

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

КУРСОВОЙ ПРОЕКТ

По курсу «Разработка веб-приложений»

ТЕМА

«Сервис планировщик полетных заданий для БПЛА»

Выполнил: Анциферов Сергей Максимович

Группа: 231-3213

Проверил: Кружалов Алексей Сергеевич

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

УТВЕРЖДАЮ
заведующая кафедрой
«Инфокогнитивные технологии»
_____ / Е. А. Пухова /
«__» _____ 2025 г.

ЗАДАНИЕ

на выполнение курсовой работы (проекта)

Анциферову Сергею Максимовичу

обучающемуся группы 231-3213,

направления подготовки 09.03.01 «Информатика и вычислительная техника»

по дисциплине «Разработка веб-приложений»

на тему «Сервис планировщик полетных заданий для БПЛА»

1. Исходные данные к работе (проекту): информационные ресурсы в сети интернет, научные публикации в открытой печати.

2. Содержание задания по курсовой работе (проекту) – перечень вопросов, подлежащих разработке:

Разрабатываемый вопрос	Объем от всего задания, %	Срок выполнения	Примечание
Раздел 1. Анализ предметной области		28.03.2025	
Задача 1.1. Обзор существующих программных продуктов по теме работы		28.03.2025	
Задача 1.2. Анализ программных инструментов разработки веб-приложений		28.03.2025	
Задача 1.3. Формулировка цели и задач работы		28.03.2025	
Раздел 2. Проектирование веб-приложения		18.04.2025	
Задача 2.1. Анализ целевой аудитории		18.04.2025	
Задача 2.2. Описание функциональности приложения (диаграмма вариантов использования, user story и т. д.)		18.04.2025	
Задача 2.3. Проектирование модели данных (ER-диаграмма, логическая и физическая схемы БД)		18.04.2025	
Задача 2.4. Разработка макетов страниц (Wireframe)		18.04.2025	
Раздел 3. Разработка веб-приложения		23.05.2025	
Задача 3.1. Разработка базовой структуры приложения и вёрстка шаблонов страниц		23.05.2025	
Задача 3.2. Реализация аутентификации пользователей		23.05.2025	
Задача 3.3. Реализация CRUD-интерфейса		23.05.2025	
Задача 3.4. Реализация карты с отметкой ключевых точек для полета		23.05.2025	
Задача 3.5. Реализация функционала администратора		23.05.2025	
Задача 3.6. Реализация генерации документов о регистрации дрона		23.05.2025	
Задача 3.7. Реализация генерации разрешительных документов на полет		23.05.2025	
Раздел 4. Оформление итогов работы		30.05.2025	
Задача 4.1. Создание Git-репозитория с кодом проекта		30.05.2025	
Задача 4.2. Деплой приложения на хостинг		30.05.2025	
Задача 4.3. Оформление отчёта о проделанной работе		30.05.2025	

Руководитель курсовой работы (проекта): старший преподаватель кафедры «Инфокогнитивные технологии»

«__» _____ 2025 г.

А. С. Кружалов

Дата выдачи задания

«__» _____ 2025 г.

Дата сдачи выполненной работы (проекта)

«__» _____ 2025 г.

Задание принял к исполнению

«__» _____ 2025 г.

(подпись)

(И. О. Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
Анализ предметной области.....	6
Обзор существующих программных продуктов по теме работы	6
Анализ программных инструментов разработки веб-приложений	8
Формулировка целей и задач работы	12
Проектирование веб-приложения.....	15
Анализ целевой аудитории.....	15
Проектирование модели данных (ER-диаграмма, логическая и физическая схемы БД).....	18
Разработка макетов страниц (Wireframe)	21
Разработка веб-приложения	28
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	43

ВВЕДЕНИЕ

В современном мире беспилотные летательные аппараты (БПЛА) нашли широкое применение в различных сферах деятельности, включая мониторинг территорий, аэрофотосъемку, доставку грузов и проведение поисково-спасательных операций. Эффективное управление парком БПЛА требует программного обеспечения, способного координировать выполнение множества полетных заданий с учетом различных ограничений и требований безопасности.

Разрабатываемый сервис предоставляет централизованное наблюдение за работой БПЛА: регистрация оборудования и формирования полетных заданий.

