## Обзор проекта

Разработанная система представляет собой интеграционное решение, состоящее из аппаратной и программной части, обеспечивающее автоматическое распознавание животного, его взвешивание и передачу информации на удалённый сервер. Архитектура построена с применением недорогих, но надёжных компонентов, таких как Raspberry Pi и Arduino, а также RFID-считывателей и весовых датчиков.

Raspberry Pi выступает как центральный управляющий модуль, координирующий взаимодействие между различными подсистемами: модулем идентификации (RFID), измерительным модулем (весы через HX711 и Arduino), системой опрыскивания и логической обработкой полученных данных. Программное обеспечение разработано на языке Python с использованием модульной архитектуры, что облегчает поддержку, расширение и переносимость системы.

Инженерные решения были приняты с учётом условий сельского хозяйства: нестабильный интернет, грязная среда, необходимость быстрой установки и минимального обслуживания. В системе реализована локальная база данных, механизмы резервирования и повторной отправки информации при восстановлении связи.

Кроме базового функционала, предусмотрена возможность добавления новых модулей: опрыскивание, визуальный мониторинг, голосовое сопровождение — всё это возможно благодаря гибкой архитектуре и централизованной системе конфигурации.

## Функциональные возможности

Работа системы начинается с полной инициализации оборудования и программного окружения. Это включает в себя определение рабочей среды (разработка или продукция), автоматическую установку зависимостей и подготовку логов. Все настройки централизованно управляются через модуль конфигурации, что позволяет легко адаптировать систему под различные сценарии без изменения основного кода.

Как только система активирована, она переходит в режим ожидания животного. Считыватель RFID постоянно опрашивает пространство вокруг антенны, и при появлении метки запускается цикл распознавания и валидации. Используется фильтрация дубликатов и контроль времени, чтобы исключить ложные считывания. В случае считывания нескольких меток одновременно система определяет наиболее стабильную, ориентируясь на частоту появления.

После успешной идентификации животного начинается этап взвешивания. Контроллер Arduino, подключённый к весовому датчику через HX711, непрерывно передаёт данные на Raspberry Pi. Полученные значения проходят статистическую фильтрацию — удаляются выбросы, рассчитываются медианные значения и производится проверка на стабильность массы. Только при выполнении всех условий данные принимаются как достоверные.

В завершение система формирует единый пакет: идентификатор животного, вес, время начала и окончания измерения, метаданные. Этот пакет отправляется на внешний сервер через HTTP-запрос. Если интернет-соединение недоступно, данные сохраняются локально и отправляются позже.

Если в конфигурации включён модуль опрыскивания, то после завершения взвешивания активируется внешний исполнительный механизм — через GPIO выдаётся команда на определённый пин, который управляет реле или насосом. Это позволяет не только взвешивать, но и обрабатывать животных в автоматическом режиме.

В системе реализована подробная система логирования. В лог scales.log пишется общая информация, отладочные сообщения и события, а в errors.log — только критические ошибки. Логи ротацируются ежедневно и хранятся в течение месяца, что удобно для аудита и диагностики.

## Развитие проекта

С момента запуска первой версии было выполнено множество доработок, направленных на повышение стабильности, расширяемости и удобства эксплуатации. Первая ключевая веха — перенос проекта в систему контроля версий GitHub с организацией кода через submodules. Это позволило выделить ядро системы, библиотеку работы с RFID, а также отдельные сервисные модули и тесты в изолированные блоки, что улучшило командную разработку и облегчило деплой.

Под Raspberry Pi с операционной системой Bullseye был разработан универсальный установочный скрипт setup.sh, включающий установку пакетов, настройку автозапуска, системных служб и Wi-Fi. Благодаря этому развертывание системы на новом устройстве занимает считанные минуты.

Дополнительно была реализована серия тестов и отладочных скриптов на Python. Эти файлы позволяют проверять корректность работы отдельных компонентов — от считывателя RFID до системы опрыскивания. Это не только ускоряет разработку, но и даёт возможность быстро диагностировать проблемы на месте.

Особое внимание было уделено повышению надёжности. Например, внедрена калибровка весов с использованием RFID-меток, что позволяет оператору выполнить настройку, просто поднеся заранее запрограммированную метку. Также в систему был добавлен класс MockGPIO, позволяющий запускать код на обычном ноутбуке без доступа к физическому оборудованию.

## Примеры использования

Система активно тестировалась в условиях сельскохозяйственной фермы, где животные становились на весовую платформу, снабжённую RFID-антенной и платформой с весовым модулем. При каждом приходе животного система определяла ID, производила измерение массы и фиксировала событие в базе. На основании этих данных ветеринар мог строить графики динамики веса, выявлять отстающих в росте и корректировать рацион.

В другом варианте реализации модуль опрыскивания использовался для автоматической обработки животных против паразитов. После успешного взвешивания и идентификации в систему загружались параметры обработки, и распыление происходило без участия человека.

Ссылки на приложения

* **Приложение 1. —** Подготовка и установка проекта на raspberry
* **Приложение 2. —** Сервисы и конфигурационный файл
* **Приложение 3. —** Основные модули проекта
* **Приложение 4. —** Тестовые скрипты проекта
* **Приложение 5. —** Принципиальная схема соединения модулей
* **Приложение 6. —** Блок схемы алгоритма проекта
* **Приложение 7.** — Информация об установленных устройствах
* **Приложение 8. —** Расчёт потребляемой мощности

**Приложение 1.**

Подготовка и установка проекта на raspberry

# 1\_test.sh

#!/usr/bin/env bash  
set -euo pipefail  
  
# Пути к монтированным разделам  
BOOT\_DIR="/media/maxat/bootfs"  
ROOTFS\_DIR="/media/maxat/rootfs"  
  
# Проверка точек монтирования  
for d in "$BOOT\_DIR" "$ROOTFS\_DIR"; do  
 if [ ! -d "$d" ]; then  
 echo "Ошибка: $d не смонтирован!" >&2  
 exit 1  
 fi  
done  
  
echo "=== 1. HDMI ==="  
cat <<EOF >> "$BOOT\_DIR/config.txt"  
# Принудительный HDMI  
hdmi\_force\_hotplug=1  
hdmi\_group=1  
hdmi\_mode=16 # 1080p60  
hdmi\_drive=2 # HDMI, не DVI  
disable\_overscan=1 # без чёрных полей  
EOF  
  
echo "=== 2. SSH ==="  
touch "$BOOT\_DIR/ssh"  
  
echo "=== 3. Wi-Fi ==="  
cp "/home/maxat/Projects/Agrarka/scales-installer/raspberry\_settings/wpa\_supplicant.conf" \  
 "$ROOTFS\_DIR/etc/wpa\_supplicant/wpa\_supplicant.conf"  
  
echo "=== 4. setup\_bullseye.sh ==="  
mkdir -p "$ROOTFS\_DIR/home/pi"  
cp "/home/maxat/Projects/Agrarka/scales-installer/raspberry\_settings/setup\_bullseye.sh" \  
 "$ROOTFS\_DIR/home/pi/"  
chmod +x "$ROOTFS\_DIR/home/pi/setup\_bullseye.sh"  
  
# === 5. Локали ===  
# Локали будут настраиваться непосредственно на устройстве через setup\_bullseye.sh  
  
echo "Локали будут сгенерированы на устройстве при первой загрузке через setup\_bullseye.sh."  
  
echo "=== 6. Отключение blanking и включение HDMI ==="  
RCLOCAL="$ROOTFS\_DIR/etc/rc.local"  
if [ ! -f "$RCLOCAL" ]; then  
 cat <<'EOF' > "$RCLOCAL"  
#!/bin/bash  
for tty in /dev/tty{1..6}; do  
 setterm --blank 0 --powersave off --powerdown 0 <"$tty" >"$tty"  
done  
/usr/bin/vcgencmd display\_power 1  
exit 0  
EOF  
 chmod +x "$RCLOCAL"  
 echo "Создан /etc/rc.local."  
else  
 grep -q 'setterm --blank 0' "$RCLOCAL" \  
 || cat <<'EOF' >> "$RCLOCAL"  
  
# --- Блок от first\_settings.sh  
for tty in /dev/tty{1..6}; do  
 setterm --blank 0 --powersave off --powerdown 0 <"$tty" >"$tty"  
done  
/usr/bin/vcgencmd display\_power 1  
# --- Конец блока  
EOF  
 echo "Дополнен /etc/rc.local."  
fi  
  
echo "=== Всё готово ==="

# cmdline.txt

console=tty1 root=PARTUUID=7984cc50-02 rootfstype=ext4 fsck.repair=yes rootwait cfg80211.ieee80211\_regdom=KZ consoleblank=0 vt.global\_cursor\_default=0

# config.txt

# For more options and information see  
# http://rptl.io/configtxt  
# Some settings may impact device functionality. See link above for details  
  
# Uncomment some or all of these to enable the optional hardware interfaces  
#dtparam=i2c\_arm=on  
#dtparam=i2s=on  
#dtparam=spi=on  
  
# Enable audio (loads snd\_bcm2835)  
dtparam=audio=on  
  
# Additional overlays and parameters are documented  
# /boot/firmware/overlays/README  
  
# Automatically load overlays for detected cameras  
camera\_auto\_detect=1  
  
# Automatically load overlays for detected DSI displays  
display\_auto\_detect=1  
  
# Automatically load initramfs files, if found  
auto\_initramfs=1  
  
# Enable DRM VC4 V3D driver  
dtoverlay=vc4-kms-v3d  
max\_framebuffers=2  
  
# Принудительный HDMI  
hdmi\_force\_hotplug=1  
hdmi\_ignore\_edid=0xa5000080 # игнорировать EDID, использовать указанный ниже режим  
hdmi\_group=2 # 2 = DMT (мониторный режим)  
hdmi\_mode=82 # DMT #82 = 1920×1080 60Hz  
hdmi\_drive=2 # звук и питание через HDMI  
disable\_overscan=1  
  
# Don't have the firmware create an initial video= setting in cmdline.txt.  
# Use the kernel's default instead.  
#disable\_fw\_kms\_setup=1  
  
# Disable compensation for displays with overscan  
disable\_overscan=1  
  
# Run as fast as firmware / board allows  
arm\_boost=1  
  
[cm4]  
# Enable host mode on the 2711 built-in XHCI USB controller.  
# This line should be removed if the legacy DWC2 controller is required  
# (e.g. for USB device mode) or if USB support is not required.  
otg\_mode=1  
  
[cm5]  
dtoverlay=dwc2,dr\_mode=host  
  
[all]

# first\_settings.sh

#!/usr/bin/env bash  
set -euo pipefail  
  
# Пути к монтированным разделам  
BOOT\_DIR="/media/maxat/bootfs"  
ROOTFS\_DIR="/media/maxat/rootfs"  
CUSTOM\_CFG="/home/maxat/Projects/Agrarka/scales-installer/raspberry\_settings/config.txt"  
  
# Проверка точек монтирования  
for d in "$BOOT\_DIR" "$ROOTFS\_DIR"; do  
 if [ ! -d "$d" ]; then  
 echo "Ошибка: $d не смонтирован!" >&2  
 exit 1  
 fi  
done  
  
# 1. Копируем config.txt  
echo "=== 1. Copy custom config.txt ==="  
if [ -f "$CUSTOM\_CFG" ]; then  
 cp "$CUSTOM\_CFG" "$BOOT\_DIR/config.txt"  
 echo "Скопирован кастомный config.txt в $BOOT\_DIR."  
else  
 echo "WARNING: $CUSTOM\_CFG не найден. Пропускаем копирование config.txt."  
fi  
  
# 2. Enable SSH  
echo "=== 2. Enable SSH ==="  
touch "$BOOT\_DIR/ssh"  
  
# 3. Отключение мастера и создание пользователя pi  
# echo "=== 3. Создание userconf.txt для отключения мастера настройки ==="  
# USERCONF\_HASH=$(echo 'pi:raspberry' | openssl passwd -6 -stdin)  
# echo "pi:$USERCONF\_HASH" > "$BOOT\_DIR/userconf.txt"  
  
# 4. Настройка автологина  
echo "=== 3. Настройка автологина ==="  
AUTOLOGIN\_DIR="$ROOTFS\_DIR/etc/systemd/system/getty@tty1.service.d"  
mkdir -p "$AUTOLOGIN\_DIR"  
  
cat <<EOF > "$AUTOLOGIN\_DIR/autologin.conf"  
[Service]  
ExecStart=  
ExecStart=-/sbin/agetty --autologin pi --noclear %I \$TERM  
EOF  
  
# 5. Wi-Fi конфигурация  
echo "=== 4. Настройка Wi-Fi ==="  
WPA\_CONF="/home/maxat/Projects/Agrarka/scales-installer/raspberry\_settings/wpa\_supplicant.conf"  
if [ -f "$WPA\_CONF" ]; then  
 cp -v "$WPA\_CONF" "$BOOT\_DIR/wpa\_supplicant.conf"  
 # Копируем также в rootfs для работы после первой загрузки  
 mkdir -p "$ROOTFS\_DIR/etc/wpa\_supplicant"  
 cp -v "$WPA\_CONF" "$ROOTFS\_DIR/etc/wpa\_supplicant/wpa\_supplicant.conf"  
else  
 echo "ERROR: Файл wpa\_supplicant.conf не найден!" >&2  
 exit 1  
fi  
  
# 6. Копирование setup\_bullseye.sh  
echo "=== 6. Копируем setup\_bullseye.sh ==="  
mkdir -p "$ROOTFS\_DIR/home/pi"  
cp "/home/maxat/Projects/Agrarka/scales-installer/raspberry\_settings/setup\_bullseye.sh" \  
 "$ROOTFS\_DIR/home/pi/"  
chmod +x "$ROOTFS\_DIR/home/pi/setup\_bullseye.sh"  
  
# 7. Удаление rc.local, если есть  
echo "=== 7. Удаление rc.local, если есть ==="  
RCLOCAL="$ROOTFS\_DIR/etc/rc.local"  
if [ -f "$RCLOCAL" ]; then  
 echo "=== Найден rc.local — удаляю ==="  
 rm -f "$RCLOCAL"  
else  
 echo "rc.local не найден — пропускаем."  
fi  
  
# 8. Отключение raspi-config.service  
echo "=== 8. Отключение raspi-config.service ==="  
chroot "$ROOTFS\_DIR" systemctl disable raspi-config.service || true  
  
# 9. Добавление systemd-сервиса для разблокировки Wi-Fi  
# echo "=== 9. Добавление unblock-wifi.service ==="  
# cat <<EOF > "$ROOTFS\_DIR/etc/systemd/system/unblock-wifi.service"  
# [Unit]  
# Description=Unblock Wi-Fi at boot  
# After=network-pre.target  
  
# [Service]  
# Type=oneshot  
# ExecStart=/usr/sbin/rfkill unblock wifi  
  
# [Install]  
# WantedBy=multi-user.target  
# EOF  
  
# ln -sf /etc/systemd/system/unblock-wifi.service \  
# "$ROOTFS\_DIR/etc/systemd/system/multi-user.target.wants/unblock-wifi.service"  
  
# 10. Создание сервиса для Wi-Fi  
echo "=== Создание сервиса для Wi-Fi ==="  
cat <<EOF > "$ROOTFS\_DIR/etc/systemd/system/wifi-autoconnect.service"  
[Unit]  
Description=AutoConnect to Wi-Fi  
After=network.target  
Wants=network-online.target  
  
[Service]  
Type=oneshot  
RemainAfterExit=yes  
ExecStart=/bin/sh -c '/sbin/wpa\_supplicant -B -i wlan0 -c /etc/wpa\_supplicant/wpa\_supplicant.conf || true'  
ExecStart=/bin/sh -c '/sbin/dhclient wlan0 || true'  
ExecStop=/bin/sh -c '/sbin/wpa\_cli terminate || true'  
ExecStop=/bin/sh -c '/sbin/dhclient -r wlan0 || true'  
  
[Install]  
WantedBy=multi-user.target  
EOF  
  
chroot "$ROOTFS\_DIR" systemctl enable wifi-autoconnect.service  
  
# 11. Отключение интерактивных служб  
echo "=== Отключение интерактивных служб ==="  
chroot "$ROOTFS\_DIR" systemctl disable raspi-config.service 2>/dev/null || true  
chroot "$ROOTFS\_DIR" systemctl mask firstboot.service 2>/dev/null || true  
chroot "$ROOTFS\_DIR" rm -f /etc/profile.d/raspi-config.sh 2>/dev/null  
  
  
echo "=== Всё готово ==="

# rc.local

#!/bin/sh -e  
  
{  
 echo "=== Начало загрузки $(date) ==="  
 echo "Запуск wpa\_supplicant..."  
 wpa\_supplicant -i wlan0 -c /etc/wpa\_supplicant/wpa\_supplicant.conf -f /bootfs/wifi\_debug.log -d &  
 sleep 5  
 echo "Проверка состояния Wi-Fi..."  
 iwconfig wlan0 >> /bootfs/wifi\_debug.log 2>&1  
 echo "Попытка получить IP..."  
 dhclient -v wlan0 >> /bootfs/wifi\_debug.log 2>&1  
 echo "Итоговое состояние сети:"  
 ip a show wlan0 >> /bootfs/wifi\_debug.log 2>&1  
 echo "=== Конец загрузки $(date) ==="  
} >> /bootfs/wifi\_debug.log 2>&1  
  
exit 0

# readme\_dhcpcd.md

# ← Static address for your TCP reader  
interface eth0  
static ip\_address=192.168.1.250/24  
# (no gateway if talking directly to the reader)  
# static routers=192.168.1.1  
# static domain\_name\_servers=8.8.8.8 1.1.1.1

# setup.sh

#!/bin/bash  
  
set -euo pipefail  
  
BASE\_DIR="/home/pi/scales7.1"  
DOWNLOADS="/home/pi/Downloads/"   
  
echo\_green() {  
 echo -e "\e[32m$1\e[0m"  
}  
  
echo\_red() {  
 echo -e "\e[31m$1\e[0m"  
}  
  
sudo apt-get update  
sudo apt-get install libminizip1  
sudo apt-get install -f   
# Проверка наличия прав суперпользователя  
if [ "$(id -u)" -ne 0 ]; then  
 echo\_red "Этот скрипт нужно запускать с правами суперпользователя (root)." >&2  
 exit 1  
fi  
  
# Создание основного каталога, если он не существует  
if [ ! -d "$BASE\_DIR" ]; then  
 mkdir -p "$BASE\_DIR"  
 echo\_green "Каталог $BASE\_DIR создан"  
else  
 echo\_green "Каталог $BASE\_DIR уже существует"  
fi  
  
# Удаление существующего подключения eth0, если оно есть  
if nmcli connection show | grep -q 'eth0'; then  
 nmcli connection delete eth0  
 echo\_green "Существующее подключение eth0 удалено"  
fi  
  
# Настройка NetworkManager для исключения eth0  
echo -e "[keyfile]\nunmanaged-devices=interface-name:eth0" > /etc/NetworkManager/NetworkManager.conf  
systemctl restart NetworkManager  
echo\_green "NetworkManager больше не управляет интерфейсом eth0."  
  
sleep 30  
  
# Включение и запуск systemd-networkd  
systemctl enable systemd-networkd  
systemctl start systemd-networkd  
echo\_green "systemd-networkd активирован и запущен."  
  
# Конфигурация для eth0 через systemd-networkd  
cat <<EOF > /etc/systemd/network/10-eth0.network  
[Match]  
Name=eth0  
  
[Network]  
Address=192.168.1.249/24  
EOF  
  
systemctl restart systemd-networkd  
echo\_green "Настройка локального интерфейса eth0 через systemd-networkd завершена."  
  
