

Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Расчетно-пояснительная записка к курсовой работе
по курсу «Проектирование автоматизированных систем»
на тему:
«Система домашней автоматизации»

Руководитель работы:

Доцент кафедры ИТАС
Ломако А.В.

Выполнил:

Студент группы 820601
Шведов А.Р.

Минск 2021

РЕФЕРАТ

СИСТЕМА ДОМАШНЕЙ АВТОМАТИЗАЦИИ «УМНЫЙ ДОМ»: курсовой проект / А.Р Шведов – Минск: БГУИР, 2021, – п.з. – 46 с., чертежей (плакатов) – 1 л. формата А1.

Курсовой проект на тему «Система домашней автоматизации» разработан с целью создания системы домашней автоматизации или так называемой системы «Умный дом». Система позволит пользователю автоматизировать базовые домашние задачи и использовать *IOT* устройства без привязки к определенной компании или вендору.

Пояснительная записка к курсовому проекту состоит из введения, 4 разделов, включающих литературный обзор по теме курсового проекта, сравнительный анализ существующих аналогов системы, описание процессов проектирования и программной реализации, а также технико-экономического обоснования разработки и внедрения системы, заключение, список использованных источников и приложения.

Для разработки автоматизированной информационной системы был выбран язык программирования *Python* и *Raspberry Pi* в качестве физического сервера системы. Система представляет собой физический хаб, принимающий сигналы с передовых устройств и датчиков, а также веб-приложение доступ, к которому, пользователь сможет получать посредством веб-браузера. Она предназначена для использования исключительно в локальной сети, однако при желании может быть переведена на доступ из глобальной сети.

Результаты, полученные в ходе курсового проектирования, могут использоваться индивидуально в любой локации. Ограничениями данной модели будут являться лишь наличие устройств, датчиков и других физических компонентов системы. Созданная модель является также хорошим примером создания UML диаграмм и может служить примером написания моделей других проектов. Автоматизация базовых домашних задачи позволит повысить качество жизни, в некоторых случаях снизить затраты (например, пользователь может понять по графику температуры, что в одной из комнат окно теряет тепло), а также помочь автоматизировать часть работы, например установку и настройку светового оборудования при работе с видео.

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет информационных технологий и управления

УТВЕРЖДАЮ
Заведующий кафедрой

(подпись)

2021г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Шведову Андрею Робертовичу

1. Тема проекта Система домашней автоматизации
2. Срок сдачи студентом законченного проекта 24.12.2021 г.
3. Исходные данные к проекту Разработать систему домашней автоматизации, позволяющую пользователю использовать умные (IOT) устройства, иметь возможность получать последнюю информацию с установленных датчиков установленных в помещении, быстро оценивать общее состояние системы. Система представляет из себя объединенные в едином умные устройства и датчики, ПО для управления и центральную панель управления. Система не должна передавать информацию в глобальную сеть.
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

ВВЕДЕНИЕ

1 СИСТЕМНО-АНАЛИТИЧЕСКАЯ ЧАСТЬ

2 ПРОЕКТНО-РАСЧЕТНАЯ ЧАСТЬ

3 РЕАЛИЗАЦИОННАЯ ЧАСТЬ

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ

5. Список графических материалов (с указанием точного назначения
необходимых рисунков и графиков)

Диаграмма вариантов использования – 1 лист формата А2.

Диаграмма классов – 1 лист формата А2.

6. Консультант по проекту (с обозначением разделов проекта) А.В. Ломако

7. Дата выдачи задания 14.09 2021 г.

8. Календарный график работы над проектом на весь период проектирования
(с обозначением сроков выполнения и трудоемкости отдельных этапов):

17.10. 2021 – 30% 1-я опроцентовка

14.11. 2021 – 60% 2-я опроцентовка

13.12. 2021 – 90% 3-я опроцентовка

16.12.2021 – 100% защита работы

Руководитель _____ А.В. Ломако
(подпись)

Задание принял к исполнению _____ А.Р. Шведов
(подпись)

СОДЕРЖАНИЕ

Введение	6
1 Системно-аналитическая часть	7
1.1 Описание и анализ объекта автоматизации	7
1.2 Постановка задачи	8
1.3 Обзор и анализ существующих аналогов	9
1.4 Методологии и инструментальные средства проектирования	12
2 Проектно-расчетная часть	15
2.1 Концептуальная модель системы	15
2.2 Информационное обеспечение	19
2.3 Модельное представление процессов	24
3 Реализационная часть	28
3.1 Инструментальное обеспечение	28
3.2 Программное обеспечение	30
3.3 Организационное обеспечение	38
Заключение	44
Список использованных источников	45

ВВЕДЕНИЕ

Автоматизация в наше время охватывает практически все сферы человеческой деятельности, от поддержания микроклимата в помещении, до управления технологическими процессами в серьезных отраслях промышленности. Потребность в автоматизации возникла еще в глубокой древности и постоянно возрастала в процессе развития человеческих возможностей и средств производства.

Домашняя автоматизация в современных условиях – чрезвычайно гибкая система, которую пользователь конструирует и настраивает самостоятельно в зависимости от собственных потребностей. Это предполагает, что каждый владелец «умного дома» самостоятельно определяет, какие устройства куда установить и какие задачи они будут исполнять.

Наиболее распространенные примеры автоматических действий в «умном доме»: автоматическое включение и выключение света, автоматическая коррекция работы отопительной системы или кондиционера, генерирование отчетов и сводок на текущий день и автоматическое уведомление о вторжении, возгорании или протечке воды.

Цель данного курсового проекта – создание системы домашней автоматизации или так называемой системы «Умный дом». Система позволит пользователю агрегировать информацию с *IOT* устройств и датчиков и управлять ими, а также работать с веб приложениями, у которых есть свой *API*.

Курсовой проект выполнен с учётом требований к содержанию и оформлению курсового проекта, а также в соответствии с методическими указаниями.

Пояснительная записка является документацией к разработанному приложению и содержит следующую информацию:

В первом разделе происходит обзор состояния вопроса. Рассматриваются основные технологии и доступные решения.

Во втором разделе дана постановка задачи, описаны этапы проектирования приложения. Приведены основные графические схемы и диаграммы.

В третьем разделе описана реализация системы и приведены примеры использования.

1 СИСТЕМНО-АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Описание и анализ объекта автоматизации

Умный дом» или же «домашняя автоматизация» также можно встретить «интеллектуальное здание» – все это термины обозначающие современные жилые и офисные здания, квартиры, дома с единой автоматизированной системой управления и мониторинга сетей и систем, таких как:

- освещение;
- микроклимат;
- безопасность;
- коммуникационные системы.

Умным домом, в современном понятии этого слова, можно назвать дом, предоставляющий удобное управление всеми системами входящими в него, а также возможность их интеграции друг с другом, что повышает функциональность каждой из них и обеспечивает согласованную работу.

К ряду основных задач домашней автоматизации можно отнести:

- 1 Повышение комфорта проживания.
- 2 Повышение уровня безопасности. Система дает знать о возможных взломах или возникающих рисках.
- 3 Экономия ресурсов за счет рационального расхода и распределения энергии.
- 4 Возможность интеграции в систему диспетчерского контроля и управления. Такого рода функционал популярен среди административных и офисных зданий, но в некоторых случаях также используется в частных домах и квартирах.

На основе вышесказанного можно сделать вывод, что умный дом со временем станет важным элементом в вопросах удобства и жизнедеятельности большинства людей. Однако такой дом является высокотехнологичным изобретением и должен иметь хорошо спланированное построение системы.

На сегодняшний день можно выделить три варианта построения системы умного дома:

- централизованные системы;
- децентрализованные системы;
- смешанные.

Централизованные системы построены на основе центрального контроллера, который, по сути, является единым мощным процессором. Включая в себя исполнительные блоки с добавлением различных панелей

управления. Такие системы распространены в основном в частном секторе. Все устройства подключаются к контроллеру, который является центром системы. Отдельные компоненты в системе могут иметь свои микроконтроллеры, но осуществляет управление и отвечает за взаимодействие всей системы центральный контроллер, который может принимать и посылать сигналы управления по различным каналам. Главным преимуществом таких систем является не высокая стоимость всей системы, а недостаток в большой зависимости большинства устройств от центрального блока, который может выйти из строя.

Децентрализованные системы отличаются от централизованных отсутствием центрального контроллера. Все устройства соединяются непосредственно друг с другом при помощи шины и оснащены своим автономным контроллером. Недостаток таких систем в относительной сложности и дороговизне каждого отдельного устройства, но окупается это надежностью, а также меньшим количеством или практически полным отсутствием проводных сетей.

Смешанные системы включают в себя элементы и принципы обеих вышеперечисленных систем. Как пример смешанной системы, можно привести *X10*. *X10* – широко используемый в области домашней автоматизации стандарт определяющий протокол и метод передачи управляющих сигналов по беспроводным каналам или обычной электропроводке на электронные модули, к которым подключены управляемые бытовые устройства.

1.2 Постановка задачи

Целью курсового проекта является создание скелета системы домашней автоматизации «Умный дом». В качестве базовых задач система должна позволять пользователю подключать умные устройства и сенсоры. Пользователь должен иметь возможность автоматизации использования, добавления определенных триггеров и условий выполнения. Так же система должна предоставлять пользователю простой и удобный интерфейс взаимодействия, который может быть доступен из локальной сети.

Система должна представлять из себя центральный сервер (*Raspberry PI*) с *ZigBee* радио-адаптером, виртуальный сервер для взаимодействия с пользователем, систему контроля и управления девайсами, работающую на центральном сервере.

Система должна предусмотреть будущее расширение функционала за счет добавления новых методов и подключения различных классов устройств.

1.3 Обзор и анализ существующих аналогов

Программы домашней автоматизации довольно распространены на сегодняшний день. Каждый вендор и компания пытается оставить пользователя в своей экосистеме. Обычно это требует того, чтобы пользователь приобретал и использовал продукты только определённой компании, так называемые проприетарные технологии.

1.3.1 *Xiaomi Smart Home Suite*. Вся работа с системой осуществляется через уже знакомое нам фирменное мобильное приложение *Smart Home*. Никаких вариантов доступа через браузер или программы для ПК не предусмотрено. Кроме того, обязательно иметь аккаунт *Xiaomi*, к которому привязываются все устройства (создать его можно и из мобильной утилиты). Отметим, что именно эта программа используется и для обновления встроенного в устройства программного обеспечения.

К плюсам относятся:

- простота использования;
- мобильное приложение;
- довольно быстрая интеграция с другими устройствами *Xiaomi*.

К минусам относятся:

- закрытый исходный код;
- поддержка устройств только от одного вендора;
- ограниченный функционал.

Пример работы с приложением представлен на рисунке 1.

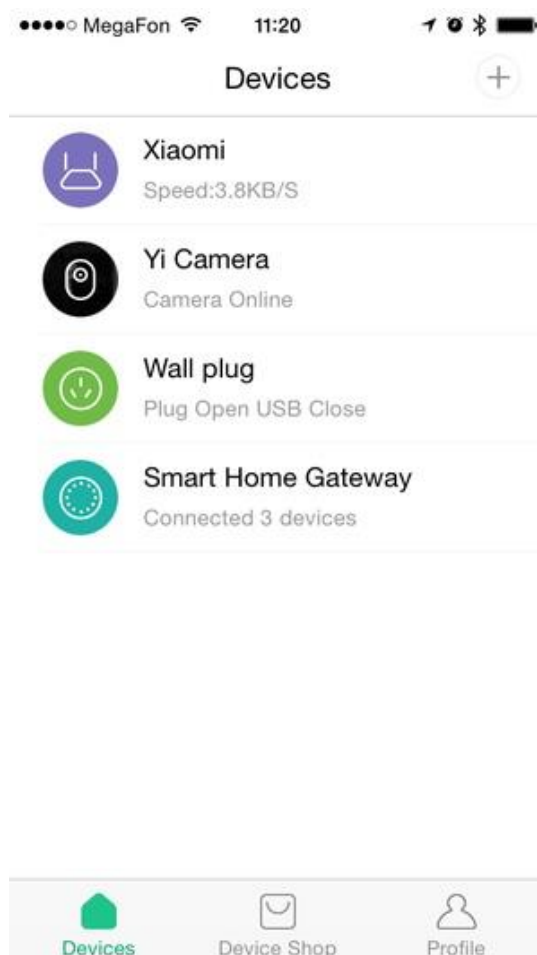


Рисунок 1 – *Xiaomi Smart Home*

1.3.2 Apple Home Kit. Программное обеспечение корпорации *Apple*, которое позволяет использовать устройства на *IOS* (*iPhone*, *iPad*, *Apple TV*, *Apple Watch*) для настройки взаимодействия с интеллектуальными бытовыми приборами. Иными словами, это программная платформа, которая синхронизирует и объединяет в сеть совместимые устройства и позволяет ими управлять в том числе с помощью голосового помощника *Siri*.