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Обзор существующих программных продуктов по теме работы

Обзор существующих программных продуктов по теме работы, выявление функциональных и архитектурных особенностей современных приложений для регистрации и планирования полетов беспилотных летательных аппаратов.

Из аналогов выберем два наиболее популярных решения: портал “Госуслуги” и приложение AirMap.

Критерии анализа

- Доступность приложения
- Наличие интерактивной карты для планирования маршрута
- Выдача документов о разрешении полета
- Регистрация БПЛА в системе

Анализ аналогов

Доступность приложения.

Госуслуги: Приложение доступно бесплатно, интегрировано в государственный портал. Требуется обязательная регистрация через учетную запись на «Госуслугах». Интерфейс на русском языке, но отсутствует мобильная версия для оперативного планирования полетов. Недостаток: Нет офлайн-доступа, требуется стабильное интернет-соединение.

AirMap: Доступен как веб-сервис и мобильное приложение (iOS/Android). Базовая версия бесплатна, но ключевые функции: автоматическая проверка маршрута, интеграция с регуляторами доступны по подписке от \$10/мес. Недостаток: Платный функционал ограничивает использование для гражданских лиц с низким бюджетом. Имеет ограничения по региону использования.

Наличие интерактивной карты для планирования маршрута.

Госуслуги: Интерактивная карта отсутствует. Пользователь указывает зону полета текстовым описанием или выбирает из списка запрещенных зон.

Недостаток: Риск ошибок из-за ручного ввода координат. Например, полет над частной территорией может быть не замечен при текстовом описании.

AirMap: Есть детализированная карта с слоями: зоны ограничений (аэропорты, военные объекты), погодные условия, высотные ограничения. Маршрут строится визуально, с автоматической проверкой на конфликты. Недостаток: Карта не учитывает актуальные российские ограничения, что снижает её полезность в РФ.

Выдача документов о разрешении полета.

Госуслуги: После регистрации БПЛА и подачи уведомления система генерирует PDF-документ с номером разрешения. Документ необходимо распечатать или сохранить в электронном виде для предъявления по требованию.

AirMap: Для США/ЕС платформа автоматически отправляет уведомление в FAA или местный регулятор. Пользователь получает цифровое разрешение с QR-кодом, который привязан к данным полета.

Регистрация БПЛА в системе.

Госуслуги: Регистрация БПЛА обязательна для дронов тяжелее 150 г. Пользователь вносит данные вручную: серийный номер, модель, вес, фото аппарата. Данные сохраняются в госреестре.

AirMap: Регистрация БПЛА добровольна (кроме случаев, требуемых локальным законодательством). Можно добавить несколько дронов в профиль, указав базовые параметры (модель, класс).

Таблица 1 – Таблица сравнений аналогов

Критерий	Госуслуги	AirMap
Доступность	Бесплатно	Платные функции, ограничения по региону
Интерактивная карта	Нет	Да
Документы о разрешении	PDF-документ	Цифровое разрешение с QR-кодом
Регистрация БПЛА	Обязательная	Добровольная

Анализ программных инструментов разработки веб-приложений

Таблица 2 – Сравнение Backend фреймворков.

Критерий	Flask	FastAPI	Django
Скорость разработки	Минималистичный, гибкий, но требует ручной настройки многих компонентов. Подходит для небольших проектов.	Автоматическая генерация документации (Swagger/OpenAPI), валидация данных через Pydantic, асинхронность. Минимальный boilerplate-код.	Монолитный, включает множество встроенных инструментов (ORM, админка), но требует времени на изучение и настройку.
Производительность	Синхронный, подходит для простых задач. Низкая производительность при высоких нагрузках.	Асинхронная обработка запросов, высокая скорость благодаря Starlette и ASGI.	Синхронный, оптимизирован для типовых задач, но уступает FastAPI в скорости обработки API.
Безопасность	Базовые функции безопасности, требуется ручная реализация многих	Встроенная валидация и сериализация данных через	Встроенные защиты (CSRF, XSS), но для API

	механизмов (например, валидация данных).	Pydantic, защита от инъекций, поддержка OAuth2.	требуется Django REST Framework, что усложняет настройку.
Современность	Устаревшая архитектура для API (синхронная, нет встроенной поддержки асинхронности).	Современный подход (ASGI, поддержка WebSockets, интеграция с GraphQL), активно развивается.	Ориентирован на классические MVC-приложения, менее гибок для микросервисов и современных API.
Поддержка MVP	Подходит для MVP, но требует дополнительных расширений (Flask-SQLAlchemy, Flask-JWT).	Идеален для MVP: быстрая реализация эндпоинтов, автоматическая документация, минимум зависимостей.	Избыточность для MVP: встроенные компоненты (админка, ORM) замедляют старт.