# # Создание или обновление Wi-Fi подключения  
# nmcli connection add type wifi ifname wlan0 con-name "REET1212scales-auto" autoconnect yes ssid 'REET1212scales' || \  
# nmcli connection modify "REET1212scales-auto" autoconnect yes wifi-sec.key-mgmt wpa-psk wifi-sec.psk '19571212'  
  
# echo\_green "WiFi подключение 'REET1212scales-auto' настроено для автоматического подключения"  
  
# Установка Git репозитория  
cd "$BASE\_DIR"   
if [ ! -d ".git" ]; then  
 git init  
 git clone https://github.com/M100ika/scales\_submodule.git  
 git config --global --add safe.directory "$BASE\_DIR"   
 echo\_green "Git репозиторий настроен"  
else  
 echo\_green "Git репозиторий уже существует"  
fi  
  
cd "$BASE\_DIR"/scales\_submodule  
  
# Установка виртуального окружения и зависимостей  
if [ ! -d "vscales" ]; then  
 python -m venv vscales  
 echo\_green "Виртуальное окружение создано"  
else  
 echo\_green "Виртуальное окружение уже существует"  
fi  
  
source "$BASE\_DIR"/scales\_submodule/vscales/bin/activate  
if [ -f "requirements.txt" ]; then  
 pip install --upgrade pip  
 pip install -r requirements.txt  
 echo\_green "Зависимости установлены"  
else  
 echo\_green "Файл requirements.txt не найден"  
fi  
  
echo\_green "Настройка виртуального окружения завершена"  
  
# Копирование конфигурационных файлов  
cp "$BASE\_DIR"/scales\_submodule/services/config.ini "$BASE\_DIR"/scales\_submodule/src/  
sudo chmod 777 "$BASE\_DIR"/scales\_submodule/src/config.ini  
chmod +x "$BASE\_DIR"/scales\_submodule/src/config.ini  
echo\_green "Копирование config.ini завершено"   
  
cp "$BASE\_DIR"/scales\_submodule/services/pcf.service /etc/systemd/system  
echo\_green "Копирование pcf.service завершено"   
  
# Перезапуск и проверка статуса сервиса  
sudo systemctl restart pcf.service  
if systemctl is-active --quiet pcf.service; then  
 echo\_green "Демон запущен"  
else  
 echo\_green "Ошибка демона"  
fi  
  
echo\_green "Настройка демона завершена"  
  
# Установка TeamViewer  
cd "$DOWNLOADS"  
wget https://download.teamviewer.com/download/linux/teamviewer-host\_armhf.deb  
dpkg -i teamviewer-host\_armhf.deb  
echo\_green "Установка TeamViewer завершена"  
  
echo\_green "Настройка завершена"  
  
# Условие удаления скрипта: только если нет ошибок  
if [ $? -eq 0 ]; then  
 rm -- "$0"  
 echo\_green "Скрипт успешно самоудалился."  
fi  
exit 0

# setup\_bullseye.sh

#!/bin/bash  
  
set -euo pipefail  
  
BASE\_DIR="/home/pi/scales7.1"  
  
echo\_green() {  
 echo -e "\e[32m$1\e[0m"  
}  
  
echo\_red() {  
 echo -e "\e[31m$1\e[0m"  
}  
  
if [ "$(id -u)" -ne 0 ]; then  
 echo\_red "Этот скрипт нужно запускать с правами суперпользователя (root)." >&2  
 exit 1  
fi  
  
echo\_green "=== Генерация локалей: en\_GB и ru\_RU ==="  
  
# Устанавливаем пакет, если его ещё нет  
apt-get update  
apt-get install -y locales  
  
# Раскомментируем нужные строки в /etc/locale.gen  
sed -i -e 's/^# \*\(en\_GB.UTF-8 UTF-8\)/\1/' \  
 -e 's/^# \*\(ru\_RU.UTF-8 UTF-8\)/\1/' \  
 /etc/locale.gen  
  
# Генерируем  
locale-gen  
  
# Убедимся, что по-умолчанию остаётся английская  
update-locale LANG=en\_GB.UTF-8  
  
echo\_green "Сейчас локали:"  
echo " LANG: $(grep '^LANG=' /etc/default/locale)"  
echo " Локали сгенерированы: $(locale -a | grep -E 'en\_GB\.UTF-8|ru\_RU\.UTF-8')"  
  
# Обновляем репозитории и ставим git, venv и прочее  
echo\_green "Обновляем apt и устанавливаем git, python3-venv и pip"  
apt-get update  
apt-get install -y git python3-venv python3-pip  
  
# Создание основного каталога, если он не существует  
if [ ! -d "$BASE\_DIR" ]; then  
 mkdir -p "$BASE\_DIR"  
 echo\_green "Каталог $BASE\_DIR создан"  
else  
 echo\_green "Каталог $BASE\_DIR уже существует"  
fi  
  
  
DHCPCD\_CONF="/etc/dhcpcd.conf"  
DHCPCD\_BLOCK=$(cat <<EOF  
  
# Статическая настройка для eth0 (Ethernet)  
interface eth0  
static ip\_address=192.168.1.249/24  
static routers=192.168.1.1  
static domain\_name\_servers=8.8.8.8 1.1.1.1  
nohook wpa\_supplicant  
EOF  
)  
  
# Добавим блок только если его ещё нет  
if ! grep -q "interface eth0" "$DHCPCD\_CONF"; then  
 echo\_green "Добавление конфигурации eth0 в $DHCPCD\_CONF"  
 echo "$DHCPCD\_BLOCK" | tee -a "$DHCPCD\_CONF" > /dev/null  
else  
 echo\_green "Конфигурация eth0 уже присутствует в $DHCPCD\_CONF"  
fi  
  
# Перезапуск служб для применения изменений  
echo\_green "Перезапуск служб для применения изменений"  
  
echo\_green "Настройки успешно применены."  
echo\_green "Настройка локального интерфейса eth0 через dhcpcd завершена."  
  
# Git clone  
SUBMODULE\_DIR="$BASE\_DIR/scales\_submodule"  
if [ ! -d "$SUBMODULE\_DIR" ]; then  
 git clone https://github.com/M100ika/scales\_submodule.git "$SUBMODULE\_DIR"  
 echo\_green "Git репозиторий scales\_submodule клонирован"  
else  
 echo\_green "Каталог $SUBMODULE\_DIR уже существует"  
fi  
if [ ! -d "$SUBMODULE\_DIR/.git" ]; then  
 git clone https://github.com/M100ika/scales\_submodule.git "$SUBMODULE\_DIR"  
 echo\_green "Git репозиторий scales\_submodule клонирован"  
else  
 echo\_green "Git репозиторий уже существует"  
fi  
  
# Настройка безопасности git  
git config --global --add safe.directory "$SUBMODULE\_DIR"  
chown -R pi:pi "$BASE\_DIR"  
  
# Настройка ветки  
cd "$SUBMODULE\_DIR"  
if git show-ref --verify --quiet refs/heads/main; then  
 git branch --set-upstream-to=origin/main main  
else  
 echo\_red "Ветка main не существует локально"  
fi  
  
# Создание логов  
mkdir -p "$SUBMODULE\_DIR/scales\_log/error\_log"  
  
# Установка виртуального окружения и зависимостей  
if [ ! -d "vscales" ]; then  
 python3 -m venv vscales  
 echo\_green "Виртуальное окружение создано"  
else  
 echo\_green "Виртуальное окружение уже существует"  
fi  
source "$BASE\_DIR"/scales\_submodule/vscales/bin/activate  
if [ -f "requirements.txt" ]; then  
 pip install --upgrade pip  
 pip install -r requirements.txt  
 echo\_green "Зависимости установлены"  
else  
 echo\_green "Файл requirements.txt не найден"  
fi  
  
echo\_green "Настройка виртуального окружения завершена"  
# Копирование конфигурационных файлов  
cp "$BASE\_DIR"/scales\_submodule/services/config.ini "$BASE\_DIR"/scales\_submodule/src/  
chmod 666 "$BASE\_DIR"/scales\_submodule/src/config.ini  
echo\_green "Копирование config.ini завершено"   
  
cp "$BASE\_DIR"/scales\_submodule/services/pcf.service /etc/systemd/system/  
echo\_green "Копирование pcf.service завершено"   
  
# Перезапуск и проверка статуса сервиса  
systemctl daemon-reload  
systemctl enable pcf.service  
systemctl restart pcf.service  
if systemctl is-active --quiet pcf.service; then  
 echo\_green "Демон запущен"  
else  
 echo\_red "Ошибка демона"  
fi  
echo\_green "Настройка демона завершена"  
echo\_green "Настройка завершена"  
# Условие удаления скрипта: только если нет ошибок  
cd /home/pi  
rm -- "$0"  
echo\_green "Скрипт успешно самоудалился."  
exit 0

# wpa\_supplicant.conf

ctrl\_interface=DIR=/var/run/wpa\_supplicant GROUP=netdev  
update\_config=1  
country=KZ  
  
network={  
 ssid="smart2"  
 psk="Kazatu2025"  
 key\_mgmt=WPA-PSK  
 priority=1  
 scan\_ssid=1}

**Приложение 2.**

Сервисы и конфигурационный файл

# config.ini

[Parameters]  
model = OrganicScalesPro7.2  
type = Scales  
serial\_number = OrgSc25-02  
url = https://smart-farm.kz:8502/api/v2/RawFeedings  
median\_url = http://194.4.56.86:8501/api/weights  
array\_url = https://smart-farm.kz:8502/v2/OneTimeWeighings  
arduino\_port = /dev/ttyACM0  
debug = 1  
database = 1  
  
[Calibration]  
calibration\_mode = 0  
rfid\_calibration\_mode = 0  
taring\_rfid =   
scaling\_rfid =   
weight = 80  
offset = 16766507  
scale = -3358.285714285714  
  
[DbId]  
id = 30  
version = 7.1  
  
[Sprayer]  
function = 1   
medicine\_pin = 23  
paint\_pin = 24  
post\_url = https://smart-farm.kz:8502/api/v2/SprayingTaskResults  
l/min = 1  
  
[RFID\_Reader]  
reader\_usb = 0  
reader\_buzzer = 1  
reader\_port = /dev/ttyUSB0  
reader\_power = 1  
reader\_timeout = 2  
  
[Relay]  
sensor\_pin = 17

# pcf.service

[Unit]  
Description=PCF Service  
After=network.target  
  
[Service]  
ExecStart=/home/pi/scales7.1/scales\_submodule/vscales/bin/python /home/pi/scales7.1/scales\_submodule/src/main\_pcf.py  
WorkingDirectory=/home/pi/  
StandardOutput=inherit  
StandardError=inherit  
Restart=always  
RestartSec=10  
User=root  
  
[Install]  
WantedBy=multi-user.target  
TimeoutStopSec=30

# update\_and\_reboot.sh

#!/bin/bash  
# Git pull и перезагрука  
# Перейти в нужную директорию  
cd /home/pi/scales7.1/scales\_submodule/  
  
# Выполнить git pull  
git pull  
echo hi  
# Перезагрузить систему  
sudo reboot now  
  
  
  
#Чтобы установить сделайте следующее   
#crontab -e  
#Добавьте строку в cron   
#0 4 \* \* \* /home/pi/scales7.1/scales\_submodule/services/update\_and\_reboot.sh >> /home/pi/scales7.1/scales\_submodule/services/update\_and\_reboot.log 2>&1

**Приложение 3.**

Основные модули проекта

# \_\_gpio\_simulator.py

class MockGPIO:  
 BCM = 'BCM'  
 OUT = 'OUT'  
 IN = 'IN'  
 HIGH = 1  
 LOW = 0  
  
 \_pin\_modes = {}  
 \_pin\_states = {}  
  
 @classmethod  
 def setmode(cls, mode):  
 print(f"GPIO mode set to {mode}")  
   
 @classmethod  
 def setwarnings(cls, flag):  
 cls.\_warnings = flag  
 print(f"GPIO warnings set to {'on' if flag else 'off'}")  
  
 @classmethod  
 def setup(cls, pin, mode):  
 if pin in cls.\_pin\_modes and cls.\_warnings:  
 print(f"Warning: Pin {pin} is being setup again.")  
 cls.\_pin\_modes[pin] = mode  
 cls.\_pin\_states[pin] = cls.LOW # Default to LOW  
 print(f"GPIO pin {pin} set up as {mode}")  
   
 @classmethod  
 def output(cls, pin, state):  
 if pin in cls.\_pin\_modes and cls.\_pin\_modes[pin] == cls.OUT:  
 cls.\_pin\_states[pin] = state  
 print(f"GPIO pin {pin} output set to {'HIGH' if state == cls.HIGH else 'LOW'}")  
 else:  
 print(f"Error: Pin {pin} is not set as OUTPUT.")  
  
 @classmethod  
 def input(cls, pin):  
 if pin in cls.\_pin\_modes and cls.\_pin\_modes[pin] == cls.IN:  
 return cls.\_pin\_states[pin]  
 else:  
 print(f"Error: Pin {pin} is not set as INPUT.")  
 return None  
  
 @classmethod  
 def cleanup(cls):  
 cls.\_pin\_modes.clear()  
 cls.\_pin\_states.clear()  
 print("GPIO cleanup completed.")

# \_adc\_data.py

import serial  
from collections import Counter  
import statistics  
  
  
class ArduinoSerial:  
 def \_\_init\_\_(self, port, baud\_rate=9600, timeout = 1, window = 50):  
 self.adc\_arr = []  
 self.calib\_arr = []  
 self.window = window  
 self.calib\_window = 50  
 self.SCALE = 1  
 self.OFFSET = 0  
 self.timeout = timeout  
 self.NOIZE = 0  
 self.port = port  
 self.baud\_rate = baud\_rate  
 self.arduino = None   
  
  
 def connect(self):   
 self.arduino = serial.Serial(self.port, self.baud\_rate, bytesize=serial.EIGHTBITS, timeout = self.timeout)  
  
  
 def disconnect(self):   
 self.arduino.close()  
  
  
 def set\_arr(self, arr):   
 self.adc\_arr = arr  
  
  
 def clean\_arr(self):  
 self.adc\_arr = []  
  
  
 def get\_arr(self):   
 return self.adc\_arr  
  
  
 def read\_data(self):   
 if not self.arduino:  
 raise ValueError(f'Arduino is not connected\n')  
 try:  
 self.arduino.write(b'\x02')   
 response = self.arduino.read(4)   
 return int.from\_bytes(response, byteorder='big', signed=False)   
 except serial.SerialTimeoutException as e:  
 print(f'Timeout error: {e}')  
  
  
 def set\_scale(self, scale):  
 self.SCALE = scale  
  
  
 def set\_offset(self, offset):   
 self.OFFSET = offset  
  
  
 def get\_offset(self):   
 return self.OFFSET  
   
   
 def get\_scale(self):   
 return self.SCALE  
  
  
 def get\_measure(self):   
 adc\_val = self.read\_data()   
 adc\_val = (adc\_val-self.OFFSET)  
 return round(adc\_val/self.SCALE, 2)  
   
 def get\_measure\_2(self, samples: int = 10) -> float:  
 """Возвращает финальный вес с учётом offset и scale."""  
 raw = self.read\_average(samples)  
 return (raw - self.OFFSET) / self.SCALE  
  
  
 def check\_weight(self):   
 adc\_val = self.get\_measure()  
 if adc\_val < 20:  
 return False  
 return True  
   
  
 def calc\_mean(self):   
 adc\_val = self.get\_measure()  
  
 if len(self.adc\_arr) == self.window:  
 self.adc\_arr.pop(0)  
  
 self.adc\_arr.append(adc\_val)  
  
 return statistics.median(self.adc\_arr) # Убираем влияние выбросов  
  
  
 def common\_filter(self):   
 return Counter(self.get\_arr()).most\_common(1)[0][0]  
   
  
 def tare(self, times=16):   
 sum = self.read\_average(times)  
 self.set\_offset(sum)  
 return sum/self.OFFSET  
  
  
 def read\_average(self, times=16):  
  
 sum = 0  
 for i in range(times):  
 sum += self.read\_data()  
 return sum / times  
  
  
 def calib\_read(self, times = 20):   
 """Use only in calibration"""  
 self.calib\_arr = []  
 for i in range(5):  
 self.read\_data()  
 self.set\_arr([])  
 for i in range(times):  
 self.calib\_arr.append(self.read\_data())  
 counter = Counter(self.calib\_arr)  
 most\_common = counter.most\_common(1)[0][0]  
 return most\_common  
   
  
 def calib\_read\_mediana(self):   
 """Use only in calibration"""  
 self.calib\_arr = []  
 for i in range(53):  
 row\_number = self.read\_data()  
 if len(self.calib\_arr)==self.calib\_window:  
 self.calib\_arr.pop(0)  
 print(self.calib\_arr)  
 self.calib\_arr.append(row\_number)  
 print('Finish')  
 return round(statistics.median(self.calib\_arr), 2)  
   
  
 def calib\_read\_average(self):   
 """Use only in calibration"""  
 self.calib\_arr = []  
 for i in range(100):  
 row\_number = self.read\_data()  
 if len(self.calib\_arr)==self.calib\_window:  
 self.calib\_arr.pop(0)  
 self.calib\_arr.append(row\_number)  
 print('Finish')  
 return round(sum(self.calib\_arr) / len(self.calib\_arr),2)  
  
  
 def get\_weight(self, times=16):   
 value = (self.read\_average(times) - self.OFFSET)  
 weight = (value / self.SCALE)  
 return weight  
   
 def read\_raw(self) -> int:  
 """Читает одно значение из Arduino (4 байта, big-endian)."""  
 assert self.arduino, "Arduino не подключена"  
 self.arduino.write(b'\x02')  
 data = self.arduino.read(4)  
 return int.from\_bytes(data, byteorder='big', signed=False)

# \_chafon\_rfid\_lib.py

import time  
from loguru import logger  
from \_config\_manager import ConfigManager  
import serial  
from serial.tools import list\_ports  
  
from chafon\_rfid.base import CommandRunner, ReaderCommand, ReaderInfoFrame, ReaderResponseFrame, ReaderType  
from chafon\_rfid.command import (CF\_GET\_READER\_INFO, CF\_SET\_BUZZER\_ENABLED, CF\_SET\_RF\_POWER)  
from chafon\_rfid.response import G2\_TAG\_INVENTORY\_STATUS\_MORE\_FRAMES  
from chafon\_rfid.transport\_serial import SerialTransport  
from chafon\_rfid.uhfreader288m import G2InventoryCommand, G2InventoryResponseFrame  
  
class RFIDReader:  
 def \_\_init\_\_(self):  
 self.config\_manager = ConfigManager()  
 self.reader\_port = self.config\_manager.get\_setting("RFID\_Reader", "reader\_port")  
 if self.reader\_port == "Отсутствует":  
 self.reader\_port = None  
 initial\_power = int(self.config\_manager.get\_setting("RFID\_Reader", "reader\_power"))  
 self.reader\_power = self.closest\_number(initial\_power)  
 self.reader\_timeout = int(self.config\_manager.get\_setting("RFID\_Reader", "reader\_timeout"))  
 self.reader\_buzzer = int(self.config\_manager.get\_setting("RFID\_Reader", "reader\_buzzer"))  
  
 self.inventory\_cmd = G2InventoryCommand(q\_value=4, antenna=0x80)  
 self.frame\_type = G2InventoryResponseFrame  
  
  
 def closest\_number(self, power):  
 numbers = [1, 3, 5, 7, 10, 15, 20, 26]  
 if power < min(numbers):  
 return min(numbers)  
 return min(numbers, key=lambda x: abs(x - power))  
  
  
 def find\_rfid\_reader(self):  
 ports = list(list\_ports.comports())  
 for port in ports:  
 try:  
 transport = SerialTransport(device=port.device)  
 runner = CommandRunner(transport)  
 get\_reader\_info\_cmd = ReaderCommand(CF\_GET\_READER\_INFO)  
   
 response = runner.run(get\_reader\_info\_cmd)  
 reader\_info = ReaderInfoFrame(response)  
   
 if reader\_info.type:   
 self.config\_manager.update\_setting("RFID\_Reader", "reader\_port", port.device)  
 return port.device  
   
 except (OSError, serial.SerialException, ValueError):  
 pass  
   
 self.config\_manager.update\_setting("RFID\_Reader", "reader\_port", "Отсутствует")  
 return None  
  
  
 def \_get\_reader\_type(self):  
 get\_reader\_info = ReaderCommand(CF\_GET\_READER\_INFO)  
 self.transport = SerialTransport(device=self.reader\_port)  
 self.runner = CommandRunner(self.transport)  
 reader\_info = ReaderInfoFrame(self.runner.run(get\_reader\_info))  
 return reader\_info.type  
  
 def \_run\_command(self, command):  
 self.transport.write(command.serialize())  
 status = ReaderResponseFrame(self.transport.read\_frame()).result\_status  
 return status  
  
 def \_set\_power(self):  
 return self.\_run\_command(ReaderCommand(CF\_SET\_RF\_POWER, data=[self.reader\_power]))  
  
 def \_set\_buzzer\_enabled(self):  
 return self.\_run\_command(ReaderCommand(CF\_SET\_BUZZER\_ENABLED, data=[self.reader\_buzzer and 1 or 0]))  
  
 def connect(self):  
 tag\_id = None  
  
 try:  
 reader\_type = self.\_get\_reader\_type()  
 if reader\_type in (ReaderType.UHFReader86, ReaderType.UHFReader86\_1):  
 get\_inventory\_cmd = G2InventoryCommand(q\_value=4, antenna=0x80)  
 frame\_type = G2InventoryResponseFrame  
 self.\_set\_power()  
 self.\_set\_buzzer\_enabled()  
 else:  
 #logger.error(f'Unsupported reader type: {reader\_type}')  
 return None  
 except ValueError as e:  
 #logger.error(f'Unknown reader type: {e}')  
 return None  
  
 start\_time = time.time()  
 while tag\_id is None:  
 try:  
 self.transport.write(get\_inventory\_cmd.serialize())  
 inventory\_status = None  
 while inventory\_status is None or inventory\_status == G2\_TAG\_INVENTORY\_STATUS\_MORE\_FRAMES:  
 if time.time() - start\_time > self.reader\_timeout:  
 logger.info("Timeout reached, stopping tag reading.")  
 return None  
 resp = frame\_type(self.transport.read\_frame())  
 inventory\_status = resp.result\_status  
 tags\_generator = resp.get\_tag()  
 try:  
 first\_tag = next(tags\_generator, None)  
 if first\_tag:  
 tag\_id = first\_tag.epc.hex()  
 break  
 except StopIteration:  
 continue  
 except KeyboardInterrupt:  
 logger.error("Operation cancelled by user.")  
 break  
 except Exception as e:  
 logger.error(f'Error: {e}')  
 continue  
  
 self.transport.close()  
 return tag\_id if tag\_id else None  
   
  
 def open(self, timeout: float = 0.1):  
 """  
 Открывает соединение и настраивает ридер (мощность, бузер).  
 timeout задаёт значение таймаута для SerialTransport.  
 """  
 if not self.reader\_port:  
 raise ValueError("RFID reader port not set. Call find\_rfid\_reader() first.")  
 self.transport = SerialTransport(device=self.reader\_port, timeout=timeout)  
 self.runner = CommandRunner(self.transport)  
 # Disable buzzer if not needed  
 self.\_run\_command(ReaderCommand(CF\_SET\_RF\_POWER, data=[self.reader\_power]))  
 self.\_run\_command(ReaderCommand(CF\_SET\_BUZZER\_ENABLED, data=[1 if self.reader\_buzzer else 0]))  
  
 def read\_tag(self, timeout: float = None) -> str:  
 """  
 Выполняет одну попытку чтения метки. Если timeout указан, ждёт до таймаута.  
 Возвращает EPC-hex или None.  
 """  
 if not self.transport or not self.runner:  
 raise RuntimeError("Transport not open. Call open() before read\_tag().")  
  
