1. Титульный лист
2. 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.

1. Проект системы

Система представляет собой эмулятор умного дома, разработанный для моделирования работы устройств и датчиков в домашней среде. Она предназначена для демонстрации и тестирования функционала умного дома без привязки к реальным устройствам и сенсорам. Основной целью системы является обеспечение удобного взаимодействия пользователя с устройствами и мониторинг различных параметров домашней среды.

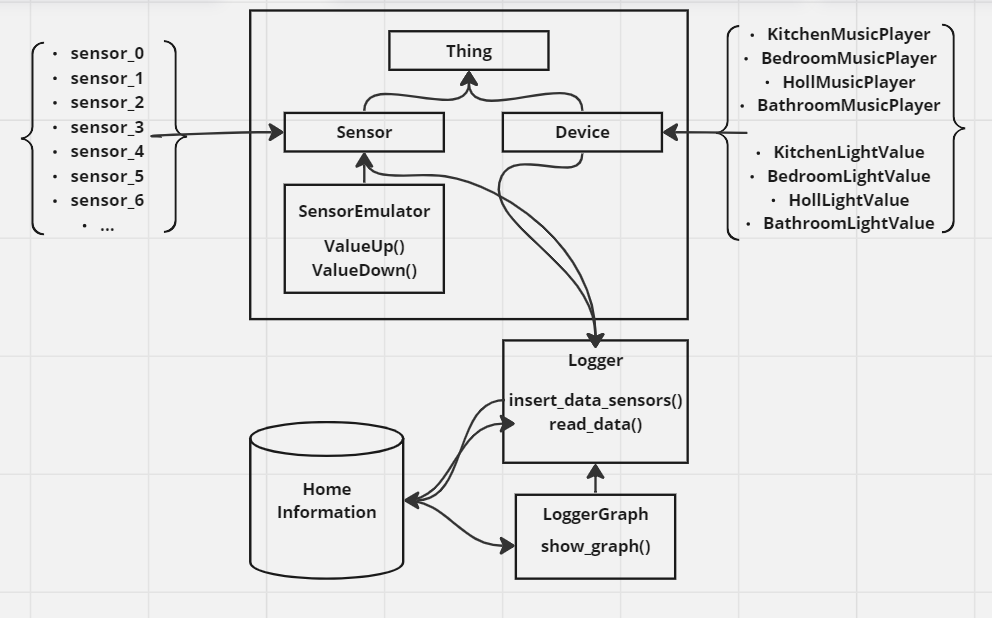
Путем эмуляции работы различных устройств и датчиков пользователь может проводить разнообразные эксперименты и исследования, проверяя взаимодействие различных компонентов системы умного дома. Это позволяет пользователям не только понять принципы работы умного дома, но и протестировать различные сценарии управления устройствами в домашней обстановке.

Интерфейс системы разработан с учетом простоты и удобства использования. Пользователь может легко взаимодействовать с устройствами через веб-интерфейс, задавая им различные значения и отслеживая состояние сенсоров. Графическое отображение данных позволяет наглядно представить изменения параметров домашней среды во времени.

Кроме того, эмулятор умного дома может быть полезным инструментом для обучения и обучающих курсов по теме "Умный дом". Студенты и учащиеся могут использовать эту систему для изучения принципов работы умного дома, а также для разработки и тестирования своих собственных проектов.

Система также предоставляет возможность расширения функциональности путем добавления новых типов устройств и сенсоров, что делает ее гибким инструментом для различных проектов и исследований в области умного дома.

Система включает в себя различные типы датчиков, обеспечивающих мониторинг различных параметров в домашней среде. Среди них можно выделить датчики влажности, температуры и дыма, каждый из которых размещен в четырех ключевых помещениях дома: на кухне, в прихожей, в ванной комнате и в спальной комнате. Каждый датчик обладает уникальным именем, что позволяет идентифицировать его в системе, а также значением, отражающим текущее состояние измеряемого параметра.



Размещение датчиков в различных помещениях обеспечивает полный охват мониторинга ключевых зон в доме, что позволяет эффективно контролировать и поддерживать комфортные условия проживания. Например, датчики влажности в ванной комнате могут помочь вовремя обнаружить утечку воды или предупредить о повышении влажности, что важно для предотвращения образования плесени и грибка.

Каждый датчик осуществляет непрерывное измерение соответствующего параметра и передает полученные данные в систему умного дома. Благодаря этому пользователь может в реальном времени отслеживать изменения в домашней среде и принимать соответствующие меры.

Дополнительно, каждый датчик может иметь симулятор, который позволяет изменять его значения для моделирования различных сценариев. Это полезная функциональность для тестирования реакции системы на различные условия, а также для обучения пользователей в области управления умным домом. Например, с помощью симулятора дымового датчика можно провести тренировочные учения по эвакуации в случае пожара.

Система умного дома является важным элементом современных технологий, предоставляя возможность автоматизации и контроля за домашней средой. Она включает в себя различные типы датчиков, каждый из которых играет ключевую роль в обеспечении безопасности, комфорта и энергоэффективности дома.

Датчики влажности являются важным компонентом системы, особенно в помещениях с повышенным уровнем влажности, таких как ванные комнаты и кухни. Они помогают предотвратить возникновение плесени и грибка, предупреждая пользователей об опасных уровнях влажности и предоставляя возможность принять соответствующие меры.

Датчики температуры играют важную роль в регулировании климата в доме. Они позволяют автоматически управлять отоплением, кондиционированием воздуха и вентиляцией, обеспечивая комфортные условия для проживания. Кроме того, они могут быть интегрированы в систему управления энергопотреблением, помогая снижать расходы на отопление и кондиционирование.

Датчики дыма и угарного газа несут на себе ответственность за обнаружение потенциально опасных ситуаций, таких как пожары или утечки газа. Они могут быстро реагировать на изменения в воздушной среде и автоматически активировать сигналы тревоги или вызов системы пожарной безопасности, что способствует своевременному реагированию и минимизации ущерба.

