температуры от -45 до 50 в теплице()
Технологические	Температура ограничена размером байта от -127 до 128
Произвольные	Минимальная длина 2, максимальная длина 5, наличие дефиса только в начале, допустимые символы (цифры, запятая, дефис).

Результаты функционального тестирования представлены в таблице 10.

Таблица 10 – Тест-кейсы

Классы эквивалентности	Входные данные	Выходные данные	Фактический результат
1,4,5,6,10,11,12	-13,1	Успешно	Вывод в консоль «Успешно»
2,4,5,6,10,11,12	1	Ошибка, неверные показатели термометра	Вывод в консоль «Ошибка, неверные показатели термометра»
3,4,5,6,10,11,12	1000,2	Ошибка, неверные показатели термометра	Вывод в консоль «Ошибка, неверные показатели термометра»
1,5,6,7,10,11,12	10,1-13,0	Ошибка, неверные показатели термометра	Вывод в консоль «Ошибка, неверные показатели термометра»
1,4,8,6,10,11,12	133	Ошибка, неверные показатели термометра	Вывод в консоль «Ошибка, неверные показатели термометра»
1,4,5,9,10,11,12	13,23	Ошибка, неверные показатели термометра	Вывод в консоль «Ошибка, неверные показатели термометра»
1,4,5,6,13	Фв,ы	Ошибка, неверные показатели термометра	Вывод в консоль «Ошибка, неверные показатели термометра»

Все тест-кейсы были пройдены успешно, что говорит о корректной работе программы.

4.2 Функциональная диаграмма и таблица решений

Для формализации логики работы системы автоматического полива необходимо четко определить, при каких условиях происходит включение исполнительного механизма. Основным методом спецификации логики управления в данном случае выступает метод «причина-следствие». Этот подход позволяет наглядно представить зависимости между входными параметрами (причинами) и выходными состояниями системы (следствиями), а также выявить все возможные комбинации условий, включая те, которые могут привести к некорректной работе.

Рассмотрим логику включения полива. Полив включается только при отсутствии ошибок, если влажность ниже установленного порога (по умолчанию) и температура не выше 40°C.

Выделим причины событий для логики включения полива (таблица 11).

Таблица 11 – Причины событий

Причины	
П1	Ошибка отсутствует
П2	Влажность почвы ниже порога
П3	Температура $\leq 40^{\circ}\text{C}$

Также выделим следствия событий для логики включения полива (таблица 12).

Таблица 12 – Следствия событий

Следствия	
С1	Полив включен
С2	Полив выключен

Для этих событий составим функциональную диаграмму (рисунок 22).

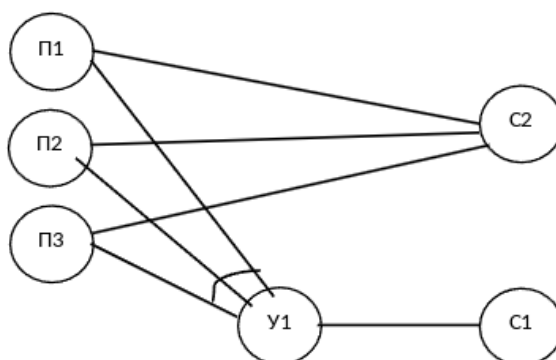


Рисунок 22 – Функциональная диаграмма

Определим таблицу решений для описанных событий (таблица 13).

Таблица 13 – Таблица решений

П1	0	1	0	0	1	1	0	1
П2	0	0	1	0	1	0	1	1
П3	0	0	0	1	0	1	1	1
С1	0	0	0	0	0	0	0	1
С2	1	1	1	1	1	1	1	0

На основе составленной таблицы решений были разработаны тест-кейсы (таблица 14).

Таблица 14 – Тест-кейсы

Входные данные	Выходные данные	Фактический результат
Ошибка есть, влажность выше порога, $t > 40$	Полив выключен	Полив выключен
Ошибка есть, влажность выше порога, $t \leq 40$	Полив выключен	Полив выключен

Продолжение таблицы 14

Входные данные	Выходные данные	Фактический результат
Ошибка есть, влажность ниже порога, $t > 40$	Полив выключен	Полив выключен
Ошибка есть, влажность ниже порога, $t \leq 40$	Полив выключен	Полив выключен
Ошибки нет, влажность выше порога, $t > 40$	Полив выключен	Полив выключен
Ошибки нет, влажность выше порога, $t \leq 40$	Полив выключен	Полив выключен
Ошибки нет, влажность ниже порога, $t > 40$	Полив выключен	Полив выключен
Ошибки нет, влажность ниже порога, $t \leq 40$	Полив включен	Полив включен

Разработанные на основе функциональной диаграммы тест-кейсы были выполнены в полном объеме. Фактические результаты полностью совпали с ожидаемыми: полив активируется только при одновременном выполнении всех трех условий, во всех остальных случаях система остается в безопасном состоянии с выключенным поливом.

4.3 Модульное тестирование

Модульное тестирование представляет собой процесс проверки отдельных модулей или компонентов программы [29], выполняемый изолированно, автоматизированно, с использованием заглушек. Основная цель такого тестирования – убедиться в корректности работы каждого модуля по отдельности. В контексте разрабатываемой системы автополива особую значимость приобретает надежность модулей, отвечающих за отсчет времени. Ошибки на уровне отдельных компонентов могут привести к нарушению режима полива, поэтому их тщательное тестирование является обязательным этапом.

В рамках данной работы модульное тестирование применялось для проверки библиотеки модуля часов реального времени DS3231, который обеспечивает точное временное управление системой. Для этого был разработан тестовый класс DS3231Test, код которого представлен в приложении Д. Результаты выполнения тестов приведены на рисунке 23.

```

Running main() from /Users/nikitahmyrenkov/Desktop/labs/тестирование/testdir/goog
gletest/googletest/src/gtest_main.cc
[=====] Running 5 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 5 tests from DS3231Test
[ RUN      ] DS3231Test.SetDateTime_EncodesAndWritesCorrectly
[       OK ] DS3231Test.SetDateTime_EncodesAndWritesCorrectly (0 ms)
[ RUN      ] DS3231Test.GetDateTime_ReadsAndFormatsCorrectly
[       OK ] DS3231Test.GetDateTime_ReadsAndFormatsCorrectly (0 ms)
[ RUN      ] DS3231Test.GetTemperature_PositiveValue
[       OK ] DS3231Test.GetTemperature_PositiveValue (0 ms)
[ RUN      ] DS3231Test.GetTemperature_IntegerValue
[       OK ] DS3231Test.GetTemperature_IntegerValue (0 ms)
[ RUN      ] DS3231Test.GetTemperature_Fraction075
[       OK ] DS3231Test.GetTemperature_Fraction075 (0 ms)
[-----] 5 tests from DS3231Test (0 ms total)

[-----] Global test environment tear-down
[=====] 5 tests from 1 test suite ran. (0 ms total)
[ PASSED  ] 5 tests.
zsh: segmentation fault ./test/ds3231_test
nikitahmyrenkov@MacBook-Pro-Nikita-2 build %

```

Рисунок 23 – Результаты модульного тестирования

Все тесты были успешно пройдены, что свидетельствует о корректной работе тестируемых методов. Полученные результаты подтверждают, что модуль часов реального времени функционирует в соответствии с требованиями, а его использование в системе автополива не внесет ошибок, связанных с временными задержками.

4.4 Метрики кода

Метрики кода представляют собой количественные характеристики, позволяющие оценить сложность, качество и читаемость программного обеспечения [30]. От структуры и понятности кода зависит не только удобство последующей доработки, но и надежность работы устройства в целом. Чрезмерно сложные или запутанные модули повышают риск некорректной работы. В связи с этим в рамках работы проведен анализ ключевых метрик для основного цикла управления поливом.

Рассмотрим метод «loop», отвечающий за основную обработку полива. Его код представлен в листинге 3.

Листинг 3 – Метод создания заказа

```

void loop() {
    //обработка команд с консоли

```

```

handleSerialCommands();
//получение информации с датчиков
byte soilmoisture = get_data_soilmoisture();
float temperature = dht.readTemperature();
byte humidity = dht.readHumidity();
bool isWatering = (soilmoisture < needSoilmoisture);
//управление поливом
digitalWrite(POWER_KEY_WATER, isWatering ? HIGH : LOW);
//передача данных второму микроконтроллеру
if (millis() - millis_int1 >= refresh_rate) {
    sendByteCommand(CMD_SOIL, soilmoisture);
sendByteCommand(CMD_TEMP, temperature);
sendByteCommand(CMD_HUM, humidity);
    millis_int1 = millis();
}
//сохранение на sd накопитель
if (millis() - millis_int2 >= refreshRateSaveData) {
    SD.write(toJson(soilmoisture, humidity, temperature)) ;
    millis_int2 = millis();
}
}

```

Данный метод выполняется на микроконтроллере, ответственном за полив. Он опрашивает датчики, включает и выключает насос, передает данные на второй контроллер и сохраняет информацию на SD-карту. Для количественной оценки его сложности используются две группы метрик: метрики размера программы и метрики Холстеда

Метрики размера программы представляют собой подсчет количества строк кода, а также строк с комментариями. Они просты в вычислении и хорошо подходят для проектов, реализованных на одном языке программирования. Результаты вычислений приведены в таблице 15.

Таблица 15 – Метрики размера программы

Метрика	Количество
Количество строк кода	23
Строки-комментарии	4

Доля комментариев составляет $(4/23) * 100 \approx 17,4 \%$. Небольшой объем метода и наличие поясняющих комментариев свидетельствуют об отсутствии избыточности и приемлемой структурированности кода.

Метрики Холстеда помогают оценить сложность программы. Полученные, благодаря подсчету, метрики помогают разработчикам понимать и контролировать сложность программного обеспечения, выявлять потенциальные области для оптимизации и повышать общее качество программного обеспечения. Для вычисления метрик Холстеда подсчитаем необходимые характеристики программы:

- число уникальных операторов n_1 ;
- число уникальных операндов n_2 ;
- общее число операторов N_1 ;
- общее число операндов N_2 .

Составим словарь операторов и операндов для вычисления метрик.

Словарь операторов представлен в таблице 16.

Таблица 16 – Словарь операторов

Операторы	Число вхождений
handleSerialCommands	1
get_data_soilmoisture	1
dht.readTemperature	1
<	1
digitalWrite	1
millis	4
toJson	1
SD.write	1
sendByteCommand	3
?	1
:	1
if	2
=	6
>=	2
-	2
{	2
}	2
;	12

Словарь операндов представлен в таблице 17.

