МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №42

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

ассистент		Д. О. Шевяков
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ПРОЕКТУ
Система автоматического регулирования микроклимата в жилом помещении «САРУС»

по дисциплине: Основные положения Интернета вещей

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. 1112 №

		Еременко Н. О.
	подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

1 Оглавление

3 Задание

Согласно выбранной или согласованной с преподавателем теме, представить проект и разработать систему интернета вещей. Система должна быть основана на веб-приложении с использованием фреймворка Flask или аналогичного ему. В системе должны применяться функции для обмена данными, реализованные в виде URL-запросов между различными частями системы.

Система должна быть разработана в парадигме объектно-ориентированного программирования и включать в себя классы, их поведение, атрибуты и иерархию. Классы должны быть реализованы в отдельном модуле. Если система включает в себя элементы, которые можно сгруппировать по функционалу, каждая из таких групп должна быть реализована в отдельном модуле.

Система должна быть разработана с использованием гибких подходов. Отдельные элементы системы должны иметь возможность работать независимо друг от друга. Система должна иметь возможность масштабирования и добавления новых сущностей без существенной модификации кода программы. В системе не должно присутствовать одинаковых фрагментов кода. Если какой-либо код используется в программе более одного раза, он должен быть оформлен в виде функции или метода. Система должна быть реализована таким образом, чтобы работоспособность каждой из функций можно было бы легко протестировать без модификации кода программы.

Система должна включать в себя, как минимум один графический интерфейс пользователя, с помощью которого можно получить доступ ко всем функциям системы. При оценке интерфейса внимание будет обращаться на функциональность, а не на привлекательность с точки зрения дизайна.

Система должна включать в себя модуль сбора и сохранения данных системы. Модуль сбора данных должен работать независимо от других модулей и частей системы, а также собирать и сохранять данные с заданным периодом. Данный модуль должен иметь функцию предварительной проверки и обработки данных перед сохранением. Должны быть продуманы, как механизмы отказа сохранения данных, так и механизмы преобразования данных к требуемому виду или формату.

Система должна включать в себя модуль анализа сохраненных данных. Модуль анализа может включать в себя, как статистические характеристики системы, так и предиктивные модели, основанные на анализе имеющихся данных.

2 Проект системы

На рисунке 1 представлена UML-диаграмма классов серверной части системы «САУРС».

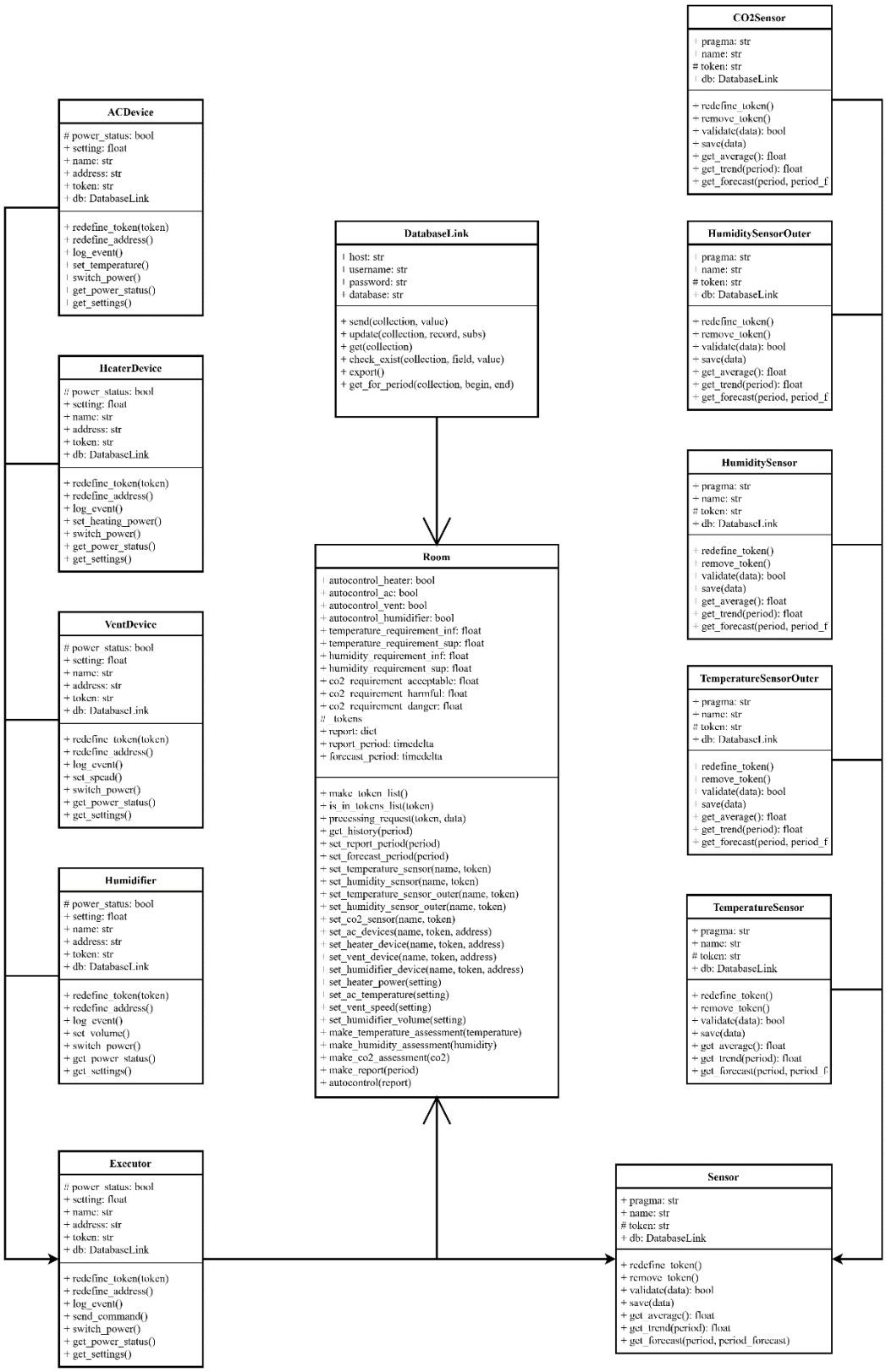


Рисунок 1 – UML-диаграмма классов серверной части системы

Основой системы является класс Room, представляющий собой модель жилой комнаты, качеств её воздуха и всех возможных способов влияния на них. Класс Room включает в себя классы DatabaseLink и интерфейсы Executor и Sensor.

DatabaseLink представляет собой класс доступа к базе данных MongoDB. Классы Executor и Sensor представляют из себя интерфейсы исполнительного устройства и датчика соответственно.

В системе присутствует четыре исполнительных устройства: HeaterDevice, ACDevice, VentDevice, Humidifier – электрообогреватель, кондиционер, активная вентиляция и увлажнитель воздуха; а так же пять датчиков: TemperatureSensor, TemperatureSensorOuter, HumiditySensor, HumiditySensorOuter, CO2Sensor – датчики температуры внутренний и уличный, датчики влажности внутренний и уличный и датчик углекислого газа.