Каждый датчик в системе обладает уникальным именем, что облегчает идентификацию и управление им. Благодаря симуляторам, встроенным в некоторые датчики, пользователи могут моделировать различные сценарии работы системы, тестировать ее реакцию на различные условия и тренироваться в действиях в чрезвычайных ситуациях.

В системе умного дома присутствуют различные устройства, предназначенные для обеспечения комфорта и удобства жизни пользователей. Среди них можно выделить музыкальные плееры и светильники, которые играют важную роль в создании атмосферы и управлении освещением в доме. Каждое устройство обладает уникальным именем и значениями, которые отражают его текущее состояние или настройки.

Музыкальные плееры позволяют пользователям наслаждаться любимой музыкой в любое время и в любой части дома. Они могут быть интегрированы с другими системами умного дома, такими как система звукоизоляции или автоматизированные занавески, что создает единое пространство для развлечений и отдыха.

Светильники, в свою очередь, обеспечивают освещение в доме и создают нужную атмосферу в различных помещениях. Они могут быть управляемыми по времени, яркости и цвету, что позволяет пользователям создавать уютное освещение в соответствии с их предпочтениями и настроением.

Пользователи могут легко взаимодействовать с устройствами через интерфейс системы, устанавливая им определенные значения или настройки. Например, они могут регулировать громкость музыкального плеера или изменять яркость и цвет светильников с помощью смартфона или планшета, что обеспечивает максимальное удобство и контроль над обстановкой в доме.

Для эффективного управления и анализа данных с датчиков и устройств система использует логгеры. Логгеры предоставляют возможность сохранять значения параметров в определенные моменты времени, что позволяет проводить последующий анализ и визуализацию данных. Благодаря этой функциональности пользователи могут отслеживать изменения в домашней среде, выявлять тенденции и принимать информированные решения для оптимизации управления умным домом.

Веб-интерфейс играет ключевую роль в обеспечении удобства взаимодействия пользователя с системой умного дома. Он представляет собой интуитивно понятный пользовательский интерфейс, доступный через любое устройство с доступом к интернету, такое как компьютер, смартфон или планшет.

Основная функциональность веб-интерфейса включает в себя возможность управления устройствами, просмотра данных с датчиков и анализа информации в виде графиков. Пользователь может легко изменять настройки устройств, устанавливать им определенные значения или включать/выключать функции, не выходя из дома.

Одним из ключевых элементов веб-интерфейса является возможность просмотра данных с датчиков в реальном времени. Пользователь может выбирать конкретные датчики или устройства для отображения их данных на графике, что позволяет наглядно представить изменения параметров домашней среды в течение определенного времени.

Графики предоставляют пользователю ценную информацию о состоянии домашней среды, позволяя выявлять тенденции, анализировать данные и принимать информированные решения. Например, пользователь может отслеживать изменения температуры в разных помещениях дома или мониторить уровень влажности для предотвращения проблем с конденсацией и плесенью.

Дополнительно, веб-интерфейс может предоставлять пользователю возможность управления системой удаленно, что особенно удобно в случае отсутствия дома или нахождения в другом помещении. Например, пользователь может включить свет или настроить температуру в доме перед возвращением домой, что создает дополнительный уровень комфорта и безопасности.

Реализация скриптов управления является важной частью функциональности системы умного дома. Эти скрипты работают автоматически и периодически обновляют данные с датчиков и устройств, обеспечивая постоянную актуализацию информации в базе данных.

Одной из основных задач скриптов является сбор данных с датчиков с определенной периодичностью. Например, они могут запускаться каждые несколько минут или часов для получения новых значений температуры, влажности или других параметров домашней среды. Это позволяет системе оперативно реагировать на изменения и обеспечивать актуальность информации.

Кроме того, скрипты управления отвечают за обновление данных с управляемых устройств. Например, они могут отправлять команды на устройства для включения или выключения света, регулирования температуры или выполнения других задач. Это позволяет системе реагировать на изменения в домашней среде или на команды пользователя, обеспечивая комфорт и безопасность.

Важной функцией скриптов является также поддержание актуальности данных в базе данных. Они обновляют информацию о состоянии датчиков и устройств, а также сохраняют историю изменений для последующего анализа и визуализации. Это позволяет пользователям получать полную и достоверную информацию о домашней среде в различные моменты времени.

Кроме того, скрипты управления могут выполнять другие задачи, такие как оптимизация работы системы, распределение нагрузки или резервное копирование данных. Они обеспечивают стабильную и надежную работу системы умного дома, что является важным условием для обеспечения комфорта и безопасности пользователей.

1. Описание объектной модели

Общая структура системы включает несколько классов, организованных по функциональным аспектам. В основном программа представляет собой эмулятор умного дома с использованием Flask и MongoDB. Программа поделена на отдельные классы, такие как классы датчиков и устройств составляют основу системы.

Абстрактный класс \*\*`Thing`\*\* служит базовым для всех датчиков и устройств, предоставляя метод \*\*`print\_name()`\*\* для вывода имени объекта. Это обеспечивает единообразие в структуре и взаимодействии с различными компонентами системы.

```jsx

class Thing(abc.ABC):

@abc.abstractmethod

def \_\_init\_\_(self, name):

self.name = name

print(f"Create thing {self.name}")

@abc.abstractmethod

def print\_name(self):

print(f'name this device is {self.name}')

```

Класс \*\*`Sensor`\*\* расширяет функционал базового класса, добавляя атрибуты \*\*`name`\*\* и \*\*`value`\*\*, которые отражают имя датчика и текущее значение измеряемого параметра.

```python

class Sensor(Thing):

def \_\_init\_\_(self, name):

super().\_\_init\_\_(name)

self.value = 0

print(f' доабвлен датчик {name} со значением {self.value}')

def print\_name(self):

super().print\_name()

