МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра информационных систем

Умная кормушка для домашних питомцев с веб-интерфейсом

Курсовая работа по дисциплине «Технологии интернет вещей»

09.03.02 Информационные системы и технологии

Профиль «Встраиваемые вычислительные системы и интернет вещей»

Зав. кафедрой _____ Борисов Д. Н. к.т.н., доцент __.__.2025

Обучающийся _____ Гамидов А. А.

Руководитель _____ Максимов А.В. ст. преподаватель

Воронеж 2025

Содержание

1 Введение

1.1 Описание проблемы

Для владельцев домашних животных, особенно тех, кто много времени проводит вне дома, обеспечение регулярного кормления питомца является важной задачей. Традиционные методы, такие как ручное кормление, не всегда подходят в условиях плотного графика или длительного отсутствия. Возникает необходимость в разработке устройства, которое обеспечит автоматическую выдачу корма, мониторинг количества пищи и возможность управления процессом через веб-интерфейс.

1.2 Цель

Разработать и реализовать умную кормушку для домашних животных, которая:

1. Автоматизирует процесс кормления;
2. Предоставляет пользователю возможность удалённого управления через веб-интерфейс;
3. Отслеживает количество корма с использованием датчиков;
4. Использует доступные и недорогие комплектующие.

2 Сравнительный анализ решений

2.1 Анализ современных устройств

Автоматизированные кормушки для животных являются популярным направлением в области "умного дома". Современные устройства обычно предлагают следующие функции:

- Автоматическая подача корма: Используются шаговые или серводвигатели для точного дозирования пищи.
- Мониторинг количества корма: Устройства оснащены датчиками веса (например, на основе HX711), которые позволяют отслеживать остаток корма в контейнере.
- Наблюдение за питомцем: Некоторые модели имеют встроенные камеры, что позволяет владельцу удалённо наблюдать за животным через мобильное приложение.
- Удалённое управление: Через Wi-Fi или Bluetooth пользователь может настраивать время кормления, порции и получать уведомления о состоянии устройства.
- Интеграция с экосистемой умного дома: Некоторые устройства поддерживают голосовые помощники, такие как Alexa или Google Assistant.

2.2 Сравнение коммерческих решений

PetSafe Smart Feed

- Функции: Автоматическая подача корма, управление через приложение, контроль порций.
- Недостатки: Высокая стоимость, отсутствие открытого доступа к программному обеспечению, что ограничивает возможности для модификации.

Xiaomi Smart Pet Feeder

- Функции: Элегантный дизайн, подключение к экосистеме Xiaomi, встроенные датчики веса.
- Недостатки: Зависимость от экосистемы Xiaomi, невозможность интеграции с другими платформами.

Локальные разработки

На рынке также присутствуют менее известные решения, которые зачастую имеют ограниченные функции и недостаточный уровень надёжности. Они часто не поддерживают удалённое управление или не имеют достаточной документации для самостоятельной настройки и ремонта.

2.3 Проблемы проприетарных решений

- Высокая стоимость: Большинство коммерческих кормушек стоят от 10 000 рублей и выше, что делает их недоступными для многих пользователей.
- Закрытость ПО: Проприетарное программное обеспечение ограничивает возможности модификации устройства и добавления новых функций.
- Ограниченная гибкость: Такие устройства обычно не позволяют пользователю интегрировать дополнительные датчики или настроить специфические сценарии работы.

2.4 Перспективы разработки DIY-решений

Самостоятельно разработанные устройства (DIY – Do It Yourself) обладают рядом преимуществ:

- Гибкость: Возможность добавить любые необходимые функции и интегрировать устройство в существующую систему умного дома.
- Экономия: Стоимость комплектующих для базовой модели умной кормушки значительно ниже стоимости коммерческих аналогов.
- Обучение: Процесс создания устройства позволяет освоить навыки программирования, электроники и работы с микроконтроллерами.

3 Описание устройства

3.1 Общая структура системы:

- **Отладочная плата на базе ESP32: ESP32 Wroom-32 – 593р**
- **Более предпочтительно с камерой: ESP32-CAM с камерой – 992р**
- **Драйвер мотора: Драйвер двигателя MX1508 – 198р**
- **Шаговый двигатель:** Nema17 17HS4023 – **856р**
- **Тензодатчик:** ARDUINO тензодатчик – **296р**
- **Датчик веса:** HX711 – **187р**
- **Набор проводов для соединений**
- **Макетная плата:** 50x70 – **42р**
- **Набор разъемов JST XH2.54мм: JST XH2.54мм – дома найдутся за бесплатно**
- **Блок питания: 5V 3A – 558р**

Итоговый набор модулей и электронных компонент может варьироваться и может предполагать несколько реализаций.

3.2 Описание компонентов устройства

- **Контроллер ESP32 Wroom-32 или ESP32-CAM.**
  - Основные функции:
  - Управление датчиками и двигателем;
  - Организация связи с веб-интерфейсом через Wi-Fi;
  - (Для ESP32-CAM) передача видео с камеры.

- **Драйвер мотора MX1508:**
  - Обеспечивает управление шаговым двигателем;
  - Простота подключения и настройки.

- **Шаговый двигатель Nema17 17HS4023:**
  - Высокая точность работы;
  - Способен обеспечить равномерную подачу корма.