Архитектурно класс Room предполагает наличие всего одного экземпляра каждого объекта ИУ и датчика. С одной стороны, это снижает гибкость системы и её надежность, однако это упущение приемлемо по двум причинам: во-первых, это упрощает процесс разработки; во-вторых, в техническом задании соответствующих требований нет. Система разрабатывается не только в учебных целях, и условия её работы таковы, что каждый датчик и ИУ в наличии в единственном экземпляре, и работает система всего в одной жилой комнате студии.

3 Описание объектной модели

3.1 Класс DatabaseLink

Класс базы данных MongoDB. Предназначен для предоставления упрощенного доступа к базе данных. Авторизуется, устанавливает соединение, совершает запросы к БД. Создается один экземпляр этого класса, после чего передается в класс Room как атрибут, а после всем ИУ и датчикам. Список методов указан в листинге 1.

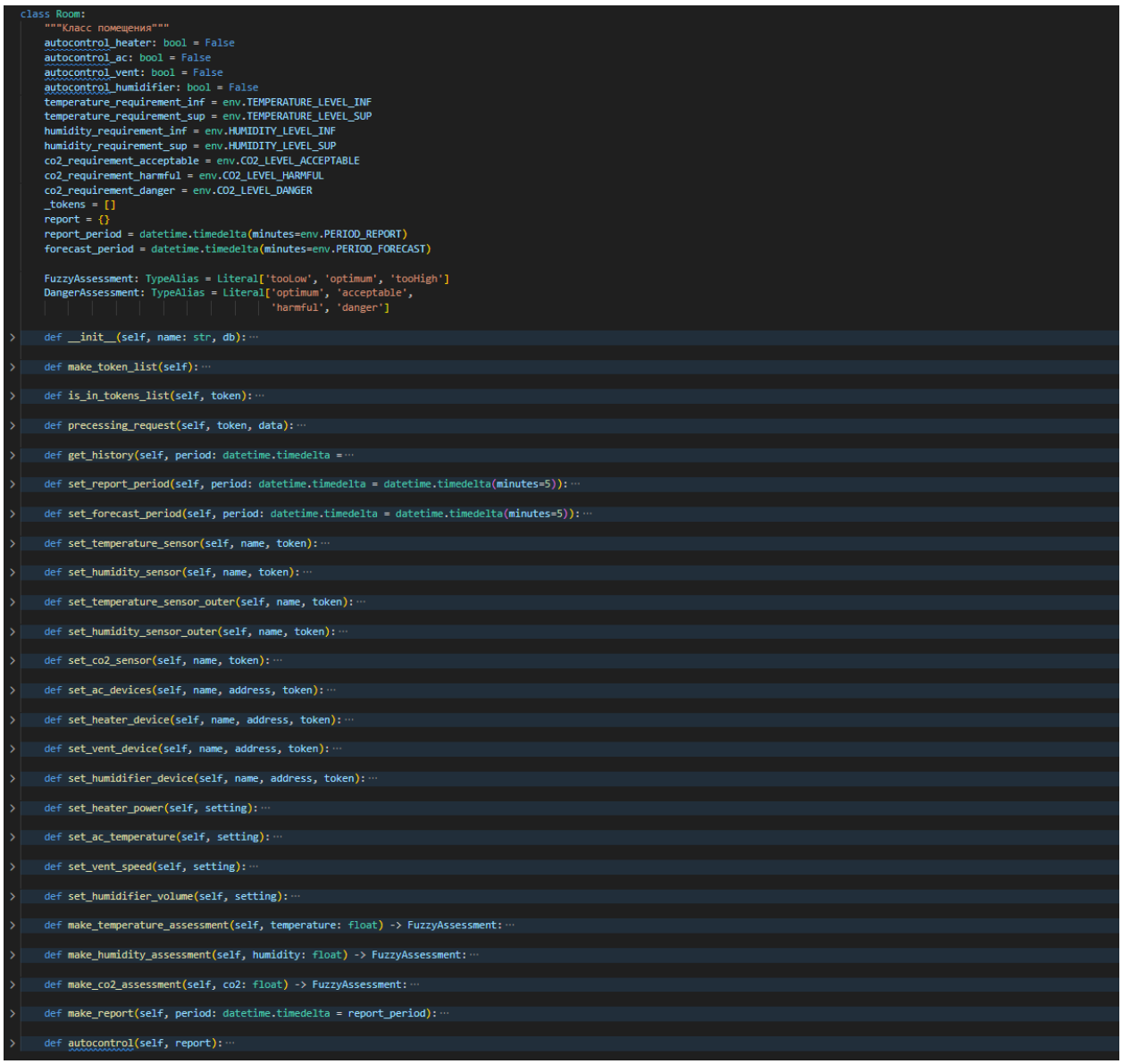
```
class DatabaseLink:
    """Класс интерфейса базы данных"""

    def __init__(self, host: str, username: str, password: str, database: str) -> None: ···

    def send(self, collection: str, value: dict): ···

    def update(self, collection: str, record: dict, subs: dict): ···

    def get(self, collection: str): ···

    def check_exist(self, collection, field, value): ···

    def export(self): ···

    def get_for_period(self, collection: str, begin: datetime.datetime, end: datetime.datetime = datetime.datetime.now()): ···
```

Листинг 1 – класс DatabaseLink и его методы

3.2 Класс Room

Главный класс в иерархии созданной архитектуры. Содержит в себе информацию обо всех ИУ и датчиках помещения, качествах воздуха. В нем происходит обработка данных, поступающих с датчиков, и автоматическое управление ИУ для поддержания качеств воздуха в установленном диапазоне.

Список методов указан в листинге 2.



Листинг 2 – класс Room и его методы

3.3 Интерфейс Sensor

Интерфейс Sensor – это абстрактный базовый класс датчика. Некоторые методы уже определены, что является неблагоприятной, но допустимой практикой. Класс датчика ответственен за авторизацию, валидацию, сохранение и доступ к информации, а также получения среднего значения за и прогноза на указанный периоды времени. Атрибут pragma указывает на тип датчика, чтобы после авторизации вызвать нужный объект для корректной обработки данных.

```python
class Sensor(abc.ABC):
    """АБК датчика"""

    pragma: str = None

    @abc.abstractmethod
    def __init__(self, name, token, db: DatabaseLink): ...

    @abc.abstractmethod
    def redefinition_token(self, token): ...

    @abc.abstractmethod
    def remove_token(self): ...

    @abc.abstractmethod
    def validate(self, data) -> bool: ...

    @abc.abstractmethod
    def save(self, data): ...

    @abc.abstractmethod
    def get_average(self) -> float: ...

    @abc.abstractmethod
    def get_trend(self, period) -> float: ...

    @abc.abstractmethod
    def get_forecast(self, period, period_forecast): ...
```

