Минобрнауки России
федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»

Институт дополнительного образования
Высшая инженерная школа

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
РАЗРАБОТКА ПРОТОТИПА МОДУЛЯ УПРАВЛЕНИЯ ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ КАПЕЛЬНОГО ПОЛИВА

по программе профессиональной переподготовки:
«Разработчик прикладного программного обеспечения (Язык Python)»

Выполнил(а):
Хошина Наталья Юрьевна
Подпись_____

Руководитель:
Первушин Алексей Олегович
Подпись_____

Санкт-Петербург 2021

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ

ТЗ	-	Техническое задание
БД	-	База данных
СУБД	-	Система управления базами данных
API	-	Application Programming Interface (интерфейс программирования приложений).
IP	-	Internet Protocol (межсетевой протокол)
ORM	-	Object-relational mapping (объектно-реляционное отображение)
SQL	-	Structured Query Language (язык структурированных запросов)

ВВЕДЕНИЕ

Современный мир диктует определенные условия существования крупных компаний, обусловленные высокой конкуренцией на быстро развивающемся рынке. Для сохранения текущих позиций и устойчивого развития требуется повышение эффективности как рабочих процессов, так и человеческого труда, обеспечивающего инженерную и коммерческую деятельность компаний.

Эффективность бизнеса выражается прежде всего в рентабельности его деятельности. Если этот показатель выше, чем у основных конкурентов, компания может позволить себе и более низкие цены реализации, что, несомненно, даст ей преимущество на рынке сбыта.

Повысить прибыль и рентабельность компании можно двумя путями: увеличить доходы и сократить затраты. Сейчас рост доходов под влиянием внешних рыночных факторов и отсутствия роста покупательной способности населения не может быть источником повышения эффективности бизнеса, поэтому сокращение затрат остается наиболее реальным способом достижения данной цели.

В результате представленного в данной работе варианта автоматизации систем управления капельным поливом удалось добиться снижения затрат, улучшения сервиса и повышения эффективности управления.

От корректности и своевременности полива зависит качество растений. Растения нормально развиваются при условии оптимального увлажнения и благоприятных температурных показателях.

Интеллектуальная система управления капельным поливом позволяет:

- осуществлять автоматический полив растений на разных участках теплицы,
- вести постоянный мониторинг растений,
- производить диагностику проблем растений по внешнему виду,

- а также экономить время и трудозатраты по уходу за растениями,
- экономить потребляемые ресурсы.

1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Существующие решения

Современные технологии призваны помочь агротехнологам в планировании ирригационного процесса с рациональным потреблением воды.

На рынке существует множество систем, автоматизирующих полив и производящих мониторинг процесса роста растений. Они помогают принимать эффективные управленческие решения. Некоторые из этих систем позволяют не только наблюдать за изменениями условий, но и дистанционно управлять системами орошения.

Для точного полива на основе рекомендаций, полученных в результате наблюдений, в рабочий процесс внедряются технологии точного орошения.

1.2 Недостатки существующих решений

Рассмотрим необходимость полива в помещениях хранения растений с динамически сменяемой экспозицией в зависимости от сезона и ассортимента. Примерами таких помещений могут быть: магазин-склад растений, склад-питомник озеленительной компании и т.д.

Основная проблема – это частая смена экспозиции и ассортимента в зависимости от сезона, наличия товара и спроса. Для определенных растений нет фиксированного места на складе. Расположение товара определяется не наличием свободного места, а специальными техническими условиями, определяемыми специалистом-растениеводом. Типовые решения по автоматизации капельного полива, как правило, унифицированы и не учитывают индивидуальные особенности конкретного растения. Применение ручного труда требует значительных людских ресурсов и временных затрат.

1.3 Пути решения имеющихся недостатков

Решением проблемы является внедрение интеллектуальной системы капельного полива, которая позволит:

- оптимизировать численность персонала;
- сократить затраты;
- усовершенствовать управление технологическим процессом полива.

Реализовать такое решение можно, построив комбинированную систему, состоящую из модуля детектирования растения и модуля полива. Детекция и классификация производится на основе полученного снимка с камеры. А полив производится согласно определенному типу растения.

2. ПОСТАНОВКА ЦЕЛИ И ЗАДАЧ

2.1 Цель работы и декомпозиция задачи

Целью данной работы является создание прототипа модуля системы интеллектуального полива.

Для достижения данной цели необходимо декомпозировать данную задачу. Разработка прототипа модуля системы интеллектуального полива включает следующие основные этапы:

- анализ предметной области;
- анализ требований технического задания (ТЗ) на разработку;
- проектирование;
- реализация в виде программного кода (кодирование);
- тестирование и анализ проделанной работы.

На этапе постановки цели и анализа требований осуществляется предварительное планирование этапов работ, сбор и обработка требований имеющегося ТЗ на разработку.

На этапе проектирования осуществляется проектирование архитектуры приложения в соответствии с требованиями, сформулированными на предыдущем этапе разработки.

На этапе кодирования осуществляется написание программного кода приложения в соответствии с разработанной архитектурой, используя выбранный набор языков программирования и сопутствующих библиотек.

На этапе тестирования и анализа проделанной работы осуществляется функциональное тестирование, а также проводится анализ соответствия полученных результатов с требованиями к разрабатываемому приложению.

На каждом из этапов разработки осуществляется также документирование полученных результатов в виде пояснительной записки.

Выбор технологии реализации задачи осуществляется на этапе проектирования и программирования программного обеспечения. Критериями выбора являются:

- актуальность используемых технологий;
- масштабируемость;
- перспективность дальнейшего развития и поддержки.

2.2 Анализ требований ТЗ на разработку

Требования к разрабатываемому модулю системы интеллектуального капельного полива определяются, полученным техническим заданием на разработку. Техническое задание было составлено на основе анализа типовых технических решений по системам капельного полива. Разрабатываемое программное обеспечение должно представлять собой сетевое приложение, состоящее из двух частей: обработка изображения и взаимодействие с системой полива и Web-интерфейса пользователя, оперирующих общей базой данных. Основным сценарием использования является определение и полив растений с возможностью дальнейшего просмотра полученных отчетов в интерфейсе Web-браузера.

Функциональные требования:

- добавление растений в базу данных с параметрами полива;
- изменение и удаление записей о растениях в базе данных;
- получение изображения с камеры;
- классификация растений по изображению, полученному с камеры, на основе предобученной нейронной сети;
- в случае успеха определения растения сохранение изображения в БД;
- выдача команды в систему полива, согласно параметрам полива из БД;
- все успешные поливы записываются в журнал (БД)

- взаимодействие со сторонними системами осуществляется посредством открытых протоколов.

Нефункциональные требования в части используемых технологий:

- основным языком программирования для разработки должен являться язык Python 3;
- использование открытых протоколов взаимодействия с системой получения изображения и системой полива;
- требования к бэк-енд части проекта:
    - в качестве основного Web-фреймворка должен использоваться фреймворк Django 3;
    - для хранения данных должна использоваться система управления базами данных (СУБД) SQLite;
    - допускается использование дополнительных модулей, фреймворков и компонентов, необходимых для реализации проекта.
- требования к фронт-енд части проекта:
    - при разработке фронт-енд части использовать язык шаблонов Django;
    - для обеспечения визуальной составляющей допускается использование фреймворка Bootstrap;
    - допускается использование кода, написанного на языке JavaScript.

Нефункциональные требования в части пользовательского интерфейса.

В ТЗ на разработку определяются следующие требования к пользовательскому интерфейсу. Страницы доступные всем пользователям:

- главная страница;

– страница с текущим состоянием полива;
– авторизация.

Для авторизованных пользователей доступны следующие страницы:

– главная страница;
– страница с текущим состоянием полива;
– авторизация;
– настройки параметров полива;
– журнал полива.

Таблица 2.1

Параметры доступа для пользователей

Страница	Функции	Пользователи	
		все	авторизованные
Главная	Отображение страницы	да	да
	Переход на страницу Полив	да	да
	Переход на страницу Журнал	нет	да
Полив	Отображение страницы	да	да
	Изменить растение	нет	да
Журнал	Отображение страницы	нет	да
Настройки	Отображение страницы	нет	да
	Изменить растение	нет	да
	Удалить растение	нет	да
	Добавить растение	нет	да
Авторизация		да	да
Выход		нет	да

3. ПРОЕКТИРОВАНИЕ

3.1 Описание архитектуры планируемого к разработке приложения

В текущей работе рассматривается создание прототипа модуля системы интеллектуального капельного полива, облегчающего процесс ухода за растениями и снижающего рутинную работу персонала. Подразумевается дальнейшее непрерывное развитие с расширением функций и возможностей системы, а также оптимизацией алгоритмов работы.

В объем работы входят следующие подзадачи:

1.) разработка структуры базы данных;

2.) разработка модуля обработки;

3.) разработка веб-интерфейса пользователя;

4.) проведение тестирования и анализ проделанной работы.

В объем работ не входит:

- управление системой получения изображений/камерой;
- управление системой полива.

Взаимодействие с этими системами осуществить через соответствующие API.

Архитектуру решения можно представить в виде структурной схемы см. рис.1.

Приложение предлагается развернуть на выделенном сервере.

Доступ к серверу подразумевает отсутствие не авторизованных пользователей, посредством установки его в помещении, закрытом от посторонних лиц. Для авторизации пользователя, необходимой и достаточной информацией являются: идентификатор пользователя и индивидуальный пароль пользователя.

В качестве базы данных предполагается использование SQLite версии 3, ввиду прогнозируемой незначительной нагрузки на базу данных. Количество растений определяется складской номенклатурой и является конечным числом.

В случае необходимости масштабирования, выбирается более производительный SQL сервер.

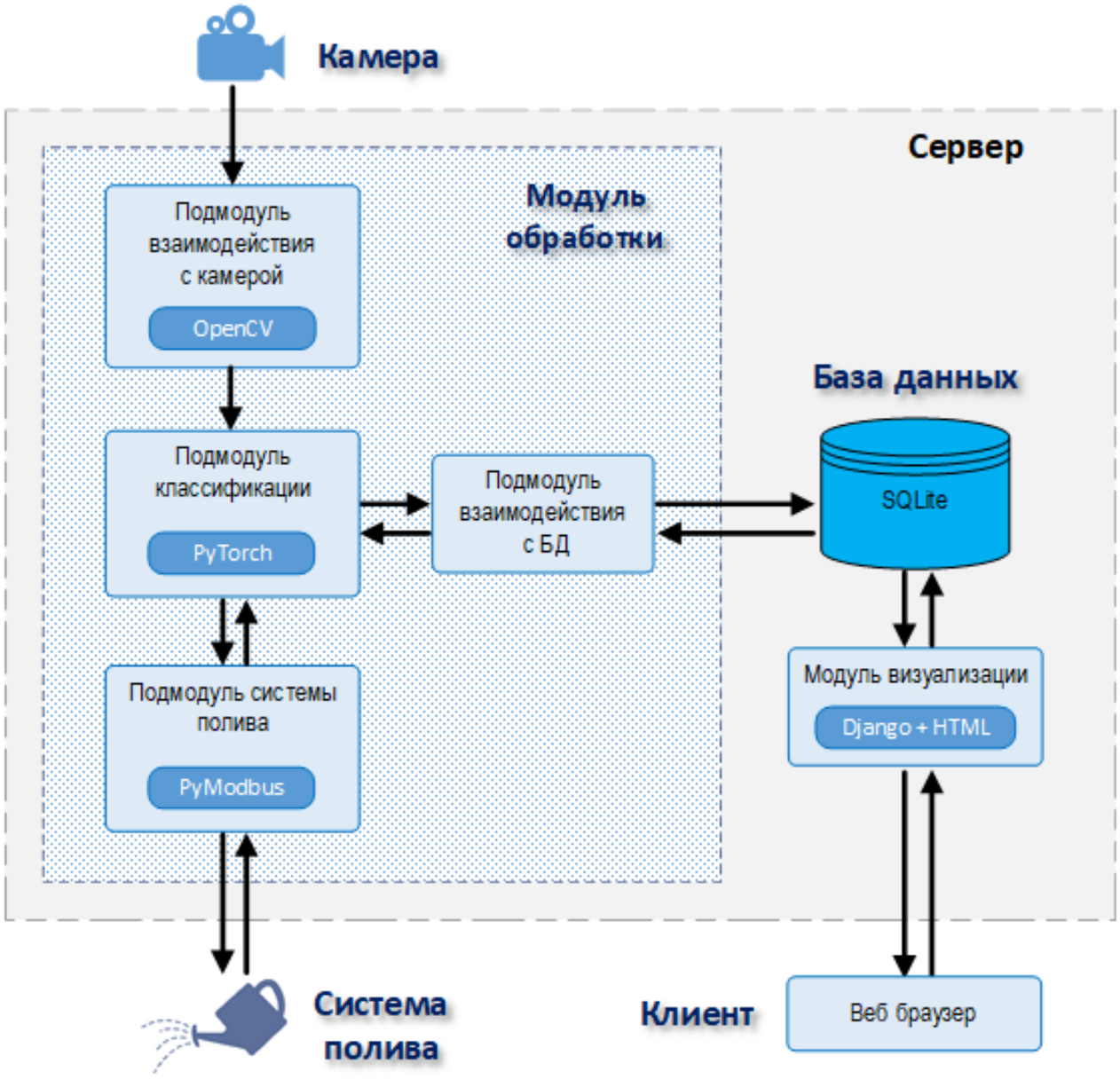

Рисунок 3.1 - Схема взаимодействия основных компонентов

Для реализации Web-сервиса планируется использование Web-фреймворка Django. Django является полнофункциональным серверным веб-фреймворком написанным на Python.