- **Датчик веса HX711:**
  - Анализирует вес корма в контейнере;
  - Обеспечивает данные для контроля запаса.

- **Тензодатчик 20 кг:**
  - Высокая точность измерения;
  - Простота интеграции с контроллером через HX711.

- **Макетная плата и провода**
  - Используются для соединения компонентов.

- **Питание**
  - Блок питания на 5V 3A обеспечивает стабильную работу всех компонентов.

4 Схематическое изображение устройства

4.1 Схемы устройства

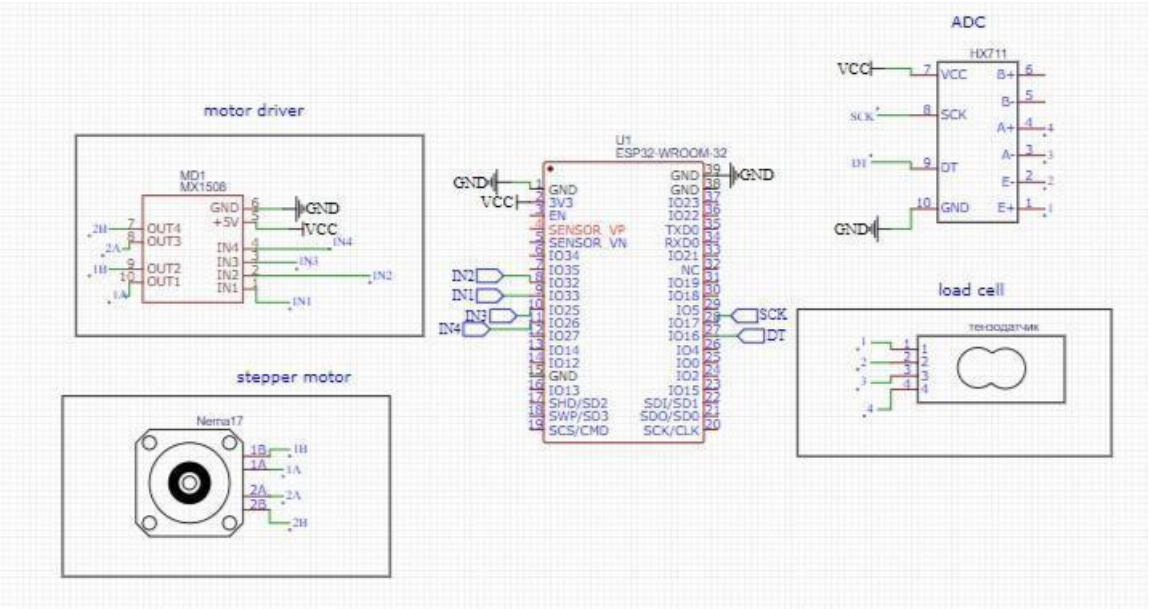

Рисунок 1 - схема устройства

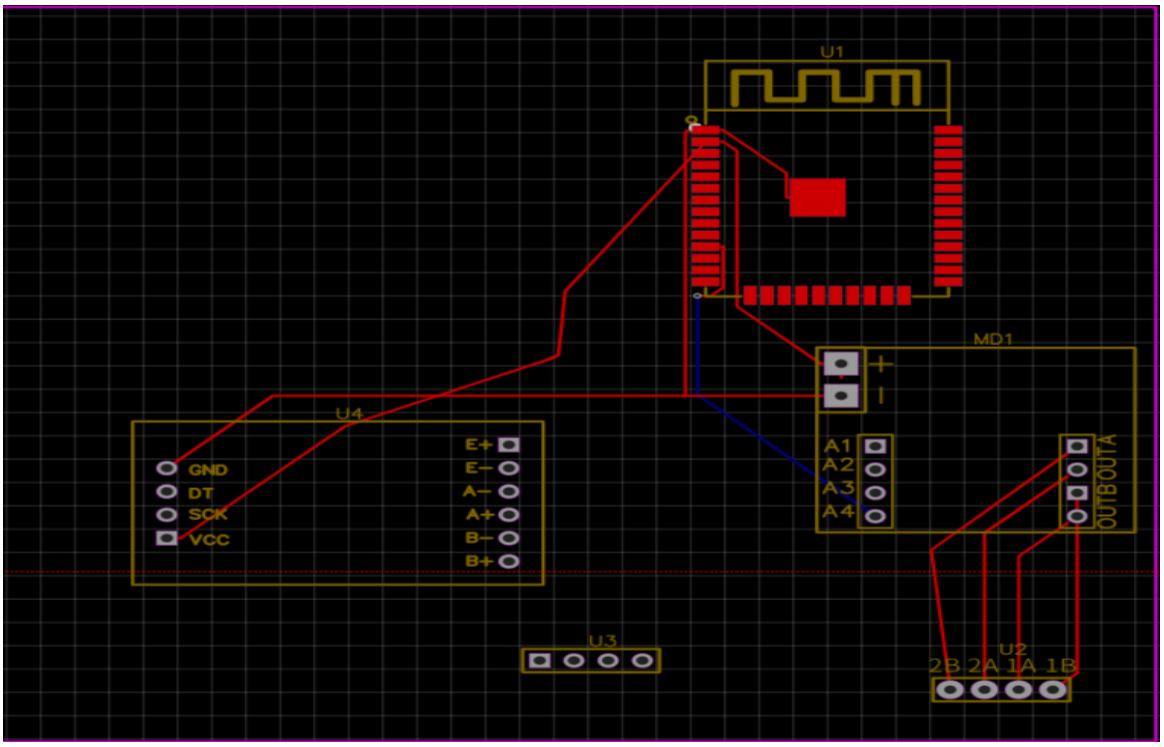

Рисунок 2 - разводка платы

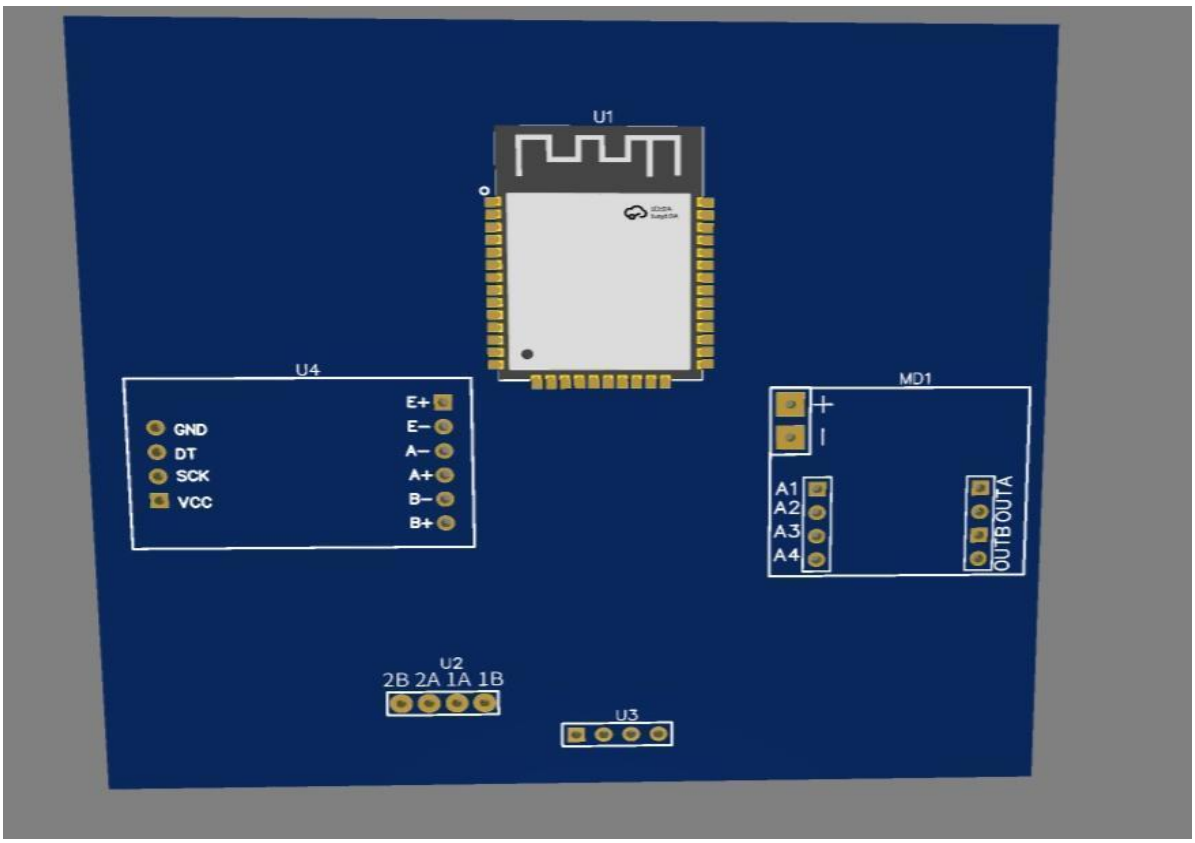

Рисунок 3 - 3D отображение(вид сверху)