Таблица 17 – Словарь операндов

Операнды	Число вхождений
soilmoisture	4
temperature	3
dht	1
isWatering	2
needSoilmoisture	1
POWER_KEY_WATER	1
HIGH	1
LOW	1
millis_int1	2
millis_int2	2
humidity	3
refreshRateSaveData	1
refresh_rate	1
CMD_SOIL	1
CMD_TEMP	1
CMD_HUM	1

Таким образом, после сложения всех уникальных операторов $n_1 = 18$, после сложения всех вхождений операторов $n_2 = 16$, после сложения всех уникальных операндов $N_1 = 44$, после сложения всех вхождений операндов $N_2 = 26$.

На основе метрик Холстеда можно вычислить целый ряд характеристик программы [31]. Рассмотрим некоторые из них.

Расчет словаря программы производится по формуле:

$$n = n_1 + n_2 = 18 + 16 = 34,$$

где n_1 – количество операторов в программе;

n_2 – количество операндов программе.

Расчет длины программы производится по формуле:

$$N = N_1 + N_2 = 44 + 26 = 80,$$

где N_1 – число вхождений всех операторов в программе;

N_2 – число вхождений всех операндов программе.

Расчет объема программы производится по формуле:

$$V = N * \log_2 n = 80 * \log_2 34 = 356,12,$$

где N – длина программы;
 n – словарь программы.

Расчет сложности программы производится по формуле:

$$D = (n_1/2) * (N_2/n_2) = (18/2) * (26/16) = 14,62,$$

где n_1 – количество операторов в программе;
 N_2 – число вхождений всех операндов программе;
 n_2 – количество операндов программе.

Расчет оценки интеллектуальных усилий производится по формуле:

$$E = D * V = 14,62 * 356,12 = 5208,29$$

где D – сложность программы;
 V – объем программы.

Полученные значения позволяют сделать выводы о характеристиках анализируемого модуля. Объем программы находится на умеренном уровне, что характерно для компактных встраиваемых приложений. Показатель сложности и требуемые умственные затраты не превышают критических порогов, однако указывают на наличие определенной структурной нагрузки, связанной в основном с использованием нескольких ветвлений и обращений к глобальным объектам

Заключение

В ходе выполнения выпускной квалификационной работы разработан программный комплекс мониторинга и управления микроконтроллерной системой автополива.

На этапе анализа требований проведен анализ предметной области и постановка задачи, выделены основные функции программного комплекса. Проанализирован рынок аналогичных программных продуктов в результате чего выявлены их достоинства и недостатки, определены функциональные и нефункциональные требования к разрабатываемому продукту и произведен анализ целевой аудитории. Осуществлено планирование и оценка затрачиваемого бюджета. Общий срок разработки составляет 43 рабочих дней. Общая сумма разработки составляет 398 549 рублей. Данные показатели говорят об экономической целесообразности разрабатываемого продукта.

На этапе проектирования выбраны технологии и инструментальные средства разработки. Спроектирована трехуровневая архитектура системы, бинарный UART-протокол обмена данными между микроконтроллерами, набор конечных точек REST API с форматами данных, структура хранения журнала показаний, а также интерфейс мобильного приложения.

На этапе разработки реализована прошивка основного микроконтроллера Arduino на языке C++ в среде Arduino IDE, обеспечивающая сбор данных с датчиков и управление насосом полива. Прошивка модуля ESP8266 разработана на языке C++ в среде Arduino IDE с использованием библиотек ESP8266WiFi, ESP8266WebServer и ArduinoJson. Мобильное приложение разработано на языке TypeScript с использованием фреймворка React Native и платформы Expo SDK 54 в среде Visual Studio Code.

На этапе тестирования проверена работоспособность и корректность разрабатываемого программного комплекса. Все компоненты системы функционируют в соответствии со спроектированной архитектурой и протоколами взаимодействия.

В дальнейшем планируется опубликовать мобильное приложение в онлайн-маркете Google Play, реализовать поддержку нескольких зон полива с независимыми пороговыми значениями, а также добавить push-уведомления при критическом отклонении параметров от заданных значений.

Таким образом, в ходе выполнения выпускной квалификационной работы успешно выполнены все поставленные задачи.

Список использованных источников

1. Интернет вещей: тренды и перспективы российского рынка. [Электронный ресурс] // TAdviser. – URL: [https://www.tadviser.ru/index.php/Статья:Интернет_вещей_\(IoT,_Россия\)](https://www.tadviser.ru/index.php/Статья:Интернет_вещей_(IoT,_Россия)) (дата обращения: 19.03.2026).
2. Gardena Smart System. [Электронный ресурс] // Официальный сайт Gardena. – URL: <https://www.gardena.com/ru/produkty/poliv/smart-sistema/> (дата обращения: 19.03.2026).
3. Home Assistant. [Электронный ресурс] // Официальный сайт проекта Home Assistant. – URL: <https://www.home-assistant.io/> (дата обращения: 19.03.2026).
4. Blynk Platform. [Электронный ресурс] // Официальный сайт Blynk. – URL: <https://blynk.io/> (дата обращения: 19.03.2026).
5. Объем российского рынка умного дома. [Электронный ресурс] // J'son & Partners Consulting. – URL: <https://json.tv/> (дата обращения: 19.03.2026).
6. Статистика доли мобильных операционных систем на мировом рынке. [Электронный ресурс] // Statcounter Global Stats. – URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide> (дата обращения: 19.03.2026).
7. Статистика доли мобильных операционных систем на российском рынке. [Электронный ресурс] // Statcounter Global Stats. – URL: <https://gs.statcounter.com/os-market-share/mobile/RU> (дата обращения: 19.03.2026).
8. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч, Р. Максимчук, М. Энгл и др. – 3-е изд. – М. : Вильямс, 2010. – 720 с.
9. Arduino. [Электронный ресурс] // Официальный сайт Arduino. – URL: <https://www.arduino.cc/> (дата обращения: 19.03.2026).

10. ESP8266 Arduino Core Documentation. [Электронный ресурс] // GitHub. – URL: <https://arduino-esp8266.readthedocs.io/> (дата обращения: 19.03.2026).
11. React Native. [Электронный ресурс] // Официальный сайт React Native. – URL: <https://reactnative.dev/> (дата обращения: 19.03.2026).
12. Visual Paradigm. [Электронный ресурс] // Официальный сайт Visual Paradigm. – URL: <https://www.visual-paradigm.com/> (дата обращения: 19.03.2026).
13. DHT11 Humidity & Temperature Sensor. [Электронный ресурс] // Mouser Electronics. – URL: <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf> (дата обращения: 19.03.2026).
14. LittleFS – A little fail-safe filesystem. [Электронный ресурс] // GitHub. – URL: <https://github.com/littlefs-project/littlefs> (дата обращения: 19.03.2026).
15. Expo Documentation. [Электронный ресурс] // Официальный сайт Expo. – URL: <https://docs.expo.dev/> (дата обращения: 19.03.2026).
16. ATmega328P Datasheet. [Электронный ресурс] // Microchip Technology. – URL: <https://www.microchip.com/en-us/product/ATmega328P> (дата обращения: 19.03.2026).
17. Филдинг, Р. Архитектурные стили и проектирование сетевых программных архитектур (REST) / Р. Филдинг // Диссертация. – Калифорнийский университет, 2000. – 162 р.
18. Основы аутентификации и авторизации в веб-приложениях. [Электронный ресурс] // OWASP Foundation. – URL: <https://owasp.org/www-project-web-security-testing-guide/> (дата обращения: 19.03.2026).
19. Mills, D. L. Network Time Protocol (Version 3) Specification, Implementation and Analysis / D. L. Mills // RFC 1305. – 1992. – 120 р.
20. Material Design Guidelines: Navigation. [Электронный ресурс] // Google. – URL: <https://m3.material.io/components/navigation-bar/overview> (дата обращения: 19.03.2026).

21. Async Storage. [Электронный ресурс] // React Native Community. – URL: <https://react-native-async-storage.github.io/async-storage/> (дата обращения: 19.03.2026).
22. Программное обеспечение основного микроконтроллера [Электронный ресурс] / Хмыренков Н.А. – URL: <https://github.com/xnuke2/testSystem> (дата обращения: 19.05.2026).
23. ArduinoJson. [Электронный ресурс] // Официальный сайт ArduinoJson. – URL: <https://arduinojson.org/> (дата обращения: 19.03.2026).
24. Fetch API. [Электронный ресурс] // MDN Web Docs. – URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата обращения: 19.03.2026).
25. Программное обеспечение микроконтроллера ESP8266 RestLolinNodeMCU [Электронный ресурс] / Хмыренков Н.А. – URL: <https://github.com/xnuke2/restLolinNodeMCU-v3> (дата обращения: 19.05.2026).
26. Visual Studio Code. [Электронный ресурс] // Официальный сайт Microsoft. – URL: <https://code.visualstudio.com/> (дата обращения: 19.03.2026).
27. Мобильное приложение WaterSoil [Электронный ресурс] / Хмыренков Н.А. – URL: <https://github.com/xnuke2/waterSoil> (дата обращения: 19.05.2026).
28. Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер. – 3-е изд. – М. : Вильямс, 2012. – 272 с.
29. Осликовски, Д. Модульное тестирование программного обеспечения. [Электронный ресурс] // Хабр. – URL: <https://habr.com/ru/articles/169381/> (дата обращения: 19.03.2026).
30. Фентон, Н. Метрики программного обеспечения: строгий и практический подход / Н. Фентон, Дж. Бигман. – 3-е изд. – CRC Press, 2014. – 617 р.
31. Halstead, M. H. Elements of Software Science / M. H. Halstead. – New York : Elsevier, 1977. – 128 р.