Предполагаемая модель взаимодействия компонентов приложения, подразумевает следующий алгоритм работы:

- обработка видеокадра (изображения). Камера присылает фото и складскую позицию (складская позиция требуется для управления конкретными устройствами полива);
- классификация полученного изображения на основе предобученной нейронной сети;
- выдача команды на полив;
- запись результатов в базу данных, для сбора статистики и аналитики.

Исходя из описанного выше, модуль обработки будет состоять из следующих подмодулей:

- Подмодуль взаимодействия с базой данных, включающий в себя следующие функции:
  - установка соединения с БД;
  - проверка соединения с БД;
  - чтение, запись, удаление из БД;
- Подмодуль взаимодействия с камерой, включающий в себя следующие функции:
  - установка соединения;
  - проверка соединения;
  - получение изображения.
- Подмодуль системы полива, включающий в себя следующие функции:
  - установка соединения;
  - проверка соединения;
  - чтение входных регистров;
  - передача команды на полив с требуемыми параметрами;
  - получение обратных сигналов.

– Подмодуль классификации, включающий в себя следующие функции:
    - нормализация и передача изображения в преобученную сеть
    - получение класса растения;
    - получение из БД параметров полива согласно определенному классу;
    - выдача команды в подмодуль полива;
    - запись результатов полива в БД (изображение, дата и время полива, параметры полива).

Алгоритм взаимодействия модулей представлен на рис. 3.2

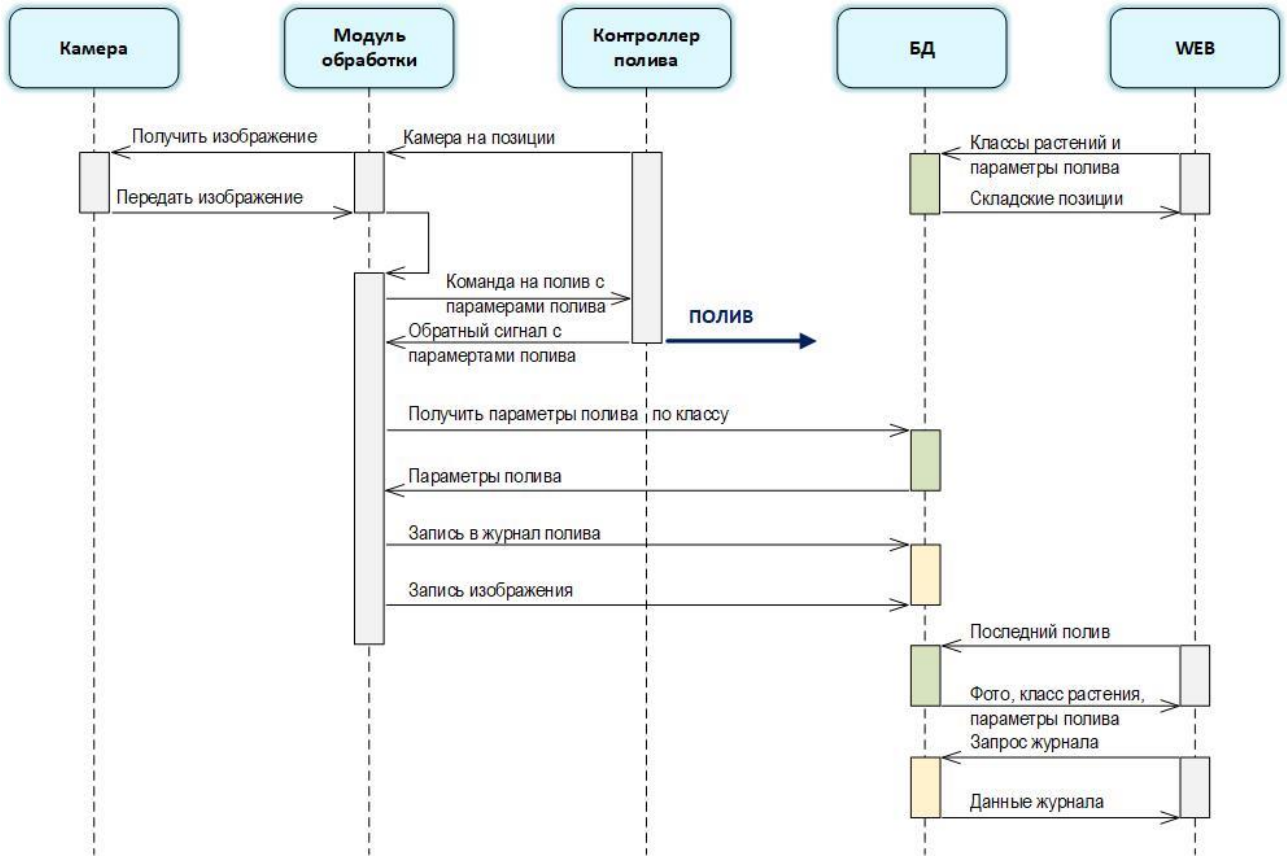

Рисунок 3.2 - Алгоритм взаимодействия модулей

4. РЕАЛИЗАЦИЯ РЕШЕНИЯ

4.1 Выбор технологий

Данная глава посвящена выбору технологий и описанию реализации программных решений.

Конечный продукт должен представлять собой приложение, состоящее из двух частей: модуль обработки, являющийся backend - частью написанный на языке Python 3 и разбитый на подмодули; Web-интерфейс, реализованный с помощью фреймворка Django 3 и использующие в качестве источника хранения информации базу данных SQLite 3.

В основе решения лежит сверточная нейронная сеть для классификации изображения ResNet18, реализованной с помощью Framework PyTorch.

Выбор PyTorch, обусловлен последними тенденциями в мире машинного обучения. На последних конференциях по машинному обучению, большинство докладов были выполнены с использованием данной библиотеки, что стало основным критерием при выборе.

Выбор протокола Modbus, обусловлен широким спектром использования при автоматизации производства, многие устройства поддерживают протокол передачи данных Modbus. Он является открытым, не требует лицензирования и прост в реализации. Поддерживает разные виды каналов передачи данных, как по интерфейсу RS-485, так и через Ethernet.

Протокол ONVIF, был выбран как стандартный протокол взаимодействия с видео камерами, имплементированный у всех производителей систем видео наблюдения. Функциональные возможности ONVIF аналогичны функциям API. Дополнительное преимущество использования ONVIF в том, что он подробно описывает, как сетевые IP-камеры, интегрируются с сетевыми программами обработки и отображения видеопотока.

Вывод:

Учитывая ТЗ, сформирован следующий стек технологий:

- выделенный сервер с ОС семейства Microsoft Windows;
- используемый язык – Python 3;
- среда разработки – PyCharm;
- база данных – SQLite версии 3;
- Web-интерфейс – Framework Django 3;
- обучение нейронной сети – Framework PyTorch;
- библиотеки – Onvif, PyModbus;
- система контроля версий – Git.

4.2 Разработка структуры базы данных

Для хранения данных в данном проекте предусматривается использование СУБД SQLite 3. Таблица базы данных спроектирована на основе классов, наследованных от классов django.Model (см. раздел 4.4). Непосредственная работа с БД будет осуществляться при помощи Django ORM. Структурная схема спроектированной базы данных приложения приведена на рис. 4.1.

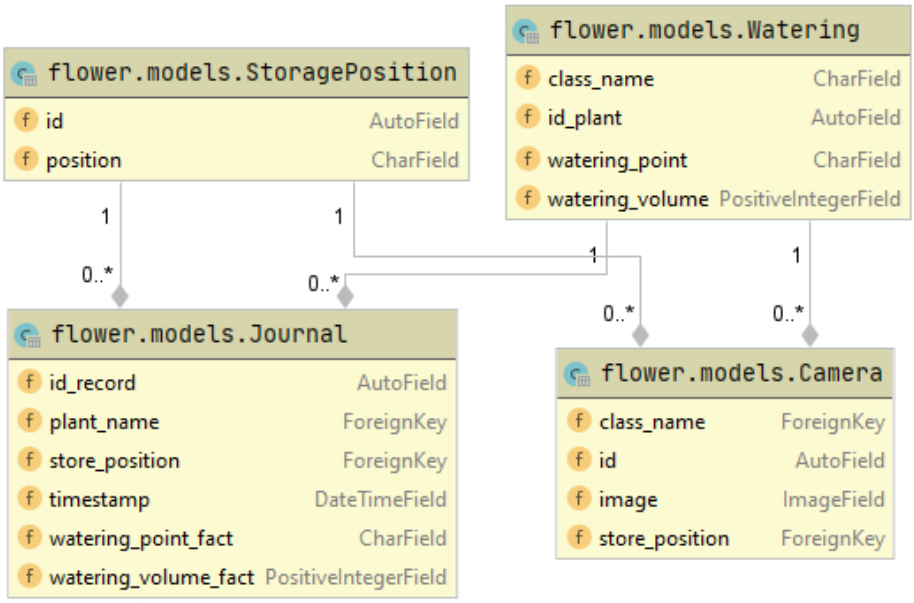

Рисунок 4.1 - Схема базы данных

БД имеет следующую структуру таблиц:

- **Watering** - содержит информацию о классе растения и параметрах полива;
- **Camera** – хранит накопленные фотографии с классом растения и их складских местах;
- **StoragePosition** - содержит список складских мест;
- **Journal** - содержит информацию о поливах.

Создание таблиц в БД отвечающих за хранение данных, относящихся к авторизации и аутентификация пользователей, реализуется посредством фреймворка Django. На схеме не приводится так как является типовым решением.

Поля всех таблиц БД и их назначение приведено в табл. 4.1

Таблица 4.1

Наименование поля	Тип поля	Назначение поля
Таблица **Watering**		
id_plant	INT, PK	Уникальный идентификатор
class_name	VARCHAR(50), UNIQUE	Название растение
watering_point	VARCHAR(50)	Тип полива
watering_volume	INT	Объем полива
Таблица **Camera**		
id	INT, PK	Уникальный идентификатор
class_name	ForeignKey	Связь с таблицей Watering
image	ImageField	Путь хранения изображения
store_possition	ForeignKey	Связь с таблицей StoragePossition
Таблица **StoragePosition**		
id	INT, PK	Уникальный идентификатор
possition	VARCHAR(10)	Складская позиция
Таблица **Journal**		
id_record	INT, PK	Уникальный идентификатор
timestamp	DATETIME	Дата и время полива
plant_name	ForeignKey	Связь с таблицей Watering
store_possition	ForeignKey	Связь с таблицей StoragePossition
watering_point _fact	VARCHAR(50)	Фактический тип полива
watering_volume_fact	INT	Фактический объем полива

4.3 Разработка модуля обработки

Задачи, выполняемые с помощью модуля обработки изображения, разбиваем на подмодули. Для каждого подмодуля создана своя директория с файлом инициализации, непосредственно самим файлом реализации задачи.

В файле *main.py* реализована основная логика работы модуля обработки. В бесконечном цикле, с помощью подмодуля системы полива отслеживается сигнал положения камеры. При получении состояния «камера на позиции» с помощью подмодуля взаимодействия с камерой получаем изображение, которое далее передается в подмодуль классификации. Результатом работы которого является класс растения. Согласно классу растения с помощью подмодуля взаимодействия с БД запрашивает информация о требуемых параметрах полива, после чего выдается команда на полив растения посредством подмодуля системы полива. Данные о поливе и изображение с классом растения записывается в БД с помощью соответствующего подмодуля.

Цикл повторяется до получения очередного сигнала о положении камеры. Дополнительные команды пользователем не вводятся. Модуль обработки работает автономно от WEB-интерфейса пользователя. Все взаимодействие осуществляется посредством обмена данными через единую базу данных.

4.3.1 Разработка подмодуля взаимодействия с базой данных

Для работы с базой данных, разработан подмодуль взаимодействия с СУБД.

Подмодуль взаимодействия с базой данных расположен в директории ‘/DataBase’. В файле *database.py*, реализован класс **DB**, содержащий функции работы с базой данных.

Экземпляр класса **DB** в методе **__init__** принимает db_path путь к базе данных. Основными методами в данном классе являются **– create_connection**, осуществляющий установку и проверку соединения и возвращающий

«соединение», **execute_read_query** метод, который отправляет запрос в базу данных и **execute_write_query** метод, осуществляющий запись в базу данных.

Чтение, запись и удаление, осуществляются путем передачи стандартного SQL запроса (SELECT, UPDATE, DELETE, INSERT) в методы **execute_read_query** и **execute_write_query** в параметре **query**.

4.3.2 Разработка подмодуля взаимодействия с камерой

Взаимодействие с системой получения изображений, т.е. IP-видеокамерой, осуществляется через API, реализованного с помощью протокола ONVIF.

