МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2021 г.

Программная реализация системы удаленного контроля и управления датчиками и устройствами климатического контроля

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____ П.О. Шабуров
«___»_____ 2021 г.

Автор работы,
студент группы КЭ-405
_____ С.М. Рассказов
«___»_____ 2021 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____ 2021 г.

Челябинск-2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___» _____2021 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Рассказову Сергею Михайловичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

Тема работы: «Программная реализация системы удаленного контроля и управления датчиками и устройствами климатического контроля» утверждена приказом по университету от 26 апреля 2021 г. №714-13/12 (приложение №73)

Срок сдачи студентом законченной работы: 1 июня 2021 г.

Исходные данные к работе:

Обеспечить основной функционал приложения:

1) Мониторинг последних пришедших в систему данных с датчиков и аппаратов.
2) Возможность отправлять и получать данные удаленными устройствами, подключенными к системе.
3) Возможность изменения данных конфигурации оборудования.
4) Возможность сохранять данные на неограниченный срок.
5) Оповещение пользователей в случае, если пришедшие данные выходят за установленные границы.

Предусмотреть ограничение доступа для пользователей не прошедших авторизацию и ограничения доступа к устройствам, не привязанным к пользователю.

Предусмотреть панель администратора для создания записей устройств, их датчиков и аппаратов.

Среда и средства реализации – по выбору автора.

Перечень подлежащих разработке вопросов:

- анализ аналогов разрабатываемого приложения;
- детализация требований к приложению;
- выбор среды и средств реализации;
- проектирование архитектуры приложения;
- организация базы данных;
- создание серверной и клиентской частей приложения;
- тестирование разработанного программного обеспечения.

5. **Дата выдачи задания:** 1 февраля 2021 г.

Руководитель работы _____/*П.О. Шабуров*/

Студент _____/*С.М. Рассказов*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение	01.03.2021	
Обзор литературы	14.03.2021	
Разработка модели, проектирование	01.04.2021	
Реализация системы	07.05.2021	
Тестирование системы	20.05.2021	
Оформление ВКР по требованиям ГОСТ	25.05.2021	
Подготовка презентации и доклада	31.05.2021	

Руководитель работы _____/*П.О. Шабуров*/

Студент _____/*С.М. Рассказов*/

АННОТАЦИЯ

С.М. Рассказов. Программная реализация системы удаленного контроля и управления датчиками и устройствами климатического контроля. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2021, 84- с., 33 ил., библиогр. список – 23 наим.

В рамках выпускной квалификационной работы разрабатывается веб-приложения для контроля датчиками температуры и аппаратами, подключенными к одному общему управляющему контроллеру. При разработке учитывались аналогичные системы, их сильные и слабые стороны. Результатом работы стало приложение, имеющее веб интерфейс, централизованную систему хранения и возможность работать с множеством пользователей.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В наше время стала очень популярна тема «IoT» - Internet of Things и всего что с этим связано. Во всем мире и в частности в России открывается множество стартапов, создаются проекты в этой области. Темой выпускной квалификационной работы является разработка веб-сервиса по мониторингу и управлению устройствами климатического контроля. Возможность удаленного слежения и управление внутренним климатом даст огромные преимущества. Так один оператор может обслуживать множество различных помещений находясь в удаленной точке.

Актуальность темы на данный момент обуславливается следующими факторами:

- качество датчиков является очень высоким;
- погрешности – минимальны;
- стоимость чипов снижается на протяжении уже нескольких лет подряд;
- количество и доступность чипов растёт из года в год;
- доступность интернета в высоком качестве;
- развитие 5G технологий;
- развитие «BigData» и нейронных сетей.

Задачи к выпускной работе:

1) Рассмотреть существующие на данный момент аналоги.
2) Выполнить анализ преимуществ и недостатков найденных систем, применить полученные результаты в процессе проектирования и разработки.
3) Составить требования к основному функционалу системы.
4) Выбрать методы и средства реализации проекта.
5) Разработать схему базы данных.
6) Выполнить программную реализацию приложения.
7) Выполнить тестирование готового продукта

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В настоящее время люди всё чаще хотят иметь возможность управлять сложными системами удаленно. Осуществлять мониторинг и не быть привязанными к определенному оборудованию.

В соответствии с заданием, требуется реализация следующего функционала:

– создание универсальной платформы для датчиков разных производителей;

– необходимо обеспечить целостность передаваемых данных с управляющего контроллера на сервер;

– реализовать защиту пользователя, защиту от неправомерного доступа к личным данным, в том числе показаний датчиков;

– интуитивно понятный интерфейс взаимодействия с датчиками и аппаратами;

– интерфейс удаленного конфигурирования оборудования при условии отсутствия у оператора знаний о функционировании конкретного датчика («черный ящик»);

– настроить удобное для пользователя оповещение о нештатных ситуациях.

В перспективе возможна доработка работы в виде расширения функционала для работы с датчиками измерения других факторов (нетемпературных). Работа с «BigData» - анализ и структурирование данных полученных за определенный период. Нейросеть – автоматический поиск зависимостей и распознавание нештатных ситуаций. Системы оповещения, такие как уведомления на смартфон или сообщения в мессенджерах. Автоматическое включение\отключения аппаратов для поддержания требуемых условий.

1.1 Обзор аналогов

В данный момент на рынке представлено множество компаний устанавливающих подобные системы. Далее представлены некоторые из них (таблица 1)

Таблица 1 - Названия компаний и веб сайты

Название компании	Веб сайт
ООО "Инженерные Технологии"	unicom1.ru
Научно-производственное предприятие КИПР	nppkipr.ru
Полтраф СНГ	poltraf.ru
Юнимон	unimon.ru
НПФ «КРУГ»	krug2000.ru

Каждая из подобных систем является закрытой и реализуется одним подрядчиком. При работе с системой имеется возможность использовать датчики только те, которые разработаны самой компанией. Многие из представленных компаний не имеют централизованной системы хранения и веб интерфейса.

1) ООО "Инженерные Технологии" – компания специализируется на разработке, производстве электронных контрольно-измерительных приборов и программного обеспечения для решения задач мониторинга, регистрации и контроля параметров микроклимата применительно в производственных и жилых помещениях, в фармацевтических и пищевых складах, агрокомплексах. Производимое оборудование соответствует требованиям FDA 21CFR Part 11 (Food and Drug Administration - Управление по санитарному надзору за качеством пищевых продуктов и медикаментов, США), а так же требованиям систем качества HACCP и GMP [1].

Из описания на сайте компании можно сделать вывод что системы, устанавливаемые этой компанией, являются ограниченными с точки зрения применяемых датчиков, компания не предоставляет удобного веб интерфейса для доступа к датчикам и удобного удаленного мониторинга. Для удаленного

управления служит программа «Гигротермион-АРМ», которая может быть установлена только на ПК с установленной операционной системой MS Windows (рисунок 1 и 2).

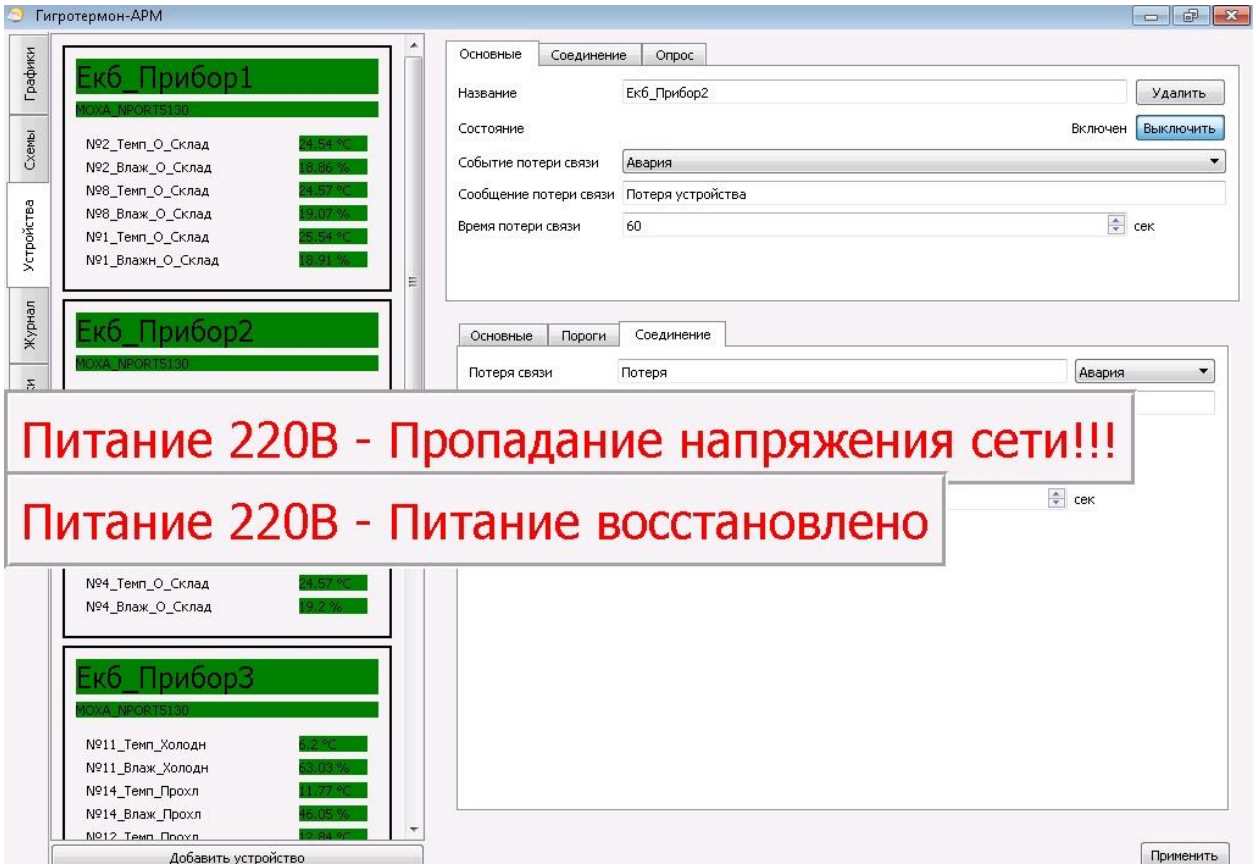

Рисунок 1 – Скриншот программы «Гигротермион-АРМ»

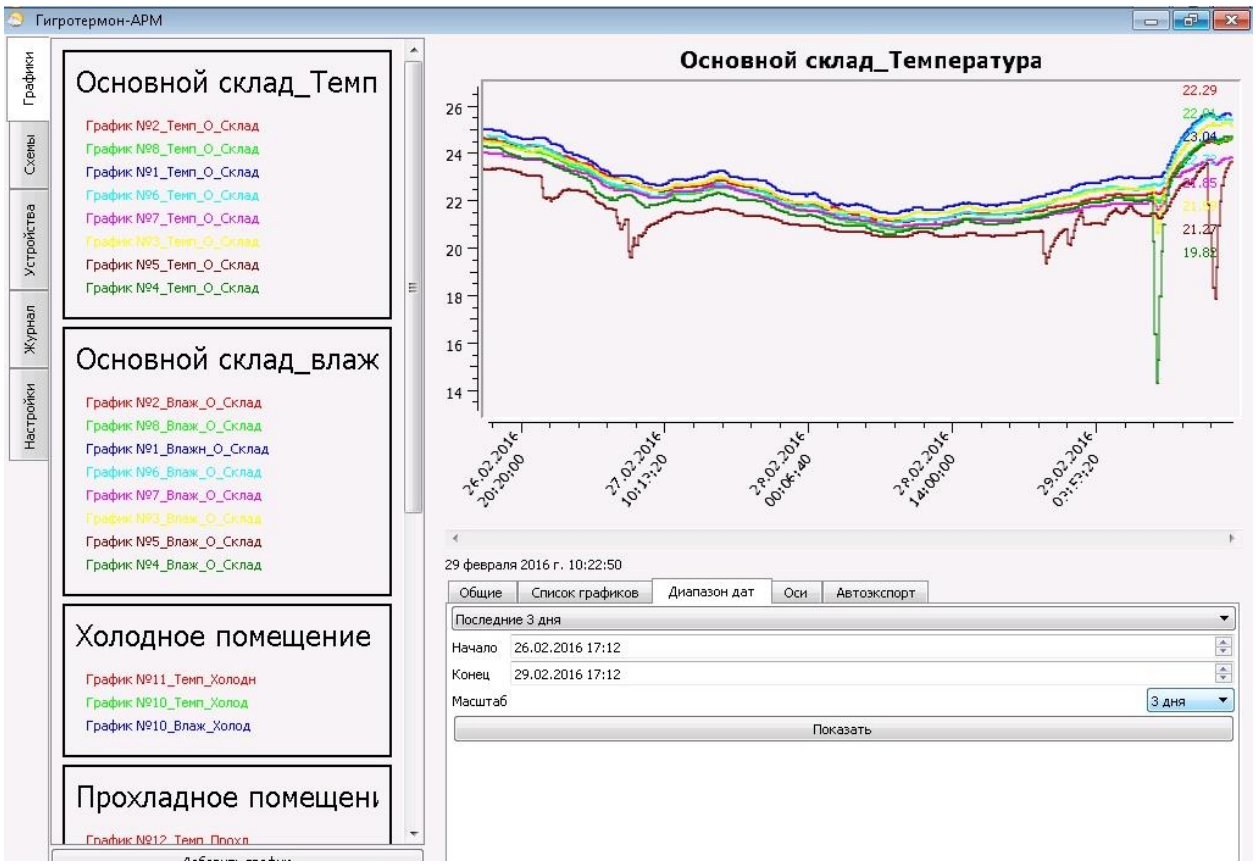

Рисунок 2 - Скриншот программы «Гигротермион-АРМ»

2) КИПР – компания сегодня является ведущим специализированным производством, выпускающим термометрическое оборудование. Все предлагаемые компанией системы полностью являются разработкой этого предприятия и право на их производство не имеет ни одна другая компания. КИПР предлагает устройства автоматического контроля температуры [2].

Также как и у предыдущего аналога система не имеет веб-интерфейса и требует установленного программного обеспечения на компьютер, управление через сеть интернет не предусмотрено. Интерфейс представляет из себя большую таблицу с показаниями (рисунок 3).

Рисунок 3 – Использование интерфейса компании КИПР

3) Полтраф СНГ – компания основана в 2006 году в г. Санкт-Петербурге. Это дочернее предприятие польского дистрибьютора Poltraf и швейцарского завода Trafag. Компания предоставляет оборудование для самых разных сфер применения: от энергетики, машиностроения и металлургии, горнодобывающей и химической промышленности до сельскохозяйственного комплекса, систем ОВК, метеорологии и гидрологии [3]. Компания не предоставляет программных средств визуализация показаний датчиков. В основном компания занимается продажей оборудования, в том числе датчиков температуры влажности и т.п. В продаже имеются датчики с цифровыми индикаторами температуры.

4) Unimon – простой и удобный сервис для удаленного контроля датчиков и управления оборудованием через Интернет и SMS. Готовый диспетчерский пульт в браузере и телефоне [4]. Продукт является похожим на то, что планируется

получить по итогу этой работы, но главное различие состоит в следующем: этот продукт позволяет использовать в системе мониторинга датчики и оборудование только собственного производства. Из плюсов стоит отметить то, что продукт имеет веб версию и приложения для мобильных операционных систем Android и IOS (рисунок 4).

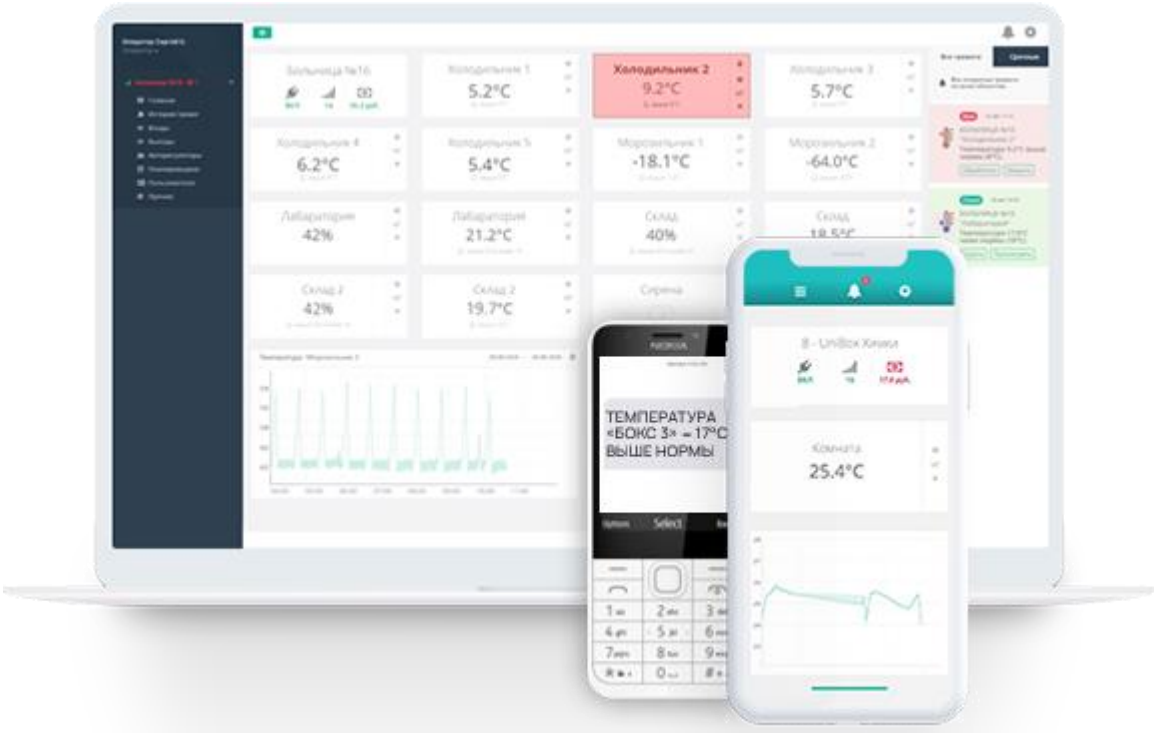

Рисунок 4 – Примеры интерфейса продукта «Unimon»

5) НПФ «КРУГ» – крупная российская компания, компетенции которой находятся в области промышленной автоматизации и разработки программного обеспечения [5]. Юридический адрес компании находится в городе Пенза, а филиалы расположены в Москве, Санкт-Петербурге, Ульяновске, Россоши, Новочеркасске и Астане. Программный продукт, предоставляемый для визуализации, должен находится на персональном компьютере под управлением операционной системы MS Windows. И подключается к «управляющему устройству» посредством интерфейса RS-485 (рисунок 5). Очевидными недостатками является следующее:

- ограничение подключения (по территориальному признаку);
- ограничение по платформе для мониторинга;
- устаревший интерфейс связи.

Рисунок 5 – Архитектура системы мониторинга от НПФ «Круг»

1.2 Анализ основных технологических решений

Язык программирования и фреймворк это фундамент, на котором стоит весь проект.