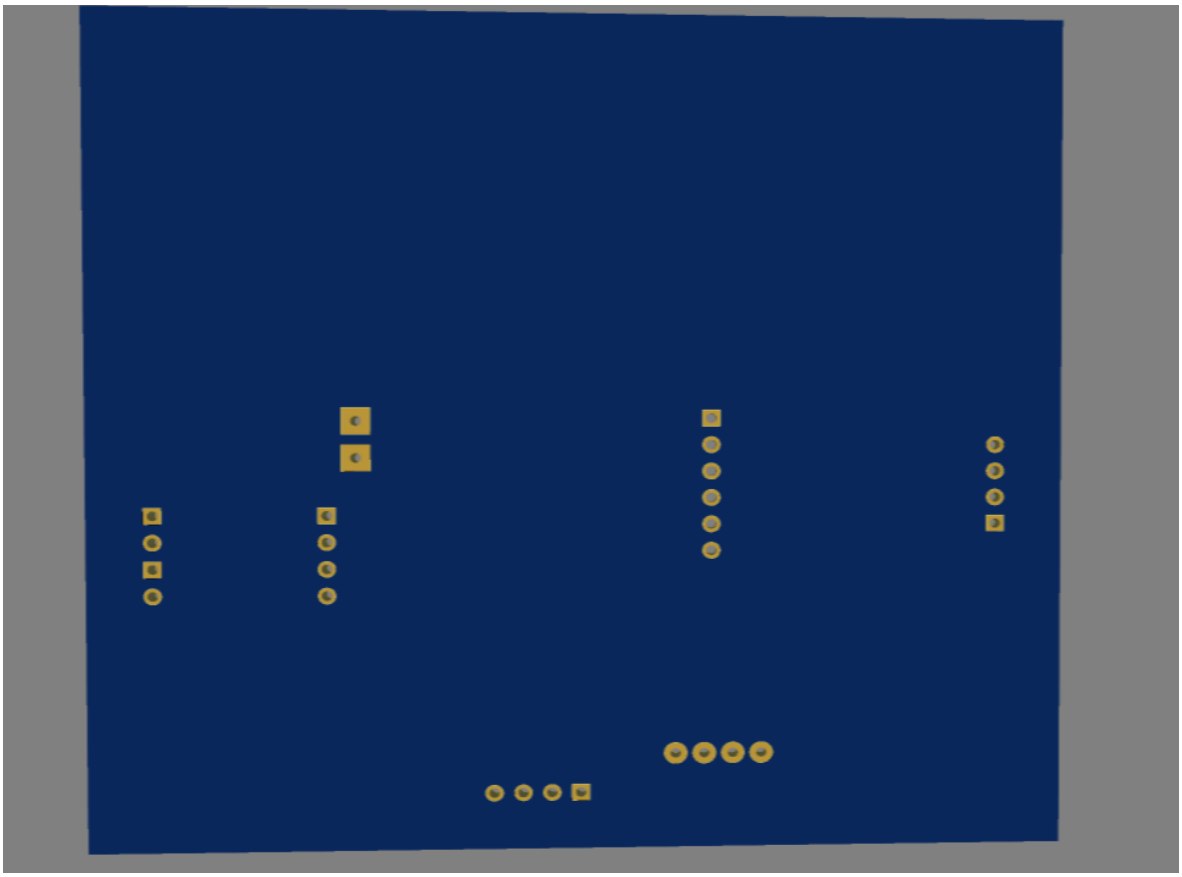

Рисунок 3 - 3D отображение(вид снизу)

4.2 Подключения компонентов

Схемы подключения(примерные):

