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

**Оглавление**

[1 Задание 4](#_Toc168060062)

[2 Проект системы 6](#_Toc168060063)

[3 Описание объектной модели 8](#_Toc168060064)

[1.1 Класс DatabaseLink 8](#_Toc168060065)

[1.2 Класс Room 8](#_Toc168060066)

[1.3 Интерфейс Sensor 9](#_Toc168060067)

[1.4 Класс TemperatureSensor 11](#_Toc168060068)

[1.5 Класс HumiditySensor 11](#_Toc168060069)

[1.6 Класс TemperatureSensorOuter 12](#_Toc168060070)

[1.7 Класс HumiditySensorOuter 12](#_Toc168060071)

[1.8 Класс CO2Sensor 13](#_Toc168060072)

[1.9 Интерфейс Executor 13](#_Toc168060073)

[1.10 Класс HeaterDevice 14](#_Toc168060074)

[1.11 Класс ACDevice 14](#_Toc168060075)

[1.12 Класс VentDevice 15](#_Toc168060076)

[1.13 Класс Humidifier 15](#_Toc168060077)

[4 Описание модуля сбора данных 16](#_Toc168060078)

[5 Описание функций анализа сохраненных данных 20](#_Toc168060079)

[1.14 Функция quick\_lstsq() 20](#_Toc168060080)

[1.15 Метод get\_average() 20](#_Toc168060081)

[1.16 Метод get\_trend() 21](#_Toc168060082)

[1.17 Метод get\_forecast() 21](#_Toc168060083)

[1.18 Литералы DangerAssessment и FuzzyAssessment 21](#_Toc168060084)

[1.19 Метод make\_temperature\_assessment() 22](#_Toc168060085)

[1.20 Метод make\_humidity\_assessment() 22](#_Toc168060086)

[1.21 Метод make\_co2\_assessment() 22](#_Toc168060087)

[1.22 Метод make\_report() 23](#_Toc168060088)

[1.23 Метод autocontrol() 25](#_Toc168060089)

[1.24 Метод send\_command() 28](#_Toc168060090)

[6 Интерфейсы 29](#_Toc168060091)

[7 Вывод 33](#_Toc168060092)

# 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.

# Проект системы

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

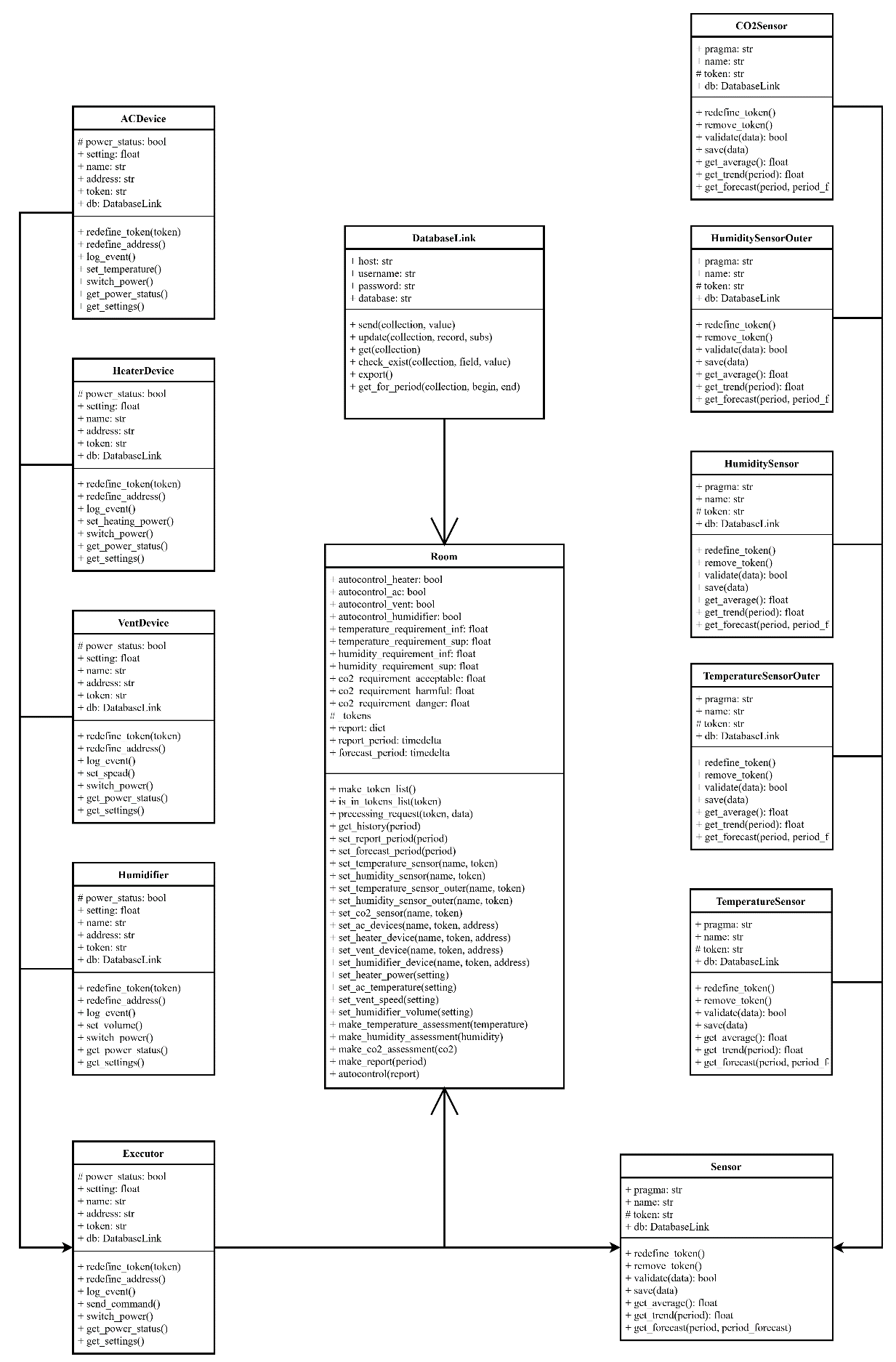


Рисунок 1 – UML-диаграмма классов серверной части системы

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.

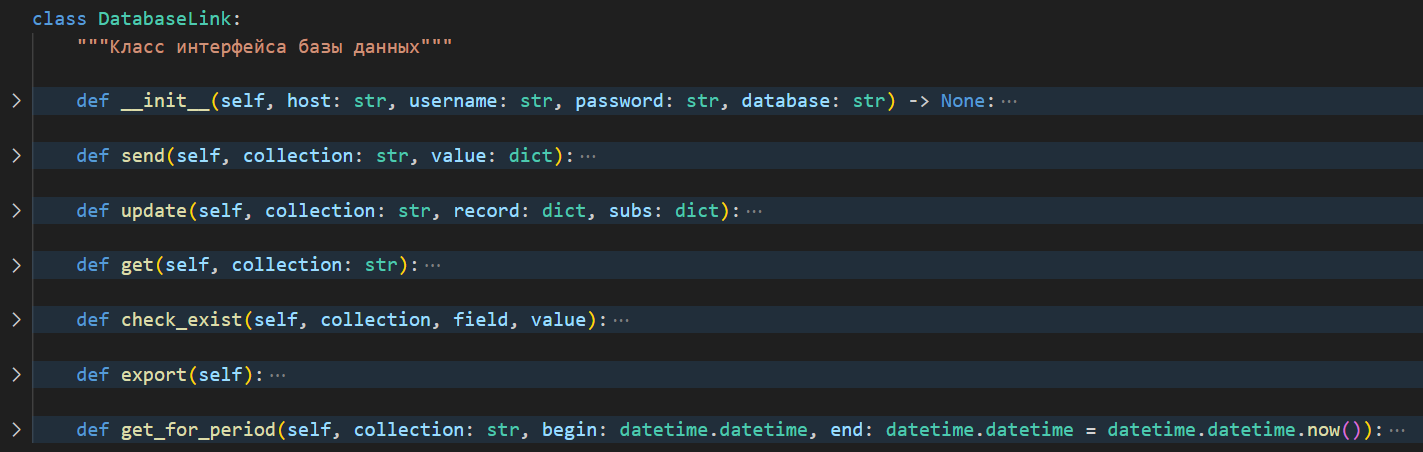
В системе присутствует четыре исполнительных устройства: HeaterDevice, ACDevice, VentDevice, Humidifier – электрообогреватель, кондиционер, активная вентиляция и увлажнитель воздуха; а также пять датчиков: TemperatureSensor, TemperatureSensorOuter, HumiditySensor, HumiditySensorOuter, CO2Sensor – датчики 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.