К плюсам относятся:

- простота использования;
- мобильное приложение, интегрированное в экосистему;
- облачная синхронизация.

К минусам относятся:

- закрытый исходный код;
- поддержка только устройств, сертифицированных самим *Apple*;
- облачная синхронизация;
- цена устройств.

Пример работы с приложением представлен на рисунке 2.

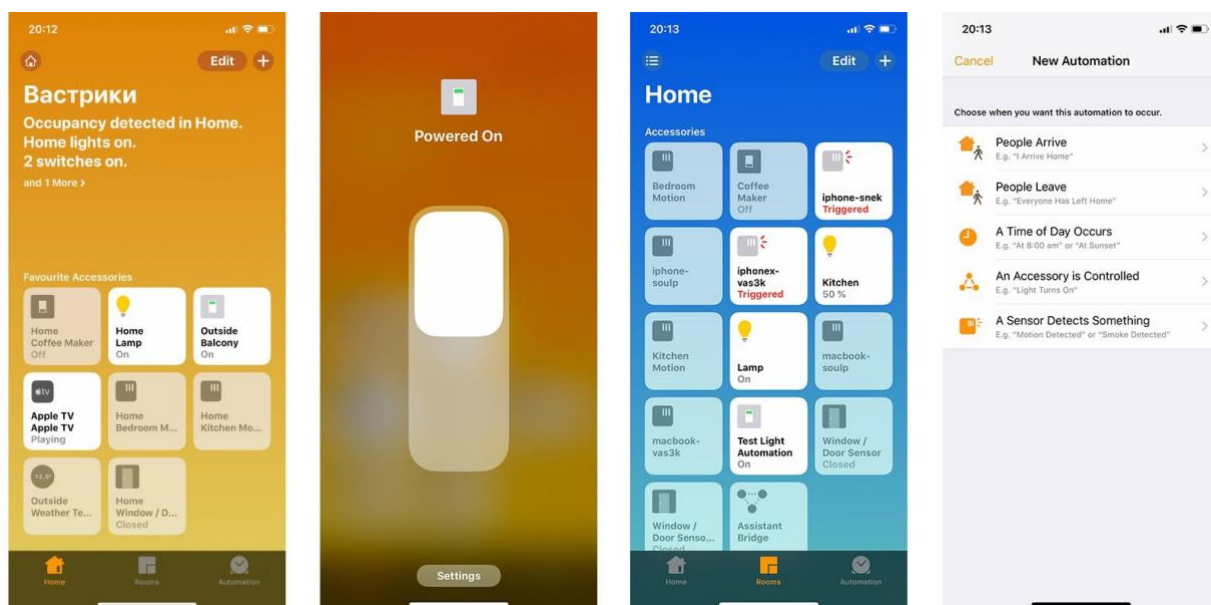


Рисунок 2 – Apple Home Kit

1.3.3 Samsung SmartThings. Программное обеспечение корпорации *Samsung*, которое позволяет использовать устройства на *Samsung* для настройки взаимодействия с интеллектуальными бытовыми приборами. Иными словами, это программная платформа, которая синхронизирует и объединяет в сеть совместимые устройства и позволяет ими управлять.

К плюсам относятся:

- простота установки;
- мобильное приложение, интегрированное в экосистему;
- облачная синхронизация.

К минусам относятся:

- закрытый исходный код;
- поддержка устройств только от одного вендора;
- облачная синхронизация;
- ограниченный функционал.

Пример работы с приложением представлен на рисунке 3.

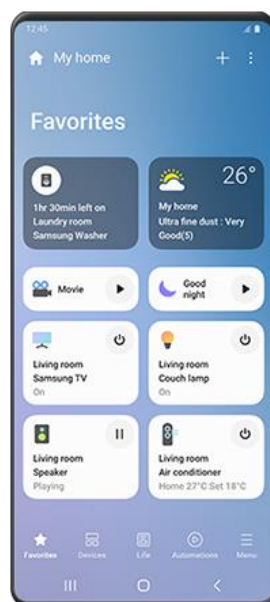


Рисунок 3 – *Samsung SmartThings*

1.4 Методологии и инструментальные средства проектирования

1.4.1 Методология *IDEF0*. Согласно методологии *IDEF0*, модель объекта управления строится в виде диаграмм, состоящих из блоков и стрелок. Блоки (*Activities*) обозначают работы (функции), выполняемые на объекте управления, а стрелки (*Arrows*) – материальные объекты и информацию, обрабатываемые в ходе выполнения работ или используемые для их выполнения. Строится набор диаграмм, последовательно детализирующих процессы функционирования объекта управления.

Общий вид блока диаграммы, построенной согласно методологии *IDEF0* (*IDEF0*-диаграммы, или *IDEF0*-модели), представлен на рисунке 4.

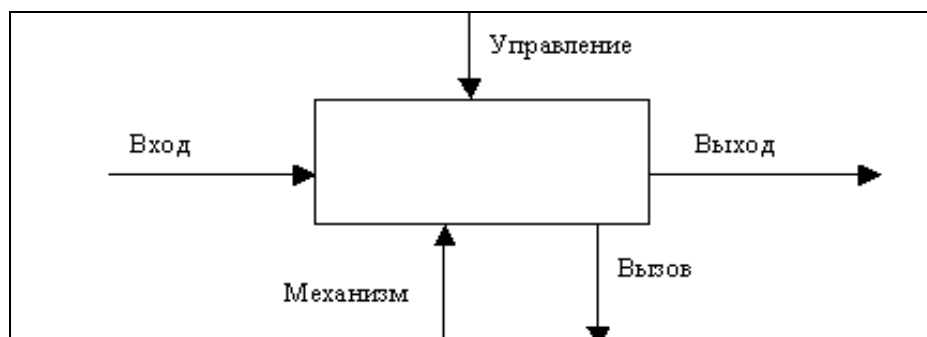


Рисунок 4 – Общий вид блока диаграммы *IDEF0*

Смысл стрелок, используемых на *IDEF0*-диаграмме, следующий.

Вход – материальные объекты (например, сырье) или информация, обрабатываемые в процессе выполнения работы для получения результата (выхода).

Управление – правила выполнения работы (методы, стандарты и т.д.).

Механизм – ресурсы для выполнения работы (персонал, станки, оборудование и т.д.).

Выход – результат выполнения работы (готовая продукция, результаты анализа информации и т.д.).

Вызов – указатель на другую модель.

1.4.2 Методология *DFD*. *DFD* (англ. *Data flow diagram* – диаграмма потоков данных) – методология графического структурного анализа, описывающая внешние по отношению к системе источники, и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ.

DFD-диаграммы включают следующие элементы:

- Работы (Activities). Обычно они обозначают операции по обработке данных.
- Стрелки (Arrows), обозначающие перемещение данных или объектов в процессе их обработки.
- Внешние ссылки (External references) – входы и выходы модели. Используются для обозначения источников данных, обрабатываемых в анализируемой системе, или приемников результатов обработки данных.
- Хранилища данных (Data stores). Используются для обозначения баз данных, массивов, картотек (при ручной обработке данных) и т.д.

Таким образом диаграммы потоков данных показывают, как каждый процесс преобразует свои входные данные в выходные, и выявляют отношения между этими процессами. *DFD* представляет моделируемую систему как сеть связанных работ. Стрелки в *DFD* показывают, как объекты (включая и данные) реально перемещаются от одного действия к другому. Диаграммы потоков данных моделируют систему как набор действий, соединенных друг с другом стрелками.

1.4.3 Методология *IDEF3* – способ описания процесса с использованием структурированного метода, позволяет представить положение вещей в предметной области как упорядоченную последовательность событий с одновременным описанием объектов, имеющих непосредственное отношение

к процессу. *IDEF3* также может быть использован как метод проектирования бизнес-процессов. *IDEF3*-моделирование органично дополняет традиционное моделирование с использованием стандарта методологии *IDEF0*.

IDEF3-диаграммы включают следующие элементы:

- единицы работы (*UOW – Units of work*), аналогичные работам на *IDEF0*-диаграммах;
- стрелки (*Arrows*), обозначающие порядок выполнения работ;
- перекрестки (*Junctions*), отображающие логические связи между работами.

1.4.4 UML (англ. *Unified Modeling Language* – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой *UML*-моделью. *UML* был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. *UML* не является языком программирования, но на основании *UML*-моделей возможна генерация кода.

2 ПРОЕКТНО-РАСЧЕТНАЯ ЧАСТЬ

2.1 Концептуальная модель системы

Проектируемая система разрабатывается для пользователя умного дома, который хочет на каждый момент времени видеть статус системы, агрегированную информацию с датчиков, при этом не передавая эти данные в интернет.

2.1.1 Построение модели начинается с описания функционирования системы в целом в виде **контекстной диаграммы**. [1] Исходя из перечня входных и выходных данных, а также основываясь на требованиях к системе, была разработана контекстная диаграмма по методологии *IDEF0*. Контекстная диаграмма представлена на рисунке 5.

