МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
**ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)**
**ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ**
**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**
**В Г. ТАГАНРОГЕ РОСТОВСКОЙ ОБЛАСТИ**

**ПИ (филиал) ДГТУ в г. Таганроге**

Факультет «Высшего образования»
наименование факультета
Кафедра «Технический сервис и информационные технологии»
наименование кафедры

# ЛАБОРАТОРНАЯ РАБОТА

Дисциплина (модуль) Перспективные информационные технологии
наименование учебной дисциплины (модуля)

на тему: Лабораторная работа 3.1 «Схемы распределительных систем с IoT. Виды и типы»

Направление подготовки/специальность 09.03.02 Информационные системы и технологии
код наименование направления подготовки/специальности

Направленность (профиль) Информационные системы и технологии

Номер зачетной книжки 2282149 Группа ВЗ ИСиТ – 41

Обучающийся _____ А.Ю.Галетко
подпись, дата И.О. Фамилия

Контрольную работу проверил _____ М.В. Орда-Жигулина
подпись, дата должность, И.О. Фамилия

Таганрог
2026г.

# Содержание

# Введение

Соблюдение стандартов, таких как ГОСТ, обеспечивает единообразие, понятность и универсальность создаваемых схем. Это позволяет разным специалистам «говорить на одном языке» при проектировании, сборке и обслуживании сложных систем. В данной работе мы будем ориентироваться на стандарты ГОСТ 2.701 и 2.702 для классификации и правил выполнения схем.

Цель лабораторной работы: изучить принципы построения схем распределительных систем IoT в соответствии со стандартами ГОСТ, освоить инструменты Google Colab, PlantUML Editor и draw.io для визуализации архитектуры и приобрести навык создания структурных и функциональных диаграмм (Deployment и State) для IoT-системы.

Для достижения поставленной цели необходимо решить следующие задачи:

– изучить принципы построения схем распределительных систем IoT в соответствии со стандартами ГОСТ;

– освоить инструменты Google Colab, PlantUML Editor и draw.io для визуализации архитектуры;

– приобрести навык создания структурных и функциональных диаграмм (Deployment и State) для IoT-системы.

## Задание 1

Типы схем по ГОСТ 2.701

Структурная – основные функциональные части (Edge/Fog/Cloud) и связи между ними.

Функциональная – процессы/потоки данных и управления (события, задержки и т.д.).

Принципиальная – уровень элементов для электроники и логики (датчики, интерфейсы).

Соединений/подключения/расположения – топология портов, шин, кабелей, размещения узлов и т.д.

Диаграммы для распределённых систем IoT (UML/SysML ↔ ГОСТ)

– Deployment – узлы и сети: Edge-устройства, Fog-шлюз (Home_Gateway), Cloud-сервисы → структурная схема.

– Component – сервисы и драйверы (MQTT-брокер, Ingestor, Threshold/ML) → структурная/функциональная.

– Sequence/Activity – обмены и потоки (publish/subscribe, алерты) → функциональная.

– State – режимы устройств/узлов (online/offline, in_work) → функциональная (состояний).

– Package/Context – границы подсистем, ответственность и уровни.

Для описания распределенных систем IoT наиболее наглядными и практичными являются структурные и функциональные схемы. В отличие от детализированных принципиальных схем, они позволяют абстрагироваться от аппаратной реализации и сосредоточиться на ключевых компонентах, связях и потоках данных, что делает их понятными для более широкой аудитории, включая не-инженеров. В контексте программной инженерии этим целям лучше всего соответствуют диаграммы UML/SysML. В данной работе мы сосредоточимся на:

– Deployment – диаграмме (UML) как аналоге структурной схемы (ГОСТ): она показывает «из чего состоит система» — устройства, узлы и физические соединения.

– State – диаграмме (UML) как аналоге функциональной схемы состояний (ГОСТ): она показывает «как ведет себя компонент» — его реакцию на события и смену состояний.

Необходимые навыки:

Базовые навыки работы с компьютером, умение пользоваться веб-браузером. Опыт работы с графическими редакторами или программированием приветствуется, но не является обязательным.

1. Компоненты системы

Система состоит из пяти компонентов, распределенных по уровням вычислений:

Уровни архитектуры:

– Краевой уровень (Edge): Робот-датчик — сбор сырых данных

– Туманный уровень (Fog): Fog-обработчик — локальная обработка

– Облачный уровень (Cloud): Ноутбук-сервер — аналитика и хранение

Связующие компоненты:

– Робот-курьер — транспортировка данных

– Смартфон — буферизация и передача

На рисунке 1 представлена схема работы компонентов системы.

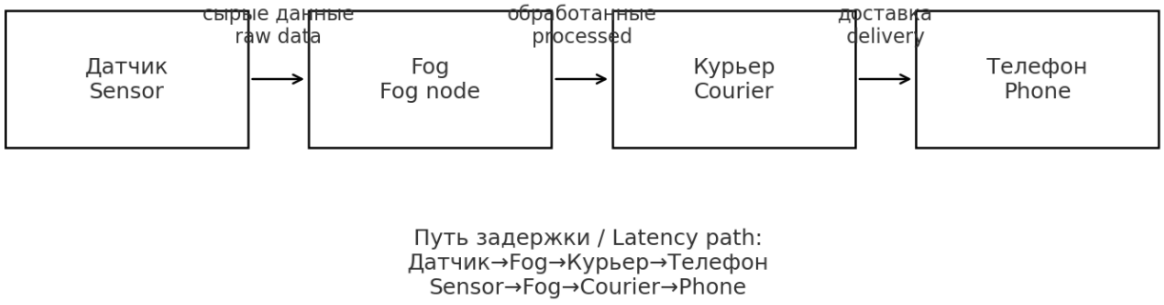



Рисунок 1 – схема работы компонентов системы

Следующая схема представляет собой логическую модель компонентов распределенной системы «Умный дом». Она показывает основные

функциональные блоки и направление потока данных между ними. Каждый компонент выполняет строго определенную роль в цепочке обработки информации: от сбора данных (робот-датчик) до их конечного анализа (ноутбук-сервер).

На рисунке 2 представлена схема архитектуры системы.

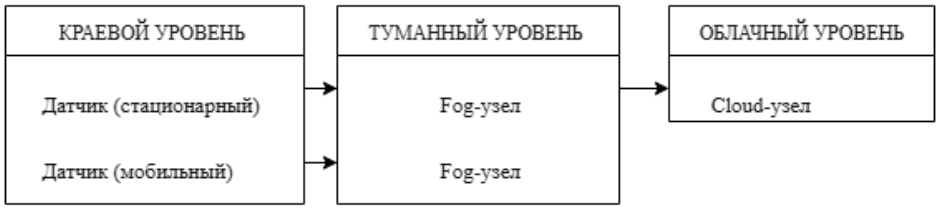



Рисунок 2 - Схема архитектуры системы

Схема на рисунке 3 является технической реализацией логической модели, показанной на Рисунке 2. Она детализирует как именно компоненты взаимодействуют на практике, показывает сетевые интерфейсы, протоколы передачи данных и физическую структуру системы. Здесь наглядно отображено разделение на сетевые сегменты (например, локальную сеть датчиков и облачный сервер).