ONVIF- это стандарт сферы видеонаблюдения, содержащий протоколы взаимодействия IP камер, IP серверов (кодировщиков), регистраторов, основанный на POST XML запросах.

Подмодуль взаимодействия с камерой расположен в директории ‘/Camera’. В файле *camera_onvif.py*, реализован класс **Camera**, описывающие взаимодействие с камерой.

Экземпляр класса **Camera** в методе **__init__** принимает ip адрес камеры. Основными методами в данном классе являются - **check_connection**, выполняющий функцию проверки соединения и возвращающий состояние соединения и метода **get_snapshot**, который осуществляет установку соединения, захват картинки и сохранения файла с текущим таймстампом.

Вспомогательный метод **set_snapshot_name** служит для формирования имени сохраняемого изображения.

4.3.3 Разработка подмодуля системы полива

Взаимодействие с системой полива, осуществляется через API, реализованного с помощью протокола ModBus.

Modbus — открытый коммуникационный протокол, основанный на архитектуре ведущий — ведомый (master-slave) [6].

Подмодуль взаимодействия с системой полива расположен в директории ‘/Modbus’. В файле *modbus.py*, реализован класс **ModbusDevice**, описывающий взаимодействие с системой полива.

В конструкторе класса **ModbusDevice** осуществляется соединение с устройством. Экземпляр данного класса в методе **__init__** принимает ip адрес контроллера.

Основными методами в данном классе являются - **modbus_check_connection**, выполняющий функцию проверки соединения и возвращаюсь состояние соединения, метод **modbus_read**, который осуществляет функцию чтения входных регистров и применяется для определения камеры на позиции, метод **modbus_plant_watering** осуществляет передачу команды на полив с требуемыми параметрами, метод **modbus_disconnect** разрывает соединение с контроллером.

4.3.4 Разработка подмодуля классификации

Подмодуль классификации расположен в директории ‘/Detector’. В файле *detect.py*, реализован класс **Predictor**, описывающие реализацию определения класса растения на основе предобученной нейронной сети ResNet.

Конструктор класса **Predictor**, в качестве параметра net_path принимает путь к сохраненной модели обученной нейронной сети. Основным методом является **predict**, определяющий класс растения. Который в качестве аргумента img_path принимает путь к изображению классифицируемого растения. Метод возвращает класс растения.

В файле *train.ipynb*, реализован набор классов, методов, функций необходимых для:

- загрузки и обработки **dataset**;
- нормализации изображений;
- загрузки предобученной нейронной сети;
- конфигурирования и переопределения слоев нейронной сети;

– цикла обучения, валидации и сбора статистики;
– проверка качества обучения и построение графиков.

Для обучения нейронной сети был собран **dataset** (522 изображения), представляющий три класса растений. В качестве модели сети, была выбрана архитектура ResNet18. Выбор данной архитектуры обусловлен ограниченными вычислительными ресурсами и средой исполнения - Google Colab.

Обучение нейронной сети выполнено с использованием фреймворка PyTorch. PyTorch — фреймворк машинного обучения для языка Python с открытым исходным кодом, созданный на базе Torch (MATLAB-подобная библиотека, предоставляющая большое количество алгоритмов для глубокого обучения) [6].

На рис. 4.2 представлены графики качества обучения настроенной нейронной сети, обученной на собранном **dataset**. Исходя из графика видно, что модель хорошо обучилась, ошибка стремиться к нулю, а качество к 1.

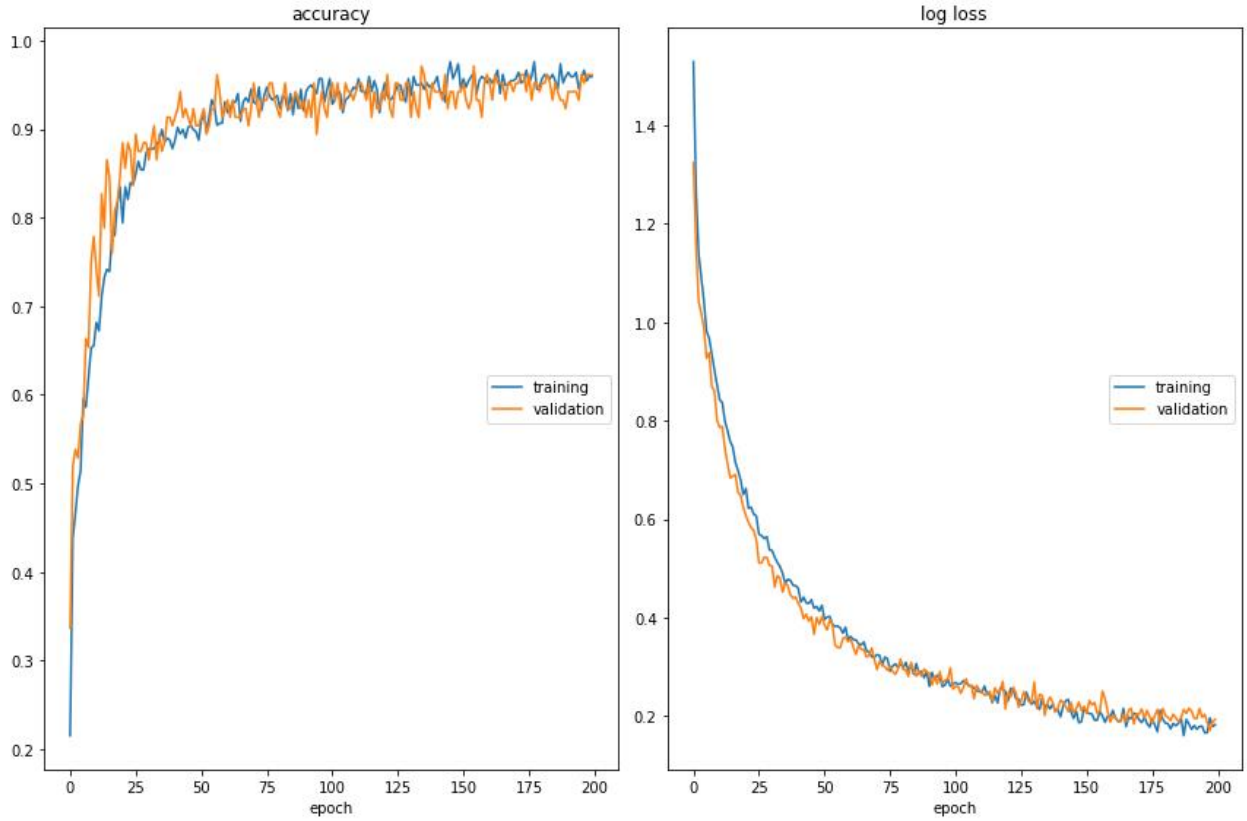

Рисунок 4.2 – Графики качества обучения нейронной сети

На рис. 4.3 представлено нормализованное изображение растения, пропущенного через обученную модель нейронной сети. В результате работы сети был получен класс растения, полностью совпадающий с действительным классом растения.

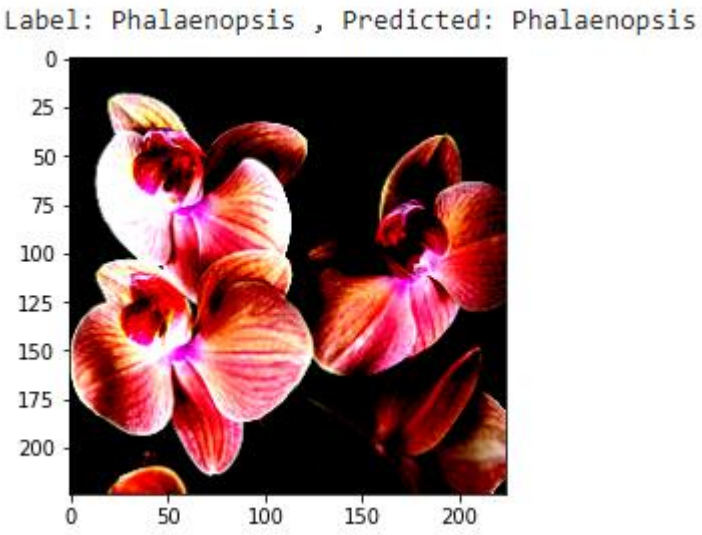

Рисунок 4.3 – Метка изображения совпадает с предсказанной

4.4 Разработка веб-интерфейса пользователя

Структура разрабатываемого Web-сервиса во многом обусловлена архитектурой используемого Web-фреймворка Django.

Фреймворк Django реализует шаблон проектирования MVT – Model-View-Template (Модель-Представление-Шаблон).

«Модель» (Model) в данной архитектуре является слоем доступа к данным. Этот слой знает, как получить к ним доступ, как проверить их, как с ними работать и как данные связаны между собой.

«Шаблон» (Template) – слой представления данных. Этот слой принимает решения относительно представления данных: как и что должно отображаться на странице или в другом типе документа.

«Представление» (View) является слоем бизнес-логики. Этот слой содержит логику, как получать доступ к моделям и применять соответствующий шаблон. Таким образом он осуществляет связь между моделями и шаблонами.

Взаимодействие слоев архитектуры фреймворка Django приведено на рис. 4.4.

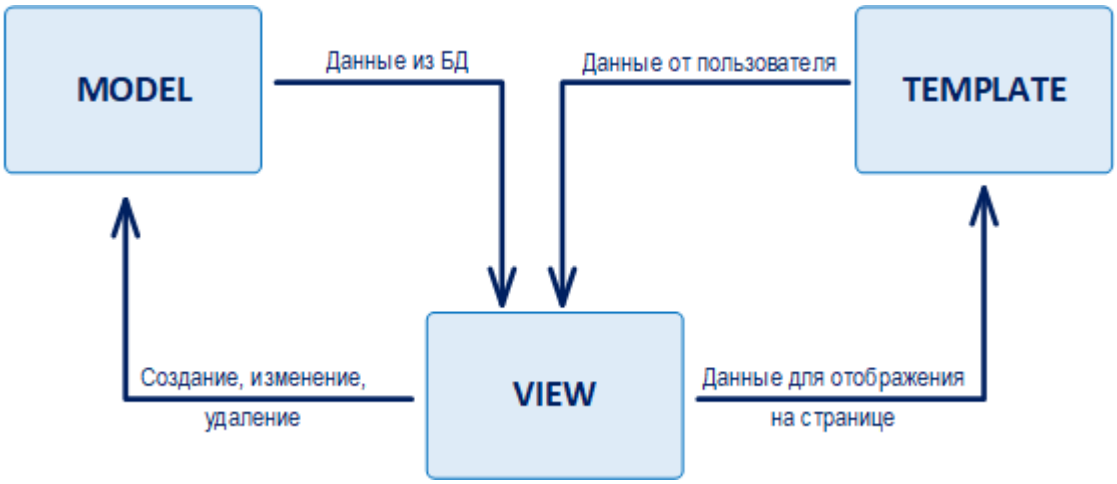

Рисунок 4.4 – Архитектура шаблона MVT фреймворка Django

4.4.1 Разработка слоя моделей

Для реализации модели данных, используются объекты Python, называемые моделями фреймворка Django 3. В файле **models.py**, реализованы классы моделей, которые наследуются от Model.

Исходя из функциональных требований ТЗ, разработан следующий набор классов слоя моделей:

Watering – описывает структуру данных таблицы, содержащую информацию о классе растения и параметрах полива, переопределены методы save и __str__;

Camera – описывает структуру данных таблицы, хранящую накопленные фотографии с классом растения и их складской позицией, переопределен метод __str__;

StoragePosition – описывает структуру данных таблицы, содержащую список складских мест, переопределен метод __str__**;**

Journal – описывает структуру данных таблицы, содержащую информацию о поливах, переопределен метод __str__.

Для атрибутов типа **models.CharField** передаем обязательный аргумент **max_length**. Атрибут **on_delete=models.cascade**, позволяет осуществлять каскадное удаление в связанных таблицах. В классах моделей реализуем метод **__str__**, который в дальнейшем позволит получать строковое представление объектов модели.

Описанные модели позволяют: получать, добавлять, сохранять, удалять и изменять записи в БД посредством обращения к объектам данных классов.

Диаграмма классов представлена на рис. 4.5.