# Описание объектной модели

## 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.

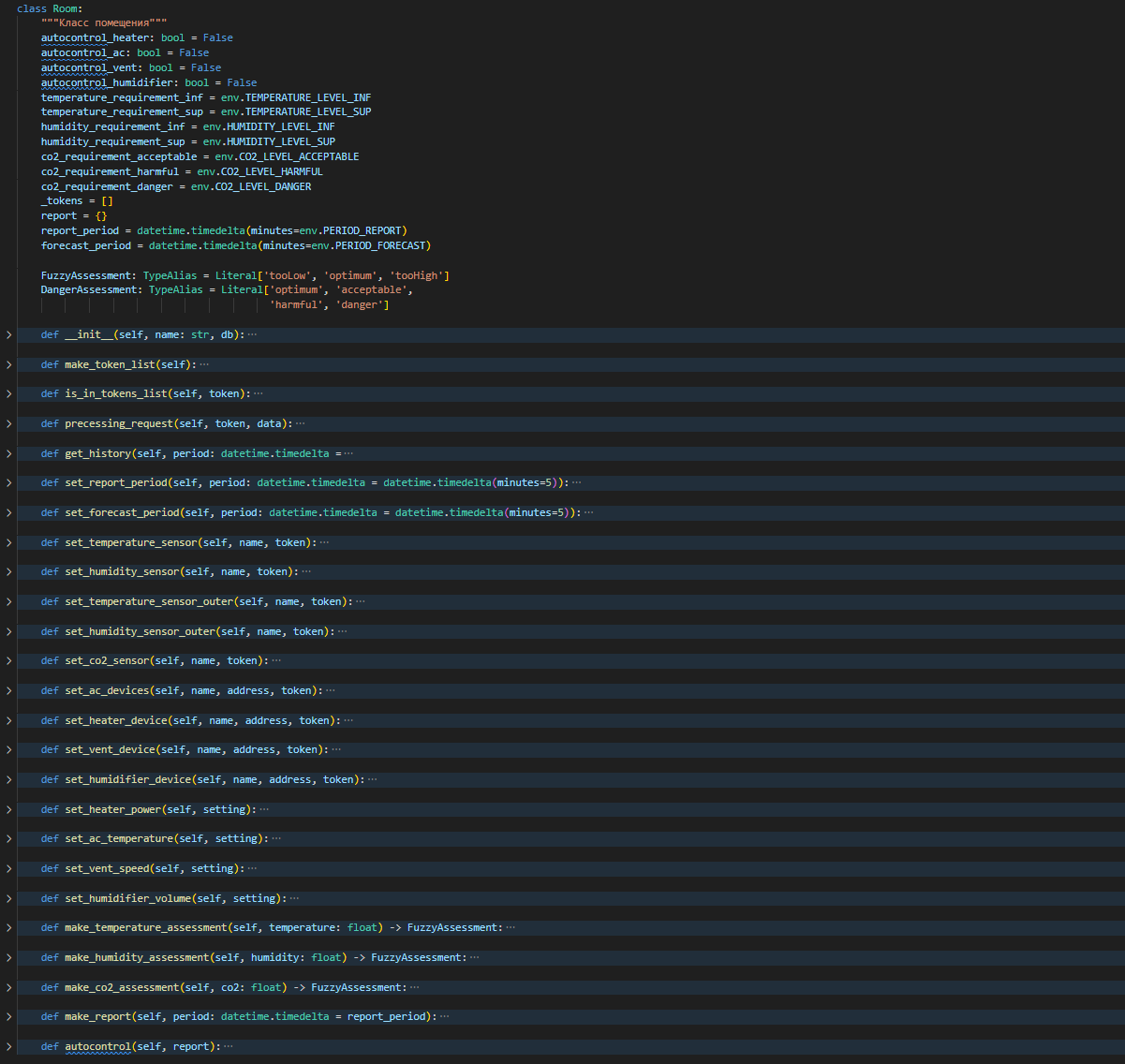


Листинг 1 – класс DatabaseLink и его методы

## 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.

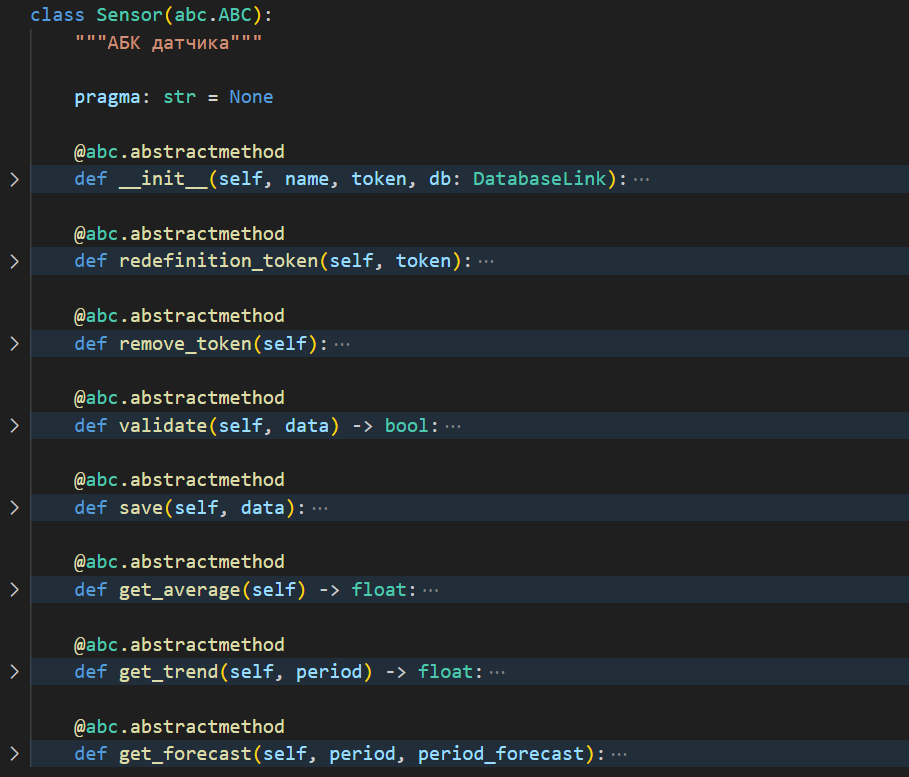
Список методов указан в листинге 2.



Листинг 2 – класс Room и его методы

## 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. Данные о доступных устройствах хранится в коллекции device БД (рисунок 2).



Листинг 3 – АБК Sensor и его методы

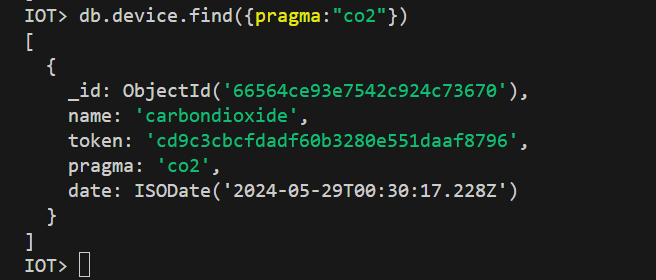
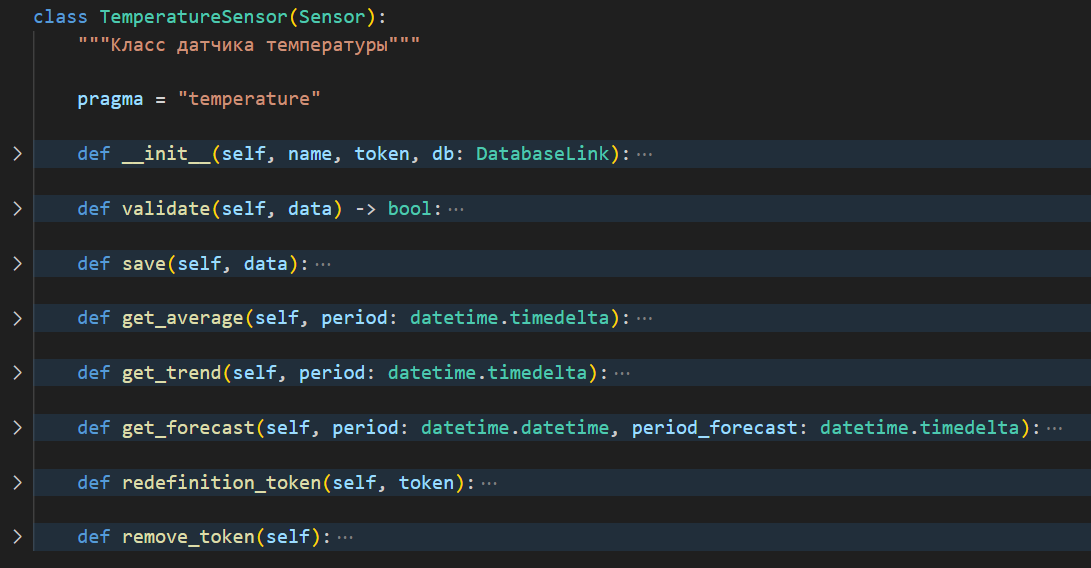


Рисунок 2 – коллекция устройств в БД

## Класс TemperatureSensor

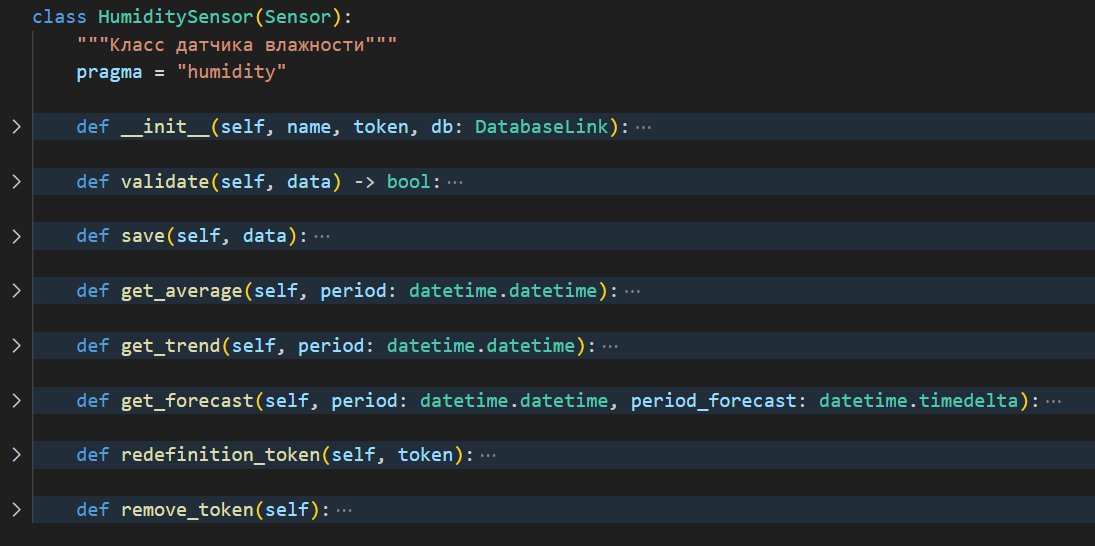
Класс датчика температуры, основан на АБК Sensor. Обрабатывает температуру воздуха в помещении. Код на листинге 4.



Листинг 4 – Класс TemperatureSensor

## Класс HumiditySensor

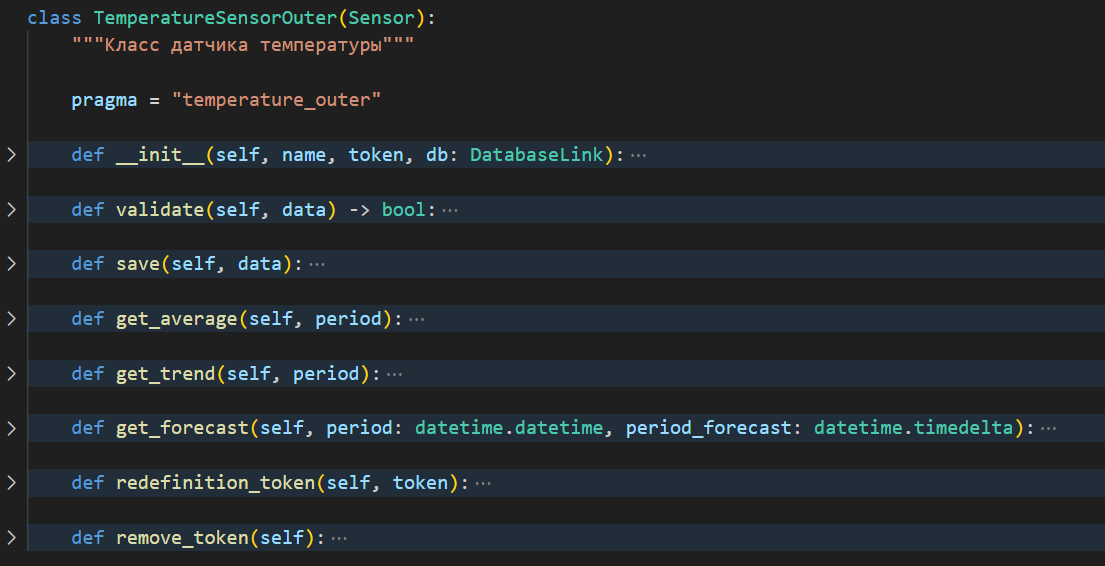
Класс датчика влажности, основан на АБК Sensor. Обрабатывает влажность воздуха в помещении. Код на листинге 5.