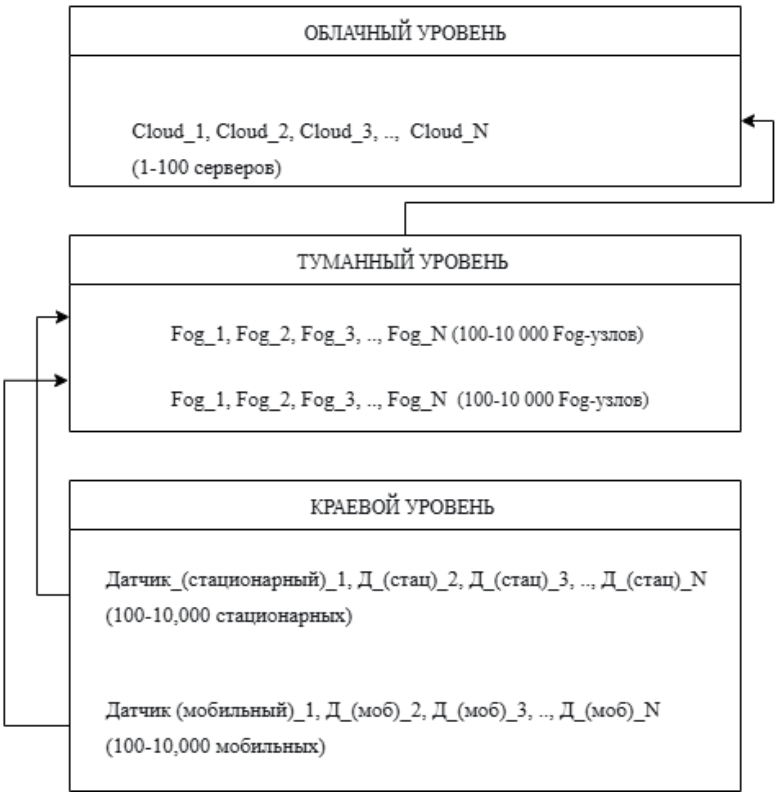



Рисунок 3 – Схема эталонной архитектуры системы

Взаимосвязь и взаимодействие между схемами

Обе схемы описывают одну и ту же систему, но на разных уровнях абстракции:

– Рисунок 2 отвечает на вопрос «ЧТО делает система?» — показывает основные роли и поток данных.

– Рисунок 3 отвечает на вопрос «КАК система это делает?» — показывает техническую реализацию и сетевые взаимодействия.

Краткое описание взаимодействия:

1. Робот-датчик собирает данные и передает их Fog-обработчику по локальному сетевому протоколу (например, Wi-Fi или Bluetooth)

2. Fog-обработчик выполняет первичную обработку и передает результат роботу-курьеру через локальную сеть

3. Робот-курьер доставляет данные смартфону через интернет (мобильная сеть 4G/5G)

4. Смартфон буферизует данные и передает их ноутбуку-серверу через облачное API или прямое сетевое соединение.

## Задание 2

GOOGLE COLAB

Google Colab – бесплатная облачная среда разработки, которая позволяет писать и выполнять код на Python прямо в браузере.

Diagrams – библиотека для Python, которая позволяет программно создавать диаграммы архитектуры облачных систем.

2.1 Пример работы в Google Colab

1. Открыли новый блокнот на https://colab.research.google.com.

2. В новой ячейке установили и импортировали библиотеку для построения графиков:

>>> !pip install diagrams

На рисунке 4 представлена реализация команды для установки и импорта библиотеки.

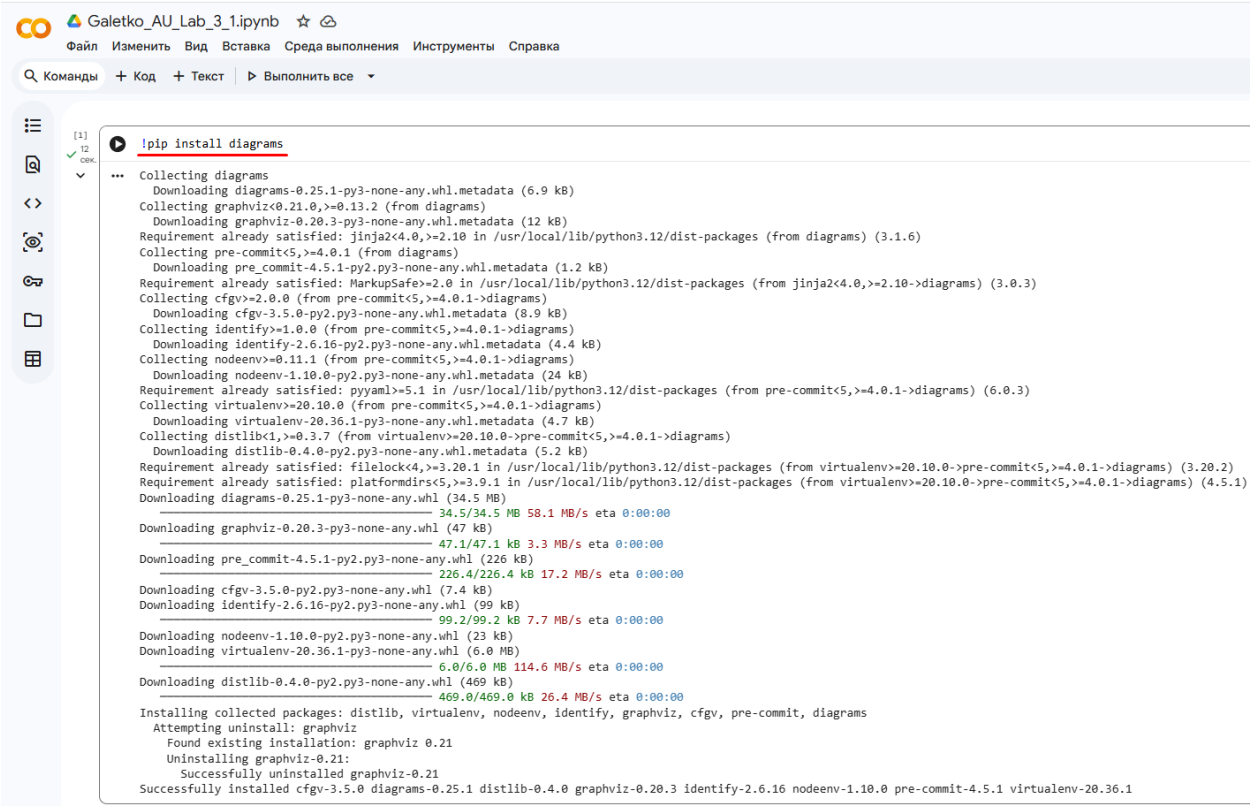



Рисунок 4 – Реализация команды для установки и импорта библиотеки

3. Создали новую ячейку и скопировали в нее следующий код для визуализации архитектуры:

```
from diagrams import Diagram
from diagrams.aws.compute import EC2
from diagrams.aws.network import InternetGateway
from diagrams.onprem.client import User
from diagrams.onprem.compute import Server
from diagrams.aws.iot import IotSensor

with Diagram("Архитектура IoT системы 'Умный дом'", show=False, filename="iot_architecture"):
    # Определяем ноды (компоненты) системы с более подходящими иконками
    sensor = IotSensor("Робот-датчик")
    fog = EC2("Fog-обработчик")
    courier = InternetGateway("Робот-курьер")
    smartphone = User("Смартфон\n(Буфер)")
    server = Server("Ноутбук-сервер")

    # Строим конвейер обработки данных
    sensor >> fog >> courier >> smartphone >> server

print("Схема сохранена как 'iot_architecture.png'")
print("Файл будет доступен в левой панели управления файлами Colab")
```

На рисунке 5 представлена реализация кода.