 # Get reader type and setup if needed  
 reader\_info = ReaderInfoFrame(self.runner.run(ReaderCommand(CF\_GET\_READER\_INFO)))  
 if reader\_info.type not in (ReaderType.UHFReader86, ReaderType.UHFReader86\_1):  
 return None  
  
 start = time.time()  
 # Отправляем инвентаризационную команду  
 self.transport.write(self.inventory\_cmd.serialize())  
 while True:  
 # Проверяем таймаут  
 if timeout and (time.time() - start) > timeout:  
 logger.debug("RFID read timeout")  
 return None  
 # Читаем фрейм и проверяем  
 frame = self.transport.read\_frame()  
 resp = self.frame\_type(frame)  
 if resp.result\_status != G2\_TAG\_INVENTORY\_STATUS\_MORE\_FRAMES:  
 for tag in resp.get\_tag():  
 return tag.epc.hex()  
 # Продолжаем, если ещё кадры  
  
 def close(self):  
 """  
 Закрывает соединение с портом.  
 """  
 if self.transport:  
 self.transport.close()  
 self.transport = None  
 self.runner = None  
  
 def \_run\_command(self, command: ReaderCommand) -> int:  
 """  
 Вспомогательная функция для отправки одиночных команд.  
 """  
 self.transport.write(command.serialize())  
 status = ReaderResponseFrame(self.transport.read\_frame()).result\_status  
 return status

# \_config\_manager.py

import configparser  
import os  
from pathlib import Path  
from loguru import logger  
#from \_headers import CONFIG\_FILE\_PATH  
SRC\_DIR = Path(\_\_file\_\_).resolve().parent  
CONFIG\_FILE\_PATH = SRC\_DIR / 'config.ini'  
  
class ConfigManager:  
 def \_\_init\_\_(self, path=CONFIG\_FILE\_PATH):  
 self.path = path  
 if not self.path.exists():  
 self.create\_config()  
  
 def create\_config(self):  
 try:  
 config = configparser.ConfigParser()  
 config.add\_section("Parameters")  
 config.add\_section("Calibration")  
 config.add\_section("DbId")  
 config.add\_section("Sprayer")  
 config.add\_section("RFID\_Reader")  
 config.add\_section("Relay")  
  
 config.set("Parameters", "model", "feeder\_model\_1")   
 config.set("Parameters", "type", "Feeder")   
 config.set("Parameters", "serial\_number", "feeder0423v21-1")   
 config.set("Parameters", "url", "https://smart-farm.kz:8502/api/v2/RawFeedings")   
 config.set("Parameters", "median\_url", "http://194.4.56.86:8501/api/weights")   
 config.set("Parameters", "array\_url", "https://smart-farm.kz:8502/v2/OneTimeWeighings")   
 config.set("Parameters", "arduino\_port", "dev/ttyUSB0")   
 config.set("Parameters", "debug", "1")  
 config.set("Parameters", "database", "0")  
   
 config.set("Calibration", "calibration\_mode", "0")   
 config.set("Calibration", "taring\_rfid", "")   
 config.set("Calibration", "scaling\_rfid", "")   
 config.set("Calibration", "weight", "80")   
 config.set("Calibration", "offset", "16766507")   
 config.set("Calibration", "scale", "-3358.285714285714")  
  
 config.set("DbId", "id", "0")   
 config.set("DbId", "version", "7.1")   
  
 config.set("Sprayer", "function", "on")   
 config.set("Sprayer", "medicine\_pin", "18")   
 config.set("Sprayer", "paint\_pin", "23")   
 config.set("Sprayer", "post\_url", "https://smart-farm.kz:8502/api/v2/SprayingTaskResults")   
 config.set("Sprayer", "l/min", "0.15")   
  
 config.set("RFID\_Reader", "reader\_usb", "0")  
 config.set("RFID\_Reader", "reader\_port", "/dev/ttyUSB0")  
 config.set("RFID\_Reader", "reader\_power", "26")  
 config.set("RFID\_Reader", "reader\_timeout", "2")  
 config.set("RFID\_Reader", "reader\_buzzer", "1")  
  
 config.set("Relay", "sensor\_pin", "17")  
  
 with open(self.path, "w") as config\_file:  
 config.write(config\_file)  
  
 except ValueError as e:  
 logger.error(f'ConfigManager, create\_config method error: {e}')  
  
 def get\_config(self):  
 try:  
 config = configparser.ConfigParser()  
 config.read(self.path)  
 return config  
 except ValueError as e:  
 logger.error(f'ConfigManager, get\_config method error: {e}')  
  
 def get\_setting(self, section, setting):  
 try:  
 config = self.get\_config()  
 value = config.get(section, setting)  
 return value  
 except ValueError as e:  
 logger.error(f'ConfigManager, get\_setting method error: {e}')  
  
 def update\_setting(self, section, setting, value):  
 try:  
 config = self.get\_config()  
 config.set(section, setting, str(value))  
 with open(self.path, "w") as config\_file:  
 config.write(config\_file)  
 except ValueError as e:  
 logger.error(f'ConfigManager, update\_setting method error: {e}')

# \_glb\_val.py

from \_config\_manager import ConfigManager  
  
  
config\_manager = ConfigManager()  
  
DEBUG = int(config\_manager.get\_setting("Parameters", "debug"))  
  
SQL\_ON = int(config\_manager.get\_setting("Parameters", "database"))  
  
EQUIPMENT\_TYPE = config\_manager.get\_setting("Parameters", "type")  
SERIAL\_NUMBER = config\_manager.get\_setting("Parameters", "serial\_number")  
  
URL = config\_manager.get\_setting("Parameters", "url")  
HEADERS = {'Content-type': 'application/json'}  
  
INTERNET\_CHECK\_INTERVAL = 300  
  
PRESET\_VALUE = 0xFFFF  
POLYNOMIAL = 0x8408  
TCP\_IP = '192.168.1.250'   
TCP\_PORT = 60000   
BUFFER\_SIZE = 1024  
  
TYPE\_SCALES = config\_manager.get\_setting("Parameters", "serial\_number") # Забираем серийный номер из config.ini  
PORT = config\_manager.get\_setting("Parameters", "arduino\_port")  
SPRAYER = True if int(config\_manager.get\_setting("Sprayer", "function")) == 1 else False  
  
  
ARDUINO\_PORT = config\_manager.get\_setting("Parameters", "arduino\_port")   
if ARDUINO\_PORT == "Отсутствует":  
 ARDUINO\_PORT == None  
  
CALIBRATION\_MODE = int(config\_manager.get\_setting("Calibration", "calibration\_mode"))  
RFID\_CABLIBRATION\_MODE = int(config\_manager.get\_setting("Calibration", "rfid\_calibration\_mode"))  
CALIBRATION\_TARING\_RFID = config\_manager.get\_setting("Calibration", "taring\_rfid")  
CALIBRATION\_SCALE\_RFID = config\_manager.get\_setting("Calibration", "scaling\_rfid")  
CALIBRATION\_WEIGHT = float(config\_manager.get\_setting("Calibration", "weight"))  
RELAY\_PIN = int(config\_manager.get\_setting("Relay", "sensor\_pin"))  
RFID\_TIMEOUT = float(config\_manager.get\_setting("RFID\_Reader", "reader\_timeout"))  
  
RFID\_READER\_USB = int(config\_manager.get\_setting("RFID\_Reader", "reader\_usb"))  
  
def closest\_number(n):  
 numbers = [1, 3, 5, 7, 10, 15, 20, 26]  
 if n < min(numbers):  
 return min(numbers)  
 return min(numbers, key=lambda x: abs(x - n))  
initial\_power = int(config\_manager.get\_setting("RFID\_Reader", "reader\_power"))  
power\_key = closest\_number(initial\_power)  
  
SET\_POWER\_MESSAGES = {1:[0x53, 0x57, 0x00, 0x25, 0xFF, 0x21, 0xC3, 0x55, 0x02, 0x01, 0x00, 0x00, 0x01, 0x01, 0x04, 0x4E, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x1E, 0x0A, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6A],  
3:[0x53, 0x57, 0x00, 0x25, 0xFF, 0x21, 0xC3, 0x55, 0x02, 0x01, 0x00, 0x00, 0x03, 0x01, 0x04, 0x4E, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x1E, 0x0A, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x68],  
5:[0x53, 0x57, 0x00, 0x25, 0xFF, 0x21, 0xC3, 0x55, 0x02, 0x01, 0x00, 0x00, 0x05, 0x01, 0x04, 0x4E, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x1E, 0x0A, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x66],  
7:[0x53, 0x57, 0x00, 0x25, 0xFF, 0x21, 0xC3, 0x55, 0x02, 0x01, 0x00, 0x00, 0x07, 0x01, 0x04, 0x4E, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x1E, 0x0A, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x64],  
10:[0x53, 0x57, 0x00, 0x25, 0xFF, 0x21, 0xC3, 0x55, 0x02, 0x01, 0x00, 0x00, 0x0A, 0x01, 0x04, 0x4E, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x1E, 0x0A, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x61],  
15:[0x53, 0x57, 0x00, 0x25, 0xFF, 0x21, 0xC3, 0x55, 0x02, 0x01, 0x00, 0x00, 0x0F, 0x01, 0x04, 0x4E, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x1E, 0x0A, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x5C],  
20:[0x53, 0x57, 0x00, 0x25, 0xFF, 0x21, 0xC3, 0x55, 0x02, 0x01, 0x00, 0x00, 0x14, 0x01, 0x04, 0x4E, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x1E, 0x0A, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x57],  
26:[0x53, 0x57, 0x00, 0x25, 0xFF, 0x21, 0xC3, 0x55, 0x02, 0x01, 0x00, 0x00, 0x1A, 0x01, 0x04, 0x4E, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x1E, 0x0A, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x51]}  
  
RFID\_READER\_POWER = SET\_POWER\_MESSAGES.get(power\_key)

# \_headers.py

import sys  
import subprocess  
import pkg\_resources  
from pkg\_resources import DistributionNotFound, VersionConflict  
  
  
def should\_install\_requirement(requirement) -> str:  
 should\_install = False  
 try:  
 pkg\_resources.require(requirement)  
 except (DistributionNotFound, VersionConflict):  
 should\_install = True  
 return should\_install  
  
  
def install\_packages(requirement\_list) -> list:  
 try:  
 requirements = [  
 requirement  
 for requirement in requirement\_list  
 if should\_install\_requirement(requirement)  
 ]  
 if len(requirements) > 0:  
 subprocess.check\_call([sys.executable, "-m", "pip", "install", \*requirements])  
 else:  
 from loguru import logger  
 logger.info("Requirements already satisfied (info).")  
 except Exception as e:  
 print(e)

# \_lib\_pcf.py

#!/usr/bin/python3  
  
"""File containing all working functions and algorithms for determining the weight of the animal and spraying.  
Author: Aidar Alimbayev and Suieubayev Maxat  
Contact: maxat.suieubayev@gmail.com  
Number: +7 775 818 48 43"""  
  
from datetime import datetime  
import json  
import requests  
import socket  
import binascii  
import timeit  
import statistics  
import time  
from collections import Counter  
from loguru import logger  
import \_values\_class as value\_data  
import \_adc\_data as ADC  
from \_chafon\_rfid\_lib import RFIDReader  
try:  
 import RPi.GPIO as GPIO  
except RuntimeError:  
 from \_\_gpio\_simulator import MockGPIO as GPIO  
  
from \_sql\_database import SqlDatabase  
  
from \_config\_manager import ConfigManager  
from \_sprayer import Sprayer  
from \_glb\_val import \*  
import select  
  
config\_manager = ConfigManager()  
  
def start\_obj():  
 try:  
 obj = ADC.ArduinoSerial(PORT)  
 obj.connect()  
 offset, scale = float(config\_manager.get\_setting("Calibration", "offset")), float(config\_manager.get\_setting("Calibration", "scale"))  
 obj.set\_offset(offset)  
 obj.set\_scale(scale)  
 time.sleep(1)  
 return obj  
 except Exception as e:  
 logger.error(f'Error connecting: {e}')  
  
  
def start\_filter(obj):  
 try:  
 for i in range(5):  
 obj.calc\_mean()  
 obj.set\_arr([])  
 except Exception as e:  
 logger.error(f'start filter function Error: {e}')  
  
  
def \_set\_power\_RFID\_ethernet():  
 try:  
 logger.info(f"Start configure antenna power")  
 s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 s.connect((TCP\_IP, TCP\_PORT))  
 s.send(bytearray(RFID\_READER\_POWER))  
 data = s.recv(BUFFER\_SIZE)  
 recieved\_data = str(binascii.hexlify(data))  
 check\_code = "b'4354000400210143'"  
 if recieved\_data == check\_code:  
 logger.info(f"operation succeeded")  
 else:   
 logger.info(f"Denied!")  
 except Exception as e:  
 logger.error(f"\_set\_power\_RFID\_ethernet: An error occurred: {e}")  
 finally:  
 s.close()   
  
def \_\_connect\_rfid\_reader\_ethernet():  
 command = bytearray([0x53, 0x57, 0x00, 0x06, 0xff, 0x01, 0x00, 0x00, 0x00, 0x50])  
 s = None  
 try:  
 logger.debug("Starting RFID Ethernet read cycle...")  
 s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 s.settimeout(RFID\_TIMEOUT)  
 s.connect((TCP\_IP, TCP\_PORT))  
  
 s.send(command)  
 time.sleep(0.5) # Увеличили задержку, чтобы дать ридеру время ответить  
  
 buffer = b''  
 try:  
 while True:  
 part = s.recv(BUFFER\_SIZE)  
 if not part:  
 break  
 buffer += part  
 except socket.timeout:  
 pass # ожидаемое поведение  
  
 hex\_data = binascii.hexlify(buffer).decode('utf-8')  
 logger.debug(f'Raw RFID response: {hex\_data}')  
 logger.debug(f'Response length: {len(hex\_data)} characters')  
  
 epcs = extract\_all\_epc\_from\_raw(hex\_data)  
 if not epcs:  
 logger.warning("No EPC tags extracted.")  
 return None  
  
 epc = epcs[-1] # можно брать первый [0], если хочется сразу реагировать  
 logger.info(f'Corrected RFID: {epc}')  
 return epc  
  
 except Exception as e:  
 logger.error(f'Error during RFID Ethernet read: {e}')  
 return None  
  
 finally:  
 if s:  
 try:  
 s.close()  
 logger.debug("RFID socket closed.")  
 except Exception as e:  
 logger.warning(f"Error closing RFID socket: {e}")  
  
def extract\_all\_epc\_from\_raw(raw\_data):  
 """  
 Ищет все EPC (12 байт = 24 символа) в длинной hex-строке.  
 Начало EPC всегда после "e200" или по шаблону: ищем метки с префиксом "e20000..."  
 """  
 epcs = []  
 for i in range(len(raw\_data)):  
 candidate = raw\_data[i:i + 24]  
 if candidate.startswith('e200') and len(candidate) == 24:  
 epcs.append(candidate)  
 return epcs  
  
  
  
def \_\_connect\_rfid\_reader\_ethernet\_2():  
 try:  
 with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:  
 s.connect((TCP\_IP, TCP\_PORT))  
 s.settimeout(RFID\_TIMEOUT)  
  
 # Отправляем команду на считывание метки  
 command = bytearray([0x53, 0x57, 0x00, 0x06, 0xff, 0x01, 0x00, 0x00, 0x00, 0x50])  
 s.send(command)  
 time.sleep(0.2)  
  
 ready = select.select([s], [], [], RFID\_TIMEOUT)  
 if ready[0]:  
 data = s.recv(BUFFER\_SIZE)  
 full\_animal\_id = binascii.hexlify(data).decode('utf-8')  
  
 logger.info(f'Raw RFID response: {full\_animal\_id}')  
 logger.info(f'Response length: {len(full\_animal\_id)} characters')  
  
 # Универсальная обработка EPC  
 corrected\_rfid = extract\_epc\_from\_raw(full\_animal\_id)  
 if corrected\_rfid:  
 logger.info(f'Corrected RFID: {corrected\_rfid}')  
 return corrected\_rfid  
 else:  
 logger.warning('Failed to extract RFID.')  
 return None  
 else:  
 logger.info("No RFID data received within timeout")  
 return None  
  
 except Exception as e:  
 logger.error(f'Error connect RFID reader: {e}')  
 return None  
  
  
def extract\_epc\_from\_raw(raw\_data):  
 """  
 Универсальная функция для извлечения EPC из ответа RFID-ридера.  
 Убирает CRC и адаптируется к разным форматам меток.  
 """  
 if len(raw\_data) < 40:  
 logger.warning("RFID response is too short.")  
 return None  
  
 # Находим возможные позиции начала EPC (обычно начинается после заголовка)  
 start\_positions = [40, 44, 48] # Возможные позиции EPC  
  
 for start in start\_positions:  
 epc\_candidate = raw\_data[start:start + 24] # EPC 12 байт (24 символа)  
   
 if len(epc\_candidate) == 24:  
 # Удаляем последние 4 символа (CRC)  
 corrected\_epc = epc\_candidate[:-4]  
 return corrected\_epc  
  
 return None  
  
  
def post\_median\_data(animal\_id, weight\_finall, type\_scales, sql\_db): # Sending data into Igor's server through JSON  
 try:  
 logger.debug(f'START SEND DATA TO SERVER:')  
 url = config\_manager.get\_setting("Parameters", "median\_url")  
 headers = {'Content-type': 'application/json'}  
 data = {"AnimalNumber" : animal\_id,  
 "Date" : str(datetime.now()),  
 "Weight" : weight\_finall,  
 "ScalesModel" : type\_scales}  
 answer = requests.post(url, data=json.dumps(data), headers=headers, timeout=30)  
 logger.debug(f'Answer from server: {answer}') # Is it possible to stop on this line in the debug?  
 logger.debug(f'Content from main server: {answer.content}')  
 if answer.status\_code != 200:  
 sql\_db.no\_internet(data)  
 logger.error(f'Response status code: {answer.status\_code}')  
 except requests.exceptions.RequestException as e:  
 logger.error(f'Error sending data to server: {e}')  
 if SQL\_ON:  
 database = SqlDatabase()  
 database.no\_internet(data)  
 else:  
 logger.info('Data sent successfully')  
  
  
def post\_array\_data(type\_scales, animal\_id, weight\_list, weighing\_start\_time, weighing\_end\_time, sql\_db):  
 try:  
 logger.debug(f'Post data function start')  
 url = config\_manager.get\_setting("Parameters", "array\_url")  
 headers = {'Content-Type': 'application/json; charset=utf-8'}  
 data = {  
 "ScalesSerialNumber": type\_scales,  
 "WeighingStart": weighing\_start\_time,  
 "WeighingEnd": weighing\_end\_time,  
 "RFIDNumber": animal\_id,  
 "Data": weight\_list  
 }   
 post = requests.post(url, data=json.dumps(data), headers=headers, timeout=30)  
 logger.debug(f'Post Data: {data}')  
 logger.debug(f'Answer from server: {post}') # Is it possible to stop on this line in the debug?  
 logger.debug(f'Content from main server: {post.content}')  
 if post.status\_code != 200:  
 sql\_db.no\_internet(data)  
 logger.error(f'Response status code: {post.status\_code}')  
 except requests.exceptions.RequestException as e:  
 logger.error(f'Error post data: {e}')  
 if SQL\_ON:  
 database = SqlDatabase()  
 database.no\_internet(data)  
  
  
def \_\_input\_with\_timeout(timeout):  
 import sys, select  
  
 logger.info(f"You have {int(timeout)} seconds to respond.")  
 ready, \_, \_ = select.select([sys.stdin], [], [], timeout)  
 if ready:  
 return sys.stdin.readline().strip()  
 else:  
 logger.warning("Input timed out.")  
 logger.error("User input timed out.")  
  