Итог: FastAPI — оптимальный выбор для MVP благодаря скорости разработки, асинхронности, встроенной безопасности и автоматической документации. Это сокращает время на интеграцию и тестирование.

Таблица 3 – Сравнение баз данных

Критерий	SQLite	MySQL	PostgreSQL
Масштабируемость	Для однопользовательских приложений, не поддерживает параллельные запросы.	Поддерживает высокие нагрузки, но ограничен в сложных транзакциях.	Оптимален для высоких нагрузок, поддержка параллелизма через MVCC.
Типы данных	Ограниченный набор типов (нет массивов, JSON).	Поддержка JSON, но с ограниченными функциями.	Расширенные типы (JSONB, геоданные, массивы), полноценная работа с JSON.
Безопасность	Минимальные встроенные механизмы (например, нет ролевой модели).	Базовые роли и привилегии.	Гибкая система прав, SSL-шифрование, защита от SQL-инъекций.
Расширяемость	Невозможно добавлять пользовательские функции и расширения.	Ограниченные возможности кастомизации.	Поддержка расширений (PostGIS, Full-Text Search).

Итог: PostgreSQL обеспечивает масштабируемость, безопасность и гибкость, а также поддерживает работу с JSON, что критично в нашем проекте.

Таблица 4 – Сравнение фронтенд фреймворков

Критерий	HTML + JS + Bootstrap	React/Angular
Скорость разработки	Мгновенный старт: нет необходимости	Требуется время на настройку окружения,

	настраивать сборщики (Webpack, Babel). Готовые компоненты Bootstrap.	изучение синтаксиса (JSX, TypeScript).
Производительность	Минимум накладных расходов: нет виртуального DOM или runtime-библиотек.	Виртуальный DOM (React) и проверки изменений могут замедлять рендеринг.
Гибкость	Полный контроль над кодом, можно интегрировать любые библиотеки.	Жесткая архитектура (особенно Angular), зависимость от lifecycle-методов.
Поддержка MVP	Идеально для прототипов: быстрое создание адаптивных интерфейсов через Bootstrap.	Избыточно для простых проектов: сложность стейт-менеджмента (Redux, NgRx).

Итог: HTML + JS + Bootstrap позволяет быстро создать MVP с минимальными затратами, избегая перегруженности инструментами.

Для разработки будут выбраны следующие средства FastAPI, PostgreSQL, HTML, JS, Bootstrap. Такой выбор позволит эффективно и быстро разрабатывать приложение, используя уже существующие шаблоны и подходы в разработке и архитектуре.

Формулировка целей и задач работы

Цели проекта

Разработать веб-приложение, предоставляющее пользователям инструменты для:

Регистрации и управления дронами с возможностью загрузки технических характеристик и фотографий
Планирования маршрутов полетов с выбором точек и генерацией документации
Получения сертификатов на зарегистрированные дроны
Администрирования моделей дронов через ролевую модель (пользователь/администратор)

Задачи проекта

1. Реализация системы аутентификации
 - a. Регистрация новых пользователей с валидацией данных
 - b. Авторизация пользователей через логин/пароль
 - c. Шифрование паролей и обеспечение безопасности сессий
 - d. Разграничение прав доступа (пользователь/администратор)
2. CRUD-интерфейс для управления дронами
 - a. Регистрация дронов:
 - b. Заполнение технических характеристик (модель, серийный номер, вес, габариты)
 - c. Загрузка и обработка фотографий дронов
 - d. Привязка дрона к пользователю
 - e. Управление дронами:
 - f. Просмотр списка зарегистрированных дронов
 - g. Редактирование информации о дронах
 - h. Удаление дронов из системы (мягкое удаление)
3. Система планирования маршрутов
 - a. Интерактивная карта:
 - b. Выбор точек маршрута на карте
 - c. Визуализация маршрута для пользователя
 - d. Генерация документации:
 - e. Создание карты маршрута с координатами