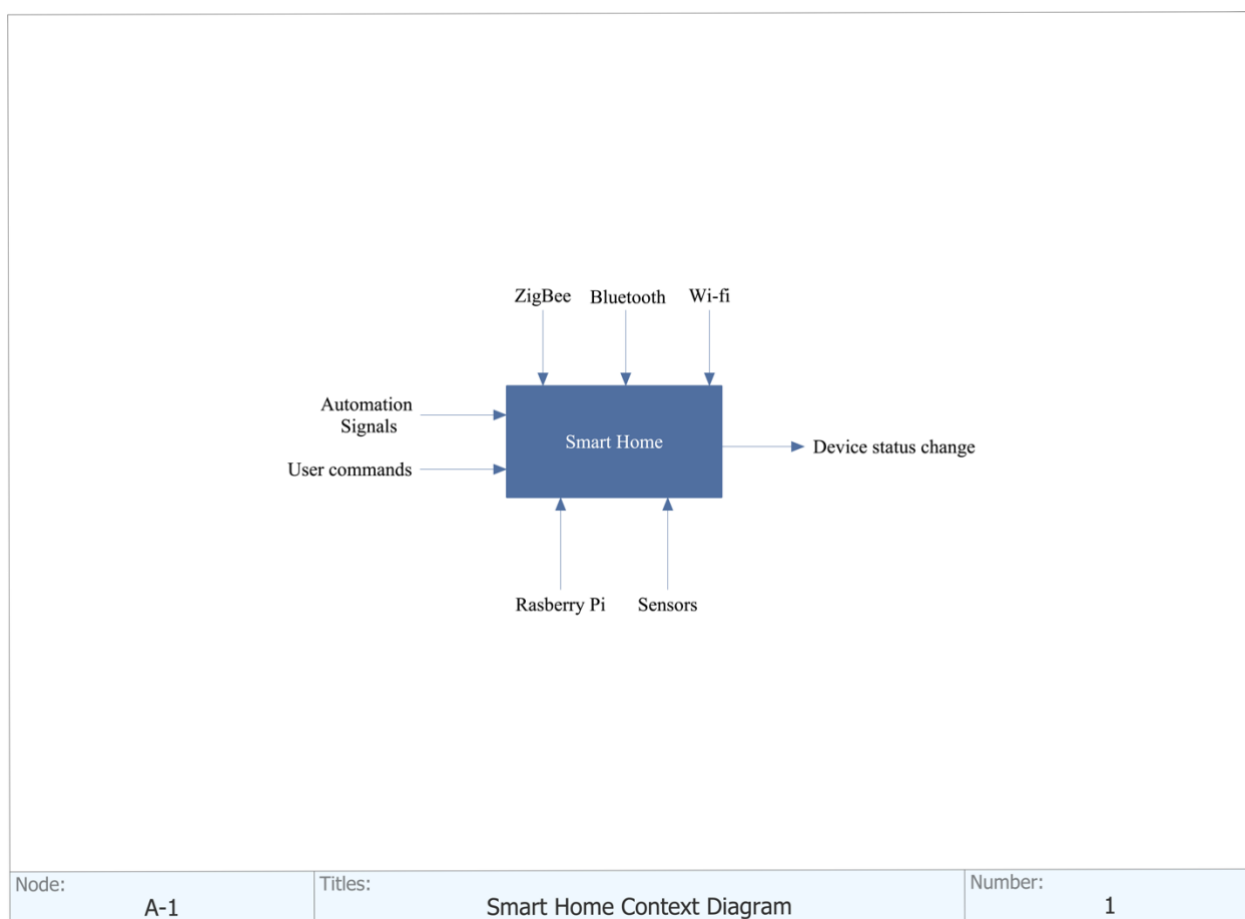


Рисунок 5 – Контекстная диаграмма

Контекстная диаграмма представляет собой общее описание работы системы и ее взаимодействия с внешними элементами или данными. На приведённой контекстной диаграмме представлены виды входной и выходной информации системы, а также механизмы и элементы управления. На вход системы поступают информационные данные, полученные от сенсоров и умных устройств;

В качестве выходной информации служат следующие данные:

- графики обработанных данных;
- уведомления.

В качестве механизмов служат датчики и устройства, а так же *Raspberry Pi*, являющаяся центральной точки обработки информации.

Элементы управления представлены следующими составляющими:

- *Wi-Fi* – стандарт беспроводного подключения *LAN* для коммуникации разных устройств, относящийся к набору стандартов *IEEE 802.11*. *Wi-Fi* использует для коммуникации устройств в малом масштабе;

- *Bluetooth* – технология беспроводной передачи данных. *Bluetooth* обеспечивает обмен информацией между такими устройствами как персональные компьютеры (настольные, карманные, ноутбуки), мобильные телефоны, принтеры, цифровые фотоаппараты, мышки, клавиатуры, джойстики, наушники, гарнитуры на надёжной, недорогой, повсеместно доступной радиочастоте для ближней связи.

- *ZigBee* – открытый глобальный стандарт беспроводной технологии, разработанный для использования цифровых радиосигналов с низким энергопотреблением для персональных сетей. Это популярный протокол связи, используемый в системе «Умный дом». Является языком, необходимым смарт-устройствам для «разговора» друг с другом.

2.1.2 Для более подробного описания и детализации рассматриваемого процесса используется диаграмма декомпозиции. Декомпозиция позволяет постепенно и структурировано представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко усваиваемой.

Контекстная диаграмма была разбита на три последовательных функциональных блока:

- «*Device action*» – функциональный блок, отвечающий за отправку команд у установку связи с периферийными устройствами;

– «*Command processing*» – функциональный блок, отвечающий за обработку и анализ информации, собранной с устройств. Он так же отвечает за передачу сигналов управления этим устройствам;

– «Пользовательский интерфейс» – функциональный блок, отвечающий за интерфейс взаимодействия, просмотр графиков и данных, получение информации о текущем состоянии системы.

Диаграмма декомпозиции контекстной диаграммы представлена на рисунке 6.

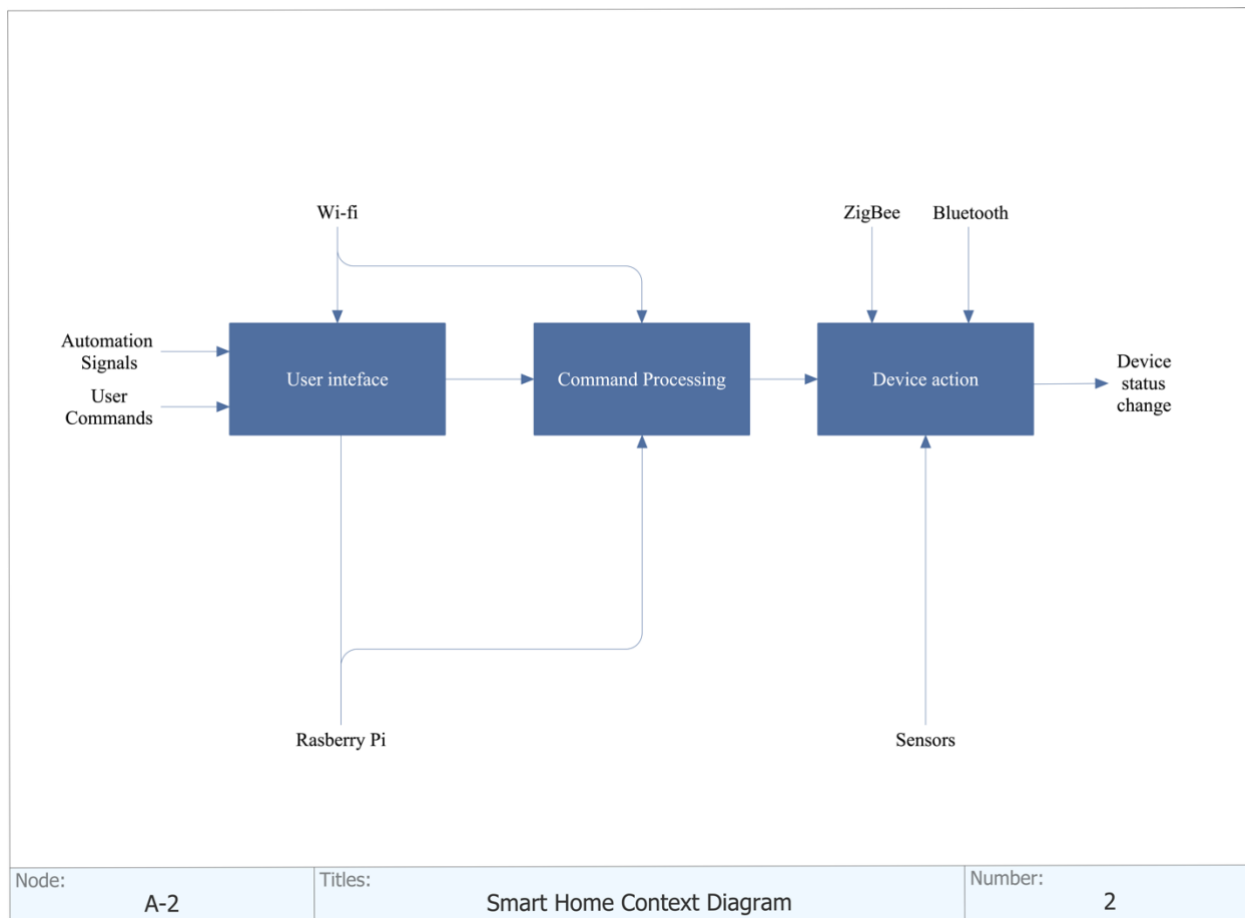


Рисунок 6 – Диаграмма декомпозиции

2.1.3 Для подробного описания процесса «*User Interface*» была использована методология *DFD*. Разработанная *DFD* диаграмма представлена на рисунке 7.

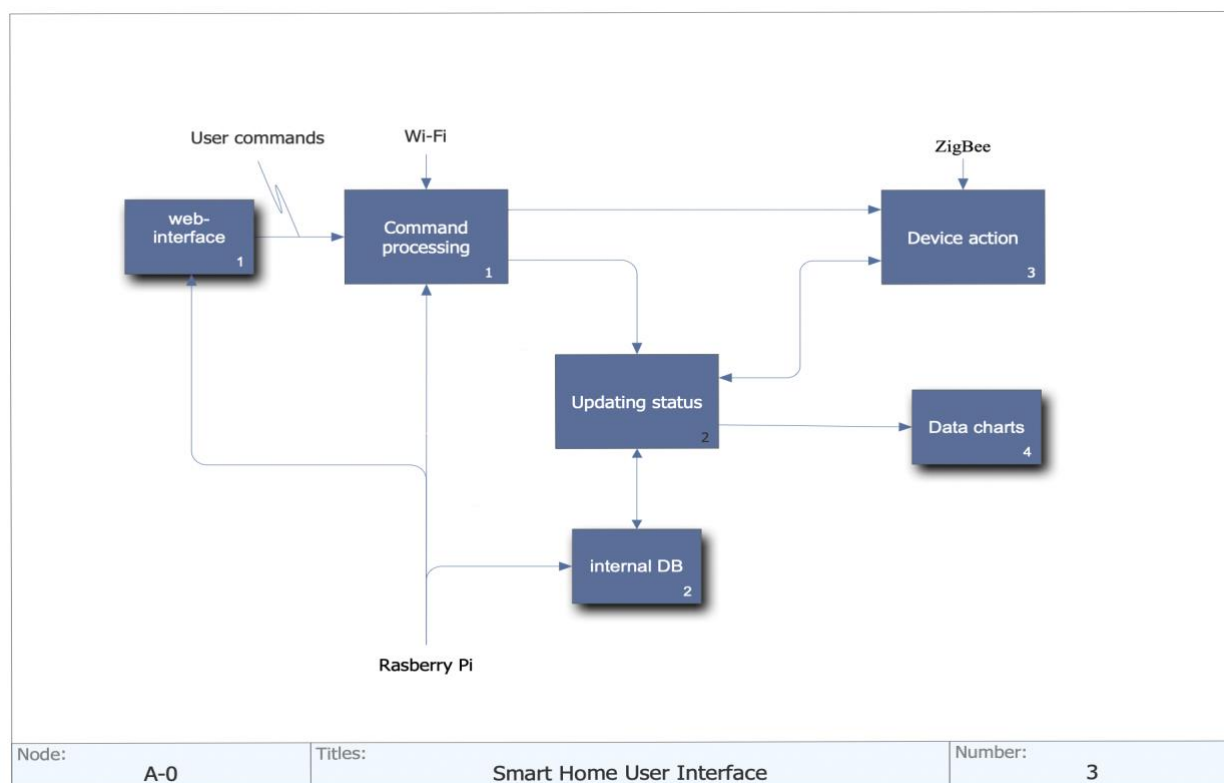


Рисунок 7 – DFD диаграмма для процесса «User Interface»

На данной диаграмме представлены следующие работы:

- «Обработка команды»;
- «Обновление статуса»;
- «Управление устройством».

Также на диаграмме представлен внешний объект «Веб-интерфейс».

Кроме этого, на диаграмме представлены следующие хранилища данных:

- «Внутренняя база данных»;
- «Диаграммы данных».

В начале процесса происходит получение и обработка команды пользователя, полученная посредством *HTTP* запроса на локальный сервер через *web*-интерфейс.

После этого сервер обновит и передаст команду на само устройство через протокол *ZigBee*. Одновременно с этим обновится статус в веб-интерфейсе и локальной БД.

Одновременно с этим в веб-интерфейсе обновится диаграмма данных (например, включив умный датчик температуры, система начнет получать информацию с него).

2.1.4 Для описания работы «Обработка запроса пользователя» с использованием структурированного метода, позволяющего представить положение вещей в предметной области как упорядоченную последовательность событий с одновременным описанием объектов, имеющих непосредственное отношение к процессу, была использована методология *IDEF3*.

Разработанная *IDEF3* диаграмма представлена на рисунке 8. Диаграмма приводит описание процессов, происходящих в работе «Обработка запроса пользователя». На данной диаграмме приведено семь процессов: «Получение запроса», «Обработка запроса», «Проверка соединения», «Сигнал устройству», «Положительный ответ», «Отрицательный ответ», «Запись в БД».