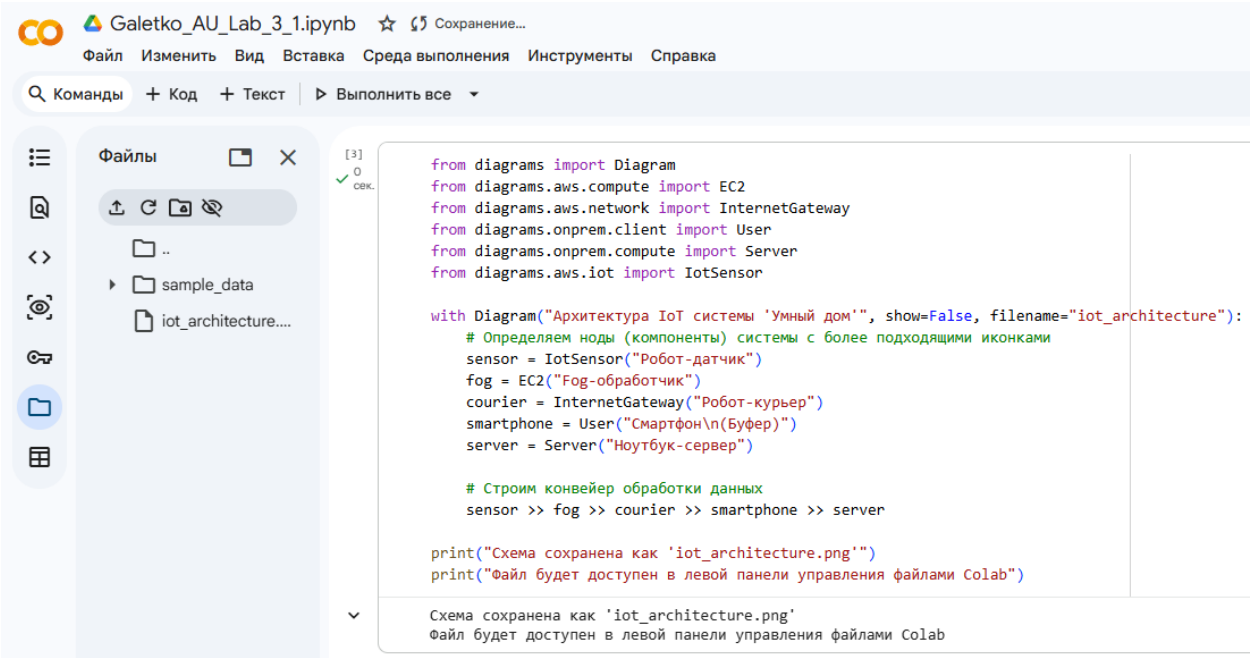



Рисунок 5 – Реализация кода

Запустили ячейку. После выполнения код создал файл iot_architecture.png в левой панели управления файлами. Открыли его, чтобы увидеть схему нашей системы.

На рисунке 6 представлена полученная диаграмма.

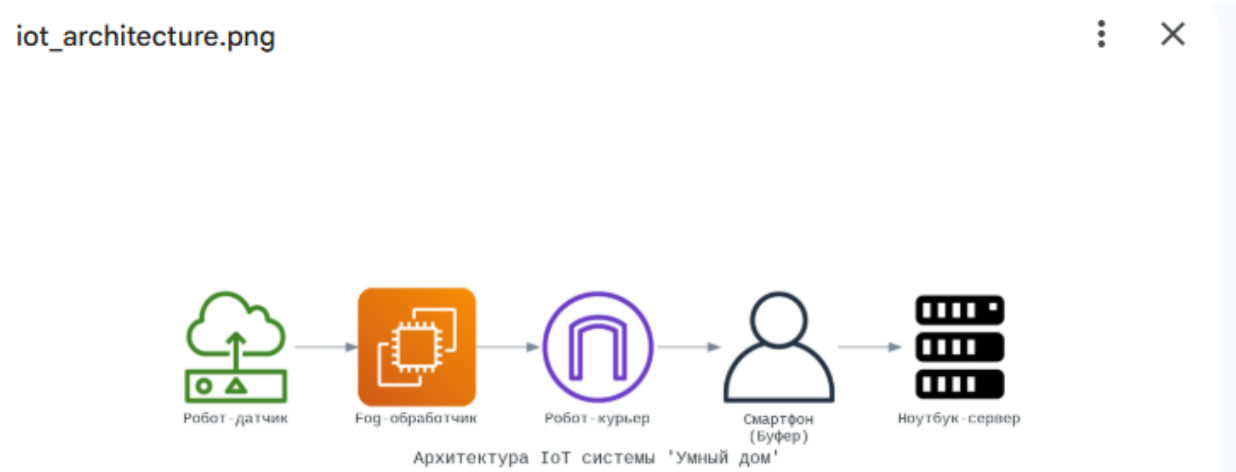



Рисунок 6 – Полученная диаграмма

2.2 Задание для построения схемы через Google Colab

Используя пример из раздела 2.1, построили Deployment-диаграмму для системы «Умный дом», отображающую физические узлы и связи между ними.

Изменили код следующим образом:

– Добавили второго «Робота-датчика»

– Добавили второго «Fog-обработчика»

– Изменили направление данных: первый датчик → первый Fog, второй датчик → второй Fog

– Оба Fog-обработчика передают данные на «Ноутбук-сервер»

– Изменили иконки для объектов

Измененный код:

```
from diagrams import Diagram, Cluster
from diagrams.aws.iot import IotCamera, IotButton, IotMqtt
from diagrams.aws.compute import Lambda, Fargate
from diagrams.aws.general import MobileClient, TraditionalServer

with Diagram("Модифицированная архитектура IoT 'Умный дом' (все
иконки                    изменены)",                    show=False,
filename="modified_iot_architecture_all_icons", direction="LR"):
    with Cluster("Edge Layer"):
        sensor1 = IotCamera("Робот-датчик 1")
        sensor2 = IotButton("Робот-датчик 2")

    with Cluster("Fog Layer"):
        fog1 = Lambda("Fog-обработчик 1")
        fog2 = Fargate("Fog-обработчик 2")
```

```
    courier = IotMqtt("Робот-курьер")
    smartphone = MobileClient("Смартфон\n(Буфер)")
    server = TraditionalServer("Ноутбук-сервер")

    sensor1 >> fog1
    sensor2 >> fog2
    fog1 >> courier
    fog2 >> courier
    courier >> smartphone >> server

print("Модифицированная схема сохранена как 'modified_iot_architecture_all_icons.png'")
print("Файл будет доступен в левой панели управления файлами Colab")
```

На рисунке 7 представлена реализация измененного кода. На рисунке 8 представлена полученная измененная диаграмма.