- f. Планирование дат и времени полетов
- g. Экспорт документов в PDF формате
- 4. Система сертификации
 - a. Автоматическая генерация сертификатов после регистрации дрона
- 5. Админ-панель
 - a. Управление моделями дронов:
 - b. Добавление новых моделей в справочник
- 6. Деплой и тестирование проекта
 - a. Деплой проекта на сервер
 - b. Проведение функционального тестирования

Ожидаемые результаты

1. Функциональное приложение
 1. Пользователи могут:
 - a. Регистрировать свои дроны с загрузкой технических характеристик и фотографий
 - b. Планировать маршруты полетов с интерактивной картой
 - c. Получать автоматически сгенерированные сертификаты на зарегистрированные дроны
 - d. Экспортировать данные о дронах и маршрутах
 2. Администраторы имеют возможность:
 - a. Управлять справочником моделей дронов (добавление, редактирование, удаление)
 - b. Контролировать процесс регистрации дронов пользователями
3. Документация
4. Техническое описание:
 - a. Архитектура системы с диаграммами компонентов
 - b. Схема базы данных и модель данных
5. Пользовательская документация:
 - a. Руководство по регистрации дронов
 - b. Инструкция по планированию маршрутов
6. Административная документация:
 - a. Руководство по управлению моделями дронов
2. Исходный код
7. Git-репозиторий:
 - a. Структурированная организация кода по модулям
8. CI/CD-пайплайн:
 - a. Автоматическое тестирование при каждом коммите

б. Интеграция с системами контроля качества кода

Связь целей с анализом целевой аудитории

Выявленные проблемы рынка:

- Отсутствие интерактивных карт: В российском сегменте текущее решение не позволяет использовать интерактивные карты, что усложняет процесс создания полетного маршрута.

Решения в проекте:

- Реализация интерактивной карты
- Высокая совместимость по функционалу с текущими российскими решениями

ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

Анализ целевой аудитории

Сегментация аудитории

Целевую аудиторию можно разделить на две ключевые группы любители и коммерческие пользователи.

Таблица 5 – Целевая аудитория.

Группа	Описание	Примеры использования
Любители	Физические лица, использующие дроны для хобби: фото/видеосъемка, развлечения	Съемка природы, семейных мероприятий, путешествий
Коммерческие пользователи	Малый бизнес и фрилансеры, применяющие БПЛА для профессиональных задач	Доставка мелких грузов, агромониторинг, инспекция объектов

Демографические характеристики:

Возраст: 18–55 лет основная активность пользователей будет приходиться на период 25 - 44 лет.

География пользователей

- Городские жители (70%) — высокая плотность запрещенных зон таких как аэропорты, инфраструктура.
- Сельские жители (30%) — потребность в мониторинге территорий, сельхозработ.

Таблица 6 – Потребности и проблемы целевых групп

Группа	Потребности	Проблемы
Любители	Простое планирование маршрутов Легализация полетов без бюрократии	Страх нарушить закон Сложности с ручным вводом координат
Коммерческие	Быстрое согласование	Потери времени из-за

пользователи	полетов с регуляторами Интеграция с госреестрами Автоматизация отчетности	ручной регистрации• Риск штрафов за нарушения
--------------	---	---

Ключевые ожидания от приложения

Упрощение регистрации БПЛА

- Автозаполнение данных дрона по серийному номеру (сканирование QR-кода)
- Интеграция с государственными реестрами (например, Росавиация)

Интерактивное планирование

- Карта с подсветкой запрещенных зон в реальном времени
- Возможность сохранять шаблоны маршрутов

Легализация полетов

- Автоматическая генерация разрешительных документов
- Уведомления об изменениях в законодательстве

Выводы и рекомендации

Приоритетные функции для разработки

- Интуитивный интерфейс для всех категорий пользователей
- Автоматизация процессов для снижения барьеров входа
- Интеграция с регуляторными органами для упрощения легализации