```

В то время как класс \*\*`Device`\*\* представляет собой управляемые устройства, такие как светильники или музыкальные плееры, с аналогичными атрибутами. Организация классов в иерархию позволяет эффективно управлять их поведением и свойствами.

```python

class Device(Thing):

def \_\_init\_\_(self, name, setValue):

super().\_\_init\_\_(name)

self.value = setValue

devices.append(self)

print(f' доабвлено устройство {name} со значением {setValue}')

def print\_name(self):

super().print\_name()

```

Класс \*\*`SensorSimulator`\*\* дополняет функционал датчиков, предоставляя симуляторы для изменения их значений. Это важный инструмент для моделирования различных сценариев и тестирования реакции системы на различные условия. Реализация методов \*\*`valueUp()`\*\* и \*\*`valueDown()`\*\* позволяет динамически изменять значения датчиков, что полезно при обучении и тестировании системы.

```python

class SensorSimulator(Sensor):

def valueUp(self, \*sensors):

for sensor in sensors:

sensor.value += 1

print(f"Change value up of sensor: {self.name}, on {sensor.value}")

def valueDown(self, \*sensors):

for sensor in sensors:

sensor.value -= 1

print(f"Change value down of sensor: {self.name}, on {sensor.value}")

```

Все классы, такие как \*\*`Sensor`\*\*, \*\*`Device`\*\* и \*\*`SensorSimulator`\*\*, инкапсулируют свои данные и функционал, что способствует более четкой и понятной организации кода. Это упрощает его сопровождение, модификацию и масштабирование в случае необходимости. Каждый класс выполняет определенную роль в системе, что способствует ее модульности и гибкости.

Для взаимодействия с базой данных предусмотрены классы логгеров, которые играют важную роль в сохранении и обработке данных системы. Абстрактный класс Logger предоставляет основные методы для работы с базой данных, такие как вставка данных и чтение данных. Этот класс служит основой для всех логгеров в системе, обеспечивая единый интерфейс для работы с различными типами данных.

```python

class Logger:

def \_\_init\_\_(self, db\_name):

self.client = pymongo.MongoClient('mongodb://localhost:27017/')

self.db = self.client[db\_name]

def insert\_data\_sensors(self, nameDB, sensorsArray):

result = {'timeOfRead': datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")}

for sensor in sensorsArray:

result[sensor.name] = sensor.value

return self.db[nameDB].insert\_one(result)

def read\_data(self, nameDB, value={}, field={}):

return self.db[nameDB].find(value, field)

```

Класс LoggerGraph является расширением базового логгера и добавляет функционал для визуализации данных в виде графиков. Это важное дополнение, позволяющее пользователям наглядно анализировать данные и выявлять закономерности или тренды. Графическое отображение данных делает информацию более доступной и понятной, что облегчает принятие решений на основе анализа.

```python

class LoggerGraph(Logger):

def \_\_init\_\_(self, db\_name):

super().\_\_init\_\_(db\_name)

print(f'Инициализация БД: {db\_name}')

def showGraph(self, name\_collection):

device\_name = request.args.get("name")

print(f'Считываем значение с датчика: {device\_name}')

if device\_name is None:

print("Имя датчика не передано в запросе.")

return {}

cursor = self.read\_data(name\_collection)

time = []

device\_values = []

for item in cursor:

if 'timeOfRead' in item and device\_name in item:

time\_str = item['timeOfRead']

time.append(dt.strptime(time\_str, '%Y-%m-%d %H:%M:%S').time())

# device\_values.append(item[device\_name])

value = item[device\_name]

if isinstance(value, (int, float)):

device\_values.append(value)

else:

print(f"Значение '{value}' не является числом, пропускаем.")

else:

print(f"Ключ 'timeOfRead' или датчик '{device\_name}' отсутствует в документе:", item)

return {}

if not device\_values:

print("Нет числовых значений для построения графика.")

return {}

# print(f"до сортировки '{device\_values}' ")

# Выстраиваем числа по порядку от наим. к наиб.

sorted\_data = sorted(zip(time, device\_values), key=lambda x: x[0])

time, device\_values = zip(\*sorted\_data)

# print(f"после сортировки '{device\_values}' ")

print(f"Ключ 'timeOfRead' или датчик '{device\_name}' отсутствует в документе:", item)

x = np.array([t.hour + t.minute / 60 for t in time])

y = np.array(device\_values)

plt.plot(x, y)

plt.xticks(rotation=90)

plt.ylim(0, 50)

plt.xlabel('Время дня')

plt.ylabel('Значение')

current\_time = dt.now().time()

plt.axvline(x=current\_time.hour + current\_time.minute / 60, color='r', linestyle='--', label='Текущее время')

hours = range(0, 25, 1) # интервал каждый час

plt.xticks(hours, [f"{h:02}:00" for h in hours], rotation=90)

plt.legend()

plt.show()

return {}

```

Класс FlaskApp является сердцем веб-приложения, обеспечивая его настройку, конфигурацию и запуск. Он обрабатывает запросы пользователя, взаимодействует с объектами системы и управляет передачей данных между клиентом и сервером. Благодаря FlaskApp пользователи могут удобно взаимодействовать с системой через веб-интерфейс, выполняя различные операции и получая актуальную информацию. Дополнительно, класс Timer играет важную роль в автоматизации процесса записи данных в базу данных. Этот таймер обеспечивает периодическое выполнение операций по сбору и сохранению данных с датчиков, что поддерживает актуальность информации и обеспечивает непрерывную работу системы. Это важное средство для поддержания функциональности и надежности системы в долгосрочной перспективе.