Листинг 3 – АБК Sensor и его методы

3.4 Класс TemperatureSensor

Класс, основанный на АБК Sensor. Обрабатывает температуру воздуха в помещении.

```python
class TemperatureSensor(Sensor):
    """Класс датчика температуры"""

    pragma = "temperature"

    def __init__(self, name, token, db: DatabaseLink): ...

    def validate(self, data) -> bool: ...

    def save(self, data): ...

    def get_average(self, period: datetime.timedelta): ...

    def get_trend(self, period: datetime.timedelta): ...

    def get_forecast(self, period: datetime.datetime, period_forecast: datetime.timedelta): ...

    def redefinition_token(self, token): ...

    def remove_token(self): ...
```

Листинг 4 – Класс

3.5 Класс HumiditySensor

Класс, основанный на АБК Sensor. Обрабатывает влажность воздуха в помещении.

```python
class HumiditySensor(Sensor):
    """Класс датчика влажности"""
    pragma = "humidity"

    def __init__(self, name, token, db: DatabaseLink): ...

    def validate(self, data) -> bool: ...

    def save(self, data): ...

    def get_average(self, period: datetime.datetime): ...

    def get_trend(self, period: datetime.datetime): ...

    def get_forecast(self, period: datetime.datetime, period_forecast: datetime.timedelta): ...

    def redefinition_token(self, token): ...

    def remove_token(self): ...
```

Листинг 5 – Класс HumiditySensor

3.6 Класс TemperatureSensorOuter

Класс, основанный на АБК Sensor. Обрабатывает температуру воздуха на улице.

```python
class TemperatureSensorOuter(Sensor):
    """Класс датчика температуры"""

    pragma = "temperature_outer"

    def __init__(self, name, token, db: DatabaseLink):...

    def validate(self, data) -> bool:...

    def save(self, data):...

    def get_average(self, period):...

    def get_trend(self, period):...

    def get_forecast(self, period: datetime.datetime, period_forecast: datetime.timedelta):...

    def redefinition_token(self, token):...

    def remove_token(self):...
```

Листинг 6 – Класс TemperatureSensorOuter

3.7 Класс HumiditySensorOuter

Класс, основанный на АБК Sensor. Обрабатывает влажность воздуха на улице.

```python
class HumiditySensorOuter(Sensor):
    """Класс датчика влажности"""
    pragma = "humidity_outer"

    def __init__(self, name, token, db: DatabaseLink):...

    def validate(self, data) -> bool:...

    def save(self, data):...

    def get_average(self, period: datetime.datetime):...

    def get_trend(self, period: datetime.datetime):...

    def get_forecast(self, period: datetime.datetime, period_forecast: datetime.timedelta):...

    def redefinition_token(self, token):...

    def remove_token(self):...
```

Листинг 7 – Класс HumiditySensorOuter

3.8 Класс CO2Sensor

Класс, основанный на АБК Sensor. Обрабатывает концентрацию углекислого газа воздуха в помещении.

```
class CO2Sensor(Sensor):
    """Класс датчика давления"""
    pragma = "co2"

    def __init__(self, name, token, db: DatabaseLink): ···

    def validate(self, data) -> bool: ···

    def save(self, data): ···

    def get_average(self, period): ···

    def get_trend(self, period): ···

    def get_forecast(self, period: datetime.datetime, period_forecast: datetime.timedelta): ···

    def redefinition_token(self, token): ···

    def remove_token(self): ···
```

Листинг 8 – Класс CO2Sensor

3.9 Интерфейс Executor

Класс интерфейса исполнительного устройства. Отвечает за отправку команд и установок удаленному ИУ и ведение журнала команд.

В данном случае мы прибегнули к не лучшей практике разработки: с одной стороны, Executor не должен иметь непосредственных реализаций и служит только как родитель для классов других исполнительных устройств. С другой же эти конкретные классы ИУ отличаются своим поведением незначительно, определяя собственные уникальные методы, но не используя некоторые методы родителя. Было принято решение написать его как класс, однако по целевому назначению использовать как интерфейс.

```python
class Executor:
    """Класс исполнительного устройства"""
    _power_status = False
    settings = None

    def __init__(self, name, address: str, _token, _db: DatabaseLink) -> None:...

    def redefinition_token(self, token):...

    def redefinition_address(self, address):...

    def log_event(self, value: dict):...

    def send_command(self, value: dict):...

    def switch_power(self, power: bool):...

    def get_power_status(self):...

    def get_settings(self):...
```

Листинг 9 – Интерфейс Executor и его методы

3.10 Класс HeaterDevice

```python
class HeaterDevice(Executor):
    """Класс обогревателя"""

    def __init__(self, name, address: str, _token, _db: DatabaseLink) -> None:...

    def set_heating_power(self, heating_power: int):...
```

Листинг 10 – Класс HeaterDevice

3.11 Класс ACDevice

```python
class ACDevice(Executor):
    """Класс кондиционера"""

    def __init__(self, name, address: str, _token, _db: DatabaseLink) -> None:...

    def set_temperature(self, temperature: int):...
```

Листинг 11 – Класс ACDevice

3.12 Класс VentDevice

```python
class VentDevice(Executor):
    """Класс вентилирующего агрегата"""

    def __init__(self, name, address: str, _token, _db: DatabaseLink) -> None:...

    def set_speed(self, speed: int):...
```

Листинг 12 – Класс VentDevice

3.13 Класс Humidifier

```python
class Humidifier(Executor):
    """Класс увлажнителя воздуха"""

    def __init__(self, name, address: str, _token, _db: DatabaseLink) -> None:...

    def set_volume(self, volume: int):...
```

Листинг 13 – Класс Humidifier

4 Описание модуля сбора данных

Данные поступают в систему из внешних датчиков через API, реализованное на REST-архитектуре. Декоратором @app.route устанавливается путь /api/device/send для отправки датчиками измерений. Метод запросов – POST. Из тела запроса, десериализованном как json, извлекаются токен безопасности и само измерение.

```
@app.route("/api/device/send", methods=["POST"])
def api_device_send():
    token = request.json["Auth"]
    value = request.json["value"]
    code = room.precessing_request(token, value)
    report = room.make_report()
    room.autocontrol(report)
    return Response(status=code)
```

Листинг – Функция api_device_send()

Далее токен безопасности и измерение передаются методу processing_request() класса Room, который проверяет факт передачи токена (иначе возвращает код состояния 403 «Запрещено»), проверяет токен (иначе возвращает код состояния 401 «Не авторизован»), проверяет наличие передаваемых данных (иначе возвращает код состояния 400 «Некорректный запрос») и, по токену, находит соответствующий ему датчик и передает данные этому датчику на обработку, и в случае успеха возвращает код состояния «200» ОК.