Критические требования

- Простота использования для новичков
- Скорость обработки для коммерческих пользователей
- Визуальная привлекательность для любителей

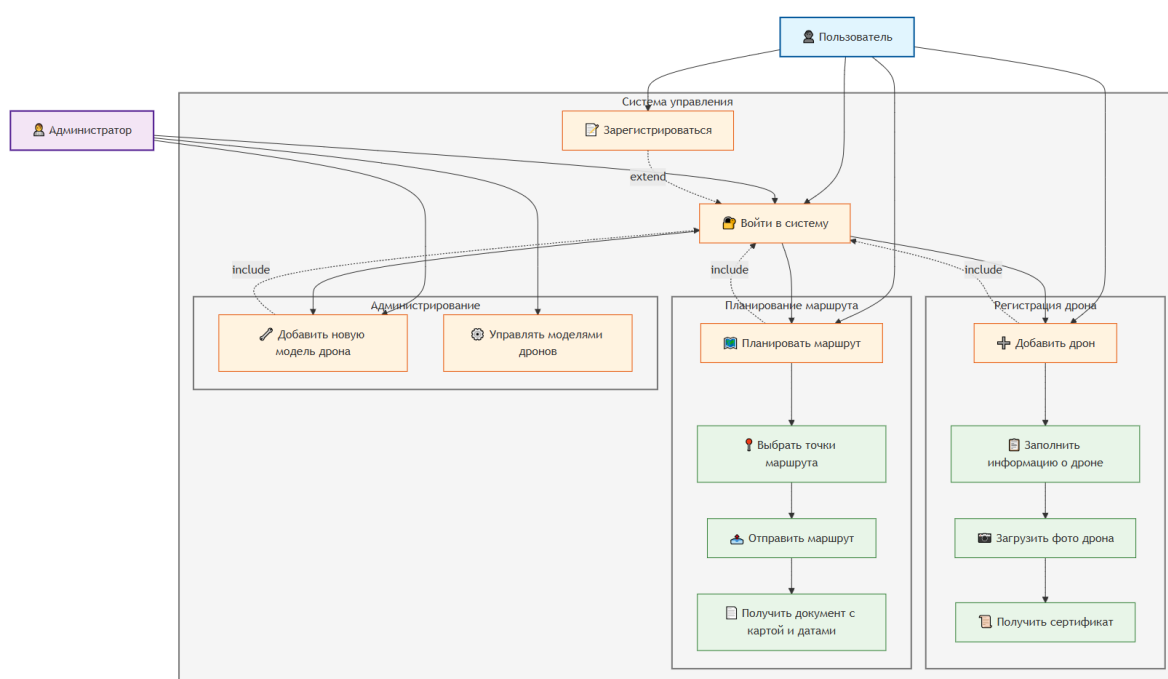
Описание функциональности приложения

Гость – Проводит регистрацию

Пилот дрона - просматривает список доступных дронов, добавляет новые дроны, создает маршруты.

Администратор - создает модели дронов доступные для регистрации, просматривает полетные маршруты пользователей.

Рисунок 1 – Диаграмма UseCase.



Проектирование модели данных (ER-диаграмма, логическая и физическая схемы БД).

Таблица 7 - Таблица модели данных Role.

Атрибут	Тип данных	Описание
id	UUID	Первичный ключ
title	VARCHAR(50)	Название роли (NOT NULL)

Таблица 8 - Таблица модели данных User.

Атрибут	Тип данных	Описание
id	UUID	Первичный ключ
first_name	VARCHAR(255)	Имя (NOT NULL)
middle_name	VARCHAR(255)	Отчество
last_name	VARCHAR(255)	Фамилия (NOT NULL)
login	VARCHAR(255)	Уникальный логин (NOT NULL, UNIQUE)
password	VARCHAR(255)	Пароль (NOT NULL)
birth_day	DATE	Дата рождения (NOT NULL)
role_id	UUID	Внешний ключ к role.id
id_card_series	INT	Серия паспорта
id_card_number	INT	Номер паспорта
created_at	TIMESTAMP	Дата создания (DEFAULT: NOW())

Таблица 9 - Таблица модели данных Model.