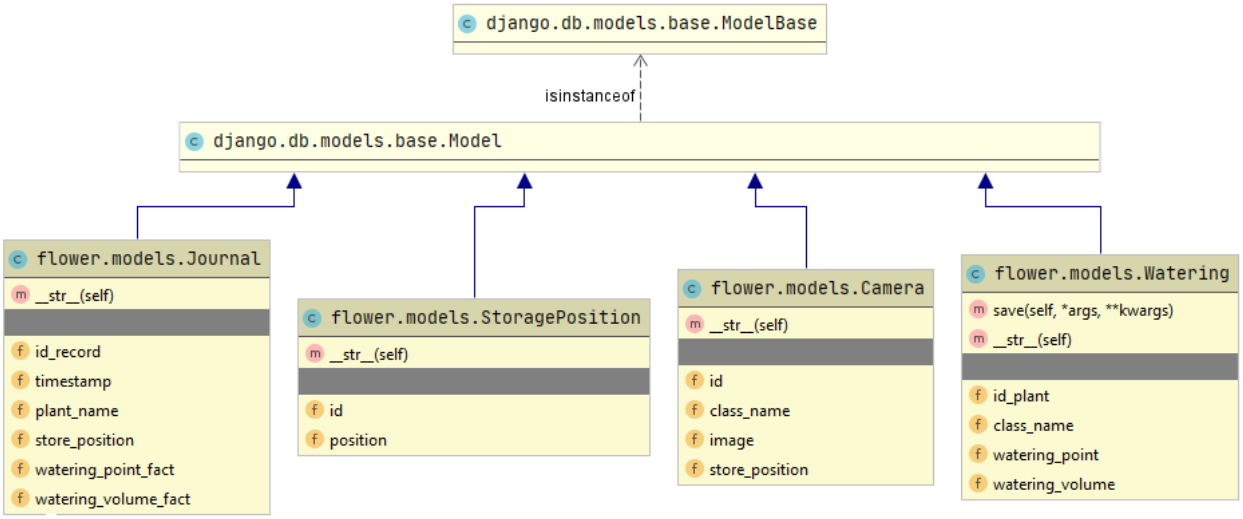

Рисунок 4.5 – Диаграмма классов слоя модели

4.4.2 Разработка слоя шаблонов

Для реализации функциональных требований ТЗ, а также требований к пользовательскому интерфейсу, предусмотрены шаблоны страниц Web-сервиса, предназначенные для вывода информации пользователю. Благодаря языку шаблонов Django мы имеем возможность выводить динамическое содержимое, формирующееся на основе данных от уровня model с помощью переменных, выделенных специальным тегом «{{ }}», а также шаблонных тегов «{% %}» и фильтров.

Разработаны следующие файлы шаблонов:

- **base.html** – базовый шаблон, который должен содержать базовую структуру web-страницы, заголовочные тэги, в нем же осуществляется подключение внешних зависимостей, таких как файлы css и js. Все последующие шаблоны наследуется от базового шаблона;
- **index.html** – шаблон, представляет собой главную страницу с общей информацией и навигацией по сайту;
- **watering.html** – шаблон, предназначенный для отображения параметров текущего/последнего полива, а также возможность внести изменения в параметры текущего полива;
- **settings.html** – шаблон, предназначенный для отображения списка имеющихся растений, а также добавления новых растений, удаления растений и для внесения изменений в параметры полива растений;
- **plant_create.html** – шаблон, который содержит форму для добавления новых растений;
- **plant_edit.html** – шаблон, который содержит форму для внесения изменения в параметры полива растений;
- **done.html** –шаблон, который будет использоваться после выполнения функций добавления, изменения и удаления растения.
- **journal.html** – шаблон, предназначенный для вывода отчетов о поливах;
- **login.html** – шаблон, предназначенный для авторизации и аутентификации пользователей;
- **logged_out.html** – шаблон, предназначенный для выхода текущего пользователя из системы.

Шаблоны реализованы внутри директории templates, находящейся в директории созданного приложения Django.

Django предоставляет встроенные функции представления для работы с авторизацией и выходом пользователя.

Для аутентификации пользователя происходит по переданному логину и паролю и использует функцию authenticate(). Она принимает два именованных аргумента, username и password, и возвращает объект User, если пароль соответствует логину. В противном случае функция возвращает None.

4.4.3 Разработка слоя представлений

В рамках фреймворка Django 3, слой представлений предназначен с одной стороны для получения, записи, изменения и удаления данных в БД с помощью обращений к слою модели и с другой стороны, рендера HTML файлов уровня шаблонов, для отображения динамических данных на страницах пользователя. Классы, разрабатываемые в рамках уровня представления должны наследоваться от класса **django.views.generic.View**. Для рендера страниц может использоваться функция **django.shortcuts.render**.

Исходя из функциональных требований ТЗ, разработан следующий набор классов слоя представлений:

- **MainView** – класс возвращает пользователю шаблон главной страницы Web-сервиса, в классе реализован метод get;
- **WaterinView** – класс отвечает за отображение данных о текущем/последнем поливе, и содержит в себе экземпляры класса Watering и Camera. В классе реализован метод get;
- **JournalView** – класс предназначен для получения списка всех поливов из БД и вывода его пользователю, посредством рендера соответствующего шаблона, реализованного через метод get;
- **SettingsView** – наследуется от класса LoginRequiredMixin для ограничения доступа к странице неавторизованному пользователю. Класс предназначен для получения списка всех растений из таблицы БД ‘Watering’ и вывода его пользователю, в классе реализован метод get;
- **LoginView** – класс предназначен для авторизации и аутентификации пользователей, реализован метод get;

- **LogoutView**– класс предназначен для выход текущего пользователя из системы, реализован метод get;
- **PlantCreate** – отвечает за создание записей о растениях в БД. Наследуется от класса LoginRequiredMixin, для ограничения доступа к странице неавторизованному пользователю. Реализованы методы get и post, для вывода формы добавления нового растения и для обработки данных, полученных от пользователя;
- **PlantEdit** – отвечает за изменения записей о растениях в БД. Наследуется от класса LoginRequiredMixin, для ограничения доступа к странице неавторизованному пользователю. Реализованы методы get и post, для вывода формы изменения выбранного растения и для обработки данных, полученных от пользователя;
- **PlantDelete** – наследуется от класса LoginRequiredMixin, для ограничения доступа к странице неавторизованному пользователю. Класс предназначен для удаления имеющихся растений, в классе реализован метод get;
- **LoginRequiredMixin** – генерируется автоматически с помощью фреймворка Django.

Таким образом, диаграмма классов слоя представления будет иметь вид, представленный на рис. 4.6.

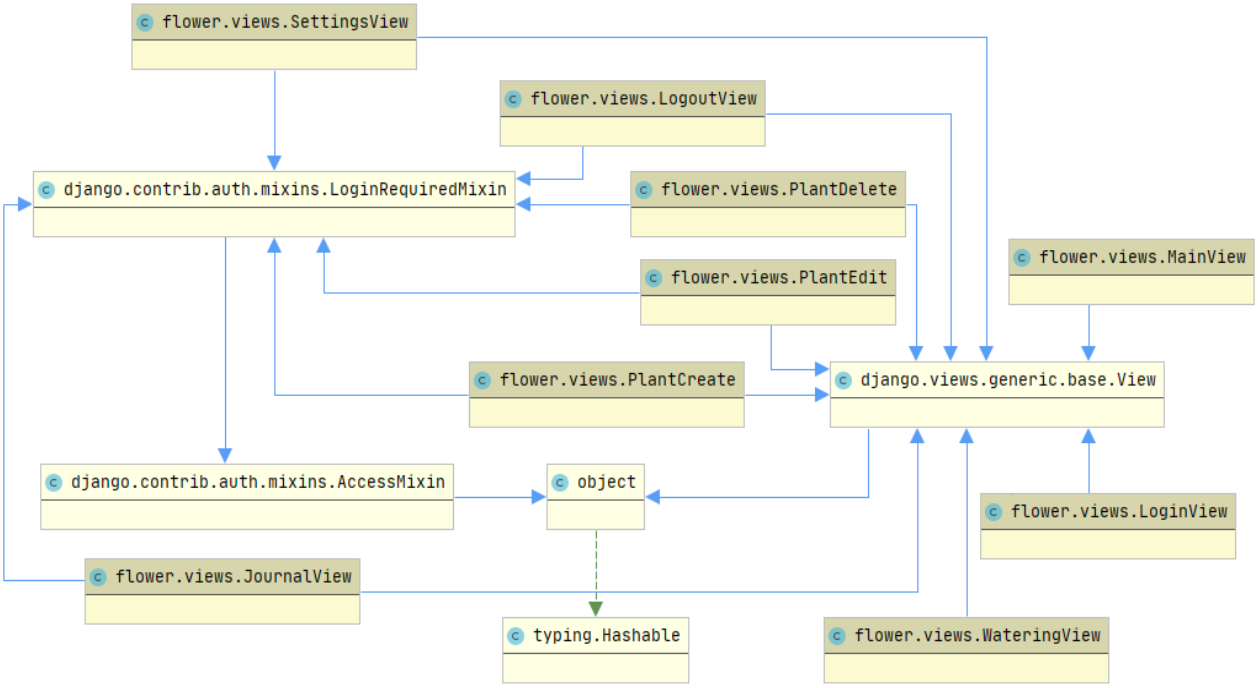

Рисунок 4.6 – Диаграмма классов слоя представлений

Так как все необходимые компоненты для отображения главной страницы уже реализованы, можно запустить проект и посмотреть на полученный результат. Ниже приведены примеры основных страниц.

Реализованный класс **MainView**, будет иметь вид, представленный на рис. 4.7.

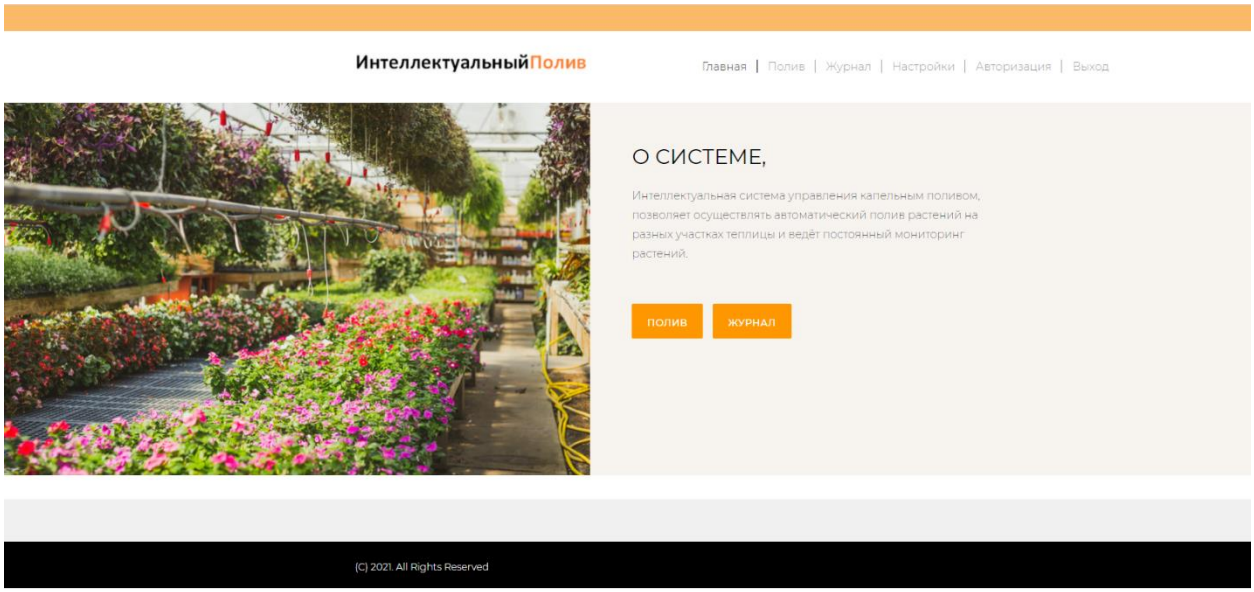

Рисунок 4.7 – Представление главной страницы

Реализованный класс **WaterinView**, будет иметь вид, представленный на рис. 4.8.

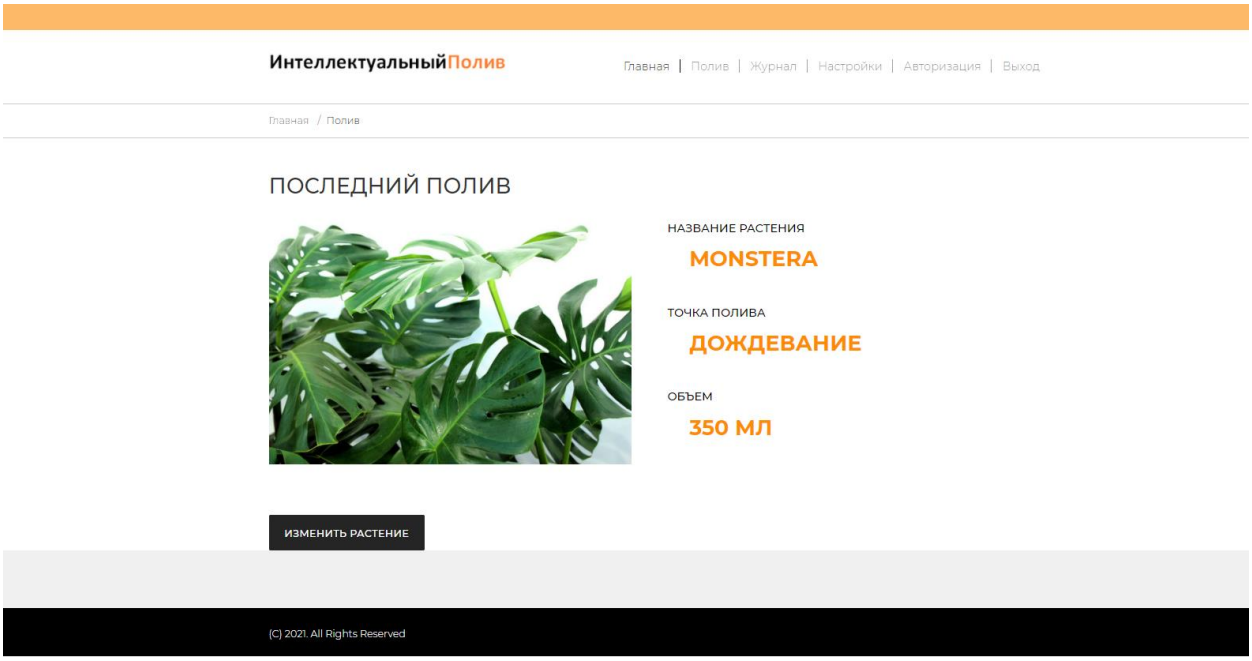

Рисунок 4.8 – Представление страницы полива

Реализованный класс **JournalView**, будет иметь вид, представленный на рисунке 4.9.