Метод save() интерфейса Sensor валидирует данные (листинги с по) и обращается к базе данных через объект класса DatabaseLink (листинг и). Суть валидации заключается в проверке величины на существование (если величина None, то возвращается False) и вхождение в диапазон правдоподобных значений.

```python
    def processing_request(self, token, data):
        if token == ("" or None):
            return 403
        if not self.is_in_tokens_list(token):
            return 401
        if data == ("" or None):
            return 400
        sensor: Sensor = self._tokens[token]
        sensor.save(data)
        return 201
```

Листинг – Метод processing_request()

```python
class TemperatureSensor(Sensor):
    """Класс датчика температуры"""

    pragma = "temperature"

    def __init__(self, name, token, db: DatabaseLink): ...

    def validate(self, data) -> bool:
        if data == None:
            return False
        return data < 100 and data > -60
```

Листинг – метод validate() класса TemperatureSensor

```python
class HumiditySensor(Sensor):
    """Класс датчика влажности"""
    pragma = "humidity"

    def __init__(self, name, token, db: DatabaseLink): ...

    def validate(self, data) -> bool:
        if data == None:
            return False
        return data < 100 and data > 0
```

Листинг – метод validate() класса HumiditySensor

```python
class CO2Sensor(Sensor):
    """Класс датчика давления"""
    pragma = "co2"

    def __init__(self, name, token, db: DatabaseLink):...

    def validate(self, data) -> bool:
        if data == None:
            return False
        return data < 100e3 and data > 0
```

Листинг – метод validate() класса CO2Sensor

```python
def save(self, data):
    if self.validate(data):
        self.db.send("co2", {"co2": data, "sensor": self.name})
```

Листинг – метод save() (на примере класса CO2Sensor)

```python
def send(self, collection: str, value: dict):
    """Создать запись в БД"""
    value |= {"date": datetime.datetime.now()}  # Время записи добавляется автоматически
    self.db[collection].insert_one(value)
```

Листинг – метод send() класса DatabaseLink

Целиком можно извлечь коллекцию из базы данных через метод get() класса DatabaseLink.

```python
def get(self, collection: str):
    """Получить записи в БД"""
    return list(self.db[collection].find().sort({"date": 1}))
```

Листинг – метод get() класса DatabaseLink

Метод get() класса DatabaseLink использует метод make_token_list() класса Room (листинг), который необходим для авторизации по токену. Однако более востребованным является метод get_for_perion() класса DatabaseLink, который выбирает из коллекции только те записи, что удовлетворяют указанному временному диапазону.

```python
def get_for_period(self, collection: str, begin: datetime.datetime, end: datetime.datetime = datetime.datetime.now()):
    """Получить записи в БД за данный промежуток времени"""
    return list(self.db[collection].find({"date": {"$gt": begin, "$lt": end}}).sort({"date": 1}))
```

Листинг – метод for_perion() класса DatabaseLink

5 Описание функций анализа сохраненных данных

5.1 Функция quick_lstsq()

Данная функция принимает на вход массив величин и массив меток времени и находит линейную функцию методом наименьших квадратов. Функция возвращает две величины: тангенс наклона прямой и её смещение. Код функции в листинге . Внутри используется функция lstsq модуля numpy.

```
def quick_lstsq(values: list, dates: list[datetime.datetime]):
    try:
        len(values) == len(dates)
    except:
        return IndexError
    A = np.vstack([[x.timestamp() for x in dates], np.ones(len(dates))]).T
    tg, shift = np.linalg.lstsq(A, values, rcond=None)[0]
    return (tg, shift)
```

Листинг – функция quick_lstsq()

5.2 Метод get_average()

Метод get_average() интерфейса Sensor. Возвращает среднее арифметическое измерений за указанный промежуток времени. Реализации отличаются только коллекцией БД, из которой объект извлекает данные соответствующей ему физической величины. В листинге продемонстрирован метод на примере класса CO2Sensor.

```
    def get_average(self, period):
        records = self.db.get_for_period(
            "co2", datetime.datetime.now()-period, datetime.datetime.now())
        try:
            len(records) > 0
        except:
            IndexError
        values = [float(x["co2"]) for x in records]
        try:
            average = sum(values)/len(values)
        except:
            ZeroDivisionError
            average = None
        return average
```

Листинг – метод get_average()

5.3 Метод get_trend()

Метод get_trend() интерфейса Sensor. Возвращает производную по времени среднего изменения физической величины за указанный промежуток времени. Реализации отличаются только коллекцией БД, из которой объект извлекает данные соответствующей ему физической величины. В листинге продемонстрирован метод на примере класса CO2Sensor.

```
    def get_trend(self, period):
        records = self.db.get_for_period(
            "co2", datetime.datetime.now()-period, datetime.datetime.now())
        try:
            len(records) > 0
        except:
            IndexError
        values = [float(x["co2"]) for x in records]
        dates = [x["date"] for x in records]
        return quick_lstsq(values, dates)[0]
```

Листинг – метод get_trend()

5.4 Метод get_forecast()

Метод get_forecast() интерфейса Sensor. Возвращает экстраполяцию физической величины через указанный промежуток времени по её производной по времени среднего изменения этой величины. Реализации во всех классах идентичны.

```
    def get_forecast(self, period: datetime.datetime, period_forecast: datetime.timedelta):
        average = self.get_average(period)
        trend = self.get_trend(period)
        forecast = trend * period_forecast.seconds + average
        return forecast
```

Листинг – метод get_forecast()

5.5 Литералы DangerAssessment и FuzzyAssessment

Литералы серии Assessment – это оценки нечеткой логики, характеризующие физическую величину. Литерал FuzzyAssessment дает оценку, находится величина в оптимальном диапазоне или выходит за его пределы: «слишком мало», «комфортно», «слишком много». В контексте температуры крайние значения можно трактовать как «холодно» и «жарко», а в контексте влажности – «сухо» и «сыро» соответственно. Литерал

DangerAssessment дает оценку опасности физической величины: «безопасно», «допустимо», «вредно» и «опасно».

```
FuzzyAssessment: TypeAlias = Literal['tooLow', 'optimum', 'tooHigh']
DangerAssessment: TypeAlias = Literal['optimum', 'acceptable',
                                      'harmful', 'danger']
```

Листинг – литералы DangerAssessment и FuzzyAssessment

5.6 Метод make_temperature_assessment()

Метод make_temperature_assessment() принимает значение температуры и возвращает её оценку в терминах литерала FuzzyAssessment. Код метода на листинге.

```
    def make_temperature_assessment(self, temperature: float) -> FuzzyAssessment:
        assessment: self.FuzzyAssessment = "optimum"
        if temperature > self.temperature_requirement_sup:
            assessment = "tooHigh"
        if temperature < self.temperature_requirement_inf:
            assessment = "tooLow"
        return assessment
```

Листинг – Метод make_temperature_assessment()

5.7 Метод make_humidity_assessment()

Метод make_humidity_assessment() принимает значение влажности и возвращает её оценку в терминах литерала FuzzyAssessment. Код метода на листинге.

```
    def make_humidity_assessment(self, humidity: float) -> FuzzyAssessment:
        assessment: self.FuzzyAssessment = "optimum"
        if humidity > self.humidity_requirement_sup:
            assessment = "tooHigh"
        if humidity < self.humidity_requirement_inf:
            assessment = "tooLow"
        return assessment
```