Приложение А
(обязательное)
Исходный код серверной части

```

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ArduinoJson.h>
#include <LittleFS.h>
#include <time.h>

#define PROTO_START 0xAA
#define PROTO_END 0x55

#define CMD_SOIL 's'
#define CMD_HUMIDITY 'h'
#define CMD_TEMP_INT 't'
#define CMD_TEMP_FRAC 'y'
#define CMD_REQUIRED 'r'

// --- WiFi ---
const char* ssid = "Xiaomi_8903";
const char* wifiPassword = "kh39lt45";

IPAddress local_ip(192, 168, 31, 100);
IPAddress gateway(192, 168, 31, 1);
IPAddress subnet(255, 255, 255, 0);
IPAddress dns(8, 8, 8, 8);

// --- NTP ---
#define NTP_SERVER "pool.ntp.org"
#define UTC_OFFSET 3 * 3600

// --- Авторизация ---
const char* adminLogin = "admin";
const char* adminPassword = "12345";
String activeSession = "";

// --- Тайминги ---
#define LOG_INTERVAL_MS (30UL * 60000UL)

// --- Данные датчиков ---
byte tempInt = 0;
byte tempFrac = 0;
byte humidity = 0;
byte soilMoisture = 0;
byte requiredMoisture = 50;

// --- Состояние ---
bool fsReady = false;
unsigned long lastLogTime = 0;

ESP8266WebServer server(80);

String getTimeStr() {
    time_t now = time(nullptr);
    struct tm* t = localtime(&now);
    char buf[20];
    snprintf(buf, sizeof(buf), "%02d:%02d:%02d", t->tm_hour, t->tm_min, t->tm_sec);
    return String(buf);
}

String getDateTimeStr() {

```

```

    time_t now = time(nullptr);
    struct tm* t = localtime(&now);
    char buf[20];
    snprintf(buf, sizeof(buf), "%02d.%02d.%04d %02d:%02d:%02d",
             t->tm_mday, t->tm_mon + 1, t->tm_year + 1900,
             t->tm_hour, t->tm_min, t->tm_sec);
    return String(buf);
}

String getTodayFileName() {
    time_t now = time(nullptr);
    struct tm* t = localtime(&now);
    char buf[16];
    snprintf(buf, sizeof(buf), "%02d-%02d.txt", t->tm_mday, t->tm_mon + 1);
    return String(buf);
}

String generateSession() {
    return String(millis()) + String(random(10000));
}

bool checkSession() {
    if (activeSession == "")
        return false;

    if (server.hasArg("session") && server.arg("session") == activeSession)
        return true;

    if (server.hasHeader("Cookie")) {
        String cookie = server.header("Cookie");
        if (cookie.indexOf("session=" + activeSession) != -1)
            return true;
    }

    return false;
}

String sessionParam() {
    return "?session=" + activeSession;
}

void handleUART() {
    while (Serial.available() >= 4) {
        if (Serial.peek() != PROTO_START) {
            Serial.read();
            continue;
        }

        Serial.read();
        char cmd = Serial.read();
        byte data[10];
        byte idx = 0;

        while (Serial.available() > 0 && idx < 10) {
            byte b = Serial.read();
            if (b == PROTO_END)
                goto packet_done;
            data[idx++] = b;
        }
        continue;
    }
}

```

```

packet_done:
    if (idx < 1)
        continue;

    switch (cmd) {
        case CMD_SOIL:      soilMoisture = data[0]; break;
        case CMD_HUMIDITY:  humidity = data[0]; break;
        case CMD_TEMP_INT:  tempInt = data[0]; break;
        case CMD_TEMP_FRAC: tempFrac = data[0]; break;
        case CMD_REQUIRED:  requiredMoisture = data[0]; break;
    }
}

void sendCommand(char cmd, byte value) {
    Serial.write(PROTO_START);
    Serial.write(cmd);
    Serial.write(value);
    Serial.write(PROTO_END);
    Serial.println();
}

void logData() {
    if (!fsReady)
        return;

    if (tempInt == 0 && tempFrac == 0 && humidity == 0 && soilMoisture == 0)
        return;

    float temp = tempInt + (float)tempFrac / 100.0;
    String fileName = getTodayFileName();

    StaticJsonDocument<128> doc;
    doc["time"] = getTimeStr();
    doc["temp"] = temp;
    doc["humidity"] = humidity;
    doc["soil"] = soilMoisture;

    File f = LittleFS.open(fileName, "a");
    if (f) {
        serializeJson(doc, f);
        f.println();
        f.close();
    }
}

void handleApiHistory() {
    if (!fsReady) {
        server.send(503, "application/json", "{\"error\":\"FS not ready\"}");
        return;
    }

    String date = server.hasArg("date") ? server.arg("date") :
getTodayFileName().substring(1, 6);
    String fileName = "/" + date + ".txt";

    File f = LittleFS.open(fileName, "r");
    if (!f) {
        server.send(404, "application/json", "{\"error\":\"file not found\"}");
    }
}

```

```

        return;
    }

    server.setContentLength(CONTENT_LENGTH_UNKNOWN);
    server.send(200, "application/json", "");
    server.sendContent("[");

    bool first = true;
    while (f.available()) {
        String line = f.readStringUntil('\n');
        line.trim();
        if (line.length() == 0)
            continue;

        if (!first)
            server.sendContent(",");
        server.sendContent(line);
        first = false;
    }

    server.sendContent("]");
    f.close();
}

void handleApiLatest() {
    if (!fsReady) {
        server.send(503, "application/json", "{\"error\":\"FS not ready\"}");
        return;
    }

    int count = server.hasArg("count") ? server.arg("count").toInt() : 10;
    if (count <= 0) count = 10;
    if (count > 50) count = 50;

    String fileName = getTodayFileName();
    File f = LittleFS.open(fileName, "r");
    if (!f) {
        server.send(404, "application/json", "[]");
        return;
    }

    String lines[50];
    int total = 0;
    while (f.available()) {
        String line = f.readStringUntil('\n');
        line.trim();
        if (line.length() > 0) {
            lines[total % count] = line;
            total++;
        }
    }
    f.close();

    int start = (total > count) ? (total % count) : 0;
    int num = (total > count) ? count : total;

    server.setContentLength(CONTENT_LENGTH_UNKNOWN);
    server.send(200, "application/json", "");
    server.sendContent("[");

    for (int i = 0; i < num; i++) {

```

```

        if (i > 0)
            server.sendContent(",");
        server.sendContent(lines[(start + i) % count]);
    }

    server.sendContent("]");
}

void handleApiFiles() {
    if (!fsReady) {
        server.send(503, "application/json", "{\"error\":\"FS not ready\"}");
        return;
    }

    Dir dir = LittleFS.openDir("/");
    server.setContentLength(CONTENT_LENGTH_UNKNOWN);
    server.send(200, "application/json", "");
    server.sendContent("[");

    bool first = true;
    while (dir.next()) {
        String name = dir.fileName();
        if (name.endsWith(".txt")) {
            if (!first)
                server.sendContent(",");
            server.sendContent "\"" + name + "\"");
            first = false;
        }
    }

    server.sendContent("]");
}

void handleRoot() {
    server.setHeader("Location", "/control");
    server.send(303);
}

void handleLogin() {
    if (checkSession()) {
        server.setHeader("Location", "/control" + sessionParam());
        server.send(303);
        return;
    }

    String html = R"(<!DOCTYPE html><html>
<head>
<meta charset='UTF-8'>
<title>Авторизация</title>
<style>
    body { font-family: Arial; max-width: 300px; margin: 50px auto; }
    .login-form { padding: 20px; border: 1px solid #ccc; border-radius: 5px; }
    input { width: 100%; margin: 5px 0; padding: 8px; box-sizing: border-box; }
    button { width: 100%; padding: 10px; background: #4CAF50; color: white;
border: none; cursor: pointer; }
</style>
</head>
<body>
<div class='login-form'>
    <h2>Авторизация</h2>

```

```

        <form action='/loginps' method='post'>
            <input type='text' name='login' placeholder='Логин' required>
            <input type='password' name='password' placeholder='Пароль' required>
            <button type='submit'>Войти</button>
        </form>
    </div>
</body>
</html>)<";

    server.send(200, "text/html", html);
}

void handleLoginPost() {
    String login = server.arg("login");
    String pass = server.arg("password");

    if (login == adminLogin && pass == adminPassword) {
        activeSession = generateSession();
        server.setHeader("Set-Cookie", "session=" + activeSession + "; path=/; max-age=3600");
        server.setHeader("Location", "/control" + sessionParam());
        server.send(303);
    } else {
        server.send(200, "text/html",
            "<script>alert('Неверный логин или пароль');window.location.href='/login';</script>");
    }
}

void handleLogout() {
    activeSession = "";
    server.setHeader("Set-Cookie", "session=; expires=Thu, 01 Jan 1970 00:00:00 GMT; path=/");
    server.setHeader("Location", "/login");
    server.send(303);
}

void handleData() {
    float temp = tempInt + (float)tempFrac / 100.0;

    StaticJsonDocument<200> doc;
    doc["temp"] = temp;
    doc["humidity"] = humidity;
    doc["soilmoisture"] = soilMoisture;
    doc["req_soilmoisture"] = requiredMoisture;

    String json;
    serializeJson(doc, json);
    server.send(200, "application/json", json);
}

void handleSendCommand() {
    if (!checkSession()) {
        server.send(401, "text/plain", "Unauthorized");
        return;
    }

    if (!server.hasArg("soilmoisture")) {
        server.send(400, "text/plain", "Missing soilmoisture");
        return;
    }
}

```

```

    requiredMoisture = (byte)server.arg("soilmoisture").toInt();
    sendCommand(CMD_REQUIRED, requiredMoisture);

    server.sendHeader("Location", "/control" + sessionParam());
    server.send(303);
}

void handleControl() {
    server.send(200, "text/html", buildHTML());
}

void handleNotFound() {
    server.send(404, "text/plain", "Not found");
}

String buildHTML() {
    bool authorized = checkSession();

    String html = R"(<!DOCTYPE html><html>
<head>
    <meta charset='UTF-8'>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Автополив</title>
    <style>
        body { font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto;
padding: 20px; text-align: center; }
        .card { background: #f5f5f5; border-radius: 10px; padding: 20px; margin:
10px 0; box-shadow: 0 2px 5px rgba(0,0,0,0.1); }
        h1 { color: #333; }
        .value { font-size: 24px; color: #2c3e50; font-weight: 700; }
        input[type=range] { width: 100%; }
        .nav { text-align: right; padding: 10px; }
        .nav a { text-decoration: none; margin-left: 10px; }
    </style>
</head>
<body>";

    if (authorized) {
        html += "<div class='nav'><a href='/logout' + sessionParam() + '"
style='color:#e74c3c'>Выйти</a></div>";
    } else {
        html += "<div class='nav'><a href='/login'
style='color:#3498db'>Войти</a></div>";
    }

    html += R"(
    <h1>Автополив</h1>
    <div class='card'>
        <h2>Температура воздуха</h2>
        <p class='value' id="temp">-- C</p>
    </div>
    <div class='card'>
        <h2>Влажность воздуха</h2>
        <p class='value' id="humidity">-- %</p>
    </div>
    <div class='card'>
        <h2>Влажность почвы</h2>
        <p class='value' id="soilmoisture">-- %</p>
    </div>