  
def \_\_calibrate(timeout):  
 start\_time = time.time()  
  
 def time\_remaining():  
 return max(0, timeout - (time.time() - start\_time))  
  
 try:  
 logger.info(f'\033[1;33mStarting the calibration process.\033[0m')  
 arduino = ADC.ArduinoSerial(config\_manager.get\_setting("Parameters", "arduino\_port"), 9600, timeout=30)  
 arduino.connect()  
  
 logger.info(f"Ensure the scale is clear. Press any key once it's empty and you're ready to proceed.")  
 time.sleep(1)  
 \_\_input\_with\_timeout(time\_remaining())  
  
 offset = arduino.calib\_read\_mediana()  
 logger.info("Offset: {}".format(offset))  
 arduino.set\_offset(offset)  
  
 logger.info("Place a known weight on the scale and then press any key to continue.")  
 \_\_input\_with\_timeout(time\_remaining())  
  
 measured\_weight = (arduino.calib\_read\_mediana() - arduino.get\_offset())  
 logger.info("Please enter the item's weight in kg.\n>")  
   
 item\_weight = \_\_input\_with\_timeout(time\_remaining())  
 scale = int(measured\_weight) / int(item\_weight)  
 arduino.set\_scale(scale)  
  
 logger.info(f"\033[1;33mCalibration complete.\033[0m")  
 logger.info(f'Calibration details\n\n —Offset: {offset}, \n\n —Scale factor: {scale}')  
   
 config\_manager.update\_setting("Calibration", "Offset", offset)  
 config\_manager.update\_setting("Calibration", "Scale", scale)  
  
 arduino.disconnect()  
 del arduino  
  
 except TimeoutError:  
 logger.error("Calibration timed out.")  
 arduino.disconnect()  
 del arduino  
 except Exception as e:  
 logger.error(f'Calibration failed: {e}')  
 arduino.disconnect()  
 del arduino  
  
  
def \_rfid\_offset\_calib():  
 try:  
 logger.info(f'\033[1;33mStarting the RFID taring process.\033[0m')  
 arduino = ADC.ArduinoSerial(ARDUINO\_PORT, 9600, timeout=1)  
 arduino.connect()  
 offset = arduino.calib\_read\_mediana()  
 arduino.set\_offset(offset)  
 config\_manager.update\_setting("Calibration", "Offset", offset)  
 logger.info(f'Calibration details\n\n —Offset: {offset}')  
 arduino.disconnect()  
 del arduino  
 logger.info(f'\033[1;33mRFID taring process finished succesfully.\033[0m')  
 except:  
 logger.error(f'RFID taring process Failed')  
 arduino.disconnect()  
  
  
def \_rfid\_scale\_calib():  
 try:  
 logger.info(f'\033[1;33mStarting the RFID scale calibration process.\033[0m')  
 logger.info(f'\033There should be {CALIBRATION\_WEIGHT} kg.\033[')  
 arduino = ADC.ArduinoSerial(ARDUINO\_PORT, 9600, timeout=1)  
 arduino.connect()  
 offset = float(config\_manager.get\_setting("Calibration", "Offset"))  
 mediana = arduino.calib\_read\_mediana()  
 logger.info(f'Mediana: {mediana}\noffset: {offset}')  
 measured\_weight = (mediana - offset)  
 logger.info(f'measured\_weight: {measured\_weight}\nCALIBRATION\_WEIGHT: {CALIBRATION\_WEIGHT}')  
 scale = measured\_weight/CALIBRATION\_WEIGHT  
 logger.info(f'calibration weight is: {CALIBRATION\_WEIGHT}')  
 arduino.set\_scale(scale)  
 config\_manager.update\_setting("Calibration", "Scale", scale)  
 logger.info(f'Calibration details\n\n —Scale factor: {scale}')  
 arduino.disconnect()  
 del arduino  
 logger.info(f'\033[1;33mRFID scale calibration process finished succesfully.\033[0m')  
 except:  
 logger.error(f'calibrate Fail')  
 arduino.disconnect()  
  
  
def \_calibrate\_or\_start():  
 try:  
 logger.info(f'\nTo calibrate the equipment, put a tick in the settings to calibration mode:\nActaul state is {"CALIBRATION\_ON" if CALIBRATION\_MODE else "CALIBRATION\_OFF"}')  
 if CALIBRATION\_MODE:  
 \_\_calibrate(timeout=120)  
  
 except Exception as e:  
 logger.error(f'Calibrate or start Error: {e}')  
  
  
def \_\_animal\_rfid():  
 try:  
 if RFID\_READER\_USB:  
 rfid\_reader = RFIDReader()  
 return rfid\_reader.connect()  
 else:  
 cow\_id = \_\_connect\_rfid\_reader\_ethernet()   
 if cow\_id is not None:  
 logger.info(f'cow\_id\_\_animal\_rfid: {cow\_id}')   
 return cow\_id  
 except Exception as e:  
 logger.error(f'RFID reader error: {e}')  
  
  
def \_\_process\_calibration(animal\_id):  
 try:  
 if RFID\_CABLIBRATION\_MODE:  
 if animal\_id == CALIBRATION\_TARING\_RFID:  
 \_rfid\_offset\_calib()  
 return True  
 elif animal\_id == CALIBRATION\_SCALE\_RFID:  
 \_rfid\_scale\_calib()   
 return True   
 return False  
 except Exception as e:  
 logger.error(f'Calibration with RFID: {e}')  
  
  
def scales\_v71():  
 try:  
 \_calibrate\_or\_start()  
 if RFID\_READER\_USB == False:  
 \_set\_power\_RFID\_ethernet()  
   
 sql\_db = SqlDatabase(db\_path='sql\_table.db')  
 last\_internet\_check = time.time()  
  
 while True:  
 cow\_id = \_\_animal\_rfid() # Считывание меток  
 if cow\_id is not None:  
 logger.info(f'scales\_v71\_cow\_id: {cow\_id}')   
 calib\_id = \_\_process\_calibration(cow\_id)   
   
 if calib\_id == False and cow\_id != None:   
 arduino = start\_obj() # Создаем объект  
 time.sleep(1) # задержка для установления связи между rasp и arduino  
   
 weight\_finall, weight\_array, weighing\_start\_time, most\_common\_animal\_id = measure\_weight(arduino, cow\_id)   
   
 logger.info("main: weight\_finall", weight\_finall)   
 weighing\_end\_time = str(datetime.now()) # Время окончания измерения  
  
 if str(weight\_finall) > '0':  
 logger.info("main: Send data to server")  
 post\_array\_data(TYPE\_SCALES, most\_common\_animal\_id, weight\_array, weighing\_start\_time, weighing\_end\_time, sql\_db)  
 post\_median\_data(most\_common\_animal\_id, weight\_finall, TYPE\_SCALES, sql\_db) # Send data to server by JSON post request  
 arduino.disconnect() # Закрываем связь  
  
 current\_time = time.time()  
 if current\_time - last\_internet\_check > INTERNET\_CHECK\_INTERVAL:  
 sql\_db.internet\_on()  
 last\_internet\_check = current\_time  
  
 except KeyboardInterrupt as e:  
 arduino.disconnect()  
 logger.error(f'Bye bye: {e}')  
   
 except Exception as k:  
 arduino.disconnect()  
 logger.error(f'Main error: {k}')  
  
  
def is\_valid\_rfid(animal\_id):  
 """  
 Проверка, что animal\_id выглядит как более-менее адекватный  
 """  
 return (  
 animal\_id and # не None и не пустая строка  
 len(animal\_id) >= 8 and # хотя бы 8 символов  
 len(animal\_id) <= 64 and # ограничим максимум  
 any(c.isalnum() for c in animal\_id) # хотя бы одна буква или цифра  
 )  
  
  
def \_take\_weight(weight, count = 50) -> float:  
 try:  
 weight.clean\_arr() # Очистим массив перед стартом  
 for \_ in range(count): # Например, взять 50 значений  
 weight.calc\_mean()  
 logger.info(weight.get\_arr())  
 time.sleep(0.01) # Делаем паузу, чтобы усреднить медленнее  
  
 #logger.info(f'ARRAY {weight.get\_arr()}')  
 return sum(weight.get\_arr()) / len(weight.get\_arr())  
 except Exception as e:  
 logger.error(f'Error \_take\_weight: {e}')  
  
  
def measure\_weight(obj, cow\_id: str) -> tuple:  
 try:  
 animal\_id\_list = []  
 animal\_id\_list.append(cow\_id)  
 weight\_arr = []  
 start\_filter(obj)  
 next\_time = time.time() + 1  
   
 drink\_start\_time = timeit.default\_timer()  
 gpio\_state = False  
 start\_timedate = str(datetime.now())  
   
 values = value\_data.Values(  
 drink\_start\_time, 0, TYPE\_SCALES, cow\_id, 0, '0', 0, 0, 0, 0, True  
 )  
  
 if SPRAYER:  
 sprayer = Sprayer(values)  
   
 weight\_on\_moment = \_take\_weight(obj, 20)  
 logger.info(f'Weight on the moment: {weight\_on\_moment}')  
  
 while weight\_on\_moment > 20:  
  
 current\_animal\_id = \_\_animal\_rfid()  
 if is\_valid\_rfid(current\_animal\_id):  
 animal\_id\_list.append(current\_animal\_id)  
 logger.info(f"RFID added to list: {current\_animal\_id}")  
 else:  
 logger.warning(f"Ignored suspicious RFID: {current\_animal\_id}")  
  
 weight\_on\_moment = \_take\_weight(obj, 20)  
 current\_time = time.time()  
 time\_to\_wait = next\_time - current\_time  
  
 if SPRAYER:  
 if not values.flag:  
 gpio\_state = sprayer.spray\_main\_function(gpio\_state)  
 values = sprayer.new\_start\_timer(gpio\_state)  
 else:  
 if time\_to\_wait < 0 and round(time.time(), 0) % 5 == 0:  
 values.flag = False  
  
 if time\_to\_wait < 0:  
 weight\_arr.append(weight\_on\_moment)  
 next\_time = time.time() + 1  
 logger.debug(f'Array weights: {weight\_arr}')  
  
   
  
 GPIO.cleanup()  
  
 if not weight\_arr:  
 logger.info("null weight list")  
 return 0, [], '', ''  
  
 most\_common\_animal\_id = None  
 most\_common\_animal\_id = Counter(animal\_id\_list).most\_common(1)[0][0] if animal\_id\_list else "UNKNOWN"  
 weight\_finall = statistics.median(weight\_arr)  
 if SPRAYER:  
 gpio\_state = sprayer.gpio\_state\_check(gpio\_state)  
  
 return weight\_finall, weight\_arr, start\_timedate, most\_common\_animal\_id  
  
 except Exception as e:  
 logger.error(f'measure\_weight Error: {e}')  
 GPIO.cleanup()  
 return 0, [], ''

# \_sprayer.py

import requests  
from requests.exceptions import HTTPError  
try:  
 import RPi.GPIO as GPIO  
except RuntimeError:  
 from \_\_gpio\_simulator import MockGPIO as GPIO  
  
import timeit  
import json  
from loguru import logger  
from \_config\_manager import ConfigManager  
from datetime import datetime, timezone  
  
config\_manager = ConfigManager()   
  
class Sprayer:  
 def \_\_init\_\_(self, values):  
 self.values = values  
 self.spray\_post = config\_manager.get\_setting("Sprayer", "post\_url")  
 self.medicine\_pin = int(config\_manager.get\_setting("Sprayer", "medicine\_pin"))  
 self.paint\_pin = int(config\_manager.get\_setting("Sprayer", "paint\_pin"))  
 self.lmin = float(config\_manager.get\_setting("Sprayer", "l/min"))  
   
 def spray\_gpio\_off(self) -> bool:  
 try:  
 logger.info("Start spray\_gpio\_off")  
 GPIO.setmode(GPIO.BCM)  
 GPIO.setwarnings(True)  
 GPIO.setup(int(self.values.pin), GPIO.OUT)  
 GPIO.output(int(self.values.pin), GPIO.LOW)  
 GPIO.cleanup()  
 position = False  
   
 if int(self.values.task\_id) != int(0):  
 end\_time = timeit.default\_timer()  
 self.values.new\_volume = int((end\_time - self.values.drink\_start\_time)/60 \* self.lmin \* 1000)  
 post\_data = self.spray\_json\_payload()  
 headers = {'Content-type': 'application/json'}   
 post\_res = requests.post(self.spray\_post, data=json.dumps(post\_data), headers=headers, timeout=30)  
 logger.info(f'Post status code {post\_res.status\_code}')  
 logger.info(f'GPIO is off. Pin number is {self.values.pin}')  
   
 return position  
 except Exception as e:  
 logger.error(f"Error: Spray\_GPIO\_off function isn't working {e}")  
 return False  
  
 def spray\_gpio\_on(self) -> bool:  
 try:  
 logger.info('Start spray\_gpio\_on')  
 GPIO.setmode(GPIO.BCM)  
 GPIO.setwarnings(True)  
 GPIO.setup(int(self.values.pin), GPIO.OUT)  
 GPIO.output(int(self.values.pin), GPIO.HIGH)  
 logger.info(f'Pump turn on is successful. Pin number is {self.values.pin}')  
 return True  
 except Exception as e:  
 logger.error(f"Error: GPIO\_on function isn't working {e}")  
 self.spray\_gpio\_off()  
 return False  
  
  
 def spray\_json\_payload(self) -> dict:  
 try:  
 logger.debug("Start spray\_json\_payload function")  
 now = datetime.now(timezone.utc)  
 iso\_string = now.isoformat(timespec='milliseconds').replace('+00:00', 'Z')  
  
 data = {  
 "EventDate": iso\_string,  
 "TaskId": self.values.task\_id,  
 "ScalesSerialNumber": self.values.type\_scales,  
 "SpayerSerialNumber": "s01000001",  
 "RFIDNumber": self.values.cow\_id,  
 "SprayingType": self.values.spraying\_type,  
 "Volume": self.values.new\_volume  
 }  
 logger.info(f'Sprayer data to server: {data}')  
 return data  
 except Exception as e:  
 logger.error(f"Error in spray\_json\_payload function: {e}")  
 return {}  
  
 def request\_get(self):  
 try:  
 cow\_id = self.values.cow\_id  
 type\_scales = self.values.type\_scales  
 url = 'https://smart-farm.kz:8502/api/v2/Sprayings?scalesSerialNumber=' + type\_scales + \  
 '&animalRfidNumber=' + cow\_id  
 request\_get = requests.get(url, timeout=5).json()  
 return request\_get  
 except Exception as e:  
 logger.error(f'request\_get function error: {e}')  
 return {}  
  
 def spray\_json\_get(self, request\_get\_json, get\_object=0):  
 try:  
 logger.info("Start spray\_json\_get")  
 self.values.task\_id = request\_get\_json[get\_object]['TaskId']  
 self.values.spraying\_type = request\_get\_json[get\_object]['SprayingType']  
 self.values.volume = request\_get\_json[get\_object]['Volume']  
 self.values.server\_time = request\_get\_json[get\_object]['ServerTime']  
 return self.values  
 except BaseException as b:  
 logger.error(f"spray\_json\_get error: {b}")  
 return self.values  
  
 def spray(self, position) -> bool:  
 try:  
 logger.info("Start spray")  
 spray\_time = (self.values.volume/1000)/self.lmin \* 60 # 10  
 self.values.spray\_duration = self.values.drink\_start\_time + spray\_time # current + 10  
 if self.values.spray\_duration >= timeit.default\_timer():  
 logger.info(f'Position is {position}')  
 if not position:  
 position = self.spray\_gpio\_on()  
 return position  
 else:  
 return position  
 else:  
 logger.info('TimeOff')  
 position = self.spray\_gpio\_off()  
 return position  
 except ValueError as err:  
 logger.error(f'Other error occurred: {err}')  
 self.spray\_gpio\_off()  
 return False  
  
 def spray\_timer(self, position) -> bool:  
 try:  
 logger.info("Start spray timer check function")  
 if self.values.spray\_duration >= timeit.default\_timer():  
 logger.info(f'Time is {self.values.spray\_duration} {timeit.default\_timer()}')  
 if not position:  
 self.spray\_gpio\_on()  
 position = True  
 return position  
 else:  
 return position  
 else:  
 logger.info('TimeOff')  
 position = self.spray\_gpio\_off()  
 return position  
 except ValueError as err:  
 logger.error(f'Other error occurred: {err}')  
 self.spray\_gpio\_off()  
 return False  
  
 def spraying\_type(self) -> int:  
 try:  
 logger.info('Spraying type function start')  
 if self.values.spraying\_type == 0:  
 return self.medicine\_pin  
 else:  
 return self.paint\_pin  
 except Exception as e:  
 logger.error(f'spraying\_type function error: {e}')  
 return -1  
  
 def spray\_main\_function(self, position) -> bool:  
 try:  
 logger.info("Start spray\_main\_function")  
 if not position:  
 request\_get\_json = self.request\_get()  
 logger.debug(f'JSON Request: {request\_get\_json}')  
 if not request\_get\_json:  
 logger.info('No tasks there')  
 self.values.flag = True  
 return position  
 else:  
 self.values = self.spray\_json\_get(request\_get\_json)  
 self.values.pin = self.spraying\_type()  
 self.values.drink\_start\_time = timeit.default\_timer()  
 logger.debug(f'Values are: {self.values}')  
 position = self.spray(position)  
 return position  
 else:  
 position = self.spray\_timer(position)  
 return position  
 except HTTPError as http\_err:  
 logger.error(f'HTTP error occurred: {http\_err}')  
 self.spray\_gpio\_off()  
 return False  
 except Exception as err:  
 logger.error(f'Other error occurred: {err}')  
 self.spray\_gpio\_off()  
 return False  
  
 def gpio\_state\_check(self, position) -> bool:  
 try:  
 logger.info("Start gpio\_state\_check")  
 if position:  
 position = self.spray\_gpio\_off()  
 return position  
 except HTTPError as http\_err:  
 logger.error(f'HTTP error occurred: {http\_err}')  
 self.spray\_gpio\_off()  
 return False  
 except Exception as err:  
 logger.error(f'Other error occurred: {err}')  
 self.spray\_gpio\_off()  
 return False  
  
 def new\_start\_timer(self, position):  
 try:  
 logger.info('New start timer function')  
 if position:  
 return self.values  
 else:  
 self.values.drink\_start\_time = timeit.default\_timer()  
 return self.values  
 except ValueError as e:  
 logger.error(f'new\_start\_timer function error: {e}')  
 return self.values

# \_sql\_database.py

import sqlite3  
import os  
from datetime import datetime  
import json  
import requests  
from loguru import logger  
from \_config\_manager import ConfigManager  
  
  
class SqlDatabase:  
 def \_\_init\_\_(self, db\_path=None):  
 self.config\_manager = ConfigManager()  
 self.\_\_url\_median = self.config\_manager.get\_setting("Parameters", "median\_url")  
 self.\_\_url\_array = self.config\_manager.get\_setting("Parameters", "array\_url")  
 self.\_\_headers = {'Content-type': 'application/json'}  
  
 if db\_path is None:  
 script\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))  
 self.\_\_sql\_table\_path = os.path.join(script\_dir, 'sql\_table.db')  
 else:  
 self.\_\_sql\_table\_path = db\_path  
  
 self.\_\_table\_check()  
  
 def no\_internet(self, payload):  
 """Автоматически определяет, в какую таблицу сохранить данные"""  
 if not payload:  
 logger.error('SqlDatabase no\_internet: Empty payload')  
 return  
  
 if "AnimalNumber" in payload:  
 table\_name = "json\_data\_median"  
 elif "ScalesSerialNumber" in payload:  
 table\_name = "json\_data\_array"  
 else:  
 logger.error('SqlDatabase no\_internet: Unknown data format')  
 return  
  
 self.\_\_insert\_data(payload, table\_name)  
  
 def internet\_on(self):  
 """Проверить интернет и попытаться отправить сохраненные данные."""  
 try:  
 # Проверка наличия интернета перед отправкой  
 requests.get("https://www.google.com", timeout=5)  
  
 # Если интернет есть, пробуем отправить все накопленные данные  
 self.\_\_send\_saved\_data("json\_data\_median", self.\_\_url\_median)  
 self.\_\_send\_saved\_data("json\_data\_array", self.\_\_url\_array)  
 except requests.RequestException:  
 logger.warning('Internet is not available. Data will remain in the database.')  
  
 def \_\_table\_check(self):  
 """Создать таблицы, если не существует"""  
 try:  
 with sqlite3.connect(self.\_\_sql\_table\_path) as db:  
 sql = db.cursor()  
 sql.execute("""  
 CREATE TABLE IF NOT EXISTS json\_data\_median (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 AnimalNumber TEXT,  
 Date TEXT,  
 Weight REAL,  
 ScalesModel TEXT  
 )""")  
 sql.execute("""  
 CREATE TABLE IF NOT EXISTS json\_data\_array (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 ScalesSerialNumber TEXT,  
 WeighingStart TEXT,  
 WeighingEnd TEXT,  
 RFIDNumber TEXT,  
 Data TEXT  
 )""")  
 db.commit()  
 except Exception as e:  
 logger.error(f'SqlDatabase \_\_table\_check: {e}')  
  
 def \_\_insert\_data(self, payload, table\_name):  
 """Сохранить данные в базу"""  
 try:  
 with sqlite3.connect(self.\_\_sql\_table\_path) as db:  
 sql = db.cursor()  
 values = self.\_\_table\_values\_convert(payload, table\_name)  
 if table\_name == "json\_data\_median":  
 sql.execute(  
 """INSERT INTO json\_data\_median   
 (AnimalNumber, Date, Weight, ScalesModel)   
 VALUES (?, ?, ?, ?);""",  
 values  
 )  
 elif table\_name == "json\_data\_array":  
 sql.execute(  
 """INSERT INTO json\_data\_array   
 (ScalesSerialNumber, WeighingStart, WeighingEnd, RFIDNumber, Data)   
 VALUES (?, ?, ?, ?, ?);""",  
 values  
 )  
 db.commit()  
 logger.info(f"Data saved locally in {table\_name}: {values}")  
 except Exception as e:  
 logger.error(f'SqlDatabase \_\_insert\_data: {e}')  
  
 def \_\_table\_values\_convert(self, payload, table\_name):  
 if table\_name == "json\_data\_median":  
 return (payload['AnimalNumber'], payload["Date"], payload["Weight"], payload["ScalesModel"])  
 elif table\_name == "json\_data\_array":  
 return (payload['ScalesSerialNumber'], payload["WeighingStart"], payload["WeighingEnd"], payload["RFIDNumber"], json.dumps(payload["Data"]))  
  
 def \_\_db\_row\_to\_json(self, db\_row, table\_name):  
 if table\_name == "json\_data\_median":  
 return {  
 "AnimalNumber": db\_row[1],  
 "Date": db\_row[2],  
 "Weight": db\_row[3],  
 "ScalesModel": db\_row[4]  
 }  
 elif table\_name == "json\_data\_array":  
 return {  
 "ScalesSerialNumber": db\_row[1],  
 "WeighingStart": db\_row[2],  
 "WeighingEnd": db\_row[3],  
 "RFIDNumber": db\_row[4],  
 "Data": json.loads(db\_row[5])  
 }  
  
 def \_\_take\_all\_data(self, table\_name):  
 try:  
 with sqlite3.connect(self.\_\_sql\_table\_path) as db:  
 sql = db.cursor()  
 sql.execute(f"SELECT \* FROM {table\_name}")  
 return sql.fetchall()  
 except Exception as e:  
 logger.error(f'SqlDatabase \_\_take\_all\_data: {e}')  
 return []  
  
 def \_\_delete\_saved\_data(self, id, table\_name):  
 try:  
 with sqlite3.connect(self.\_\_sql\_table\_path) as db:  
 sql = db.cursor()  
 sql.execute(f"DELETE FROM {table\_name} WHERE id = ?", (id,))  
 db.commit()  
 except Exception as e:  
 logger.error(f'SqlDatabase \_\_delete\_saved\_data: {e}')  
  
 def \_\_send\_saved\_data(self, table\_name, url):  
 """Отправить все сохранённые данные"""  
 try:  
 all\_data = self.\_\_take\_all\_data(table\_name)  
 for row in all\_data:  
 id = row[0]  
 post\_data = self.\_\_db\_row\_to\_json(row, table\_name)  
 response = requests.post(url, data=json.dumps(post\_data), headers=self.\_\_headers, timeout=5)  
 if response.status\_code == 200:  
 self.\_\_delete\_saved\_data(id, table\_name)  
 logger.info(f'Successfully sent data from {table\_name}. ID: {id}')  
 else:  
 logger.warning(f'Failed to send data from {table\_name}. ID: {id}. Status code: {response.status\_code}')  
 except Exception as e:  
 logger.error(f'SqlDatabase \_\_send\_saved\_data: {e}')