Листинг – Метод make_ humidity _assessment()

5.8 Метод make_co2_assessment()

Метод make_co2_assessment() принимает значение концентрации углекислого газа и возвращает её оценку в терминах литерала DangerAssessment. Код метода на листинге.

```
def make_co2_assessment(self, co2: float) -> FuzzyAssessment:
    assessment: self.FuzzyAssessment = "danger"
    if co2 < self.co2_requirement_danger:
        assessment = "harmful"
    if co2 < self.co2_requirement_harmful:
        assessment = "acceptable"
    if co2 < self.co2_requirement_acceptable:
        assessment = "optimum"
    return assessment
```

Листинг – Метод make_ co2 _assessment()

5.9 Метод make_report()

Метод make_report() класса Room. Принимает значение периода, за который требуется сформировать отчет. Метод можно разделить на блоки по этапам обработки данных. Первый блок проверяет наличие объектов датчиков; второй обращается к методу get_average() датчиков и записывает в переменные среднее значение за период; третий проверяет наличие всех трех средних значений; четвертый обращается к методам серии make assessment, записывая в переменные оценки величин; пятый обращается к методу get_trend() и записывает в переменную производную по времени изменения величины; шестой обращается к методам серии make forecast и записывает в переменные данные об экстраполяции; седьмой обращается к методам серии make assessment, чтобы дать оценку экстраполяции; восьмой обращается к методу get_power_status() для получения состояния ИУ; и, наконец, формирует словарь отчета, который и возвращает. Экстраполяции качеств уличного воздуха не производится, поскольку считается, что он достаточно инертен; к тому же на его состояние повлиять невозможно. Код метода на листинге.

```python
    def make_report(self, period: datetime.timedelta = report_period):
        """Составить оценку атмосферных параметров"""

        if not self.temperature_sensor:
            print("Temperature sensor not set!")
            return None
        if not self.humidity_sensor:
            print("Humidity sensor not set!")
            return None
        if not self.co2_sensor:
            print("CO2 sensor not set!")
            return None

        temperature = self.temperature_sensor.get_average(period)
        humidity = self.humidity_sensor.get_average(period)
        co2 = self.co2_sensor.get_average(period)
        temperature_outer = self.temperature_sensor_outer.get_average(period)
        humidity_outer = self.humidity_sensor_outer.get_average(period)

        if not self.temperature_sensor.validate(temperature) or not self.humidity_sensor.validate(humidity) or not self.co2_sensor.validate(co2):
            return None

        temperature_assessment: self.FuzzyAssessment =\
            self.make_temperature_assessment(temperature)
        humidity_assessment: self.FuzzyAssessment =\
            self.make_humidity_assessment(humidity)
        co2_assessment: self.DangerAssessment =\
            self.make_co2_assessment(co2)
        temperature_outer_assessment: self.FuzzyAssessment =\
            self.make_temperature_assessment(temperature_outer)
        humidity_outer_assessment: self.FuzzyAssessment =\
            self.make_humidity_assessment(humidity_outer)

        temperature_trend = self.temperature_sensor.get_trend(period)
        humidity_trend = self.humidity_sensor.get_trend(period)
        co2_trend = self.co2_sensor.get_trend(period)

        temperature_forecast = self.temperature_sensor.get_forecast(
            period, self.forecast_period)
        humidity_forecast = self.humidity_sensor.get_forecast(
            period, self.forecast_period)
        co2_forecast = self.co2_sensor.get_forecast(
            period, self.forecast_period)

        temperature_forecast_assessment =\
            self.make_temperature_assessment(temperature_forecast)
        humidity_forecast_assessment =\
            self.make_humidity_assessment(humidity_forecast)
        co2_forecast_assessment =\
            self.make_co2_assessment(co2_forecast)

        vent_power = self.vent_device.get_power_status()
        ac_power = self.ac_device.get_power_status()
        heater_power = self.heater_device.get_power_status()
        humidifier_power = self.humidifier_device.get_power_status()
        report = {
            "value": {
                "temperature": np.round(temperature, 1),
                "humidity": np.round(humidity, 0),
                "co2": np.round(co2, 0),
                "temperature_outer": np.round(temperature_outer, 1),
                "humidity_outer": np.round(humidity_outer, 0),
            },
            "assessment": {
                "temperature": temperature_assessment,
                "humidity": humidity_assessment,
                "co2": co2_assessment,
                "temperature_outer": temperature_outer_assessment,
                "humidity_outer": humidity_outer_assessment,
            },
            "trend": {
                "temperature": temperature_trend,
                "humidity": humidity_trend,
                "co2": co2_trend,
            },
            "forecast": {
                "temperature": temperature_forecast,
                "humidity": humidity_forecast,
                "co2": co2_forecast,
            },
            "forecast_assessment": {
                "temperature": temperature_forecast_assessment,
                "humidity": humidity_forecast_assessment,
                "co2": co2_forecast_assessment,
            },
            "devices_status": {
                "vent": vent_power,
                "ac": ac_power,
                "heater": heater_power,
                "humidifier": humidifier_power,
            },
            "forecast_for_period": int(self.forecast_period.seconds),
            "for_moment": datetime.datetime.now().isoformat(),
            "for_period": int(period.seconds),
        }

        self.report = report

        return report
```

Листинг – метод make_report()

5.10 Метод autocontrol()

Метод класса Room. Отвечает за анализ отчета и реализацию САР.

```
def autocontrol(self, report):
    """Использовать ИУ для повышения атмосферных качеств"""
    temperature_forecast = report["forecast_assessment"]["temperature"]
    humidity_forecast = report["forecast_assessment"]["humidity"]
    temperature_outer = report["assessment"]["temperature_outer"]
    humidity_outer = report["assessment"]["humidity_outer"]
    co2_forecast = report["forecast_assessment"]["co2"]
    co2 = report["assessment"]["co2"]

    ac_power: bool = False
    heater_power: bool = False
    humidifier_power: bool = False
    vent_power: bool = False

    match temperature_forecast:
        case "tooLow":
            match humidity_forecast:
                case "tooLow":
                    if temperature_outer != "tooLow" and \
                            humidity_outer != "tooLow":
                        vent_power = True
                    else:
                        heater_power = True
                        humidifier_power = True
                case "optimum":
                    if temperature_outer != "tooLow" and \
                            humidity_outer == "optimum":
                        vent_power = True
                    else:
                        heater_power = True
                case "tooHigh":
                    if temperature_outer != "tooLow" and \
                            humidity_outer != "tooHigh":
                        vent_power = True
                    else:
                        heater_power = True
        case "optimum":
            match humidity_forecast:
                case "tooLow":
                    if temperature_outer == "optimum" and \
                            humidity_outer != "tooLow":
                        vent_power = True
                    else:
                        humidifier_power = True
                case "optimum":
                    pass
                case "tooHigh":
                    if temperature_outer == "optimum" and \
                            humidity_outer != "tooHigh":
                        vent_power = True
                    else:
```