Взаимосвязи между классами играют ключевую роль в обеспечении функциональной целостности системы. Классы датчиков и устройств тесно взаимодействуют с классами логгеров, обеспечивая непрерывную запись данных в базу данных. Это позволяет системе эффективно сохранять информацию о состоянии домашней среды и работе устройств для последующего анализа или использования. При обработке запросов пользователя класс FlaskApp использует методы классов датчиков и устройств для обновления значений и выполнения действий. Это обеспечивает плавное взаимодействие между пользователем и системой, где пользователь может легко управлять устройствами и отслеживать состояние домашней среды через веб-интерфейс. Важным аспектом функциональности системы является возможность взаимодействия между различными датчиками. С помощью симуляторов, встроенных в некоторые датчики, система позволяет моделировать различные сценарии работы и изменять значения параметров. Это особенно полезно для тестирования реакции системы на различные условия и обучения пользователей в области управления умным домом.

1. Описание модуля сбора данных

Система сохраняет данные, полученные с датчиков и устройств. Включенные параметры могут быть следующими: значения датчиков (например, температура, влажность, уровень дыма), настройки и состояния устройств (например, уровень освещенности, воспроизводимый трек на музыкальном плеере).

Эти параметры сохраняются в базе данных системы с использованием логгеров, которые обеспечивают надежное хранение информации для последующего анализа и использования. Благодаря этой функциональности, пользователи могут получать доступ к историческим данным о состоянии домашней среды и работе устройств, что позволяет им более глубоко анализировать процессы и принимать информированные решения. Кроме того, сохранение данных позволяет системе выполнять автоматические задачи, такие как оптимизация работы устройств или предупреждение об опасных ситуациях, что повышает уровень безопасности и комфорта для пользователей.

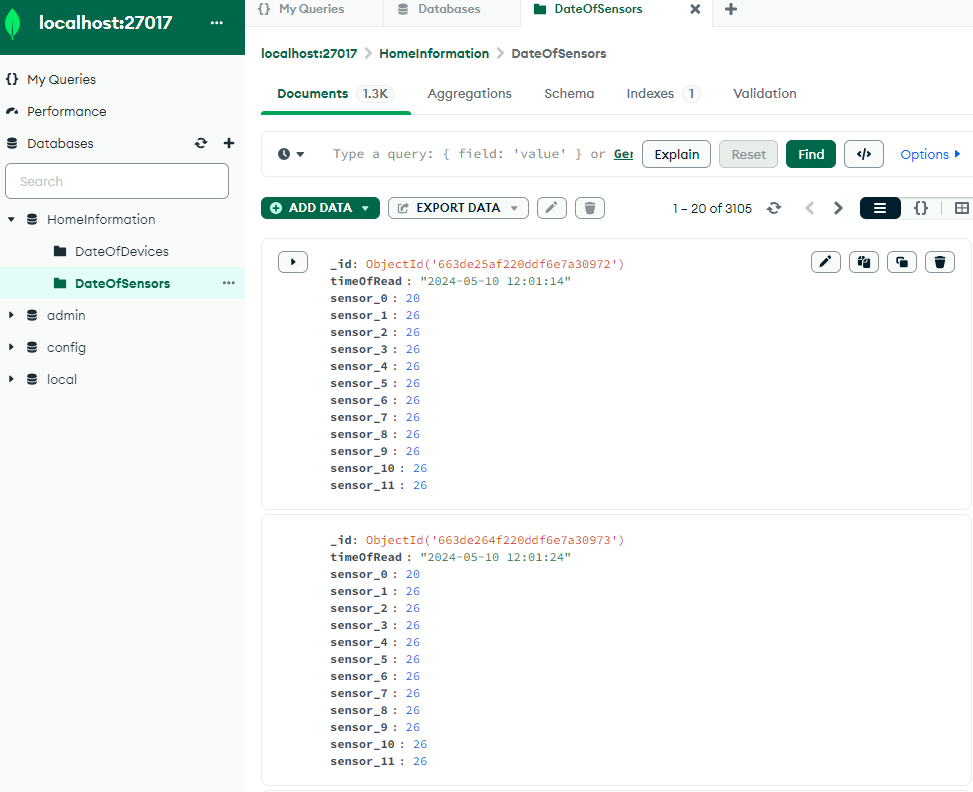
При анализе сохраненных данных пользователи могут обнаруживать различные изменения и отклонения в работе системы умного дома. Например, они могут заметить, что уровень влажности в ванной комнате повышается в определенное время дня, что может свидетельствовать о неисправности оборудования или проблеме с вентиляцией. Такие данные могут быть полезны при планировании обслуживания или ремонта устройств, а также для оптимизации условий проживания.

Важным аспектом хранения данных является их защита и конфиденциальность. Система умного дома должна обеспечивать безопасное хранение информации, чтобы защитить частную жизнь пользователей и предотвратить возможные угрозы безопасности. Для этого данные могут быть зашифрованы и храниться на защищенных серверах с ограниченным доступом, что поможет предотвратить несанкционированный доступ к конфиденциальной информации.

Для хранения данных системы используется база данных MongoDB, которая содержит коллекции для записи информации о датчиках и устройствах. Данные датчиков записываются в коллекцию "DateOfSensors", в то время как данные устройств сохраняются в коллекцию "DateOfDevices". Это позволяет эффективно организовать данные и обеспечить быстрый доступ к нужной информации.