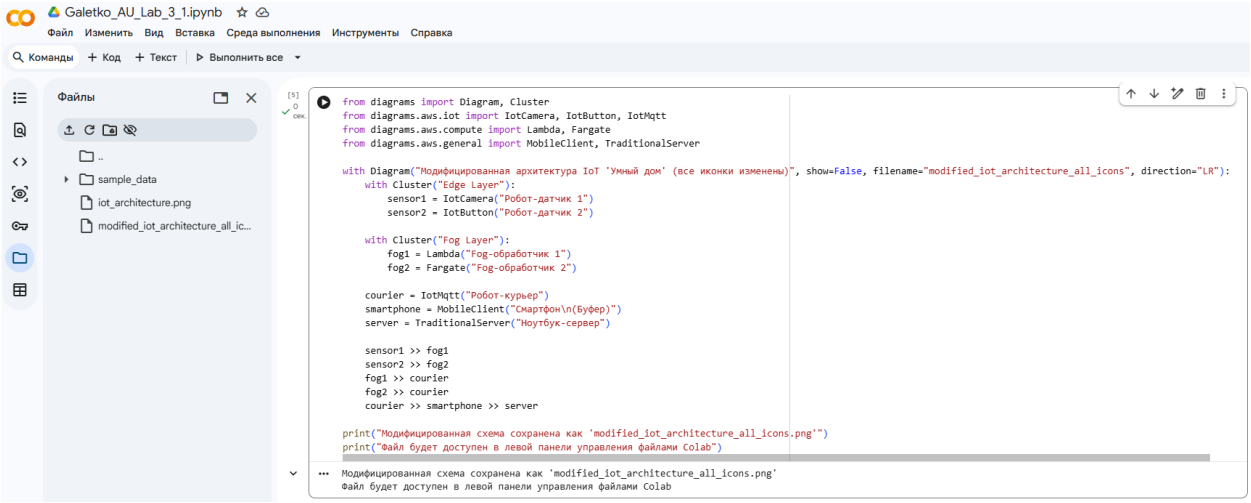



Рисунок 7 – Реализация измененного кода

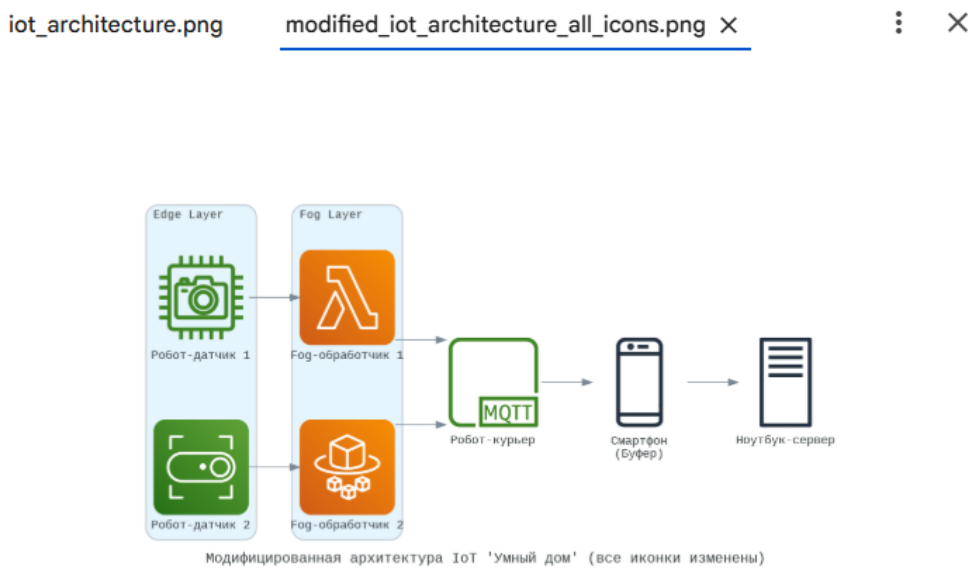



Рисунок 8 – Полученная измененная диаграмма

## Задание 3

PLANTUML EDITOR

PlantUML Editor — это инструмент для создания, редактирования и визуализации диаграмм с помощью простого текстового языка.

3.1 Работа в PlantUML Editor

1. Открыли онлайн-редактор PlantUML: https://www.plantuml.com/plantuml/

2. Скопировали и вставили следующий код для создания Deployment-диаграммы в панель редактора.

Листинг:

```
@startuml
skinparam shadowing false
skinparam defaultFontName Arial
skinparam nodesep 10
skinparam ranksep 20

title Deployment-диаграмма IoT системы "Умный дом"

node "Краевой уровень (Edge)" as edge_layer {
    [Робот-датчик 1]
    [Робот-датчик 2]
    [Робот-датчик N]
}

node "Туманный уровень (Fog)" as fog_layer {
    (Fog-обработчик 1)
    (Fog-обработчик 2)
}

cloud "Облачный уровень (Cloud)" as cloud_layer {
    (Ноутбук-сервер)
}

[Робот-датчик 1] --> (Fog-обработчик 1) : Wi-Fi
[Робот-датчик 2] --> (Fog-обработчик 1) : BLE
[Робот-датчик N] --> (Fog-обработчик 2) : LoRaWAN
(Fog-обработчик 1) --> (Ноутбук-сервер) : HTTPS
(Fog-обработчик 2) --> (Ноутбук-сервер) : MQTT
@enduml
```

На рисунке 9 предсавлена структурная схема, созданная с помощью PlantUML.

```
@startuml
skinparam shadowing false
skinparam defaultFontName Arial
skinparam nodesep 10
skinparam ranksep 20

title Deployment-диаграмма IoT системы "Умный дом"

node "Краевой уровень (Edge)" as edge_layer {
    [Робот-датчик 1]
    [Робот-датчик 2]
    [Робот-датчик N]
}

node "Туманный уровень (Fog)" as fog_layer {
    (Fog-обработчик 1)
    (Fog-обработчик 2)
}

cloud "Облачный уровень (Cloud)" as cloud_layer {
    (Ноутбук-сервер)
}

[Робот-датчик 1] --> (Fog-обработчик 1) : Wi-Fi
[Робот-датчик 2] --> (Fog-обработчик 1) : BLE
[Робот-датчик N] --> (Fog-обработчик 2) : LoRaWAN
(Fog-обработчик 1) --> (Ноутбук-сервер) : HTTPS
(Fog-обработчик 2) --> (Ноутбук-сервер) : MQTT
@enduml
```

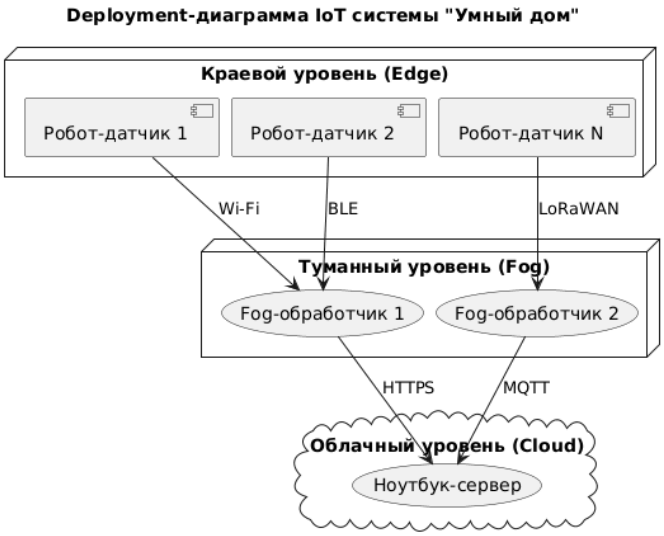



Рисунок 9 – Структурная схема, созданная с помощью PlantUML

3. Диаграмма появилась сама, мы увидели структурную схему, отображающую узлы системы и связи между ними.