Листинг – метод autocontrol() (часть первая из двух)

```
                        pass
            case "tooHigh":
                match humidity_forecast:
                    case "tooLow":
                        if temperature_outer != "tooHigh" and \
                                humidity_outer != "tooHigh":
                            vent_power = True
                        else:
                            ac_power = True
                    case "optimum":
                        if temperature_outer != "tooHigh" and \
                                humidity_outer == "optimum":
                            vent_power = True
                        else:
                            ac_power = True
                    case "tooHigh":
                        if temperature_outer != "tooHigh" and \
                                humidity_outer != "tooHigh":
                            vent_power = True
                        else:
                            ac_power = True

        if co2_forecast == "danger" or co2 == "danger":
            vent_power = True
            if not self._autocontrol_vent:
                self.vent_device.switch_power(vent_power)
        if (co2_forecast == "harmful" or co2 == "harmful") and \
                temperature_forecast == "optimum":
            vent_power = True

        if self.autocontrol_vent and vent_power:
            self.vent_device.switch_power(vent_power)
        if self.autocontrol_ac and ac_power:
            self.ac_device.switch_power(ac_power)
        if self.autocontrol_heater and heater_power:
            self.heater_device.switch_power(heater_power)
        if self.autocontrol_humidifier and humidifier_power:
            self.humidifier_device.switch_power(humidifier_power)
```

Листинг – метод autocontrol() (часть вторая из двух)

На вход метод принимает отчет. Также разделим этапы работы метода на блоки. В первом в переменные записываются данные экстраполяции из отчета; во втором создаются флаги включения ИУ; в третьем избирается одно из девяти возможных состояний качества воздуха в пространстве температура-влажность, и на основе оценки выбирается оптимальный путь для улучшения качества воздуха. Предпочтительным является вентилирование, как самый энергоэффективный и вместе с тем снижающий концентрацию углекислого газа. В четвертом блоке на основе оценки концентрации углекислоты принимается решение об аварийном вентилировании вне зависимости от настройки перехвата управления. В пятом блоке, с учетом установок перехвата управления ИУ, отдается команда на его включение или выключение.

Как было сказано, всего возможных состояний качества воздуха в пространстве температура-влажность девять: «холодно и сыро», «холодно», «холодно и сухо», «сыро», «комфортно», «сухо», «жарко и сыро», «жарко», «жарко и сухо». Код метода в листинге.

6 Интерфейсы

Графический интерфейс пользователя разделен на четыре страницы: «Главная», «Отчет», «Настройки» и «Устройства».

Страница «Главная», представленная на рисунке , встречает пользователя и дает общее представление о системе, её функциях и возможностях.

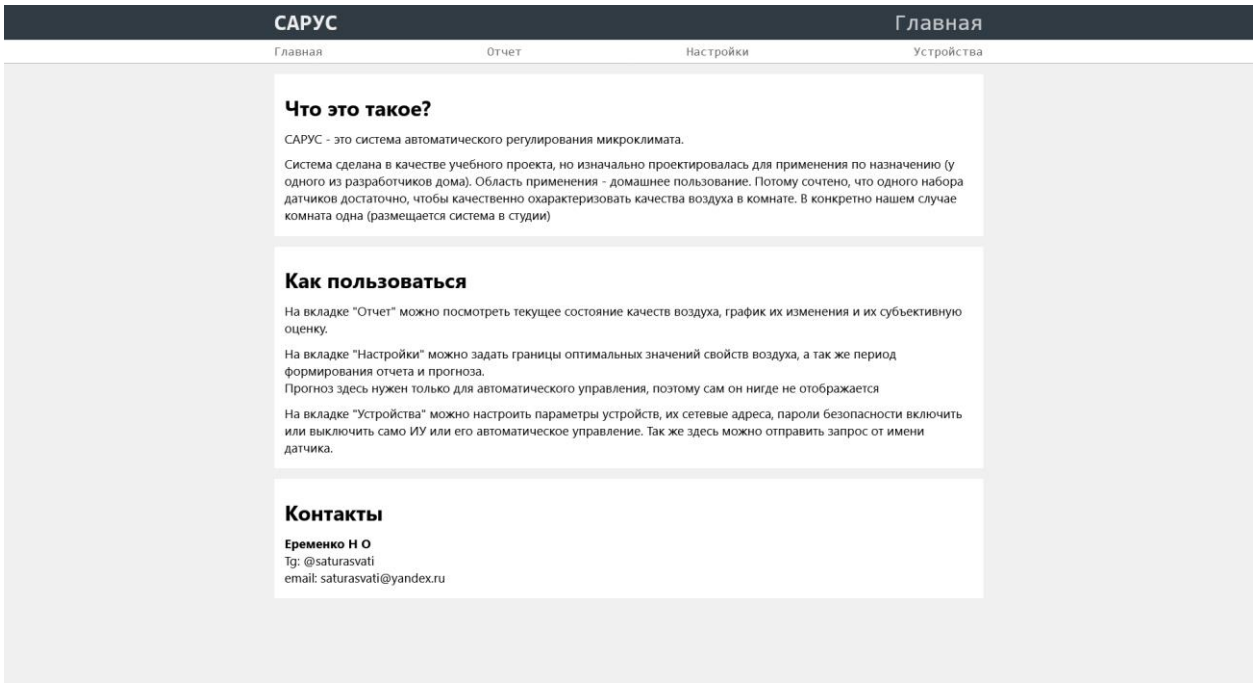


Рисунок – Страница приветствия

Страница «Отчет», представленная на рисунке , отображает данные с датчиков за последние n-минут и их субъективную характеристику (для наглядности выделено цветом). Так же можно построить график каждого из параметров за определенный период времени (за построение графиков отвечает open-source js-библиотека plotly).