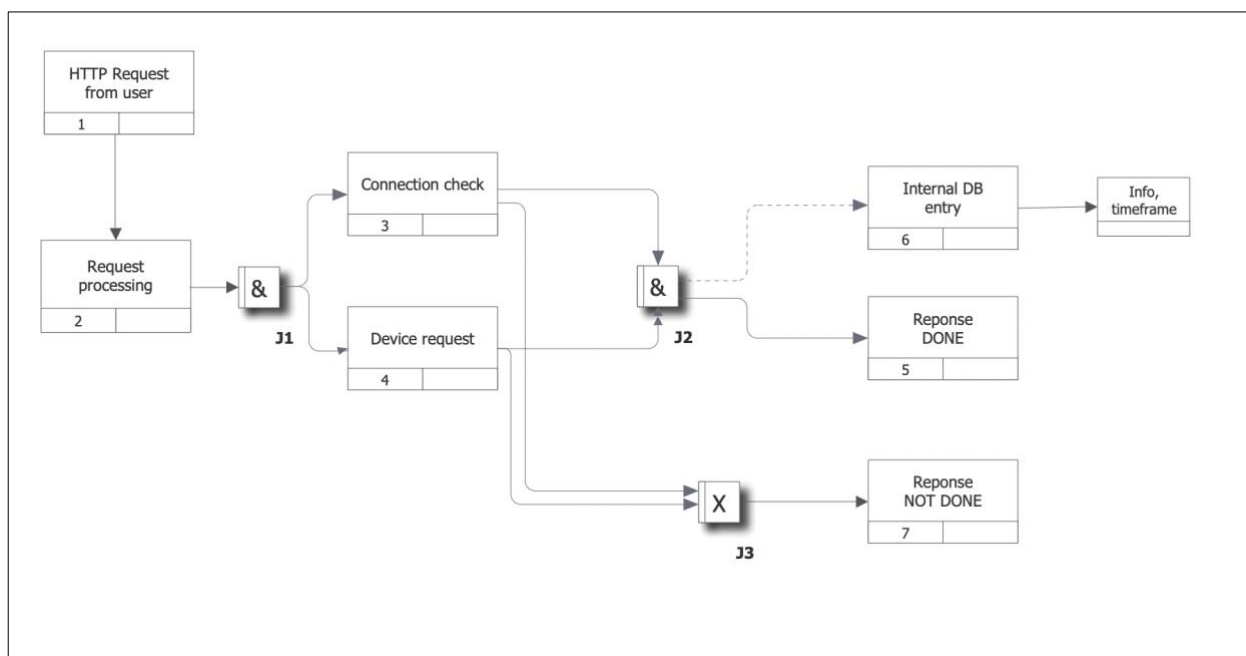


Рисунок 8 – Диаграмма *IDEF3* для работы «Обработка запроса пользователя»

2.2 Информационное обеспечение

2.2.1 Проектирование структуры локальной базы данных. Проектирование баз данных – процесс создания схемы базы данных и определения необходимых ограничений целостности.

Основной этап процесс проектирования базы данных информационной системы – преобразование требований к данным в структуры данных. На

выходе получаем СУБД-ориентированную структуру базы данных и спецификации прикладных программ. На этом этапе часто моделируют базы данных применительно к различным СУБД и проводят сравнительный анализ моделей.

Для реляционных БД (т.е. БД, где данные представлены в виде таблиц) логическая модель включает:

- описание таблиц;
- описание связей между таблицами;
- описание атрибутов.

При реализации автоматизированной системы необходимо хранение большого объема информации, в связи с чем возникает необходимость в использовании базы данных, которую изначально необходимо спроектировать. Основная цель проектирования базы данных – это сокращение избыточности хранимых данных, и впоследствии, экономия объема используемой памяти, устранение возможности возникновения противоречий из-за хранения в разных местах информации об одном и том же объекте.

Основные задачи проектирования баз данных:

- обеспечение хранения в БД всей необходимой информации;
- обеспечение возможности получения данных по всем необходимым запросам;
- сокращение избыточности и дублирования данных;
- обеспечение целостности данных (правильности их содержания): исключение противоречий в содержании данных, исключение потерь и т.д.

Схема структуры локальной базы данных разрабатываемой системы представлена на рисунке 9.

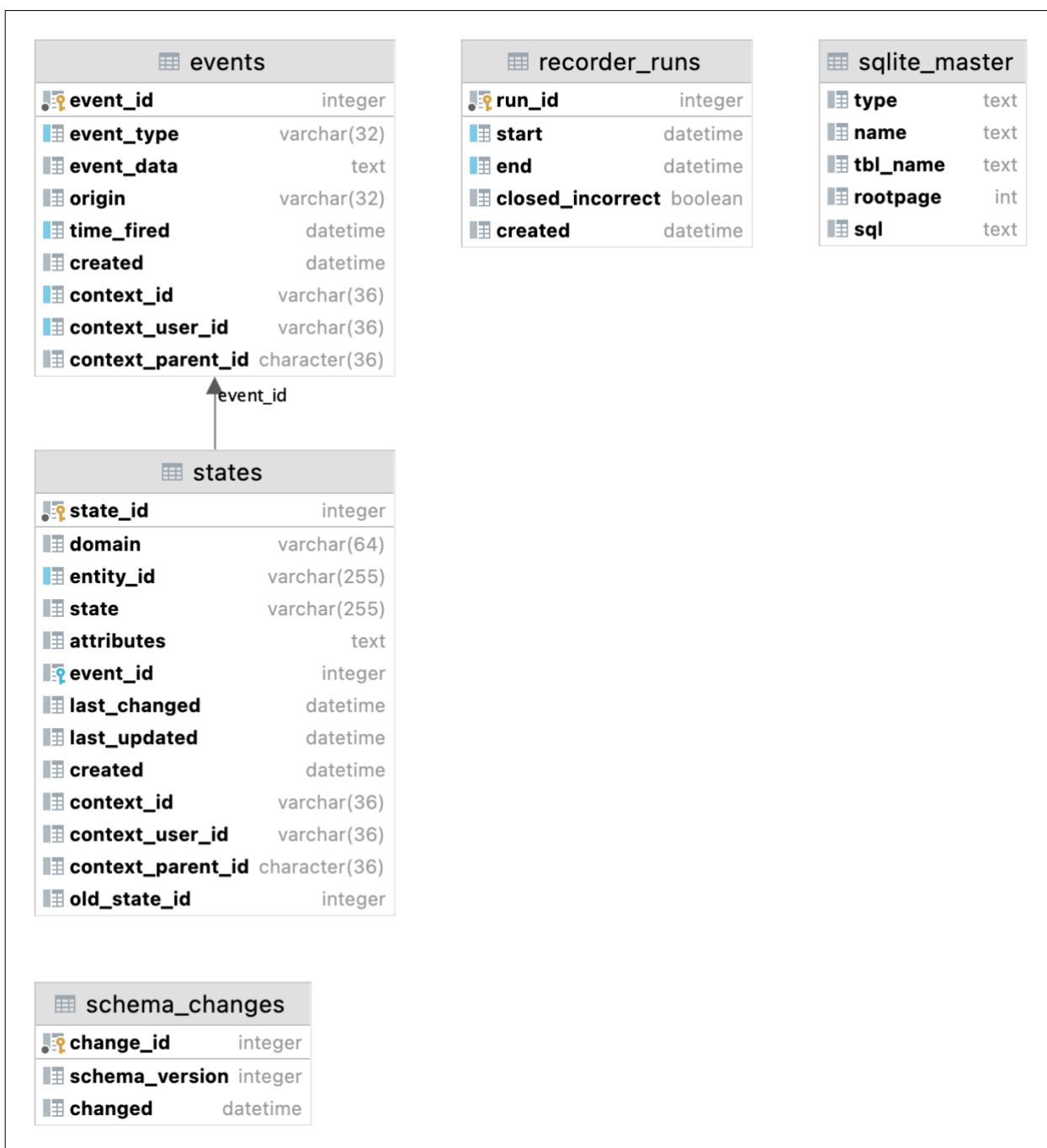


Рисунок 9 – Структура локальной базы данных

В структуре базы данных есть четыре различных сущности (*sqlite_master* на рисунке 9 является вспомогательной таблицей и к системе не относится). Структура сущности «*events*» представлена в таблице 1.

Таблица 1 – Структура сущности «events»

Название	Тип	Описание
<i>event_id</i>	<i>integer</i>	Уникальный идентификатор, первичный ключ
<i>event_type</i>	<i>varchar</i>	Вид события
<i>event_data</i>	<i>text</i>	Дополнительная информация о событии
<i>origin</i>	<i>varchar</i>	Информация о том, что вызвало событие
<i>time_fired</i>	<i>datetime</i>	Время события
<i>context_id</i>	<i>varchar</i>	Уникальный идентификатор контекста события.
<i>context_user_id</i>	<i>varchar</i>	Уникальный идентификатор пользователя события.
<i>context_parent_id</i>	<i>varchar</i>	Уникальный идентификатор вызвавшего событие.

Структура сущности «states» представлена в таблице 2.

Таблица 2 – Структура сущности «states»

Название	Тип	Описание
<i>state_id</i>	<i>integer</i>	Уникальный идентификатор, первичный ключ.
<i>old_state_id</i>	<i>integer</i>	Уникальный идентификатор предыдущего статуса.
<i>created</i>	<i>datetime</i>	Дата.
<i>domain</i>	<i>varchar</i>	«домен» или область, к которой относится статус
<i>entity_id</i>	<i>varchar</i>	Идентификатор устройства
<i>attribute</i>	<i>text</i>	Дополнительная информация об устройстве.
<i>state</i>	<i>varchar</i>	Описание статуса (<i>online/disabled</i>).
<i>last_changed</i>	<i>datetime</i>	Дата последнего изменения.
<i>last_updated</i>	<i>datetime</i>	Дата последнего обновления статуса.

Продолжение таблицы 2

<i>context_id</i>	<i>varchar</i>	Уникальный идентификатор контекста события.
<i>context_user_id</i>	<i>varchar</i>	Уникальный идентификатор пользователя события.
<i>context_parent_id</i>	<i>varchar</i>	Уникальный идентификатор вызвавшего событие.

Структура сущности «*schema_changes*» представлена в таблице 3.

Таблица 3 – Структура сущности «*schema_changes*»

Название	Тип	Описание
<i>change_id</i>	<i>varchar</i>	Уникальный идентификатор, первичный ключ.
<i>schema_version</i>	<i>int</i>	Номер схемы.
<i>changed</i>	<i>datetime</i>	Дата изменения.

Структура сущности «*recorder_runs*» представлена в таблице 4.

Таблица 4 – Структура сущности «*recorder_runs*»

Название	Тип	Описание
<i>run_id</i>	<i>varchar</i>	Уникальный идентификатор, первичный ключ.
<i>“start”</i>	<i>datetime</i>	Дата начала записи.
<i>“end”</i>	<i>datetime</i>	Дата окончания записи.
<i>closed_incorrect</i>	<i>bool</i>	Статус закрытия записи.
<i>created</i>	<i>datetime</i>	Дата.

2.2.2 Для описания взаимодействия элементов системы была создана **диаграмма классов**. Диаграмма классов (*Class Diagram*) определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами.

В разрабатываемой системе выделим 5 основных классов:

- «*System*» – базовый класс, являющийся родительским для всех других. Является основной точкой агрегации всех действий и событий;

- «*Sensor*» – класс, представляющий функционал взаимодействия с сенсорами и различными датчиками;
- «*HouseHolds*» – класс, с содержащий информацию об основных домашних задачах пользователя, имеющий функционал для создания новых уведомлений, задач и автоматизаций;
- «*HomeTheatre*» – класс, предоставляющий функционал для управления домашним кинотеатром и являющимся родительским для таких классов как «Телевизор», «Колонки»;
- «*Home Security*» – класс, предоставляющий доступ к функционалу безопасности дома. Реализует методы уведомления и мониторинга.