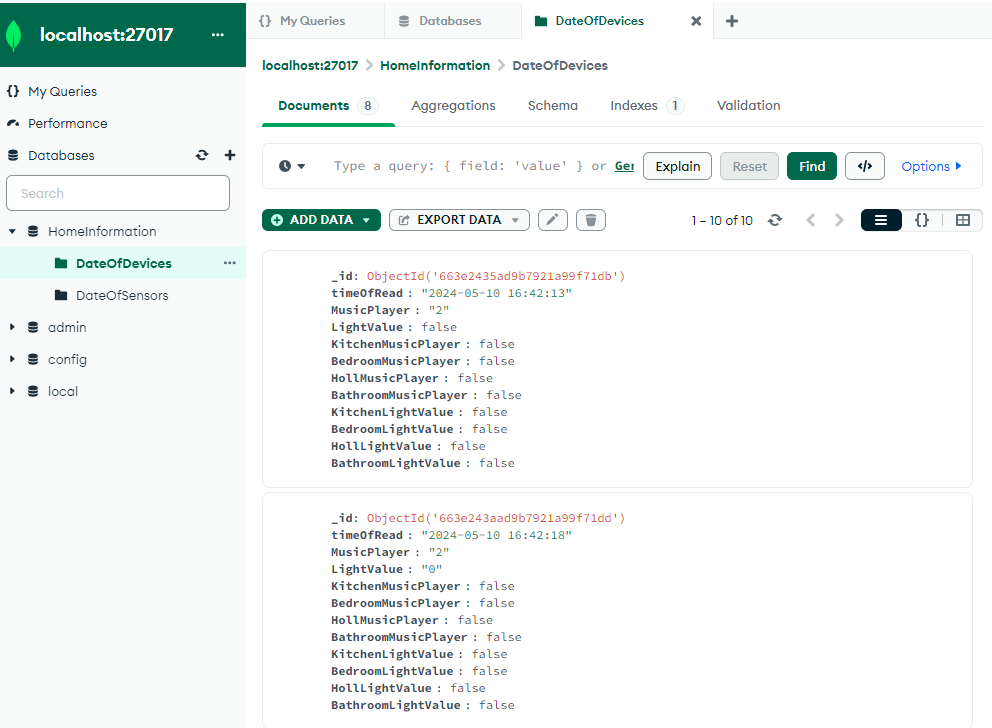
Каждая запись в базе данных содержит уникальный идентификатор (\_id), который является ключевым элементом для идентификации записи. Кроме того, каждая запись включает временную метку (timeOfRead), указывающую на момент времени, когда данные были получены.

Например, для записи данных о датчиках в коллекции "DateOfSensors" используется следующая структура данных:



Каждый датчик обозначается своим уникальным идентификатором (например, sensor\_0, sensor\_1 и т.д.), а его значение записывается в соответствующее поле. Таким образом, данные о всех датчиках за определенный период времени сохраняются в одной записи, облегчая последующий доступ и анализ данных.

Схема хранения данных об устройствах в коллекции "DateOfDevices" аналогична, где каждое устройство представлено своим именем (например, MusicPlayer, LightValue и т.д.) и соответствующим значением. Например:

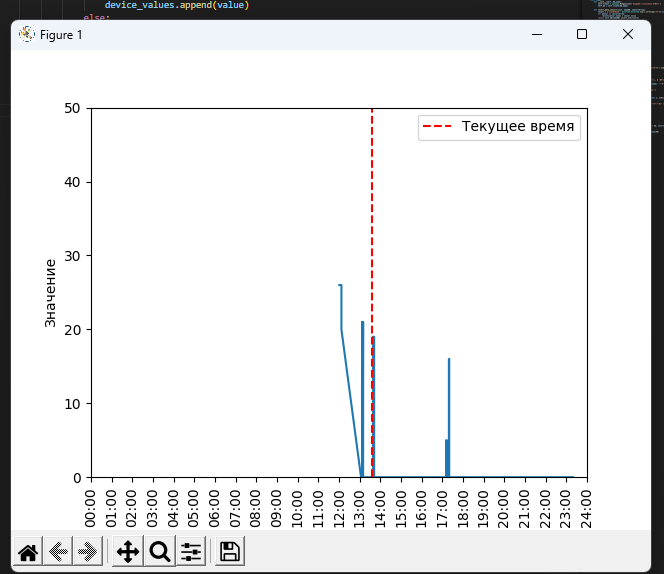


Данные с датчиков и устройств записываются в базу данных с определенной периодичностью, определяемой скриптом управления. В текущей реализации, периодичность записи составляет 10 секунд, но может быть настроена в соответствии с требованиями системы или пользователя.

Для обеспечения корректности данных, система проводит некоторые проверки перед записью в базу данных. В частности, происходит проверка типов данных, чтобы убедиться, что сохраняемые значения соответствуют ожидаемым типам. Также проводится проверка диапазонов значений для числовых параметров, чтобы удостовериться, что они находятся в допустимых пределах. В случае обнаружения ошибок или некорректных данных, система обрабатывает исключения и принимает соответствующие меры, например, пропускает запись некорректных данных.

1. Описание функций анализа сохраненных данных

Функции анализа сохраненных данных в системе умного дома играют ключевую роль в обеспечении безопасности, комфорта и энергоэффективности жилища. Они позволяют не только отслеживать текущее состояние различных параметров домашней среды, но и проводить детальный анализ и прогнозирование изменений в ней. Например, расчет среднего значения температуры или влажности за определенный период времени может дать представление о типичных условиях в доме и помочь выявить аномалии, такие как резкое повышение или понижение температуры, что может свидетельствовать о неполадках в системе отопления или о воздействии внешних факторов, таких как утечка воды или пожар.