Отдельно

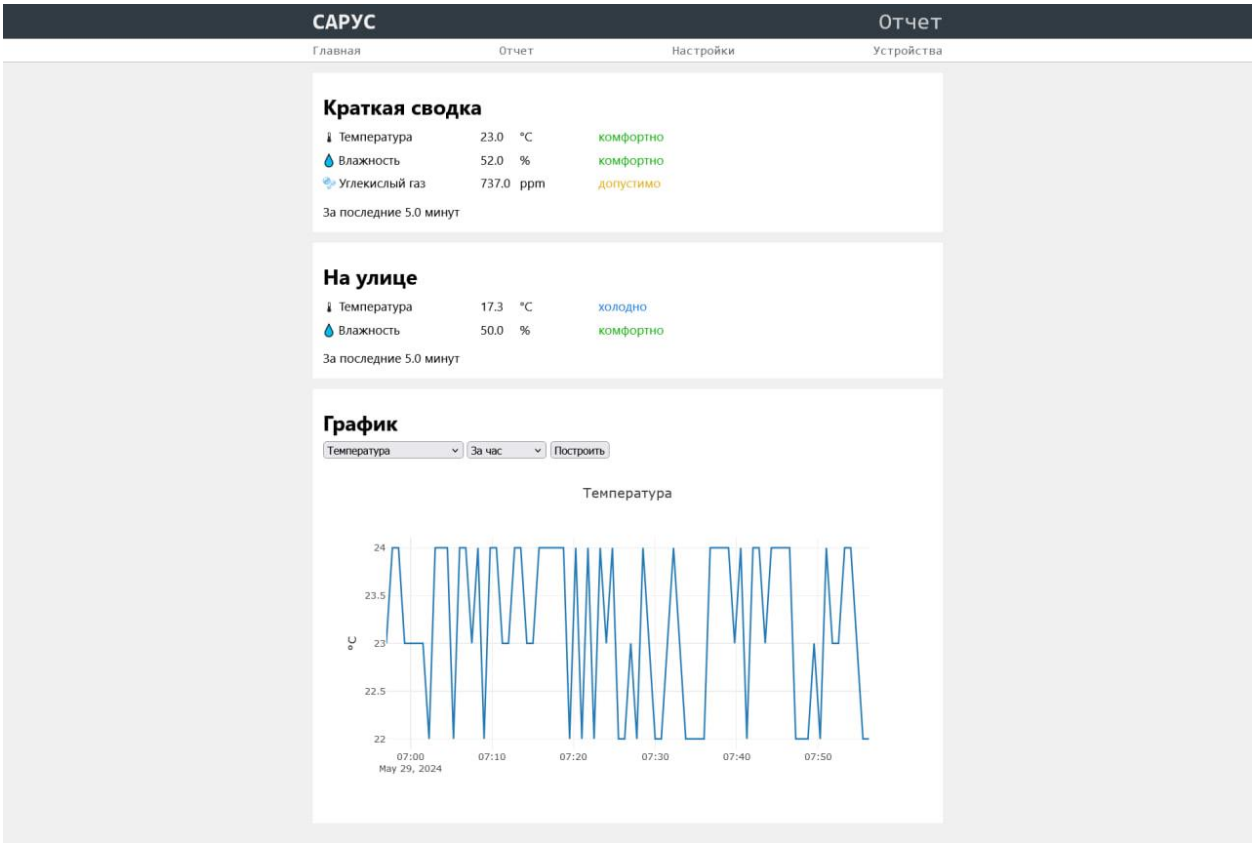


Рисунок – Страница отчета

Страница «Настройки», представленная на рисунке , отвечает за параметры, которымит руководствуется Система Автоматического Регулирования. К этим параметрам относятся граничные значения оптимального диапазона температуры, влажности и концентрации углекислого газа. Так же здесь устанавливаются периоды, за который формируются отчеты и на который строится прогноз.

Отдельно отметим, что строящийся системой прогноз не несет в себе никакой полезной информации для пользователя и нужен только для функционирования САР. Прогноз отображает тенденцию изменения величины и дает оценку, выйдет она за оптимальный диапазон через заданный промежуток времени или нет. Поэтому в отчете для пользователя он не выводится.

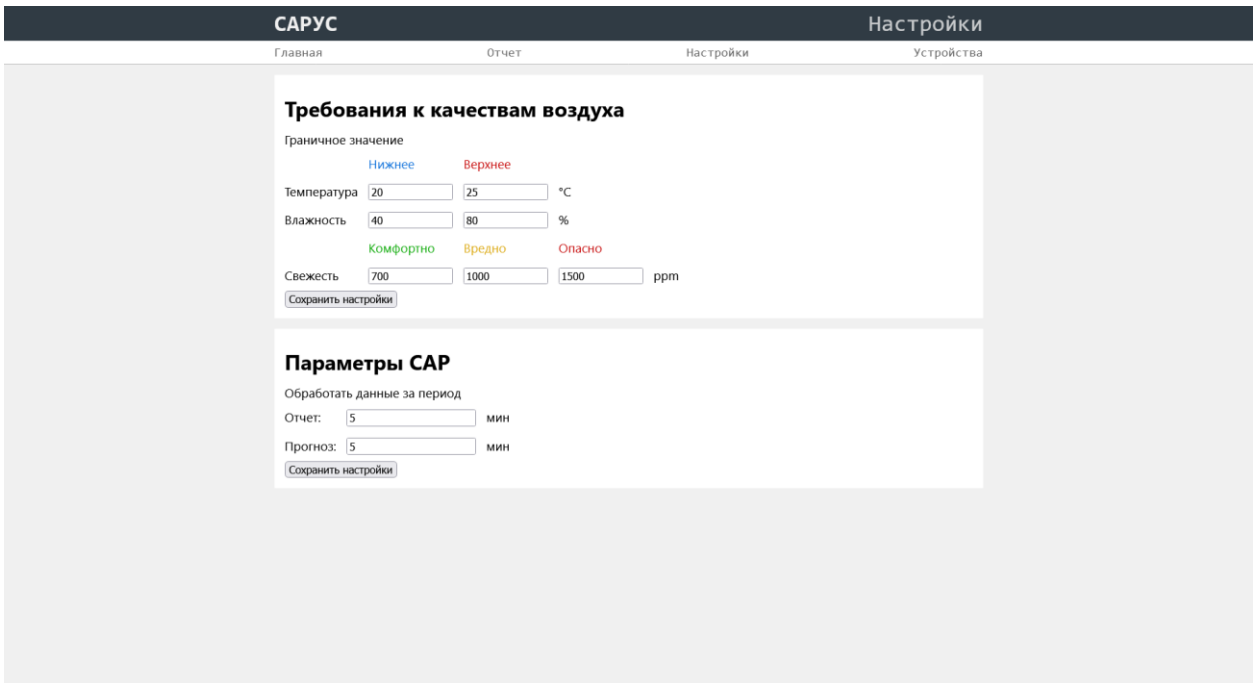


Рисунок – Страница настроек САР

Страница «Устройства», представленная на рисунке , отвечает за настройку ИУ и датчиков. В частности, включением и выключением ИУ, включением и выключением перехвата управления ИУ САР, указанием нового токена безопасности и сетевого адреса ИУ. Относительно датчиков интерфейс позволяет задать новый токен безопасности или удалить его, если он скомпрометирован. Так же здесь можно сымитировать запрос от датчика.