4. Создали State-диаграмму. Очистили редактор и вставили следующий код:

```
@startuml
skinparam defaultFontName Arial

title State-диаграмма Fog-обработчика

[*] --> Ожидание

Ожидание --> Сбор_данных : получен запрос
Сбор_данных --> Обработка : данные считаны
Обработка --> Передача : результат готов
Передача --> Ожидание : отправлено подтверждение

Ожидание --> Перегрузка : очередь переполнена
Перегрузка --> [*] : аварийное отключение
@enduml
```

На рисунке 10 представлена State-диаграмма.

```
@startuml
skinparam defaultFontName Arial

title State-диаграмма Fog-обработчика

[*] --> Ожидание

Ожидание --> Сбор_данных : получен запрос
Сбор_данных --> Обработка : данные считаны
Обработка --> Передача : результат готов
Передача --> Ожидание : отправлено подтверждение

Ожидание --> Перегрузка : очередь переполнена
Перегрузка --> [*] : аварийное отключение
@enduml
```

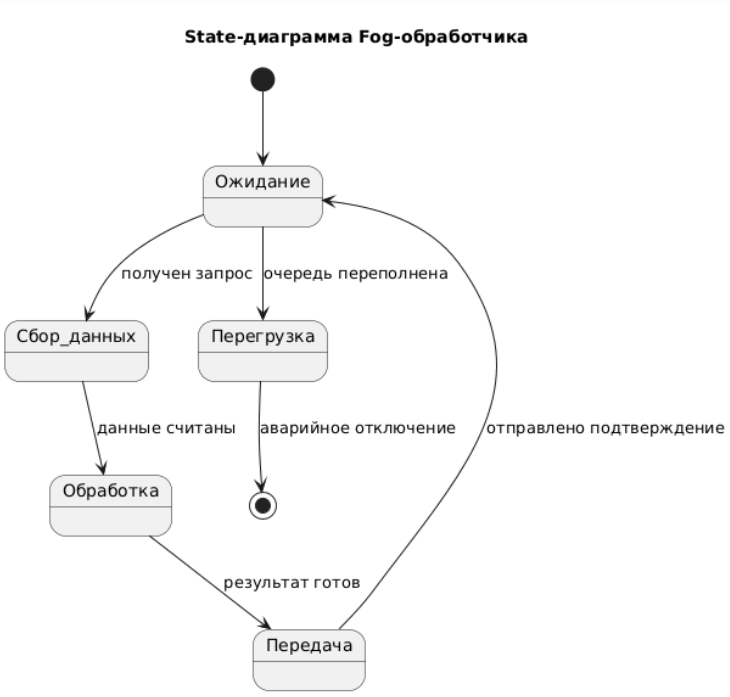



Рисунок 10 – State-диаграмма

3.2 Задание для построения схемы через PlantUML Editor

Используя примеры выше, создали в PlantUML Editor:

1. Deployment-диаграмму, отображающую не менее 3 краевых устройств, 2 Fog-узлов и 1 облачного сервера. Изменили названия компонентов и типы соединений на свои.

Листинг:

```
@startuml
skinparam shadowing false
skinparam defaultFontName Arial
skinparam nodesep 10
skinparam ranksep 20

title Deployment-диаграмма IoT системы "Умный дом"

node "Краевой уровень (Edge)" as edge_layer {
    [Датчик температуры 1]
    [Датчик влажности 2]
    [Датчик движения 3]
}

node "Туманный уровень (Fog)" as fog_layer {
    (Fog-шлюз 1)
    (Fog-шлюз 2)
}

cloud "Облачный уровень (Cloud)" as cloud_layer {
    (Сервер анализа)
}
```

```
[Датчик температуры 1] --> (Fog-шлюз 1) : ZigBee
[Датчик влажности 2] --> (Fog-шлюз 1) : Bluetooth
[Датчик движения 3] --> (Fog-шлюз 2) : Wi-Fi
(Fog-шлюз 1) --> (Сервер анализа) : Ethernet
(Fog-шлюз 2) --> (Сервер анализа) : MQTT
@enduml
```

На рисунке 11 представлена Deployment-диаграмма.

```
@startuml
skinparam shadowing false
skinparam defaultFontName Arial
skinparam nodesep 10
skinparam ranksep 20

title Deployment-диаграмма IoT системы "Умный дом"

node "Краевой уровень (Edge)" as edge_layer {
    [Датчик температуры 1]
    [Датчик влажности 2]
    [Датчик движения 3]
}

node "Туманный уровень (Fog)" as fog_layer {
    (Fog-шлюз 1)
    (Fog-шлюз 2)
}

cloud "Облачный уровень (Cloud)" as cloud_layer {
    (Сервер анализа)
}

[Датчик температуры 1] --> (Fog-шлюз 1) : ZigBee
[Датчик влажности 2] --> (Fog-шлюз 1) : Bluetooth
[Датчик движения 3] --> (Fog-шлюз 2) : Wi-Fi
(Fog-шлюз 1) --> (Сервер анализа) : Ethernet
(Fog-шлюз 2) --> (Сервер анализа) : MQTT
@enduml
```

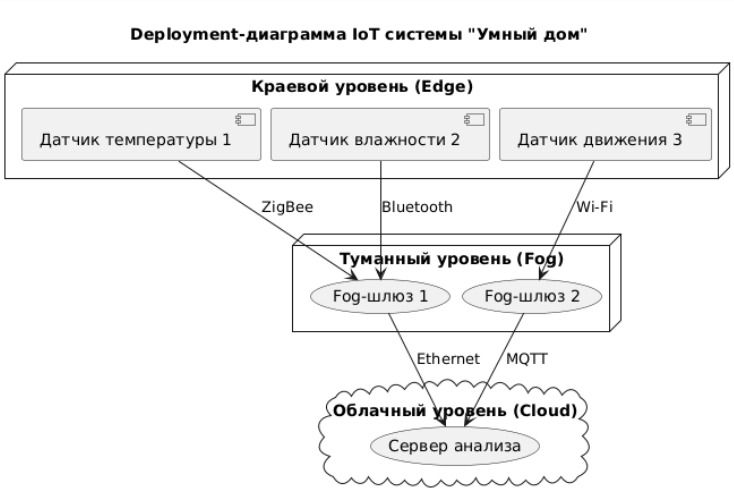



Рисунок 11 – Deployment-диаграмма

2. Создали State-диаграмму для «Робота-курьера». Добавили состояния. Сохранили обе диаграммы как PNG-файлы для отчета.

Листинг:

```
@startuml
skinparam defaultFontName Arial

title State-диаграмма Робота-курьера

[*] --> Зарядка
Зарядка --> Ожидание_задания : Заряжен
Ожидание_задания --> Навигация_к_смартфону : Получено задание
Навигация_к_смартфону --> Передача_данных : Прибыл
Передача_данных --> [*] : Данные переданы

Передача_данных --> Перегрузка : Буфер полон
Перегрузка --> Ожидание_задания : Проблема решена
@enduml
```

На рисунке 12 представлена State -диаграмма «Робота-курьера».

```
@startuml
skinparam defaultFontName Arial

title State-диаграмма Робота-курьера

[*] --> Зарядка
Зарядка --> Ожидание_задания : Заряжен
Ожидание_задания --> Навигация_к_смартфону : Получено задание
Навигация_к_смартфону --> Передача_данных : Прибыл
Передача_данных --> [*] : Данные переданы

Передача_данных --> Перегрузка : Буфер полон
Перегрузка --> Ожидание_задания : Проблема решена
@enduml
```


Рисунок 12 – State -диаграмма «Робота-курьера»

## Задание 4

DRAW.IO

Draw.io (с 2020 года официально называется Diagrams.net) — бесплатный онлайн-сервис для создания диаграмм, блок-схем и визуальных схем любой сложности.