Атрибут	Тип данных	Описание
id	UUID	Первичный ключ
title	VARCHAR(255)	Название модели (NOT NULL)
factory	VARCHAR(255)	Производитель
description	TEXT	Описание модели
weight	FLOAT	Вес (CHECK: weight >

		0)
max_range	FLOAT	Максимальная дальность (CHECK: max_range > 0)

Таблица 10 - Таблица модели данных File.

Атрибут	Тип данных	Описание
id	UUID	Первичный ключ
title	VARCHAR(255)	Название файла
base64_data	TEXT	Данные в Base64
mime_type	VARCHAR(50)	MIME-тип файла
created_at	TIMESTAMP	Дата создания (DEFAULT: NOW())
Атрибут	Тип данных	Описание

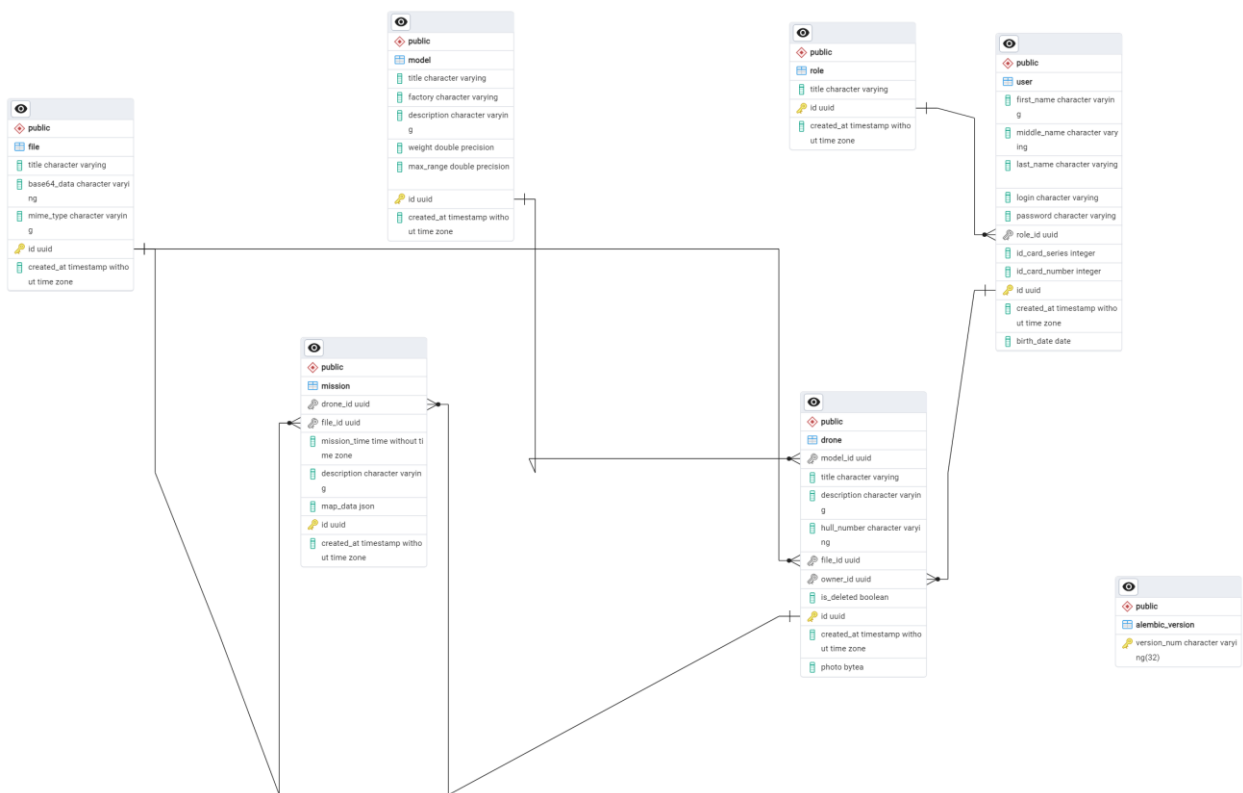
Таблица 11 - Таблица модели данных Drone.