Листинг 5 – Класс HumiditySensor

## Класс TemperatureSensorOuter

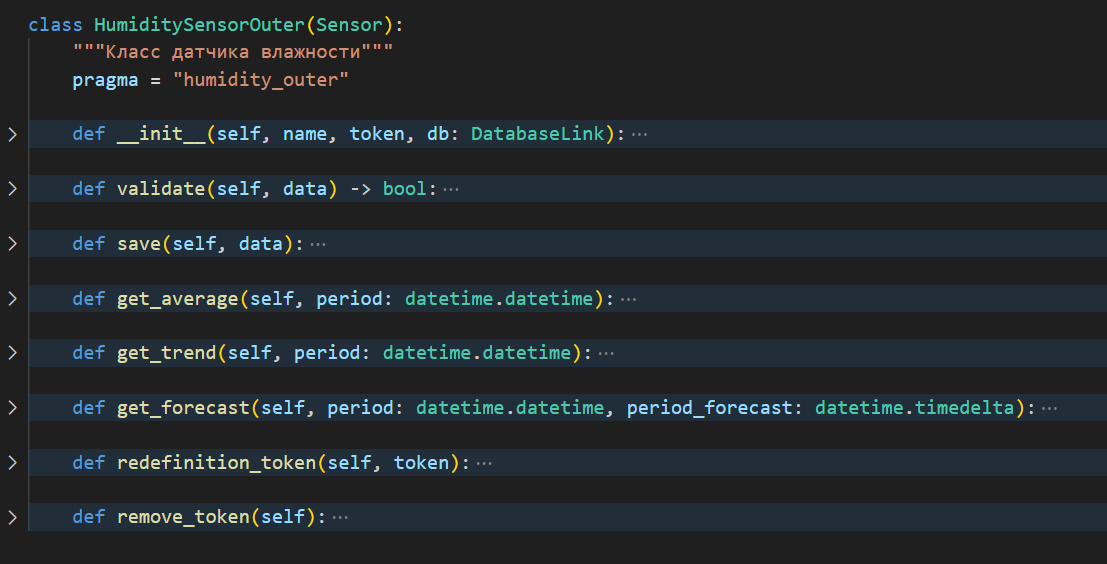
Класс датчика температуры уличного, основан на АБК Sensor. Обрабатывает температуру воздуха на улице. Код на листинге 6.



Листинг 6 – Класс TemperatureSensorOuter

## Класс HumiditySensorOuter

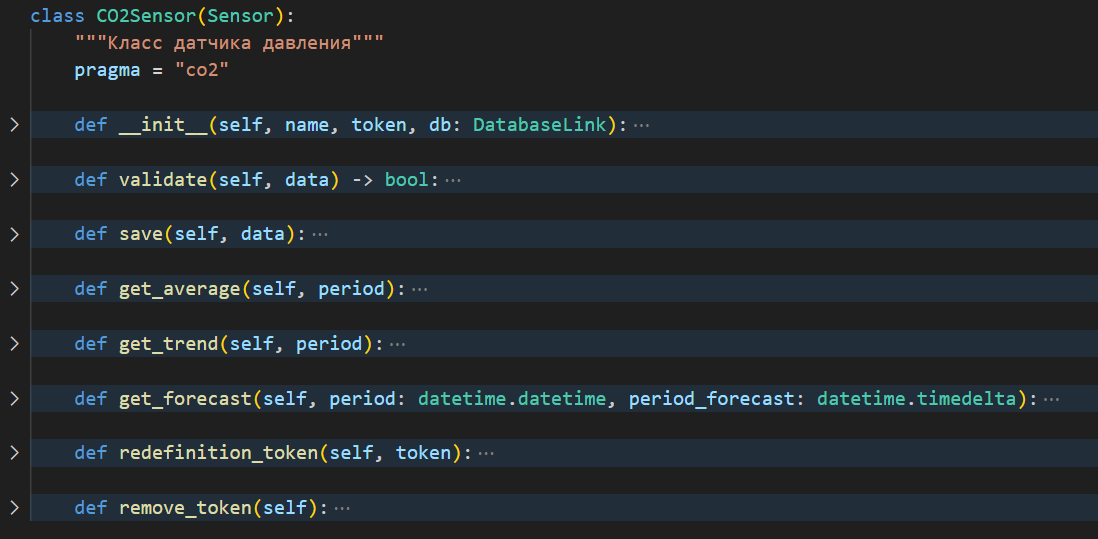
Класс датчика влажности уличного, основан на АБК Sensor. Обрабатывает влажность воздуха на улице. Код на листинге 7.



Листинг 7 – Класс HumiditySensorOuter

## Класс CO2Sensor

Класс датчика углекислого газа, основан на АБК Sensor. Обрабатывает концентрацию углекислого газа воздуха в помещении. Код на листинге 8.

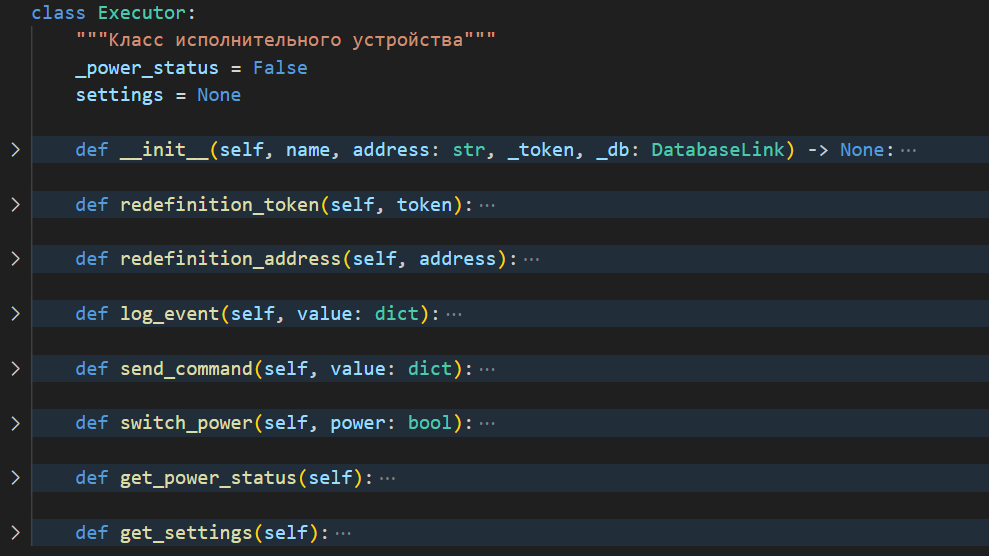


Листинг 8 – Класс CO2Sensor

## Интерфейс Executor

Класс интерфейса исполнительного устройства. Отвечает за отправку команд и установок удаленному ИУ и ведение журнала команд. Методы интерфейса на листинге 9.

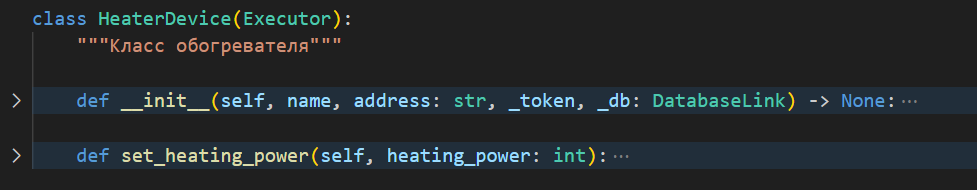
В данном случае мы прибегнули к не лучшей практике разработки: с одной стороны, Executor не должен иметь непосредственных реализаций и служит только как родитель для классов других исполнительных устройств. С другой же эти конкретные классы ИУ отличаются своим поведением незначительно, определяя собственные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



Листинг 9 – Интерфейс Executor и его методы

## Класс HeaterDevice

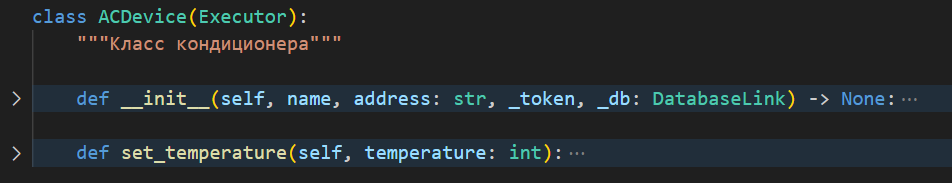
Класс обогревателя, наследуется от интерфейса Executor. Код на листинге 10.



Листинг 10 – Класс HeaterDevice

## Класс ACDevice

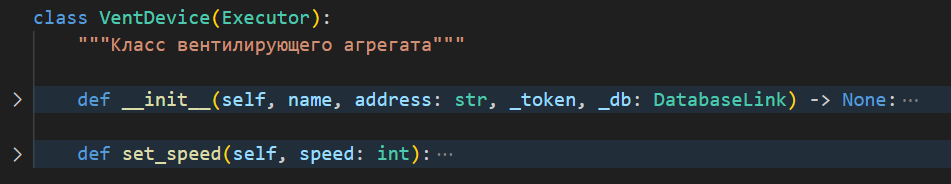
Класс кондиционера, наследуется от интерфейса Executor. Код на листинге 11.



Листинг 11 – Класс ACDevice

## Класс VentDevice

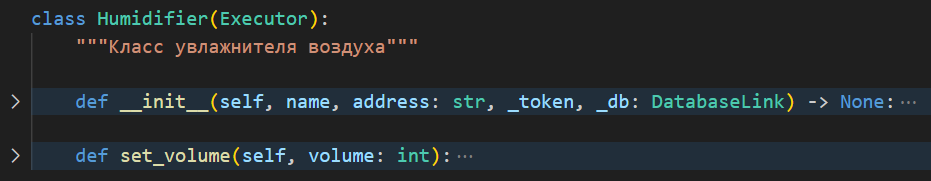
Класс вентиляции, наследуется от интерфейса Executor. Код на листинге 12.



Листинг 12 – Класс VentDevice

## Класс Humidifier

Класс увлажнителя воздуха, наследуется от интерфейса Executor. Код на листинге 13.



Листинг 13 – Класс Humidifier

# 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. Соответствующая функция на листинге 14.



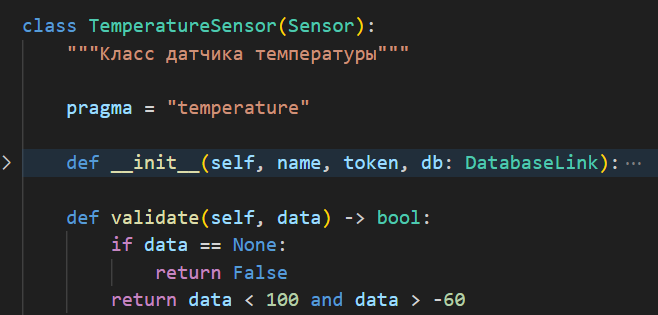
Листинг 14 – Функция api\_device\_send()

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.

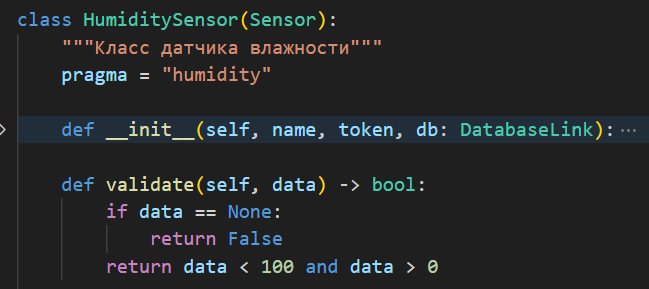
Метод save() интерфейса Sensor валидирует данные (листинги с 16 по 18) и обращается к базе данных через объект класса DatabaseLink (листинг 19 и 20). Суть валидации заключается в проверке величины на существование (если величина None, то возвращается False) и вхождение в диапазон правдоподобных значений. Полученная величина сохраняется объектом датчика в соответствующую коллекцию (рисунок ). Все используемые коллекции выведены на рисунке .