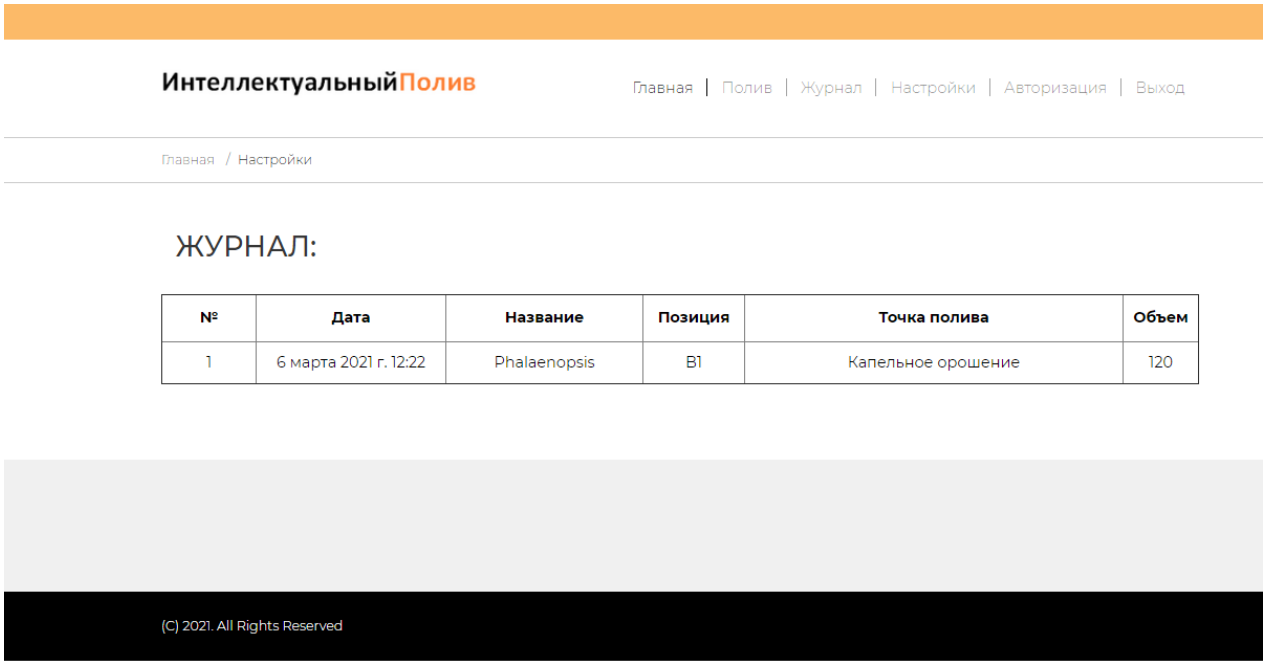

Рисунок 4.9 – Представление страницы Журнала

Реализованный класс **SettingsView**, будет иметь вид, представленный на рисунке 4.10.

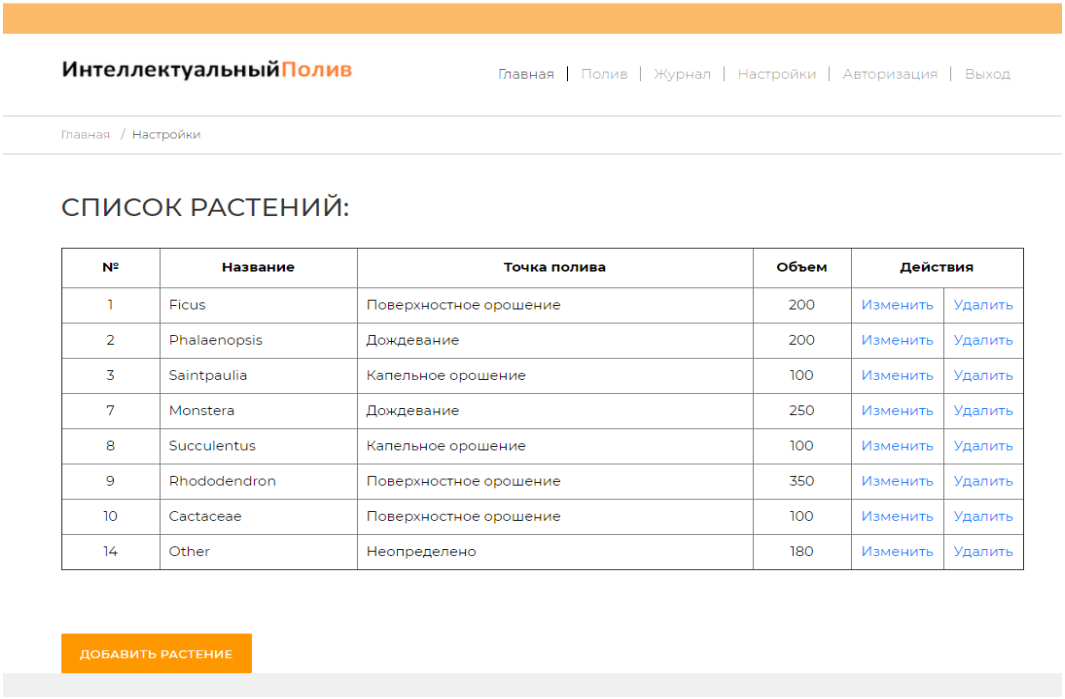

Рисунок 4.10 – Представление страницы Настроек

Реализованный класс **LoginView**, будет иметь вид, представленный на рисунке 4.11.

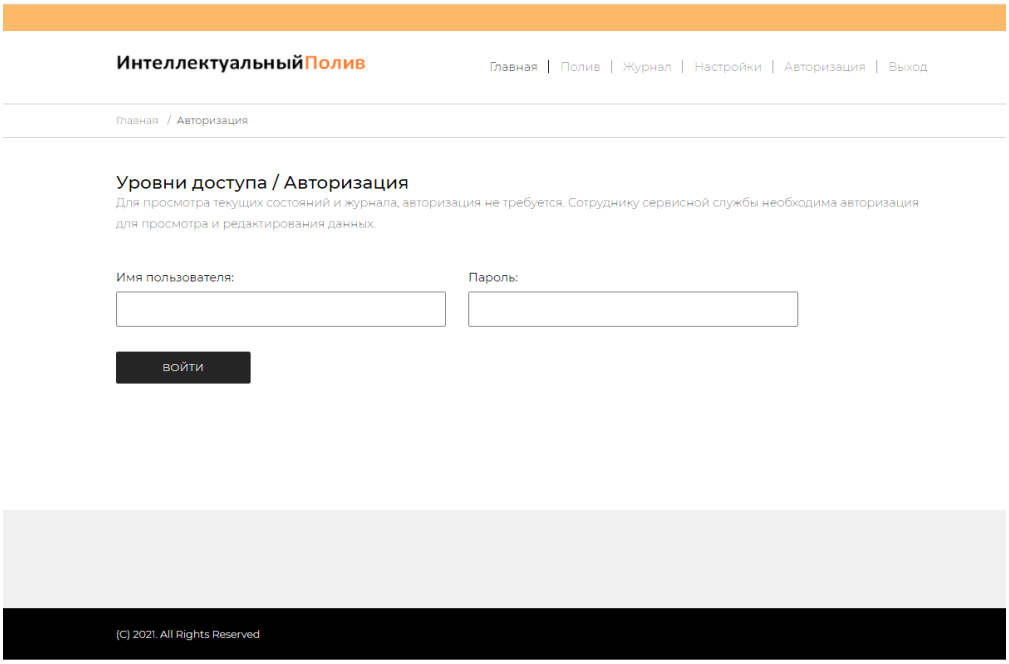

Рисунок 4.11 – Представление страницы авторизации

Реализованный класс **PlantCreate**, будет иметь вид, представленный на рисунке 4.12.

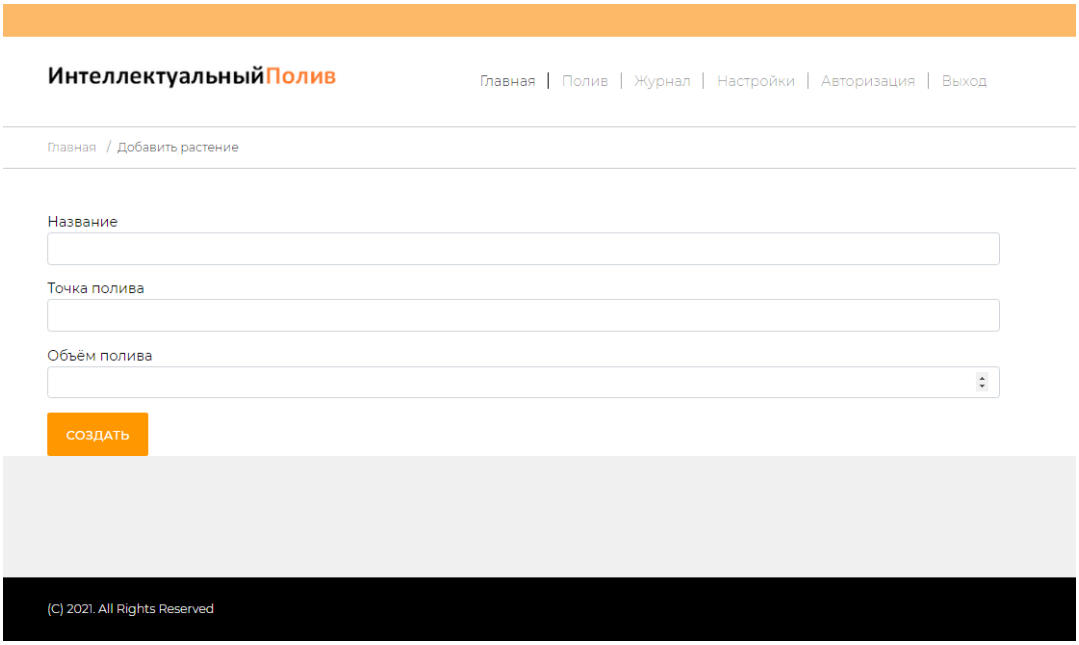

Рисунок 4.12 – Представление страницы добавления нового растения

Реализованный класс **PlantEdit**, будет иметь вид, представленный на рисунке 4.13.

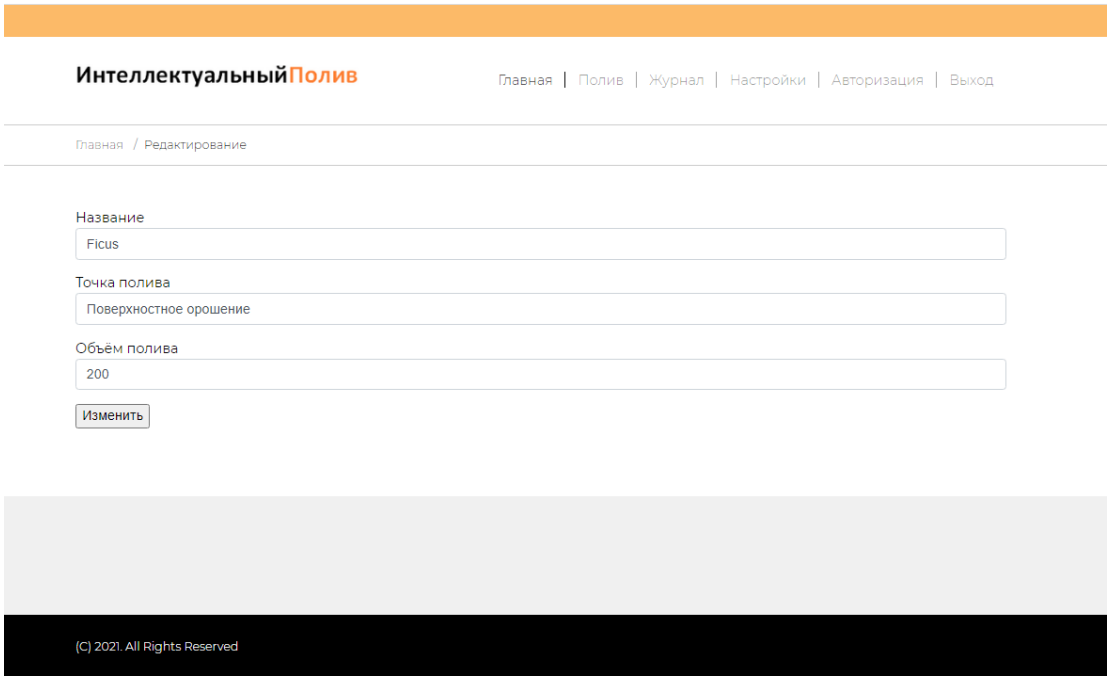

Рисунок 4.13 – Представление страницы Редактирования

5. ТЕСТИРОВАНИЕ

Тестирование проектируемого программного продукта ставит перед собой следующие цели:

- удостовериться, что отельные модули работают правильно – модульное тестирование;
- проверить успешную взаимную работу всех модулей целиком – интеграционное тестирование.