4.1 Работа в Draw.io

1. Перешли на сайт https://app.diagrams.net/.

2. Создали новую диаграмму ("File" -> "New").

3. В левой панели выбрали раздел "Network". Перетащили на холм иконки, соответствующие компонентам IoT-системы:

– Дачик (Sensor) для робота-датчика.

– Компьютер (Computer) или Сервер (Server) для Fog-обработчика и ноутбука-сервера.

– Мобильное устройство (Mobile) для смартфона.

– Из раздела "General" добавили стрелки и блоки для соединений и подписей.

4. Соединили компоненты линиями, имитируя поток данных от датчика к серверу. Подписали каждый компонент и линию связи (протокол).

На рисунке 13 представлена созданная диаграмма.

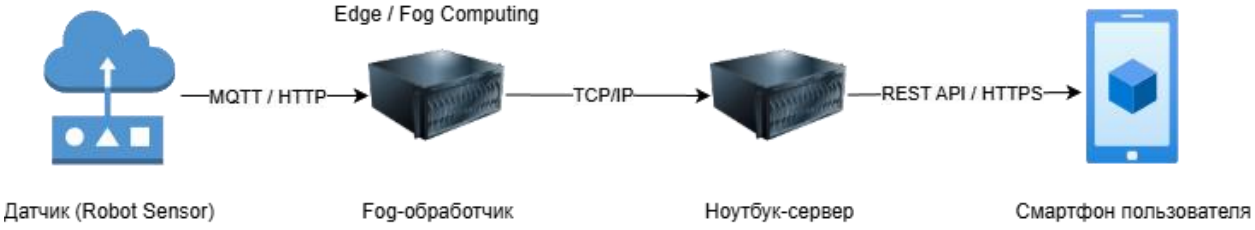



Рисунок 13 – созданная диаграмма

4.2 Задание для построения схемы через Draw.io

Повторили в Draw.io логическую модель системы (аналог Рисунка 2 из данного документа), используя базовые фигуры (прямоугольники, стрелки) из раздела "General". Схема наглядно показывает пять основных

компонентов системы и направление потока данных между ними. Сохранили диаграмму в формате PNG.

На рисунке 14 представлена логическая модель системы.

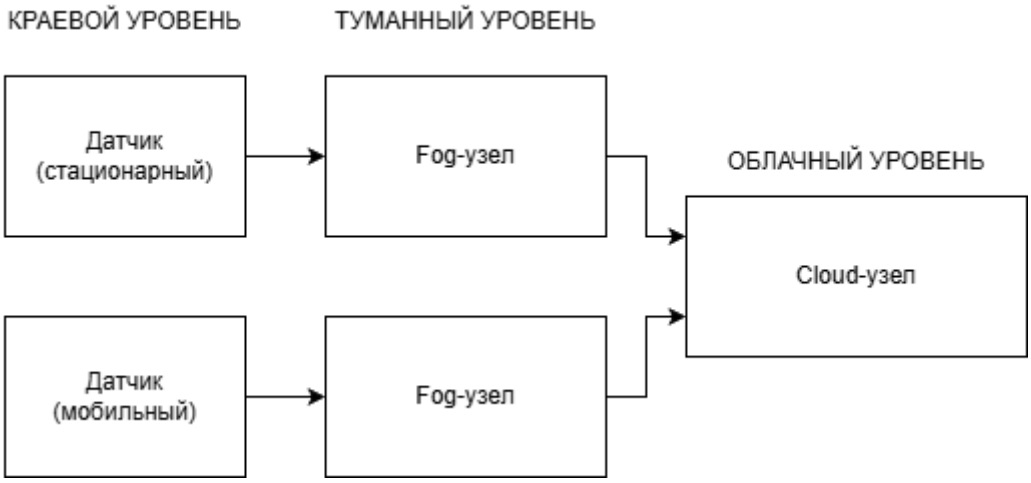



Рисунок 14 – Логическая модель системы

## Задание 5

Индивидуальное задание.

Используя один из изученных инструментов – PlantUML, построила две диаграммы для распределенной IoT-системы, соответствующей моему варианту №9.

За основу взяла трехуровневую архитектуру «Край-Туман-Облако».

| **Вариант** | **Краевые устройства (Edge)** | **Fog-узлы** | **Облачные серверы (Cloud)** | **Характеристика системы** |
|---|---|---|---|---|
| 9 | 6 | 3 | 1 | Умеренная нагрузка |

1. Deployment-диаграмма (структурная схема), отображающая физические узлы системы (краевые устройства, Fog-узлы, облачные серверы) и связи между ними.

Листинг:

```
@startuml
skinparam backgroundColor #F8F8F8
skinparam shadowing false
skinparam nodesep 40
skinparam ranksep 60
skinparam defaultFontName Arial
skinparam stereotypeCBackgroundColor #ADD1B2
skinparam stereotypeIBackgroundColor #ADD1B2

title Вариант 9: Deployment-диаграмма распределённой IoT-системы\n(6 Edge, 3 Fog, 1 Cloud, умеренная нагрузка)

package "Edge (Краевой уровень) - 6 устройств" #LightBlue {
  [Датчик 1 (камера)] as e1
  [Датчик 2 (кнопка)] as e2
  [Датчик 3 (термостат)] as e3
  [Датчик 4 (аналитика)] as e4
  [Датчик 5 (промышленный)] as e5
  [Датчик 6 (события)] as e6
}

package "Fog (Туманный уровень) - 3 узла" #LightGreen {
  [Fog-узел 1] as f1
  [Fog-узел 2] as f2
  [Fog-узел 3] as f3
}
```

```
package "Cloud (Облачный уровень)" #LightGray {
  [Ноутбук-сервер (аналитика и хранение)] as c
}

[Робот-курьер\n(транспортировка данных)] as courier
[Смартфон\n(буферизация)] as phone

e1 --> f1 : ZigBee
e2 --> f1 : Wi-Fi
e3 --> f2 : Bluetooth
e4 --> f2 : ZigBee
e5 --> f3 : Wi-Fi
e6 --> f3 : Ethernet

f1 --> courier : Ethernet
f2 --> courier : Ethernet
f3 --> courier : Ethernet

courier --> phone : Bluetooth
phone --> c : Интернет

note bottom of f1
  Локальная обработка
  и фильтрация данных
end note

note bottom of courier
  Транспортный робот
  (передача между уровнями)
end note

note bottom of c
  Глубокая аналитика
  и долговременное хранение
end note

@enduml
```

На рисунке 15 представлена реализация кода. На рисунке 16 изображена Deployment-диаграмма.