1.2.1 Выбор языка программирования

1.2.1.1 PHP

PHP (рекурсивный акроним словосочетания PHP: Hypertext Preprocessor) - это распространенный язык программирования общего назначения с открытым исходным кодом. PHP специально сконструирован для веб-разработок и его код может внедряться непосредственно в HTML.

PHP отличается от JavaScript тем, что PHP-скрипты выполняются на сервере и генерируют HTML, который посылается клиенту. Есть возможность даже настроить свой сервер таким образом, чтобы обычные HTML-файлы обрабатывались процессором PHP, так что клиенты даже не смогут узнать, получают ли они обычный HTML-файл или результат выполнения скрипта [6].

Создание скриптов для выполнения на стороне сервера. PHP традиционно и наиболее широко используется именно таким образом. Для этого необходимы три вещи. Интерпретатор PHP (в виде программы CGI или серверного модуля), веб-сервер и браузер. Для того чтобы можно было просматривать результаты выполнения PHP-скриптов в браузере, нужен работающий веб-сервер и установленный PHP. Просмотреть вывод PHP-программы можно в браузере, получив PHP-страницу, сгенерированную сервером. В случае, если проект находится еще на стадии разработки, можно использовать свой домашний компьютер вместо сервера (локальный сервер).

Также в PHP включена поддержка большинства современных веб-серверов, таких как Apache, IIS и многих других. В принципе, подойдет любой веб-сервер, способный использовать бинарный файл FastCGI PHP, например, lighttpd или nginx.

Одним из значительных преимуществ PHP является поддержка широкого круга баз данных. Создать скрипт, использующий базы данных, - довольно

просто. Можно воспользоваться расширением, специфичным для отдельной базы данных (таким как mysql) или использовать уровень абстракции от базы данных, такой как PDO, или подсоединиться к любой базе данных, поддерживающей «Открытый Стандарт Соединения Баз Данных» (ODBC), с помощью одноименного расширения ODBC. Для других баз данных, таких как CouchDB, можно воспользоваться подключением по URL или сокетами.

PHP также поддерживает взаимодействие с другими сервисами через такие протоколы, как LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (на платформах Windows) и многих других. Кроме того, предусмотрена возможность работать с сетевыми сокетами напрямую. PHP поддерживает стандарт обмена сложными структурами данных WDDX практически между всеми языками веб-программирования. Обращая внимание на взаимодействие между различными языками, следует упомянуть о поддержке объектов Java и возможности их использования в качестве объектов PHP [7].

Laravel

Один из самых популярных фреймворков языка PHP. Он является бесплатным и имеет открытый код, предназначенный для разработки с использованием архитектурной модели MVC.

Ключевые особенности:

– удобный синтаксис;

– расширенная функциональность, за счет дополнений;

– встроенные функции для управления маршрутизацией, управлением пользователями, кэшированием и многим другим;

– интеграция Laravel со сторонними библиотеками и платформами, такими как AWS;

– запуск задач асинхронно в фоновом режиме для повышения производительности.

Laravel за счет своей популярности имеет высокоактивное сообщество, что означает, что поиск помощи или учебных пособий не представляет из себя

проблемы. Если опыта использования фреймворка еще нет, это делает Laravel ещё более привлекательным вариантом [8].

CodeIgniter

CodeIgniter (англ) — это фреймворк PHP, использующий архитектуру Model View Controller (MVC). С точки зрения новичка, это означает, что CodeIgniter использует различные компоненты для решения конкретных задач разработки. Этот подход популярен среди разработчиков, потому что он позволяет создавать масштабируемые веб-приложения с меньшими размерами.

Ключевые особенности:

- подробная документация;
- масштабируемость приложения, благодаря использованию архитектуры MVC.

Есть много плюсов в использовании CodeIgniter. Тем не менее, нельзя называть его лучшим PHP фреймворк, потому что он также имеет некоторые недостатки. Например, его обновления несколько нерегулярны, поэтому он не может быть лучшим вариантом для приложений, которым требуются стандарты безопасности высокого уровня [9].

Symfony

Ключевые особенности:

- гибкая структура, которая позволяет настраивать отдельные компоненты;
- встроенная система тестирования;
- подробная документация.

Symfony может быть хорошим вариантом, если необходима модульность. Фреймворк позволяет использовать компоненты, вместо полной библиотеки, которая может сделать это лучшей средой PHP, если нужна только небольшая часть. За счет очень гибкой структуры этот фреймворк становится менее производительным.

1.2.1.2 Python

Python (пито́н) — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис языка Python не перегружен служебными операторами и интуитивно понятен. В то же время стандартная библиотека включает большой набор полезных функций.

Python поддерживает структурное, обобщенное, объектно-ориентированное, функциональное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция (это означает, что для любого объекта можно получить всю информацию о его внутренней структуре), механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты [10].

Django

Высокоуровневый фреймворк, предназначенный для быстрой и эффективной разработки. Его архитектура выстроена так, чтобы максимально экономить время и силы разработчика, а в будущем — время и деньги заказчиков. Основные принципы философии Django:

DRY(Don’t repeat yourself) - это принцип разработки программного обеспечения, нацеленный на снижение повторения информации различного рода, особенно в системах с множеством слоёв абстрагирования.

RAD (rapid application development — быстрая разработка приложений) - концепция организации технологического процесса разработки программных продуктов, ориентированная на максимально быстрое получение качественного результата в условиях сильных ограничений по срокам и бюджету и нечётко определённых требований к продукту. — стимулируют повторное использование кода и уменьшают избыточность [11].

Django использует шаблон проектирования Model-View-Controller (MVC).

MVC (Model-View-Controller) это модель разделения кода для решения разных задач. Делится она на 3 категории, это:

1) Модель — этот компонент отвечает за данные.

2) Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.

3) Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя.

Ключевые особенности:

- контроль версий для баз данных (миграции);
- собственный движок шаблонов;
- объектно-реляционные отображения (ORM);
- маршрутизация URL;
- поддержка веб-серверов;
- поддержка аутентификации;
- поддержка интернационализации.

Плюсы

1) Масса библиотек. Базовую функциональность не нужно писать самому — многое уже написано, остаётся только импортировать соответствующие библиотеки. Как правило, библиотеки написаны качественно и хорошо задокументированы.

2) Сообщество и документация. У Django подробная документация и дружелюбное сообщество, поэтому всегда можно найти ответ на свой вопрос.

3) Масштабируемость. Если изначально неизвестно, насколько сильно ваш проект вырастет и вырастет ли вообще (как это обычно и бывает со стартапами), Django позволяет начать с малого и масштабироваться по мере необходимости.

Минусы

1) Django без надстроек не поддерживает WebSockets, поэтому он плохо подходит для работы в реальном времени.

2) Готовые библиотеки — это в целом хорошо, но часто они снижают гибкость.

Используется в некоторых масштабных проектах, например:

1) Instagram — социальная сеть для обмена фотографиями и видео.

2) Spotify — музыкальная платформа, рекомендующая пользователям музыку в зависимости от их предпочтений.

Pyramid

Синтаксически менее нагруженный инструмент. Этот фреймворк подходит для проектов любого размера. В нём есть полезные особенности для создания сложных приложений или масштабирования изначально небольших сайтов под возросшую нагрузку [12].

Ключевые особенности:

- удобные инструменты для работы со статичными объектами;
- удобная работа с предикатами и рендерарами страниц;
- генерация URL.

Плюсы

1) Гибкость и удобство. Любой компонент фреймворка, будь то база данных или движок шаблонов, может быть заменён. Можно даже использовать несколько разных компонентов одновременно (например, подключить две разные базы данных).

2) Удобные Ajax-запросы. Благодаря системе декораторов и представлений можно отправлять XHR-запросы без дополнительных усилий со стороны разработчика.

3) Поддержка SQLAlchemy. SQLAlchemy обеспечивает удобную работу с базами данных даже для сложных запросов.

Минусы

1) Требует времени на развёртывание и подготовку к разработке.

2) Чтобы использовать всю гибкость и удобство, нужно хорошо разбираться в Pyramid.

3) Для простых приложений SQLAlchemy может быть чересчур громоздкой.

4) Используется в некоторых масштабных проектах, например:
5) Charte.ca — онлайн редактор графиков для неспециалистов.
6) Easy Blog Networks — хостинг-провайдер.
7) Substance-D — среда разработки веб-приложений.

TurboGears

TurboGears — фреймворк с открытым исходным кодом для быстрой разработки приложений, работающих с данными. Он поддерживает SQLAlchemy, Genshi, WebOb, and Repoze, так что TurboGears прекрасно подходит для любой системы, требующей хорошей поддержки баз данных [13].

Ключевые особенности:

- поддержка горизонтального секционирования данных;
- интеграция с Java-script библиотекой MochiKit;
- поддержка нескольких баз данных одновременно;
- поддержка архитектуры MVC;
- ToscaWidgets;
- шаблоны PasteScript;
- валидация через FormEncode.

Плюсы

1) Гибкость. TurboGears можно использовать и как микро-фреймворк для быстрого прототипирования, и как полноценный «фулстек» фреймворк для поддержки масштабных проектов.

2) Расширяемость. Можно создавать собственные плагины или дополнять функциональность имеющихся.

Минусы

1) Множество возможностей для расширения, которые не встроены изначально и при их настройке можно легко запутаться или не справится с их взаимодействием.

Используется в некоторых масштабных проектах, например:

1) Apache Allura — онлайн-платформа для управления репозиториями кода, созданная SourceForge.

2) Kamisons — сайт по продаже зонтов.

1.2.1.3 C#

C# — это объектно- и компонентно-ориентированный язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО [14].

Написание Веб сервиса на этом языке подразумевает платформу ASP.NET

ASP.NET (Active Server Pages для .NET) — платформа разработки веб-приложений, в состав которой входит: веб-сервисы, программная инфраструктура, модель программирования, от компании Майкрософт. ASP.NET входит в состав платформы .NET Framework и является развитием более старой технологии Microsoft ASP.

.NET Framework — это программная платформа, выпущенная компанией Microsoft, которая подходит для разных языков программирования.

Считается, что платформа .NET Framework явилась ответом компании Microsoft на набравшую к тому времени большую популярность платформу Java. ASP.NET основывается на Common Language Runtime: разработчики могут писать код для ASP.NET, используя практически любые языки программирования, некоторые из которых входят в комплект .NET Framework (C#, Visual Basic.NET и JScript .NET), а другие могут быть установлены дополнительно (IronRuby, IronPython, PHP, Perl, Smalltalk, Haskell и др.).

Некоторые особенности ASP.NET:

– компилируемый код выполняется быстрее, а большинство ошибок отлавливается ещё на стадии разработки;

– расширяемый набор элементов управления и библиотек классов, ускоряющий разработку;

– возможность кэширования всей страницы, её частей или данных, используемых на странице;

– возможность разделения визуальной части и бизнес-логики по разным файлам, есть возможность выделять часто используемые шаблоны пользовательских элементов управления, таких как меню сайта, наличие master-страниц для задания шаблонов оформления, поддержка AJAX (расширение ASP.NET AJAX);

– расширяемые модели событий, обработки запросов и серверных элементов управления;

– поддержка CRUD-операций при работе с таблицами через GridView;

– возможно создание веб-приложений, которые реализуют шаблон Model-View-Controller (ASP.NET MVC Framework).

.NET достаточно широко распространён в сфере разработки внутрикорпоративных программных продуктов, но в веб-разработке всё же встречается относительно редко, как и другие программные продукты корпорации Microsoft. Использование .NET «тянет» за собой покупку и иного ПО от корпорации Microsoft (серверной ОС, СУБД и т.п.). Технология достаточно дорогая в разработке и сопровождении: кроме затрат на покупку лицензий на необходимое ПО существенный вклад в бюджет проектов вносят высокие зарплаты разработчиков.

В результате анализа особенностей основных языков программирования веб приложений выбор пал на Python. Это очень перспективный язык с отличной документацией и поддержкой. Также фреймворк Django для этого языка кажется подходящим для поставленных задач.

1.2.2 Выбор среды разработки

IDE (или интегрированная среда разработки) — это программа, предназначенная для разработки программного обеспечения. Как следует из названия, IDE объединяет несколько инструментов, специально предназначенных для разработки. Эти инструменты обычно включают редактор, предназначенный

для работы с кодом. Например, подсветка синтаксиса и автодополнение, инструменты сборки, выполнения и отладки; и определённую форму системы управления версиями.

Большинство IDE поддерживают множество языков программирования и имеют много функций, из-за чего могут быть большими, занимать много времени для загрузки и установки и требуют глубоких знаний для правильного использования.

С другой стороны, есть редакторы кода, которые представляют собой текстовый редактор с подсветкой синтаксиса и возможностями форматирования кода. Большинство хороших редакторов кода могут выполнять код и использовать отладчик, а лучшие даже могут взаимодействовать с системами управления версиями. По сравнению с IDE, хороший редактор кода, как правило, легковесней и быстрее, но зачастую ценой меньшей функциональности.

1.2.2.1 Eclipse

Eclipse является бесплатной программной платформой с открытым исходным кодом, контролируется организацией Eclipse Foundation. Написана на языке программирования Java и основной целью её создания является повышение продуктивности процесса разработки программного обеспечения.

Для разработки на языке программирования python есть специальные плагины, самый популярный из них PyDev.

PyDev — надстройка над Eclipse, превращающая его в интегрированную среду разработки на Python.

Основные возможности:

- автопродолжение кода (Code completion);
- подсветка синтаксиса (Syntax highlighting);
- анализ кода (Code analysis);
- возможность перехода к определению функции (Go to definition);
- возможность рефакторинга (Refactoring);
- отладчик (Debugger);

- удалённый отладчик (Remote debugger);
- интерактивная консоль (Interactive console);
- интеграция с unittest-ами (Unittest integration);
- покрытие кода тестами (Code coverage).

Преимущества: Если есть опыт работы с этой IDE, то уставновка плагина и его освоение будет крайне простым и быстрым.

Недостатки: если пользователь только начинает изучать Python или разработку в целом, Eclipse может стать непосильной ношей.

1.2.2.2 Sublime Text

Sublime Text, написан инженером из Google с мечтой о лучшем текстовом редакторе. Является весьма популярным редактором кода для python. Доступен на всех платформах. Sublime Text имеет встроенную поддержку редактирования Python-кода, а также богатый набор расширений, называемых пакетами, которые расширяют возможности синтаксиса и редактирования.

Установить дополнительный Python-пакет может быть непросто — все пакеты Sublime Text написаны на Python, поэтому для установки пакетов сообщества зачастую может потребоваться выполнить Python-скрипт непосредственно в редакторе.

Преимущества:у Sublime Text большое количество поклонников. Как редактор кода, Sublime Text быстрый, лёгкий и имеет хорошую поддержку.

Недостатки: Sublime Text не является бесплатным, хотя вы можете использовать пробный период сколько угодно. Установка расширений может создать множество проблем. Кроме того, в редакторе нет поддержки отладки и запуска кода.

1.2.2.3 PyCharm

Одной из лучших полнофункциональных IDE, предназначенных именно для Python, является PyCharm. Существует как бесплатный open-source (Community), так и платный (Professional) варианты IDE. PyCharm доступен на Windows, Mac OS X и Linux.

PyCharm «из коробки» поддерживает разработку на Python напрямую — откройте новый файл и начинайте писать код. Вы можете запускать и отлаживать код прямо из PyCharm. Кроме того, в IDE есть поддержка проектов и системы управления версиями.

Преимущества: это среда разработки для Python с поддержкой огромного функционала хорошо взаимодействующего друг с другом. Также среда может похвастаться большим и отзывчивым сообществом. В ней «из коробки» можно редактировать, запускать и отлаживать Python-код.

Недостатки: PyCharm может медленно загружаться, а настройки по умолчанию, возможно, придётся подкорректировать для существующих проектов.

В результате анализа сред разработки, выбрана PyCharm. Она предоставляет возможность создания виртуального окружения и локального виртуального сервера, что позволяет не устанавливать дополнительного ПО, такого как nginx или apache. Также имеет все возможности текстового редактора файлов форматов HTML,CSS и другие.

1.2.3 Выбор базы данных и системы управления базой данных

База данных является одним из важнейших компонентов веб сервиса. От неё зависит сложность разработки, быстродействие всей системы и отказоустойчивость проекта. В данной работе планируется использовать реляционную базу данных. На сегодняшний день существует множество различных баз данных, рассмотрим несколько из них: MySql, PostgreSql, , Access, MariaDB, SQLite.

MySQL – это «Open Source» проект, который почти во всех случаях может быть использован бесплатно, разработчики лишь рекомендуют приобрести лицензию если эта разработка приносит деньги в бизнесе. Это способствует развитию проекта. Также плюсами этого проекта, является то, что MySQL может быть запущен как на персональных компьютерах под управления операционной системы MS Windows, так и на серверном оборудований с высокой мощностью или малых серверах развернутых на Linux дистрибутивах [15]. СУБД обладает

отличной переносимостью. Она зарекомендовала себя благодаря тому, что многие большие проекты такие как: Facebook, Google, Twitter используют её.

PostgreSQL - это мощная объектно-реляционная база данных с открытым исходным кодом, активная разработка которой насчитывает более 30 лет, что принесло ей прочную репутацию за надежность, функциональность и производительность.

В официальной документации можно найти огромное количество информации, описывающей, как установить и использовать PostgreSQL [16]. С подробной таблицей характеристик можно ознакомиться по ссылке 17 библиографического списка.

Microsoft Office Access или просто Microsoft Access — реляционная система управления базами данных (СУБД) корпорации Microsoft. Входит в состав пакета Microsoft Office. Имеет широкий спектр функций, включая связанные запросы, связь с внешними таблицами и базами данных. Благодаря встроенному языку VBA, в самой Access можно писать приложения, работающие с базами данных. Подходит скорее для учебных проектов или проектов малого бизнеса, язык VBA является устаревшим, и в целом СУБД для серьезных проектов подходит не в полной мере из-за своих ограничений.

MariaDB Server - одна из самых популярных реляционных баз данных с открытым исходным кодом. Она сделана оригинальными разработчиками MySQL и имеет открытый исходный код. MariaDB входит в состав большинства облачных решений и используется по умолчанию во многих дистрибутивах Linux.

MariaDB Server основан на ценностях производительности, стабильности и открытости. Недавние новые функции включают расширенную кластеризацию с Galera Cluster 4, функции совместимости с Oracle Database и Temporal Data Tables, позволяющие запрашивать данные в том виде, в котором они были в любой момент в прошлом [17].

SQLite – компактная СУБД с открытым исходным кодом. Основным отличием является архитектура. В основном СУБД являются клиент-серверными приложениями, то есть база данных работает как отдельный процесс, а

приложение к нему подключается. SQLite является встраиваемой СУБД, то есть является отдельным файлом до 140 ТБ. Обычно её используют в мобильной разработке. Также её можно использовать для небольших учебных проектов, но стоит учитывать ограниченный функционал. При создании проекта Django в среде PyCharm SQLite является базой по умолчанию.

В результате исследования баз данных была составлена сравнительная таблица (таблица 2).

Таблица 2 – Сравнительная таблица исследуемых баз данных

Критерий / Название	Открытый код	Гибкость запросов	Надежность	Интеграция с фреймворком
MySQL	+	+	+	-
PostgreSQL	+	+	+	+
Microsoft Access	-	-	-	-
MariaDB	+	-	-	-
SQLite	+	-	-	+

В результате анализа баз данных и их взаимодействие с языком программирования python была выбрана PostgreSQL. Главным критерием при её выборе была глубокая интеграция фреймворка Django с ней, и в целом язык python разрабатывался для тесного взаимодействия именно с этой базой данных.