«Модульное тестирование, или юнит-тестирование (англ. unit testing) – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. Цель модульного тестирования – изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. [9]»

«Интеграционное тестирование (англ. Integration testing, иногда называется англ. Integration and Testing, аббревиатура англ. I&T) – одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию. Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определённые в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования [8].»

Тестирование приложения проводилось в несколько этапов для каждого разрабатываемого модуля:

1) База данных тестировалась на запись, удаление и чтение информации, взятой из набора тестовых данных.

2) Тестирование подмодулей проводилось как индивидуально, так и в интегрированном виде. Обработка данных выполнялась успешно, после завершения которой, обработанные данные в автоматическом режиме записывались в базу данных. Во время тестирования проводилось сравнение обработанных исследований, в ходе которого программа показала свою работоспособность.

3) Графический интерфейс проверялся на корректность вывода информации из базы данных. Была выполнена проверка на взаимодействие пользователя с графической частью – протестированы действия по удалению, изменению и сравнению данных, в ходе которых все блоки программы отработали корректно.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы все стоящие передо мной задачи были выполнены в достаточной мере. Проведен анализ существующего программного обеспечения на предмет возможности его использования. Были сформулированы требования к разрабатываемому приложению, спроектирована его архитектура и был разработан его прототип. Тестирование прототипа показало, что приложение удовлетворяет всем требованиям, представленным к нему. Оно в необходимой мере выполняет возложенный на него функционал.

В результате анализа был сделан вывод об возможности успешного внедрения программного обеспечения в рабочий процесс подразделения.

В дальнейшем возможна доработка существующего функционала.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт языка программирования Python. Документация по языку Python 3 [электронный документ] – (https://docs.python.org/3/). Проверено 22.03.2021.

2. Официальный сайт фреймворка Django 3. Документация [электронный документ] – (https://docs.djangoproject.com/en/3.1/). Проверено 22.03.2021.

3. Официальный сайт фреймворка PyTorch. Документация [электронный документ] – (https://pytorch.org/tutorials/). Проверено 22.03.2021.

4. Википедия. PyTorch [Электронный документ]. (https://ru.wikipedia.org/wiki/PyTorch). Проверено 22.03.2021.

5. Официальный сайт библиотеки PyModbus. Документация [электронный документ] – (https://pymodbus.readthedocs.io/en/latest/index.html). Проверено 22.03.2021.

6. Википедия. Modbus [Электронный документ]. (https://ru.wikipedia.org/wiki/Modbus#:~:text=Modbus%20%E2%80%94%20%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9%20%D0%BA%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9%20%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%2C%20%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9,%D0%BE%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8%20%D1%81%D0%B2%D1%8F%D0%B7%D0%B8%20%D0%BC%D0%B5%D0%B6%D0%B4%D1%83%20%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%BC%D0%B8%20%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%B0%D0%BC%D0%B8). Проверено 22.03.2021.

6. ONVIF TM Application Programmer's Guide Version 1.0 May 2011. (https://www.onvif.org/wp-content/uploads/2016/12/ONVIF_WG-APG-Application_Programmers_Guide-1.pdf) [Электронный документ]. Проверено 22.03.2021.

7. Википедия. Интеграционное тестирование: [Электронный документ]. (https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Проверено 22.03.2021.

9. Википедия. Модульное тестирование: [Электронный документ]. (https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).
Проверено 22.03.2021.

ПРИЛОЖЕНИЕ 1

Исходный код разработанного Web-сервиса

1. Файлы проекта web-интерфейса

Файл Flower_01/ Flower_01/settings.py:

```
1    import os
2    from pathlib import Path
3
4    # Build paths inside the project like this: BASE_DIR /
     'subdir'.
5    BASE_DIR = Path(__file__).resolve().parent.parent
6    PIC_DIR = Path(__file__).resolve().root
7
8    MEDIA_ROOT = os.path.join(BASE_DIR, 'flower/static')
9    MEDIA_URL = 'uploads/'
10
11
12   # SECURITY WARNING: keep the secret key used in production
      secret!
13   SECRET_KEY =
    'a=q^ayxdnu9si78zk@5k09vey1)krfnwp(sm8*b^m9+5re=%l3'
14
15   # SECURITY WARNING: don't run with debug turned on in
      production!
16   DEBUG = True
17
18   ALLOWED_HOSTS = []
19
20
21   # Application definition
22
23   INSTALLED_APPS = [
24       'flower.apps.FlowerConfig',
25       'django.contrib.admin',
26       'django.contrib.auth',
27       'django.contrib.contenttypes',
28       'django.contrib.sessions',
29       'django.contrib.messages',
30       'django.contrib.staticfiles',
31
32   ]
33
34   MIDDLEWARE = [
35       'django.middleware.security.SecurityMiddleware',
36       'django.contrib.sessions.middleware.SessionMiddleware',
37       'django.middleware.common.CommonMiddleware',
38       'django.middleware.csrf.CsrfViewMiddleware',
```

```
39
        'django.contrib.auth.middleware.AuthenticationMiddleware',
40      'django.contrib.messages.middleware.MessageMiddleware',
41      'django.middleware.clickjacking.XFrameOptionsMiddleware',
42  ]
43
44  ROOT_URLCONF = 'Flower_01.urls'
45
46  TEMPLATES = [
47      {
48          'BACKEND':
            'django.template.backends.django.DjangoTemplates',
49          'DIRS': [os.path.join(BASE_DIR, 'templates')]
50          ,
51          'APP_DIRS': True,
52          'OPTIONS': {
53              'context_processors': [
54                  'django.template.context_processors.debug',
55                  'django.template.context_processors.request',
56
                    'django.contrib.auth.context_processors.auth',
57
              'django.contrib.messages.context_processors.messages
              ',
58                ],
59          },
60      },
61  ]
62
63  WSGI_APPLICATION = 'Flower_01.wsgi.application'
64
65
66  # Database
67  #
    https://docs.djangoproject.com/en/3.1/ref/settings/#databases
68
69  DATABASES = {
70      'default': {
71          'ENGINE': 'django.db.backends.sqlite3',
72          'NAME': BASE_DIR / 'db.sqlite3',
73      }
74  }
75
76
77  # Password validation
78  # https://docs.djangoproject.com/en/3.1/ref/settings/#auth-
        password-validators
79
80  AUTH_PASSWORD_VALIDATORS = [
81      {
```

```
82          'NAME':
            'django.contrib.auth.password_validation.UserAttribute
            SimilarityValidator',
83      },
84      {
85          'NAME':
            'django.contrib.auth.password_validation.MinimumLength
            Validator',
86      },
87      {
88          'NAME':
            'django.contrib.auth.password_validation.CommonPasswor
            dValidator',
89      },
90      {
91          'NAME':
            'django.contrib.auth.password_validation.NumericPasswo
            rdValidator',
92      },
93  ]
94
95
96  # Internationalization
97  # https://docs.djangoproject.com/en/3.1/topics/i18n/
98
99  LANGUAGE_CODE = 'ru'
100
101
102 TIME_ZONE = 'UTC'
103
104 USE_I18N = True
105
106 USE_L10N = True
107
108 USE_TZ = True
109
110
111 # Static files (CSS, JavaScript, Images)
112 # https://docs.djangoproject.com/en/3.1/howto/static-files/
113
114 STATIC_URL = '/static/'
115
116 # Redirect to home URL after login (Default redirects to
    /accounts/profile/)
117 LOGIN_REDIRECT_URL = '/'
```

Файл Flower_01/ Flower_01/urls.py:

```
1   from django.contrib import admin
2   from django.urls import path, include
3
4   urlpatterns = [
```

```
5       path('admin/', admin.site.urls),
6       path('', include('flower.urls')),
7       path('accounts/', include('django.contrib.auth.urls'))
8   ]
```

Файл Flower_01/ Flower_01/ asgi.py:

```
1   import os
2
3   from django.core.asgi import get_asgi_application
4
5   os.environ.setdefault('DJANGO_SETTINGS_MODULE',
      'Flower_01.settings')
6
7   application = get_asgi_application()
```

Файл Flower_01/ Flower_01/ wsgi.py:

```
1   import os
2
3   from django.core.wsgi import get_wsgi_application
4
5   os.environ.setdefault('DJANGO_SETTINGS_MODULE',
    'Flower_01.settings')
6
7   application = get_wsgi_application()
```

2. Файлы приложения web-интерфейса

Файл Flower_01/ flower/ models.py:

```
1    from django.db import models
2    from django.shortcuts import reverse
3
4
5    class StoragePosition(models.Model):
6        """
7         Создаем таблицу, в которой будет храниться список
        складских мест
8        """
9        id = models.AutoField(primary_key=True)
10       position = models.CharField(max_length=10)
11
12       def __str__(self):
13           return f'{self.position}'
14
15
16   class Watering(models.Model):
17       """
```

```
18      Создаем таблицу, в которой будут храниться данные о
        классе растения и параметрах полива
19      """
20      id_plant = models.AutoField(primary_key=True)
21      class_name = models.CharField(unique=True, max_length=50)
22      watering_point = models.CharField(max_length=50)
23      watering_volume = models.PositiveIntegerField()
24      # watering_schedule =
        models.PositiveIntegerField(blank=True)
25
26      def save(self, *args, **kwargs):
27          super().save(*args, **kwargs)
28
29      def __str__(self):
30          return f'{self.class_name} '
31
32
33  class Camera(models.Model):
34      """
35      Создаем таблицу, в которой будут храниться накопленные
        фотографии с классом растения и их складскими  местами
36      """
37      id = models.AutoField(primary_key=True)
38      class_name = models.ForeignKey('Watering',
                            on_delete=models.CASCADE)
39      image = models.ImageField(blank=True,
         upload_to='uploads')
40      store_position = models.ForeignKey('StoragePosition',
                            on_delete=models.CASCADE)
41
42      def __str__(self):
43          return f'{self.class_name} - {self.store_position}'
44
45
46  class Journal(models.Model):
47      """
48      Создаем таблицу, в которой будут храниться данные о
        поливах
49      """
50      id_record = models.AutoField(primary_key=True)
51      timestamp = models.DateTimeField()
52      plant_name = models.ForeignKey('Watering',
                            on_delete=models.CASCADE)
53      store_position = models.ForeignKey('StoragePosition',
                            on_delete=models.CASCADE)
54      watering_point_fact = models.CharField(max_length=50)
55      watering_volume_fact = models.PositiveIntegerField()
56
57      def __str__(self):
58          return f'{self.plant_name} - {self.store_position} -
             {self.watering_point_fact} -
             {self.watering_volume_fact}'
```