# \_values\_class.py

#!/usr/bin/python3  
  
"""File containing the main variables for automatic control of animal spraying.  
Author: Suieubayev Maxat  
Contact: maxat.suieubayev@gmail.com  
Number: +7 775 818 48 43"""  
  
from dataclasses import dataclass  
from loguru import logger  
  
@dataclass(frozen=True)  
class Pin:  
 pcf\_model\_5: list = 40, 22  
 pcf\_model\_6: list = 40, 32  
 pcf\_model\_7: list = 40, 43  
 pcf\_model\_10: list = 40, 54  
  
  
@dataclass  
class Values:  
 drink\_start\_time: float  
 spray\_duration: int  
 type\_scales: str  
 cow\_id: str  
 pin: int  
 server\_time: str  
 task\_id: int  
 new\_volume: int  
 spraying\_type: int  
 volume: float  
 flag: bool  
  
 def default(self):  
 self.drink\_start\_time = 0  
 self.spray\_duration = 0  
 self.type\_scales = ''  
 self.cow\_id = ''  
 self.pin = 0  
 self.server\_time = ''  
 self.task\_id = 0  
 self.new\_volume = 0  
 self.spraying\_type = 0  
 self.volume = 0  
 return self

# main\_pcf.py

#!/usr/bin/python3   
#/home/pi/scales7.1/scales\_submodule/src/main\_pcf.py  
  
"""Scales main file. Additional function: sprayer. Version 7.1  
Author: Suieubayev Maxat.  
Contact number +7 775 818 48 43. Email maxat.suieubayev@gmail.com"""  
  
from \_headers import install\_packages  
  
requirement\_list = ['loguru',   
 'requests',   
 'pyserial',  
 'RPi.GPIO',   
 'wabson.chafon-rfid']  
install\_packages(requirement\_list)  
  
from \_lib\_pcf import scales\_v71  
from loguru import logger  
from \_config\_manager import ConfigManager  
import os  
try:  
 import RPi.GPIO  
 log\_dir = "/home/pi/scales7.1/scales\_submodule/loguru/scales\_log"  
 error\_log\_dir = f"{log\_dir}/error\_log"  
 os.makedirs(log\_dir, exist\_ok=True)  
 os.makedirs(error\_log\_dir, exist\_ok=True)  
 os.system(f"chmod -R 777 {log\_dir}")  
  
except RuntimeError:  
 from \_\_gpio\_simulator import MockGPIO  
 log\_dir = '../feeder\_log'  
from \_glb\_val import DEBUG  
  
config\_manager = ConfigManager()  
debug\_level = "DEBUG" if DEBUG == 1 else "CRITICAL"  
  
"""Инициализация logger для хранения записи о всех действиях программы"""  
logger.add(f"{log\_dir}/scales.log", format="{time} {level} {function} {message}",   
level=debug\_level, rotation="1 day", retention= '1 month', compression="zip")   
  
"""Инициализация logger для хранения записи об ошибках программы"""  
logger.add(f"{error\_log\_dir}/errors.log", format="{time} {level} {file} {function}:{line} {message}",   
level="ERROR", rotation="1 day", retention= '1 month', compression="zip") # Настройка логгера  
  
@logger.catch() # Показывает ошибки, не работает если их обрабатывать  
def main():  
 try:  
 scales\_v71()  
 except Exception as e:   
 logger.error(f'Error: {e}')  
   
  
main()

**Приложение 4**

Тестовые скрипты проекта

# adder\_calibration.py

#!/usr/bin/env python3  
import sys  
import time  
from pathlib import Path  
from loguru import logger  
  
import statistics  
from collections import deque  
from pathlib import Path  
from loguru import logger  
from time import sleep  
  
# Добавляем src в PYTHONPATH  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_config\_manager import ConfigManager  
import \_lib\_pcf as lib  
from \_adc\_data import ArduinoSerial # именно этот класс  
# from \_chafon\_rfid\_lib import RFIDReader # не нужен здесь  
  
PRESENCE\_THRESHOLD = 20.0 # кг — порог присутствия животного  
READ\_PERIOD = 0.1 # секунда между считываниями  
PRESENCE\_COUNT\_THRESHOLD = 5 # >threshold подряд → корова встала (0.5 с)  
ABSENCE\_COUNT\_THRESHOLD = 5 # <threshold подряд → корова ушла (0.5 с)  
WINDOW\_SIZE = 10 # сколько последних замеров держать в буфере  
  
config\_manager = ConfigManager()  
  
  
def calibrate(  
 scale\_dev: ArduinoSerial,  
 cfg: ConfigManager,  
 timeout: float = 120.0,  
 samples\_zero: int = 20,  
 samples\_span: int = 20  
):  
 """  
 Калибровка по двум шагам:  
 1) Смещение (offset)  
 2) Чувствительность (scale)  
 """  
 deadline = time.time() + timeout  
  
 try:  
 scale\_dev.connect()  
  
 # === Шаг 1: Offset ===  
 logger.info("=== Шаг 1: калибровка нуля ===")  
 logger.info("Убедитесь, что платформа пуста и нажмите Enter")  
 lib.\_\_input\_with\_timeout(  
 deadline - time.time()  
 )  
 raw\_zero = scale\_dev.calib\_read\_average(samples\_zero)  
 scale\_dev.set\_offset(raw\_zero)  
 logger.info(f"raw\_zero = {raw\_zero:.2f}")  
  
 # === Шаг 2: Span ===  
 logger.info("=== Шаг 2: калибровка чувствительности ===")  
 logger.info("Положите эталонный груз на платформу и нажмите Enter")  
 lib.\_\_input\_with\_timeout(  
 deadline - time.time()  
 )  
  
 raw\_with\_weight = scale\_dev.calib\_read\_average(samples\_span)  
 delta = raw\_with\_weight - scale\_dev.get\_offset()  
 logger.info(f"raw\_with\_weight = {raw\_with\_weight:.2f}, Δ = {delta:.2f}")  
  
 if delta <= 0:  
 raise RuntimeError("Ошибка span: Δ должно быть > 0. Проверьте подключение и массу.")  
  
 logger.info("Введите массу эталонного груза в кг (например, 20):")  
 weight\_str = lib.\_\_input\_with\_timeout(  
 deadline - time.time()  
 )  
 try:  
 weight\_kg = float(weight\_str)  
 if weight\_kg <= 0:  
 raise ValueError  
 except ValueError:  
 raise ValueError("Неверный ввод массы, ожидается положительное число.")  
  
 scale\_factor = delta / weight\_kg  
 scale\_dev.set\_scale(scale\_factor)  
 logger.info(f"scale\_factor = {scale\_factor:.6f} raw/кг")  
  
 # === Сохранение в конфиг ===  
 # В конфиг пишем именно числовые строки, без предварительного форматирования  
 cfg.update\_setting("Calibration", "Offset", raw\_zero)  
 cfg.update\_setting("Calibration", "Scale", scale\_factor)  
  
 logger.success("Калибровка успешно завершена")  
 logger.info(f"Итоги: offset={raw\_zero:.2f}, scale={scale\_factor:.6f}")  
  
 except TimeoutError:  
 logger.error("Калибровка прервана по таймауту ввода")  
 except Exception as e:  
 logger.exception(f"Ошибка во время калибровки: {e}")  
 finally:  
 scale\_dev.disconnect()  
  
  
def main():  
 # Получаем объект для работы с HX711  
 try:  
 y = input("Для начала введите 'y' и нажмите Enter")  
 if y == 'y':  
 logger.info("Старт калибровки")  
 arduino = lib.start\_obj()  
 if not isinstance(arduino, ArduinoSerial):  
 logger.error("start\_obj() вернул не тот тип устройства для калибровки")  
 logger.info("Старт калибровки сумматора")  
 calibrate(arduino, config\_manager)   
   
 arduino = lib.start\_obj()  
 window\_buf = deque(maxlen=WINDOW\_SIZE)  
 weight\_arr = []  
 collecting = False  
 logger.info("Ожидаем, пока корова встанет на весы...")  
  
 while True:  
 # 3) Снимаем показание  
 w = arduino.get\_measure\_2()  
 logger.debug(f"Текущий вес: {w:.2f} кг")  
  
 if not collecting:  
 # Ждём прихода: считаем подряд чтения ≥ threshold  
 if w >= PRESENCE\_THRESHOLD:  
 presence\_count += 1  
 if presence\_count >= PRESENCE\_COUNT\_THRESHOLD:  
 collecting = True  
 absence\_count = 0  
 weight\_arr.clear()  
 logger.info("Корова встала — начинаем сбор данных.")  
 else:  
 presence\_count = 0  
  
 else:  
 # Корова на весах: собираем пока она не уедет  
 if w >= PRESENCE\_THRESHOLD:  
 absence\_count = 0  
 weight\_arr.append(round(w,2))  
 logger.debug(f" Собираем: {w:.2f} кг")  
 else:  
 absence\_count += 1  
 if absence\_count >= ABSENCE\_COUNT\_THRESHOLD:  
 # Корова ушла — заканчиваем  
 final\_weight = statistics.median(weight\_arr) if weight\_arr else 0.0  
 logger.info(f"Корова ушла — итоговый вес: {final\_weight:.2f} кг")  
 break  
  
 sleep(READ\_PERIOD)  
  
 # 4) Дальнейшая обработка  
 # здесь можно, например, отправить final\_weight на сервер  
 # или сохранить весь массив weight\_arr  
 logger.info(f"Собранные замеры ({len(weight\_arr)}): {weight\_arr}")  
  
 except KeyboardInterrupt:  
 logger.info("Калибровка прервана пользователем")  
 except Exception as e:  
 logger.exception(f"Ошибка во время калибровки: {e}")  
 finally:  
 arduino.disconnect()  
 logger.info("Отключение от устройства")  
 logger.info("Сохранение конфигурации")  
 logger.info("Выход из скрипта калибровки")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# calc\_volume.py

def calc\_volume():  
 print('Please input volume: ')  
 volume = float(input())  
 lm = 5  
 print(f'l/m = {lm}')  
 print(f'Time = {(volume/1000)/lm \* 60} sec')  
  
  
def test\_try():  
 print("TEST")  
 weight\_array = [50 for i in range(10)]  
 print(weight\_array)  
  
def main():  
 test\_try()  
  
main()

# chafon\_test.py

import time  
import binascii  
from chafon\_rfid.transport\_serial import SerialTransport  
from chafon\_rfid.uhfreader288m import G2InventoryCommand, G2InventoryResponseFrame  
from loguru import logger  
  
DEVICE\_PORT = "/dev/ttyUSB0" # или 'COM3' на Windows  
BAUD\_RATE = 57600 # Обычно 57600 или 115200  
READ\_TIMEOUT = 1.0 # Секунды ожидания ответа  
  
def main():  
 logger.info("Connecting to USB RFID reader...")  
  
 try:  
 transport = SerialTransport(device=DEVICE\_PORT, baud\_rate=BAUD\_RATE, timeout=READ\_TIMEOUT)  
 logger.success(f"Connected to {DEVICE\_PORT} at {BAUD\_RATE} baud.")  
  
 inventory\_command = G2InventoryCommand(q\_value=4, antenna=0x80)  
  
 while True:  
 logger.debug("Sending inventory command...")  
 transport.write(inventory\_command.serialize())  
  
 try:  
 response\_data = transport.read\_frame()  
 frame = G2InventoryResponseFrame(response\_data)  
  
 for tag in frame.get\_tag():  
 epc\_hex = tag.epc.hex()  
 logger.success(f"Tag EPC: {epc\_hex}")  
  
 except Exception as e:  
 logger.warning(f"No tag or error: {e}")  
  
 time.sleep(0.1)  
  
 except Exception as e:  
 logger.error(f"Error opening transport: {e}")  
  
 finally:  
 logger.info("Closing transport...")  
 transport.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# gpio\_test.py

#!/usr/bin/python3  
  
import RPi.GPIO as GPIO  
import time  
import sys  
from pathlib import Path  
  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_config\_manager import ConfigManager  
  
  
config\_manager = ConfigManager()  
  
medicine\_pin = int(config\_manager.get\_setting("Sprayer", "medicine\_pin"))  
  
GPIO.setmode(GPIO.BCM) # Устанавливаем BCM-нумерацию пинов  
GPIO.setup(medicine\_pin, GPIO.OUT) # Устанавливаем 18-й пин как выход  
GPIO.output(medicine\_pin, GPIO.LOW)  
  
def main():  
 try:  
 print("Enter On time: ")  
 on\_time = input()  
 print("Enter off time: ")  
 off\_time = input()  
 count = 0  
 while(True):  
 print("On")  
 GPIO.output(medicine\_pin, GPIO.HIGH) # Подаём 3.3 вольта на 18-й пин  
 #GPIO.output(23, GPIO.HIGH)  
 time.sleep(float(on\_time)) # Ждём одну секунду  
 count += 1  
 print("off")  
 GPIO.output(medicine\_pin, GPIO.LOW) # Подаём 3.3 вольта на 18-й пин  
 #GPIO.output(23, GPIO.LOW)  
 time.sleep(float(off\_time))  
 except KeyboardInterrupt as e:  
 print("Ok! Bye!")  
 print("Count of cycles: ", count)  
 GPIO.cleanup() # Возвращаем пины в исходное состояние  
  
main()

# post\_test.py

import sys  
from pathlib import Path  
  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_config\_manager import ConfigManager  
import time  
import timeit  
from datetime import datetime, timedelta  
import statistics  
try:  
 import RPi.GPIO as GPIO  
except RuntimeError:  
 from \_\_gpio\_simulator import MockGPIO as GPIO  
  
  
from \_sprayer import Sprayer  
import \_values\_class as value\_data  
from \_glb\_val import \*  
import json  
import json  
import requests  
   
from loguru import logger  
  
debug\_level = "DEBUG" if DEBUG == 1 else "CRITICAL"  
  
"""Инициализация logger для хранения записи о всех действиях программы"""  
logger.add(sys.stdout, format="{time} {level} {message}", level=debug\_level)   
  
config\_manager = ConfigManager()  
  
def post\_median\_data(animal\_id, weight\_finall, type\_scales): # Sending data into Igor's server through JSON  
 try:  
 logger.debug(f'START SEND DATA TO SERVER:')  
 url = config\_manager.get\_setting("Parameters", "median\_url")  
 headers = {'Content-type': 'application/json'}  
 data = {"AnimalNumber" : animal\_id,  
 "Date" : str(datetime.now()),  
 "Weight" : weight\_finall,  
 "ScalesModel" : type\_scales}  
 answer = requests.post(url, data=json.dumps(data), headers=headers, timeout=3)  
 logger.debug(f'Answer from server: {answer}') # Is it possible to stop on this line in the debug?  
 logger.debug(f'Content from main server: {answer.content}')  
 except Exception as e:  
 logger.error(f'Error send data to server {e}')  
  
  
def post\_array\_data(type\_scales, animal\_id, weight\_list, weighing\_start\_time, weighing\_end\_time):  
 try:  
 logger.debug(f'Post data function start')  
 url = config\_manager.get\_setting("Parameters", "array\_url")  
 headers = {'Content-Type': 'application/json; charset=utf-8'}  
 data = {  
 "ScalesSerialNumber": type\_scales,  
 "WeighingStart": weighing\_start\_time,  
 "WeighingEnd": weighing\_end\_time,  
 "RFIDNumber": animal\_id,  
 "Data": weight\_list  
 }   
 post = requests.post(url, data=json.dumps(data), headers=headers, timeout=3)  
 logger.debug(f'Post Data: {data}')  
 logger.debug(f'Answer from server: {post}') # Is it possible to stop on this line in the debug?  
 logger.debug(f'Content from main server: {post.content}')  
 except Exception as e:  
 logger.error(f'Error post data: {e}')  
  
  
def main():  
 cow\_id = '940000401030\_test'  
 weight\_array = [50.00 for i in range(60)]  
 #weight\_array = [50.06, 49.94, 49.74, 49.76, 49.87, 49.93, 50.02, 49.97, 49.95, 49.98, 49.88, 49.91, 50.2, 179.95, 668.44, 1274.77, 1899.45, 2517.68, 3087.39, 3368.32, 3367.92, 3367.45]  
 weighing\_start\_time = str(datetime.now())  
 weighing\_end\_time = str(datetime.now() + timedelta(seconds=30))  
 weight\_finall = statistics.median(weight\_array)  
 post\_array\_data(TYPE\_SCALES, cow\_id, weight\_array, weighing\_start\_time, weighing\_end\_time)  
 post\_median\_data(cow\_id, weight\_finall, TYPE\_SCALES)   
  
  
main()

# request.py

import requests  
from loguru import logger  
import socket  
import binascii  
  
  
@logger.catch  
def \_\_request\_get(): # Get data from the server  
 try:  
 cow\_id = "FC6599A1DF00" # Это бирка  
 type\_scales = "scales0623v61-6-rasp" # Это серийный номер весов  
 url = 'https://smart-farm.kz:8502/api/v2/Sprayings?scalesSerialNumber=' + type\_scales + \  
 '&animalRfidNumber=' + cow\_id # Когда отправляет запрос учитывается сериый номер и бирка cow\_id   
 request\_get = requests.get(url, timeout=5).json()  
 print(request\_get)  
 return request\_get  
 except Exception as e:  
 print('request get func error')  
  
def connect\_rfid\_reader(): # Connection to RFID Reader through TCP and getting cow ID in str format  
 try:  
 logger.debug(f'START RFID FUNCTION')  
 TCP\_IP = '192.168.1.250' # chafon 5300 reader address  
 TCP\_PORT = 60000 # chafon 5300 port  
 BUFFER\_SIZE = 1024  
 animal\_id = "b'435400040001'" # Id null starting variable  
 animal\_id\_new = "b'435400040001'"  
 null\_id = "b'435400040001'"  
 logger.debug(f'START Animal ID animal\_id: {animal\_id}')  
 logger.debug(f'START Null id null\_id : {null\_id}')  
  
 if animal\_id == null\_id: # Send command to reader waiting id of animal  
 s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 s.connect((TCP\_IP, TCP\_PORT))  
 s.send(bytearray([0x53, 0x57, 0x00, 0x06, 0xff, 0x01, 0x00, 0x00, 0x00,  
 0x50])) # Chafon RU5300 Answer mode reading mode command  
 data = s.recv(BUFFER\_SIZE)  
 animal\_id = str(binascii.hexlify(data))  
 animal\_id\_new = animal\_id[:-5] # Cutting the string from unnecessary information after 4 signs  
 animal\_id\_new = animal\_id\_new[-12:] # Cutting the string from unnecessary information before 24 signs  
 logger.debug(f'Raw ID animal\_id: {animal\_id}')  
 logger.debug(f'New ID animal\_id\_new: {animal\_id\_new}')  
 logger.debug(f'Null id null\_id : {str(null\_id)}')  
 s.close()  
 if animal\_id\_new == null\_id: # Id null return(0)  
 connect\_rfid\_reader()  
 else: # Id check return(1)  
 animal\_id = "b'435400040001'"  
 logger.debug(f'Success step 2 RFID. animal id new: {animal\_id\_new}')  
 return animal\_id\_new  
 except Exception as e:  
 logger.error(f'Error connect to Arduino {e}')  
 else:  
 logger.debug(f'2 step RFID')  
  
def main():  
 logger.info(f'If you want to know your rfid number enter 1')  
 logger.info(f'if you want to take the task from server enter 2')  
 z = input()  
 if z == "1":  
 print(connect\_rfid\_reader())  
 else:  
 \_\_request\_get()  
  
  
main()