2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

2.1 Технические

1) Работа системы на популярных хостингах.
2) Отсутствие необходимости специфического серверного оборудования.

2.2 Функциональные

1) Возможность программно передавать информацию о датчиках в систему использую api запросы.
2) Возможность программно передавать информацию о статусе аппаратов подключенных к системе через управляющий контроллер используя api запросы.
3) Многопользовательский доступ к управлению и мониторингу.
4) Возможность добавлять несколько установок для одного пользователя.
5) Система оповещения пользователя по e-mail, возможно дополнить другими способами.
6) Система должна иметь веб интерфейс с возможностью мониторинга параметров управляющего контроллера и их изменения удаленно.

2.3 Системные

1) Возможность масштабирования при большом количестве пользователей.
2) Хранение данных в течение трёх месяцев с момента получения.
3) Работа с устройствами разных производителей.
4) Устойчивость к потере сети.
5) Централизация хранения.

2.4 Подсистемы

1) Система мониторинга.
2) Система управления.
3) Модуль обработки входящей информации.
4) Контроль доступа к данным.
5) Модуль оповещения.
6) Модуль администратора

Функциональные требования к подсистеме “Система мониторинга”:

1) Возможность изменять границы значений для датчиков.
2) Выделение участков, выходящих за границы и оповещение пользователя.

Функциональные требования к подсистеме “Система управления”:

1) Возможность устанавливать собственные имена для управляющих контроллеров, датчиков и аппаратов.
2) Возможность установки границ для контроллеров и датчиков.
3) Возможность отправлять команды на включение и выключение аппаратов.

Функциональные требования к подсистеме «Модуль обработки входящей информации»:

Вся входящая информация должна проверяться на правильность и заноситься в базу. Для данных с датчиков необходимо проверять выход за допустимые границы. Если всё в порядке, то данные просто заносятся в базу. Если параметры имеют слишком высокие или низкие значения, то информация передается в модуль оповещения, а в системе мониторинга на этом акцентируется внимание.

Функциональные требования к подсистеме «Контроль доступа к данным»:

Для доступа к информации пользователь должен авторизоваться. Вся информация об установке должна быть доступна только её владельцу указанному в системе. При получении новых данных с контроллеров использовать ключ для проверки устройства.

Функциональные требования к подсистеме «Модуль оповещения»:

Система должна отправлять оповещения на электронную почту в случае возникновения нештатных ситуаций. В перспективе возможно SMS оповещение и сообщения в мессенджерах. При создании мобильного приложения реализовать «push» уведомления.

Функциональные требования к подсистеме “Модуль администратора”:

Иметь доступ к всем данным пользователя(кроме пароля). Возможность добавлять управляющие контроллеры, датчики и аппараты. Их привязка друг к другу и пользователю.

Нефункциональные требования системы

Разработка на языке «python» с использованием фреймворка «Django». Возможны вставки на других языках программирования.

Использовать PostgreSQL в качестве базы данных.

2.5 Требования к пользователям

Регистрация пользователей осуществляется администратором системы, как и регистрация оборудования, которое имеет пользователь. Отдельно должны быть переданы данные о датчиках и аппаратах. В оборудование должны быть указаны параметры, полученные от администратора. Такие как идентификатор установки, идентификатор для каждого датчика и аппарата. А также адрес веб сервера для передачи и получения информации от системы.

2.6 Требования к оборудованию

Оборудование должно иметь доступ в интернет любым способом. Также оборудование должно иметь возможность отправлять POST запросы. Контроллер должен иметь функционал, позволяющий сериализовывать данные в формат json и десериализовывать из него.

2.7 Требования к объёму данных

Объём данных не регламентируется, так как напрямую зависит от количества пользователей и времени работы системы.

3 ПРОЕКТИРОВАНИЕ

3.1 Описание процесса

Процесс работы системы начинается с добавления в базу данных пользователя и создание его профиля.

Отдельным шагом является добавление в базу данных информации об управляющем контроллере, его датчиков и аппаратов с которыми предполагается работа. Клиент должен получить идентификаторы для своего оборудования и записать их в контроллер.

Далее устанавливается связь между пользователем и управляющим контроллером. Эта связь имеет отношение «многие ко многим», так как пользователь может иметь несколько таких устройств и один контроллер может быть подконтрольным нескольким пользователям.

Для того, чтобы система получила данные, управляющий контроллер должен физически их отправить на сервер через api или форму, а для получения текущей конфигурации должны сделать соответствующие api запросы.

После получения системой данных с управляющего контроллера происходит их валидация (проверка на правильность). Далее в зависимости от их типа (аппарат или датчик) выполняются разные действия. Для датчика это сравнение пришедшей температуры с пришедшими границами и в случае неудовлетворения отправляется сообщение на адрес электронной почты указанной в профиле и выставляется флаг «alarm» в базе данных. Далее данные записываются в базу данных. В случае, если данные не выходят за допустимые границы или пришли данные о статусе аппарата, то данные просто записываются в базу данных.

3.2 Архитектура предлагаемого решения

Система представляет из себя программный продукт, располагающийся на удаленном сервере. Подключение к ней осуществляется посредством сети Internet. Задачей системы является получение, анализ и хранение получаемых данных с

управляющих контроллеров. А также их удаленное конфигурирование и визуализация параметров.

На рисунке 6 схематично изображена архитектура приложения.

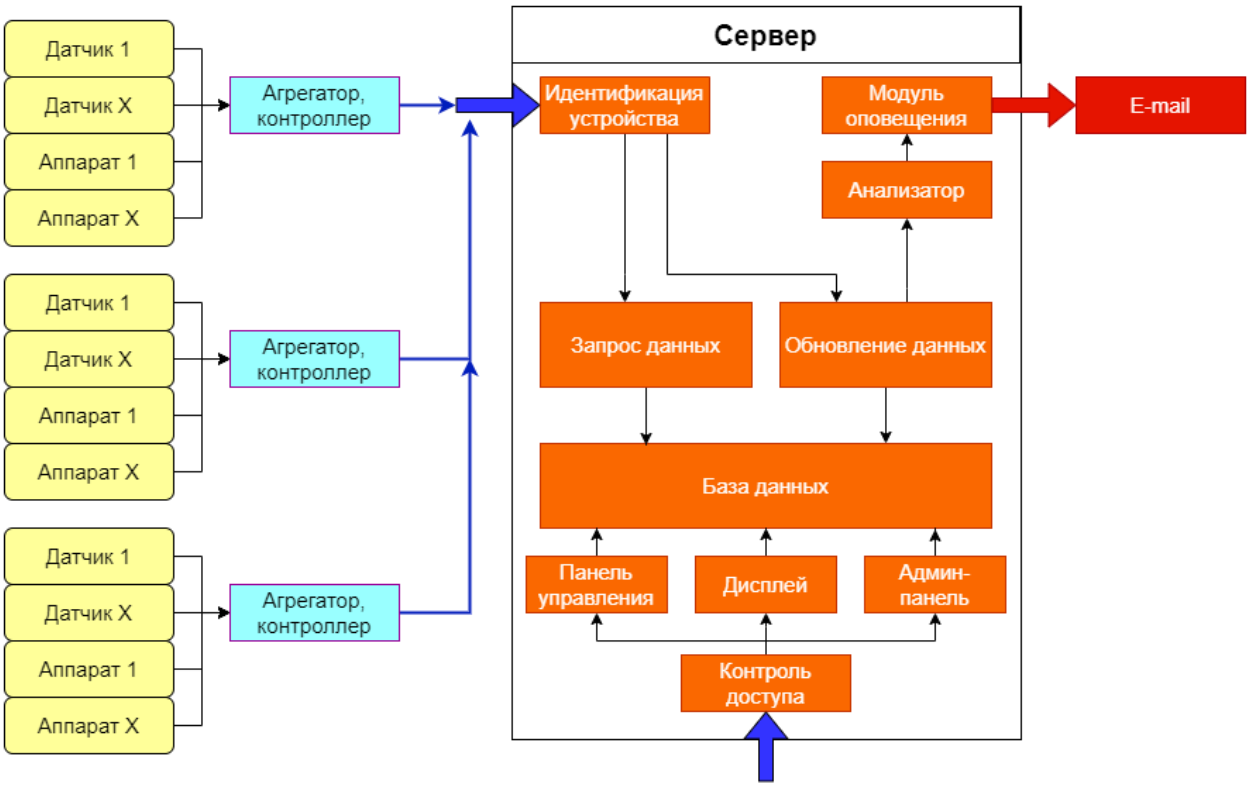

Рисунок 6 - Изображение проекта схематично

4 РЕАЛИЗАЦИЯ

Для реализации был выбран язык программирования Python и фреймворк Django. Базой данных является PostgreSQL, а средой разработки PyCharm. После создания проекта необходимо выполнить первоначальную настройку. Сконфигурировать виртуальное окружение и создать приложение внутри проекта. Фреймворк Django предполагает, что на одном веб сайте может быть несколько приложений не связанных друг с другом. После создания, необходимо в настройках его подключить. По умолчанию Django в качестве базы данных использует SQLite. Так как для разработки была выбрана PostgreSQL, то нужно внести данные о ней в конфигурационный файл после чего можно приступать к разработке.

4.1 Model (База данных)

Django предоставляет уровень абстракции над базой данных и позволяет работать с данными в ней как с обычными объектами. То есть при разработке программист не создает отдельный таблицы с полями в конкретной базе данных, а создает модели в проекте. При помощи команд виртуального окружения «makemigrations» и «migrate» осуществляется преображение модели в таблицы [19,22]. Схема базы данных (моделей) представлена содержит 1 стандартную модель и 7 созданных в рамках данной работы (рисунок7).

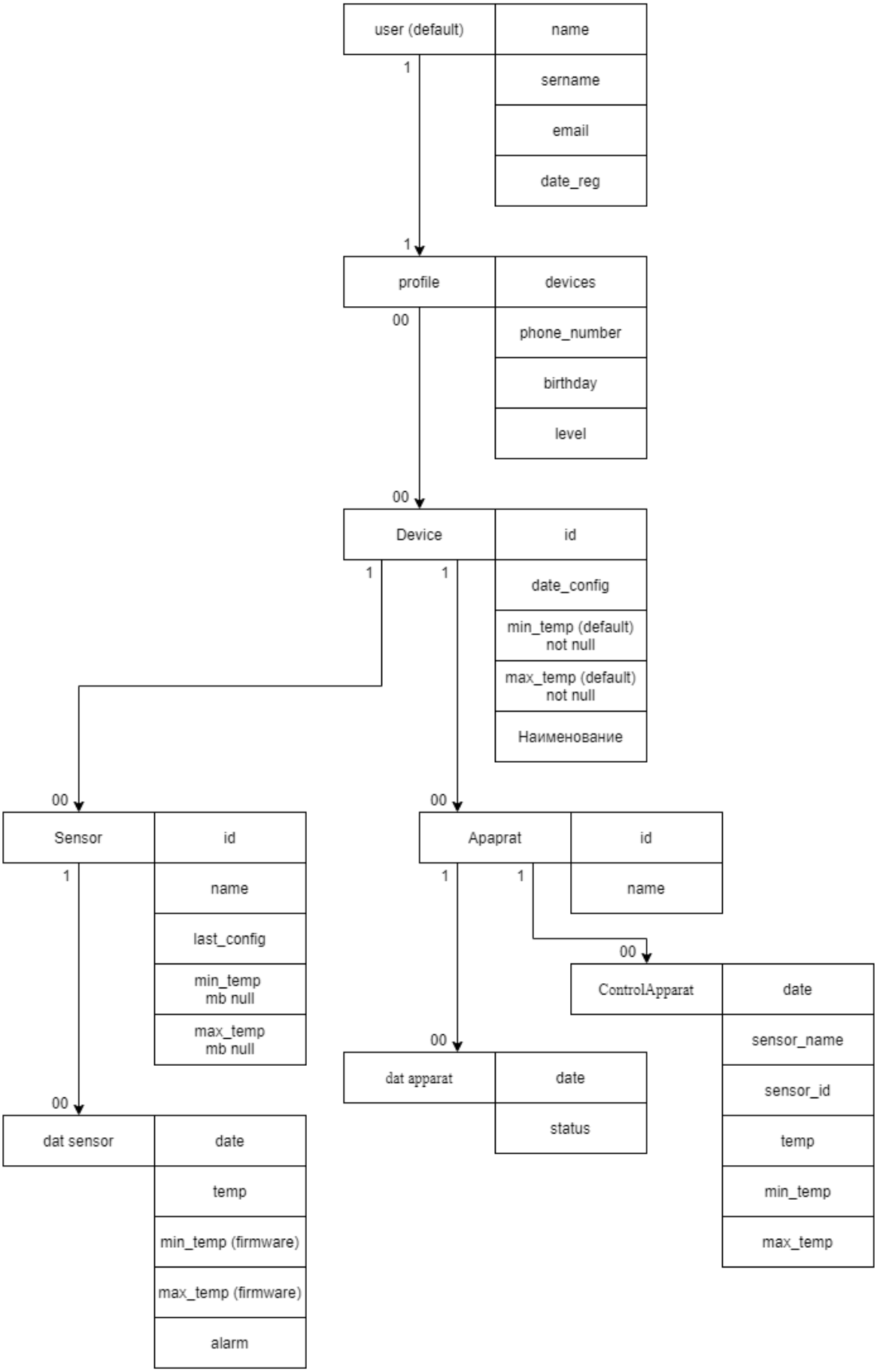

Рисунок 7 – Схема базы данных (моделей)

4.1.1 Users

Users это модель создаваемая Django по умолчанию. В основном она служит для хранения служебной информации. Описание полей находится в таблице 3.

Таблица 3 – Описание полей модели Users

Название поля	Описание поля	Служит для хранения:
Username	Текстовое поле. Обязательно для заполнения.	Логина
Password	Поле пароля. Обязательно для заполнения.	Пароля пользователя. Django автоматически шифрует все хранимые пароли с помощью алгоритма pbkdf2_sha256.
First name	Текстовое поле.	Имени пользователя.
Last name	Текстовое поле.	Фамилии пользователя.
Email address:	Текстовое поле. Имеет маску почтового адреса.	Адреса электронной почты пользователя
Active	Логическое поле.	Статуса активности аккаунта
Staff status	Логическое поле.	Статуса сотрудника
Superuser status	Логическое поле.	Статуса суперпользователя
Groups	Мультивыбор	Групп, к которым принадлежит пользователь

Продолжение таблицы 3

Название поля	Описание поля	Служит для хранения:
User permissions	Мультивыбор	Прав на действия в системе
Last login	Дата и время	Даты и времени последней авторизации
Date joined	Дата и время	Даты и времени создания аккаунта

4.1.2 Profile

Profile – модель служит для хранения дополнительных данных о пользователе. Описание полей находится в таблице 4. Исходный код модели приведен в листинге 1.

Листинг 1 – Исходный код модели Profile

```
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    devices = models.ManyToManyField(Device, blank=True)
    phonenumber = models.CharField(verbose_name="phone number", max_length=11)
    birthdate = models.DateField(verbose_name="birth date")
    level = models.PositiveIntegerField('level', default=10)
    class Meta:
        verbose_name = 'Профиль'
        verbose_name_plural = 'Профили'
```

Таблица 4 – Описание полей модели Profile

Название поля	Описание поля	Служит для хранения:
user	Связь один к одному с моделью Users	Привязке профиля к аккаунту
devices	Связь многие ко многим с моделью Device	Привязке профиля к управляющим контроллерам
phonenumber	Текстовое поле.	Номера телефона
birthdate	Дата	Даты рождения

Продолжение таблицы 4

Название поля	Описание поля	Служит для хранения:
level	Положительное целое число	Хранит уровень доступа пользователя

4.1.3 Device

Device – модель служит для хранения информации об управляющих контроллерах. Описание полей находится в таблице 5. Исходный код модели приведен в листинге 2.

Листинг 2 – Исходный код модели Device

```
class Device(models.Model):
    date_last_config = models.DateTimeField('date_last_config')
    min_temperature = models.IntegerField(default=18)
    max_temperature = models.IntegerField(default=28)
    name = models.CharField(max_length=255, default='NoName')
    key = models.CharField(max_length=64, default=get_key)

    def get_sensors(self):
        return self.device_sensor.all()

    def get_apparats(self):
        return self.device_apparat.all()

    class Meta:
        verbose_name = 'Устройство'
        verbose_name_plural = 'Устройства'
```

Таблица 5 – описание полей модели Device

Название поля	Описание поля	Служит для хранения:
date_last_config	Дата и время	Даты и времени последнего изменения конфигурации
min_temperature	Число с плавающей точкой	Нижней границы температуры. Используется в качестве значения по умолчанию, если граница на датчике не установлена.

Продолжение таблицы 5

Название поля	Описание поля	Служит для хранения:
max_temperature	Число с плавающей точкой	Верхней границы температуры. Используется в качестве значения по умолчанию, если граница на датчике не установлена.
name	Текстовое поле.	Пользовательского имени управляющего контроллера.
key	Текстовое поле	Ключа оборудования. Он служит для проверки входящей с контроллера информации.

При добавлении информации об управляющем контроллере ключ генерируется автоматически. Исходный код генератора ключа приведен в листинге 3.

Листинг 3 – Исходный код генератора ключей для управляющих контроллеров

```
def get_key():
    chars = '+-
/*!&$#?=@<>abcdefghijklnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'
    key = ''
    for i in range(64):
        key += random.choice(chars)
    return key
```

4.1.4 Sensor

Sensor – модель для хранения информации о датчиках подключенных к контроллерам. Описание полей находится в таблице 6. Исходный код модели приведен в листинге 4.

Листинг 4 – Исходный код модели Sensor

```
class Sensor(models.Model):
    device = models.ForeignKey('Device',
related_name='device',on_delete=models.CASCADE)
    date_last_config = models.DateTimeField(auto_now_add=True)
    min_temperature = models.FloatField('min_temperature', null=True, blank=True)
    max_temperature = models.FloatField('max_temperature', null=True, blank=True)
    name = models.CharField(max_length=250, null=True)
    class Meta:
        verbose_name = 'Датчик'
        verbose_name_plural = 'Датчики'
```

Таблица 6 – описание полей модели Sensor

Название поля	Описание поля	Служит для хранения:
device	Внешний ключ	Информации к какому контроллеру принадлежит датчик
date_last_config	Дата и время	Даты и времени последнего изменения конфигурации
min_temperature	Число с плавающей точкой.	Нижней границы температуры. Не обязательно для заполнения.
max_temperature	Число с плавающей точкой	Верхней границы температуры. Не обязательно для заполнения
name	Текстовое поле.	Пользовательского имени датчика.

4.1.5 DatSensor

DatSensor – модель служит для хранения информации поступающей с датчиков в систему. Описание полей находится в таблице 7. Исходный код модели приведен в листинге 5.