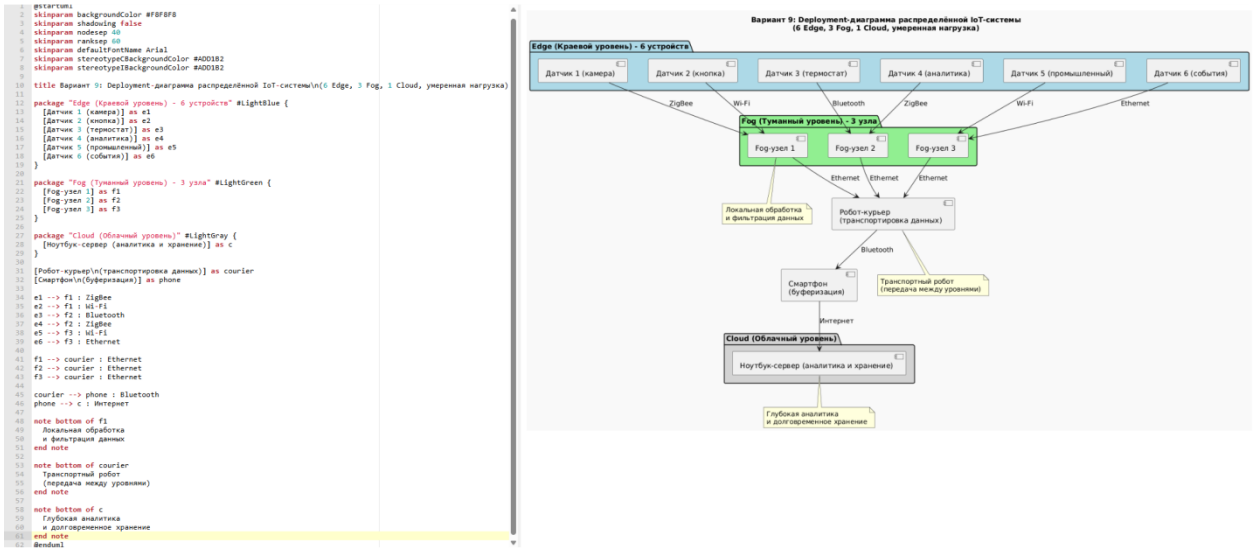



Рисунок 15 – Реализация кода

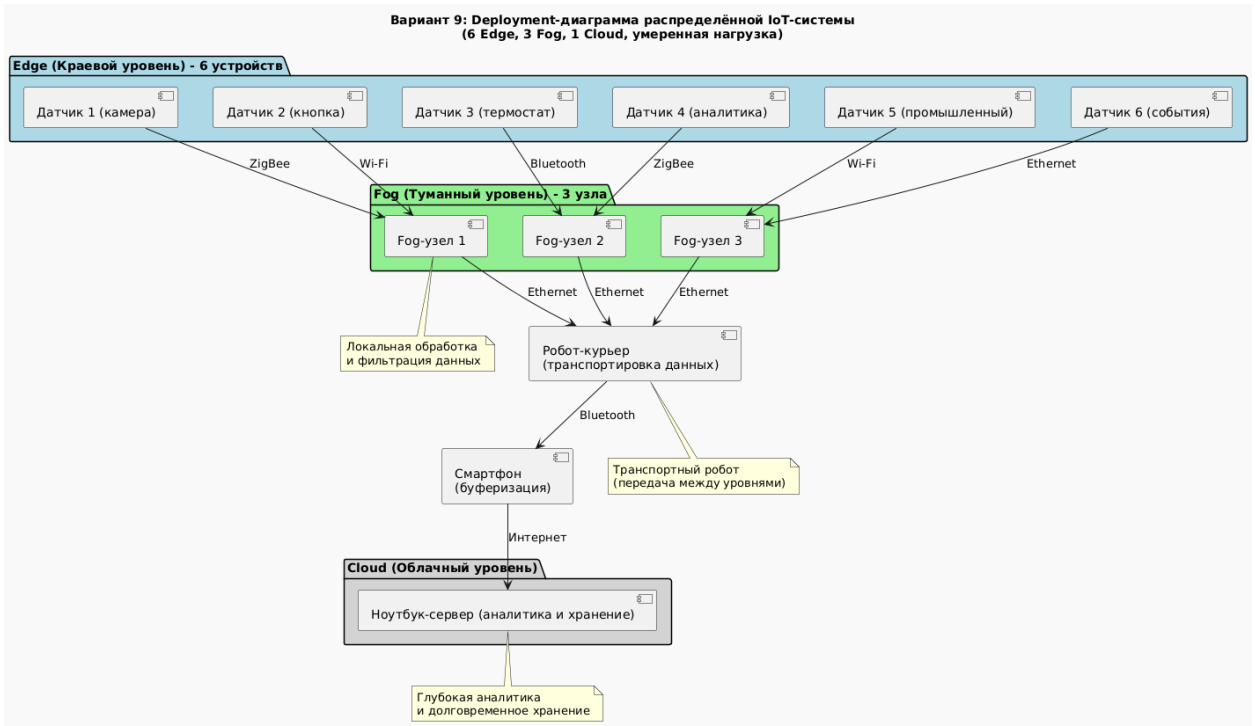



Рисунок 16 – Deployment-диаграмма

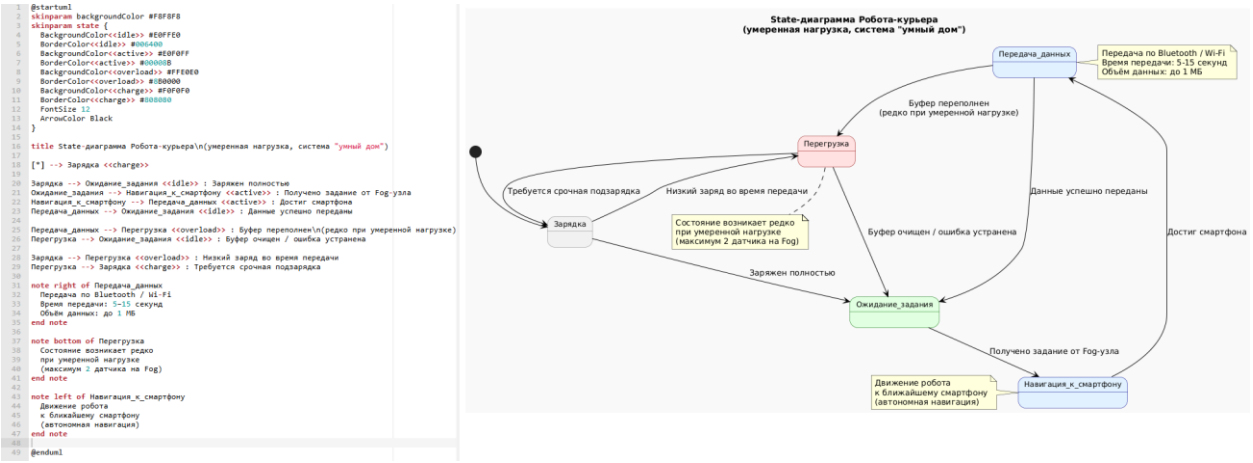
2. State-диаграмму (функциональная схема состояний), описывающую изменение состояния одного из Fog-узлов (например: Режим ожидания -> Получение задачи -> Обработка -> Передача результата -> Перегрузка/Ожидание).