# rfid\_ethernet\_test.py

import sys  
import socket  
import binascii  
from pathlib import Path  
from loguru import logger  
import time  
  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_config\_manager import ConfigManager  
from \_chafon\_rfid\_lib import RFIDReader  
from \_lib\_pcf import \_\_connect\_rfid\_reader\_ethernet, \_set\_power\_RFID\_ethernet  
from \_glb\_val import RFID\_READER\_USB  
  
# === Конфигурация и логгирование ===  
config\_manager = ConfigManager()  
  
logger.remove()  
logger.add(sys.stderr, format="{time} {level} {message}", level="DEBUG")  
  
RFID\_TIMEOUT = int(config\_manager.get\_setting("RFID\_Reader", "reader\_timeout"))  
  
def read\_rfid():  
 if RFID\_READER\_USB:  
 logger.info("Using USB-connected RFID reader")  
 reader = RFIDReader()  
 tag\_id = reader.connect()  
 logger.info(f"RFID tag ID (USB): {tag\_id}")  
 return tag\_id  
 else:  
 \_set\_power\_RFID\_ethernet()  
 tag\_id = \_\_connect\_rfid\_reader\_ethernet()  
 if tag\_id:  
 logger.info(f"RFID tag ID (Ethernet): {tag\_id}")  
 else:  
 logger.error("Failed to read RFID tag over Ethernet.")  
 return tag\_id  
  
def main():  
 logger.info("RFID Reader Test Menu\n1 — Read tag\n2 — Exit")  
 choice = input("Enter choice: ").strip()  
 if choice == "1":  
 try:  
 while True:  
 tag\_id = read\_rfid()  
   
 if tag\_id:  
 logger.success(f"RFID tag detected: {tag\_id}")  
 time.sleep(0.5)   
  
 except KeyboardInterrupt:  
 logger.info("Interrupted by user.")  
 else:  
 logger.info("Exiting.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# rfid\_test.py

import sys  
  
PRESET\_VALUE = 0xFFFF  
POLYNOMIAL = 0x8408  
  
def checksum(pucY):  
 uiCrcValue = PRESET\_VALUE  
 for ucY in pucY:  
 uiCrcValue = uiCrcValue ^ ucY  
 for ucJ in range(8):  
 if uiCrcValue & 0x0001:  
 uiCrcValue = (uiCrcValue >> 1) ^ POLYNOMIAL  
 else:  
 uiCrcValue = (uiCrcValue >> 1)  
 return uiCrcValue  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('%X' % (checksum(bytearray.fromhex(sys.argv[1])),))

# rfid\_usb\_test.py

import sys  
from pathlib import Path  
from loguru import logger  
import time  
  
# Добавляем путь к src, где лежит \_chafon\_rfid\_lib.py  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_chafon\_rfid\_lib import RFIDReader  
  
# Настройка логирования  
logger.remove()  
logger.add(sys.stderr, format="{time} {level} {message}", level="DEBUG")  
  
  
def main():  
 # Создаём экземпляр  
 reader = RFIDReader()  
  
 # Если порт не задан в config.ini — ищем автоматически  
 if not reader.reader\_port:  
 port = reader.find\_rfid\_reader()  
 if port:  
 logger.info(f"Найден ридер на порту {port}")  
 else:  
 logger.error("Не удалось найти RFID-ридер. Проверьте подключение.")  
 return  
  
 # Открываем соединение (timeout на чтение 0.05 с)  
 try:  
 reader.open(timeout=0.05)  
 except Exception as e:  
 logger.error(f"Ошибка при открытии порта: {e}")  
 return  
  
 logger.info("Начинаем читать метки (Ctrl+C для выхода)")  
 try:  
 while True:  
 tag = reader.read\_tag(timeout=1.0)  
 if tag:  
 logger.info(f"Прочитана метка: {tag}")  
 # Маленькая пауза, чтобы не забивать логи  
 time.sleep(0.1)  
 except KeyboardInterrupt:  
 logger.info("Выход по Ctrl+C")  
 finally:  
 reader.close()  
 logger.info("Соединение закрыто")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# rfid\_usb\_test\_2.py

import sys  
from pathlib import Path  
from loguru import logger  
import time  
  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_chafon\_rfid\_lib import RFIDReader  
  
logger.remove()  
logger.add(sys.stderr, format="{time} {level} {message}", level="DEBUG")  
  
def main():  
 logger.info(f'Начало теста rfid\_reader модуля - считывание через USB')  
 logger.info(f'Пожалуйста заполните ../config/config.ini [RFID\_Reader] reader\_port = \nК примеру: /dev/ttyUSB0')  
 logger.info(f'Если вы не знаете порт пожалуйста сделайте следующие шаги: ')  
 logger.info(f'1. Отключите USB от rfid\_reader;')  
 logger.info(f'2. Откройте терминал (ctrl+alt+t) и введите следующую команду: /dev/tty\*;')  
 logger.info(f'3. Запомните список. Список возможно будет большой, но внимание нужно обратить на ttyUSB\* или ttyACM\*;')  
 logger.info(f'4. Вставьте USB обратно в raspberry и снова введите команду: /dev/tty\*;')  
 logger.info(f'5. Должен появиться новый ttyUSB\* или ttyACM\*')  
 logger.info(f'6. Перепишите его в ../config/config.ini [RFID\_Reader] reader\_port = СЮДА\nК примеру: /dev/ttyUSB0')  
 try:  
 rfid\_reader = RFIDReader()  
 except Exception as e:  
 logger.error(f'Не удалось инициализировать RFIDReader: {e}')  
 logger.info('Программа будет закрыта через 10 секунд...')  
 time.sleep(10)  
 sys.exit(1)  
  
 start\_time = time.time()  
 timeout = 600   
 try:  
 while True:  
 current\_time = time.time()  
 animal\_id = rfid\_reader.connect()  
 logger.info(animal\_id)  
 time.sleep(0.1)  
 if current\_time - start\_time > timeout:  
 logger.info('10 минут прошли. Пока.')  
 break  
 except Exception as e:  
 logger.error(f'Error: {e}')  
   
main()

# sprayer\_test.py

import sys  
from pathlib import Path  
  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_config\_manager import ConfigManager  
import time  
import timeit  
import datetime  
try:  
 import RPi.GPIO as GPIO  
except RuntimeError:  
 from \_\_gpio\_simulator import MockGPIO as GPIO  
  
  
from \_sprayer import Sprayer  
import \_values\_class as value\_data  
from \_glb\_val import \*  
  
   
from loguru import logger  
  
debug\_level = "DEBUG" if DEBUG == 1 else "CRITICAL"  
  
"""Инициализация logger для хранения записи о всех действиях программы"""  
logger.add(sys.stdout, format="{time} {level} {message}", level=debug\_level)   
  
  
def process(cow\_id: str) -> tuple:  
 logger.info(f'Sprayer test started for cow\_id: {cow\_id}')  
 try:  
  
 weight\_arr = []  
 next\_time = time.time() + 1  
   
 drink\_start\_time = timeit.default\_timer()  
 gpio\_state = False  
   
 values = value\_data.Values(  
 drink\_start\_time, 0, TYPE\_SCALES, cow\_id, 0, '0', 0, 0, 0, 0, True  
 )  
  
 logger.info(f'Initial values set: {values}')  
  
 if SPRAYER:  
 sprayer = Sprayer(values)  
 logger.info('Sprayer initialized.')  
 i = 200  
 while True:  
  
 current\_time = time.time()  
 time\_to\_wait = next\_time - current\_time  
  
 if SPRAYER:  
 if not values.flag:  
 gpio\_state = sprayer.spray\_main\_function(gpio\_state)  
 values = sprayer.new\_start\_timer(gpio\_state)  
 else:  
 if time\_to\_wait < 0 and round(time.time(), 0) % 5 == 0:  
 values.flag = False  
  
 if time\_to\_wait < 0:  
 i+=1  
 weight\_arr.append(i)  
 next\_time = time.time() + 1  
 logger.debug(f'Array weights: {weight\_arr}')  
 time.sleep(1)  
 if i == 220:  
 break  
   
 GPIO.cleanup()  
  
 if not weight\_arr:  
 logger.info("null weight list")  
 return 0, [], ''  
  
 if SPRAYER:  
 gpio\_state = sprayer.gpio\_state\_check(gpio\_state)  
  
 except KeyboardInterrupt as e:  
 logger.error(f'measure\_weight Error: {e}')  
 GPIO.cleanup()  
 if SPRAYER:  
 gpio\_state = sprayer.gpio\_state\_check(gpio\_state)  
   
  
def main():  
 logger.info(f'Starting test of \_sprayer.py.')  
 cow\_id = '4354001c004501c38d2010132e25010f0101e2806894000040103003'  
 process(cow\_id)  
  
main()

# tag\_id.py

import binascii  
  
def calculate\_crc(data):  
 crc = 0xFFFF  
 for byte in data:  
 crc ^= byte << 8  
 for \_ in range(8):  
 if crc & 0x8000:  
 crc = (crc << 1) ^ 0x1021  
 else:  
 crc <<= 1  
 crc &= 0xFFFF  
 return crc  
  
def verify\_crc(tag):  
 # Convert the tag hex string to bytes  
 tag\_bytes = binascii.unhexlify(tag)  
   
 # Extract the data and the CRC value from the tag  
 data = tag\_bytes[:-2]  
 received\_crc = int.from\_bytes(tag\_bytes[-2:], byteorder='big')  
   
 # Calculate the CRC from the data  
 calculated\_crc = calculate\_crc(data)  
   
 # Compare the calculated CRC with the received CRC  
 return calculated\_crc == received\_crc  
  
def tags\_id\_dict(tags):  
 # Test the function with the given tags  
 for tag in tags:  
 is\_valid = verify\_crc(tag)  
 print(f"Tag {tag} is {'valid' if is\_valid else 'invalid'}")  
  
def inventory\_tag():  
 import socket  
 from chafon\_rfid.command import G2\_TAG\_INVENTORY  
 from chafon\_rfid.response import G2InventoryResponseFrame  
 from chafon\_rfid.transport import TcpTransport  
  
 # Параметры подключения к считывателю  
 reader\_ip = '192.168.1.250'   
 reader\_port = 60000  
   
 # Создаем транспорт  
 transport = TcpTransport(reader\_ip, reader\_port)  
  
 # Отправляем команду на инвентаризацию  
 command = G2\_TAG\_INVENTORY()  
 transport.write(command.serialize())  
  
 # Читаем ответ  
 response\_frame = G2InventoryResponseFrame(transport.read\_frame())  
  
 # Проверяем, есть ли полученные теги  
 if response\_frame.tag\_count > 0:  
 for tag in response\_frame.tags:  
 # Печатаем EPC каждой прочитанной метки  
 print(f'Read tag EPC: {tag.epc}')  
  
 # Закрываем соединение  
 transport.close()  
   
  
def main():  
 tags = [  
 "4354001c004501c38d140a0c05d8010f0101e2806894000050103003cc70b0c1",  
 "4354001c004501c38d13091e0102010f0101e2806894000040103003e195ba57",  
 "4354001c004501c38d13091e0102010f0101e2806894000040103003ed88be54",  
 "4354001c004501c38d140a0c05d8010f0101e2806894000050103003cc70b0c1"  
 ]  
 inventory\_tag()  
  
main()

# task\_sprayer.py

from loguru import logger  
from datetime import datetime  
import requests  
  
  
def sprayer\_task\_request(cow\_id):  
  
 current\_time = datetime.utcnow()  
  
 formatted\_time = current\_time.strftime('%Y-%m-%dT%H:%M:%S.%fZ')  
  
 logger.debug(f"Start spray\_json\_payload function")  
 data = {  
 "EventDate": formatted\_time,  
 "TaskId": 'values.task\_id',  
 "ScalesSerialNumber": 'values.type\_scales',  
 "SpayerSerialNumber": "s01000001",  
 "RFIDNumber": 'values.cow\_id',  
 "SprayingType":' values.spraying\_type',  
 "Volume": 'values.new\_volume'  
 }  
   
def \_\_request\_get(cow\_id, type\_scales):   
 try:  
 url = 'https://smart-farm.kz:8502/api/v2/Sprayings?scalesSerialNumber=' + type\_scales + \  
 '&animalRfidNumber=' + cow\_id  
 request\_get = requests.get(url, timeout=5).json()  
 return request\_get  
 except Exception as e:  
 logger.error(f'request get func error {e}')  
  
   
def main():  
 cow\_id = '940000501030'  
 type\_scales = 'OrgSc-02'  
 #sprayer\_task\_request(cow\_id)  
 request\_get\_json = \_\_request\_get(cow\_id, type\_scales)  
 logger.debug(f'JSON Request: {request\_get\_json}')  
   
main()

# test\_fm.py

import sys  
from pathlib import Path  
from loguru import logger  
  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_config\_manager import ConfigManager  
  
  
config\_manager = ConfigManager()  
from time import sleep  
  
logger.remove()  
logger.add(sys.stderr, format="{time} {level} {file}:{line} {message}", level="DEBUG")  
  
def main():  
 try:  
 logger.info(f'\033[1;35mFeeder project. Test.\033[0m')  
 logger.info(f'RFID\_TIMEOUT = {float(config\_manager.get\_setting("RFID\_Reader", "reader\_timeout"))}')  
  
 finally:  
 logger.info("Bye!")  
  
main()

# test\_med.py

import statistics  
  
def read\_data():  
 print(f'Enter: ')  
 return int(input())  
  
def calib\_read\_2(times=10):  
 """Use only in calibration"""  
 calib\_arr = []  
   
 # Collect data 'times' times  
 for \_ in range(times):  
 calib\_arr.append(read\_data())  
   
 # Calculate and return the average value  
 average\_value = sum(calib\_arr) / len(calib\_arr)  
 return average\_value  
  
def calib\_read(times=10):  
 """Use only in calibration"""  
 calib\_arr = []  
 calib\_arr\_2 = []  
   
 # Collect data 'times' times  
 for \_ in range(times):  
 number = read\_data()  
 calib\_arr.append(number)  
 calib\_arr\_2.append(number)  
  
 median\_value = statistics.median(calib\_arr)  
 average\_value = sum(calib\_arr\_2) / len(calib\_arr\_2)  
 from collections import Counter  
  
 counter = Counter(calib\_arr)  
 most\_common = counter.most\_common(1)[0][0]  
 return median\_value, average\_value, most\_common  
  
def main():  
 median, average, most\_common = calib\_read()  
 print(f'med: {median}')  
 print(f'ave: {average}')  
 print(f'mc: {most\_common}')  
  
main()

# weight0805.py

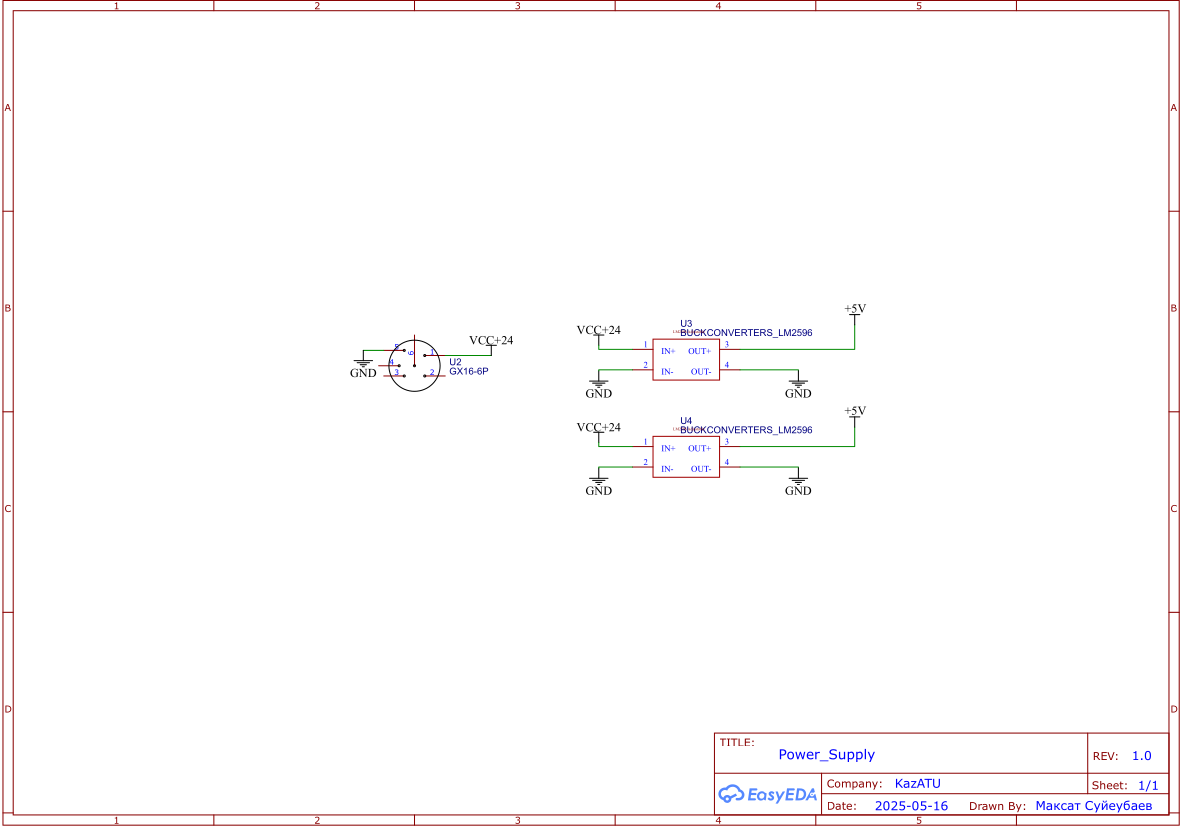
import sys  
import time  
import statistics  
from collections import deque  
from pathlib import Path  
from loguru import logger  
from time import sleep  
from typing import Optional, Callable  
  
# Добавляем папку src в путь поиска модулей  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_config\_manager import ConfigManager  
import \_lib\_pcf as lib  
  
config\_manager = ConfigManager()  
  
# Параметры детекции и стабилизации  
PRESENCE\_THRESHOLD = 20 # кг, считать, что животное есть на весах  
STABILITY\_THRESHOLD = 0.5 # кг, максимальное отклонение от медианы  
STABLE\_DURATION = 3 # секунды, стабильное состояние  
WINDOW\_SIZE = 10 # сколько последних замеров держать в буфере  
READ\_PERIOD = 0.1 # секунда между считываниями  
  
# Настройка логера  
logger.remove()  
logger.add(  
 sys.stderr,  
 format="{time:HH:mm:ss.SSS} | {level} | {file}:{line} | {message}",  
 level="DEBUG"  
)  
  
  
def create\_stable\_weight\_detector(  
 presence\_thr: float,  
 stability\_thr: float,  
 stable\_duration: float,  
 window\_size: int  
) -> Callable[[float], Optional[float]]:  
 """  
 Возвращает функцию detector(w), которую нужно вызывать для каждого нового замера w.  
 detector возвращает None, пока вес не установится,  
 и как только на протяжении stable\_duration подряд все замеры  
 попадают в [медиана окна ± stability\_thr], отдаёт медиану этих замеров.  
 """  
 window\_buf = deque(maxlen=window\_size)  
 valid\_readings = []  
 stable\_start = None  
  
 def detector(w: float) -> Optional[float]:  
 nonlocal stable\_start  
 window\_buf.append(w)  
 med = statistics.median(window\_buf)  
  
 # если мы ещё не «увидели» животных (медиана окна < presence\_thr),  
 # сбрасываем всё и ждём  
 if med < presence\_thr:  
 stable\_start = None  
 valid\_readings.clear()  
 return None  
  
 # отфильтровываем выбросы относительно медианы окна  
 if abs(w - med) <= stability\_thr:  
 # при первом попавшемся «правильном» замере запускаем таймер  
 if stable\_start is None:  
 stable\_start = time.time()  
 valid\_readings.append(w)  
 else:  
 # обнаружился выброс — сбрасываем всю историю и таймер  
 stable\_start = None  
 valid\_readings.clear()  
 return None  
  
 # проверяем, не прошло ли уже время стабильности  
 if time.time() - stable\_start >= stable\_duration:  
 # возвращаем медиану накопленных «правильных» показаний  
 return statistics.median(valid\_readings)  
  
 return None  
 return detector  
  
def main():  
 arduino = None  
 try:  
 logger.info("Feeder project. Start weight collection until cow leaves.")  
 lib.\_calibrate\_or\_start()  
 arduino = lib.start\_obj()  
 sleep(1) # даём Arduino прогреться  
  
 window\_buf = deque(maxlen=WINDOW\_SIZE)  
 weight\_arr = []  
 cow\_on = False  
 logger.info("Ожидаем прихода коровы на весы...")  
 while True:  
 w = arduino.get\_measure\_2()  
 window\_buf.append(w)  
 med = statistics.median(window\_buf)  
 # Если коровы ещё нет  
 if not cow\_on:  
 logger.debug(f"Current median = {med:.2f} kg (ожидаем ≥ {PRESENCE\_THRESHOLD})")  
 if med >= PRESENCE\_THRESHOLD:  
 cow\_on = True  
 logger.info("Корова встала на весы — начинаем сбор данных.")  
 else:  
 # Корова уже на весах — собираем каждое показание  
 weight\_arr.append(w)  
 logger.debug(f"Collecting weight: {w:.2f} kg")  
  