Листинг 5 – Исходный код модели DatSensor

```
class DatSensor(models.Model):
    sensor = models.ForeignKey('Sensor',
related_name='DatSensor',on_delete=models.CASCADE)
    date_input = models.DateTimeField(auto_now_add=True)
    temperature = models.FloatField('temperature')
    min_temperature = models.FloatField('min_temperature')
    max_temperature = models.FloatField('max_temperature')
    alarm = models.BooleanField(default=False)
    class Meta:
        verbose_name = 'Данные с датчика'
        verbose_name_plural = 'Данные с датчиков'
```

Таблица 7 – Описание полей модели DatSensors

Название поля	Описание поля	Служит для хранения:
date_input	Дата и время	Информации о дате и времени поступления информации
temperature	Число с плавающей точкой.	Температура на датчике в момент формирования пакета
min_temperature	Число с плавающей точкой.	Нижней границы температуры имеющейся в контроллере на момент отправки пакета
max_temperature	Число с плавающей точкой	Верхней границы температуры имеющейся в контроллере на момент отправки пакета

Продолжение таблицы 7

Название поля	Описание поля	Служит для хранения:
Alarm	Логическое поле	Выставляется или не выставляется обработчиком внутри системы. В интерфейсе имеет опознавательный знак в виде иконки восклицательного знака.

4.1.6 Apparat

Apparat – модель служащая для хранения информации о аппаратах подключенных к контроллерам. Описание полей находится в таблице 8. Исходный код модели приведен в листинге 6.

Листинг 6 – Исходный код модели Apparat

```
class Apparat(models.Model):
    device = models.ForeignKey('Device',
related_name='apparats',on_delete=models.CASCADE)
    name = models.CharField('Наименование', max_length=100)
    class Meta:
        verbose_name = 'Аппарат'
        verbose_name_plural = 'Аппараты'
```

Таблица 8 – описание полей модели Apparat

Название поля	Описание поля	Служит для хранения:
device	Внешний ключ	Информации к какому контроллеру принадлежит аппарат
name	Текстовое поле.	Пользовательского имени аппарата.

4.1.7 DatApparat

DatApparat – модель служит для хранения информации поступающей об аппаратах с управляющего контроллера. Описание полей находится в таблице 9. Исходный код модели приведен в листинге 7.

Листинг 7 – Исходный код модели DatApparat

```
class DatApparat(models.Model):
    apparat = models.ForeignKey('Apparat',
related_name='DatApparat',on_delete=models.CASCADE)
    date_input = models.DateTimeField(auto_now_add=True)
    status = models.BooleanField()
    class Meta:
        verbose_name = 'Данные с аппарата'
        verbose_name_plural = 'Данные с аппаратов'
```

Таблица 9 – описание полей модели DatApparat

Название поля	Описание поля	Служит для хранения:
apparat	Внешний ключ	Информации к какому аппарату принадлежит информация
date_input	Дата и время	Информации о дате и времени поступления информации
status	Логическое поле	Информации о статусе аппарата (включен или выключен)

4.1.8 ControlApparat

ControlApparat – модель служит для хранения информации о командах посылаемых системой в ручном или автоматическом режиме. Описание полей находится в таблице 10. Исходный код модели приведен в листинге 8.

Листинг 8 – Исходный код модели ControlApparat

```
class ControlApparat(models.Model):
    apparat = models.ForeignKey('Apparat', related_name='ControlApparat',
on_delete=models.CASCADE)
    date_update = models.DateTimeField('Date_update',auto_now_add=True)
    sensor_name = models.CharField(max_length=250, null=True, blank=True)
    sensor_id = models.IntegerField(null=True, blank=True)
    temperature = models.FloatField(null=True, blank=True)
    min_temperature = models.FloatField('min_temperature',null=True, blank=True)
    max_temperature = models.FloatField('max_temperature',null=True, blank=True)
    status = models.BooleanField(default=False)
    class Meta:
        verbose_name = 'Управление апаратом'
        verbose_name_plural = 'Управление апаратами'
```

Таблица 10 – Описание полей модели ControlApparat

Название поля	Описание поля	Служит для хранения:
apparat	Внешний ключ	Информации к какому аппарату принадлежит информация
date_update	Дата и время	Информации о дате и времени поступления команды
sensor_name	Текстовое поле.	Имени датчика показавшего выход за границы
sensor_id	Целое число	Идентификатора датчика показавшего выход за границы
temperature	Число с плавающей точкой.	Температуры на датчике в момент формирования команды

Продолжение таблицы 10

Название поля	Описание поля	Служит для хранения:
min_temperature	Число с плавающей точкой.	Нижней границы температуры имеющейся в контроллере на момент отправки пакета
max_temperature	Число с плавающей точкой	Верхней границы температуры имеющейся в контроллере на момент отправки пакета
status	Логическое поле	Информации о необходимом статусе аппарата (включить или выключить)

4.2 View (Представления)

4.2.1 Обычные представления

Обычные представления вызываются при переходе на определенный url адрес сайта. Для взаимодействия с клиентом описаны следующие представления: oneUser, oneDevice и deviceHistory.

oneUser – представление формирует содержимое на веб странице соответствующее текущем пользователю. А именно выводится список управляющих контроллеров, датчиков и аппаратов, подключенных к ним и последняя поступившая информация о них. Исходный код представления приведен в листинге 9.

Листинг 9 – Исходный код представления oneUser

```
    def oneUser(request):
        if request.user.is_authenticated:
            username = request.user.first_name + ' ' + request.user.last_name
            context = QuerySets.deviceByUser(request.user)
            context.update({'username': username})
            return render(request, 'polls/displayDevice.html', context)
        else:
            return redirect(reverse('mainpage'))
```

Как видно из листинга формирование контекста выполняется функцией QuerySets.deviceByUser. Для удобства запросы выведены в отдельный класс QuerySets. Исходный код функции deviceByUser приведен в листинге 10.

Листинг 10 – Исходный код функции deviceByUser

```
    def deviceByUser(user:User, statussensors=[DatSensor],
statusapparats=[DatApparat]):
        userDeviceList = Device.objects.filter(profile=user.profile)
        statussensors.clear()
        statusapparats.clear()
        for device in userDeviceList:
            statussensors.extend(QuerySets.getLastStatusSensors(device))
            statusapparats.extend(QuerySets.getLastStatusApparat(device))
        return{'all_device_list': userDeviceList,
               'statussensors': statussensors,
               'statusapparats': statusapparats}
```

oneDevice – представление формирует содержимое на веб странице соответствующее выбранному контроллеру. По своей сути является сокращенным вариантом предыдущего представления.

deviceHistory – представление формирует содержимое на веб странице также соответствующее выбранному контроллеру, но показания датчиков и статусы аппаратов выводятся за всё время работы системы.

4.2.2 Формы

Django позволяет генерировать формы в автоматическом режиме по модели. Также есть возможность их настраивать, разбивать на поля и настраивать каждое поле отдельно [21, 22].

4.2.2.1 LoginForm

Первая форма, с которой встречается пользователь это форма аутентификации. Исходный код формы аутентификации приведен в листинге 11.

Листинг 11 – Исходный код формы аутентификации

```
class LoginForm(forms.Form):
    username = forms.CharField(label = '', widget=forms.TextInput(attrs={'class' :
'authform'}))
    password = forms.CharField(label = '', widget=forms.PasswordInput(attrs={'class'
: 'authform'}))
```

То есть создаем класс наследуемый от класса Django – “forms” и указываем в нём два поля из того же класса. Для того чтобы при генерации формы она имела класс, это нужно для css разметки, при создании указываются “attrs”. Помимо класса можно также указать любые css параметры. Но кроме создания формы, необходимо сделать её обработчик. По умолчанию форма при рендеринге её в HTML не имеет тегов “<form>” и кнопки применения. Для её полноценного функционирования необходимо это добавить. Исходный код формы аутентификации в HTML файле приведен в листинге 12.

Листинг 12 – Исходный код формы в HTML файле

```
<form action="." method="post" margin-top="0">
{{ form.as_p }}
{% csrf_token %}
<input type="image" src="{% static 'img/exit.png' %}" height="20px">
</form>
```

Здесь указаны ссылка на саму себя, точнее на страницу вызова формы. Так как остальные ссылки сайта закрыты для не аутентифицированного пользователя, то это работает корректно. Далее в фигурных скобках указывается способ представления формы. Всего на выбор даётся три варианта:

1) As_table –представляет форму в HTML тегах таблицы.
2) As_p - представляет форму в HTML тегах “<p>”, что является обычным текстовым абзацем.
3) As_ul представляет форму в HTML тегах “<ul>”, что является тегами меню.

Кроме того, Django позволяет выводить каждое поле по отдельности. Для этого нужно указать все необходимые параметры в пределах одной формы.

Пример генерации формы, где каждое поле выводится отдельно, приведен в листинге 13.

Листинг 13 – Пример формы аутентификации, где каждое поле выводится отдельно

```
<form action="." method="post" margin-top="0">
{{ form.name }}
{{ form.login }}
{% csrf_token %}
<input type="image" src="{% static 'img/exit.png' %}" height="20px">
</form>
```

В данной работе это будет применено далее.

Следующая строка подключает защиту от «межсайтовой подделки запросов». Это служит для безопасности пользователя, чтобы запрос не был подделан или перенаправлен на сторонние сервера.

И далее необходимо объявить тег <input> который будет служить для перехода к самому представлению. Это может быть ссылка, кнопка, картинка и т.д.

Исходный код представления для аутентификации приведен в листинге 14.

Листинг 14 – Исходный код представления аутентификации

```
def user_login(request):
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            user = authenticate(username=cd['username'], password=cd['password'])
            if user is not None:
                if user.is_active:
                    login(request, user)
                    HttpResponseRedirect('/')
                else:
                    return HttpResponse('Disabled account')
            else:
                return HttpResponse('Invalid login')
    else:
        form = LoginForm()
    if request.user.id:
        username = request.user.first_name+' '+request.user.last_name
    else:
        username = request.user.username
    return render(request, 'MainPage.html', {'form': form, 'username':username})
```

4.2.2.2 ControlPanel

Для каждого управляющего контроллера есть своя панель управления. Она включает в себя множество форм. Эти формы служат для настройки управляющих контроллеров (имя, границы температура). Для каждого датчика привязанного к такому контроллеру (имя, границы). Для каждого аппарата (имя) Также для аппаратов выведена информация о последней команде сформированной системой. И формы формирования этих команд.

Исходный код каждой из используемых форм приведен в листинге 15.

Листинг 15 – Исходный код для форм используемых в панели управления

```
class DeviceForm(forms.ModelForm):
    name = forms.CharField(label='', widget=forms.TextInput(attrs={'class' :
'controlform'}))
    min_temperature = forms.FloatField(widget=forms.TextInput(attrs={'class' :
'controlform'}))
    max_temperature = forms.FloatField(widget=forms.TextInput(attrs={'class' :
'controlform'}))
    date_last_config = forms.DateTimeField()
    class Meta:
        model = Device
        fields = ('__all__')

class SensorForm(forms.ModelForm):
    name = forms.CharField(widget=forms.TextInput(attrs={'class' : 'controlform'}))
    min_temperature =
forms.FloatField(required=False,widget=forms.TextInput(attrs={'class' :
'controlform'}))
    max_temperature =
forms.FloatField(required=False,widget=forms.TextInput(attrs={'class' :
'controlform'}))
    date_last_config =
forms.DateTimeField(required=False,widget=forms.TextInput(attrs={'class' :
'controlform'}))
    class Meta:
        model = Sensor
        fields = ('id','name', 'min_temperature', 'max_temperature')

class ApparatForm(forms.ModelForm):
#    id = forms.IntegerField()
    name = forms.CharField(widget=forms.TextInput(attrs={'class' : 'controlform'}))
    class Meta:
        model = Apparat
        fields = ('name',)

class ControlApparatForm(forms.ModelForm):
    sensor_name = forms.CharField(required=False)
    sensor_id = forms.IntegerField(required=False)
    temperature = forms.FloatField(required=False)
    min_temperature = forms.FloatField(required=False)
    max_temperature = forms.FloatField(required=False)
    status = forms.BooleanField(required=False)
    class Meta:
        model = ControlApparat
```

```
        fields = (
            'sensor_name',
            'sensor_id',
            'temperature',
            'min_temperature',
            'max_temperature',
            'status'
        )
```

Исходный код представлений для обработки этих форм представлен в листинге 16.

Листинг 16 – Исходный код представлений для обработки форм панели управления

```
    class Control():
        def editdevice(request, device_id):
            if request.method == "POST":
                device = get_object_or_404(Device, id=device_id)
                deviceform = DeviceForm(request.POST, instance=device)
                if deviceform.is_valid():
                    device = deviceform.save(commit=False)
                    device.date_last_config = datetime.datetime.now()
                    device.save()
                return HttpResponseRedirect('/polls/controldevice/' +
str(device_id))
            else:
                return HttpResponse('Error (you have NO post request) editdevice')

        def editsensor(request, device_id, sensor_id):
            if request.method == "POST":
                sensor = get_object_or_404(Sensor, id=sensor_id)
                sensorform = SensorForm(request.POST, instance=sensor)
                if sensorform.is_valid():
                    sensor = sensorform.save(commit=False)
                    sensor.date_last_config = datetime.datetime.now()
                    sensor.save()
                return HttpResponseRedirect('/polls/controldevice/' +
str(device_id))
            else:
                return HttpResponse('Error (you have NO post request) editdevice')

        def editapparat(request, device_id, apparat_id):
            if request.method == "POST":
                apparat = get_object_or_404(Apparat, id=apparat_id)
                apparatform = ApparatForm(request.POST, instance=apparat)
                if apparatform.is_valid():
                    apparat = apparatform.save(commit=False)
                    apparat.device = Device.objects.get(id=device_id)
                    apparatform.save()
                return HttpResponseRedirect('/polls/controldevice/' +
str(device_id))
            else:
                return HttpResponse('Error (you have NO post request) editapparat')

        @csrf_exempt
        def controlapparat(request, device_id, apparat_id):
            if request.method == "POST":
                apparatcontrolform = ControlApparatForm(request.POST)
```

```
                if apparatcontrolform.is_valid():
                    apparacontrol = apparatcontrolform.save(commit=False)
                    apparacontrol.date_last_config = datetime.datetime.now()
                    apparacontrol.apparat = Apparat.objects.get(id=apparat_id)
                    apparacontrol.save()
                return HttpResponseRedirect('/polls/controldevice/' +
str(device_id))
            else:
                return HttpResponse('Error (you have NO post request)
controlapparat')

        def controlDeviceForm(request, device_id, sensorforms=[SensorForm],
apparatforms=[ApparatForm],
                              commandapparats=[ControlApparat], sensorids=[0],
apparatids1=[0], apparatids2=[0]):
            if request.user.is_authenticated:
                device = Device.objects.get(id=device_id)
                if device not in Device.objects.filter(profile__user=request.user):
                    return redirect(reverse('mainpage'))
                sensors = Sensor.objects.filter(device=device)
                apparats = Apparat.objects.filter(device=device)
                username = request.user.first_name + ' ' + request.user.last_name
                if request.method == "POST":
                    return HttpResponse('Error (you have post request)
controlDeviceForm')
                else:
                    deviceform = DeviceForm(instance=device)
                    sensorforms.clear()
                    apparatforms.clear()
                    sensorids.clear()
                    commandapparats.clear()
                    apparatids1.clear()
                    apparatids2.clear()
                    for sensor in sensors:
                        sensorforms.append(SensorForm(instance=sensor))
                        sensorids.append(sensor.id)
                    sensorids.reverse()
                    for apparat in apparats:
                        apparatids1.append(apparat.id)
                        apparatids2.append(apparat.id)
                        apparatforms.append(ApparatForm(instance=apparat))
                        commandapparats.append(

ControlApparat.objects.filter(apparat=apparat).order_by('date_update').values(
                                  'apparat__name', 'status', 'date_update').last())
                    apparatids1.reverse()
                    apparatids2.reverse()
                context = {'ControlApparatForm': ControlApparatForm(),
                           'deviceid': device_id,
                           'deviceform': deviceform,
                           'sensorforms': sensorforms,
                           'apparatforms': apparatforms,
                           'commandapparats': commandapparats,
                           'sensorids': sensorids,
                           'apparatids1': apparatids1,
                           'apparatids2': apparatids2,
                           'username': username}
                return render(request, 'polls/controlDevice.html', context)
            return redirect(reverse('mainpage'))
```

4.2.3 Api запросы

Отдельным пунктом в представлениях является API. API это интерфейс общения между программным обеспечением. Для их создания использовались базовые классы Django rest framework [23]. Api запросы воспринимаются системой только POST вида. Форматом запросов был выбран json, так как является самым популярным на данный момент.

В API реализован следующий функционал:

1) Обновление данных (получение текущей информации с управляющего контроллера). Устройство передает свой идентификационный номер и ключ, а также информацию которую необходимо занести в систему. Это может быть информация о текущих показаниях датчиков или о статусе подключенных аппаратов. Исходный код для обработчика информации поступающей о датчиках представлен в листинге 17.

Листинг 17 – Исходный код обработчика api запроса на добавление данных о показаниях датчика

```
    class PostSensorDat(APIView):
        def post(self, request):
            device_id = request.data.get('device_id')
            device_key = request.data.get('device_key')
            if Inspector.checkDeviceKey(device_id, device_key):
                sensor_id_in = request.data.get('sensor_id')
                temperature_in = request.data.get('temperature')
                min_temperature_in = request.data.get('min_temperature')
                max_temperature_in = request.data.get('max_temperature')
                datsensor = DatSensor(sensor_id=sensor_id_in,
temperature=temperature_in, min_temperature=min_temperature_in,
max_temperature=max_temperature_in, alarm=False)
                datsensor.save()
                Inspector.checkRangeFromApiDatSensor(datsensor)
                return HttpResponse('success')
            return HttpResponse('KEY ERROR!')
        def get(self, request):
            return HttpResponse('You must have POST request')
```

В коде можно увидеть вызов класса Inspector. Этот класс отвечает за проверку полученных данных. В данном случае он проверяет данные пришедшие от датчика, а именно выход за границы. Исходный код функции проверки приведен в листинге 18.

Листинг 18 – Исходный код функции проверки данных api запроса PostSensorDat

```
    def checkRangeFromApiDatSensor(datsensor = DatSensor):
        if datsensor.temperature < datsensor.min_temperature or
datsensor.temperature > datsensor.max_temperature:
            device = Device.objects.get(device_sensor__DatSensor=datsensor)
            profilelist =
Profile.objects.filter(devices__profile__devices=device).distinct()
            if len(profilelist) > 1:
                for profile in profilelist:
                    email = User.objects.get(profile=profile).email
                    Notifications.sendmail(email)
            else:
                email = User.objects.get(profile=profilelist.first()).email
                Notifications.sendmail(email)
            datsensor.alarm = True
            datsensor.save()
```

Также при выходе за границы вызывается класс Notifications. Этот класс отвечает за оповещение, в данном случае отправка сообщений на адрес электронной почты.

Исходный код функции отправки сообщений приведен в листинге 19.

Листинг 19 – Исходный код функции отправки сообщений

```
    def sendmail(email):
        data = """
        Внимание, один из ваших датчиков зафиксировал выход за границы температуры,
обратите внимание!
            """
        send_mail('Alarm!', data, "TermoControl", [email], fail_silently=False)
```

Для того чтобы данная функция заработала необходимо подключить библиотеку send_mail и в настройках указать SMTP сервер и параметры подключения, такие как логин и пароль.