```

```

<div class='card'>
  <h2>Порог полива</h2>
  <p class='value' id="req_soilmoisture">-- %</p>
</div>";

if (authorized) {
  html += "<div class='card'><h2>Установить порог</h2>";
  html += "<form action='/sendCommand' + sessionParam() + '' method='POST'>";
  html += "<input type='range' id='target' name='soilmoisture' min='0'";
  html += "max='100' value='" + String(requiredMoisture) + "'>";
  html += "<p id='targetValue'>" + String(requiredMoisture) + "%</p>";
  html += "<input type='submit' value='Установить'></form></div>";
  html += R"(<script>
document.getElementById('target').oninput = function() {
  document.getElementById('targetValue').textContent = this.value + '%';
};
</script>)" ;
}

html += R"(<script>
setInterval(function() {
  fetch('/data')
    .then(function(r) { return r.json(); })
    .then(function(d) {
      document.getElementById('temp').textContent = d.temp + ' C';
      document.getElementById('humidity').textContent = d.humidity + ' %';
      document.getElementById('soilmoisture').textContent = d.soilmoisture + '
%';
      document.getElementById('req_soilmoisture').textContent =
d.req_soilmoisture + ' %';
    });
}, 2000);
</script>
</body></html>)" ;

return html;
}

void setup() {
  Serial.begin(9600);
  delay(10);

  // WiFi
  WiFi.config(local_ip, gateway, subnet, dns);
  WiFi.begin(ssid, wifiPassword);
  while (WiFi.status() != WL_CONNECTED)
    delay(500);

  // NTP - ждем синхронизации
  configTime(UTC_OFFSET, 0, NTP_SERVER);
  while (time(nullptr) < 100000)
    delay(100);

  // LittleFS
  fsReady = LittleFS.begin();

  // Первая запись сразу после старта
  if (fsReady)
    logData();
}

```

```

// HTTP routes
server.on("/", handleRoot);
server.on("/login", handleLogin);
server.on("/loginps", HTTP_POST, handleLoginPost);
server.on("/logout", handleLogout);
server.on("/data", handleData);
server.on("/sendCommand", HTTP_POST, handleSendCommand);
server.on("/control", handleControl);
server.on("/api/history", handleApiHistory);
server.on("/api/history/latest", handleApiLatest);
server.on("/api/files", handleApiFiles);
server.onNotFound(handleNotFound);

server.begin();
}

void loop() {
  handleUART();
  server.handleClient();

  unsigned long now = millis();
  if (now - lastLogTime >= LOG_INTERVAL_MS) {
    lastLogTime = now;
    logData();
  }
}

```

Приложение Б
(обязательное)

Исходный код мобильного приложения

Листинг Б.1 – Файл app/(tabs)/index.tsx

```
import React, { useState, useEffect } from 'react';
import { View, ScrollView, StyleSheet, TouchableOpacity, LayoutAnimation,
ActivityIndicator } from 'react-native';
import StyledText from '@components/StyledText';
import SensorCard from '@components/SensorCard';
import ChartView from '@components/ChartView';
import DatePicker from '@components/DatePicker';
import { COLORS } from '@constants/ui';
import { useSensorData } from '@hooks/useSensorData';
import { useAvailableDates } from '@hooks/useAvailableDates';
import { useHistory } from '@hooks/useHistory';
import { useDailySummary } from '@hooks/useDailySummary';
import { getTargetValues, TargetValues } from '@services/storage';

type SensorType = 'temp' | 'humidity' | 'soil';
type Mode = 'detail' | 'daily';

export default function DashboardScreen() {
  const { data, error } = useSensorData();
  const [selectedSensor, setSelectedSensor] = useState<SensorType | null>(null);
  const [mode, setMode] = useState<Mode>('detail');
  const [selectedDate, setSelectedDate] = useState<string | null>(null);

  const { dates, loading: datesLoading } = useAvailableDates();
  const { data: dailyData, loading: dailyLoading } = useDailySummary();
  const { data: historyData, loading: historyLoading } =
useHistory(selectedDate);
  const [targets, setTargets] = useState<TargetValues>({ targetSoilMoisture: 50,
targetTemp: 25, targetHumidity: 50 });

  useEffect(() => {
    getTargetValues().then(setTargets);
  }, []);

  const validDates = dailyData.map((d) => d.date);
  const displayDates = validDates.length > 0 ? validDates : dates;

  useEffect(() => {
    if (displayDates.length > 0 && !selectedDate) {
      setSelectedDate(displayDates[displayDates.length - 1]);
    }
  }, [displayDates.length]);

  function toggleSensor(sensor: SensorType) {
    LayoutAnimation.configureNext(LayoutAnimation.Presets.easeInEaseOut);
    setSelectedSensor(selectedSensor === sensor ? null : sensor);
  }

  function getChartData() {
    if (!selectedSensor) return [];
  }
}
```

```

    if (mode === 'daily') {
      return dailyData.map((d) => ({
        value: selectedSensor === 'temp' ? d.avgTemp : selectedSensor ===
        'humidity' ? d.avgHumidity : d.avgSoil,
        label: d.date,
      }));
    }

    const now = new Date();
    const todayStr = `${String(now.getDate()).padStart(2, '0')}-${
    String(now.getMonth() + 1).padStart(2, '0')}`;
    const isToday = selectedDate === todayStr;

    const currentValue = data
      ? selectedSensor === 'temp' ? data.temp : selectedSensor === 'humidity' ?
      data.humidity : data.soilmoisture
      : null;

    const step = Math.max(Math.floor(historyData.length / 5), 1);
    const points = historyData.map((entry, i) => ({
      value: selectedSensor === 'temp' ? entry.temp : selectedSensor ===
      'humidity' ? entry.humidity : entry.soil,
      label: (i % step === 0 && i <= historyData.length - step) ?
      entry.time.slice(0, 5) : '',
    }));

    if (isToday && currentValue !== null) {
      points.push({ value: currentValue, label: undefined });
    }

    return points;
  }

  function getChartColor() {
    if (selectedSensor === 'temp') return COLORS.ACCENT_ORANGE;
    if (selectedSensor === 'humidity') return COLORS.ACCENT_BLUE;
    return COLORS.ACCENT_GREEN;
  }

  function getChartUnit() {
    return selectedSensor === 'temp' ? '°C' : '%';
  }

  function getReferenceValue() {
    if (selectedSensor === 'soil') return data?.req_soilmoisture ??
    targets.targetSoilMoisture;
    if (selectedSensor === 'temp') return targets.targetTemp;
    return targets.targetHumidity;
  }

  function getChartTitle() {
    if (selectedSensor === 'temp') return 'Температура';
    if (selectedSensor === 'humidity') return 'Влажность воздуха';
  }

```

```

    return 'Влажность почвы';
}

const chartLoading = mode === 'daily' ? dailyLoading : historyLoading;

return (
  <ScrollView style={styles.container} contentContainerStyle={styles.content}>
    <StyledText style={styles.title}>Датчики</StyledText>

    {error && (
      <View style={styles.errorCard}>
        <StyledText style={styles.errorText}>{error}</StyledText>
      </View>
    )}

    <View style={styles.grid}>
      <TouchableOpacity style={styles.cardWrapper} onPress={() =>
toggleSensor('temp')}>
        <SensorCard
          label="Температура"
          value={data ? String(data.temp) : '--'}
          unit="°C"
          color={COLORS.ACCENT_ORANGE}
          active={selectedSensor === 'temp'}
        />
      </TouchableOpacity>
      <TouchableOpacity style={styles.cardWrapper} onPress={() =>
toggleSensor('humidity')}>
        <SensorCard
          label="Влажность воздуха"
          value={data ? String(data.humidity) : '--'}
          unit="%"
          color={COLORS.ACCENT_BLUE}
          active={selectedSensor === 'humidity'}
        />
      </TouchableOpacity>
      <TouchableOpacity style={styles.cardWrapper} onPress={() =>
toggleSensor('soil')}>
        <SensorCard
          label="Влажность почвы"
          value={data ? String(data.soilmoisture) : '--'}
          unit="%"
          color={COLORS.ACCENT_GREEN}
          active={selectedSensor === 'soil'}
        />
      </TouchableOpacity>
      <View style={styles.cardWrapper}>
        <SensorCard
          label="Порог полива"
          value={data ? String(data.req_soilmoisture) : '--'}
          unit="%"
          color={COLORS.PRIMARY_TEXT}
        />
      </View>
    </View>
  </ScrollView>
);

```

```

</View>

{selectedSensor && (
  <View style={styles.chartSection}>
    <View style={styles.toggle}>
      <TouchableOpacity
        style={[styles.toggleButton, mode === 'detail' &&
styles.toggleActive]}
        onPress={() => setMode('detail')}
      >
        <StyledText style={[styles.toggleText, mode === 'detail' &&
styles.toggleTextActive]}>
          За день
        </StyledText>
      </TouchableOpacity>
      <TouchableOpacity
        style={[styles.toggleButton, mode === 'daily' &&
styles.toggleActive]}
        onPress={() => setMode('daily')}
      >
        <StyledText style={[styles.toggleText, mode === 'daily' &&
styles.toggleTextActive]}>
          По дням
        </StyledText>
      </TouchableOpacity>
    </View>

    {mode === 'detail' && (
      (datesLoading || dailyLoading) ? (
        <ActivityIndicator color={COLORS.ACCENT_GREEN} />
      ) : (
        <DatePicker dates={displayDates} selected={selectedDate}
onSelect={setSelectedDate} />
      )
    )}

    {chartLoading ? (
      <ActivityIndicator color={COLORS.ACCENT_GREEN} style={{ marginTop:
16 }} />
    ) : (
      <ChartView
        title={getChartTitle()}
        data={getChartData()}
        color={getChartColor()}
        unit={getChartUnit()}
        referenceValue={getReferenceValue()}
      />
    )}
  </View>
)}
</ScrollView>
);
}

```

```

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: COLORS.PRIMARY_BACKGROUND,
  },
  content: {
    paddingHorizontal: 20,
    paddingTop: 80,
    paddingBottom: 40,
  },
  title: {
    fontSize: 24,
    fontWeight: '700',
    marginBottom: 24,
  },
  grid: {
    flexDirection: 'row',
    flexWrap: 'wrap',
    gap: 12,
    marginBottom: 20,
  },
  cardWrapper: {
    flex: 1,
    minWidth: '45%',
  },
  errorCard: {
    backgroundColor: COLORS.CARD_BACKGROUND,
    borderRadius: 8,
    padding: 12,
    marginBottom: 16,
    borderWidth: 1,
    borderColor: COLORS.ACCENT_RED,
  },
  errorText: {
    color: COLORS.ACCENT_RED,
    fontSize: 14,
    textAlign: 'center',
  },
  chartSection: {
    marginTop: 4,
  },
  toggle: {
    flexDirection: 'row',
    backgroundColor: COLORS.CARD_BACKGROUND,
    borderRadius: 10,
    padding: 4,
    marginBottom: 12,
  },
  toggleButton: {
    flex: 1,
    paddingVertical: 10,
    alignItems: 'center',
    borderRadius: 8,
  },
  toggleActive: {