 # Проверяем, ушла ли корова  
 if med < PRESENCE\_THRESHOLD:  
 logger.info("Корова покинула весы — завершаем сбор.")  
 break  
 sleep(READ\_PERIOD)  
 logger.info(f"Собрано {len(weight\_arr)} замеров: {weight\_arr}")  
 # Здесь можно дальше обрабатывать weight\_arr:  
 # отправить на сервер, сохранить в файл и т.д.  
 except KeyboardInterrupt:  
 logger.info("Прервано пользователем.")  
 except Exception as e:  
 logger.error(f"Ошибка: {e}")  
 finally:  
 if arduino:  
 logger.info("Отключаем Arduino.")  
 arduino.disconnect()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

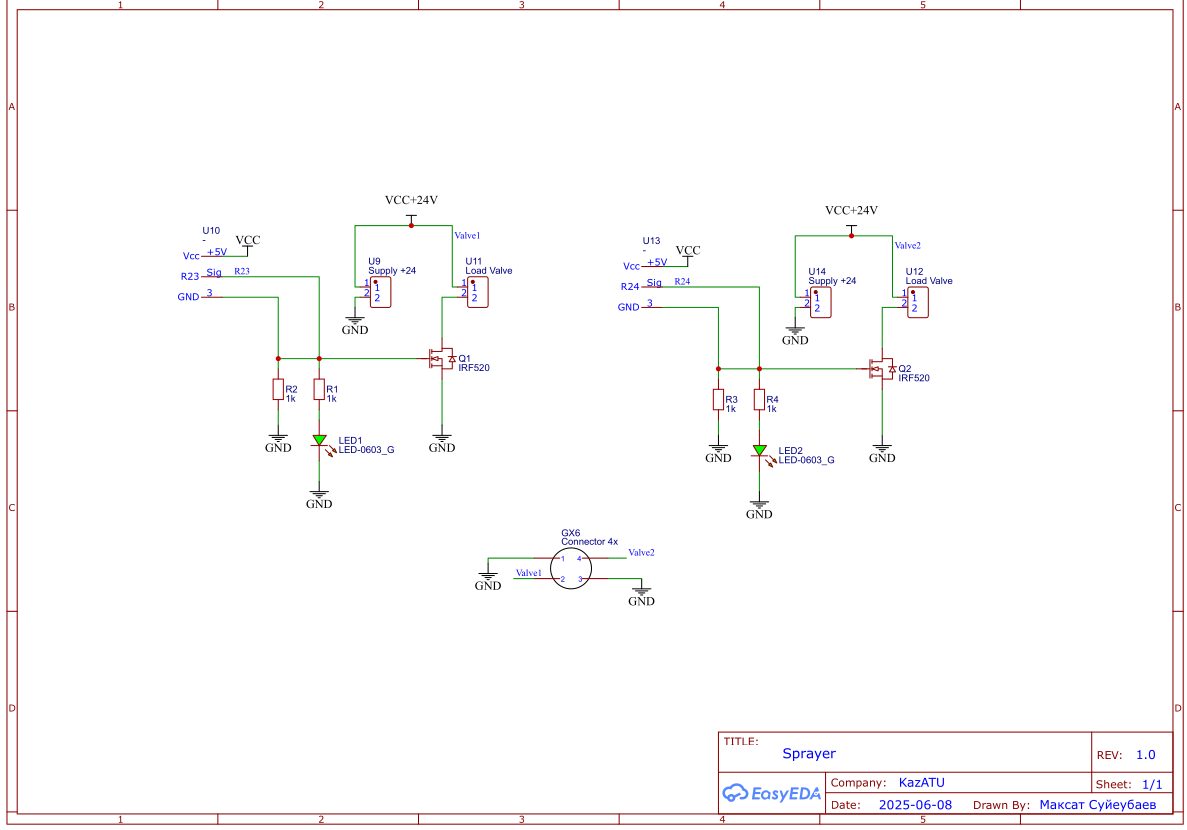
# weight\_measure\_test.py

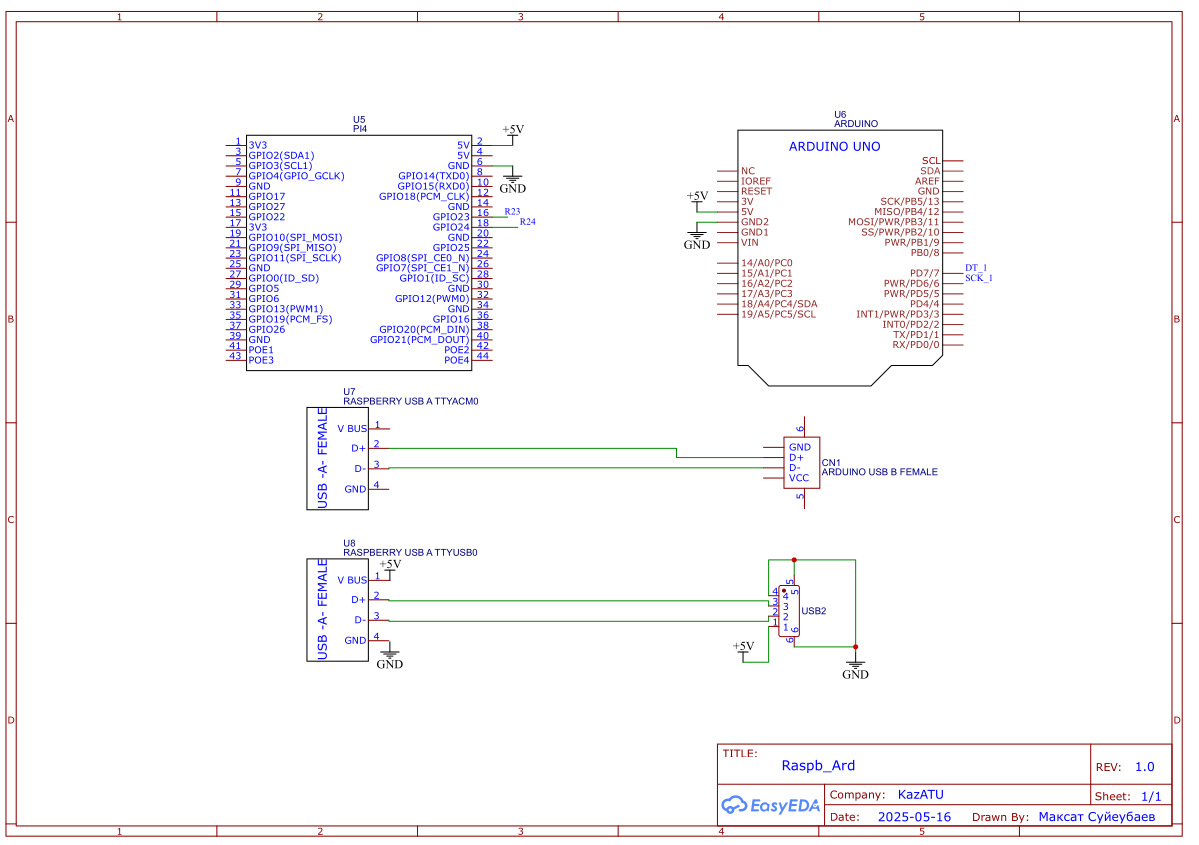
import sys  
from pathlib import Path  
from loguru import logger  
  
sys.path.append(str(Path(\_\_file\_\_).resolve().parent.parent / 'src'))  
  
from \_config\_manager import ConfigManager  
import \_lib\_pcf as lib  
config\_manager = ConfigManager()  
from time import sleep  
  
logger.remove()  
logger.add(sys.stderr, format="{time} {level} {file}:{line} {message}", level="DEBUG")  
  
def main():  
 try:  
 logger.info(f'\033[1;35mFeeder project. Weight measurment test file.\033[0m')  
 lib.\_calibrate\_or\_start()  
 arduino\_start = lib.start\_obj()  
 while True:  
 weight = arduino\_start.get\_measure\_2()  
 logger.info(f"Weight is: {weight}\n")  
 sleep(0.1)  
 finally:  
 logger.info("Bye!")  
 arduino\_start.disconnect()  
   
main()