Созданная диаграмма представлена на рисунке 10.

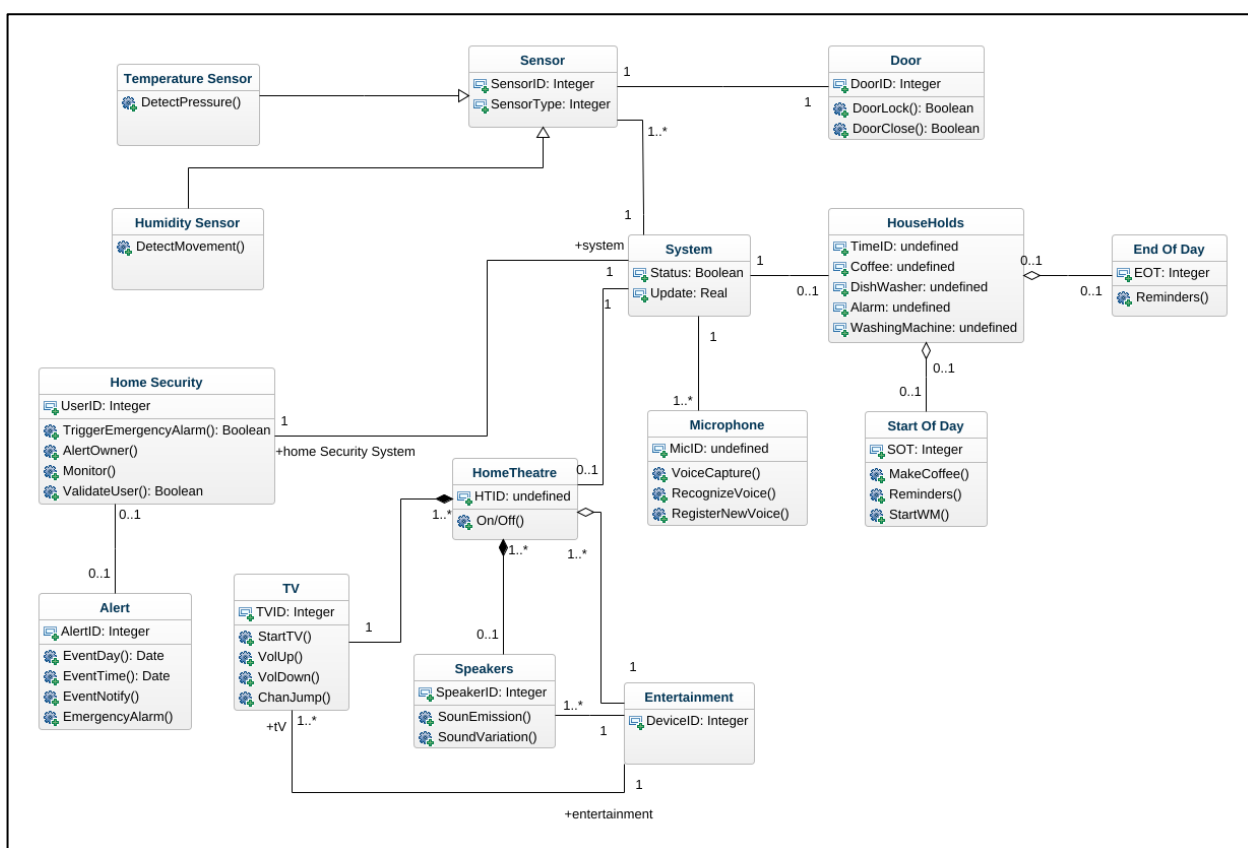


Рисунок 10 – Диаграмма классов

2.3 Модельное представление процессов

2.3.1 Диаграмма вариантов использования (Use case diagram) – диаграмма, на которой отражены отношения, существующие между действующими лицами и вариантами использования. [2]

Основная задача диаграммы – представлять собой единое средство, дающее возможность заказчику, конечному пользователю и разработчику совместно обсуждать функциональность и поведение системы.

В рамках данного проекта рассматривается взаимодействие действующего лица «Пользователь» со системой. Разработанная диаграмма вариантов использования представлена на рисунке 11.

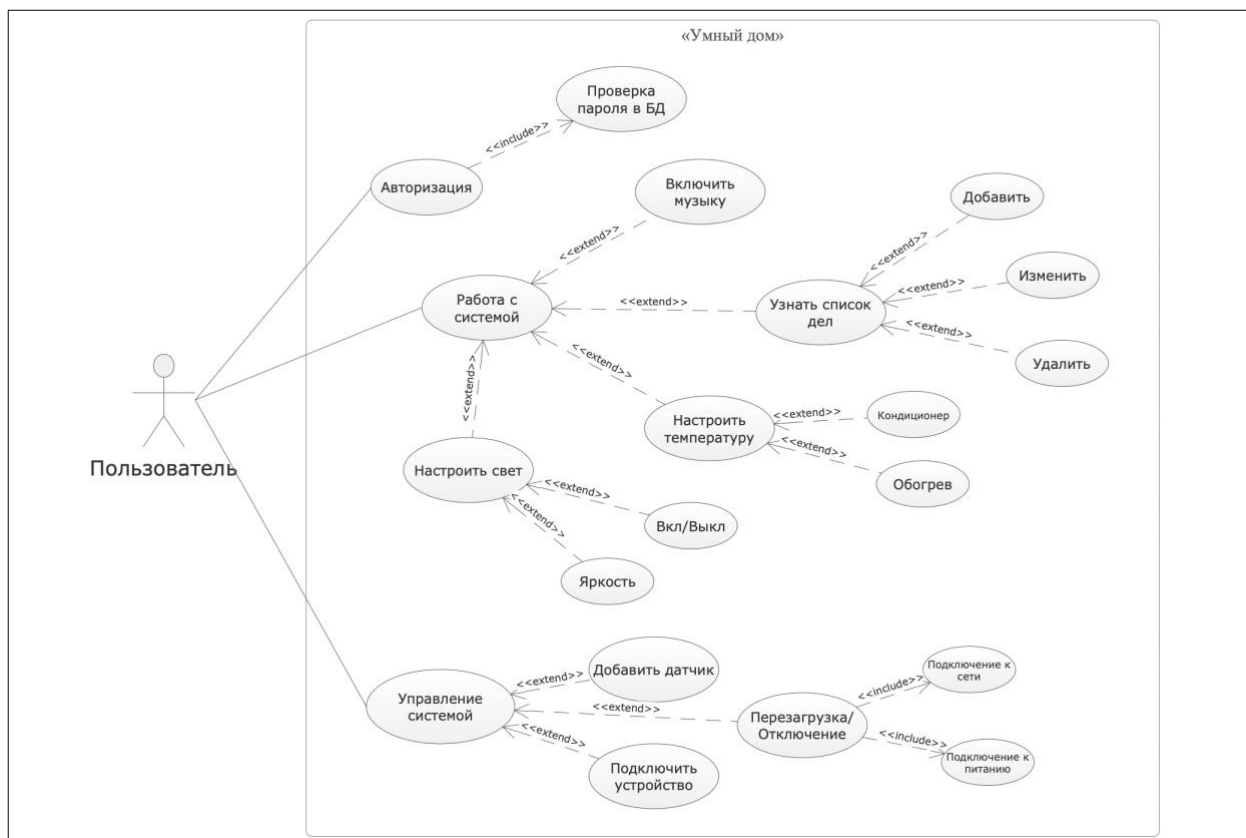


Рисунок 11 – Диаграмма вариантов использования

Система имеет следующие варианты использования:

- «Авторизация». Данный вариант использования отвечает за получения доступа к системе. Его расширяет прецедент «Проверка пароля». Т.к. система не рассчитана на нескольких пользователей, она не нуждается в возможности добавления новых пользователей;

- «Управление системой». Данный вариант использования включает в себя добавление новых устройств, датчиков, а также физическое отключение или перезагрузка системы;

- «Работа с системой». Данный вариант использования подразумевает пользовательский интерфейс, посредством которого пользователь получает информацию о состоянии системы, данных с датчиков. Он расширяется

прецедентами «Настроить свет», «Узнать список дел», «Настроить температуру» и «Включить музыку».

2.3.2 Диаграммы последовательности описывают поведение взаимодействующих групп объектов. Как правило, диаграмма последовательности охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Вариант использования «Включить лампочку» представлен на рисунке 12.

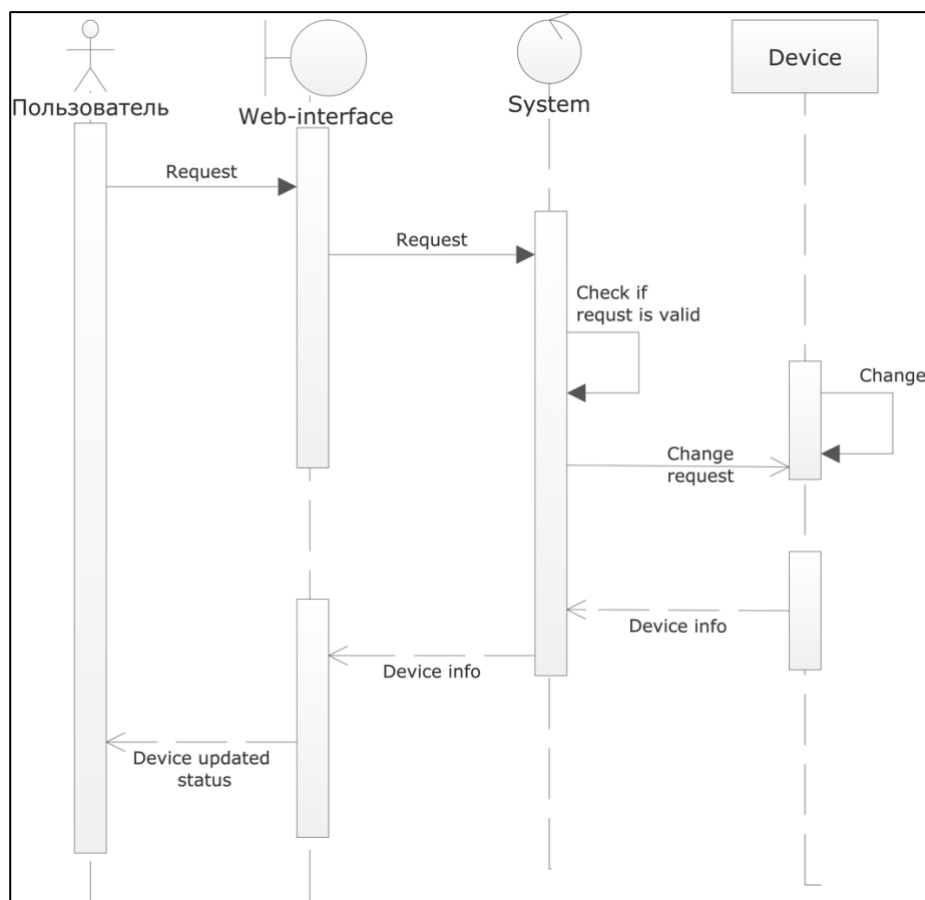


Рисунок 12 – Диаграмма последовательности

2.3.3 Диаграммы деятельности. При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности.

Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов.

На рисунке 13 представлена диаграмма деятельности «Включить музыку».