Исходный код настроек для взаимодействия с Google Mail приведен в листинге 20.

Листинг 20 – Исходный код настроек почтового клиента для Google Mail

```
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'addres@gmail.com'
EMAIL_HOST_PASSWORD = '*********'
DEFAULT_FROM_EMAIL = 'addres1@gmail.com'
```

Исходный код для обработчика информации поступающей об аппаратах представлен в листинге 21.

Листинг 21 – Исходный код обработчика api запроса на добавление данных о статусе аппарата.

```
    class PostApparatDat(APIView):
        def post(self, request):
            device_id = request.data.get('device_id')
            device_key = request.data.get('device_key')
            if Inspector.checkDeviceKey(device_id, device_key):
                apparat_id_in = request.data.get('apparat_id')
## относится ли к аппарату
                status_in = request.data.get('status')
                datapparat = DatApparat(apparat_id=apparat_id_in, status=status_in)
###проверка на адекватность введенных значений
                datapparat.save()
                return HttpResponse('success')
            return HttpResponse('KEY ERROR!')
        def get(self, request):
            return HttpResponse('You must have POST request')
```

2) Получение данных о собственной конфигурации. Устройство выполняет запрос используя свой идентификатор и ключ. Система в ответ формирует ответ о том, какие границы имеет каждый из датчиков и сам контроллер. Также в ответ приходит информации о последней команде поступившей для аппаратов. Исходный код обработчика api запроса на получение конфигурации представлен в листинге 22.

Листинг 22 – Исходный код обработчика api запроса на получения конфигурации

```
class GetCofigDevice(APIView):
    def post(self,request):
        device_id = request.data.get('device_id')
        device_key = request.data.get('device_key')
        if Inspector.checkDeviceKey(device_id,device_key):
            device = Device.objects.filter(id=device_id)
            sensors = Device.objects.get(id=device_id).get_sensors()
            aparats = Device.objects.get(id=device_id).get_apparats()
            results_list = list(chain(device, sensors))
            for apparat in aparats:
                app_last_cmd = apparat.get_last_command()
                results_list.append(app_last_cmd)
            results = list()
            for entry in results_list:
                item_type = entry.__class__.__name__.lower()
                if isinstance(entry, Device):
                    serializer = DeviceSerializers(entry)
                if isinstance(entry, Sensor):
                    serializer = SensorSerializers(entry)
```

```
                if isinstance(entry, ControlApparat):
                    serializer = ControlApparatSerializers(entry)
                results.append({'type': item_type, 'data': serializer.data})
            return HttpResponse(json.dumps(results),content_type='application/json')
        return HttpResponse('KEY ERROR!')
    def get(self, request):
        return HttpResponse('You must have POST request')
```

4.3 Controller

Для того чтобы приложение понимала какое из представлений вызвать она использует url ссылки. Для удобства каждая из ссылок прописывается не отдельно, а создаются паттерны по виду этих ссылок [20, 21]. Так любой запрос попадающий на сайт сначала обрабатывается в корневом файле проекта с названием «urls.py»

Исходный код файла urls.py представлен в листинге 23.

Листинг 23 – Исходный код файла urls.py

```
from django.contrib import admin
from django.urls import path, include
import mysite.views

urlpatterns = [

    path('', mysite.views.user_login, name='mainpage'),
    path('logout/', mysite.views.logout_view, name='logout'),
    path('admin/', admin.site.urls),
    path('polls/', include('polls.urls')),
]
```

В этом файле указано 4 паттерна:

1) ' ' – пустой. Это все запросы, которые адресуется к корню сайта. Например, example.com или 192.168.0.1. В данном случае при получении такого запроса открывается форма аутентификации на главной странице.

2) 'logout/' – выход. Это запрос, имеющий следующую структуру: example.com/logout/. В данном случае при получении такого запроса пользователь выходит из системы.

3) 'admin/' - панель администратора. При получении такого запроса подключаются ссылки, используемые в Django по умолчанию в административной панели.

4) 'polls/' – так как фреймворк Django подразумевает множество приложений, то в данном случае это все запросы адресованные приложению

«polls». В данном случае он подключает другой файл «urls.py» находящийся в директории с приложением.

Контроллер «polls» - это контроллер именно приложения. Его исходный код представлен в листинге 24.

Листинг 24 – Исходный код контроллера приложения polls

```
from django.urls import path
from . import views
from .api import Api
from .formhandler import FormHandler

app_name = 'polls'
urlpatterns = [

    path('api/getdconfigdevice/', Api.GetCofigDevice.as_view()),
    path('api/postsensordat/', Api.PostSensorDat.as_view()),
    path('api/postapparatdat/', Api.PostApparatDat.as_view()),

    path('user/', views.ShowDevice.oneUser, name='oneUser'),
    path('device/<int:device_id>/', views.ShowDevice.oneDevice, name='deviceInfo'),
    path('devicehistory/<int:device_id>/', views.ShowDevice.deviceHistory,
name='deviceHistory'),

    path('controldevice/<int:device_id>/', FormHandler.Control.controlDeviceForm,
name='controlDeviceForm'),
    path('editdevice/<int:device_id>/', FormHandler.Control.editdevice,
name='editdevice'),
    path('editsensor/<int:device_id>/<int:sensor_id>/',
FormHandler.Control.editsensor, name='editsensor'),
    path('editapparat/<int:device_id>/<int:apparat_id>/',
FormHandler.Control.editapparat, name='editapparat'),
    path('commandapparat/<int:device_id>/<int:apparat_id>/',
FormHandler.Control.controlapparat, name='controlapparat'),
]
```

Здесь описаны паттерны для всех представлений, в том числе обработчиков для форм.

5 ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ

Взаимодействие с пользователем начинается с главной страницы (рисунок 8).

Рисунок 8 – Главная страница сайта

Главная страница является обложкой для всего сайта. Предполагается, что здесь будет размещаться информация о компании, некоторые преимущества и контакты. Пока пользователь не аутентифицирован - другие страницы ему не

доступны и при попытке перехода в ответ он получит ту же самую страницу. В левом верхнем углу расположены два поля для ввода и иконка двери. Для аутентификации необходимо заполнить поля и нажать на иконку (рисунок 9).

Рисунок 9 – Поля аутентификации

После того как эти действия будут выполнены в зависимости от введенных данных система выдаст сообщение о том что аутентификация не выполнена из-за ошибки. Либо, в случае успеха главная страница изменится и в левом верхнем углу будет отображаться имя и фамилия аутентифицированного пользователя (рисунок 10).

Рисунок 10 – Имя аутентифицированного пользователя

На странице также отображается иконка двери, но в данном случае она несёт противоположный смысл, а именно выход из системы.

Пользователи делятся на две основных группы: клиенты и администраторы. Взаимодействие клиентов с сервисом происходит через раздел «Мои девайсы». С администратором через административную панель, доступную по адресу «/admin» (рисунок 11).

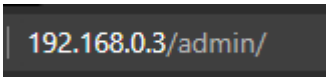

Рисунок 11 – Адрес панели администратора

5.1 Клиентский интерфейс

Интерфейс клиентов включает в себя четыре основных раздела:

– Мониторинг всех своих устройств. В данном разделе располагается перечень управляющих контроллеров с подключенными к ним датчиками температуры и аппаратами климатического контроля. Помимо их наличия в интерфейсе указаны последние данные с этих устройств (рисунок 12).

Для датчиков это последние поступившие данные о температуре, верхней и нижней границах температуры указанных на устройстве в момент формирования запроса. Дата и время обработки запроса системой, а также пользовательское имя датчика.

Для аппаратов это последние поступившие данные о статусе аппарата (включен или выключен). Дата и время обработки запроса системой и пользовательское имя устройства.

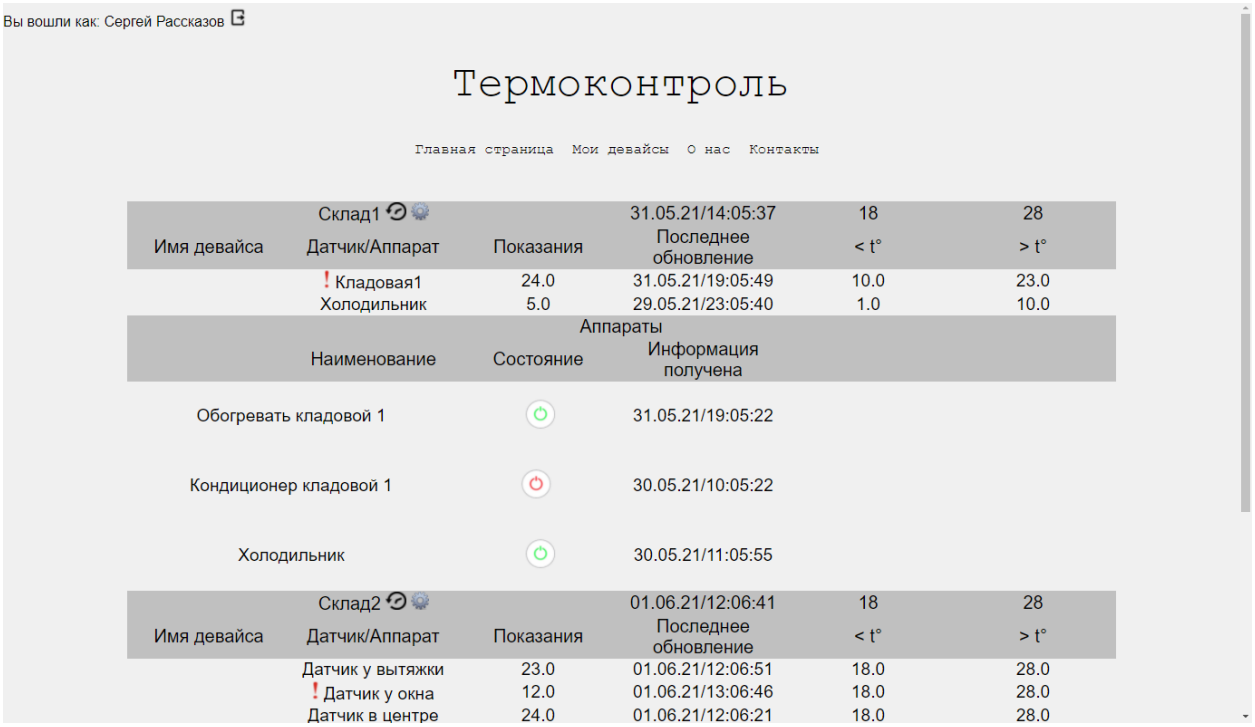

Рисунок 12 – Раздел «Мои девайсы»

– Мониторинг одного контроллера. Раздел является сокращенной формой «мониторинга всех контроллера» и представляет информацию только об одном управляющем контроллере (рисунок 13).

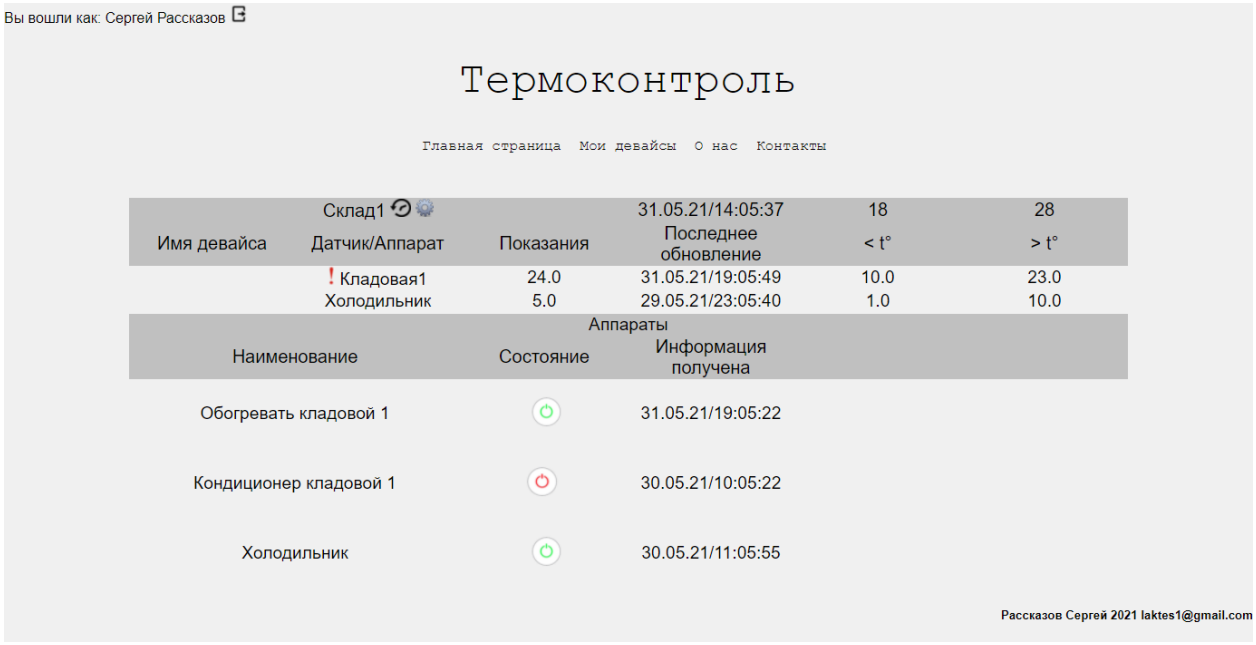

Рисунок 13 – Информация об одном конкретном контроллере

– История. Раздел является расширенной версией раздела «Мониторинг одного контроллера». Разница заключается в том, что отображается не только последние пришедшие данные, а все пришедшие данные за всё время работы системы (рисунок 14).

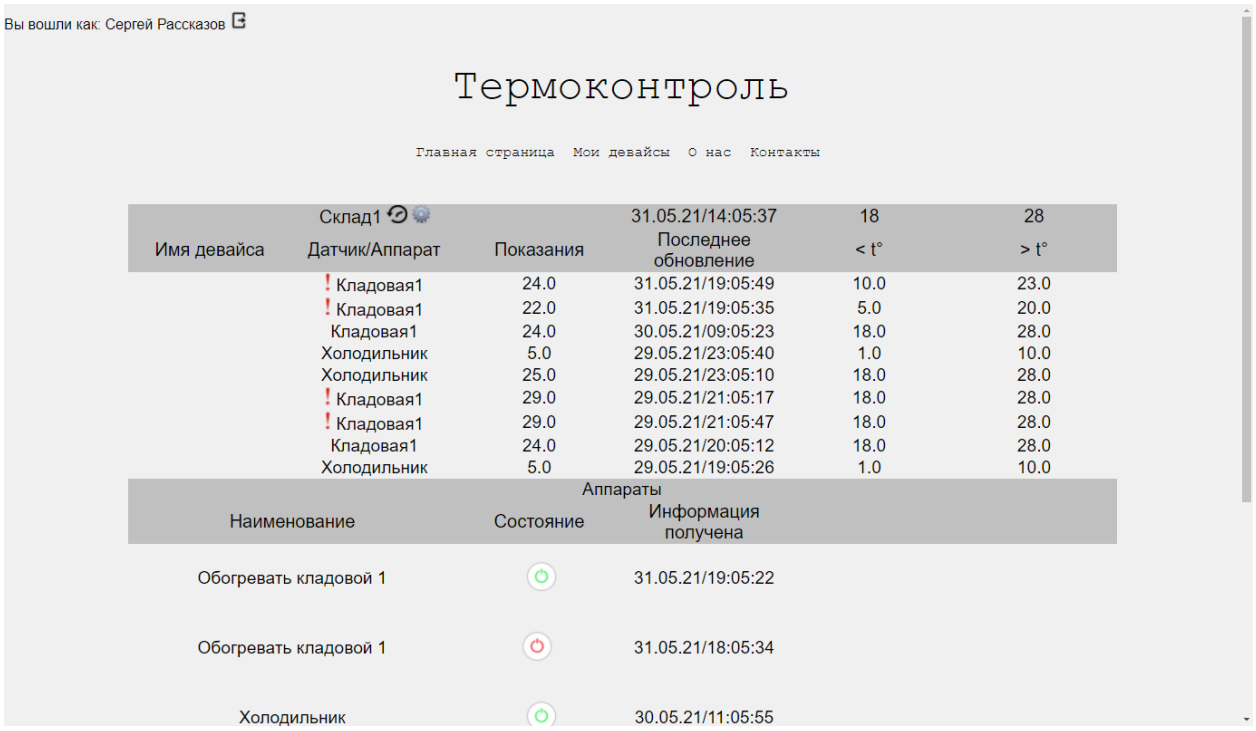

Рисунок 14 – История состояний одного конкретного контроллера

– Панель управления. Раздел представляет из себя перечень форм привязанных к одному «управляющему контроллеру». Для него форма позволяет

изменять его пользовательское имя, верхнюю и нижнюю границы температуры. Для датчиков формы позволяют сделать аналогичные действия. Для аппаратов позволяют изменить статус (включить или выключить) на уровне системы и отображает последний статус, который был изменен из панели управления (рисунок 15).

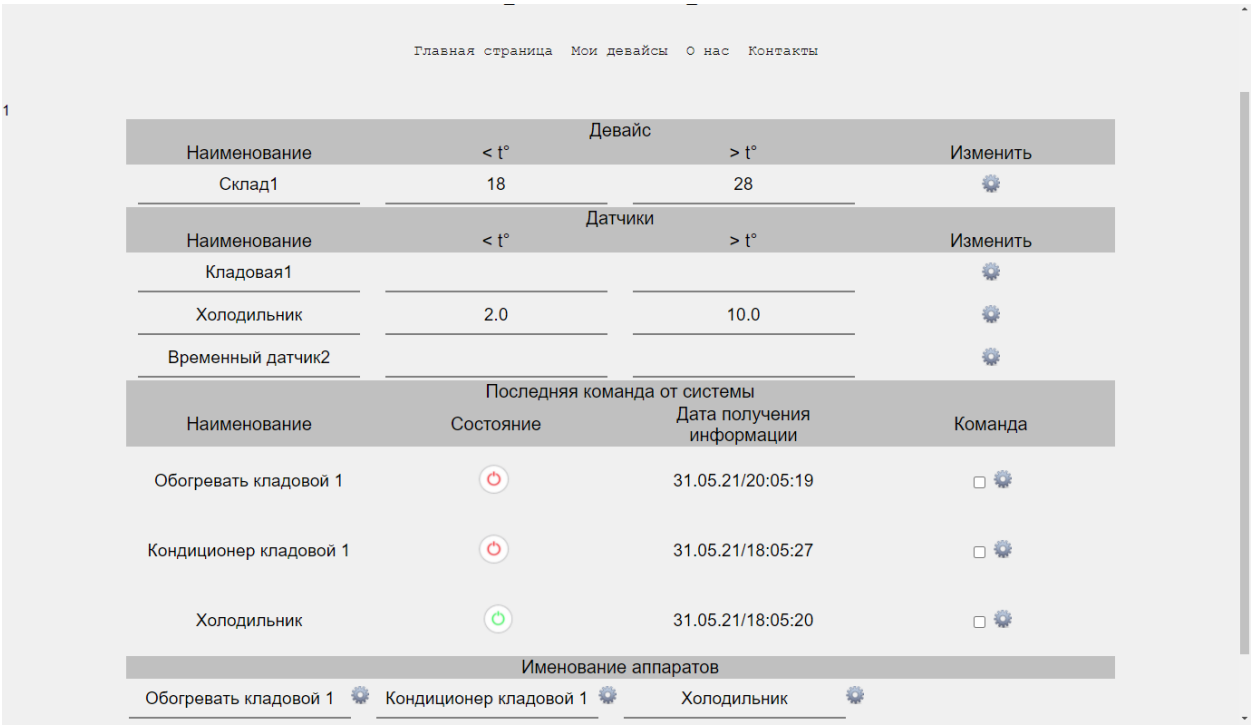

Рисунок 15 – Панель управления одним конкретным контроллером

5.2 Интерфейс администратора

Так как приложение было написано с помощью фреймворка «Django», то панель администратора осталась не тронутой [22].