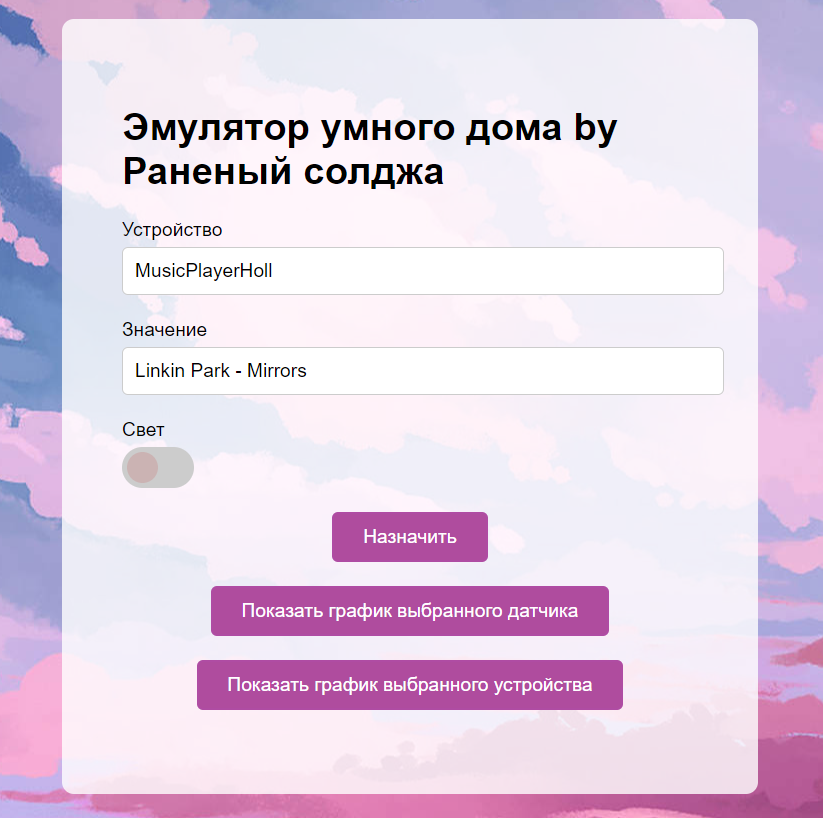
Стандартное отклонение является еще одним важным показателем, который позволяет оценить степень изменчивости данных и выявить возможные аномалии. Например, высокое стандартное отклонение температуры может указывать на нестабильность работы системы отопления или наличие тепловых потерь в доме.

Медиана, минимальное и максимальное значения, а также процентили также имеют практическое значение при анализе данных о домашней среде. Например, минимальное и максимальное значения температуры могут помочь определить диапазон изменений температуры внутри дома, а процентили позволяют выявить экстремальные значения, которые могут потребовать особого внимания.

На основе результатов анализа данных управляющие системы могут принимать автоматические решения для поддержания комфортных условий проживания и оптимизации расхода ресурсов. Например, если анализ показывает, что температура в доме начинает приближаться к критическому уровню, система умного дома может автоматически включить кондиционер для охлаждения воздуха или уведомить владельца об этом.

1. Интерфейсы. Демонстрация разработанных графических интерфейсов пользователя и функций, которые можно использовать с их помощью

Использование двух полей вместо отдельных для каждого устройства делает интерфейс более универсальным и адаптируемым. Пользователю предоставлены два поля - "Name device" и "Value device", которые позволяют указать имя устройства и его значение независимо от его типа. Такой подход делает возможным управление различными устройствами без создания отдельных полей для каждого из них, что сделало интерфейс более гибким и удобным для использования.



Кнопка "Toggle" отвечает за переключение основного освещения в помещении: при ее нажатии происходит изменение состояния освещения, а значение записывается в поле "Value device".

Функционал кнопки "Записать данные" заключается в том, что при ее активации система проходит по массиву всех устройств и ищет устройство с именем, указанным в поле "Name device". Если такое устройство найдено, его значение обновляется в соответствии с указанным значением в поле "Value device".

Кнопки "Показать график устройства" и "Показать график сенсора" предоставляют возможность отображения графика значений конкретного устройства или сенсора по его имени, указанному в поле "Name device". Такой подход позволяет пользователям быстро и удобно анализировать данные, представленные в графическом виде.

1. Выводы

В ходе разработки и анализа работы системы умного дома были выявлены несколько ключевых моментов, определяющих ее функциональность и применимость в повседневной жизни. Система умного дома представляет собой комплексное программное решение, направленное на автоматизацию управления устройствами и мониторинг параметров домашней среды. Структура системы базируется на модели классов, включающих датчики, устройства, логгеры и веб-интерфейс. Это обеспечивает единообразие в организации и управлении компонентами системы.

Важным аспектом функциональности системы является ее гибкость и расширяемость. Использование абстрактных классов и принципов объектно-ориентированного программирования позволяет легко добавлять новые типы устройств и датчиков, а также модифицировать существующий функционал без значительных изменений в коде. Это делает систему адаптивной к различным потребностям пользователей и способствует ее долгосрочной эффективной работе.

Одним из ключевых преимуществ системы является возможность взаимодействия с данными в реальном времени. Пользователи могут мониторить состояние своего дома, получать уведомления о важных событиях и дистанционно управлять устройствами через веб-интерфейс. Это позволяет им поддерживать комфортные условия проживания, обеспечивать безопасность и оптимизировать энергопотребление, даже находясь вдали от дома.

Дополнительно, система предоставляет возможность анализировать сохраненные данные и строить статистические отчеты. Это помогает пользователям выявлять тенденции, определять проблемные ситуации и принимать информированные решения для улучшения качества жизни. Например, анализ изменений температуры или влажности может помочь выявить неисправности в системах отопления и кондиционирования воздуха, а также предотвратить возникновение пожаров и других аварийных ситуаций.

В заключение, система умного дома представляет собой мощный инструмент для автоматизации и контроля домашней среды. Ее гибкость, функциональность и возможность анализа данных делают ее незаменимой помощницей для современных пользователей, стремящихся обеспечить комфорт, безопасность и энергоэффективность своего дома.

Приложения

main.css

/\* Общие стили \*/

body {

  font-family: Arial, sans-serif;

  background-image: url('Sprites/bg.jpg');

  background-size: cover; /\* Растягиваем изображение на весь экран \*/

  background-position: center; /\* Центрируем изображение по центру экрана \*/

  margin: 0;

  padding: 0;

  display: flex;

  justify-content: center; /\* Центрируем контент по горизонтали \*/

  align-items: center; /\* Центрируем контент по вертикали \*/

  min-height: 100vh; /\* Чтобы контент был посередине на всю высоту экрана \*/

}

.container {

  width: 30%; /\* Ширина контейнера на 50% от ширины экрана \*/

  background-color: rgba(255, 255, 255, 0.8); /\* Белый цвет фона с прозрачностью 0.7 \*/

  padding: 50px; /\* Добавим немного отступов для контента внутри контейнера \*/

  border-radius: 10px; /\* Закругленные углы на 10px \*/

}

/\* Стили для контейнера с текстовыми полями \*/

.text-field {

  margin-bottom: 20px;