```

```

        backgroundColor: COLORS.ACCENT_GREEN,
      },
      toggleText: {
        fontSize: 14,
        fontWeight: '600',
        color: COLORS.SECONDARY_TEXT,
      },
      toggleTextActive: {
        color: COLORS.PRIMARY_TEXT,
      },
    },
  ));

```

Листинг Б.2 – Файл app/login.tsx

```

import React, { useState } from 'react';
import { View, TextInput, TouchableOpacity, StyleSheet, ActivityIndicator } from
'react-native';
import { useRouter } from 'expo-router';
import StyledText from '@components/StyledText';
import { COLORS } from '@constants/ui';
import { useAuth } from '@context/AuthContext';

export default function LoginScreen() {
  const { login } = useAuth();
  const router = useRouter();
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);

  async function handleLogin() {
    if (!username || !password) {
      setError('Введите логин и пароль');
      return;
    }
    setError('');
    setLoading(true);
    try {
      await login(username, password);
    } catch (e: any) {
      setError(e.message || 'Ошибка авторизации');
    } finally {
      setLoading(false);
    }
  }

  return (
    <View style={styles.container}>
      <View style={styles.card}>
        <StyledText style={styles.title}>Авторизация</StyledText>

        <TextInput

```

```

        style={styles.input}
        placeholder="Имя"
        placeholderTextColor={COLORS.SECONDARY_TEXT}
        value={username}
        onChangeText={setUsername}
        autoCapitalize="none"
        autoCorrect={false}
      />

      <TextInput
        style={styles.input}
        placeholder="Пароль"
        placeholderTextColor={COLORS.SECONDARY_TEXT}
        value={password}
        onChangeText={setPassword}
        secureTextEntry
      />

      {error ? <StyledText style={styles.error}>{error}</StyledText> : null}

      <TouchableOpacity style={styles.button} onPress={handleLogin}
disabled={loading}>
        {loading ? (
          <ActivityIndicator color="#fff" />
        ) : (
          <StyledText style={styles.buttonText}>Войти</StyledText>
        )}
      </TouchableOpacity>

      <TouchableOpacity style={styles.cancelLink} onPress={() =>
router.back()}>
        <StyledText style={styles.cancelLinkText}>Отмена</StyledText>
      </TouchableOpacity>
    </View>
  </View>
);
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: COLORS.PRIMARY_BACKGROUND,
    justifyContent: 'center',
    paddingHorizontal: 24,
  },
  card: {
    backgroundColor: COLORS.CARD_BACKGROUND,
    borderRadius: 12,
    padding: 24,
    borderWidth: 1,
    borderColor: COLORS.CARD_BORDER,
  },
  title: {
    fontSize: 22,

```

```

    fontWeight: '700',
    textAlign: 'center',
    marginBottom: 24,
  },
  input: {
    backgroundColor: COLORS.INPUT_BACKGROUND,
    borderWidth: 1,
    borderColor: COLORS.INPUT_BORDER,
    borderRadius: 8,
    padding: 12,
    color: COLORS.PRIMARY_TEXT,
    fontSize: 16,
    marginBottom: 12,
  },
  error: {
    color: COLORS.ACCENT_RED,
    fontSize: 14,
    marginBottom: 12,
    textAlign: 'center',
  },
  button: {
    backgroundColor: COLORS.ACCENT_GREEN,
    borderRadius: 8,
    padding: 14,
    alignItems: 'center',
    marginTop: 8,
  },
  buttonText: {
    fontSize: 16,
    fontWeight: '600',
  },
  cancelLink: {
    marginTop: 16,
    alignItems: 'center',
  },
  cancelLinkText: {
    color: COLORS.SECONDARY_TEXT,
    fontSize: 14,
  },
},
));

```

Листинг Б.3 – Файл components/ChartView.tsx

```

import React from 'react';
import { View, StyleSheet, Dimensions } from 'react-native';
import { LineChart } from 'react-native-gifted-charts';
import StyledText from '@components/StyledText';
import { COLORS } from '@constants/ui';

interface ChartDataPoint {
  value: number;
  label?: string;
  dataPointText?: string;
}

```

```

interface ChartViewProps {
  title: string;
  data: ChartDataPoint[];
  color: string;
  unit: string;
  referenceValue?: number;
}

const screenWidth = Dimensions.get('window').width;

const CHART_HEIGHT = 150;
const Y_EXTRA_TOP = 10;

export default function ChartView({ title, data, color, unit, referenceValue }:
ChartViewProps) {
  if (data.length === 0) {
    return (
      <View style={styles.card}>
        <StyledText style={styles.title}>{title}</StyledText>
        <StyledText style={styles.empty}>Нет данных</StyledText>
      </View>
    );
  }

  const chartData = data;

  const maxVal = Math.max(...chartData.map((d) => d.value));
  const step = 3;
  const yMax = Math.ceil(maxVal / step) * step;
  const yMin = 0;
  const range = yMax - yMin;

  function refLabelTop(): number {
    if (referenceValue === undefined || range === 0) return 0;
    const ratio = (yMax - referenceValue) / range;
    return Y_EXTRA_TOP + ratio * CHART_HEIGHT - 8;
  }

  const chartWidth = screenWidth - 120;
  const spacing = (chartWidth - 20) / Math.max(chartData.length - 1, 1);

  return (
    <View style={styles.card}>
      <StyledText style={styles.title}>{title}</StyledText>
      <View style={styles.chartContainer}>
        <LineChart
          data={chartData}
          width={chartWidth}
          height={150}
          color={color}
          dataPointsColor={color}
          thickness={2}
          dataPointsRadius={3}

```

```

    spacing={spacing}
    paddingRight={30}
    endSpacing={30}
    hideRules
    showReferenceLine1={referenceValue !== undefined}
    referenceLine1Position={referenceValue}
    referenceLine1Config={referenceValue !== undefined ? {
      color,
      thickness: 1,
      type: 'dashed',
      dashWidth: 4,
      dashGap: 4,
    } : undefined}
    yAxisColor={COLORS.CARD_BORDER}
    xAxisColor={COLORS.CARD_BORDER}
    yAxisTextStyle={styles.axisText}
    xAxisLabelTextStyle={styles.xAxisText}
    labelsExtraHeight={20}
    pointerConfig={{
      pointerStripColor: color,
      pointerStripWidth: 1,
      pointerColor: color,
      radius: 5,
      pointerLabelWidth: 80,
      pointerLabelHeight: 30,
      activatePointersOnLongPress: false,
      autoAdjustPointerLabelPosition: true,
      pointerLabelComponent: (items: any[]) => (
        <View style={styles.tooltip}>
          <StyledText style={styles.tooltipText}>
            {items[0]?.value}{unit}
          </StyledText>
        </View>
      ),
    }}
    curved
    startFillColor={color}
    startOpacity={0.15}
    endOpacity={0}
    areaChart
    scrollToEnd={false}
    disableScroll
  />
  {referenceValue !== undefined && (
    <StyledText style={[[styles.referenceLabel, { color, top: refLabelTop()
    ]}}>
      {referenceValue}{unit}
    </StyledText>
  )}
</View>
</View>
);
}

const styles = StyleSheet.create({

```

```

card: {
  backgroundColor: COLORS.CARD_BACKGROUND,
  borderRadius: 12,
  padding: 16,
  borderWidth: 1,
  borderColor: COLORS.CARD_BORDER,
  marginBottom: 16,
},
title: {
  fontSize: 14,
  color: COLORS.SECONDARY_TEXT,
  marginBottom: 12,
},
empty: {
  color: COLORS.SECONDARY_TEXT,
  fontSize: 14,
  textAlign: 'center',
  paddingVertical: 20,
},
axisText: {
  color: COLORS.SECONDARY_TEXT,
  fontSize: 10,
},
xAxisText: {
  color: COLORS.SECONDARY_TEXT,
  fontSize: 9,
  width: 40,
  textAlign: 'center',
},
chartContainer: {
  position: 'relative',
},
referenceLabel: {
  position: 'absolute',
  right: 0,
  fontSize: 10,
},
tooltip: {
  backgroundColor: COLORS.PRIMARY_BACKGROUND,
  borderRadius: 4,
  padding: 4,
  borderWidth: 1,
  borderColor: COLORS.CARD_BORDER,
},
tooltipText: {
  fontSize: 12,
  textAlign: 'center',
},
});

```

Листинг Б.4 – Файл context/AuthContext.tsx

```

import React, { createContext, useContext, useEffect, useState } from 'react';
import { login as authLogin, logout as authLogout } from '@services/auth';
import { getSessionToken, setSessionToken, clearSessionToken } from
'@services/storage';

```

```

import { useSettings } from '../SettingsContext';

interface AuthContextType {
  session: string | null;
  isAuthenticated: boolean;
  isLoading: boolean;
  login: (username: string, password: string) => Promise<void>;
  logout: () => Promise<void>;
}

const AuthContext = createContext<AuthContextType>({
  session: null,
  isAuthenticated: false,
  isLoading: true,
  login: async () => {},
  logout: async () => {},
});

export function AuthProvider({ children }: { children: React.ReactNode }) {
  const { deviceIp } = useSettings();
  const [session, setSession] = useState<string | null>(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    getSessionToken().then((token) => {
      setSession(token);
      setIsLoading(false);
    });
  }, []);

  async function login(username: string, password: string) {
    if (!deviceIp) throw new Error('IP устройства не настроен');
    const baseUrl = `http://${deviceIp}`;
    const token = await authLogin(baseUrl, username, password);
    await setSessionToken(token);
    setSession(token);
  }

  async function logout() {
    if (deviceIp && session) {
      try {
        await authLogout(`http://${deviceIp}`, session);
      } catch {}
    }
    await clearSessionToken();
    setSession(null);
  }

  return (
    <AuthContext.Provider
      value={{
        session,
        isAuthenticated: session !== null,

```

```

        isLoading,
        login,
        logout,
      }}>
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  return useContext(AuthContext);
}

```

Листинг Б.5 – Файл hooks/useSensorData.ts

```

import { useEffect, useRef, useState } from 'react';
import { createApiClient, SensorData } from '@services/api';
import { useSettings } from '@context/SettingsContext';
import { useAuth } from '@context/AuthContext';

export function useSensorData() {
  const { deviceIp } = useSettings();
  const { session } = useAuth();
  const [data, setData] = useState<SensorData | null>(null);
  const [error, setError] = useState<string | null>(null);
  const intervalRef = useRef<ReturnType<typeof setInterval> | null>(null);

  useEffect(() => {
    if (!deviceIp) return;

    const api = createApiClient(`http://${deviceIp}`, session);

    async function poll() {
      try {
        const result = await api.getSensorData();
        setData(result);
        setError(null);
      } catch (e: any) {
        setError(e.message || 'Ошибка подключения');
      }
    }

    poll();
    intervalRef.current = setInterval(poll, 2000);

    return () => {
      if (intervalRef.current) clearInterval(intervalRef.current);
    };
  }, [deviceIp, session]);

  return { data, error };
}