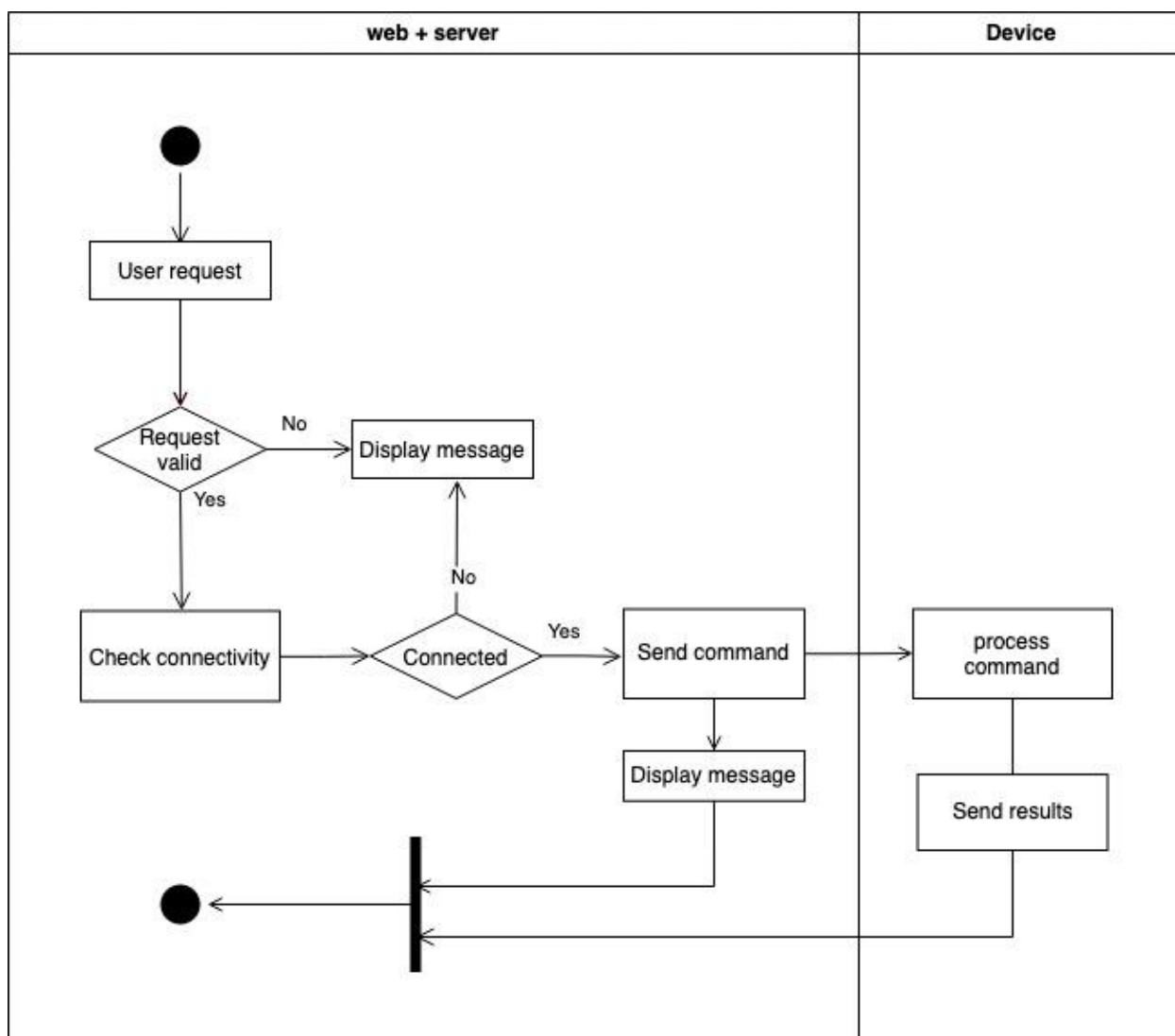


Рисунок 13 – Диаграмма деятельности

3 РЕАЛИЗАЦИОННАЯ ЧАСТЬ

3.1 Инструментальное обеспечение

Система представляет из себя серверное приложение и веб-интерфейс, написанные на языке *Python* и связки *HTML+CSS+Javascript*.

Для разработки приложения использованы следующие программные средства:

- язык программирования *Python*;
- СУБД *SQLite*;
- *Raspberry Pi 4* в качестве физического сервера для системы.
- *Docker* для сборки приложения в виде микросервиса;
- библиотеки *PyXiaomiGateway*, *aiodiscover*, *stookalert*, *PySocks*, *python-telegram-bot* и др.;
- среда разработки *Visual Studio Code*.

3.1.1 Язык программирования *Python*. Это высокоуровневый язык программирования общего назначения, обладающий большой стандартной библиотекой, содержащей множество полезных функций и поддерживающий несколько парадигм программирования.

Python является мультипарадигмальным языком программирования, поддерживающим императивное, процедурное, структурное, объектно-ориентированное программирование, метапрограммирование и функциональное программирование.

Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений с глобальной блокировкой интерпретатора (*GIL*), высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты. [4]

3.1.2 *Raspberry Pi* — одноплатный компьютер размером с банковскую карту, изначально разработанный как бюджетная система для обучения информатике, но позже получивший более широкое применение и известность.

На сегодняшний день существует 11 разновидностей *Raspberry Pi*. Последние версии оснащены беспроводными *WiFi* и *Bluetooth* модулями, расширяющими границы применения мини-ПК в области *Ethernet*-технологий.

Данная плата, как и обычный ПК, работает под управлением одной из специализированных операционных систем. В зависимости от области применения или личных симпатий, каждый может выбрать для себя свою. Ниже приведён перечень наиболее популярных ОС для *Raspberry Pi* с их кратким описанием.

Raspbian – данная операционная система в 2015 году была представлена как основная для *Raspberry Pi*. Она по максимуму оптимизирована для процессоров с АРМ-архитектурой и достаточно активно продолжает развиваться. Основой операционной системы является *Debian GNU/Linux*. В состав дистрибутива входят: программа компьютерной алгебры *Mathematica*; модифицированная версия *Minecraft PI*; урезанная версия *Chrome*.

Debian – операционная система с открытым исходным кодом. В состав *Debian* входит более 59000 пакетов уже скомпилированного ПО. Система использует ядро *Linux* или *FreeBSD*.

Ubuntu – система основана на *Debian GNU/Linux*. По популярности *Ubuntu* занимает первое место среди дистрибутивов *Linux*, предназначенных для web-серверов.

3.1.3 Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть развёрнут на любой *Linux*-системе с поддержкой контрольных групп в ядре, а также предоставляет набор команд для управления этими контейнерами. [4] Изначально использовал возможности *LXC*, с 2015 года начал использовать собственную библиотеку, абстрагирующую виртуализационные возможности ядра *Linux* — *libcontainer*.

3.1.4 Физически схема компонентов выглядит следующим образом:

- *Raspberry Pi*;
- *ZigBee* адаптер для приема сигналов по протоколу *ZigBee* (рисунок 14);
- *Bluetooth* адаптер;
- блок питания;



Рисунок 14 – ZigBee адаптер

Фото полного набора для установки приведено на рисунке 15.



Рисунок 15 – Набор для Raspberry Pi 4

3.2 Программное обеспечение

3.2.1 Приведем пример кода основного класса, который реализует и являются родительскими для всего функционала.

```

class HomeAssistant:
    """Root object of the Home Assistant home automation."""

    auth: AuthManager
    http: HomeAssistantHTTP = None # type: ignore
    config_entries: ConfigEntries = None # type: ignore

    def __init__(self) -> None:
        """Initialize new Home Assistant object."""
        self.loop = asyncio.get_running_loop()
        self._pending_tasks: list = []
        self._track_task = True
        self.bus = EventBus(self)
        self.services = ServiceRegistry(self)
        self.states = StateMachine(self.bus, self.loop)
        self.config = Config(self)
        self.components = loader.Components(self)
        self.helpers = loader.Helpers(self)
        # This is a dictionary that any component can store any data on.
        self.data: dict = {}
        self.state: CoreState = CoreState.not_running
        self.exit_code: int = 0
        # If not None, use to signal end-of-loop
        self._stopped: asyncio.Event | None = None
        # Timeout handler for Core/Helper namespace
        self.timeout: TimeoutManager = TimeoutManager()

    @property
    def is_running(self) -> bool:
        """Return if Home Assistant is running."""
        return self.state in (CoreState.starting, CoreState.running)

    @property
    def is_stopping(self) -> bool:
        """Return if Home Assistant is stopping."""
        return self.state in (CoreState.stopping, CoreState.final_write)

```

```

def start(self) -> int:
    """Start Home Assistant.

    Note: This function is only used for testing.
    For regular use, use "await hass.run()".
    """

    # Register the async start
    fire_coroutine_threadsafe(self.async_start(), self.loop)

    # Run forever
    # Block until stopped
    _LOGGER.info("Starting Home Assistant core loop")
    self.loop.run_forever()
    return self.exit_code

async def async_run(self, *, attach_signals: bool = True) -> int:
    """Home Assistant main entry point.

    Start Home Assistant and block until stopped.

    This method is a coroutine.
    """
    if self.state != CoreState.not_running:
        raise RuntimeError("Home Assistant is already running")

    # _async_stop will set this instead of stopping the loop
    self._stopped = asyncio.Event()

    await self.async_start()
    if attach_signals:
        # pylint: disable=import-outside-toplevel
        from homeassistant.helpers.signal import async_register_signal_handling

        async_register_signal_handling(self)

    await self._stopped.wait()
    return self.exit_code

```



```
def add_job(self, target: Callable[..., Any], *args: Any) -> None:
    """Add a job to be executed by the event loop or by an executor.
```

If the job is either a coroutine or decorated with @callback, it will be run by the event loop, if not it will be run by an executor.

target: target to call.

args: parameters for method to call.

"""

if target is None:

raise ValueError("Don't call add_job with None")

*self.loop.call_soon_threadsafe(self.async_add_job, target, *args)*

@callback

```
def async_add_job(
```

*self, target: Callable[..., Any], *args: Any*

```
) -> asyncio.Future | None:
```

"""Add a job to be executed by the event loop or by an executor.

If the job is either a coroutine or decorated with @callback, it will be run by the event loop, if not it will be run by an executor.

This method must be run in the event loop.

target: target to call.

args: parameters for method to call.

"""

if target is None:

raise ValueError("Don't call async_add_job with None")

if asyncio.iscoroutine(target):

return self.async_create_task(cast(Coroutine, target))

*return self.async_add_hass_job(HassJob(target), *args)*

@callback

```
def create_task(self, target: Awaitable) -> None:
```

"""Add task to the executor pool.

target: target to call.
"""

self.loop.call_soon_threadsafe(self.async_create_task, target)

@callback

def async_create_task(self, target: Awaitable) -> asyncio.Task:
 """Create a task from within the eventloop.

This method must be run in the event loop.

target: target to call.
"""

task: asyncio.Task = self.loop.create_task(target)

if self._track_task:
 self._pending_tasks.append(task)

return task

@callback

def async_add_executor_job(
 *self, target: Callable[..., T], *args: Any*
) -> Awaitable[T]:
 """Add an executor job from within the event loop."""
 *task = self.loop.run_in_executor(None, target, *args)*

If a task is scheduled

if self._track_task:
 self._pending_tasks.append(task)

return task

@callback

def async_track_tasks(self) -> None:
 """Track tasks so you can wait for all tasks to be done."""
 self._track_task = True

```

@callback
def async_stop_track_tasks(self) -> None:
    """Stop track tasks so you can't wait for all tasks to be done."""
    self._track_task = False

```

```

@callback
def async_run_job(
    self, target: Callable[..., None | Awaitable], *args: Any
) -> asyncio.Future | None:
    """Run a job from within the event loop.

```

This method must be run in the event loop.

target: target to call.

args: parameters for method to call.