Интерфейс администратора предоставляет доступ ко всем таблицам базы данных с возможностью их корректировки, сортировки, фильтрации и тому подобные манипуляции (рисунок 16).

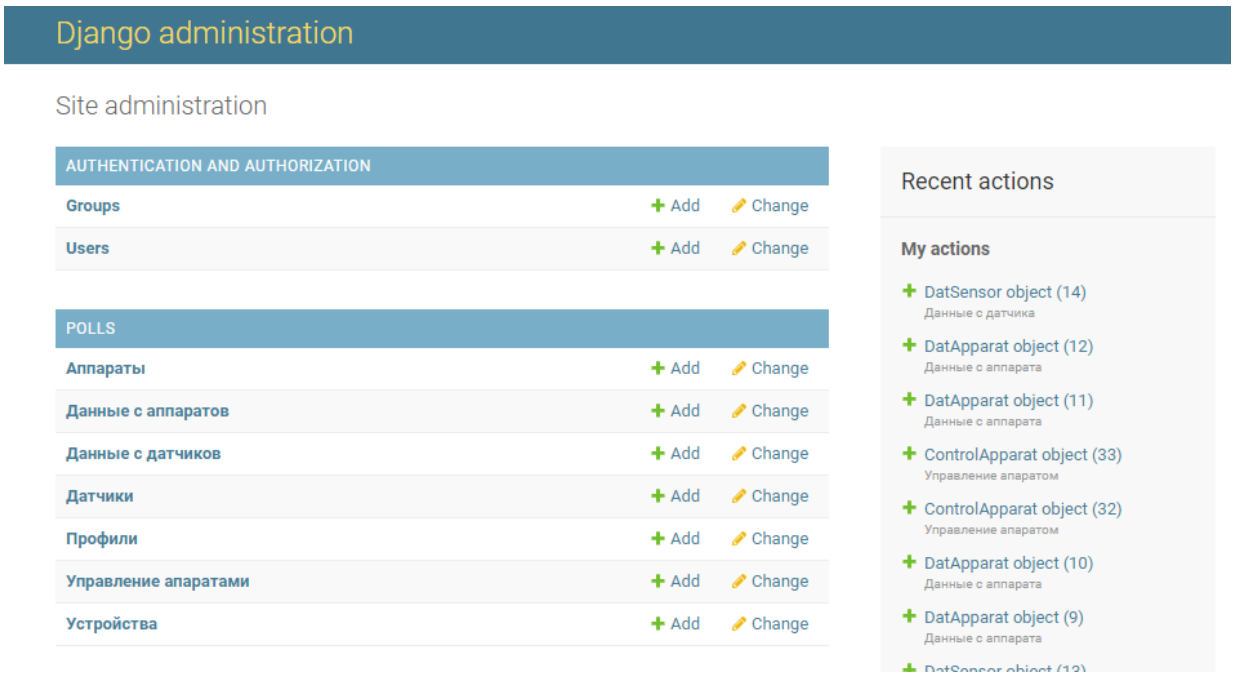

Рисунок 16 – Модели доступные администратору

Далее представлены некоторые из них.

5.2.1 Users

Интерфейс содержит имя пользователя, адрес электронной почты, имя и фамилию, а также информацию о том, является ли пользователь сотрудником. На данной странице также можно фильтровать пользователей по трём параметрам: сотрудник, суперпользователь и активный. Присутствует возможность поиска по отображаемым полям (рисунок 17)..

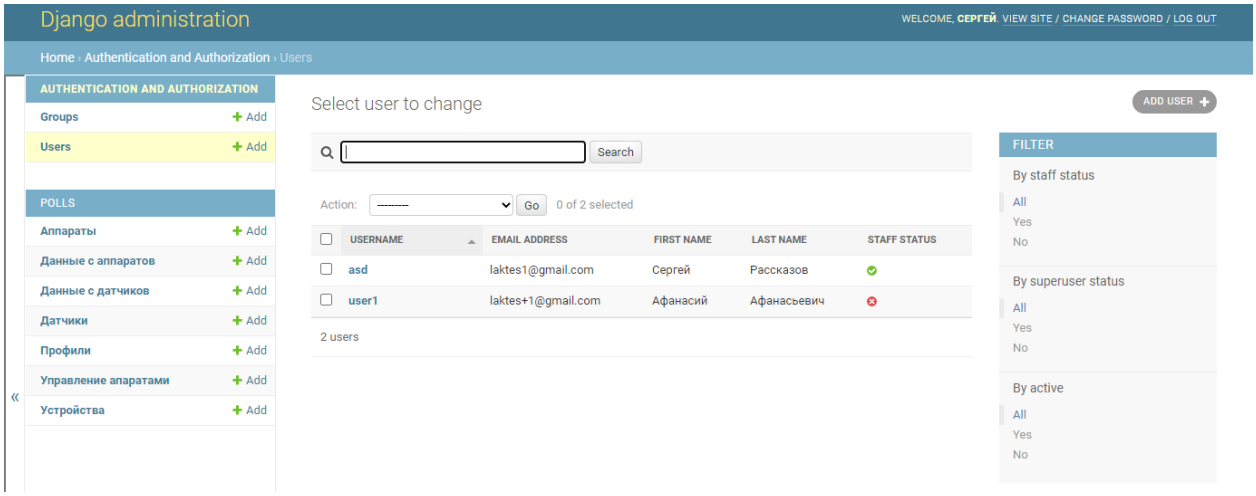

Рисунок 17 – Интерфейс администратора при работе с экземплярами модели User

Добавление пользователя состоит из двух этапов.

1) Создание логина и пароля. Стоит обратить внимание на то, что пароль должен быть достаточно сложным для его принятия системой и не должен совпадать с логином (рисунок 18).

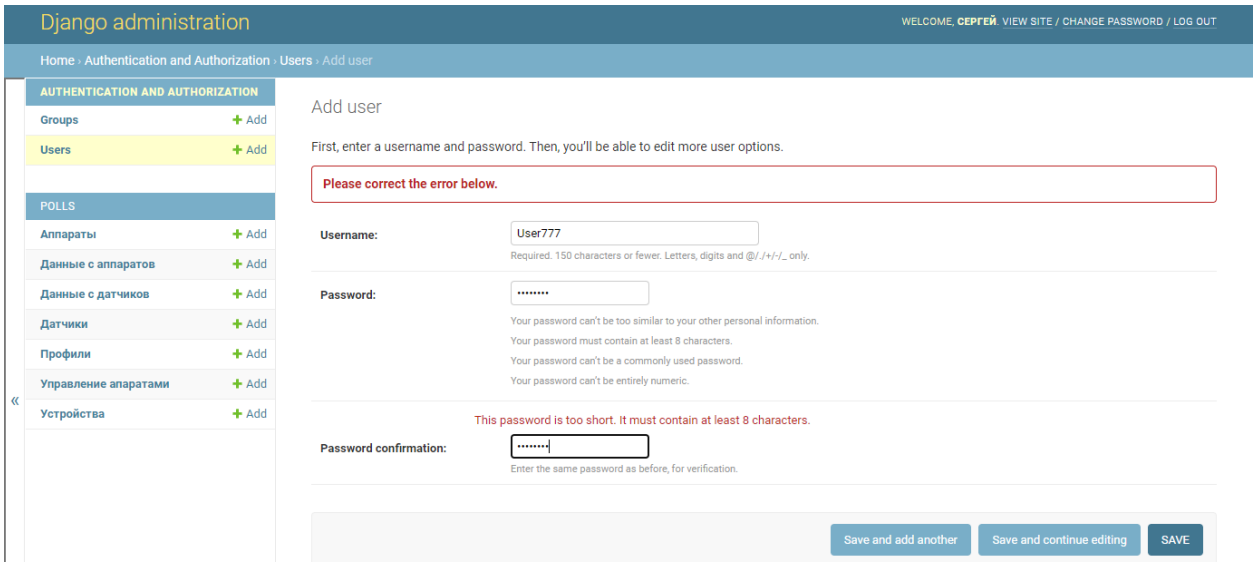

Рисунок 18 – Первый этап регистрации пользователя

2) Заполнение данных об имени, фамилии, почтовом адресе. Так же есть дополнительные параметры, такие как:

– активный пользователь;
– сотрудник;
– суперпользователь.

Данные параметры имеют, как правило, служебную нагрузку, и используются внутри системы для идентификации и аутентификации пользователей и их прав (рисунок 19).

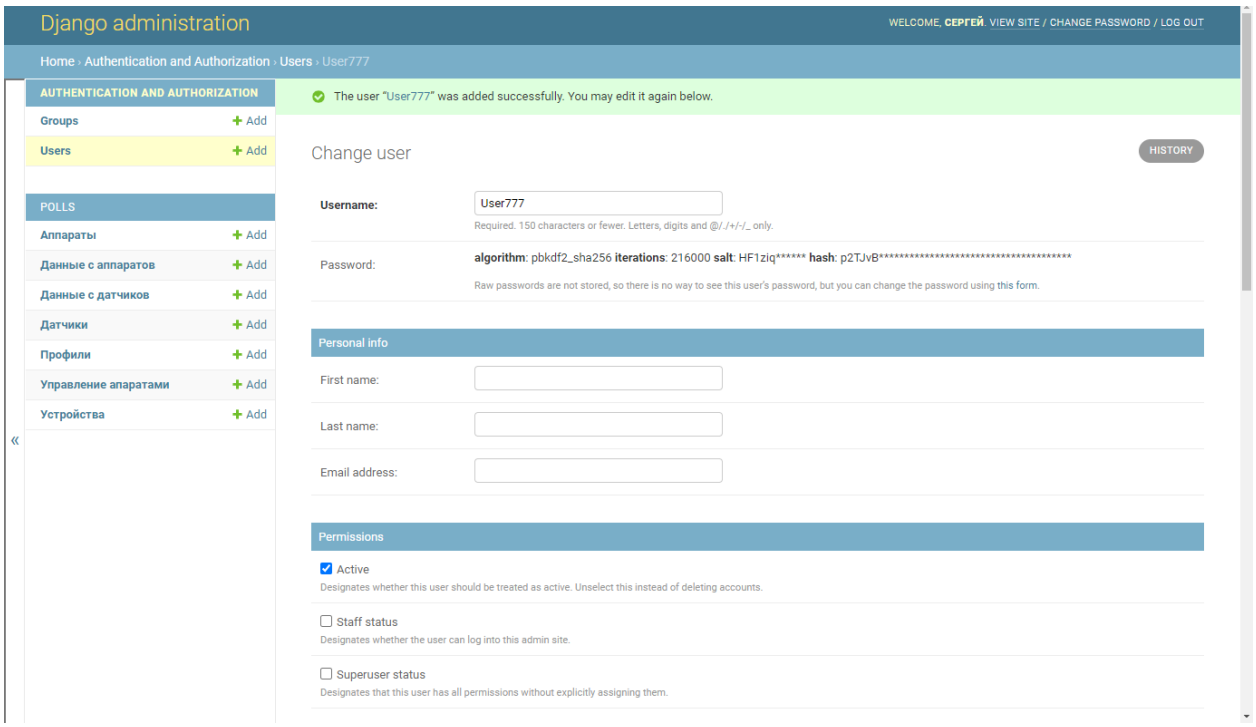

Рисунок 19 – Второй этап регистрации пользователя

5.2.2 Устройства (управляющие контроллеры)

Управляющие контроллеры это основной элемент приложения, на котором завязано всё остальное. В интерфейсе отображается имя контроллера, его идентификационный номер. Последнюю дату изменения настроек и температурные границы для датчиков. Также есть возможность фильтрации и поиска по указанным полям (рисунок 20).

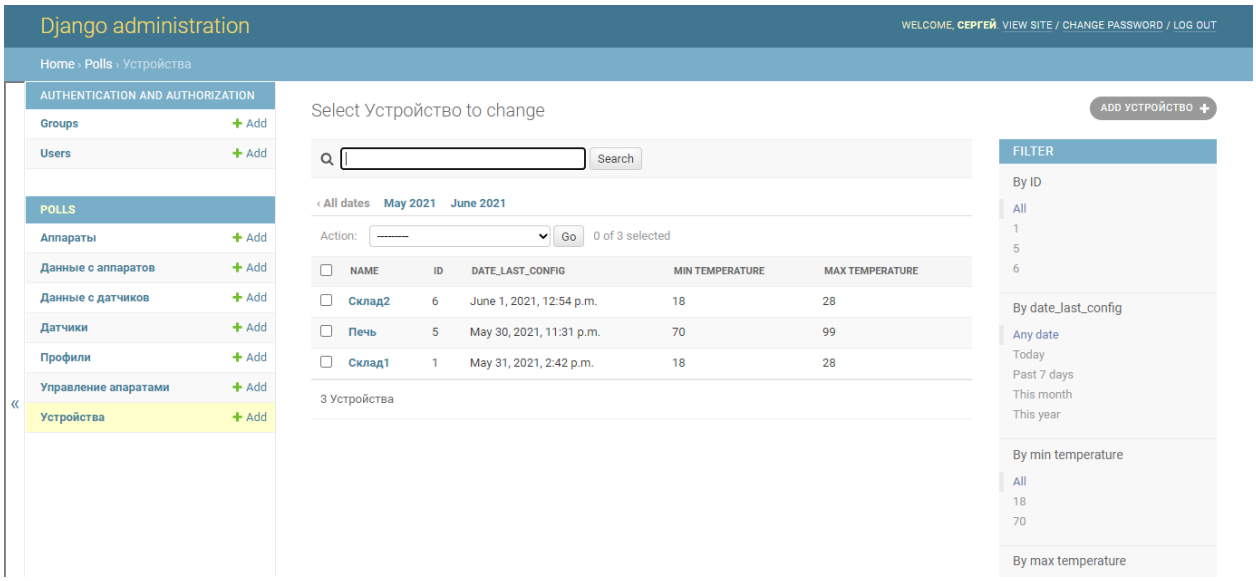

Рисунок 20 – Интерфейс администратора при работе с экземплярами модели Device

Исходный код настроек для данной страницы представлен в листинге 25.

Листинг 25 – Исходный код настроек для страницы работы с экземплярами модели Device

```
@admin.register(Device)
class DeviceAdmin(admin.ModelAdmin):
    list_display = ('name','id','date_last_config',
'min_temperature','max_temperature')
    list_filter = ('id','date_last_config', 'min_temperature','max_temperature')
    search_fields = ('id','datetimeConf', 'min_temperature','max_temperature')
```

Из листинга видно, что объявлены поля, которые будут выводиться в панели. Перечень полей, по которым может осуществляться фильтрация и поиск

В правом верхнем углу находится кнопка «ADD УСТРОЙСТВО», при её нажатии осуществляется переход на страницу добавления управляющего контроллера (рисунок 21).

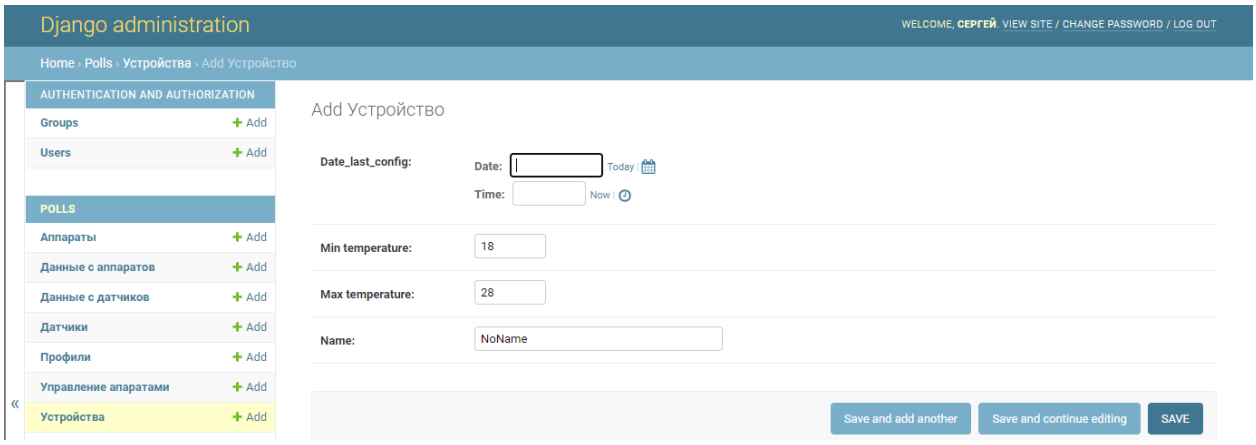

Рисунок 21 – Страница добавления управляющего контроллера

Страница добавления управляющего контроллера имеет поля, которые были ранее заданы в модели. Также из скриншота видно, что некоторые поля уже заполнены. Это следствие выставления в модели значений по умолчанию. Кнопки «Today» и «Now» заполняют соответствующие поля текущей датой и временем.

5.2.3 Профили

В интерфейсе отображены следующие поля: имя пользователя, к которому привязан профиль, телефонный номер и уровень доступа. Также есть возможность фильтрации и поиска по указанным полям (рисунок 22).