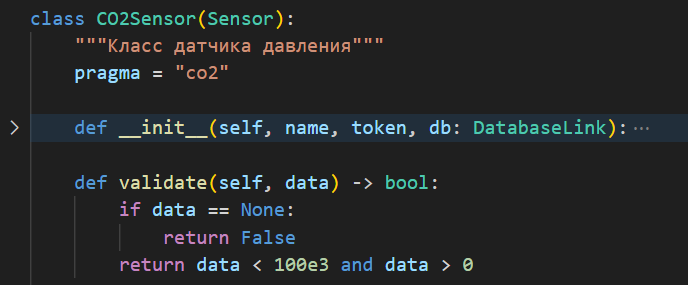
Листинг 15 – Метод processing\_request()



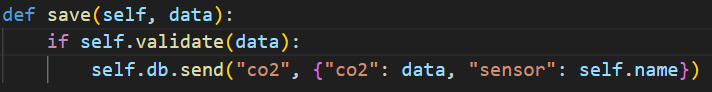
Листинг 16 – Метод validate() класса TemperatureSensor



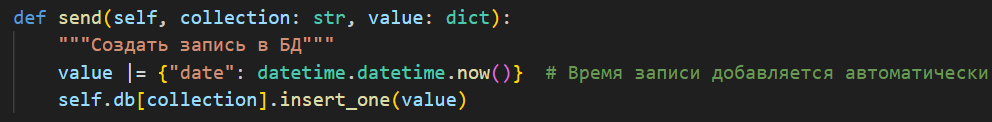
Листинг 17 – Метод validate() класса HumiditySensor



Листинг 18 – Метод validate() класса CO2Sensor

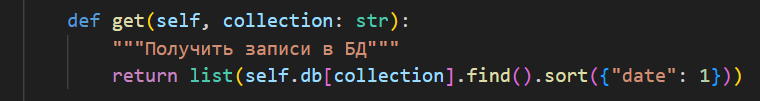


Листинг 19 – Метод save() (на примере класса CO2Sensor)



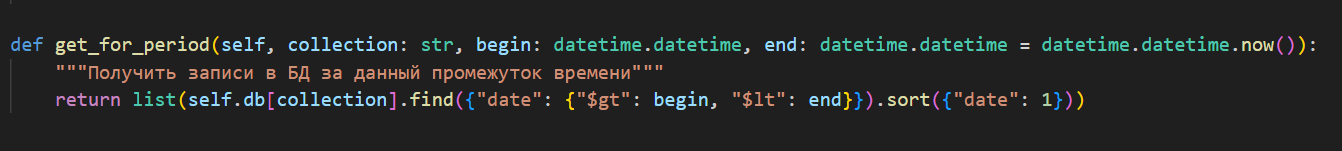
Листинг 20 – Метод send() класса DatabaseLink

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



Листинг 21 – 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.



Листинг 22 – Метод for\_perion() класса DatabaseLink



Рисунок 3 – Коллекции системы «САУРС» в БД

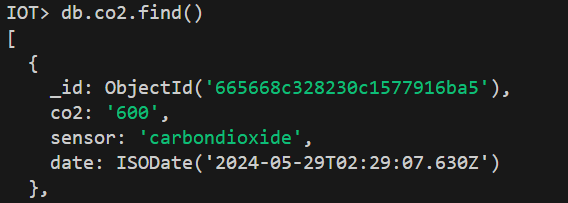
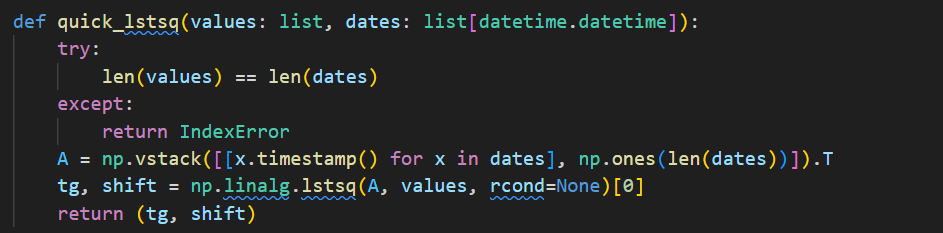


Рисунок 4 – Коллекция измерений в БД (на примере углекислого газа)

# Описание функций анализа сохраненных данных

## Функция quick\_lstsq()

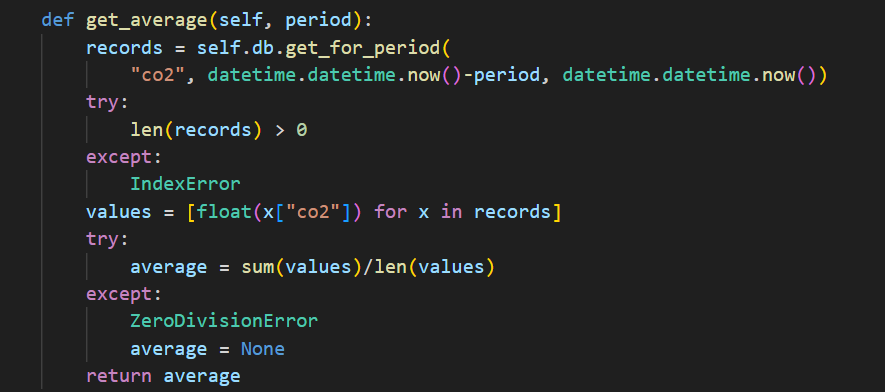
Данная функция принимает на вход массив величин и массив меток времени и находит линейную функцию методом наименьших квадратов. Функция возвращает две величины: тангенс наклона прямой и её смещение. Код функции в листинге 23. Внутри используется функция lstsq модуля numpy.



Листинг 23 – Функция quick\_lstsq()

## Метод get\_average()

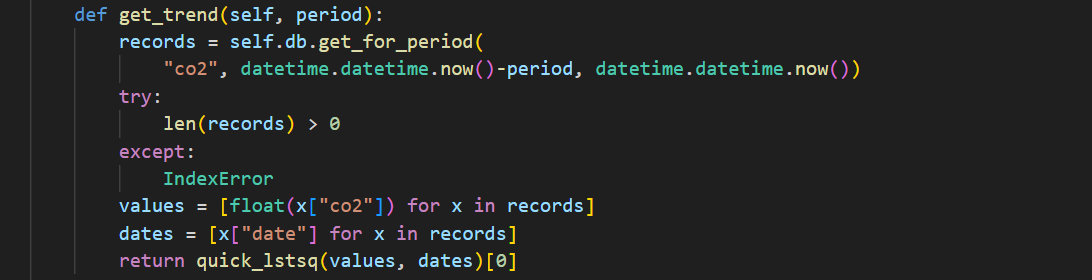
Метод get\_average() интерфейса Sensor. Возвращает среднее арифметическое измерений за указанный промежуток времени. Реализации отличаются только коллекцией БД, из которой объект извлекает данные соответствующей ему физической величины. В листинге 24 продемонстрирован метод на примере класса CO2Sensor.



Листинг 24 – Метод get\_average()

## Метод get\_trend()

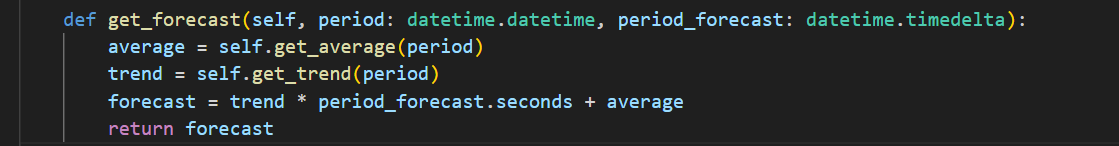
Метод get\_trend() интерфейса Sensor. Возвращает производную по времени среднего изменения физической величины за указанный промежуток времени. Реализации отличаются только коллекцией БД, из которой объект извлекает данные соответствующей ему физической величины. В листинге 25 продемонстрирован метод на примере класса CO2Sensor.



Листинг 25 – Метод get\_trend()

## Метод get\_forecast()

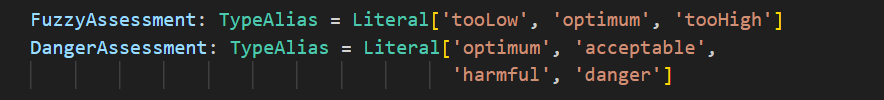
Метод get\_forecast() интерфейса Sensor. Возвращает экстраполяцию физической величины через указанный промежуток времени по её производной по времени среднего изменения этой величины. Реализации во всех классах идентичны. Код приведен на листинге 26.



Листинг 26 – Метод get\_forecast()

## Литералы DangerAssessment и FuzzyAssessment

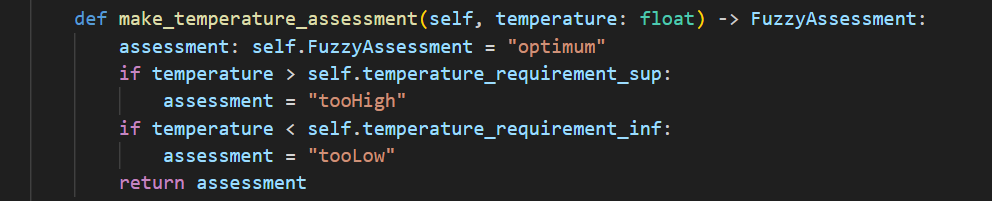
Литералы серии Assessment – это оценки нечеткой логики, характеризующие физическую величину. Литерал FuzzyAssessment дает оценку, находится величина в оптимальном диапазоне или выходит за его пределы: «слишком мало», «комфортно», «слишком много». В контексте температуры крайние значения можно трактовать как «холодно» и «жарко», а в контексте влажности – «сухо» и «сыро» соответственно. Литерал DangerAssessment дает оценку опасности физической величины: «безопасно», «допустимо», «вредно» и «опасно». Определения на листинге 27.



Листинг 27 – Литералы DangerAssessment и FuzzyAssessment

## Метод make\_temperature\_assessment()

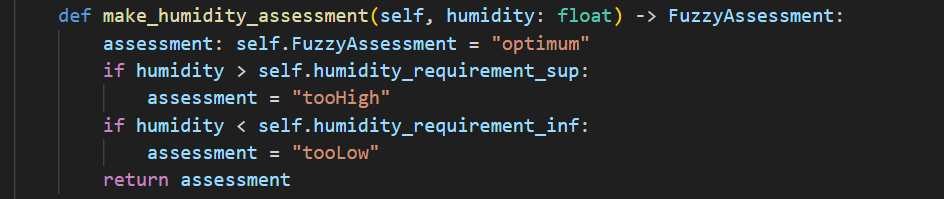
Метод make\_temperature\_assessment() принимает значение температуры и возвращает её оценку в терминах литерала FuzzyAssessment. Код метода на листинге 28.



Листинг 28 – Метод make\_temperature\_assessment()

## Метод make\_humidity\_assessment()

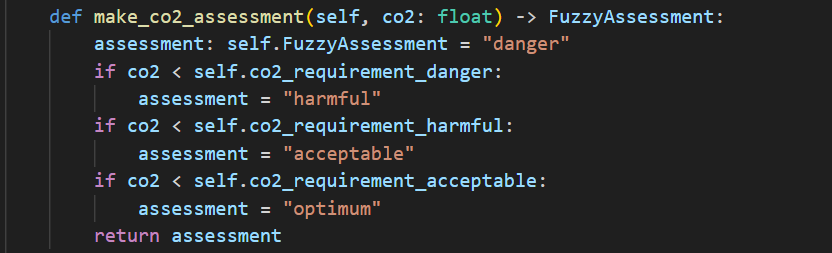
Метод make\_humidity\_assessment() принимает значение влажности и возвращает её оценку в терминах литерала FuzzyAssessment. Код метода на листинге 29.