"""

```

    if asyncio.iscoroutine(target):
        return self.async_create_task(cast(Coroutine, target))

    return self.async_run_hass_job(HassJob(target), *args)

```

```

def block_till_done(self) -> None:
    """Block until all pending work is done."""
    asyncio.run_coroutine_threadsafe(
        self.async_block_till_done(), self.loop
    ).result()

```

```

async def async_block_till_done(self) -> None:
    """Block until all pending work is done."""
    # To flush out any call_soon_threadsafe
    await asyncio.sleep(0)
    start_time: float | None = None

```

```

while self._pending_tasks:
    pending = [task for task in self._pending_tasks if not task.done()]
    self._pending_tasks.clear()
    if pending:
        await self._await_and_log_pending(pending)

```

```

    if start_time is None:
        # Avoid calling monotonic() until we know
        # we may need to start logging blocked tasks.
        start_time = 0
    elif start_time == 0:
        # If we have waited twice then we set the start
        # time
        start_time = monotonic()
    elif monotonic() - start_time > BLOCK_LOG_TIMEOUT:
        # We have waited at least three loops and new tasks
        # continue to block. At this point we start
        # logging all waiting tasks.
        for task in pending:
            _LOGGER.debug("Waiting for task: %s", task)
    else:
        await asyncio.sleep(0)

def stop(self) -> None:
    """Stop Home Assistant and shuts down all threads."""
    if self.state == CoreState.not_running: # just ignore
        return
    fire_coroutine_threadsafe(self.async_stop(), self.loop)

async def async_stop(self, exit_code: int = 0, *, force: bool = False) -> None:
    """Stop Home Assistant and shuts down all threads.

```

The "force" flag commands `async_stop` to proceed regardless of Home Assistant's current state. You should not set this flag unless you're testing.

This method is a coroutine.

"""

```

if not force:
    # Some tests require async_stop to run,
    # regardless of the state of the loop.
    if self.state == CoreState.not_running: # just ignore
        return

```

```

    if self.state in [CoreState.stopping, CoreState.final_write]:
        _LOGGER.info("Additional call to async_stop was ignored")
        return
    if self.state == CoreState.starting:
        # This may not work
        _LOGGER.warning(
            "Stopping Home Assistant before startup has completed may fail"
        )

    # stage 1
    self.state = CoreState.stopping
    self.async_track_tasks()
    self.bus.async_fire(EVENT_HOMEASSISTANT_STOP)
    try:
        async with
self.timeout.async_timeout(STAGE_1_SHUTDOWN_TIMEOUT):
        await self.async_block_till_done()
    except asyncio.TimeoutError:
        _LOGGER.warning(
            "Timed out waiting for shutdown stage 1 to complete, the shutdown will
continue"
        )

    # stage 2
    self.state = CoreState.final_write
    self.bus.async_fire(EVENT_HOMEASSISTANT_FINAL_WRITE)
    try:
        async with
self.timeout.async_timeout(STAGE_2_SHUTDOWN_TIMEOUT):
        await self.async_block_till_done()
    except asyncio.TimeoutError:
        _LOGGER.warning(
            "Timed out waiting for shutdown stage 2 to complete, the shutdown will
continue"
        )

    # stage 3
    self.state = CoreState.not_running

```

```

self.bus.async_fire(EVENT_HOMEASSISTANT_CLOSE)

# Prevent run_callback_threadsafe from scheduling any additional
# callbacks in the event loop as callbacks created on the futures
# it returns will never run after the final `self.async_block_till_done`
# which will cause the futures to block forever when waiting for
# the `result()` which will cause a deadlock when shutting down the executor.
shutdown_run_callback_threadsafe(self.loop)

try:
    async with
self.timeout.async_timeout(STAGE_3_SHUTDOWN_TIMEOUT):
    await self.async_block_till_done()
except asyncio.TimeoutError:
    _LOGGER.warning(
        "Timed out waiting for shutdown stage 3 to complete, the shutdown will
continue"
    )

self.exit_code = exit_code
self.state = CoreState.stopped

if self._stopped is not None:
    self._stopped.set()

```

3.3 Организационное обеспечение

3.3.1 Для доступа к системе пользователю необходимо иметь доступ к локальной сети *Wi-Fi* и возможность использовать браузер. При введении адреса *https://homeassistant.my* откроется веб-интерфейс системы.

При первом запуске пользователь должен создать свою учетную запись и пароль, который будет использован в дальнейшем для авторизации. Форма регистрации показана на рисунке 16.

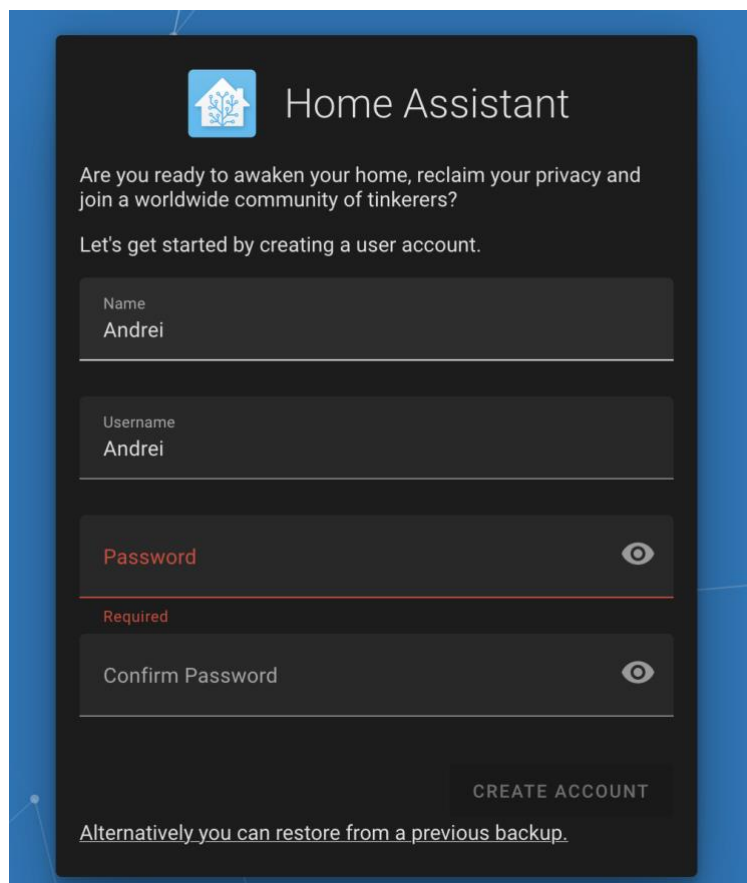
The image shows a user registration interface for Home Assistant. At the top left is the Home Assistant logo (a blue square with a white house icon). To its right is the text "Home Assistant". Below the logo and title, there is a motivational message: "Are you ready to awaken your home, reclaim your privacy and join a worldwide community of tinkerers?" followed by "Let's get started by creating a user account." The registration form consists of four input fields: "Name" with the value "Andrei", "Username" with the value "Andrei", "Password" (with a red "Required" label and a toggle eye icon), and "Confirm Password" (with a toggle eye icon). A "CREATE ACCOUNT" button is located at the bottom right of the form. Below the button, there is a link: "Alternatively you can restore from a previous backup." The entire form is set against a dark background with a blue border.

Рисунок 16 – Форма регистрации пользователя

Однако большую часть времени пользователь будет попадать на домашнюю страницу – *Dashboard*, на которой добавлена информация с датчиков, показаны уведомления и отчеты. Пример домашней страницы пользователя показан на рисунке 17.

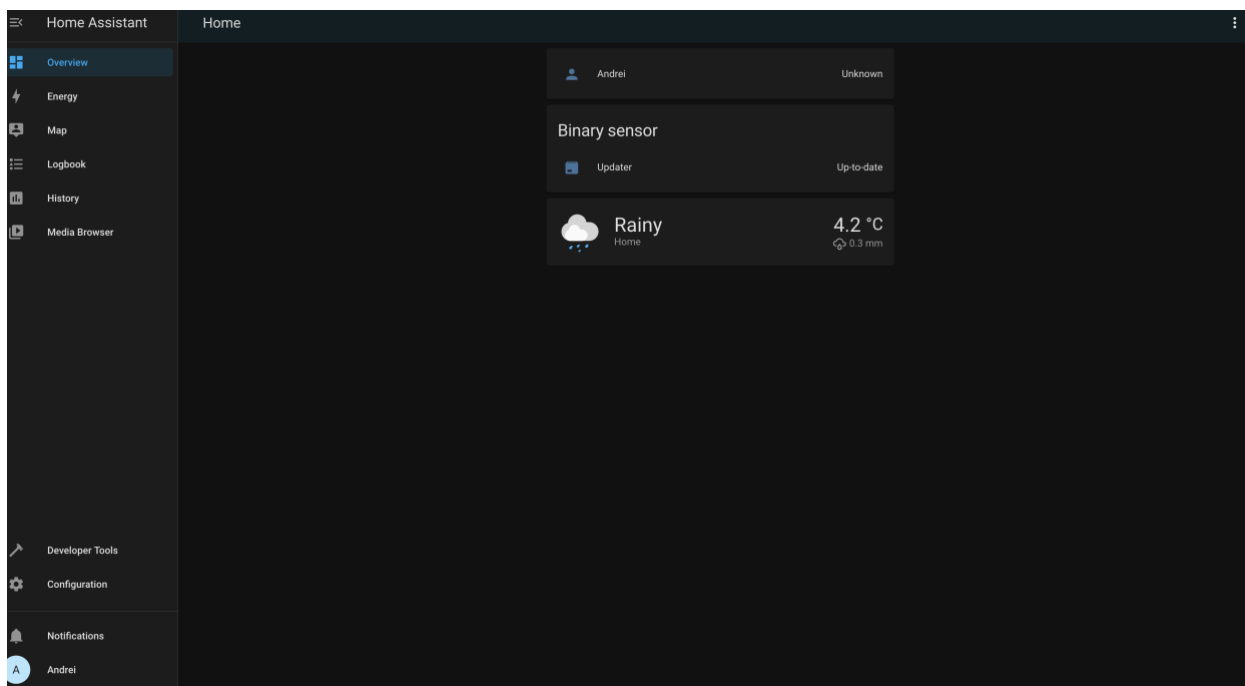


Рисунок 17 – Домашняя страница пользователя

3.3.2 Приведем пример добавления умного девайса *Xiomi* в систему. После установки и настройки *Mi Spart Led Bulb*, пользователь должен получить токен девайса и локальный *IP* адрес. Так же необходимо включить режим разработчика на устройстве.

После этого пользователь может просто добавить интеграцию через панель настроек. Пример добавления умной лампочки показан на рисунках 18 и 19.