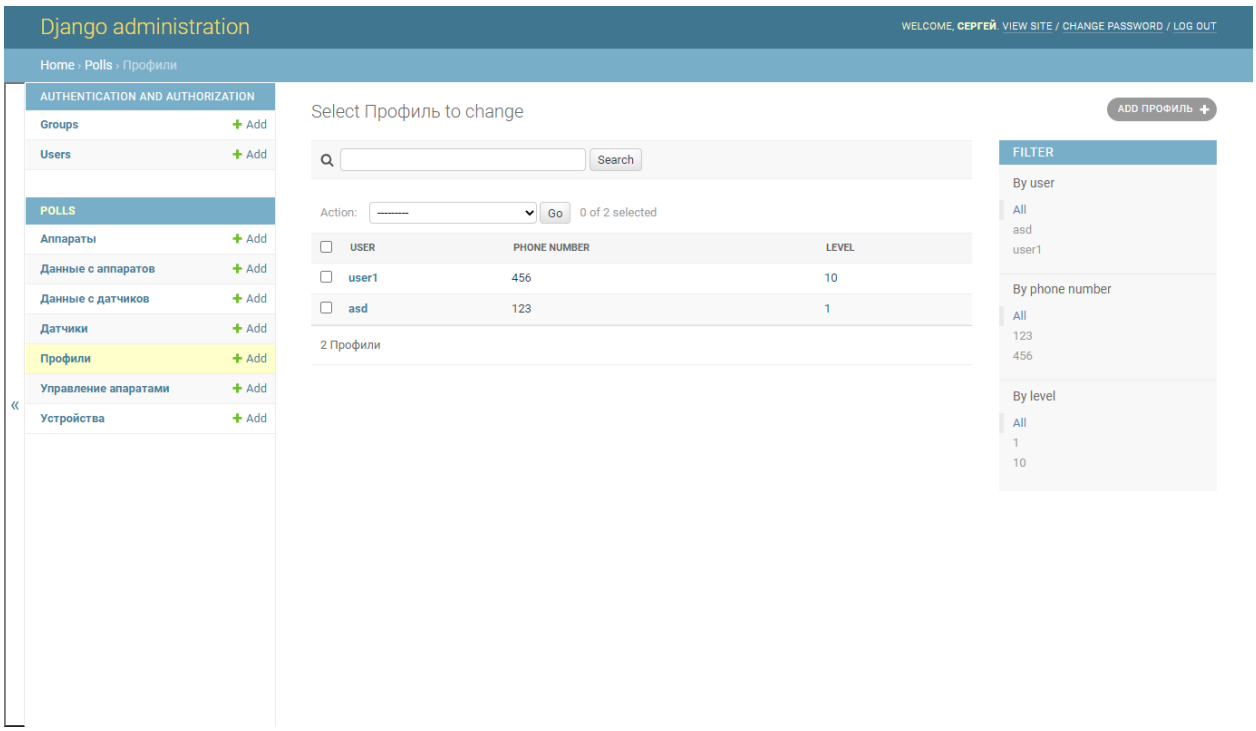

Рисунок 22 - Интерфейс администратора при работе с экземплярами модели Profile

Исходный код настроек для данной страницы представлен в листинге 26.

Листинг 26 – Исходный код настроек для страницы работы с экземплярами модели Profile

```
@admin.register(Profile)
class ProfileAdmin(admin.ModelAdmin):
    list_display = ('user', 'phonenumber','level')
    list_filter = ('user', 'phonenumber','level')
    search_fields = ('user__username', 'phonenumber')
    filter_horizontal = ("devices","devices")
    save_on_top =  True
```

В правом верхнем углу находится кнопка «ADD ПРОФИЛЬ», при её нажатии осуществляется переход на страницу добавления профиля (рисунок 23).

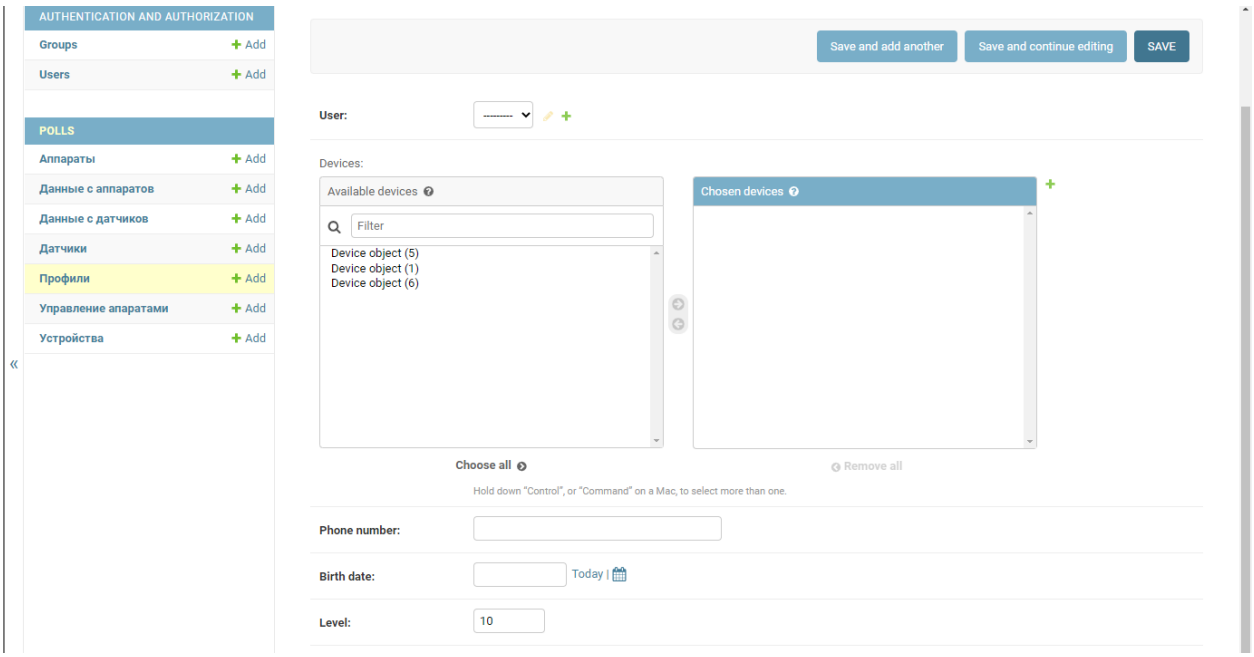

Рисунок 23 – Страница добавления профиля

Страница добавления профиля имеет поля, которые были ранее заданы в модели. Для добавления профиля необходимо выбрать пользователя, к которому профиль будет привязан. Далее выбираются контроллеры, владельцем которых, является пользователем. Указывается номер телефона, дата рождения и уровень доступа. Кнопка «Today» заполняет соответствующее поле текущей датой.

5.2.4 Датчики

В интерфейсе отображены следующие поля: идентификатор датчика, его название, контроллер к которому он относится, дата и время последнего изменения конфигурации, и температурные границы. Границы могут быть не заданы, тогда данные нужно брать из привязанного контроллера (рисунок 24).

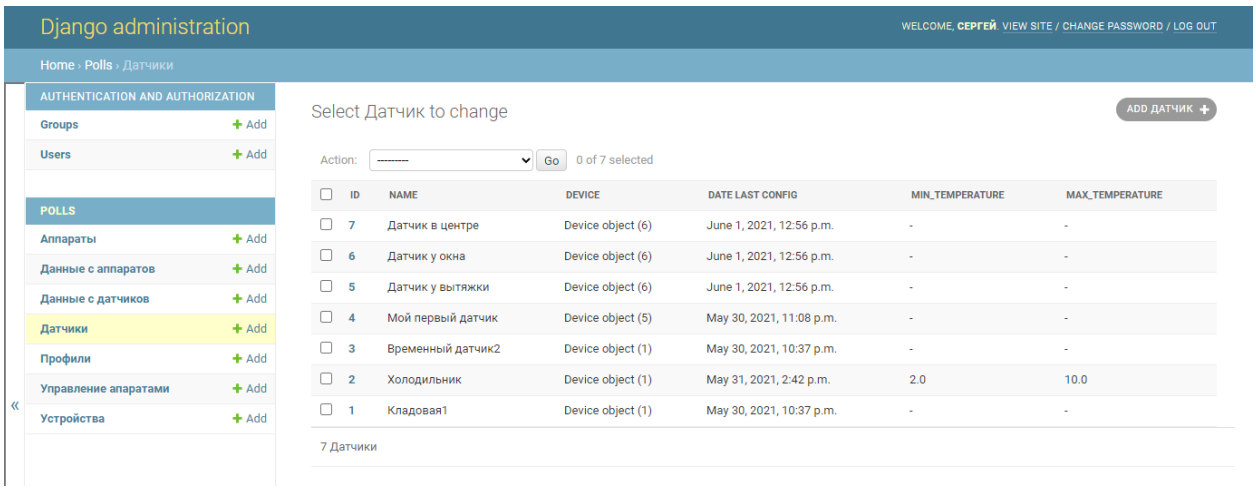

Рисунок 24 - Интерфейс администратора при работе с экземплярами модели Sensor

В правом верхнем углу находится кнопка «ADD ДАТЧИК», при её нажатии осуществляется переход на страницу добавления датчика (рисунок 25).

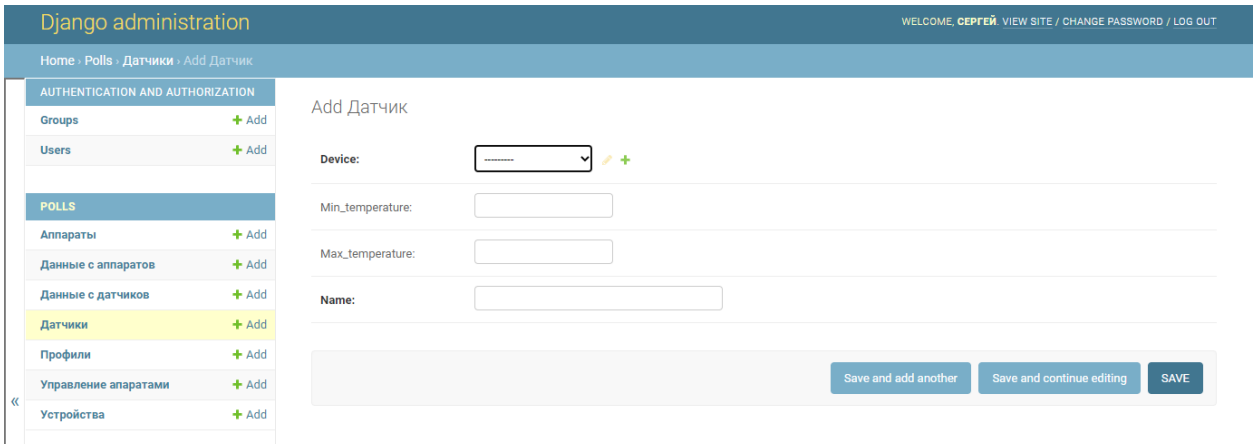

Рисунок 25 - Страница добавления датчика

Страница добавления датчика имеет поля, которые были ранее заданы в модели. Для добавления датчика необходимо выбрать контроллер, к которому будет привязан датчик. Далее можно ввести температурные границы, но это не обязательно и ввести имя датчика.

5.2.5 Аппараты

В интерфейсе отображены следующие поля: идентификатор аппарата, его название, контроллер к которому он относится (рисунок 26).

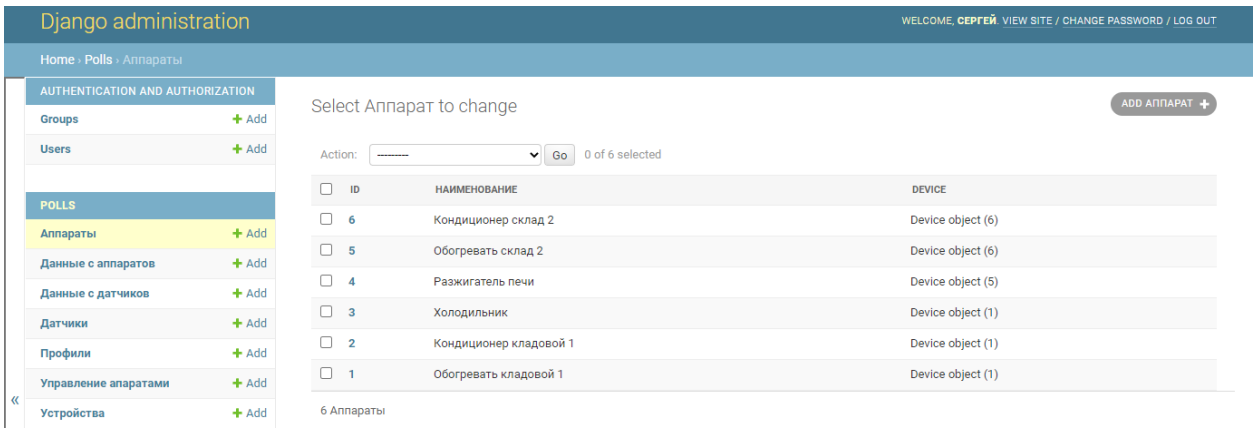

Рисунок 26 - Интерфейс администратора при работе с экземплярами модели Apparat

В правом верхнем углу находится кнопка «ADD АППАРАТ», при её нажатии осуществляется переход на страницу добавления аппарата (рисунок 27).

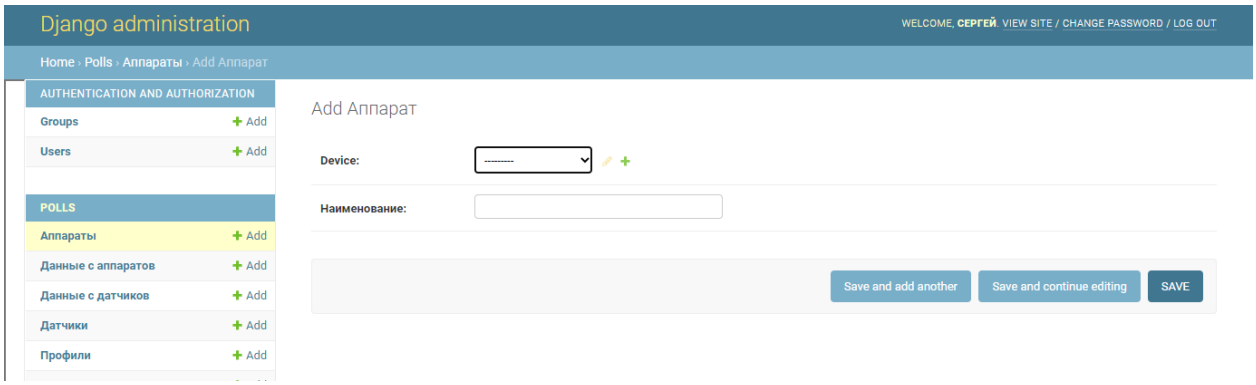

Рисунок 27 – Страница добавления аппарата

Страница добавления датчика имеет поля, которые были ранее заданы в модели. Для добавления аппарата необходимо выбрать контроллер, к которому будет привязан аппарат. Далее необходимо ввести имя датчика.

6 ТЕСТИРОВАНИЕ

6.1 Тестирование формы аутентификации

Для построения форм с помощью фреймворка используются встроенные виджеты. Форма аутентификации имеет два поля, это логин и пароль. Для логина используется виджет InputText без дополнительных параметров. Это значит, что поле может принять любой символ. По умолчанию все поля обязательны для заполнения.

Данные формы не будут отправлены, пока оба поля не заполнены (рисунок 28 и 29).

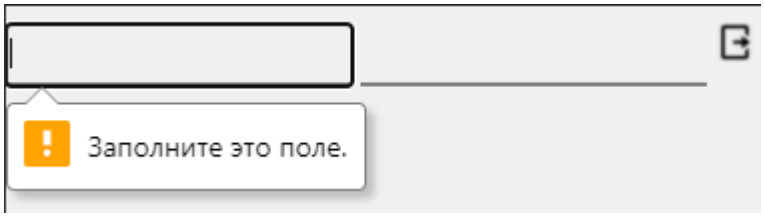

Рисунок 28 – Пустое поле ввода логина

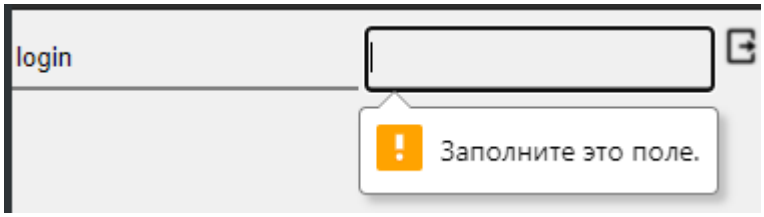

Рисунок 29 – Пустое поле ввода пароля

При вводе неверных данных отображается пустая страница с сообщением об ошибке (рисунок 30).

Рисунок 30 – Сообщение о неверном логине или пароле

Если логин и пароль введены правильно, то страница обновляется и в верхнем левом углу указано имя и фамилия идентифицированного пользователя (рисунок 31).

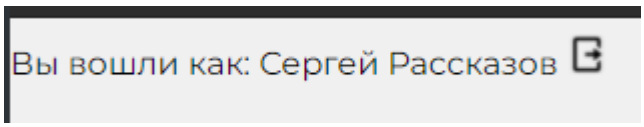

Рисунок 31 – Результат ввода корректного логина и пароля

6.2 Тестирование доступности разделов

Система доступа к разделам сайта сделана таким образом, что не авторизированный пользователь не может увидеть никакую страницу, кроме главной. При попытке загрузить раздел «Мои девайсы» или страниц настройки устройства, его истории и последних данных запрос перенаправляется на главную страницу.

Для авторизированного пользователя все эти разделы доступны, но осуществляется проверка на связь между пользователем и устройством. То есть если устройство не принадлежит пользователю, то его также перенаправит на главную страницу.

6.3 Тестирования отображения последних данных и всех данных

Проведено тестирование системы мониторинга. Отправлены тестовые данные о состоянии датчиков и аппаратов и проведена сверка с содержимым соответствующих страниц. Все данные отображаются корректно.

6.4 Тестирование форм конфигурирования управляющего контроллера

Также как и форма аутентификации, формы конфигурации сгенерированы фреймворком Django.

В панели используется несколько основных полей для ввода, это

- текстовое поле;
- поле для ввода чисел с плавающей запятой;
- «Checkbox»;

В текстовое поле можно поместить любые символы, их количество ограничено 250. В поле для ввода чисел с плавающей точкой можно поместить целое число, оно будет преобразовано в число с плавающей точкой. Также вместо знака «точка» для разделения можно использовать знак «запятая», в любом случае

данные будут преобразованы в необходимый формат. Поле типа «Checkbox» - является выбором типа команды (включение и выключение).

Поля обязательные для заполнения: наименования устройств, датчиков и аппаратов. Температурные границы для управляющего контроллера.

Поля не обязательные для заполнения: температурные границы для датчиков, статусы аппаратов.

6.5 Тестирование api запросов

Тестирование api запросов через веб браузер не представляется возможным, так как напрямую можно отправить только get запросы, а в работе запросов используется метод post.

Для решения данной задачи можно использовать различное программное обеспечение. Например, для браузера Google Chrome есть приложение с названием «Advanced REST client». Оно предоставляет возможность отправлять запросы различных типов и различного содержания (рисунок 32).