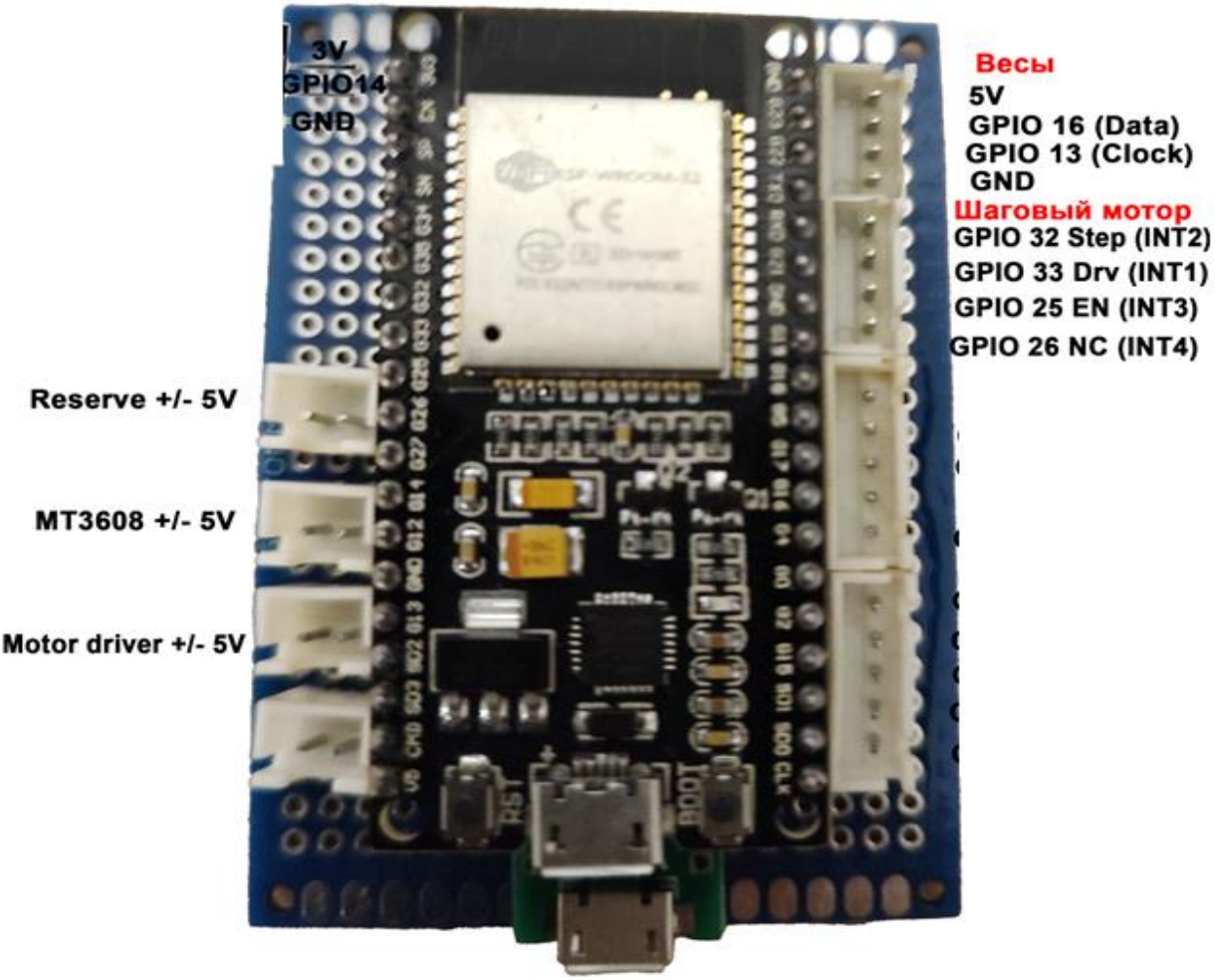

Рисунок 4 - Схемы подключения

Подключение драйвера MX1508:

Общая схема подключения к драйверу и микроконтроллеру:

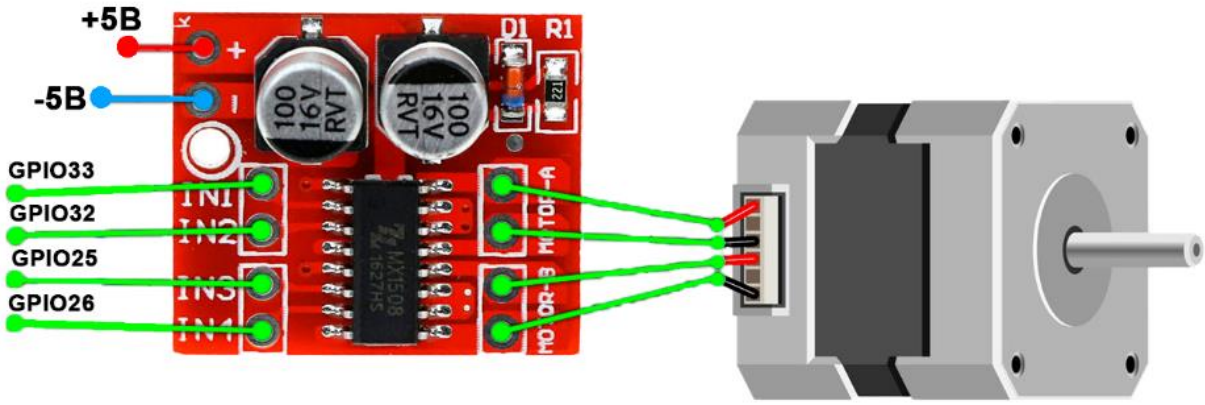

Рисунок 5 - Подключение драйвера MX1508:

Подключение весов:

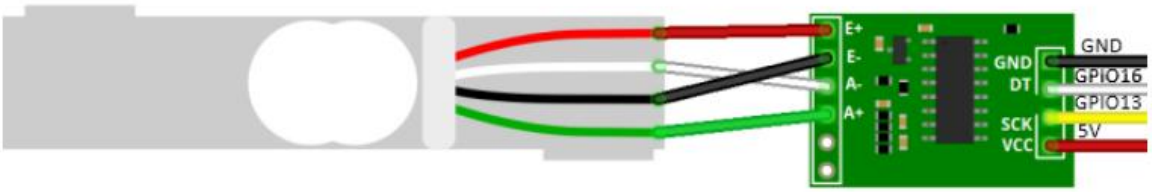

Рисунок 6 - Подключение весов

5 Диаграммы

Диаграмма последовательностей (Sequence Diagram)