Файл Flower_01/ flower/views.py:

```
1    from django.views.generic import View
2    from django.contrib.auth.mixins import LoginRequiredMixin
3    from django.shortcuts import render
4    from .forms import *
5    from .models import *
6
7
8    class MainView(View):
9        def get(self, request):
10           """
11           Открытие Главной страницы
12           :param request: request
13           :return: index.html
14           """
15           return render(request, 'flower/index.html')
16
17
18   class WateringView(View):
19       model = Watering
20       img = Camera
21
22       def get(self, request):
23           """
24           Вывод на странице информации  о текущем/последнем
             поливе
25           :param request:request
26           :return: watering.html
27           """
28           pic = Camera.objects.all().last()
29           pic_class = str(pic.class_name)[:-1]
30           plant = Watering.objects.get(class_name=pic_class)
31
32           return render(request, 'flower/watering.html',
                           context={'plant': plant, 'pic': pic})
33
34
35   class JournalView(LoginRequiredMixin, View):
36       redirect_field_name = 'login'
37
38       def get(self, request):
39           """
40           Вывод на странице списка всех поливов из БД. Страница
            доступна только авторизаванным пользователям
41           :param request:request
42           :return: journal.html
43           """
44           plant = Journal.objects.all()
45           return render(request, 'flower/journal.html',
                 {'plant': plant})
```

```
46
47
48  class SettingsView(LoginRequiredMixin, View):
49      redirect_field_name = 'login'
50
51      def get(self, request):
52
53          plant = Watering.objects.all()
54          return render(request, 'flower/settings.html',
                          {'plant': plant})
55
56
57  class LoginView(View):
58      def get(self, request):
59          return render(request, 'registration/login.html')
60
61
62  class LogoutView(LoginRequiredMixin, View):
63      redirect_field_name = 'login'
64
65      def get(self, request):
66          return render(request,
               'registration/logged_out.html')
67
68
69  class PlantCreate(LoginRequiredMixin, View):
70      model_form = SettingsForm
71      template = 'flower/plant_create.html'
72      redirect_field_name = 'login'
73
74      def get(self, request):
75          """
76          Вывод формы добавления нового растения. Страница
           доступна только авторизаванным пользователям
77          :param request:request
78          :return: plant_create.html
79          """
80          form = self.model_form()
81          return render(request, self.template,
                          context={'form': form})
82
83      def post(self, request):
84          """
85          Вывод формы для обработки и записи данных, полученных
           от пользователя
86          :param request:request
87          :return: plant_create.html
88          """
89          bound_form = self.model_form(request.POST)
90
91          if bound_form.is_valid():
92              bound_form.save()
```

```
93              func_st = 'Растение добавлено!'
94              return render(request, 'flower/done.html',
                              {'func_st': func_st})
95          return render(request, self.template,
                          context={'form': bound_form})
96
97
98  class PlantEdit(LoginRequiredMixin, View):
99      model = Watering
100     model_form = SettingsForm
101     template = 'flower/plant_edit.html'
102     redirect_field_name = 'login'
103
104     def get(self, request, id_plant):
105         """
106         Вывод формы изменения выбранного растения. Страница
            доступна только авторизаванным пользователям
107         :param request: request
108         :param id_plant: номер записи
109         :return: plant_edit.html
110         """
111         obj = self.model.objects.get(
                      id_plant__exact=id_plant)
112         bound_form = self.model_form(instance=obj)
113         return render(request, self.template,
                          context={'form': bound_form,
                          self.model.__name__.lower(): obj})
114
115     def post(self, request, id_plant):
116         """
117         Вывод формы для обработки и записи данных, полученных
            от пользователя
118         :param request: request
119         :param id_plant: номер записи
120         :return: plant_edit.html
121         """
122         obj = self.model.objects.get(
                      id_plant__exact=id_plant)
123         bound_form = self.model_form(request.POST,
                      instance=obj)
124
125         if bound_form.is_valid():
126             bound_form.save()
127             func_st = 'Запись о растении изменена!'
128             return render(request, 'flower/done.html',
                              {'func_st': func_st})
129         return render(request, self.template,
                          context={'form': bound_form,
                          self.model.__name__.lower(): obj})
130
131
132 class PlantDelete(LoginRequiredMixin, View):
```

```
133     redirect_field_name = 'login'
134
135     def get(self, request, id_plant):
136         """
137         Функция удаления выбранного растения из БД. Страница
            доступна только авторизаванным пользователям
138         :param request: request
139         :param id_plant: номер записи
140         :return: done.html
141         """
142         plant = Watering.objects.get(
                    id_plant__exact=id_plant)
143         plant.delete()
144         func_st = 'Запись о растении удалена!'
145         return render(request, 'flower/done.html',
                          {'func_st':
                          func_st})
```

Файл Flower_01/ flower/forms.py:

```
1   from django import forms
2   from .models import *
3
4
5   class SettingsForm(forms.ModelForm):
6       """
7       Создаем форм из моделей
8       """
9
10      class Meta:
11          """
12           Создаем форму модели Watering
13          """
14          model = Watering
15          fields = ['class_name', 'watering_point',
                      'watering_volume']
16          labels = {'class_name': 'Название', 'watering_point':
                    'Точка полива', 'watering_volume': 'Объём
                    полива'}
17          widgets = {'class_name':
                    forms.TextInput(attrs={'class': 'form-
                    control'}),
18                     'watering_point':
                     forms.TextInput(attrs={'class': 'form-
                     control'}),
19                     'watering_volume':
                    forms.NumberInput(attrs={'class': 'form-
                    control'})
20                     }
```

Файл Flower_01/ flower/urls.py:

```
1   from django.urls import path
```

```
2    from .views import *
3    from . import views
4
5    from django.contrib.staticfiles.urls import
       staticfiles_urlpatterns
6    from django.conf import settings
7    from django.conf.urls.static import static
8
9    urlpatterns = [
10       path('', MainView.as_view(), name='home'),
11       path('watering/', WateringView.as_view(), name='water'),
12       path('journal/', JournalView.as_view(), name='journal'),
13       path('settings/', SettingsView.as_view(), name='sett'),
14       path('login/', LoginView.as_view(), name='login'),
15       path('logout/', views.LogoutView.as_view(),
             name='logout'),
16       path('update/edit/<int:id_plant>/', PlantEdit.as_view(),
             name='plant_edit'),
17       path('create/', PlantCreate.as_view(),
             name='plant_create'),
18       path('delete/<int:id_plant>/', PlantDelete.as_view(),
         name='plant_delete'),
19   ]
20
21   urlpatterns += static(settings.MEDIA_URL,
                 document_root=settings.MEDIA_ROOT)
22
23     urlpatterns += staticfiles_urlpatterns()
```

ПРИЛОЖЕНИЕ 2

Исходный код разработанного модуля обработки

Файл Detection/ main.py:

```
1    from Modbus.modbus import *
2    from DataBase.database import *
3    from Detector.detector import *
4    from Camera.camera_onvif import *
5    import configparser
6    import time
7
8
9    def main():
10       # IP адреса устройств, к которым будем подключаться
           беруться из файла настроек
11       config = configparser.ConfigParser()
12       # TODO: сделать нормальные относительные пути
13       config.read("D:/Diploma/Detection/settings.ini")
14
15       # создаем экземляры наших устройств
16       db = DB('D:/Diploma/Flower_01/db.sqlite3')
17       cam = Camera(config["CAMERA"]["IP"])
18       device = ModbusDevice(config["MODBUS"]["IP"])
19       detector = Detector(
             "D:/Diploma/Detection/Detector/model_all.pth")
20
21       # Проверка соединений
22       if cam.check_connection & device.modbus_check_connection:
23           print("Camera and device connection created.")
24
25       # отслеживаем изменение состояние входа
26       trigger = False
27       # складская позиция фиксированная
28       store_position_id = 1
29
30       while True:
31           system_stop = device.modbus_read(
                config["MODBUS"]["DI2"])
32           input_state = device.modbus_read(
                config["MODBUS"]["DI1"])
33
34           # Если подана команда на остановку (2 регистр в ON)
               выходим из цикла и завершаем программу
35           if system_stop:
36               break
37
38           elif input_state & (input_state != trigger):
39               # Захватываем изображение. Получаем имя файла с
                   изображением
```

```
40              file_name = cam.get_snapshot()
41              # Определяем класс растения
42              img_class = detector.predict(
                  config["CAMERA"]["DATA_DIR"] + "/" + file_name)
43              # Соединяемся с БД
44              db_connection = db.create_connection()
45              # Формируем запрос к БД о параметрах полива по
                  классу растения
46
47              db_query_watering = "SELECT id_plant, class_name,
                  watering_volume, watering_point  " \
48                                  "FROM flower_watering WHERE
                   class_name='" + img_class + "'"
49
50              db_result_watering = db.execute_read_query(
                  db_connection, db_query_watering)
51              plant_name_id = db_result_watering[0][0]
52              watering_volume_fact = db_result_watering[0][2]
53              watering_point_fact = db_result_watering[0][3]
54
55              # Формируем запрос к БД запись изображения
56              db_query_camera = "INSERT INTO flower_camera (" \
57                                "'class_name_id' , " \
58                                "'store_position_id', " \
59                                "'image') " \
60                                "VALUES ('"\
61                                + str(plant_name_id) + "', '" \
62                                + str(store_position_id) + "',
                                    'uploads/" + file_name + "')"
63
64              db.execute_write_query(db_connection,
                      db_query_camera)
65
66              # добавить полив
67              if watering_point_fact == "Дождевание":
68                  watering_zone = 1
69              elif watering_point_fact == "Дождевание":
70                  watering_zone = 2
71              elif watering_point_fact == "Дождевание":
72                  watering_zone = 3
73              else:
74                  watering_zone = 0
75
76              watering_result = device.modbus_plant_watering(
                   watering_zone, watering_volume_fact)
77
78              # Получаем время операции для журнала
79              ts = datetime.datetime.now()
80              timestamp = str(ts.year) + \
81                          "-" + \
82                          str(ts.month) + \
83                          "-" + \
```

```
84                              str(ts.day) + \
85                              " " + \
86                              str(ts.hour) + \
87                              ":" + \
88                              str(ts.minute) + \
89                              ":" + \
90                              str(ts.second)
91                  # Формируем запрос к БД запись изображения
92                  db_query_journal = "INSERT INTO flower_journal ("
93                                     "'timestamp' , " \
94                                     "'watering_volume_fact', " \
95                                     "'plant_name_id', " \
96                                     "'store_position_id', " \
97                                     "'watering_point_fact') " \
98                                     "VALUES ('" \
99                                     + str(timestamp) + "', '" \
100                                    + str(watering_volume_fact) +
                                         "', '" \
101                                    + str(plant_name_id) + "', '"
102                                    + str(store_position_id) + "',
                                          '" \
103                                    + str(watering_point_fact) +
                                         "')"
104
105                 if watering_result:
106                     db.execute_write_query(db_connection,
                          db_query_journal)
107
108                 db_connection.close()
109
110                 trigger = input_state
111
112             elif not input_state:
113                 trigger = input_state
114
115             time.sleep(5)
116
117         device.modbus_disconnect()
118         return
119
120
121     if __name__ == '__main__':
122         main()
```

Файл Detection/settings.ini:

```
1    [COMMON]
2    [MODBUS]
3    IP=192.168.0.200
4    PORT=502
5    DI1=101
6    DI2=132
```

```
7    ZONE_LOW=125
8    ZONE_MID=125
9    ZONE_HIGH=125
10   VOLUME=431
11   [CAMERA]
12   IP=192.168.0.20
13   URL=http://192.168.0.20/jpeg/jpeg.jpg
14   DATA_DIR=D:\Diploma\Flower_01\flower\static\uploads
15   [DATABASE]
16   DB_PATH=D:\Diploma\Flower_01\db.sqlite3
```

Файл Detection/ Camera/ camera_onvif.py:

```
1    # версия от 2020.03.16
2
3    import os
4    import datetime
5    import requests
6    import configparser
7
8
9    def set_snapshot_name():
10       """ Функция создания имени файла
11       Генерирует имя файл из текущей даты и времени
12       """
13
14       ts = datetime.datetime.now()
15       file_name = str(ts.year) + \
16                   "-" + \
17                   str(ts.month) + \
18                   "-" + \
19                   str(ts.day) + \
20                   "-" + \
21                   str(ts.hour) + \
22                   "-" + \
23                   str(ts.minute) + \
24                   "-" + \
25                   str(ts.second) + \
26                   ".jpg"
27       return file_name
28
29
30   class Camera:
31       """ Базовый класс для камер
32       :param ip: IP адрес камеры (default 192.168.0.20)
33       """
34
35       def __init__(self, ip='192.168.0.20'):
36           self.ip = ip
37
38           # чтение и загрузка конфигурации из файла
39           config = configparser.ConfigParser()
```

```
40          config.read("D:/Diploma/Detection/settings.ini")
41
42          self.data_dir = config["CAMERA"]["DATA_DIR"]
43          os.chdir(self.data_dir)
44
45      @property
46      def check_connection(self):
47          """ Функция проверки соединения
48          """
49          try:
50              response = requests.get("http://" + self.ip)
51              if response.status_code == 200:
52                  print(f"Camera with {self.ip} is online.")
53              return True
54          except requests.ConnectionError:
55              print(f"Camera with {self.ip} is offline")
56              return False
57
58      def get_snapshot(self):
59          """ Функция захвата картинки и сохраниния файла с
            текущим таймстампом
60
61          """
62
63          file_snapshot_name = set_snapshot_name()
64          url = 'http://' + self.ip + '/jpeg/jpeg.jpg'
65          r = requests.get(url)
66          with open(self.data_dir + '\\' + file_snapshot_name,
             'wb') as f:
67              f.write(r.content)
68          return file_snapshot_name
```

Файл Detection/ Camera/ database.py:

```
1   import sqlite3
2
3
4   class DB:
5       def __init__(self, db_path):
6           self.db_path = db_path
7
8       def create_connection(self):
9           """ Функция соединения с базой данных
10
11          :return: Возвращает "соединение"
12          """
13          sqlite_connection = None
14          try:
15              sqlite_connection = sqlite3.connect(self.db_path)
16              print("Connection to SQLite DB successful")
17          except sqlite3.Error as e:
18              print(f"The error '{e}' occurred")
```

```
19
20          return sqlite_connection
21
22      @staticmethod
23      def execute_read_query(connection, query):
24          """ Извлечение данных из записей
25
26          :param connection: Принимает "соединение"
27          :param query: Запрос в формате SQL
28          :return: Возвращает резуьтат запроса
29          """
30          cursor = connection.cursor()
31          result = None
32          try:
33              cursor.execute(query)
34              result = cursor.fetchall()
35              cursor.close()
36              return result
37          except sqlite3.Error as e:
38              print(f"The error '{e}' occurred")
39
40      @staticmethod
41      def execute_write_query(connection, query):
42          """
43          Запись данных в базу данных
44          :param connection: Принимает "соединение"
45          :param query: Запрос в формате SQL
46          """
47          try:
48              cursor = connection.cursor()
49              print("Подключен к SQLite")
50
51              cursor.execute(query)
52              connection.commit()
53              print("Запись успешно вставлена в таблицу ",
                      cursor.rowcount)
54              cursor.close()
55          except sqlite3.Error as error:
56              print("Ошибка при работе с SQLite", error)
```