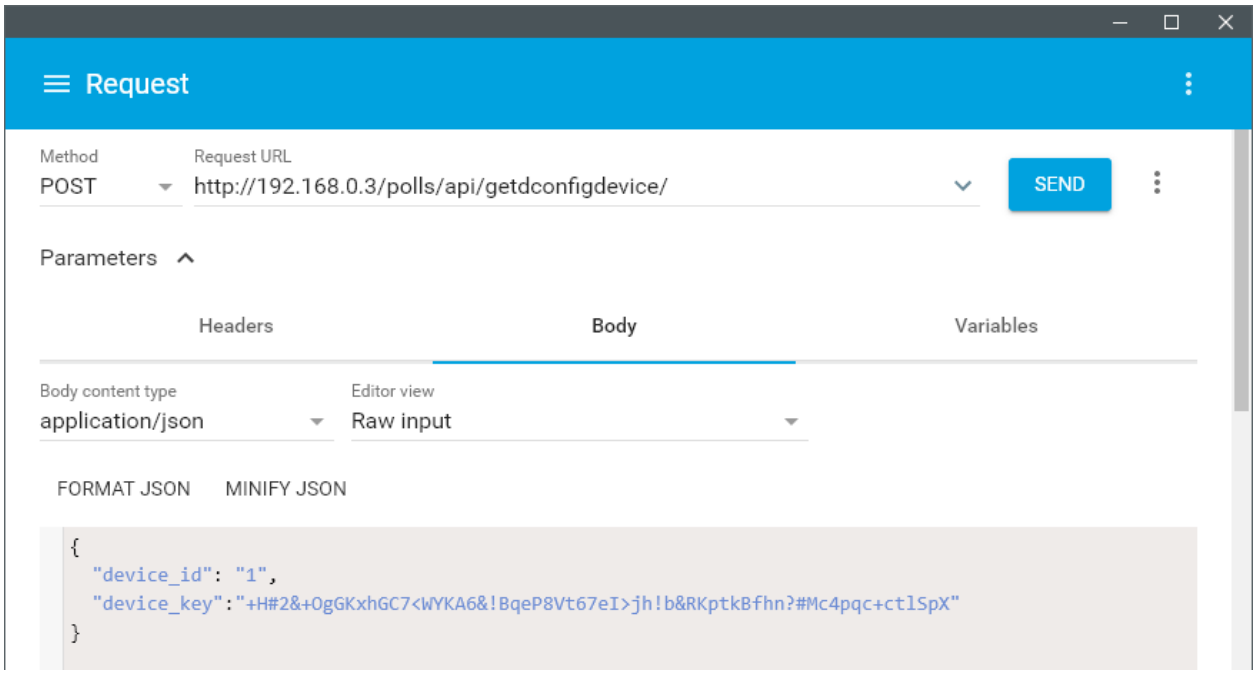

Рисунок 32 – Интерфейс приложения Advanced REST client

В данной работе используется 3 вида api запросов:

1) Получение конфигурации

Запрос должен быть отправлен по адресу: «/polls/api/getdconfigdevice/». Иметь формат «json» и содержать два поля, это идентификатор устройства и его ключ. Эти поля служат для идентификации устройства и защиты информации. Пример запроса для устройства, имеющего идентификатор «1» представлен в листинге 27.

Листинг 27 – Пример запроса конфигурации для устройства с идентификатором «1»

```
{
  "device_id": "1",
  "device_key":"+H#2&+OgGKxhGC7<WYKA6&!BqeP8Vt67eI>jh!b&RKptkBfhn?#Mc4pqc+ctlSpX"
}
```

Ответом на данный запрос является текст также в json формате содержащий информацию о температурных границах устройства, каждого датчика и данные о последней сформированной системой команде для каждого аппарата. Пример ответа на данный запрос представлен в листинге 28.

Листинг 28 – Пример ответа на запрос конфигурации для устройства

```
{
"type": "device",
"data": {
"id": 1,
"date_last_config": "2021-05-31T14:42:37.853163+05:00",
"min_temperature": 18,
"max_temperature": 28,
"name": "Склад1"
}
},
  {
"type": "sensor",
"data": {
"id": 1,
"date_last_config": "2021-06-04T11:19:03.239950+05:00",
"min_temperature": 1,
"max_temperature": null,
"name": "экспериментальный"
}
},
  {
"type": "sensor",
"data": {
"id": 2,
"date_last_config": "2021-06-01T22:16:01.865878+05:00",
"min_temperature": 2,
"max_temperature": 10,
"name": "Холодильник"
}
},
```

```
  {
"type": "sensor",
"data": {
"id": 3,
"date_last_config": "2021-05-30T22:37:32.372280+05:00",
"min_temperature": null,
"max_temperature": null,
"name": "Временный датчик2"
}
},
  {
"type": "controlapparat",
"data": {
"apparat": {
"id": 1,
"name": "Обогревать кладовой 1"
},
"status": true,
"date_update": "2021-06-01T22:16:05.326699+05:00"
}
},
  {
"type": "controlapparat",
"data": {
"apparat": {
"id": 2,
"name": "Кондиционер кладовой 1"
},
"status": false,
"date_update": "2021-05-31T18:26:27.075847+05:00"
}
},
  {
"type": "controlapparat",
"data": {
"apparat": {
"id": 3,
"name": "Холодильник1"
},
"status": true,
"date_update": "2021-05-31T18:05:20.891683+05:00"
}
}
```


В случае если ключ и идентификатор не подходят друг другу, то ответом является сообщение «KEY ERROR!»

2) Отправка данных датчика

Запрос должен быть отправлен на адрес «polls/api/postsensordat/». Иметь тот же формат и такие же поля. Но помимо них, запрос должен содержать также поля о текущей температуре и установленных на устройстве границах температуры. Пример запроса на сохранение данных с датчика выглядит представлен в листинге 29.

Листинг 29 – Пример запроса на отправку данных о датчике 2 с управляющего контроллера

```
{
  "device_id": "1",
  "device_key":"+H#2&+OgGKxhGC7<WYKA6&!BqeP8Vt67eI>jh!b&RKptkBfhn?#Mc4pqc+ctlSpX",
  "sensor_id" : 2,
  "temperature" : 28,
  "min_temperature" : 20,
  "max_temperature" : 30
}
```

3) Отправка данных аппарата

Запрос должен быть отправлен на адрес «polls/api/postapparatdat/». Иметь тот же формат. Содержать поля идентификатора устройства и его ключа. Также запрос должен содержать поле идентификатора аппарата и его статус. Пример запроса представлен в листинге 30.

Листинг 30 – Пример запроса на обновление статуса аппарата 2

```
{
  "device_id": "1",
  "device_key":"+H#2&+OgGKxhGC7<WYKA6&!BqeP8Vt67eI>jh!b&RKptkBfhn?#Mc4pqc+ctlSpX",
  "apparat_id" : 2,
  "status" : "True"
}
```

Для запросов на обновление данных датчиков и аппаратов ответное сообщение формируется так: в случае, если ключ и идентификатор не подходят друг другу, то ответом является сообщение «KEY ERROR!». Если всё заполнено правильно, то в ответ будет получено сообщение «success». Если какое из полей заполнено не корректно, то сообщение будет содержать надпись «BAD DATA»

6.6 Тестирование системы оповещения.

В системе предусмотрено оповещение по электронной почте, в случае если датчик одного из управляющих контроллеров, принадлежащего пользователю, выйдет за установленные температурные границы. Для тестирования был послан api запрос с данными выходящими за границы с одной и другой стороны, после чего на почтовый ящик пришло два сообщения говорящие о данных ситуациях.

6.7 Тестирование доступа к панели администратора

Ограничение доступа к панели администратора осуществляется с помощью поля «is_staff» модели User. В случае, если значение поля является False, то при попытке входа отображается сообщение об ошибке (рисунок 33).

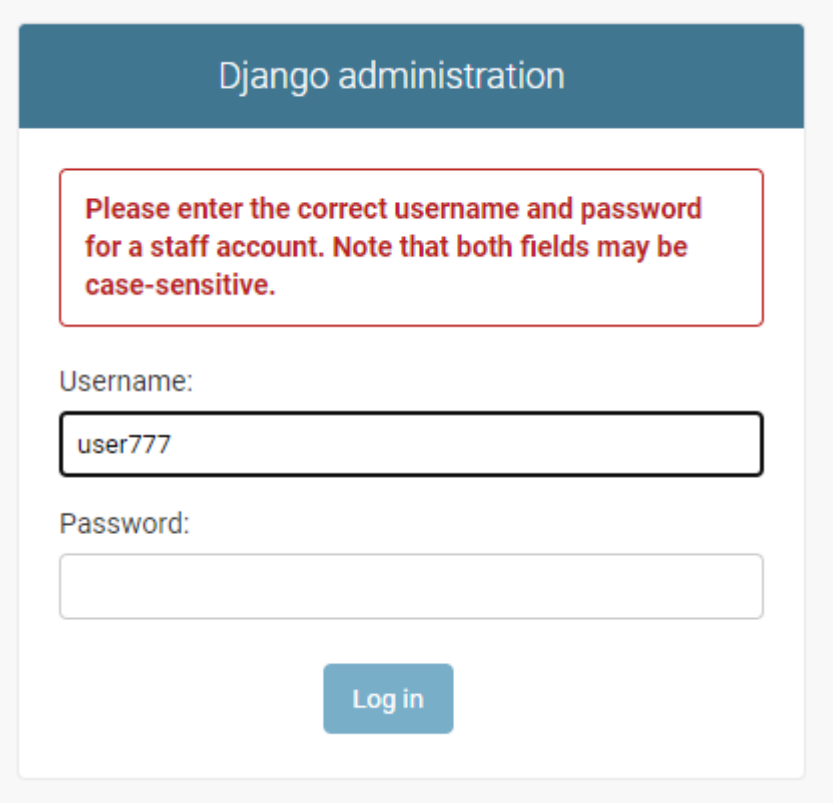

Рисунок 33 – Ошибка входа в панель администратора не сотрудником

7 ПРОТОТИП УСТРОЙСТВА ВЗАИМОДЕЙСТВУЮЩЕГО С СИСТЕМОЙ

В рамках выпускной квалификационной работы был также написан программный код для микроконтроллера Atmega328p. Он является 8-ми разрядным CMOS контроллером с относительно низким энергопотреблением, основанным на усовершенствованной AVR RISC архитектуре.

К Atmega328p подключен модуль ENC28J60. Он предоставляет возможность подключения контроллера к сети интернет через LAN. Особенностью данной платы - является отсутствие аппаратной поддержки протоколов передачи информации. Это значит, что всё преобразование данных происходит программно. Для удобства была подключена свободно распространяемая библиотека «EtherCard.h».

Также к контроллеру подключен температурный датчик DS18B20 – это датчик температуры, который обладает разрешением преобразования от 9 до 12 разрядов. Большинство параметров контроля задаются самостоятельно, пользователем. Они сохраняются в памяти и могут быть перенастроены в будущем. Датчик DS18B20 использует протокол интерфейса 1-Wire для обмена данными. Для удобной и корректной работы с датчиком были подключены следующие библиотеки: «OneWire.h», «DallasTemperature.h»

Исходный код для контроллера представлен в листинге 31.

Листинг 31 – Исходный код для управляющего контроллера

```
#include <EtherCard.h>
#include <OneWire.h>
#include <DallasTemperature.h>

static byte mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 };
static byte hisip[] = { 192,168,0,3 };

#define ONE_WIRE_BUS 2
#define VARIABLE "1"  //sensor_id
#define device "1"
#define key "+H#2&+OgGKxhGC7<WYKA6&!BqeP8Vt67eI>jh!b&RKptkBfhn?#Mc4pqc+ctlSpX"
#define mintemp "-20"
#define maxtemp "70"
#define PATH "polls/api/postsensordat/"

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensor(&oneWire);
byte Ethernet::buffer[700];
static uint32_t timer;
```

```
Stash stash;

const char website[] PROGMEM = "192.168.0.3";
float t;
static char outstr[5];


// called when the client request is complete
static void my_callback (byte status, word off, word len) {
  Serial.println(">>>");
  Ethernet::buffer[off+300] = 0;
  Serial.print((const char*) Ethernet::buffer + off);
}

void setup () {
  Serial.begin(57600);
  Serial.println(F("\n[webClient]"));

  if (ether.begin(sizeof Ethernet::buffer, mymac) == 0)
    Serial.println(F("Failed to access Ethernet controller"));
  if (!ether.dhcpSetup())
    Serial.println(F("DHCP failed"));

  ether.printIp("IP:  ", ether.myip);
  ether.printIp("GW:  ", ether.gwip);
  ether.printIp("DNS: ", ether.dnsip);
  ether.copyIp(ether.hisip, hisip);
//  if (!ether.dnsLookup(website))
//    Serial.println("DNS failed");

  ether.printIp("SRV: ", ether.hisip);
  sensor.begin();
  sensor.setResolution(12);
}

void loop () {
  float temperature;
  ether.packetLoop(ether.packetReceive());

  if (millis() > timer) {
    sensor.requestTemperatures();
    temperature = sensor.getTempCByIndex(0);
    timer = millis() + 10000;
    dtostrf(temperature,2, 2, outstr);
    Serial.println("<<< REQ");
    Serial.print("t=");
    Serial.print(outstr);

    byte sd = stash.create();
    stash.print('{');
    stash.print('"');
    stash.print("device_id");
    stash.print('"');
    stash.print(':');
    stash.print('"');
    stash.print(device);
    stash.print('"');
    stash.print(',');

    stash.print('"');
    stash.print("device_key");
    stash.print('"');
    stash.print(':');
    stash.print('"');
```

```
    stash.print(key);
    stash.print('"');
    stash.print(',');

    stash.print('"');
    stash.print("sensor_id");
    stash.print('"');
    stash.print(':');
    stash.print(VARIABLE);
    stash.print(',');

    stash.print('"');
    stash.print("temperature");
    stash.print('"');
    stash.print(':');
    stash.print(outstr);
    stash.print(',');

    stash.print('"');
    stash.print("min_temperature");
    stash.print('"');
    stash.print(':');
    stash.print(mintemp);
    stash.print(',');

    stash.print('"');
    stash.print("max_temperature");
    stash.print('"');
    stash.print(':');
    stash.print(maxtemp);
    stash.print('}');

    stash.save();

    Stash::prepare(PSTR("POST /$F HTTP/1.1" "\r\n"
                "Host: $F" "\r\n"
                "content-Type: application/json" "\r\n"
                "content-Length: $D" "\r\n"
                "\r\n"
                "$H"),
    PSTR(PATH), website, stash.size(), sd);
    ether.tcpSend();
  }
}
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было выполнено следующее:

1) Проведен обзор аналогов системы. Были выявлены сильные и слабые стороны подобных систем.

2) Определены технические, системные, функциональные и нефункциональные требования к приложению.

3) Проведен анализ основных технологий разработки подобных систем. Был выявлен более удобный и интересный язык программирования для данной задачи. Также во многом язык был выбран благодаря функционалу фреймворка Django . Выявлена более подходящая среда разработки и база данных для данного проекта.

4) Спроектирована архитектура приложения.

5) Программно реализовано приложение, удовлетворяющее всем требованиям к системе.

6) Произведено тестирование всех форм ввода.

В Веб-приложении были реализованы следующие функции:

1) Аутентификация пользователя.

2) Контроль за доступом к данным.

3) Мониторинг пришедших данных с устройств за весь период и только последние данные.

4) Возможность изменения конфигурационных данных запрашиваемых устройствами из удобного пользовательского интерфейса.

5) Возможность изменения данных о необходимом статусе устройств подключенных к управляющему контроллеру.

6) Система оповещения. Выход за границы значений датчиков.

Возможные пути развития веб-приложения:

1) Связывание датчиков и аппаратов, для автоматического включения и отключения при достижении определенных значений показаний.

2) Создание мобильного приложения на основе веб-приложения. Для доступа к данным можно использовать имеющиеся api обработчики с некоторой модернизацией.

3) Расширение функционала за счет использования не только термо-датчиков, но и любых других имеющих схожие значения.

4) Для аппаратов использование не двух статусов (включен и выключен), а более.

5) Разделение пользователей на владельцев и наблюдателей. Для того чтобы страница управления устройством была доступна не всем имеющим связь пользователям.

6) Улучшение системы оповещений. Добавление мессенджеров или sms-оповещения за счет сторонних сервисов.

7) Доработка интерфейса на основе отзывов пользователей.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. О компании [Электронный ресурс]. – Режим доступа: https://gigrotermon.ru/ru/o-kompanii (Дата обращения: 22.03.2021)
2. Купить термоподвески по выгодным ценам. Оборудование для модернизации элеватора [Электронный ресурс]. – Режим доступа: https://nppkipr.ru/o-kompanii/(Дата обращения: 23.03.2021)
3. «Полтраф СНГ» — эксклюзивные поставки контрольно-измерительного оборудования [Электронный ресурс]. – Режим доступа: https://poltraf.ru/about/ (Дата обращения: 23.03.2021)
4. Контактная информация - Unimon.ru [Электронный ресурс]. – Режим доступа: https://unimon.ru/contacts/ (Дата обращения: 23.03.2021)
5. Контактная информация - Unimon.ru [Электронный ресурс]. – Режим доступа: https://www.krug2000.ru/about.html (Дата обращения: 23.03.2021)
6. PHP: Что такое PHP? – Manual [Электронный ресурс]. – Режим доступа: https://www.php.net/manual/ru/intro-whatis.php (Дата обращения: 23.03.2021)
7. PHP: Возможности PHP – Manual [Электронный ресурс]. – Режим доступа: https://www.php.net/manual/ru/intro-whatcando.php (Дата обращения: 23.03.2021)
8. Installation - Laravel - The PHP Framework For Web Artisans [Электронный ресурс]. – Режим доступа: https://laravel.com/docs/8.x (Дата обращения: 23.03.2021)
9. CodeIgniter Documentation [Электронный ресурс]. – Режим доступа: https://codeigniter.com/docs (Дата обращения: 23.03.2021)
10. 3.9.2 Documentation [Электронный ресурс]. – Режим доступа: https://docs.python.org/3/ (Дата обращения: 23.03.2021)
11. Django documentation | Django documentation | Django [Электронный ресурс]. – Режим доступа: https://docs.djangoproject.com/en/3.1/ (Дата обращения: 23.03.2021)
12. Documentation [Электронный ресурс]. – Режим доступа: https://trypyramid.com/documentation.html (Дата обращения: 23.03.2021)

13. The TurboGears Documentation — TurboGears 2.3.12 documentation [Электронный ресурс]. – Режим доступа: https://turbogears.readthedocs.io/en/tg2.3.12/ (Дата обращения: 23.03.2021)
14. Документация по C#. Начало работы, руководства, справочные материалы. | Microsoft Docs [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (Дата обращения: 23.03.2021)
15. Яргер, Р.Дж. MySQL и mSQL: Базы данных для небольших предприятий и Интернета / Р.Дж. Яргер, Дж. Риз, Т. Кинг. - М.: СПб: Символ-Плюс, 2015. - 560 с.
16. PostgreSQL: The world's most advanced open source database [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/ (Дата обращения: 24.03.2021)
17. PostgreSQL: Feature Matrix [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/about/featurematrix/ (Дата обращения: 24.03.2021)
18. Фонд MariaDB - MariaDB.org [Электронный ресурс]. – Режим доступа: https://mariadb.org/ (Дата обращения: 24.03.2021)
19. Лутц М. Изучаем Python, том 2, 5-е изд.; пер. с англ. Ю. Н. Артеменко – СПб.: ООО “Диалектика”, 2020. – 720 с.
20. Златопольский, Д.М. Основы программирования на языке Python. – Москва : ДМК Пресс, 2017. – 284 с.
21. Гэддис Т. Начинаем программировать на Python, 4-е изд.; пер. с англ. СПб.: БХВ-Петербург - СПб.: БХВ-Петербург, 2019. – 768 с.
22. Django documentation [Электронный ресурс]. Режим доступа: https://docs.djangoproject.com/en/3.2/ (Дата обращения: 25.05.2021)
23. Django REST Framework [Электронный ресурс]. Режим доступа: http://www.django-rest-framework.org/ (Дата обращения: 25.05.2021)