Листинг 29 – Метод make\_ humidity \_assessment()

## 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 30.

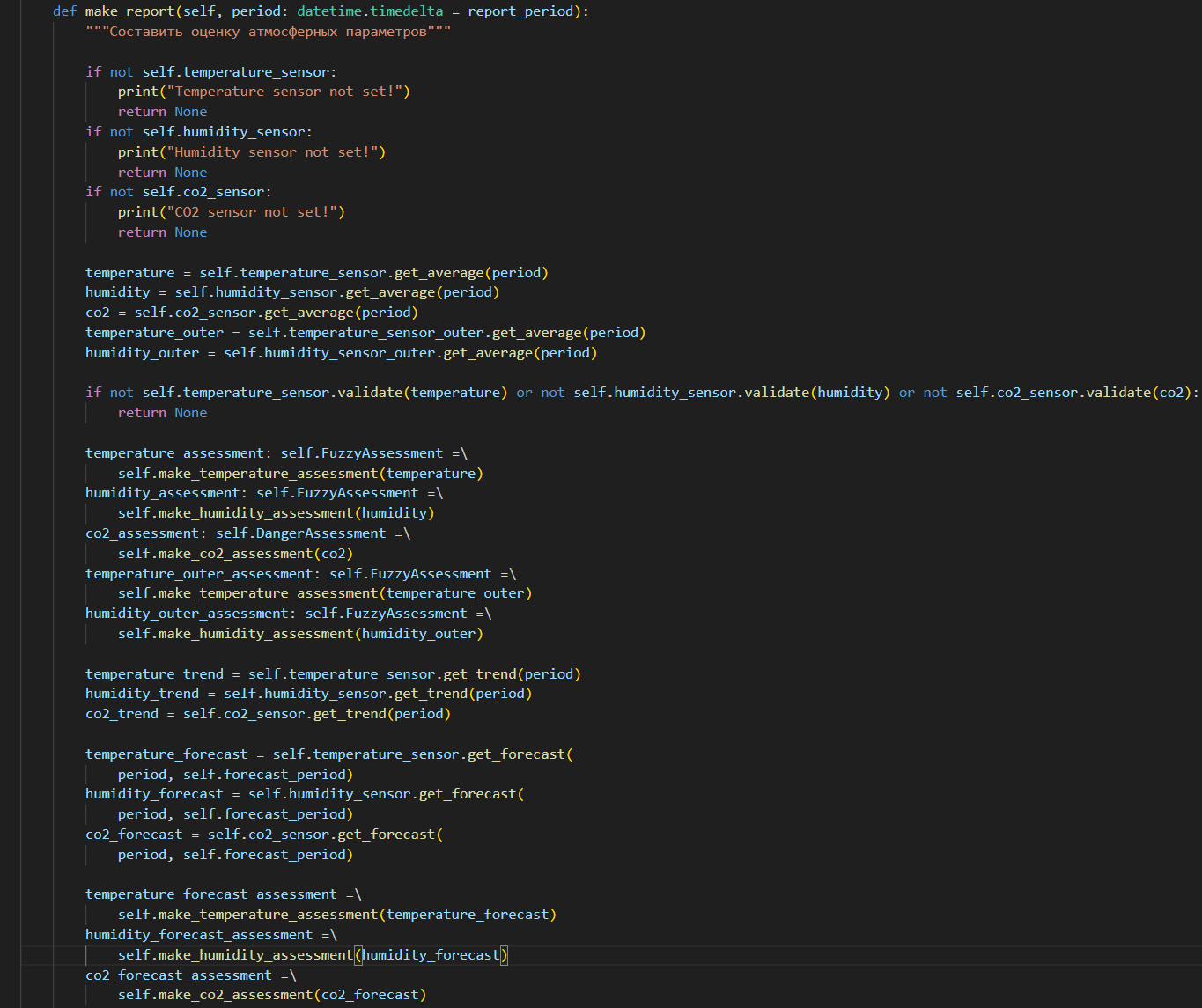


Листинг 30 – Метод make\_ co2 \_assessment()

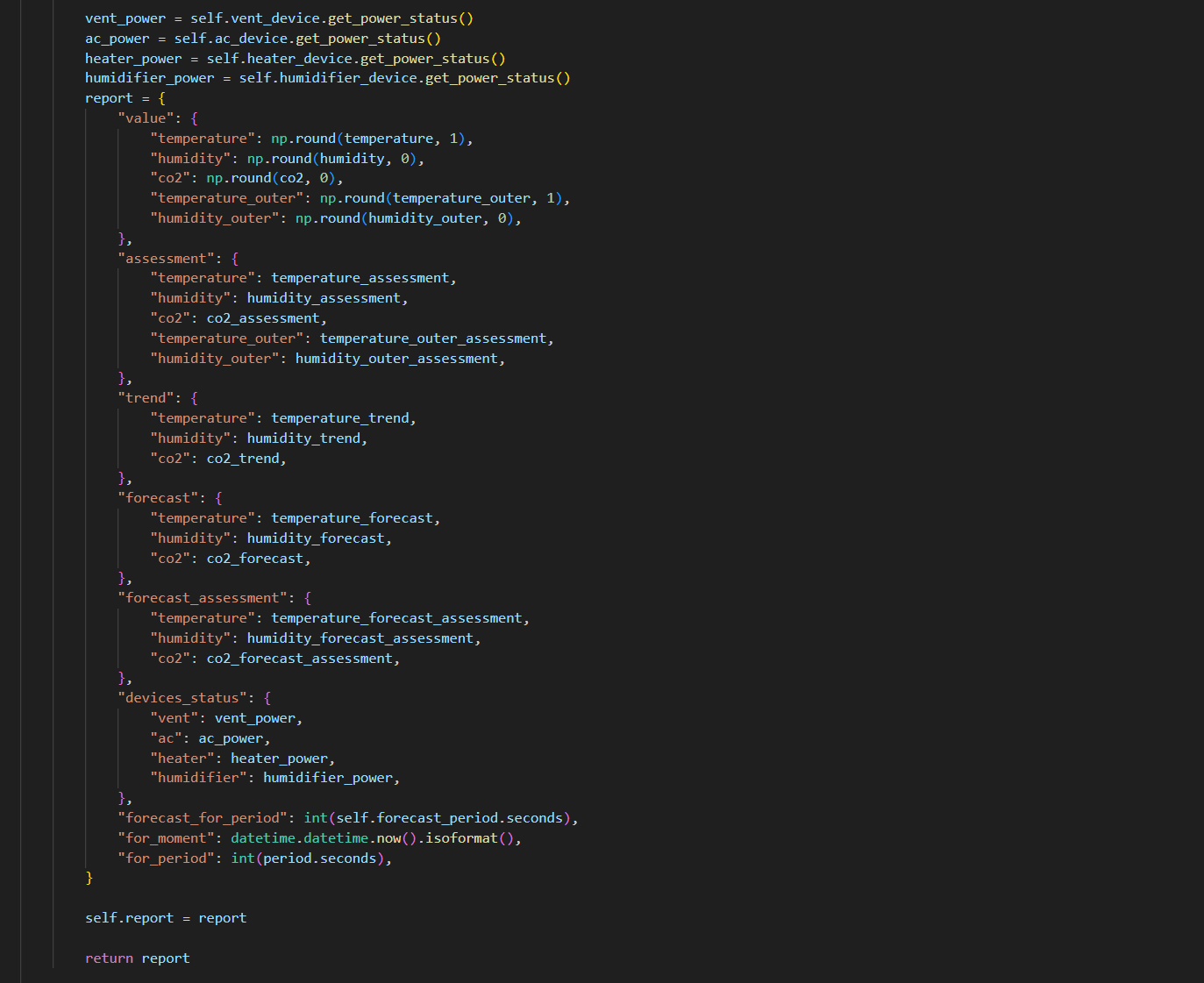
## Метод make\_report()

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 31.

С примером сформированного отчета можно ознакомиться на рисунке 5.



Листинг 31.1 – Метод make\_report() (часть первая из двух)



Листинг 31.2 – Метод make\_report() (часть вторая из двух)