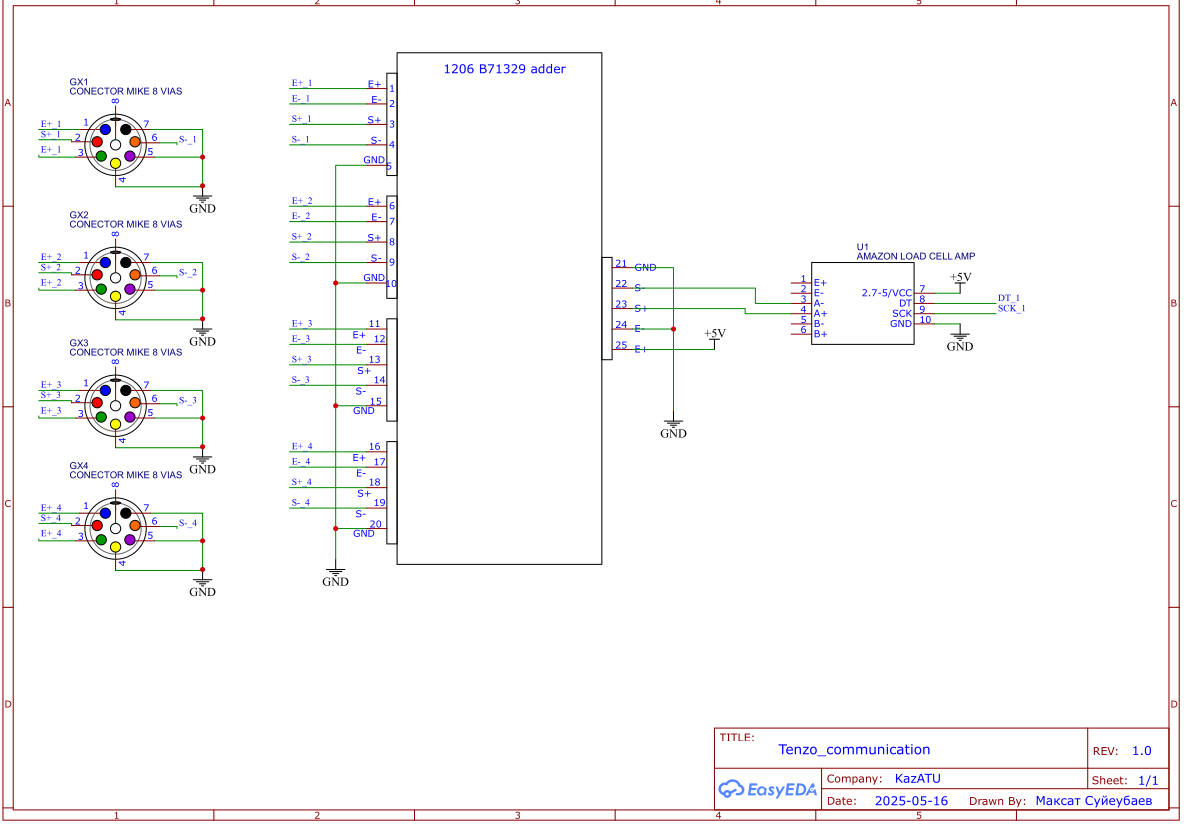
**Приложение 5**

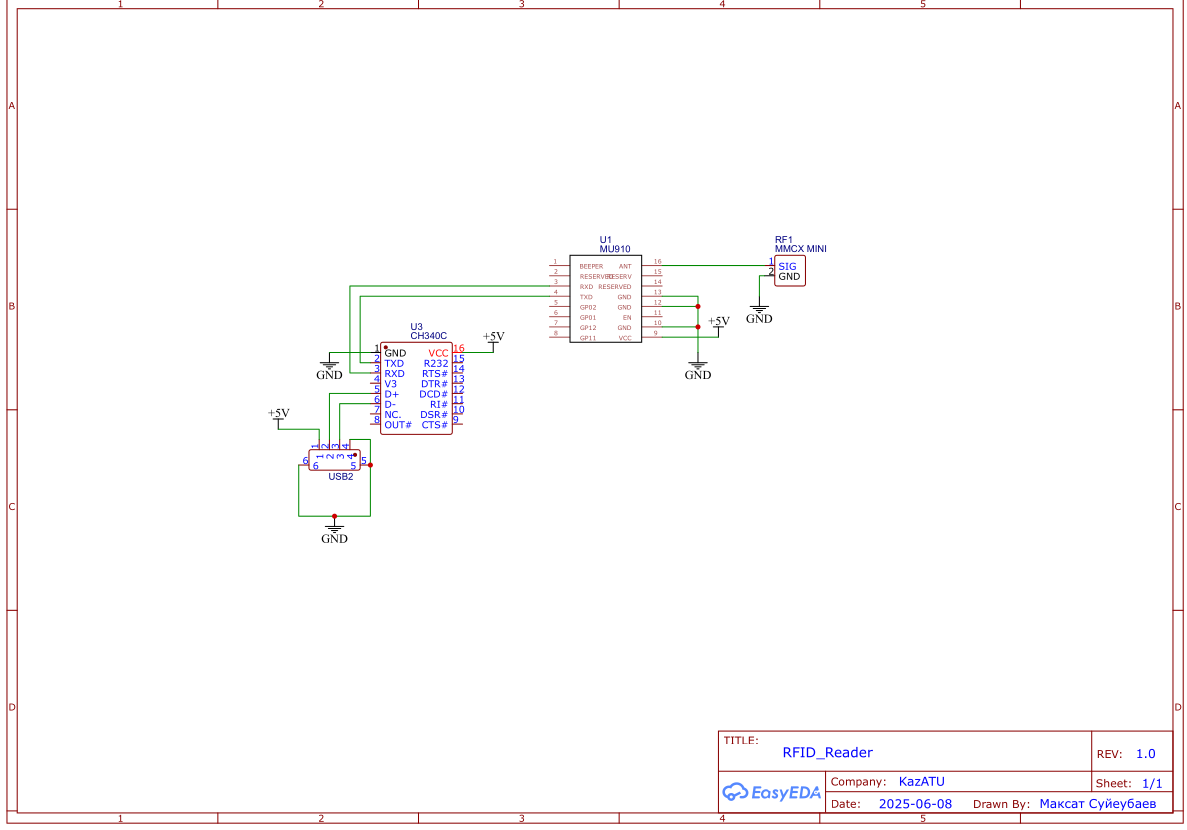
Схема соединения модулей проекта





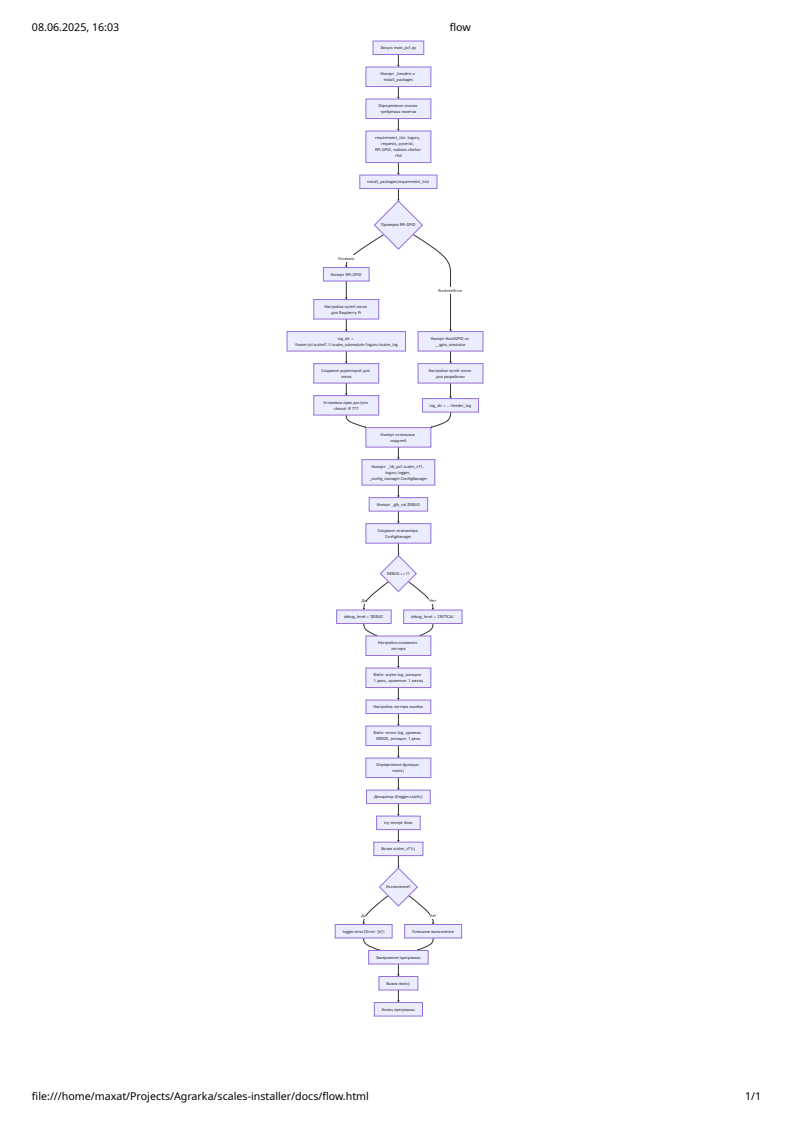


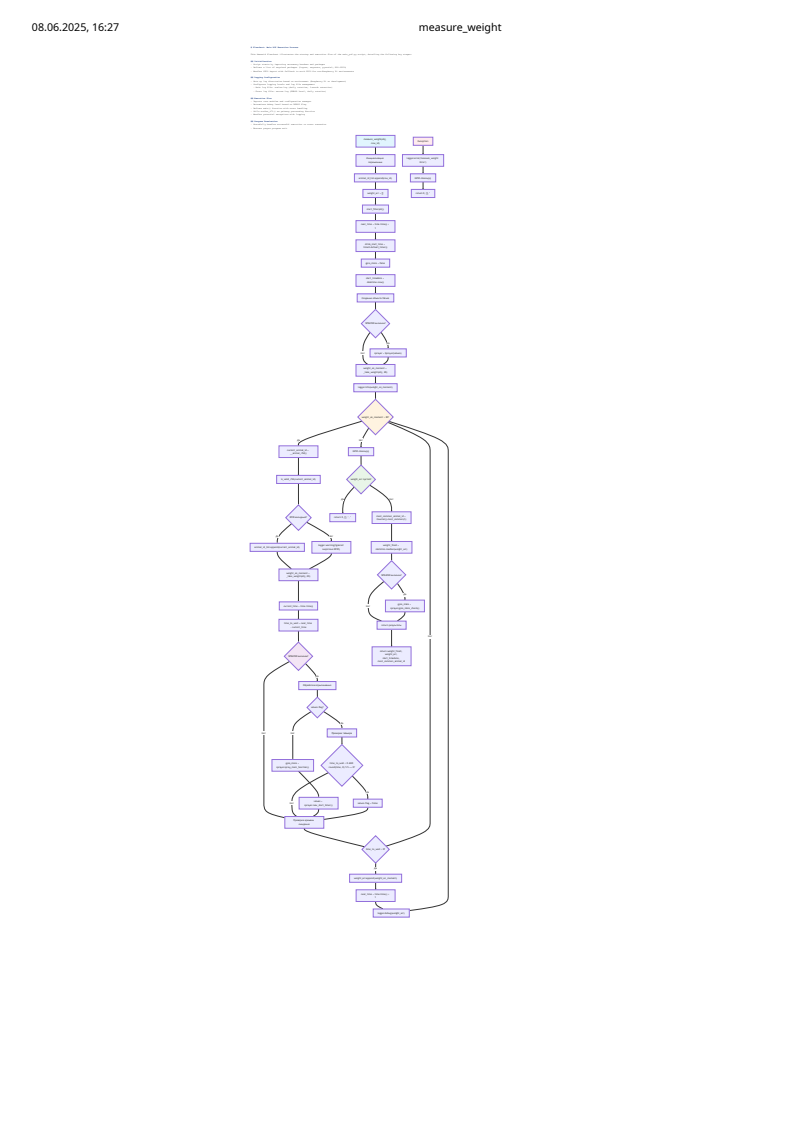


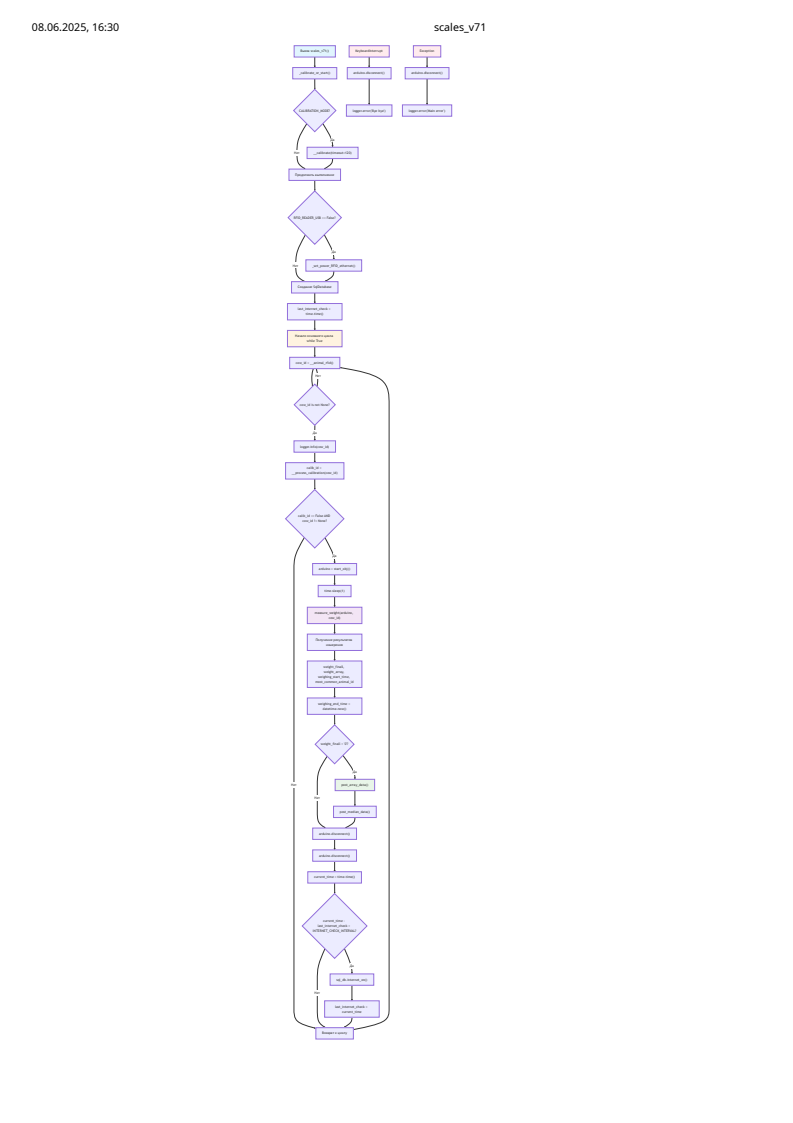


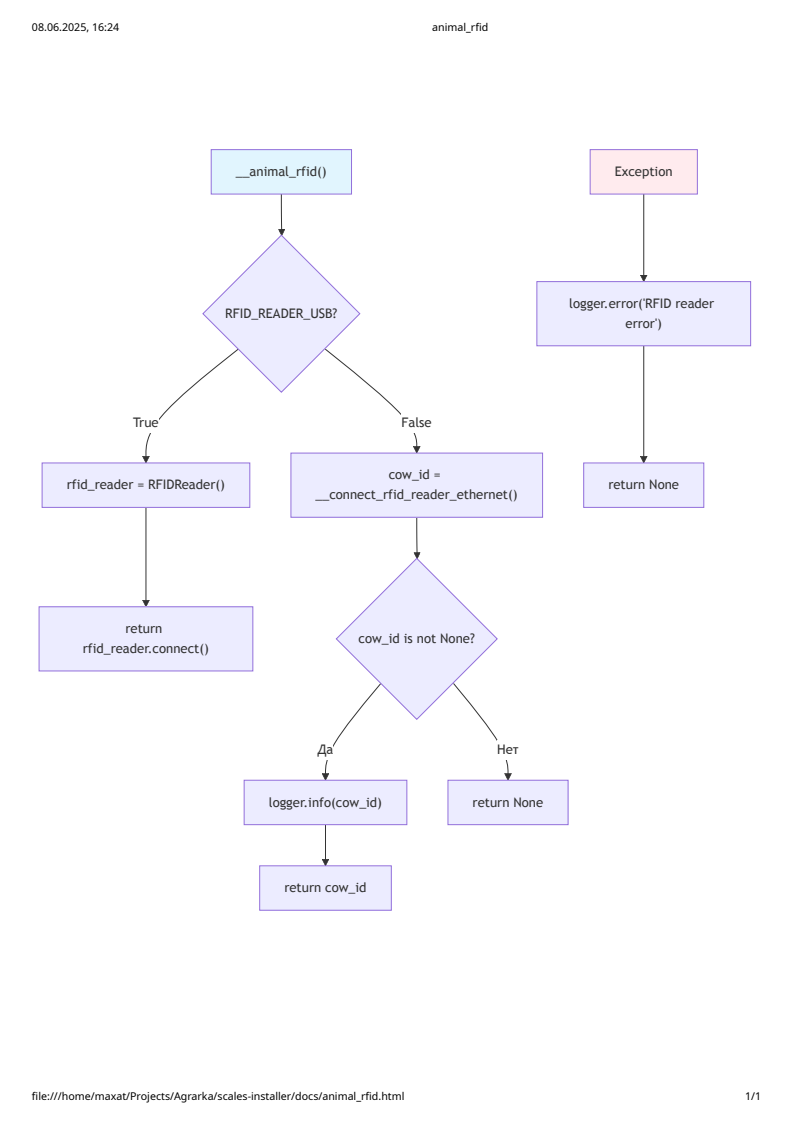
**Приложение 6**

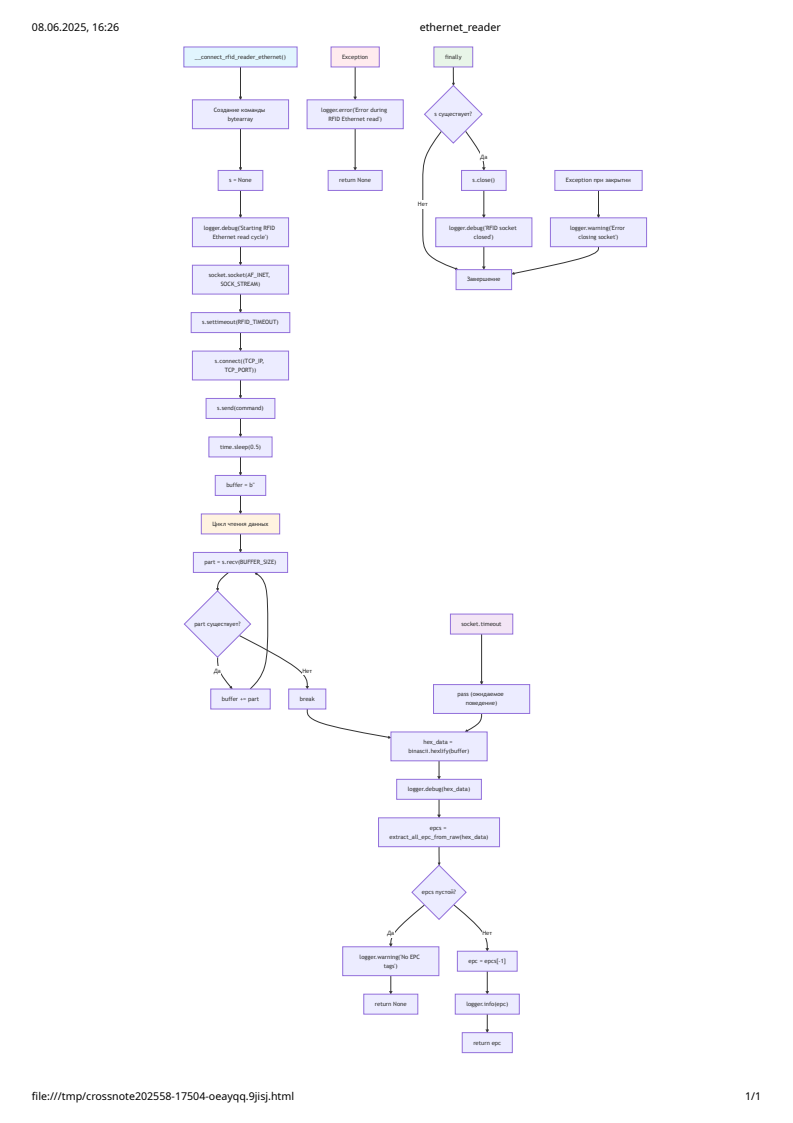
Блок схемы алгоритма проекта

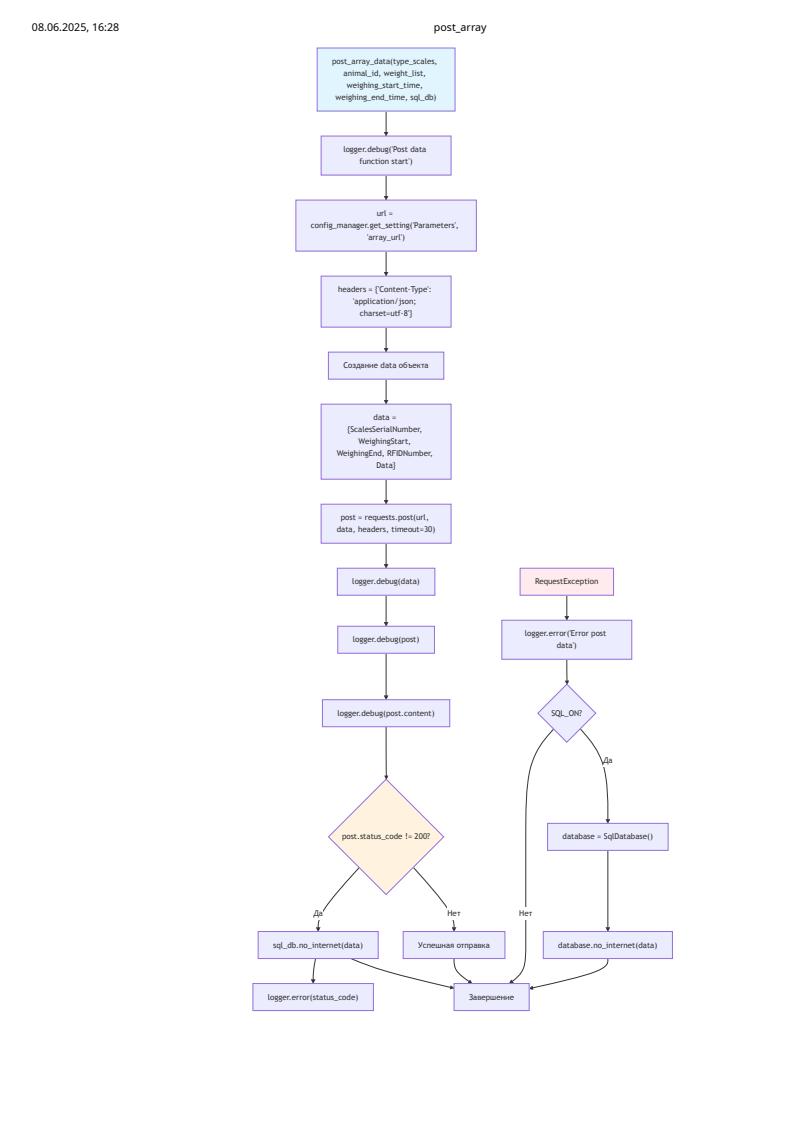


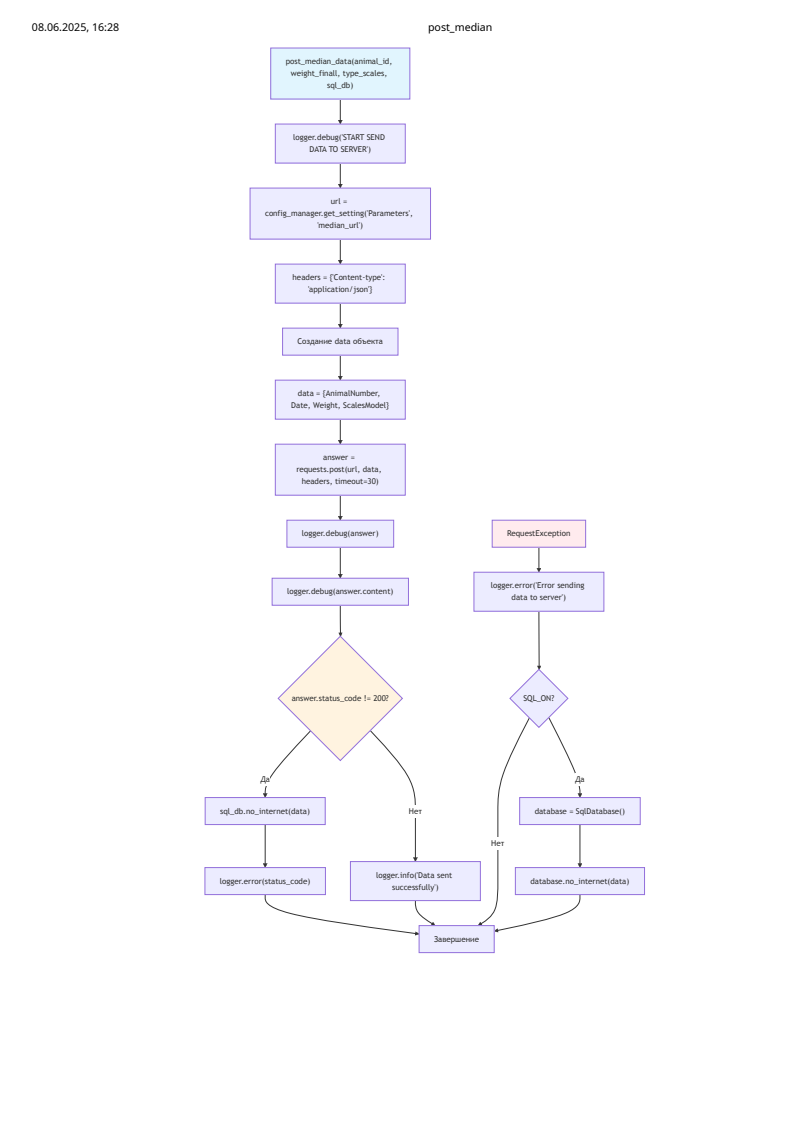


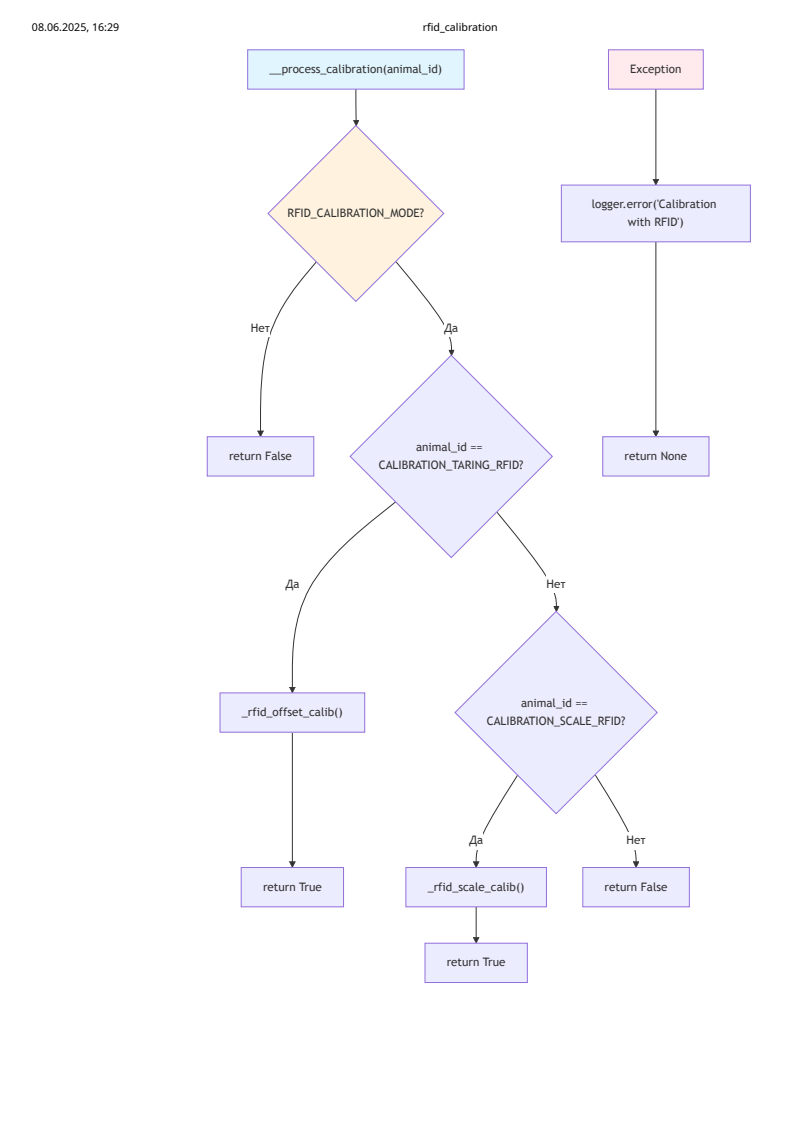


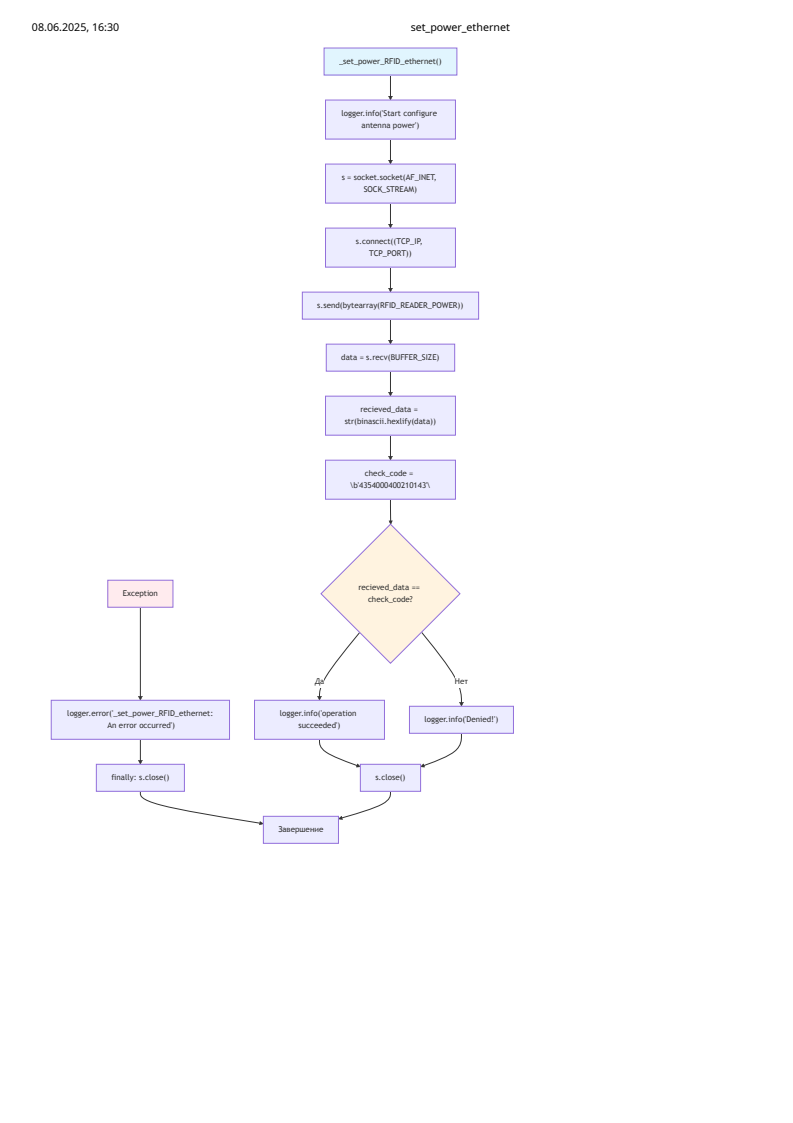












**Приложение 7.**

Информация об установленных устройствах

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тип оборудования | **Модель** | **Год и месяц создания** | **Модификация оборудования** | **Доп информация** | **Серийный номер** | **Местоположение** | **Примечание** | **Пользователь:Пароль** | **Калибровка (вес)** | **Ip add** |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-01 | Шагалы (удаленный участок) |  | pi:Astana789 | 26.05.2025 | 172.16.0.51 |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-02 | Шагалы (удаленный участок) | Замена транзистора-1шт | pi:Astana789 | 26.05.2025 |  |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-03 | Шагалы (удаленный участок) | Замена транзистора-1шт | pi:Astana789 | 26.05.2025 |  |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-04 | Шагалы (удаленный участок) |  | pi:Astana789 | 26.05.2025 | 172.16.0.136 |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-05 | Шагалы (удаленный участок) | Замена транзистора-1шт | pi:Astana789 | 26.05.2025 |  |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-06 | Шагалы (удаленный участок) |  | pi:Astana789 | 26.05.2025 |  |

| Тип оборудования | **Модель** | **Год и месяц создания** | **Модификация оборудования** | **Доп информация** | **Серийный номер** | **Местоположение** | **Примечание** | **Пользователь:Пароль** | **Калибровка (вес)** | **Ip add** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-07 | Шагалы (удаленный участок) |  | pi:Astana789 | 26.05.2025 | 172.16.0.98 |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-08 | Шагалы (удаленный участок) | Замена транзистора-1шт | pi:Astana789 | 26.05.2025 | 172.16.0.28 |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-09 | Шагалы (удаленный участок) | Разъем для опрыскивания | pi:Astana789 | 26.05.2025 |  |
| Весы + опрыскивание | OrganicScalesPro7.2 | 04.2025 | 7.2 | Опрыскивание + Сбор данных без интернета | OrgSc25-10 | Шагалы (удаленный участок) | Замена транзистора-2шт Замена Ардуино | pi:Astana789 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

**Приложение 8.**

Расчёт потребляемой мощности

Raspberry Pi 4:

PRPi=5В×1.2А=6Вт

Периферия:

Pпериф =5В×1А=5Вт

Суммарно на выходе DC-DC:

Pвыход общее=6+5=11Вт

Учитываем КПД DC-DC LM2596 (~80%):

P вход на DC-DC = 11/0.8 =13.75Вт

Добавим электроклапан (12 В, 0.6 А):

P клапан =12×0.6=7.2Вт

Общая мощность от блока питания 12 В:

P итого =13.75+7.2=20.95Вт

E=20.95Вт×1ч=20.95Вт\cdotpч=0.02095кВт\ч

Общая нагрузка: ≈ 21 Вт

Потребление за 1 час: ≈ 0.021 кВт·ч

1 система потребляет примерно 1.75А

10 систем 17.5А

Блок питания 4А

Nблоковпитания = 17.5/4 примерно 5