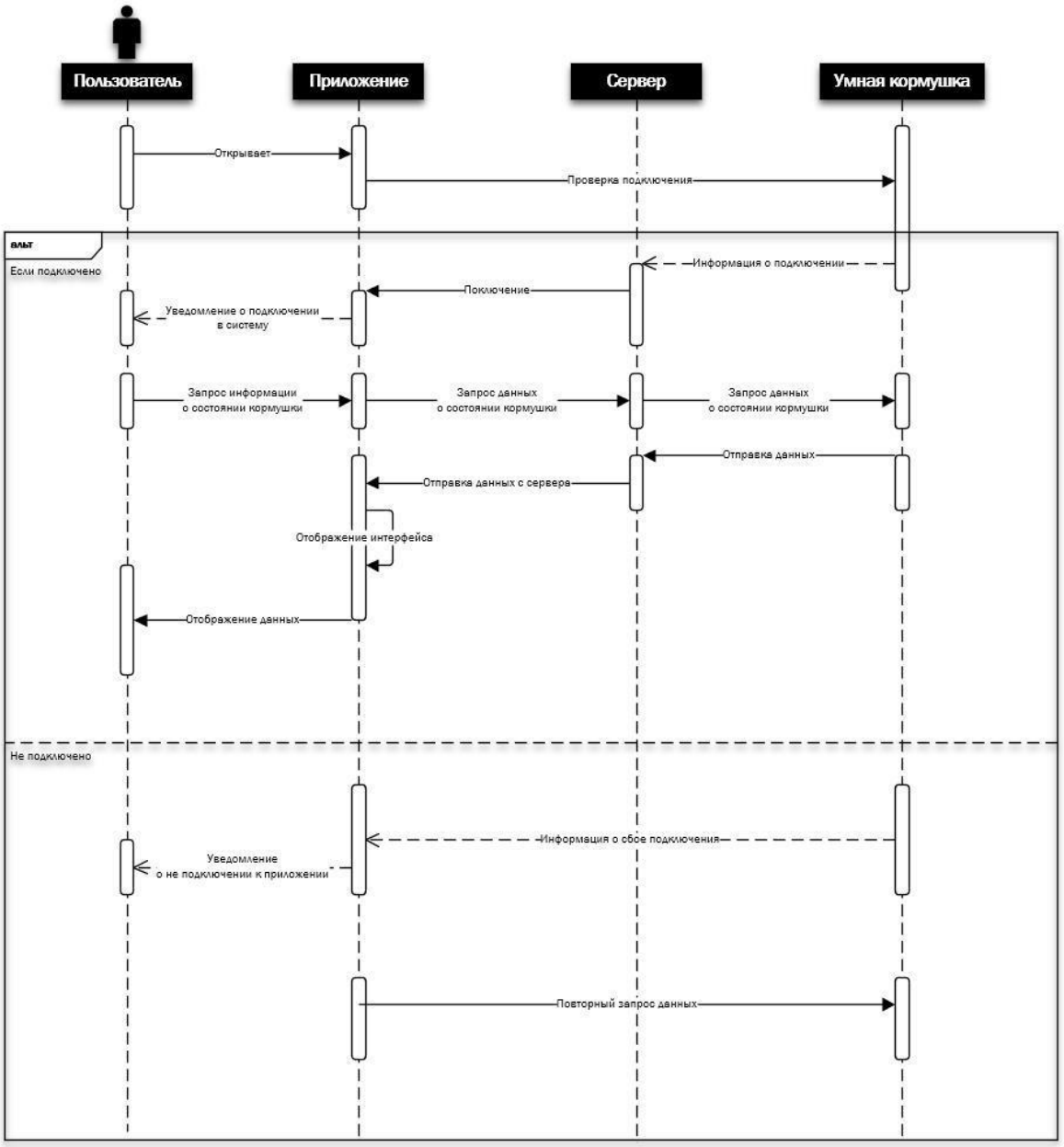

Диаграмма UML

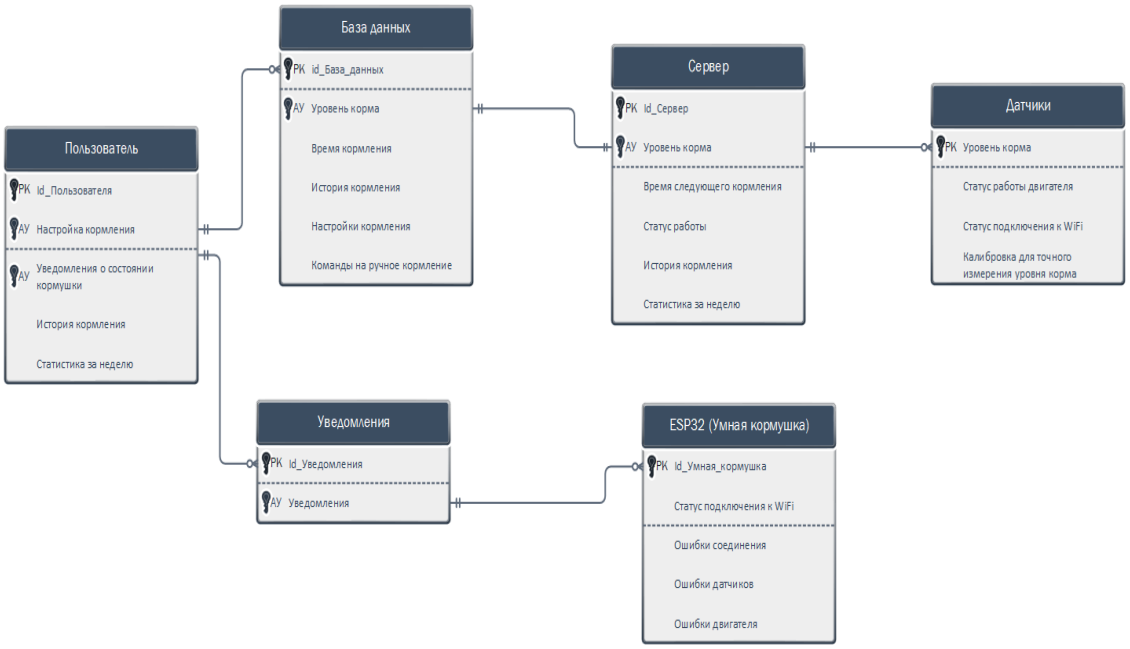

Диаграмма вариантов использования (Use Case)

Прецеденты (Use Cases):

Регистрация: Пользователь создаёт учётную запись в системе для доступа к функциям кормушки.

Управление кормлением: Пользователь настраивает время кормления, порции и режимы работы кормушки.

Наблюдение за питомцем: Пользователь просматривает видео с камеры кормушки (если она интегрирована).

Просмотр состояния кормушки: Пользователь проверяет уровень корма, статус подключения и историю кормлений.

Получение уведомлений: Система отправляет пользователю уведомления (низкий уровень корма, ошибки).

Подключение устройства:
Пользователь подключает кормушку к Wi-Fi и настраивает связь с приложением.