Листинг:

```
@startuml
skinparam backgroundColor #F8F8F8
skinparam state {
  BackgroundColor<<idle>> #E0FFE0
  BorderColor<<idle>> #006400
  BackgroundColor<<active>> #E0F0FF
```

```
  BorderColor<<active>> #00008B
  BackgroundColor<<overload>> #FFE0E0
  BorderColor<<overload>> #8B0000
  BackgroundColor<<charge>> #F0F0F0
  BorderColor<<charge>> #808080
  FontSize 12
  ArrowColor Black
}

title State-диаграмма Робота-курьера\n(умеренная нагрузка,
система "умный дом")

[*] --> Зарядка <<charge>>

Зарядка --> Ожидание_задания <<idle>> : Заряжен полностью
Ожидание_задания --> Навигация_к_смартфону <<active>> : Получено
задание от Fog-узла
Навигация_к_смартфону --> Передача_данных <<active>> : Достиг
смартфона
Передача_данных --> Ожидание_задания <<idle>> : Данные успешно
переданы

Передача_данных --> Перегрузка <<overload>> : Буфер
переполнен\n(редко при умеренной нагрузке)
Перегрузка --> Ожидание_задания <<idle>> : Буфер очищен / ошибка
устранена

Зарядка --> Перегрузка <<overload>> : Низкий заряд во время
передачи
Перегрузка --> Зарядка <<charge>> : Требуется срочная подзарядка

note right of Передача_данных
  Передача по Bluetooth / Wi-Fi
  Время передачи: 5–15 секунд
  Объём данных: до 1 МБ
end note

note bottom of Перегрузка
  Состояние возникает редко
  при умеренной нагрузке
  (максимум 2 датчика на Fog)
end note

note left of Навигация_к_смартфону
  Движение робота
  к ближайшему смартфону
  (автономная навигация)
end note

@enduml
```

На рисунке 17 представлена реализация кода. На рисунке 18 изображена State-диаграмма.



```
@startuml
skinparam backgroundColor #F8F8F8
skinparam state {
  BackgroundColor<<idle>> #E0FFE0
  BorderColor<<idle>> #006400
  BackgroundColor<<active>> #E0F0FF
  BorderColor<<active>> #000088
  BackgroundColor<<overload>> #FFE0E0
  BorderColor<<overload>> #880000
  BackgroundColor<<charge>> #F0F0F0
  BorderColor<<charge>> #808080
  FontSize 12
  ArrowColor Black
}

title State-диаграмма Робота-курьера\n(умеренная нагрузка, система "умный дом")

[*] --> Зарядка <<charge>>

Зарядка --> Ожидание_задания <<idle>> : Заряжен полностью
Ожидание_задания --> Навигация_к_смартфону <<active>> : Получено задание от Fog-узла
Навигация_к_смартфону --> Передача_данных <<active>> : Достиг смартфона
Передача_данных --> Ожидание_задания <<idle>> : Данные успешно переданы

Передача_данных --> Перегрузка <<overload>> : Буфер переполнен\n(редко при умеренной нагрузке)
Перегрузка --> Ожидание_задания <<idle>> : Буфер очищен / ошибка устранена

Зарядка --> Перегрузка <<overload>> : Низкий заряд во время передачи
Перегрузка --> Зарядка <<charge>> : Требуется срочная подзарядка

note right of Передача_данных
  Передача по Bluetooth / Wi-Fi
  Время передачи: 5–15 секунд
  Объём данных: до 1 МБ
end note

note bottom of Перегрузка
  Состояние возникает редко
  при умеренной нагрузке
  (максимум 2 датчика на Fog)
end note

note left of Навигация_к_смартфону
  Движение робота
  к ближайшему смартфону
  (автономная навигация)
end note

@enduml
```




Рисунок 17 – Реализация кода

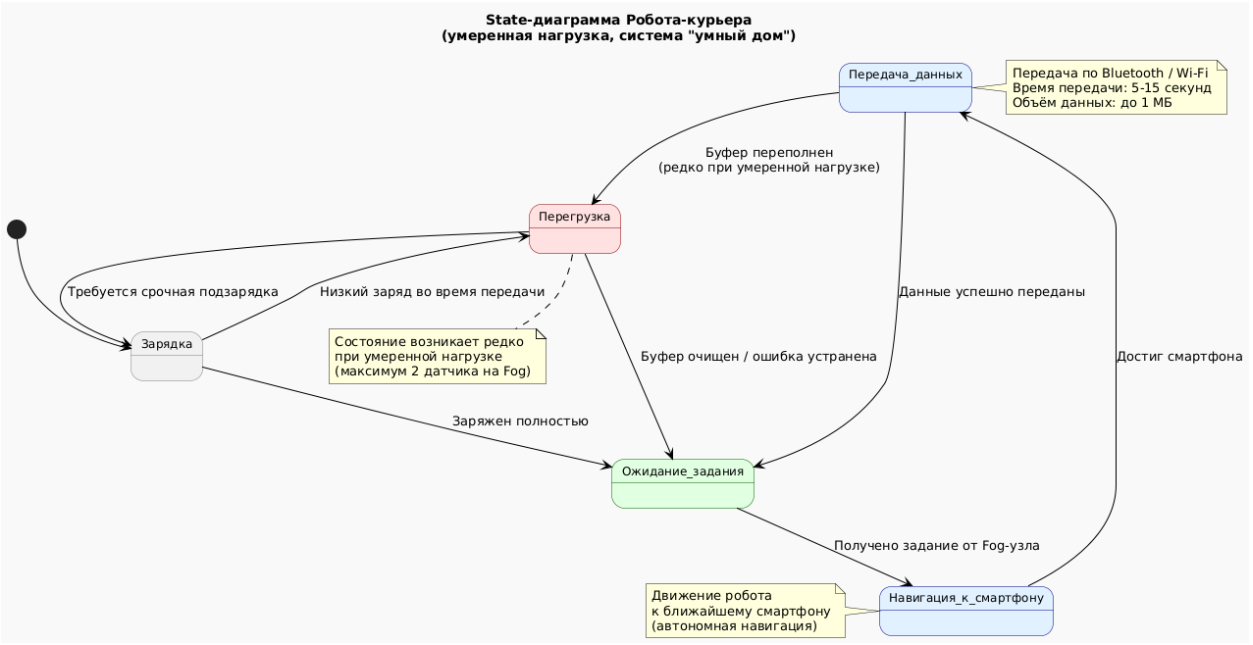



Рисунок 18 - State-диаграмма

За основу взяла трехуровневую архитектуру «Край-Туман-Облако».

# Заключение

В ходе выполнения лабораторной работы были достигнуты все поставленные цели и задачи. Выполненная работа позволила изучить принципы построения схем распределительных систем IoT в соответствии со стандартами ГОСТ, освоить инструменты Google Colab, PlantUML Editor и draw.io для визуализации архитектуры и приобрести навык создания структурных и функциональных диаграмм (Deployment и State) для IoT-системы.

В процессе работы были разработаны:

– Deployment-диаграммы, отражающие физическую структуру системы (узлы Edge, Fog, Cloud и связи между ними).

– State-диаграммы, моделирующие поведение компонентов (например, робота-курьера и Fog-обработчика) в ответ на события и изменения состояний.

Особое внимание уделено соответствию диаграмм стандартам ГОСТ 2.701 и 2.702, а также их адаптации к современным нотациям UML/SysML. В индивидуальном задании была реализована трехуровневая архитектура «Край–Туман–Облако» для системы с умеренной нагрузкой, включающая 6 краевых устройств, 3 Fog-узла и 1 облачный сервер.

Таким образом, в ходе выполнения работы получены практические компетенции, необходимые для проектирования и документирования распределённых IoT-систем с использованием современных инструментов и стандартов.