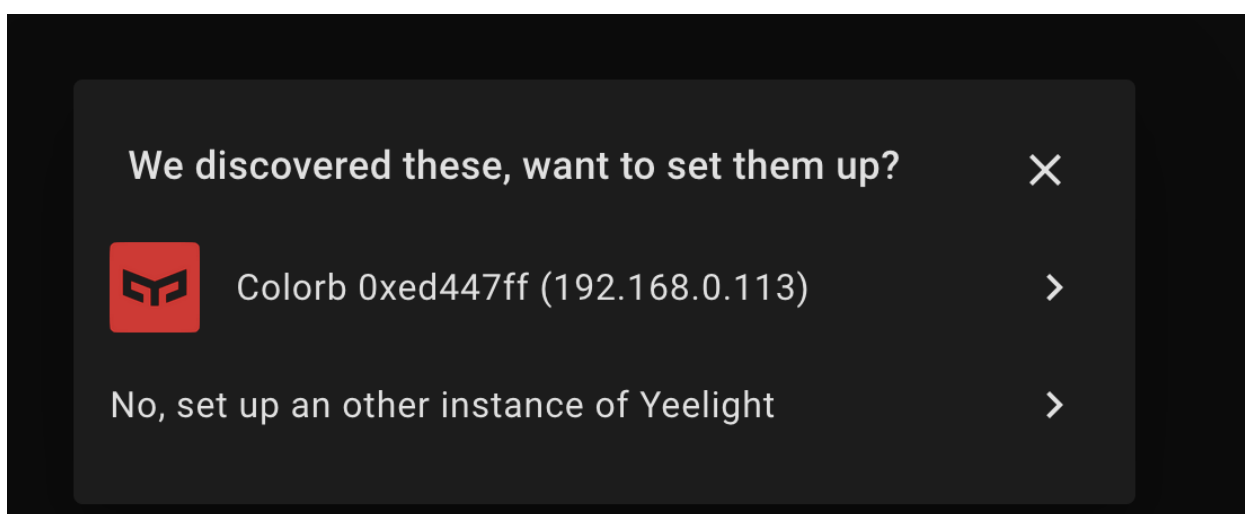


Рисунок 18 – Экран добавления умного устройства

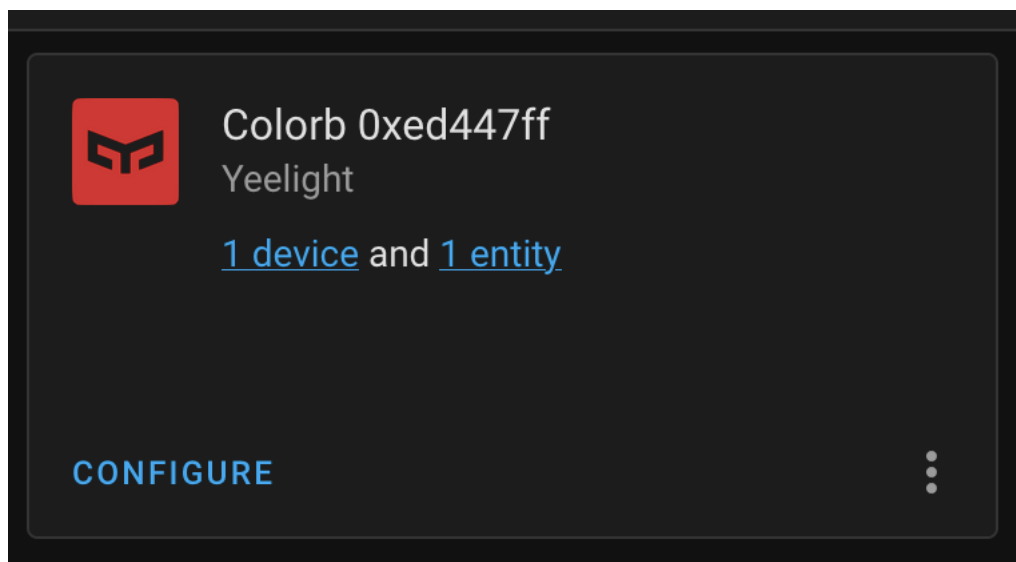


Рисунок 19 – Умная лампочка в разделе «Добавленные конфигурации»

Добавив устройство на главную страницу, пользователь может менять его конфигурацию путем нажатия кнопок на веб-интерфейсе. Пример включения лампочки приведен на рисунке 20.

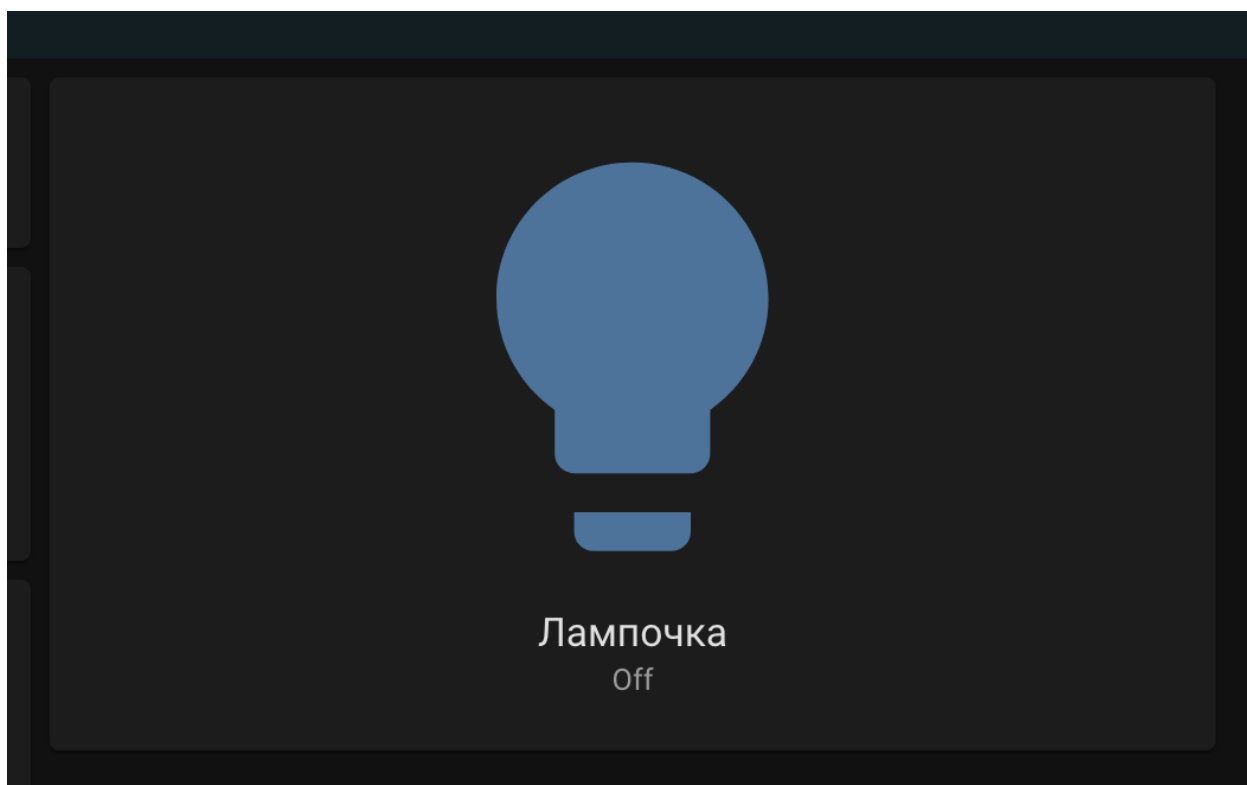


Рисунок 20 – Управление умной лампочкой *Xiaomi*

3.3.2 Приведем пример добавления уведомления об открытии двери, когда пользователь не находится дома. Для этого необходимы уже настроенные интеграция с телеграмм [4] и датчиком открытия дверей.

Для создания уведомления пользователю нужно всего лишь добавить запись в файл *configuration.yaml*. Например, вот так выглядит уведомление об открытии двери и проигрывание сообщения на умной колонке:

```
alias: Telegram - acknowledge door left open alarm
trigger:
  - event_data:
      data: /door_left_open_alarm_ack
      event_type: telegram_callback
      platform: event
action:
  - service: telegram_bot.answer_callback_query
    data_template:
      callback_query_id: '{{ trigger.event.data.id }}'
      message: A message has been played.
  - service: tts.google_translate_say
    entity_id: media_player.speaker
    data:
      message: "Warning! Door has been opened! Please leave immediately
from my room!"
```

Пример взаимодействия с домашней системой через телеграмм-бота приведен на рисунке 21.

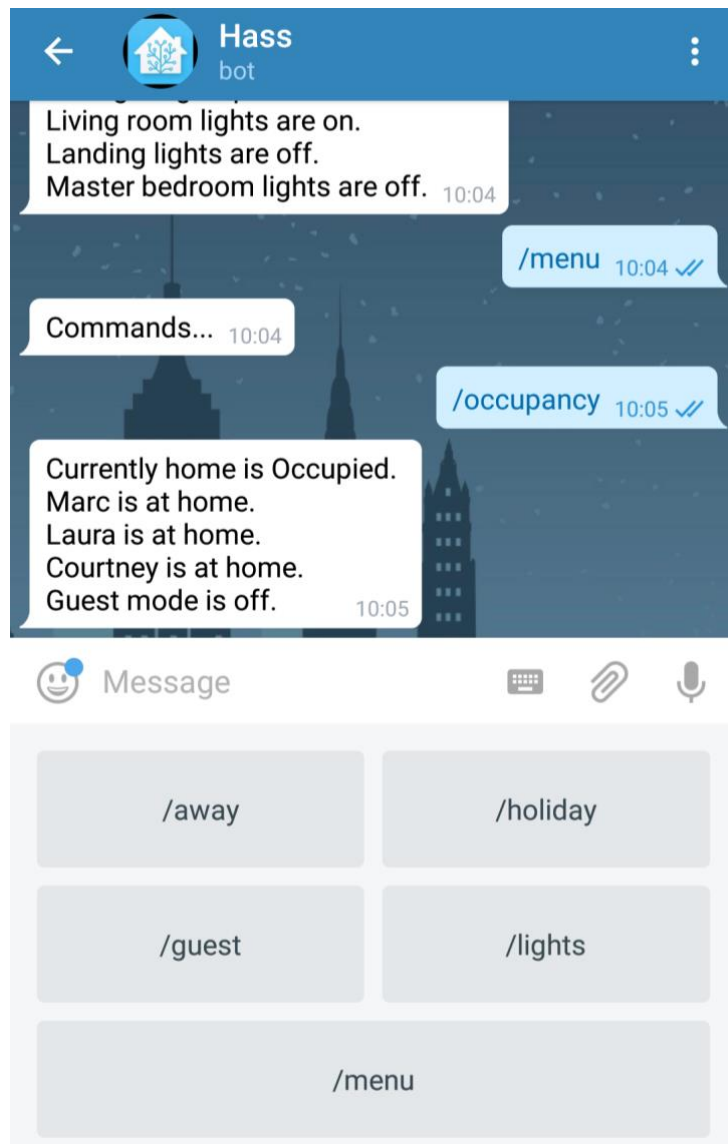


Рисунок 21 – Телеграмм-бот

ЗАКЛЮЧЕНИЕ

В курсовой работе использованы методы проектирования автоматизированных систем. Показан метод графического представления автоматизированной системы с помощью диаграмм.

Для «системы домашней автоматизации» разработаны диаграммы *IDEF0*, *IDEF3*, *DFD* и *UML*-диаграммы. Изучены понятия класс, атрибут, операция, связь, сущность и т.д.

Определен набор технологий для реализации программного обеспечения системы. Спроектирован и разработан пользовательский интерфейс, и настроены физические устройства, которые позволяют максимально просто и удобно пользоваться всеми функциями системы.

Система позволит пользователю агрегировать информацию с IoT устройств и датчиков и управлять ими, а также работать с веб приложениями, у которых есть свой API. Таким образом система удовлетворяет всем техническим и функциональным требованиям, имеет легкий в использовании веб-интерфейс, позволяющий снизить затраты и время на подключение каждого устройства и датчика по-отдельности.

Функциональность системы можно расширять за счет добавления нового функционала и новых периферийных устройств.

Кроме этого, система может быть улучшена за счет предоставления доступа из глобальной сети, чтобы пользователь смог получать доступ в системы за пределами дома (локальной сети).

В ходе проведения курсовой работы были углублены, закреплены и конкретизированы теоретические знания в области автоматизации процессов, приобретены навыки по проектированию.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Гайдамакин, Н. А. Автоматизированные информационные системы, базы и банки данных. Вводный курс: Учебное пособие / Н. А. Гайдамакин. – Москва. : Гелиос АРВ, 2002. – 68 с.

[2] Бабич, А. В. *UML*: Первое знакомство / А.В. Бабич. – Москва.: БИНОМ. Лаборатория знаний, 2008. – 342 с.

[3] *Home Assistant Telegram* [Электронный ресурс]. – Режим доступа: <https://www.home-assistant.io/integrations/telegram/>.

[4] Прохоренок Н., Дронов В. Введение / *Python 3*. Самое необходимое, 2-е изд. / Дронов В — БХВ-Петербург, 2019. – 608 с.

[5] *Dirk Merkel. Docker: lightweight Linux containers for consistent development and deployment* (англ.) / *Dirk Merkel – Linux Journal*. — 2014. — Vol. March, no. 239.

ВЕДОМОСТЬ КУРСОВОГО ПРОЕКТА

Обозначение					Наименование					Дополнительные сведения				
					<u>Текстовые документы</u>									
БГУИР КП 1–53 01 02 01 08 ПЗ					Пояснительная записка					46 с.				
					<u>Графические документы</u>									
ГУИР.000.001.ПД					Диаграмма вариантов использования					Формат А2				
ГУИР.000.02.ПД					Диаграмма классов					Формат А2				