Файл Detection/ Detector/ detector.py:

```
1   import torch
2   import torchvision as tv
3   from PIL import Image
4
5
6   class Detector:
7       """ Базовый класс для детектора
8
9       """
```

```
10      def __init__(self, model_path="", classes=['Ficus',
            'Unknown', 'Saintpaulia', 'Phalaenopsis']):
11          """
12          Конструктор класса Detector
13          :param model_path: путь к сохраненной модели
14          :param classes: классы растений из обученного
             датасета
15          """
16          self.model_path = model_path
17          self.classes = classes
18          self.model = torch.load(model_path,
              map_location=torch.device('cpu'))
19          self.model.eval()
20
21      @staticmethod
22      def image_transform(image):
23          """ Функциця преоразования картинки в тензор и
             нормализации
24
25          :param image: входное изображение
26          :return: возвращает нормализованный тензор
27          """
28
29          stats = ((0.485, 0.456, 0.406), (0.229, 0.224,
                    0.225))
30          normalize = tv.transforms.Normalize(*stats,
               inplace=True)
31          transforms = tv.transforms.Compose([
32              tv.transforms.ToTensor(),
33              normalize
34          ])
35
36          return transforms(image)
37
38      def predict(self, image_path):
39          """ Функция классификации растения
40
41          :param image_path: путь к изображению растения
42          :return: возвращает класс растения
43          """
44
45          with open(image_path, 'rb') as f:
46              img = Image.open(f)
47              img.convert('RGB')
48          image = self.image_transform(img)
49          xb = image.unsqueeze(0)
50
51          # Получаем предсказание класса растения от модели
52          yb = self.model(xb)
53
54          # Выбираем наибольную вероятность
55          _, preds = torch.max(yb, dim=1)
```

```
56          value = torch.max(yb, dim=1)
57
58          # Если коэффициент предсказания меньше 0, то считаем,
              что модель не угадала и даем класс "Unknown"
59          if value[0].item() > 0:
60              # Retrieve the class label
61              return self.classes[preds[0].item()]
62          else:
63              return "Unknown"
64
```

Файл Detection/ Detector/ train.py:

```
1   # -*- coding: utf-8 -*-
2   """FC_3.ipynb
3
4   Automatically generated by Colaboratory.
5
6   Original file is located at
7
    https://colab.research.google.com/drive/1R8m_TZcUr9dvJ10siIUS
    HAlFJVtLWuNC
8
9   Уставнока библиотек
10  """
11
12  pip install yadisk --quiet
13
14  pip install livelossplot --quiet
15
16  # Commented out IPython magic to ensure Python compatibility.
17  # Импорт необходимых библиотек
18  import torch
19  from torch import nn
20  from torch import optim
21  from livelossplot import PlotLosses
22  import torchvision as tv
23  import time
24  import os
25  import yadisk
26  import tqdm.notebook as tqdm
27  import matplotlib.pyplot as plt
28  from torch.utils.data import Dataset
29  from torch.utils.data import DataLoader
30  from PIL import Image
31  # %matplotlib inline
32
33  """Загрузка датасета"""
34
35  # Константы и параметы
36  token_ya ="AgAAAAAEmIbuAAZqsLC2BS8SQU_woAdey85OBsI"
    # Токен Яндекс диска
```

```
data_dir = './Dataset'
dataset_file = 'Dataset.zip'

# Загрузка файла с датасетом

y = yadisk.YaDisk(token=token_ya)
print("Downloading", dataset_file, 'from Yandex.Disk')
y.download('/'+ dataset_file, dataset_file)
print("Unpacking", dataset_file, "...")
!unzip -q Dataset.zip
print("Deleting", dataset_file, "...")
if os.path.isfile(dataset_file):
    os.remove(dataset_file)
    print("Deleted", dataset_file)
else:    ## Show an error ##
    print("Error: %s file not found" % myfile)

# Служебная функция, потом удалить
# Удаление папки Dataset
# import shutil
# if os.path.exists(data_dir ) and os.path.isdir(data_dir):
#     shutil.rmtree(data_dir )

"""Приведение данных к необходимому формату"""

print(os.listdir(data_dir))

# Переименование файлов с разбивкой по классам
def rename_files(root_dir):
    classes = os.listdir(root_dir)
    for classes in classes:
        for file in os.listdir(root_dir + '/' + classes):
            if file.endswith('jpg'):
                os.rename((root_dir + '/' + classes + '/' +
 file),(root_dir + '/' + classes + '/' + classes + "_" +
 file))

rename_files(data_dir)

def parse_species(fname):
    parts = fname.split('_')
    return parts[0]

def open_image(path):
    with open(path, 'rb') as f:
        img = Image.open(f)
        return img.convert('RGB')

class FlowersDataset(Dataset):
    def __init__(self, root_dir, transform):
        super().__init__()
        self.root_dir = root_dir
```

```
87          self.files = []
88          self.classes = [fname for fname in
                    os.listdir(root_dir) if fname != 'flowers']
89          for classes in self.classes:
90              for file in os.listdir(root_dir + '/' + classes):
91                  if file.endswith('jpg'):
92                      self.files.append(file)
93          self.transform = transform
94
95      def __len__(self):
96          return len(self.files)
97
98      def __getitem__(self, i):
99          fname = self.files[i]
100         species = parse_species(fname)
101         fpath = os.path.join(self.root_dir, species, fname)
102         img = self.transform(open_image(fpath))
103         class_idx = self.classes.index(species)
104         return img, class_idx
105
106 stats = ((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
107 normalize = tv.transforms.Normalize(*stats, inplace=True)
108 transoforms = tv.transforms.Compose([
109     tv.transforms.RandomHorizontalFlip(),
110     tv.transforms.RandomVerticalFlip(),
111     tv.transforms.ToTensor(),
112     normalize
113     ])
114
115 transoforms_pred = tv.transforms.Compose([
116     tv.transforms.ToTensor(),
117     normalize
118     ])
119
120 dataset = FlowersDataset(data_dir, transform=transoforms)
121
122 print("Датасет состит из" ,len(dataset))
123
124 def denormalize(images, means, stds):
125     if len(images.shape) == 3:
126         images = images.unsqueeze(0)
127     means = torch.tensor(means).reshape(1, 3, 1, 1)
128     stds = torch.tensor(stds).reshape(1, 3, 1, 1)
129     return images * stds + means
130
131 def show_image(img_tensor, label):
132     print('Label:', dataset.classes[label], '(' + str(label)
            + ')')
133     img_tensor = denormalize(img_tensor,
          *stats)[0].permute((1, 2, 0))
134     plt.imshow(img_tensor)
135
```

```
136  show_image(*dataset[28]);
137
138  """Data Loaders
139
140  """
141
142  device = torch.device("cuda") if torch.cuda.is_available()
                else torch.device("cpu")
143  device
144
145  from torch.utils.data import random_split
146
147  random_seed = 20
148  torch.manual_seed(random_seed)
149
150  val_pct = 0.2
151  val_size = int(val_pct * len(dataset))
152  train_size = len(dataset) - val_size
153
154  train_ds, valid_ds= random_split(dataset, [train_size,
val_size])
155  len(train_ds), len(valid_ds)
156
157  # Внимание! Если считаем на CPU то num_workers=0. Если на GPU
     num_workers=3.
158
159  batch_size = 32
160
161  if device == "cpu":
162    train_dl = DataLoader(train_ds, batch_size, shuffle=True,
           num_workers=0, pin_memory=True)
163    valid_dl = DataLoader(valid_ds, batch_size*2,
           num_workers=0, pin_memory=True)
164  else:
165    train_dl = DataLoader(train_ds, batch_size, shuffle=True,
           num_workers=2, pin_memory=True)
166    valid_dl = DataLoader(valid_ds, batch_size*2,
           num_workers=2, pin_memory=True)
167
168  dataloaders = {
169      "train": train_dl,
170      "validation": valid_dl
171  }
172
173  print(train_dl.dataset.dataset.classes)
174
175  from torchvision.utils import make_grid
176
177  def show_batch(dl):
178      for images, labels in dl:
179          fig, ax = plt.subplots(figsize=(12, 12))
180          ax.set_xticks([]); ax.set_yticks([])
```

```
181         images = denormalize(images, *stats)
182         ax.imshow(make_grid(images[:64], nrow=8).permute(1,
               2, 0))
183         break
184
185 show_batch(train_dl)
186
187 """Формируем модель Model Architecture  ResNet18"""
188
189 model = tv.models.resnet18(pretrained=True)
190
191 ## Убираем требование градиента:
192 for param in model.parameters():
193     param.requires_grad = False
194
195 model.fc = nn.Linear(in_features=512, out_features=4)
196
197 model.named_modules
198
199 params_to_update = []
200 for name,param in model.named_parameters():
201     if param.requires_grad == True:
202         params_to_update.append(param)
203
204 """Тренируем модель"""
205
206 def train_model(model, criterion, optimizer, num_epochs=10):
207     liveloss = PlotLosses()
208     model = model.to(device)
209
210     for epoch in range(num_epochs):
211         logs = {}
212         for phase in ['train', 'validation']:
213             if phase == 'train':
214                 model.train()
215             else:
216                 model.eval()
217
218             running_loss = 0.0
219             running_corrects = 0
220
221             for inputs, labels in dataloaders[phase]:
222                 inputs = inputs.to(device)
223                 labels = labels.to(device)
224
225                 outputs = model(inputs)
226                 loss = criterion(outputs, labels)
227
228                 if phase == 'train':
229                     optimizer.zero_grad()
230                     loss.backward()
231                     optimizer.step()
```

```
232
233                 _, preds = torch.max(outputs, 1)
234                 running_loss += loss.detach() *
                     inputs.size(0)
235                 running_corrects += torch.sum(preds ==
                     labels.data)
236
237             epoch_loss = running_loss /
                          len(dataloaders[phase].dataset)
238             epoch_acc = running_corrects.float() /
                          len(dataloaders[phase].dataset)
239
240             prefix = ''
241             if phase == 'validation':
242                 prefix = 'val_'
243
244             logs[prefix + 'log loss'] = epoch_loss.item()
245             logs[prefix + 'accuracy'] = epoch_acc.item()
246
247         liveloss.update(logs)
248         liveloss.send()
249
250 criterion = nn.CrossEntropyLoss()
251 optimizer = optim.Adam(model.parameters(), lr=0.001)
252
253 train_model(model, criterion, optimizer, num_epochs=200)
254
255 def to_device(data, device):
256     """Move tensor(s) to chosen device"""
257     if isinstance(data, (list,tuple)):
258         return [to_device(x, device) for x in data]
259     return data.to(device, non_blocking=True)
260
261 # Функция предсказания картинки
262 def predict_image(img, model):
263     # Convert to a batch of 1
264     xb = to_device(img.unsqueeze(0), device)
265     # Get predictions from model
266     yb = model(xb)
267     # Pick index with highest probability
268     _, preds  = torch.max(yb, dim=1)
269     value = torch.max(yb, dim=1)
270
271 # Если коэффициент предсказания меньше 3, то считаем, что мы
    не угадали и даем класс "Unknown"
272     if value[0].item() > 0:
273       # Retrieve the class label
274       return dataset.classes[preds[0].item()]
275     else:
276       return "Unknown"
277
278 # Предсказываем картинку
```

```
279 img, label = valid_ds[10]
280 plt.imshow(img.permute(1, 2, 0).clamp(0, 1))
281 print('Label:', dataset.classes[label], ', Predicted:',
        predict_image(img, model))
282
283 # Загрузка картинки из папки и показ
284 path = './Pictures/Image04.jpg'
285 with open(path, 'rb') as f:
286     img = Image.open(f)
287     img.convert('RGB')
288 plt.imshow(img)
289
290 # Перегоняем в тензор с нормализацией и трансформацией
291 img = transoforms_pred(img)
292
293 # Предсказываем картинку
294
295 plt.imshow((img.permute(1, 2, 0).clamp(0, 1)))
296 print('Predicted:', predict_image(img, model))
297
298 # https://stackoverflow.com/questions/42703500/best-way-to-
    save-a-trained-model-in-
    pytorch?newreg=e978ffd881af4cb6833f6ae071bebc63
299
300 # Case # 1: Save the model to use it yourself for inference:
301 model_dir = './Model/Case_1/model.pth'
302 torch.save(model.state_dict(), model_dir)
303
304 # Later to restore:
305 # model.load_state_dict(torch.load(filepath))
306 # model.eval()
307
308 # Case # 2: Save model to resume training later
309 model_dir = './Model/Case_2/model.pth'
310 state = {
311     'state_dict': model.state_dict()
312     }
313 torch.save(state, model_dir)
314
315 # Case # 3: Model to be used by someone else with no access
    to your code
316 model_dir = './Model/Case_3/model.pth'
317 torch.save(model, model_dir)
```