6 Код для подключения датчиков к микроконтроллеру

- Подключение библиотек:

```
#include <WiFi.h>
#include <WebServer.h>
#include <HX711.h>
#include <ESP32Servo.h>
```

- Конфигурация сети

```
// Конфигурация Wi-Fi
const char* ssid = "Name";
const char* password = "Password";
```

- Управление шаговым двигателем

```
const int motorPin1 = 12;  // IN1 драйвера MX1508
const int motorPin2 = 14;  // IN2 драйвера MX1508
const int feedSteps = 100; // Шагов для одной порции

void feedPet() {
  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);
  delay(5);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);
  delay(5);
}
```

- Датчик веса(HX711+тензодатчик)

```
HX711 scale;
scale.begin(16, 4);
scale.set_scale(2280.f);  // Калибровочный коэффициент
scale.tare();

float currentWeight = scale.get_units(5);
```

- Калибровка датчика веса

```
void calibrateScale(float knownWeight) {
  scale.tare();
  float reading = scale.get_units(10);
  float newScale = reading / knownWeight;
  scale.set_scale(newScale);
}
```

- Веб-интерфейс(Автоматическое обновление данных без перезагрузки страницы)

```
function updateStatus() {
  fetch('/status')
    .then(response => response.json())
    .then(data => {
      // Обновление уровня корма
      document.getElementById('food-level-value').textContent = data.weight + '
kg';
      document.getElementById('food-level-bar').style.width =
        Math.min(100, Math.max(0, data.weight * 10)) + '%';
```

```
        // Обновление статуса
        const statusElement = document.getElementById('device-status');
        statusElement.textContent = data.status;
        statusElement.style.color = data.status === 'feeding' ? '#4fc3f7' :
'#2ecc71';
    });
}

setInterval(updateStatus, 5000);
```

7 Заключение

В ходе выполнения курсовой работы была разработана умная кормушка для домашних питомцев с возможностью управления через веб-интерфейс. Устройство позволяет автоматически подавать корм в заданное время и контролировать его количество с помощью датчиков веса. Также реализована система уведомлений и визуализации данных в реальном времени.

Основное внимание было уделено доступности компонентов, простоте конструкции и гибкости программного обеспечения, что делает проект удобным для самостоятельной сборки и доработки. Использование микроконтроллера ESP32 обеспечило стабильную работу устройства и возможность беспроводного соединения по Wi-Fi, а интерфейс, реализованный средствами HTML и JavaScript, позволил обеспечить комфортное взаимодействие пользователя с системой.

Разработка данного устройства позволяет решить проблему регулярного кормления домашних животных при отсутствии хозяина и способствует повышению уровня автоматизации в быту. В перспективе проект может быть расширен функциями видеонаблюдения, интеграцией с голосовыми помощниками и мобильными приложениями.

Таким образом, поставленные цели и задачи были успешно реализованы, а полученные результаты подтвердили работоспособность и практическую значимость предложенного решения.