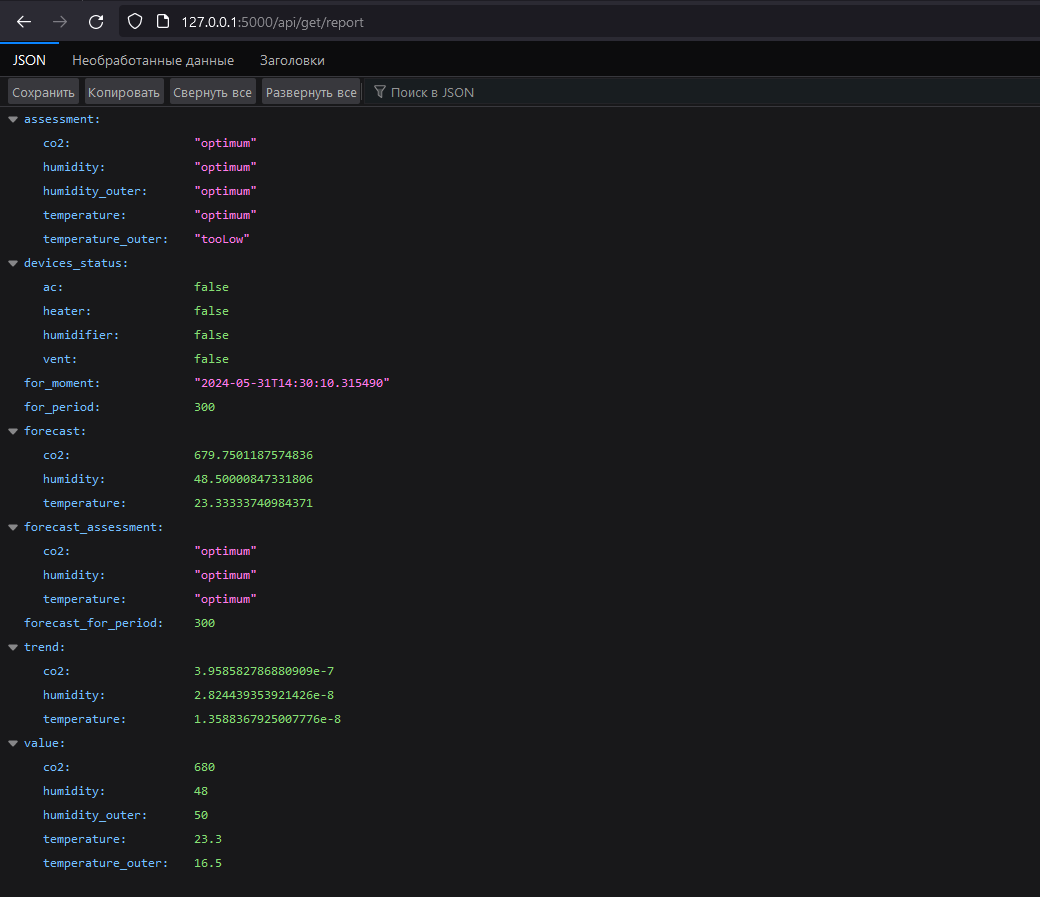


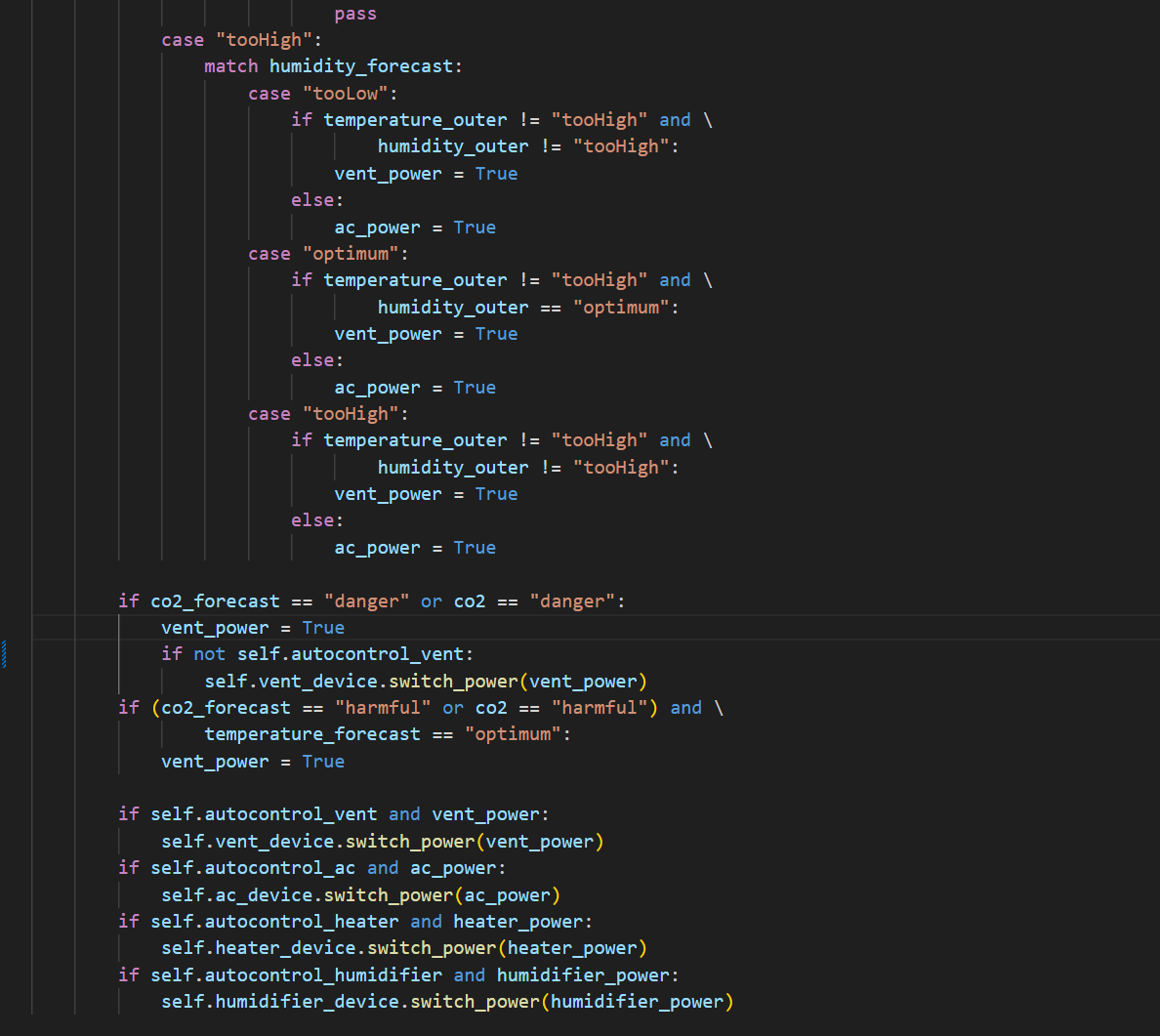
Рисунок 5 – Пример машиночитаемого отчета

## Метод autocontrol()

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



Листинг 32.1 – Метод autocontrol() (часть первая из двух)



Листинг 32.2 – 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.

## Метод send\_command()

Метод send\_command() интерфейса Executor принимает словарь параметров и осуществляет POST-запрос на ИУ. Метод используется методами переключения питания switch\_power() и методами задания установок серии set классов, наследственных от интерфейса Executor.

Метод log\_event() интерфейса Executor является оберткой для метода send() класса DatabaseLink, вызывается при каждом вызове метода send\_command() и выполняет функцию ведения журнала отданных ИУ команд.

С примером записи в журнале можно ознакомиться на рисунке 6.

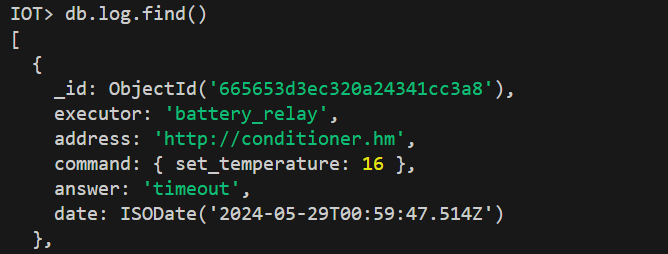


Рисунок 6 – Коллекция журнала команд в БД

# Интерфейсы

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

Страница «Главная», представленная на рисунке 7, встречает пользователя и дает общее представление о системе, её функциях и возможностях.

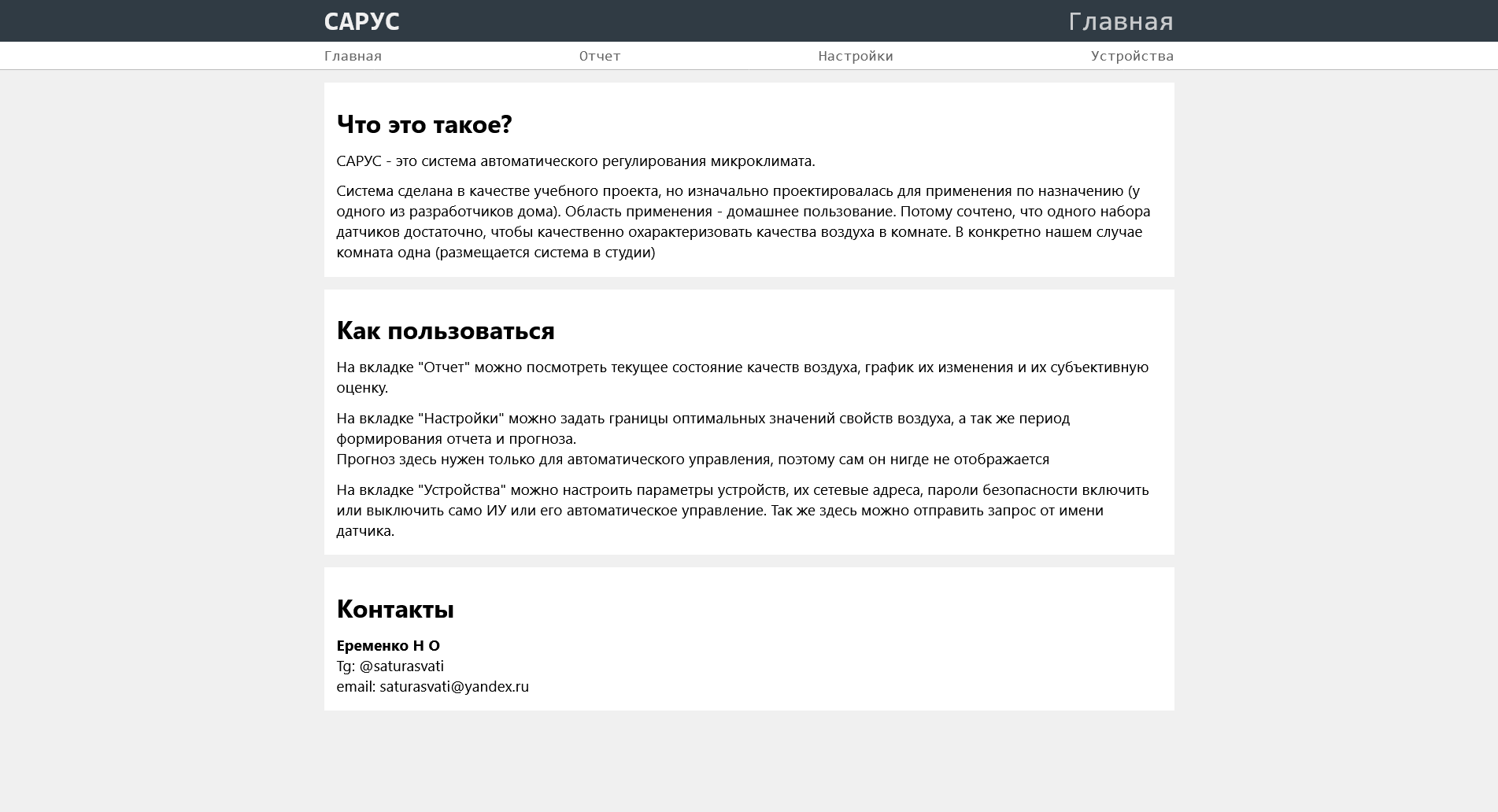


Рисунок 7 – Страница приветствия

Страница «Отчет», представленная на рисунке 8, 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).

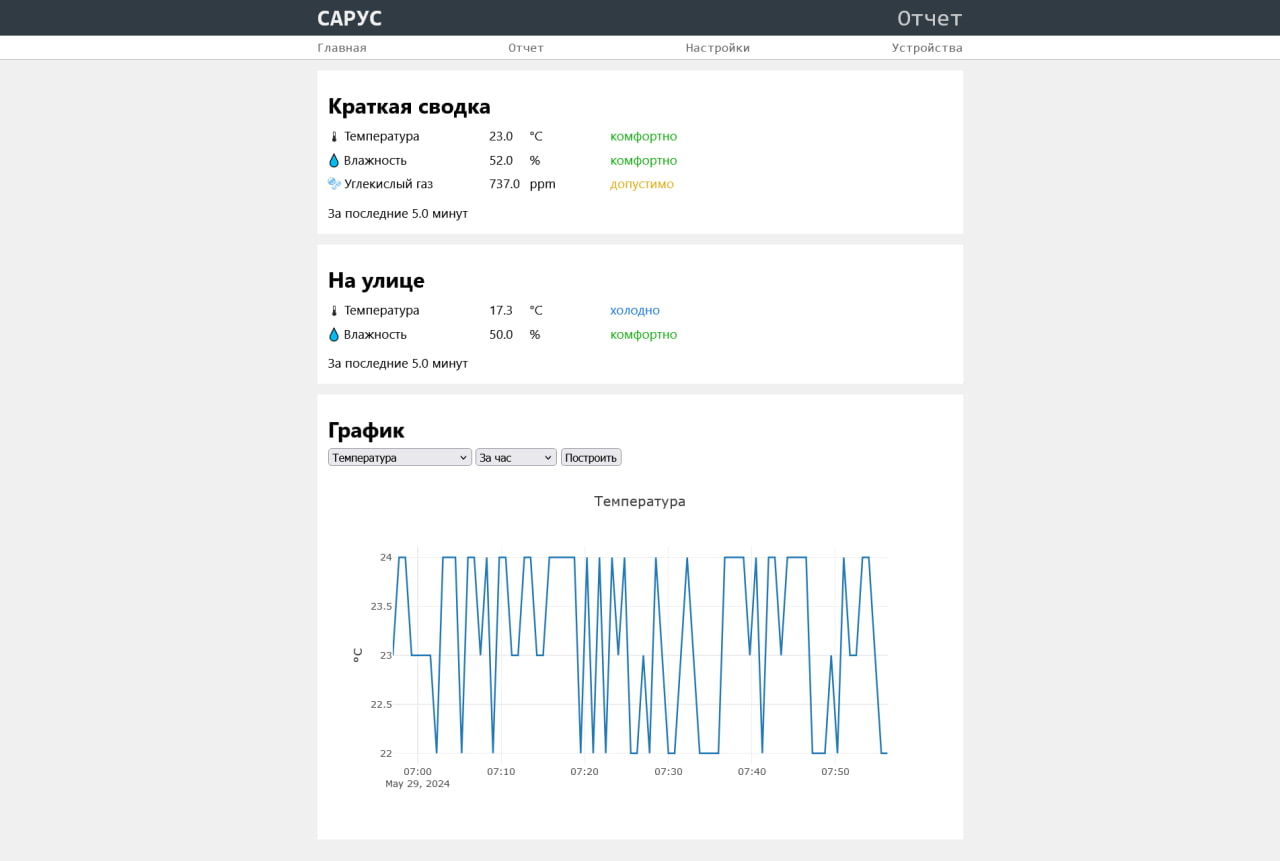


Рисунок 8 – Страница отчета

Страница «Настройки», представленная на рисунке 9, отвечает за параметры, которыми 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.