```

Листинг Б.6 – Файл services/api.ts

```
import { ENDPOINTS } from '@/constants/api';

export interface SensorData {
  temp: number;
  humidity: number;
  soilmoisture: number;
  req_soilmoisture: number;
}

export interface HistoryEntry {
  time: string;
  temp: number;
  humidity: number;
  soil: number;
}

export function createApiClient(baseUrl: string, session: string | null) {
  function buildUrl(path: string, params?: Record<string, string>): string {
    let url = `${baseUrl}${path}`;
    const searchParams = new URLSearchParams();
    if (session) searchParams.set('session', session);
    if (params) Object.entries(params).forEach(([k, v]) => searchParams.set(k, v));
    const qs = searchParams.toString();
    if (qs) url += `?${qs}`;
    return url;
  }

  return {
    getSensorData: async (): Promise<SensorData> => {
      const res = await fetch(buildUrl(ENDPOINTS.DATA));
      if (!res.ok) throw new Error(`HTTP ${res.status}`);
      return res.json();
    },

    getHistory: async (date: string): Promise<HistoryEntry[]> => {
      const res = await fetch(buildUrl(ENDPOINTS.HISTORY, { date }));
      if (!res.ok) throw new Error(`HTTP ${res.status}`);
      return res.json();
    },

    getLatest: async (count: number): Promise<HistoryEntry[]> => {
      const res = await fetch(buildUrl(ENDPOINTS.LATEST, { count: String(count) }));
      if (!res.ok) throw new Error(`HTTP ${res.status}`);
      return res.json();
    },

    getFiles: async (): Promise<string[]> => {
```

```

    const res = await fetch(buildUrl(ENDPOINTS.FILES));
    if (!res.ok) throw new Error(`HTTP ${res.status}`);
    return res.json();
  },

  setThreshold: async (value: number): Promise<void> => {
    await fetch(buildUrl(ENDPOINTS.SEND_COMMAND), {
      method: 'POST',
      headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
      body: `soilmoisture=${value}`,
    });
  },
};
}

```

Приложение В

(обязательное)

Исходный код библиотеки DS3231 и контроллера Arduino

Листинг В.1 – Файл DS3231.h

```
#pragma once
#include <Arduino.h>
#include <Wire.h>

#define DS3231_ADDR 0x68
#define REGISTER_TIME 0x00
#define REGISTER_TEMPERATURE 0x11

class DS3231_Class {
public:
    /**
     * @brief Получает единственный экземпляр класса DS3231_Class
     * @return Ссылка на экземпляр DS3231_Class
     */

    DS3231_Class() { Wire.begin(); } // конструктор
    /**
     * @brief Устанавливает дату и время в DS3231_Class
     * @param sec Секунды (0-59)
     * @param min Минуты (0-59)
     * @param hour Часы (0-23)
     * @param weekday День недели (1-7, 1=понедельник)
     * @param day День месяца (1-31)
     * @param month Месяц (1-12)
     * @param year Год (0-99, 00=2000)
     */
    void setDateTime(byte sec, byte min, byte hour,
        byte weekday, byte day, byte month, byte year);

    /**
     * @brief Получает текущую дату и время в виде строки
     * @return Строка с датой и временем в формате "DD.MM.YYYY HH:MM:SS"
     */
    String getDateTime();

    /**
     * @brief Получает температуру с датчика DS3231_Class
     * @return Температура в градусах Цельсия
     */
    float getTemperature();

private:

    /**
     * @brief Получает сырые данные времени и даты
     * @param dateTime Указатель на массив для хранения данных (7 байт)
     */
    void getDateTimeRaw(byte* dateTime);

    /**
     * @brief Получает сырые данные температуры
     * @param temp Указатель на массив для хранения данных (2 байта)
     */
    void getTemperatureRaw(byte* temp);
};

// Глобальный экземпляр для удобства использования
```

```
extern DS3231_Class DS3231;
```

Листинг В.2 – Файл BasicUsage.ino

```
#include <DS3231.h>
```

```
void setup() {  
    Serial.begin(9600);  
    // Установка времени (опционально)  
    DS3231.setDateTime(0,40,18,7,15,6,25);  
    // Получение и вывод даты/времени  
    String datetime = DS3231.getDateTime();  
    Serial.print("Дата и время: ");  
    Serial.println(datetime);  
  
    // Получение и вывод температуры  
    float temp = DS3231.getTemperature();  
    Serial.print("Температура: ");  
    Serial.print(temp);  
    Serial.println(" °C");  
}  
  
void loop() {  
    // Пример периодического обновления (раз в секунду)  
    String currentTime = DS3231.getDateTime();  
    Serial.println(currentTime);  
    delay(1000);  
}
```