8 Приложение

```
#include <WiFi.h>
#include <WebServer.h>
#include <HX711.h>
#include <ESP32Servo.h>

const char* ssid = "Name";
const char* password = "Password";

const int motorPin1 = 12;
const int motorPin2 = 14;
const int stepsPerRevolution = 200;
const int feedSteps = 100;

// Датчик веса HX711
const int LOADCELL_DOUT_PIN = 16;
const int LOADCELL_SCK_PIN = 4;
HX711 scale;

// Веб-сервер
WebServer server(80);

// Переменные состояния
float currentWeight = 0;
bool feedingInProgress = false;
String lastFeedingTime = "Never";
int portionSize = 1; // Размер порции (1-5)

void setup() {
  Serial.begin(115200);

  // Инициализация пинов
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);

  // Инициализация датчика веса
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
  scale.set_scale(2280.f);
  scale.tare();

  // Подключение к Wi-Fi
  WiFi.begin(ssid, password);
  Serial.println("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi. IP address: ");
  Serial.println(WiFi.localIP());

  // Настройка веб-сервера
  server.on("/", handleRoot);
  server.on("/feed", handleFeed);
  server.on("/status", handleStatus);
  server.on("/calibrate", handleCalibrate);
  server.on("/setportion", handleSetPortion);
  server.on("/style.css", handleCSS);
  server.begin();
}

void loop() {
```

```
  server.handleClient();
  static unsigned long lastWeightUpdate = 0;
  if (millis() - lastWeightUpdate > 5000) {
    updateWeight();
    lastWeightUpdate = millis();
  }
}

// Функция кормления
void feedPet(int portions = 1) {
  if (feedingInProgress) return;

  feedingInProgress = true;
  Serial.println("Feeding started");

  for (int i = 0; i < portions; i++) {
    for (int j = 0; j < feedSteps; j++) {
      digitalWrite(motorPin1, HIGH);
      digitalWrite(motorPin2, LOW);
      delay(5);
      digitalWrite(motorPin1, LOW);
      digitalWrite(motorPin2, HIGH);
      delay(5);
    }
    delay(500);
  }

  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);

  lastFeedingTime = getTimeString();
  updateWeight();
  feedingInProgress = false;
  Serial.println("Feeding completed");
}

void updateWeight() {
  if (scale.is_ready()) {
    currentWeight = scale.get_units(5);
  }
}

String getTimeString() {
  unsigned long seconds = millis() / 1000;
  unsigned long minutes = seconds / 60;
  unsigned long hours = minutes / 60;
  return String(hours) + "h " + String(minutes % 60) + "m " + String(seconds %
60) + "s ago";
}

// Обработчики веб-запросов
void handleRoot() {
  String html = R"=====(
<!DOCTYPE html>
<html>
<head>
  <title>Smart Pet Feeder</title>
  <link rel="stylesheet" href="/style.css">
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <div class="container">
    <h1>Smart Pet Feeder</h1>
```

```
    <div class="status-card">
      <h2>Current Status</h2>
      <div class="status-item">
        <span class="label">Food Level:</span>
        <span class="value">)=====" + String(currentWeight, 2) + R"=====(
kg</span>
      </div>
      <div class="status-item">
        <span class="label">Last Feeding:</span>
        <span class="value">)=====" + lastFeedingTime + R"=====(</span>
      </div>
      <div class="status-item">
        <span class="label">Portion Size:</span>
        <span class="value">)=====" + String(portionSize) + R"=====(</span>
      </div>
    </div>

    <div class="control-panel">
      <h2>Controls</h2>
      <div class="button-group">
        <a href="/feed" class="btn feed-btn">Feed Now</a>
      </div>

      <form action="/setportion" method="get" class="portion-form">
        <label for="portion">Set Portion Size (1-5):</label>
        <input type="number" id="portion" name="size" min="1" max="5"
value=")=====" + String(portionSize) + R"=====(">
        <button type="submit" class="btn">Set</button>
      </form>

      <form action="/calibrate" method="get" class="calibrate-form">
        <label>Calibrate Scale:</label>
        <input type="text" name="weight" placeholder="Known weight (kg)">
        <button type="submit" class="btn calibrate-btn">Calibrate</button>
      </form>
    </div>
  </div>
</body>
</html>
)=====";

  server.send(200, "text/html", html);
}

void handleCSS() {
  String css = R"=====(
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: #f5f5f5;
  margin: 0;
  padding: 20px;
  color: #333;
}

.container {
  max-width: 800px;
  margin: 0 auto;
  background-color: white;
  border-radius: 10px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
  padding: 20px;
}
```

```
h1 {
  color: #2c3e50;
  text-align: center;
  margin-bottom: 30px;
}

.status-card {
  background-color: #f8f9fa;
  border-radius: 8px;
  padding: 20px;
  margin-bottom: 20px;
}

.status-item {
  display: flex;
  justify-content: space-between;
  margin-bottom: 10px;
  padding: 8px 0;
  border-bottom: 1px solid #eee;
}

.label {
  font-weight: bold;
  color: #555;
}

.value {
  color: #2c3e50;
}

.control-panel {
  background-color: #f8f9fa;
  border-radius: 8px;
  padding: 20px;
}

.button-group {
  margin-bottom: 20px;
  text-align: center;
}

.btn {
  display: inline-block;
  background-color: #3498db;
  color: white;
  padding: 10px 20px;
  border-radius: 5px;
  text-decoration: none;
  font-weight: bold;
  border: none;
  cursor: pointer;
  transition: background-color 0.3s;
}

.btn:hover {
  background-color: #2980b9;
}

.feed-btn {
  background-color: #2ecc71;
  font-size: 18px;
  padding: 15px 30px;
```

```
}

.feed-btn:hover {
  background-color: #27ae60;
}

.calibrate-btn {
  background-color: #e74c3c;
}

.calibrate-btn:hover {
  background-color: #c0392b;
}

form {
  margin-bottom: 20px;
}

label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
}

input[type="number"],
input[type="text"] {
  width: 100%;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 5px;
  margin-bottom: 10px;
  box-sizing: border-box;
}

@media (max-width: 600px) {
  .container {
    padding: 10px;
  }

  .btn {
    width: 100%;
    box-sizing: border-box;
  }
}
)=====";

  server.send(200, "text/css", css);
}

void handleFeed() {
  feedPet(portionSize);
  server.sendHeader("Location", "/");
  server.send(303);
}

void handleStatus() {
  String json = "{";
  json += "\"weight\":" + String(currentWeight, 2) + ",";
  json += "\"lastFeeding\":\"" + lastFeedingTime + "\",";
  json += "\"portionSize\":" + String(portionSize) + ",";
  json += "\"status\":\"" + (feedingInProgress ? "feeding" : "idle") + "\"";
  json += "}";
```

```
  server.send(200, "application/json", json);
}

void handleCalibrate() {
  if (server.hasArg("weight")) {
    float knownWeight = server.arg("weight").toFloat();
    if (knownWeight > 0) {
      calibrateScale(knownWeight);
    }
  }
  server.sendHeader("Location", "/");
  server.send(303);
}

void handleSetPortion() {
  if (server.hasArg("size")) {
    int newPortion = server.arg("size").toInt();
    if (newPortion >= 1 && newPortion <= 5) {
      portionSize = newPortion;
    }
  }
  server.sendHeader("Location", "/");
  server.send(303);
}

void calibrateScale(float knownWeight) {
  scale.set_scale();
  scale.tare();
  delay(5000);

  float reading = scale.get_units(10);
  float newScale = reading / knownWeight;
  scale.set_scale(newScale);
}
```