}

.switch-field {

  margin-bottom: 20px;

}

.send\_button {

  margin-bottom: 20px;

}

.text-field\_\_label {

  display: block;

  font-size: 16px;

  margin-bottom: 5px;

}

.text-field\_\_input {

  width: 100%;

  padding: 10px;

  font-size: 16px;

  border: 1px solid #ccc;

  border-radius: 5px;

}

/\* Стили для переключателя \*/

.switch {

  position: relative;

  display: inline-block;

  width: 60px;

  height: 34px;

}

.switch input {

  opacity: 0;

  width: 0;

  height: 0;

}

.slider {

  position: absolute;

  cursor: pointer;

  top: 0;

  left: 0;

  right: 0;

  bottom: 0;

  background-color: #ccc;

  -webkit-transition: .4s;

  transition: .4s;

  border-radius: 34px;

}

.slider:before {

  position: absolute;

  content: "";

  height: 26px;

  width: 26px;

  left: 4px;

  bottom: 4px;

  background-color: rgb(203, 179, 179);

  -webkit-transition: .4s;

  transition: .4s;

  border-radius: 50%;

}

input:checked + .slider {

  background-color: #cb6d9c;

}

input:focus + .slider {

  box-shadow: 0 0 1px #60f321;

}

input:checked + .slider:before {

  -webkit-transform: translateX(26px);

  -ms-transform: translateX(26px);

  transform: translateX(26px);

}

/\* Стили для кнопки \*/

.custom-btn {

  display: block; /\* Изменяем отображение на блочный элемент \*/

  background-color: #af4c9e;

  border: none;

  color: white;

  text-align: center; /\* Выравниваем текст по центру \*/

  font-size: 16px;

  padding: 10px 24px;

  cursor: pointer;

  border-radius: 5px;

  text-decoration: none;

  -webkit-transition-duration: 0.4s;

  transition-duration: 0.4s;

  overflow: hidden;

  position: relative;

  margin: 0 auto; /\* Центрируем кнопку по горизонтали \*/

}

.custom-btn:after {

  content: "";

  background: #9d3bae;

  display: block;

  position: absolute;

  z-index: -1;

  transition: all 0.4s;

}

.custom-btn:hover:after {

  width: 100%;

  left: 0;

}

/\* Дополнительные стили для кнопки \*/

.btn-3 {

  color: #FFFFFF;

  border: 2px solid #af4ca3;

}

.btn-3 span {

  cursor: pointer;

  display: inline-block;

  position: relative;

  transition: 0.4s;

}

.btn-3 span:after {

  content: '\00bb';

  position: absolute;

  opacity: 0;

  top: 0;

  right: -20px;

  transition: 0.5s;

}

.btn-3:hover span {

  padding-right: 25px;

}

.btn-3:hover span:after {

  opacity: 1;

  right: 0;

}

Script.js

function set\_values(addr) {

    // alert(addr)

    $.ajax({

        type: 'GET', //тип запроса

        url: addr, // адрес, на который отправлен запрос

        dataType: 'json', //тип данных, ожидаемый от сервера

        conectType:'application/json', //тип передаваемых данных

        data:{ //данные запроса

            "value": document.getElementById("value").value,

            "name": document.getElementById("name").value,

            "check": Number(document.getElementById("check").checked)

        },

    });

}

mainUI.html  
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Smart home emulator</title>

    <link rel="stylesheet" type="text/css" href="../static/css/main.css">

    <script src="http://code.jquery.com/jquery-3.3.1.js"></script>

    <script src="../static/js/scripts.js"></script>

</head>

<body>

    <div class="container">

      <h1>Эмулятор умного дома by Раненый солджа</h1>

      <div class="text-field">

        <label class="text-field\_\_label" for="name">Устройство</label>

        <input class="text-field\_\_input" type="text" id="name" placeholder='К примеру "MusicPlayer"'>

      </div>

      <div class="text-field">

        <label class="text-field\_\_label" for="name">Значение</label>

        <input class="text-field\_\_input" type="text" id="value" placeholder='К примеру "Linkin Park - Mirror"'>

      </div>

      <div class="switch-field">

        <label class="text-field\_\_label">Свет</label>

        <label class="switch">

            <input type="checkbox" id="check">

            <span class="slider round"></span>

        </label>

      </div>

      <div class="send\_button">

        <button class="custom-btn btn-3" onclick='set\_values("/SetValues")'><span>Назначить</span></button>

      </div>

      <div class="send\_button">

        <button class="custom-btn btn-3" onclick='set\_values("/GraphSensors")'><span>Показать график выбранного датчика</span></button>

      </div>

      <div class="send\_button">

        <button class="custom-btn btn-3" onclick='set\_values("/GraphDevices")'><span>Показать график выбранного устройства</span></button>

      </div>

    </div>

</body>

</html>

App.py

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

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from threading import Timer

import things

import re

from array import \*

from things import sensors, devices

from logger import LoggerGraph as lg

app = Flask(\_\_name\_\_)

array\_name\_sensors = []

for count in range(12):

    array\_name\_sensors.append("sensor\_"+ str(count))

    sensors.append(things.Sensor(array\_name\_sensors[count]))

sensorsWet = (sensors[0], sensors[1], sensors[2], sensors[3])

sensorsTemp = (sensors[4], sensors[5], sensors[6], sensors[7])

sensorsSmoke = (sensors[8])

MusicPlayer = things.Device('MusicPlayer', 0)

LightValue = things.Device('LightValue', False)

KitchenMusicPlayer = things.Device('KitchenMusicPlayer', False)

BedroomMusicPlayer = things.Device('BedroomMusicPlayer', False)

HollMusicPlayer = things.Device('HollMusicPlayer', False)

BathroomMusicPlayer = things.Device('BathroomMusicPlayer', False)

KitchenLightValue = things.Device('KitchenLightValue', False)