Приложение Г
(рекомендуемое)
Результаты тестирования серверной части

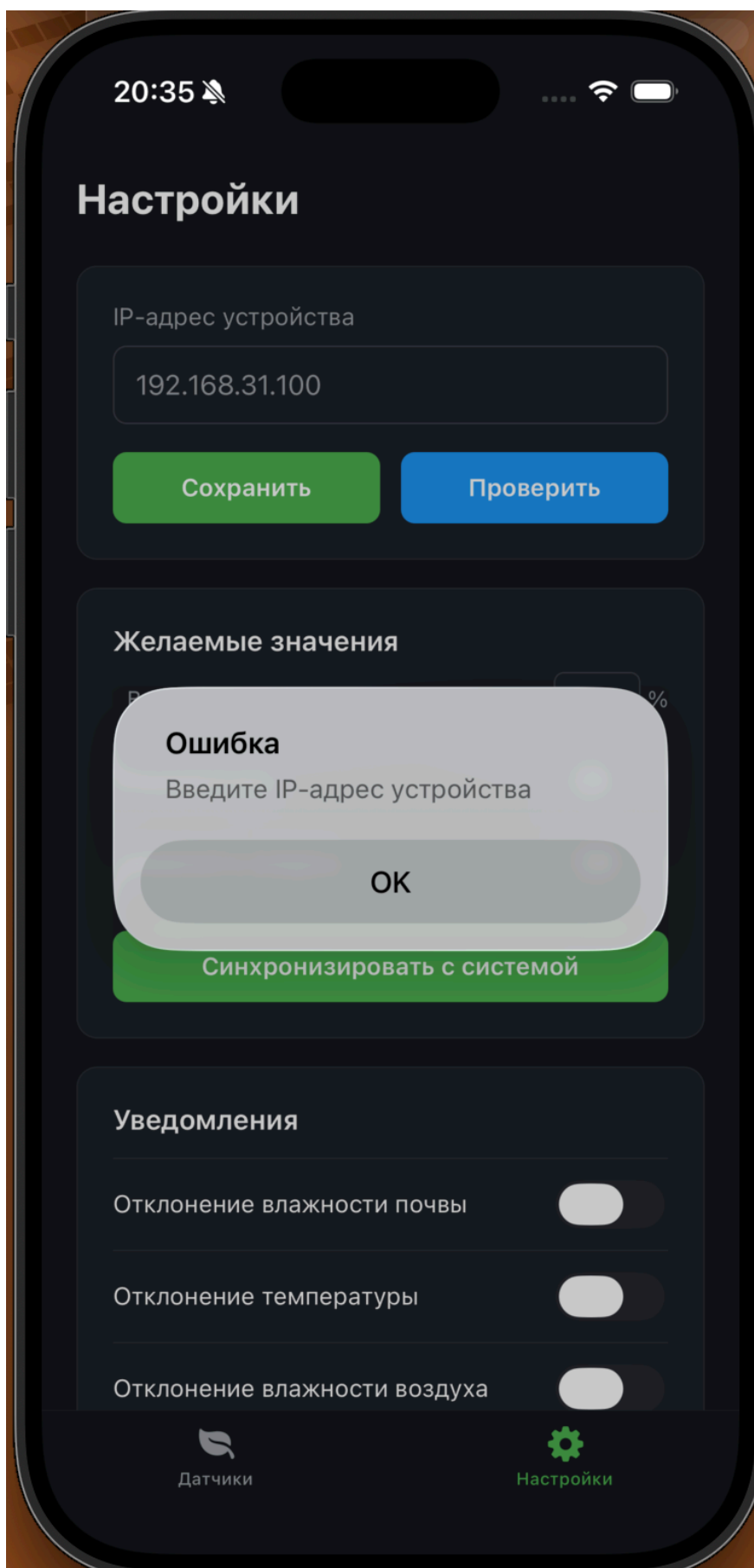


Рисунок Г.1 – Ввод пустого ip

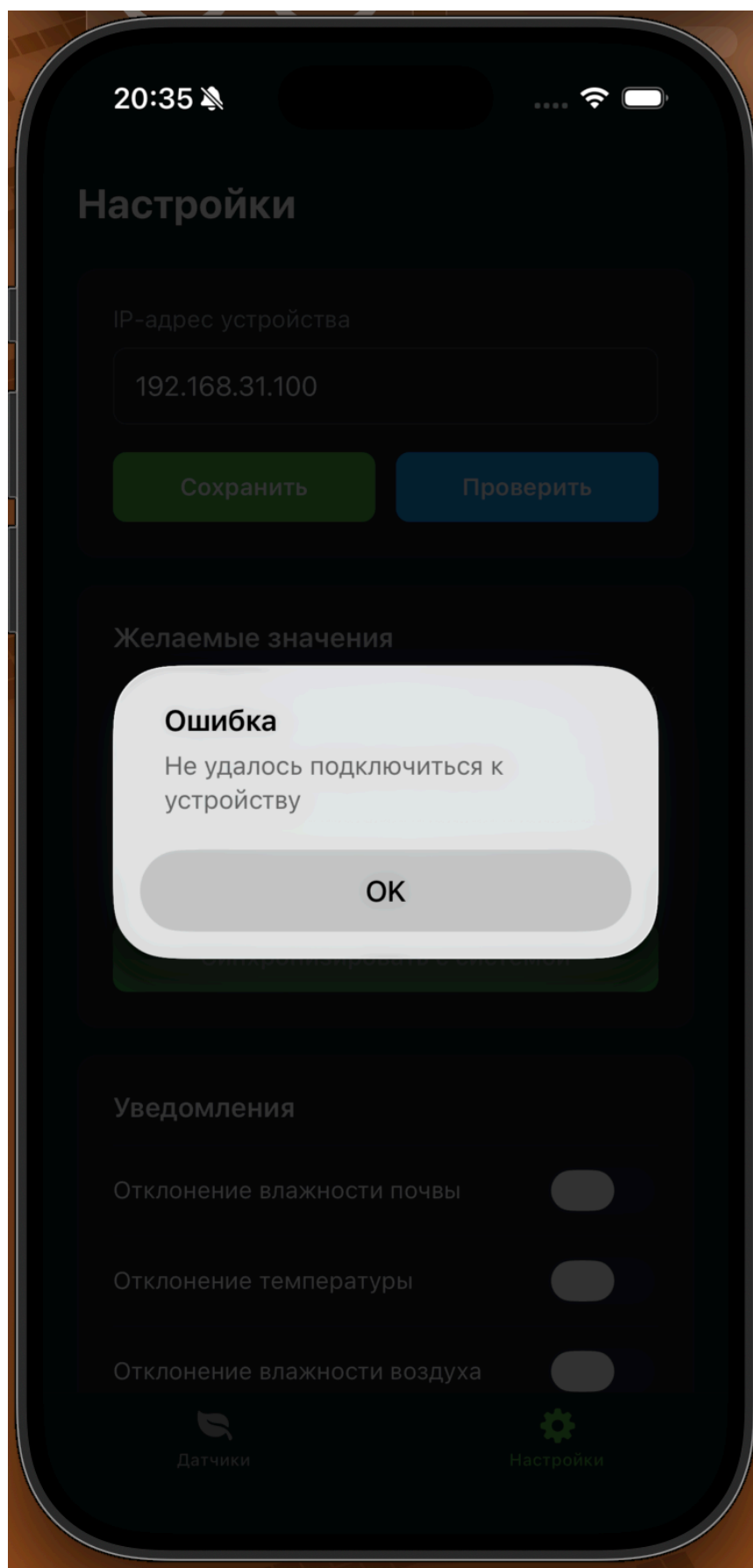


Рисунок Г.2 – Ввод недоступного ip

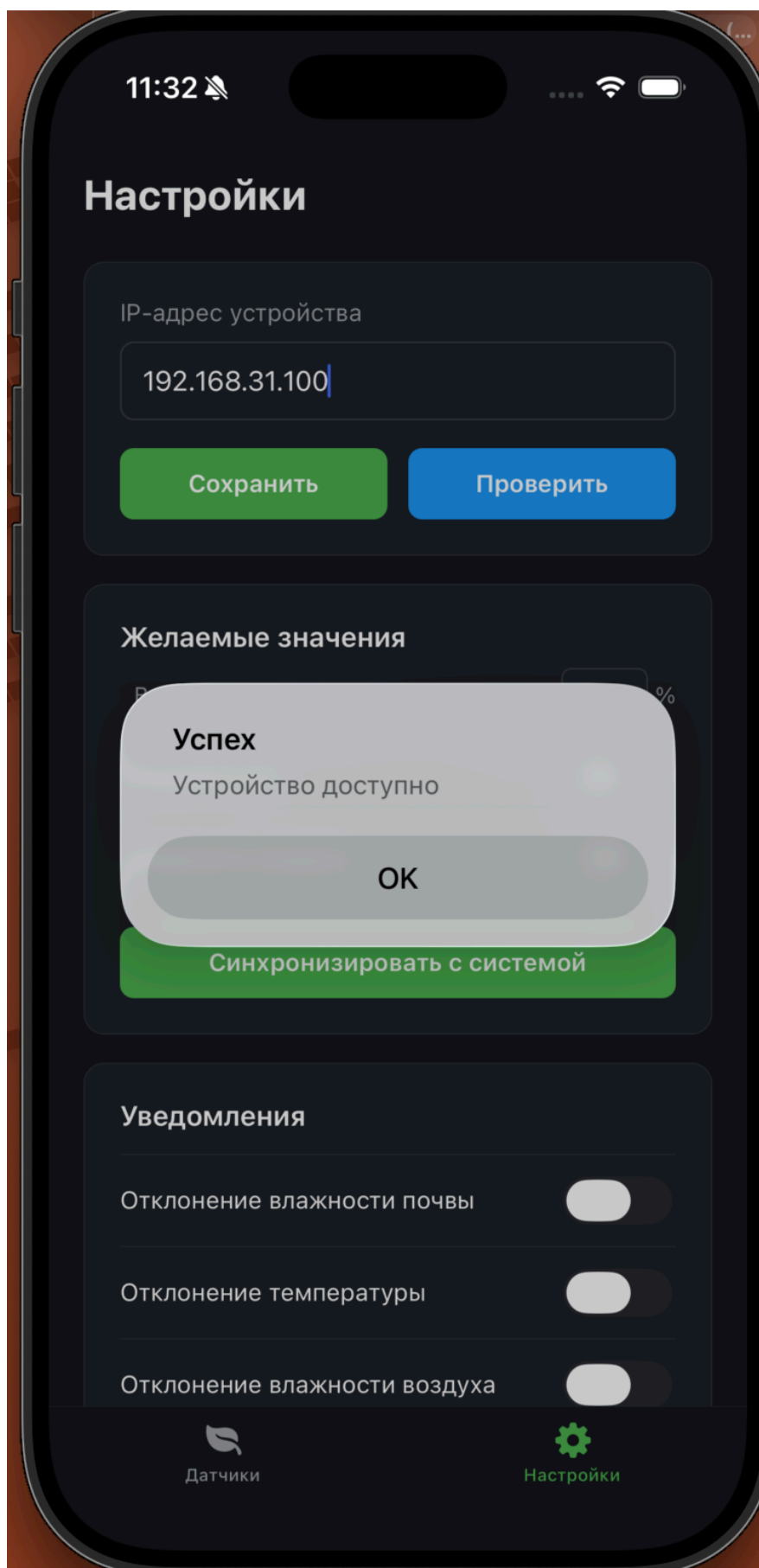


Рисунок Г.3 – Успешное подключение к устройству

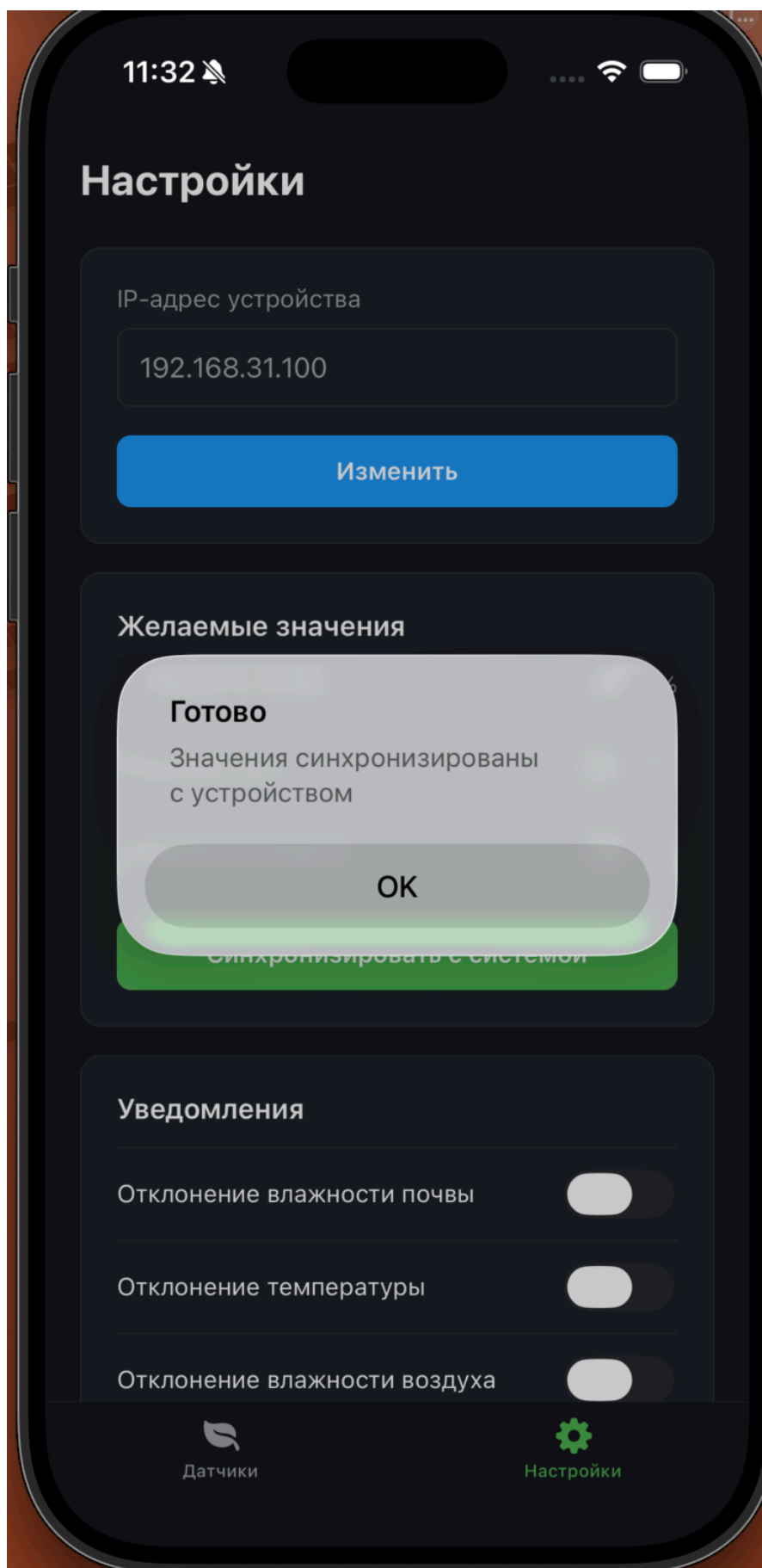


Рисунок Г.4 – Сохранение пороговых значений

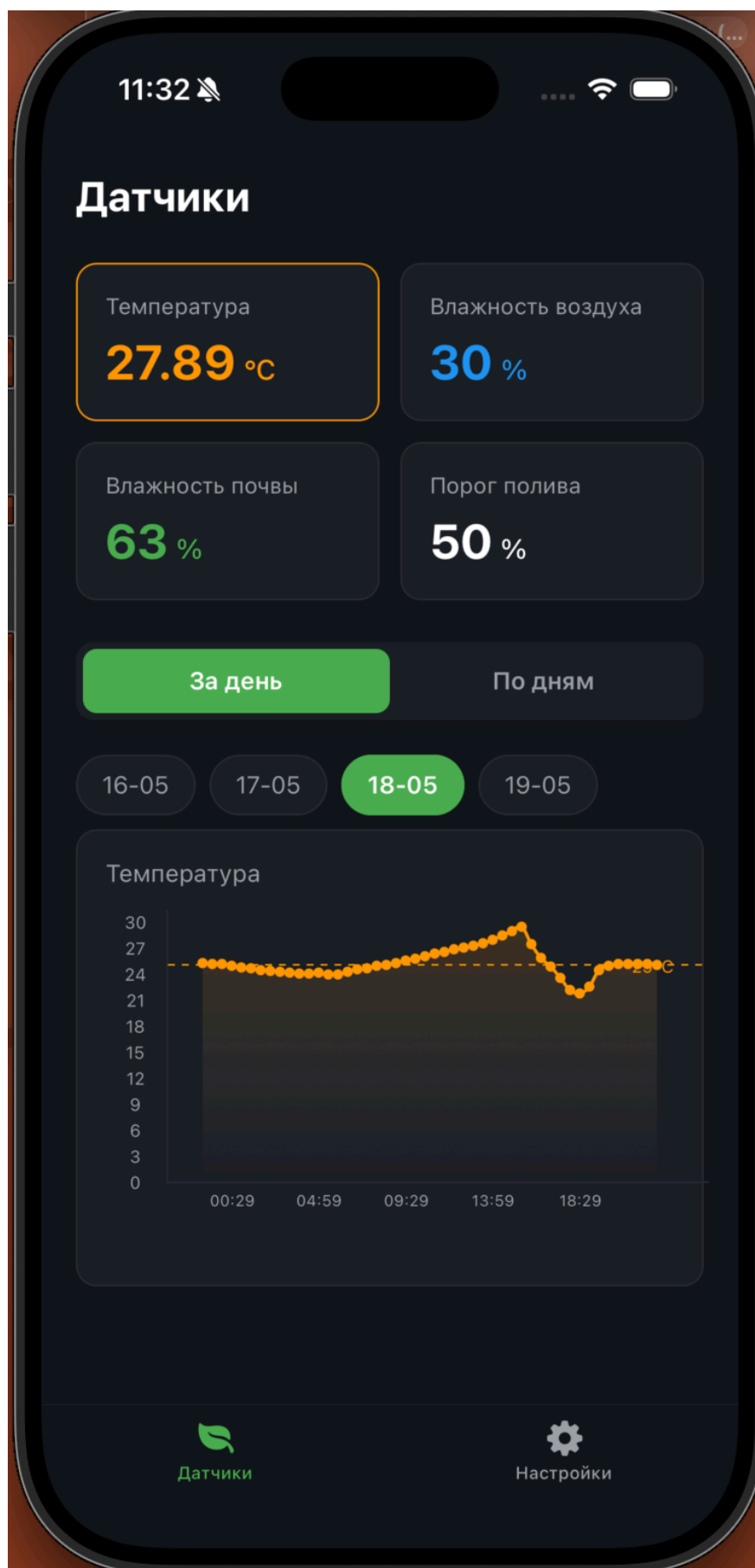


Рисунок Г.5 – Отображение текущих показаний при успешном подключении

Приложение Д
(рекомендуемое)
Исходный код функций тестирования

```

#include <gtest/gtest.h>
#include <gmock/gmock.h>

using ::testing::_;
using ::testing::Return;
using ::testing::SetArrayArgument;
using ::testing::DoAll;

// ----- Mock-класс для TwoWire -----
class TwoWireMock {
public:
    MOCK_METHOD(void, begin, (), ());
    MOCK_METHOD(void, beginTransmission, (uint8_t), ());
    MOCK_METHOD(size_t, write, (uint8_t), ());
    MOCK_METHOD(size_t, write, (const uint8_t*, size_t), ());
    MOCK_METHOD(uint8_t, endTransmission, (), ());
    MOCK_METHOD(uint8_t, requestFrom, (uint8_t, uint8_t), ());
    MOCK_METHOD(int, available, (), ());
    MOCK_METHOD(uint8_t, read, (), ());
};

// Глобальный объект Wire, который будет использоваться в DS3231
TwoWireMock Wire;

// Подключаем тестируемый модуль (предполагается, что Wire.h
теперь указывает на наш mock)
#include "DS3231.h"

// ----- Тестовый класс -----
class DS3231Test : public ::testing::Test {
protected:
    void SetUp() override {
        // Сбрасываем mock перед каждым тестом (Google Mock делает
это автоматически)
    }
};

// ----- Тест метода setDateTime -----
TEST_F(DS3231Test, SetDateTime_EncodesAndWritesCorrectly) {
    // Конструктор DS3231_Class вызывает Wire.begin()
    EXPECT_CALL(Wire, begin()).Times(1);

    DS3231_Class ds3231;

    // Ожидаемая последовательность операций I2C
    EXPECT_CALL(Wire, beginTransmission(DS3231_ADDR)).Times(1);
    EXPECT_CALL(Wire, write(0x00)).Times(1); // адрес регистра
секунд

    // Ожидаем запись 7 байт данных в BCD формате
    uint8_t expectedData[7] = {
        0x12, // секунды 12

```

```

    0x34, // минуты 34
    0x56, // часы 56 (некорректное, но для проверки упаковки)
    0x03, // день недели 3
    0x78, // день месяца 78 (некорректное)
    0x09, // месяц 9
    0x20 // год 20
};

EXPECT_CALL(Wire, write(_, 7)).WillOnce([&](const uint8_t*
data, size_t len) {
    for (size_t i = 0; i < len; ++i) {
        EXPECT_EQ(data[i], expectedData[i]);
    }
    return len;
});

EXPECT_CALL(Wire, endTransmission()).Times(1);

// Вызов тестируемого метода
ds3231.setDateTime(12, 34, 56, 3, 78, 9, 20);
}

// ----- Тест метода getDateTIme -----
TEST_F(DS3231Test, GetDateTIme_ReadsAndFormatsCorrectly) {
    EXPECT_CALL(Wire, begin()).Times(1);
    DS3231_Class ds3231;

    // Имитируем данные, которые вернет DS3231 (в BCD)
    uint8_t simulatedData[7] = {
        0x45, // секунды 45
        0x30, // минуты 30
        0x12, // часы 12
        0x04, // день недели (игнорируется)
        0x31, // день 31
        0x05, // месяц 05
        0x24 // год 24 -> 2024
    };

    // Ожидаемые вызовы I2C при чтении