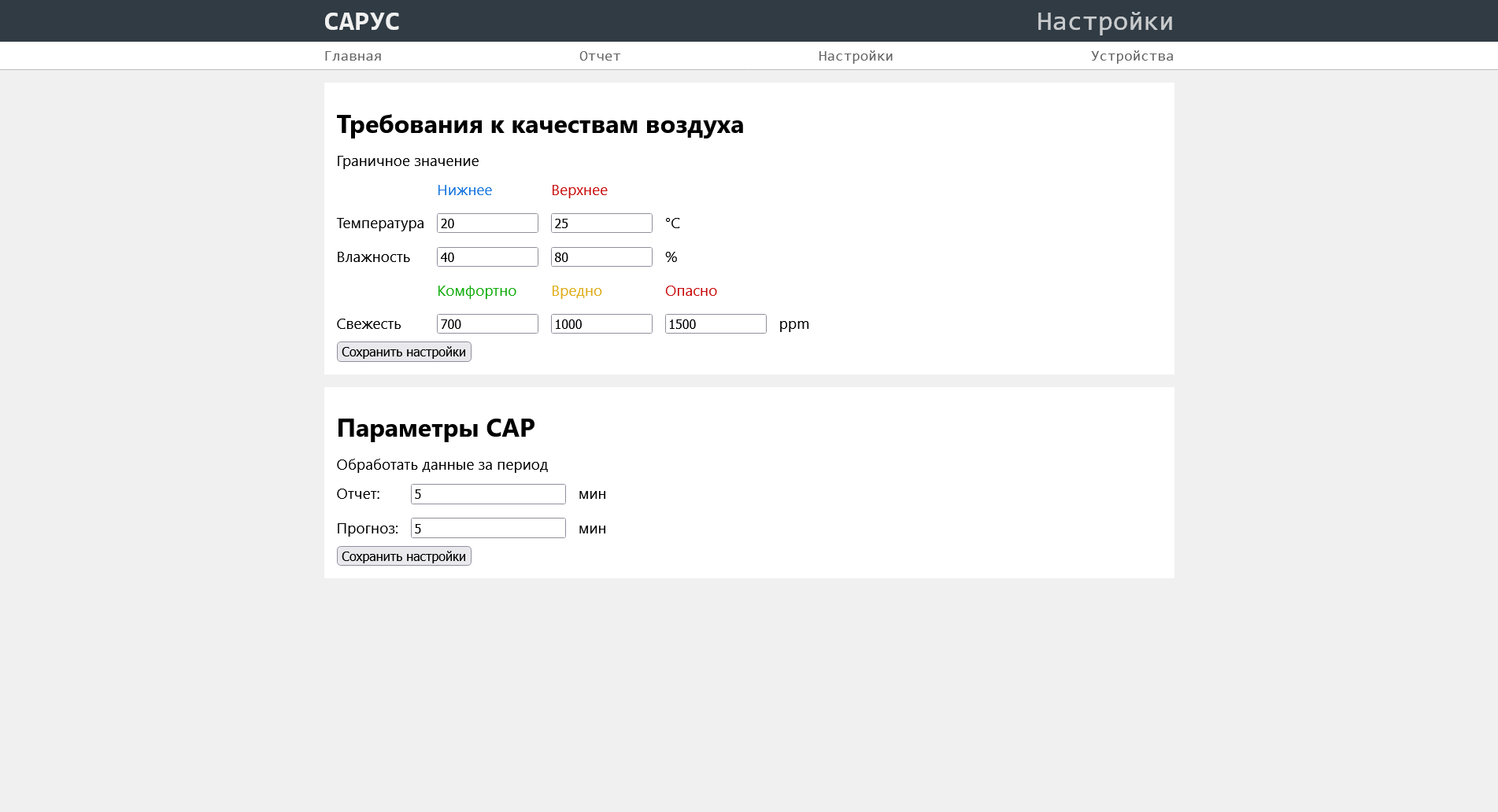


Рисунок 9 – Страница настроек САР

Страница «Устройства», представленная на рисунке 10, 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.

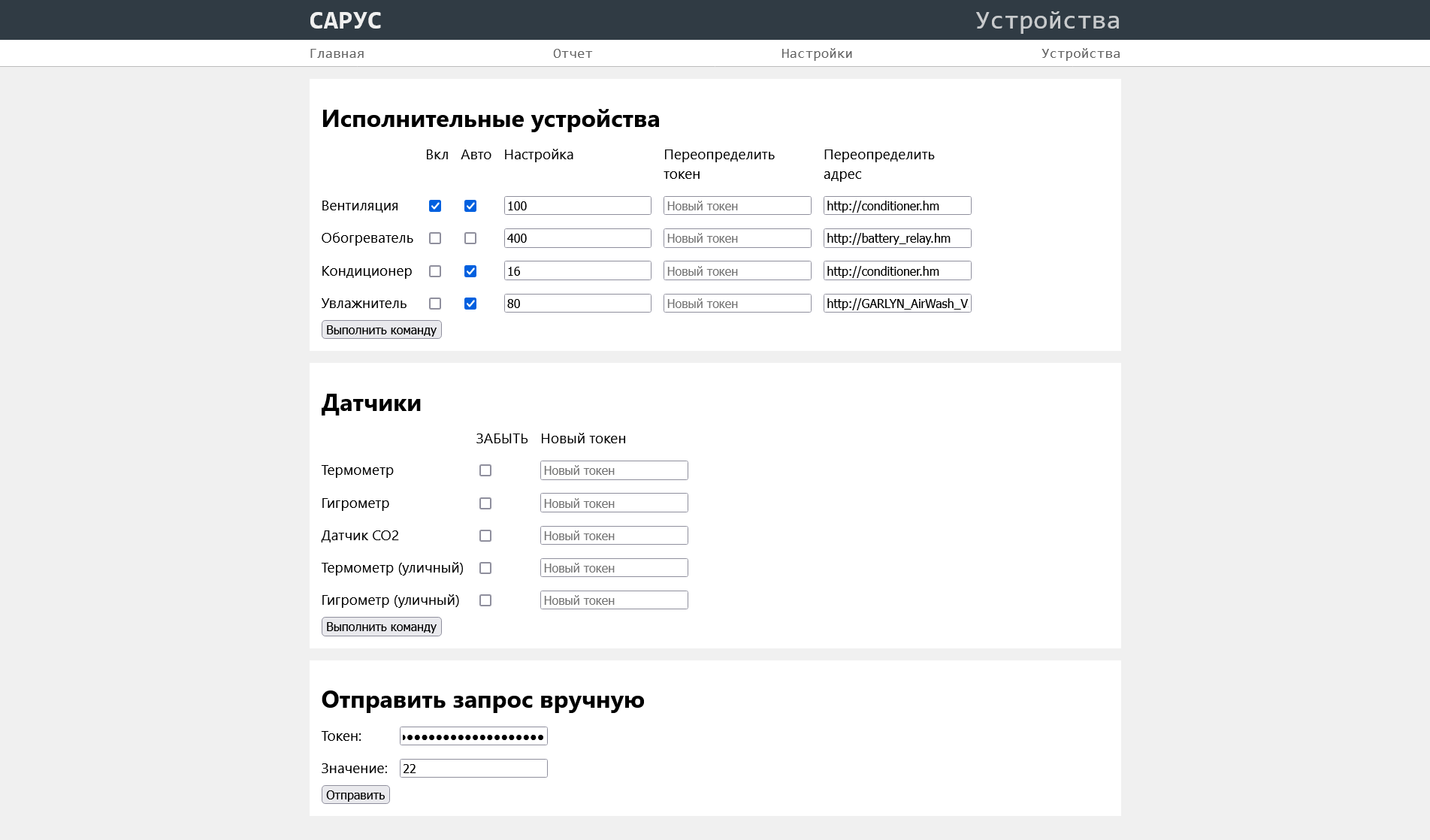


Рисунок 10 – Страница настроек ИУ и датчиков

Страница ошибки 404 «Не найдено», представленная на рисунке 11, говорит пользователю, что он ошибся адресом и пытается получить доступ к несуществующему ресурсу.

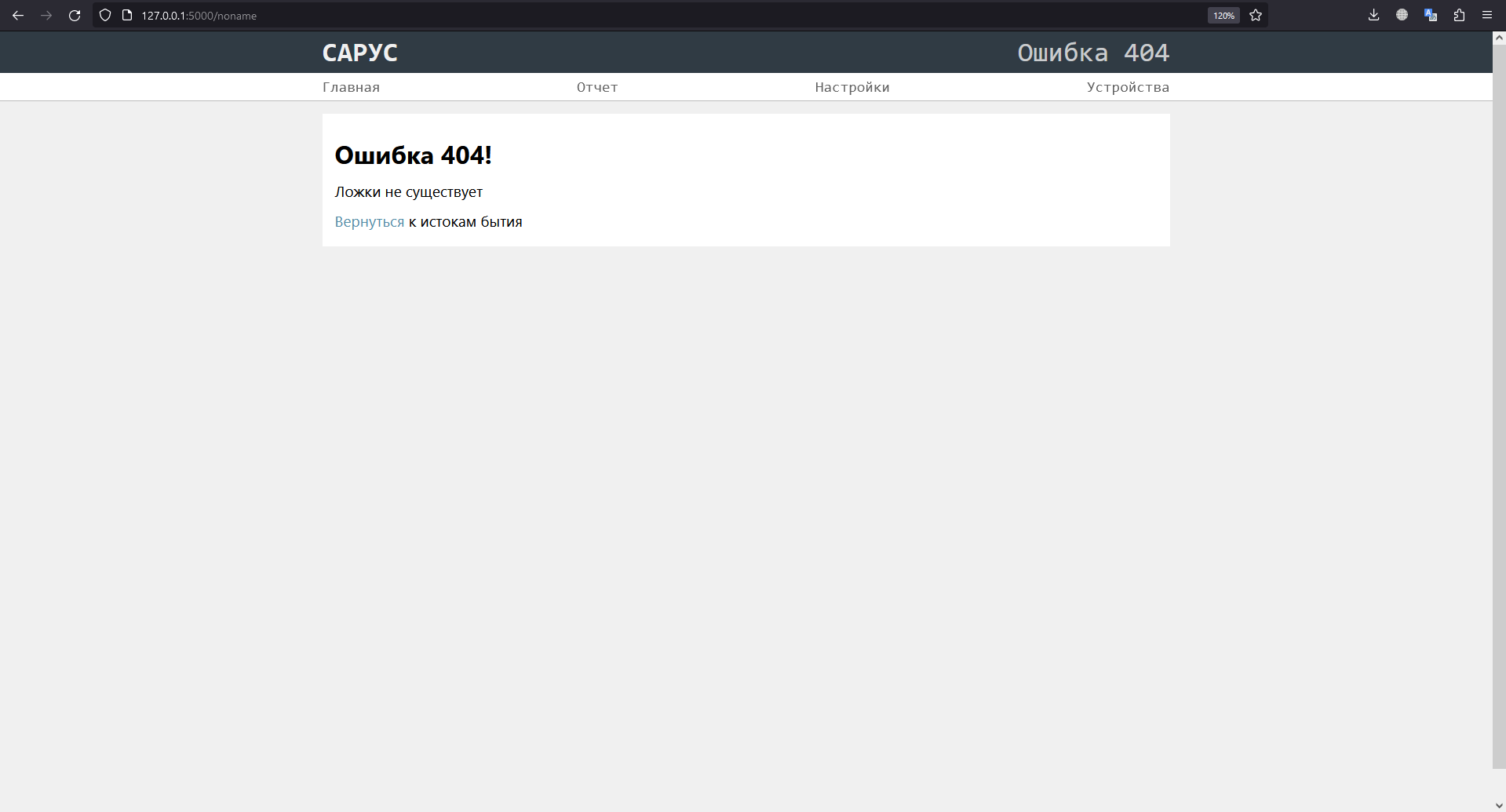


Рисунок 11 – Страница ошибки 404 «Не найдено»