BedroomLightValue = things.Device('BedroomLightValue', False)

HollLightValue = things.Device('HollLightValue', False)

BathroomLightValue = things.Device('BathroomLightValue', False)

logger\_instance = lg('HomeInformation')

def log\_sensorsData():

    logger\_instance.insert\_data\_sensors('DateOfSensors', sensors)

    Timer(10, log\_sensorsData).start()

log\_sensorsData()

@app.route('/')

def connect\_interface():

    return render\_template('mainUI.html')

@app.route('/SetValues')

def set\_values():

    print(f'Название: { request.args.get("name")}, Значение: {request.args.get("value")}')

    for device in devices:

        if(device.name == request.args.get("name")):

            value = request.args.get("value")

            if value.isdigit():

                device.value = int(value)

            else:

                device.value = request.args.get("value")

            logger\_instance.insert\_data\_sensors('DateOfDevices', devices)

    LightValue.value = request.args.get("check")

    return json.dumps({'Название:': request.args.get('name'), 'Значение:': request.args.get('value')})

@app.route('/GraphSensors')

def read\_data\_sensors():

    logger\_instance.showGraph('DateOfSensors')

    print("GraphSensors")

@app.route('/GraphDevices')

def read\_data\_devices():

    logger\_instance.showGraph('DateOfDevices')

    print("GraphDevices")

@app.route('/up')

def upSensorValue():

    things.SensorSimulator.valueUp(\*sensors)

    result = {}

    for sensor in sensors:

        result[sensor.name] = sensor.value

    return result

@app.route('/down')

def downSensorValue():

    things.SensorSimulator.valueDown(\*sensors)

    result = {}

    for sensor in sensors:

        result[sensor.name] = sensor.value

    return result

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

    app.run()

logger.py

import pymongo

import datetime

import numpy as np

import matplotlib.pyplot as plt

from flask import request

from datetime import datetime as dt

import pymongo

import datetime

import numpy as np

import matplotlib.pyplot as plt

from datetime import datetime as dt

from flask import request

class Logger:

    def \_\_init\_\_(self, db\_name):

        self.client = pymongo.MongoClient('mongodb://localhost:27017/')

        self.db = self.client[db\_name]

    def insert\_data\_sensors(self, nameDB, sensorsArray):

        result = {'timeOfRead': datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")}

        for sensor in sensorsArray:

            result[sensor.name] = sensor.value

        return self.db[nameDB].insert\_one(result)

    def read\_data(self, nameDB, value={}, field={}):

        return self.db[nameDB].find(value, field)

class LoggerGraph(Logger):

    def \_\_init\_\_(self, db\_name):

        super().\_\_init\_\_(db\_name)

        print(f'Инициализация БД: {db\_name}')

    def showGraph(self, name\_collection):

        device\_name = request.args.get("name")

        print(f'Считываем значение с датчика: {device\_name}')

        if device\_name is None:

            print("Имя датчика не передано в запросе.")

            return {}

        cursor = self.read\_data(name\_collection)

        time = []

        device\_values = []

        for item in cursor:

            if 'timeOfRead' in item and device\_name in item:

                time\_str = item['timeOfRead']

                time.append(dt.strptime(time\_str, '%Y-%m-%d %H:%M:%S').time())

                # device\_values.append(item[device\_name])

                value = item[device\_name]

                if isinstance(value, (int, float)):

                    device\_values.append(value)

                else:

                    print(f"Значение '{value}' не является числом, пропускаем.")

            else:

                print(f"Ключ 'timeOfRead' или датчик '{device\_name}' отсутствует в документе:", item)

                return {}

        if not device\_values:

            print("Нет числовых значений для построения графика.")

            return {}

        # print(f"до сортировки '{device\_values}' ")

        # Выстраиваем числа по порядку от наим. к наиб.

        sorted\_data = sorted(zip(time, device\_values), key=lambda x: x[0])

        time, device\_values = zip(\*sorted\_data)

        # print(f"после сортировки '{device\_values}' ")

        print(f"Ключ 'timeOfRead' или датчик '{device\_name}' отсутствует в документе:", item)

        x = np.array([t.hour + t.minute / 60 for t in time])

        y = np.array(device\_values)

        plt.plot(x, y)

        plt.xticks(rotation=90)

        plt.ylim(0, 50)

        plt.xlabel('Время дня')

        plt.ylabel('Значение')

        current\_time = dt.now().time()

        plt.axvline(x=current\_time.hour + current\_time.minute / 60, color='r', linestyle='--', label='Текущее время')

        hours = range(0, 25, 1)  # интервал каждый час

        plt.xticks(hours, [f"{h:02}:00" for h in hours], rotation=90)

        plt.legend()

        plt.show()

        return {}

logger.py

import abc

sensors = []

devices = []

class Thing(abc.ABC):

    @abc.abstractmethod

    def \_\_init\_\_(self, name):

        self.name = name

        print(f"Create thing {self.name}")

    @abc.abstractmethod

    def print\_name(self):

        print(f'name this device is {self.name}')

class Sensor(Thing):

    def \_\_init\_\_(self, name):

        super().\_\_init\_\_(name)

        self.value = 0

        print(f' доабвлен датчик {name} со значением {self.value}')

    def print\_name(self):

        super().print\_name()

class Device(Thing):

    def \_\_init\_\_(self, name, setValue):

        super().\_\_init\_\_(name)

        self.value = setValue

        devices.append(self)

        print(f' доабвлено устройство {name} со значением {setValue}')

    def print\_name(self):

        super().print\_name()

class SensorSimulator(Sensor):

    def valueUp(self, \*sensors):

        for sensor in sensors:

            sensor.value += 1

            print(f"Change value up of sensor: {self.name}, on {sensor.value}")

    def valueDown(self, \*sensors):

        for sensor in sensors:

            sensor.value -= 1

            print(f"Change value down of sensor: {self.name}, on {sensor.value}")