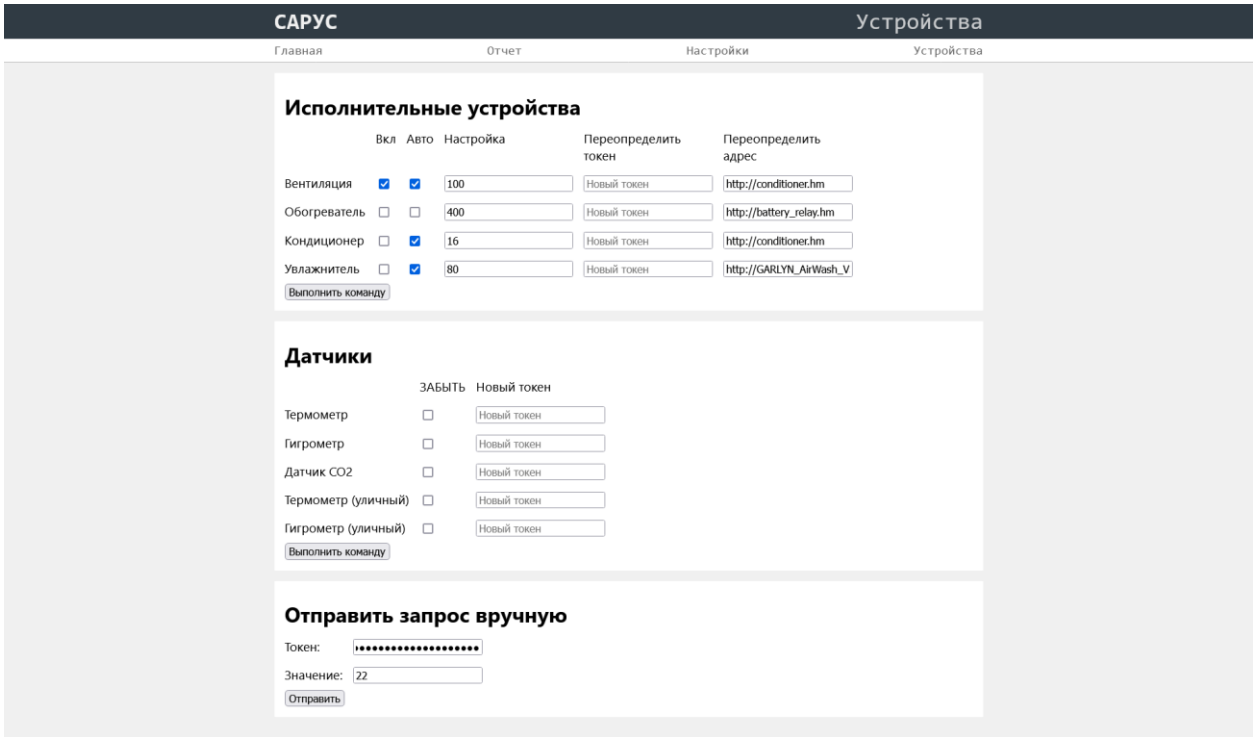


Рисунок – Страница настроек ИУ и датчиков

Страница ошибки 404 «Не найдено», представленная на рисунке , говорит пользователю, что он ошибся адресом и пытается получить доступ к несуществующему ресурсу.

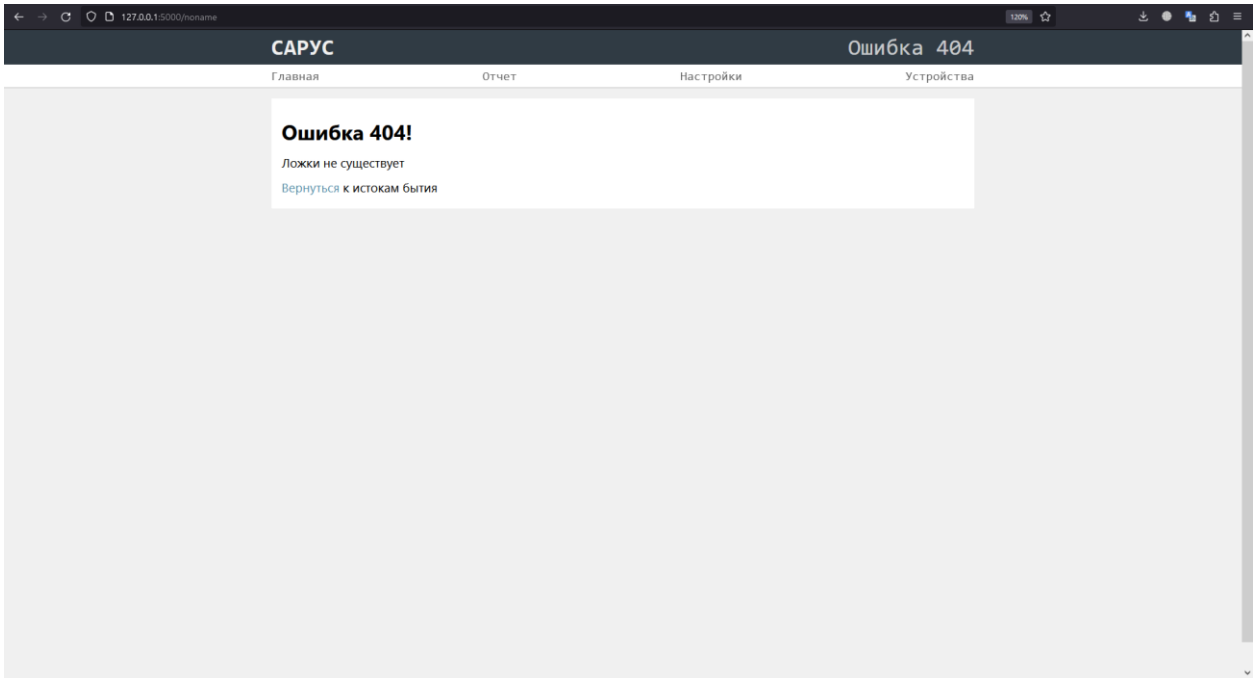


Рисунок – Страница ошибки 404 «Не найдено»

7 Вывод

Система «САРУС», безусловно, по уровню качества низка: как скуден её функционал, так и кодовая база использует не лучшие практики и не лучшие архитектурные решения. Из главных недостатков последнего можно отметить ограниченность системы одним набором ИУ и датчиков, и тем более – единственной комнатой. Немаловажным является и низкая отказоустойчивость: в случае отсутствия «свежих» данных с датчиков приложение не отвечает и прекращает свою работу до тех пор, пока данные не поступят. Однако эта проблема пускай и кажется критической, но на самом деле логически самоустраняемая: если данные отсутствуют, то система и её САР в любом случае окажутся бесполезны. Но решение всех этих проблем выходит далеко за рамки учебного проекта.

Также система полностью лишена обеспечения безопасности за исключением авторизации ИУ и датчиков, поскольку безопасность тех является архитектурным решением. Но решение это принято сознательно, вновь же по причине учебной природы проекта: вся система работает в изолированной локальной сети.

Главное (и это следует подчеркнуть) – система работоспособна и соответствует техническому заданию. Она корректно принимает, обрабатывает и реагирует на поступающие данные, отображает их в пользовательском интерфейсе.

В завершение отметим, что по разработке пользовательского интерфейса и его служб в пояснительной записке было уделено несправедливо мало внимания: были написаны шаблоны страниц, стили, js-скрипты и API для сообщения между фронт- и бэкендом. С их исходными кодами можно ознакомиться в приложениях .