# 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 с А по .

Приложение А

Листинг файла main.py

from flask import Flask, render\_template, request, Response

from classes import \*

import datetime

import env

# Prepare

db = DatabaseLink(database=env.DB\_DATABASE, host=env.DB\_HOST,

username=env.DB\_USERNAME, password=env.DB\_PASSWORD)

# Make room "Main"

room = Room("main", db)

room.set\_temperature\_sensor("temperature",

env.SENSOR\_TEMPERATURE\_TOKEN)

room.set\_humidity\_sensor("humidity",

env.SENSOR\_HUMIDITY\_TOKEN)

room.set\_temperature\_sensor\_outer("temperature\_outer",

env.SENSOR\_TEMPERATURE\_OUTER\_TOKEN)

room.set\_humidity\_sensor\_outer("humidity\_outer",

env.SENSOR\_HUMIDITY\_OUTER\_TOKEN)

room.set\_co2\_sensor("carbondioxide",

env.SENSOR\_CO2\_TOKEN)

room.set\_ac\_devices("battery\_relay",

env.EXECUTOR\_AC\_ADDRESS,

env.EXECUTOR\_AC\_TOKEN)

room.set\_heater\_device("conditioner",

env.EXECUTOR\_HEATER\_ADDRESS,

env.EXECUTOR\_HEATER\_TOKEN)

room.set\_vent\_device("window\_opener",

env.EXECUTOR\_VENT\_ADDRESS,

env.EXECUTOR\_VENT\_TOKEN)

room.set\_humidifier\_device("GARLYN\_AirWash\_V30",

env.EXECUTOR\_HUMIDIFIER\_ADDRESS,

env.EXECUTOR\_HUMIDIFIER\_TOKEN)

room.set\_heater\_power(400) # [W]

room.set\_ac\_temperature(16) # [degC]

room.set\_vent\_speed(100) # [%]

room.set\_humidifier\_volume(80) # [%]

room.make\_token\_list()

room.autocontrol\_vent = True

room.autocontrol\_ac = True

room.autocontrol\_heater = True

room.autocontrol\_humidifier = True

# Flask functions

app = Flask(\_\_name\_\_, static\_folder="static")

# Errors

@app.errorhandler(404)

def page\_not\_found(error):

return render\_template("error.html", error=404, comment="Ложки не существует")

# API

@app.route("/api/device/send", methods=["POST"])

def api\_device\_send():

token = request.json["Auth"]

value = request.json["value"]

code = room.processing\_request(token, value)

report = room.make\_report()

room.autocontrol(report)

return Response(status=code)

@app.route("/api/device/settings", methods=["POST"])

def api\_device\_settings():

settings: dict[dict] = request.json

if ("executor\_on" in settings and len(settings["executor\_on"])):

for exe, val in settings["executor\_on"].items():

match exe:

case "ac":

room.ac\_device.switch\_power(val)

case "vent":

room.vent\_device.switch\_power(val)

case "heater":

room.heater\_device.switch\_power(val)

case "humidifier":

room.humidifier\_device.switch\_power(val)

if ("executor\_setting" in settings and len(settings["executor\_setting"])):

for exe, val in settings["executor\_setting"].items():

val = int(val)

match exe:

case "ac":

room.ac\_device.set\_temperature(val)

case "vent":

room.vent\_device.set\_speed(val)

case "heater":

room.heater\_device.set\_heating\_power(val)

case "humidifier":

room.humidifier\_device.set\_volume(val)

if ("executor\_address" in settings and len(settings["executor\_address"])):

for exe, val in settings["executor\_address"].items():

match exe:

case "ac":

room.ac\_device.redefinition\_address(val)

case "vent":

room.vent\_device.redefinition\_address(val)

case "heater":

room.heater\_device.redefinition\_address(val)

case "humidifier":

room.humidifier\_device.redefinition\_address(val)

if ("executor\_autocontrol" in settings and len(settings["executor\_autocontrol"])):

for exe, val in settings["executor\_autocontrol"].items():

match exe:

case "ac":

print(exe, val)

room.autocontrol\_ac=val

case "vent":

print(exe, val)

room.autocontrol\_vent=val

case "heater":

print(exe, val)

room.autocontrol\_heater=val

case "humidifier":

print(exe, val)

room.autocontrol\_humidifier=val

print(room.autocontrol\_ac,room.autocontrol\_heater,room.autocontrol\_humidifier,room.autocontrol\_vent)

if ("executor\_token" in settings and len(settings["executor\_token"])):

for exe, val in settings["executor\_token"].items():

match exe:

case "ac":

room.ac\_device.redefinition\_token(val)

case "vent":

room.vent\_device.redefinition\_token(val)

case "heater":

room.heater\_device.redefinition\_token(val)

case "humidifier":

room.humidifier\_device.redefinition\_token(val)

room.make\_token\_list()

if ("sensor\_new\_token" in settings and len(settings["sensor\_new\_token"])):

for sensor, val in settings["sensor\_new\_token"].items():

match sensor:

case "temperature":

room.temperature\_sensor.redefinition\_token(val)

case "temperature\_outer":

room.temperature\_sensor\_outer.redefinition\_token(val)

case "humidity":

room.humidity\_sensor.redefinition\_token(val)

case "humidity\_outer":

room.humidity\_sensor\_outer.redefinition\_token(val)

case "co2":

room.co2\_sensor.redefinition\_token(val)

room.make\_token\_list()

if ("sensor\_remove" in settings and len(settings["sensor\_remove"])):

for sensor, val in settings["sensor\_remove"].items():

match sensor:

case "temperature":

room.temperature\_sensor.remove\_token()

case "temperature\_outer":

room.temperature\_sensor\_outer.remove\_token()

case "humidity":

room.humidity\_sensor.remove\_token()

case "humidity\_outer":

room.humidity\_sensor\_outer.remove\_token()

case "co2":

room.co2\_sensor.remove\_token()

room.make\_token\_list()

return Response(status=200)

@app.route("/api/common/settings", methods=["POST"])

def api\_get\_settings():

settings = request.json

settings["co2"]["acceptable"] = float(settings["co2"]["acceptable"])

settings["co2"]["harmful"] = float(settings["co2"]["harmful"])

settings["co2"]["danger"] = float(settings["co2"]["danger"])

settings["temperature"]["inf"] = float(settings["temperature"]["inf"])

settings["temperature"]["sup"] = float(settings["temperature"]["sup"])

settings["humidity"]["inf"] = float(settings["humidity"]["inf"])

settings["humidity"]["sup"] = float(settings["humidity"]["sup"])

settings["period"]["forecast"] = int(settings["period"]["forecast"])

settings["period"]["report"] = int(settings["period"]["report"])

if room.co2\_sensor.validate(settings["co2"]["acceptable"]):

room.co2\_requirement\_acceptable = settings["co2"]["acceptable"]

if room.co2\_sensor.validate(settings["co2"]["harmful"]):

room.co2\_requirement\_harmful = settings["co2"]["harmful"]

if room.co2\_sensor.validate(settings["co2"]["danger"]):

room.co2\_requirement\_danger = settings["co2"]["danger"]

if room.temperature\_sensor.validate(settings["temperature"]["inf"]):

room.temperature\_requirement\_inf = settings["temperature"]["inf"]

if room.temperature\_sensor.validate(settings["temperature"]["sup"]):

room.temperature\_requirement\_sup = settings["temperature"]["sup"]

if room.humidity\_sensor.validate(settings["humidity"]["inf"]):

room.humidity\_requirement\_inf = settings["humidity"]["inf"]

if room.humidity\_sensor.validate(settings["humidity"]["sup"]):

room.humidity\_requirement\_sup = settings["humidity"]["sup"]

if settings["period"]["forecast"] > 10 and settings["period"]["forecast"] < 3600:

room.forecast\_period = datetime.timedelta(seconds=settings["period"]["report"])

if settings["period"]["report"] > 10 and settings["period"]["report"] < 3600:

room.report\_period = datetime.timedelta(seconds=settings["period"]["forecast"])

return Response(status=201)

@app.route("/api/get/settings")

def api\_send\_settings():

answer = {

"co2": {"acceptable": room.co2\_requirement\_acceptable,

"harmful": room.co2\_requirement\_harmful,

"danger": room.co2\_requirement\_danger},

"temperature": {"inf": room.temperature\_requirement\_inf,

"sup": room.temperature\_requirement\_sup},

"humidity": {"inf": room.humidity\_requirement\_inf,

"sup": room.humidity\_requirement\_sup},

"period": {"forecast": room.forecast\_period.seconds,

"report": room.report\_period.seconds}

}

return answer

@app.route("/api/get/device\_status")

def api\_get\_devices\_statistic():

answer = {"executor\_on": {},

"executor\_setting": {},

"executor\_address": {},

"executor\_autocontrol": {}, }

report = room.make\_report()

answer["executor\_on"] = report["devices\_status"]

answer["executor\_setting"]["ac"] = room.ac\_device.settings

answer["executor\_setting"]["vent"] = room.vent\_device.settings

answer["executor\_setting"]["heater"] = room.heater\_device.settings

answer["executor\_setting"]["humidifier"] = room.humidifier\_device.settings

answer["executor\_address"]["ac"] = room.ac\_device.address

answer["executor\_address"]["vent"] = room.vent\_device.address

answer["executor\_address"]["heater"] = room.heater\_device.address

answer["executor\_address"]["humidifier"] = room.humidifier\_device.address

answer["executor\_autocontrol"]["ac"] = room.autocontrol\_ac

answer["executor\_autocontrol"]["vent"] = room.autocontrol\_vent

answer["executor\_autocontrol"]["heater"] = room.autocontrol\_heater

answer["executor\_autocontrol"]["humidifier"] = room.autocontrol\_humidifier

return answer

@app.route("/api/get/report", methods=["GET"])

def api\_get\_report():

if request.args.get("period"):

period = datetime.timedelta(seconds=int(request.args.get("period")))

report = room.make\_report(period)

else:

report = room.make\_report()

return report

@app.route("/api/get/data", methods=["GET"])

def api\_get\_data():

if request.args.get("period"):

period = datetime.timedelta(seconds=int(request.args.get("period")))

history = room.get\_history(period)

else:

history = room.get\_history(datetime.timedelta(hours=6))

return history

# UI

@app.route("/")

def main\_page():

return render\_template("index.html")

@app.route("/report")

def report\_page():

report = room.make\_report()

return render\_template("report.html", report=report)

@app.route("/devices")

def devices\_page():

return render\_template("devices.html")

@app.route("/settings")

def settings\_page():

return render\_template("settings.html")

if \_\_name\_\_ == '\_\_main\_\_':

app.debug = True

app.run()