    EXPECT_CALL(Wire, beginTransmission(DS3231_ADDR)).Times(1);
    EXPECT_CALL(Wire, write(REGISTER_TIME)).Times(1);
    EXPECT_CALL(Wire, endTransmission()).Times(1);
    EXPECT_CALL(Wire, requestFrom(DS3231_ADDR,
7)).Times(1).WillOnce(Return(7));

    // Каждый вызов read() возвращает следующий байт из
    simulatedData
    {
        testing::InSequence seq;
        for (int i = 0; i < 7; ++i) {
            EXPECT_CALL(Wire,
read()).WillOnce(Return(simulatedData[i]));

```

```

    }
}

String result = ds3231.getDateTime();
// Ожидаемый формат: DD.MM.YYYY HH:MM:SS
EXPECT_EQ(result, "31.05.2024 12:30:45");
}

// ----- Тест метода getTemperature для положительных значений ---
--
TEST_F(DS3231Test, GetTemperature_PositiveValue) {
    EXPECT_CALL(Wire, begin()).Times(1);
    DS3231_Class ds3231;

    // Температура +25.5°C: целая часть 0x19 (25), дробная часть
    0x80 (биты 7-6 = 2 → 0.5)
    uint8_t tempInt = 0x19;
    uint8_t tempFrac = 0x80;

    EXPECT_CALL(Wire, beginTransmission(DS3231_ADDR)).Times(1);
    EXPECT_CALL(Wire, write(REGISTER_TEMPERATURE)).Times(1);
    EXPECT_CALL(Wire, endTransmission()).Times(1);
    EXPECT_CALL(Wire, requestFrom(DS3231_ADDR,
2)).Times(1).WillOnce(Return(2));
    {
        testing::InSequence seq;
        EXPECT_CALL(Wire, read()).WillOnce(Return(tempInt));
        EXPECT_CALL(Wire, read()).WillOnce(Return(tempFrac));
    }

    float temperature = ds3231.getTemperature();
    EXPECT_FLOAT_EQ(temperature, 25.5f);
}

// ----- Тест метода getTemperature для целого значения -----
TEST_F(DS3231Test, GetTemperature_IntegerValue) {
    EXPECT_CALL(Wire, begin()).Times(1);
    DS3231_Class ds3231;

    // Температура 18.0°C: дробная часть 0x00 (биты 7-6 = 0)
    uint8_t tempInt = 0x12; // 18
    uint8_t tempFrac = 0x00;

    EXPECT_CALL(Wire, beginTransmission(DS3231_ADDR)).Times(1);
    EXPECT_CALL(Wire, write(REGISTER_TEMPERATURE)).Times(1);
    EXPECT_CALL(Wire, endTransmission()).Times(1);
    EXPECT_CALL(Wire, requestFrom(DS3231_ADDR,
2)).Times(1).WillOnce(Return(2));
    {
        testing::InSequence seq;
        EXPECT_CALL(Wire, read()).WillOnce(Return(tempInt));
        EXPECT_CALL(Wire, read()).WillOnce(Return(tempFrac));
    }
}

```

```

    }

    float temperature = ds3231.getTemperature();
    EXPECT_FLOAT_EQ(temperature, 18.0f);
}

// ----- Тест метода getTemperature для значения с дробью 0.75 ---
--
TEST_F(DS3231Test, GetTemperature_Fraction075) {
    EXPECT_CALL(Wire, begin()).Times(1);
    DS3231_Class ds3231;

    // Температура 10.75°C: дробная часть 0xC0 (биты 7-6 = 3 →
    0.75)
    uint8_t tempInt = 0x0A; // 10
    uint8_t tempFrac = 0xC0;

    EXPECT_CALL(Wire, beginTransmission(DS3231_ADDR)).Times(1);
    EXPECT_CALL(Wire, write(REGISTER_TEMPERATURE)).Times(1);
    EXPECT_CALL(Wire, endTransmission()).Times(1);
    EXPECT_CALL(Wire, requestFrom(DS3231_ADDR,
2)).Times(1).WillOnce(Return(2));
    {
        testing::InSequence seq;
        EXPECT_CALL(Wire, read()).WillOnce(Return(tempInt));
        EXPECT_CALL(Wire, read()).WillOnce(Return(tempFrac));
    }

    float temperature = ds3231.getTemperature();
    EXPECT_FLOAT_EQ(temperature, 10.75f);
}

```