Атрибут	Тип данных	Описание
id	UUID	Первичный ключ
model_id	UUID	Внешний ключ к model.id
title	VARCHAR(255)	Название дрона (NOT NULL)
description	TEXT	Описание дрона
hull_number	VARCHAR(100)	Серийный номер (NOT NULL, UNIQUE)
photo	Bytes	Фото дрона
file_id	UUID	Внешний ключ к file.id
owner_id	UUID	Внешний ключ к user.id
created_at	TIMESTAMP	Дата создания (DEFAULT: NOW())
delete	Bool	Мягкое удаление (DEFAULT: False)

Таблица 12 - Таблица модели данных Mission.

id	UUID	Первичный ключ
drone_id	UUID	Внешний ключ к drone.id
file_id	UUID	Внешний ключ к file.id
mission_time	TIMESTAMP	Время миссии (NOT NULL)
created_at	TIMESTAMP	Дата создания (DEFAULT: NOW())
description	TEXT	Описание миссии
map_data	JSONB	Данные карты в формате JSON

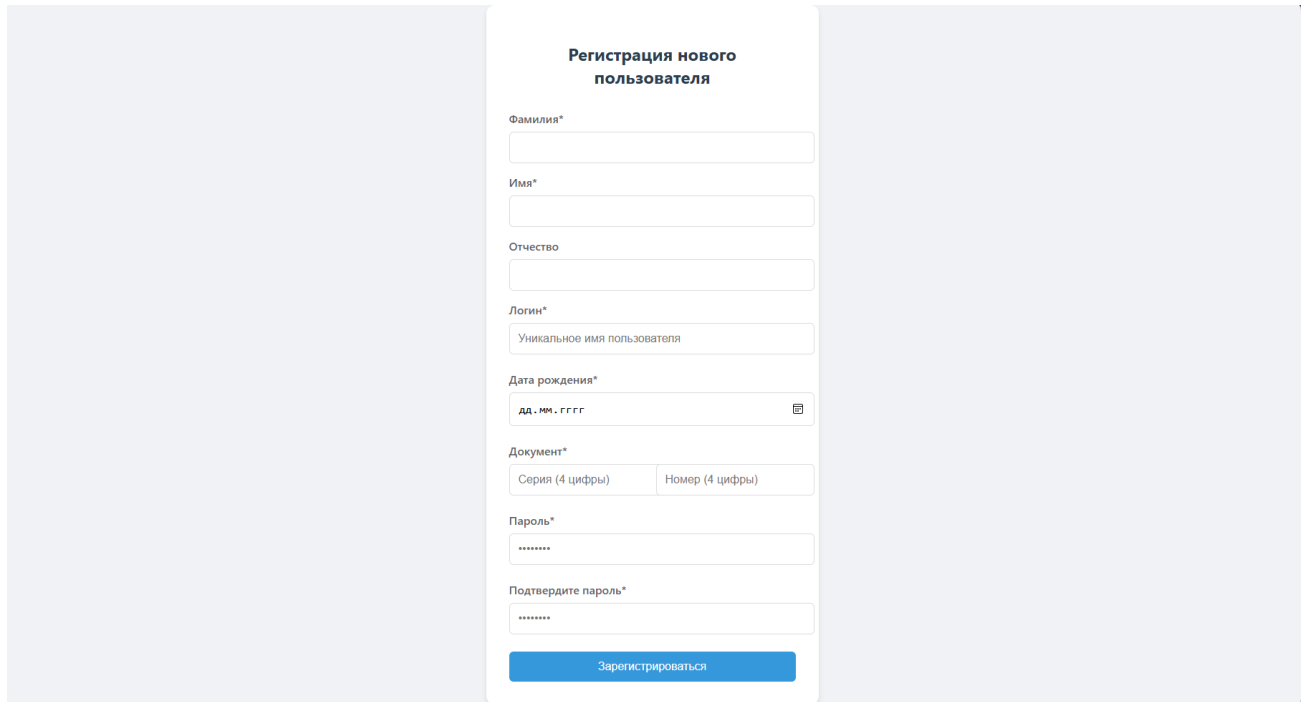
Рисунок 2 – ER диаграмма.



Разработка макетов страниц (Wireframe)

Макеты пользователя

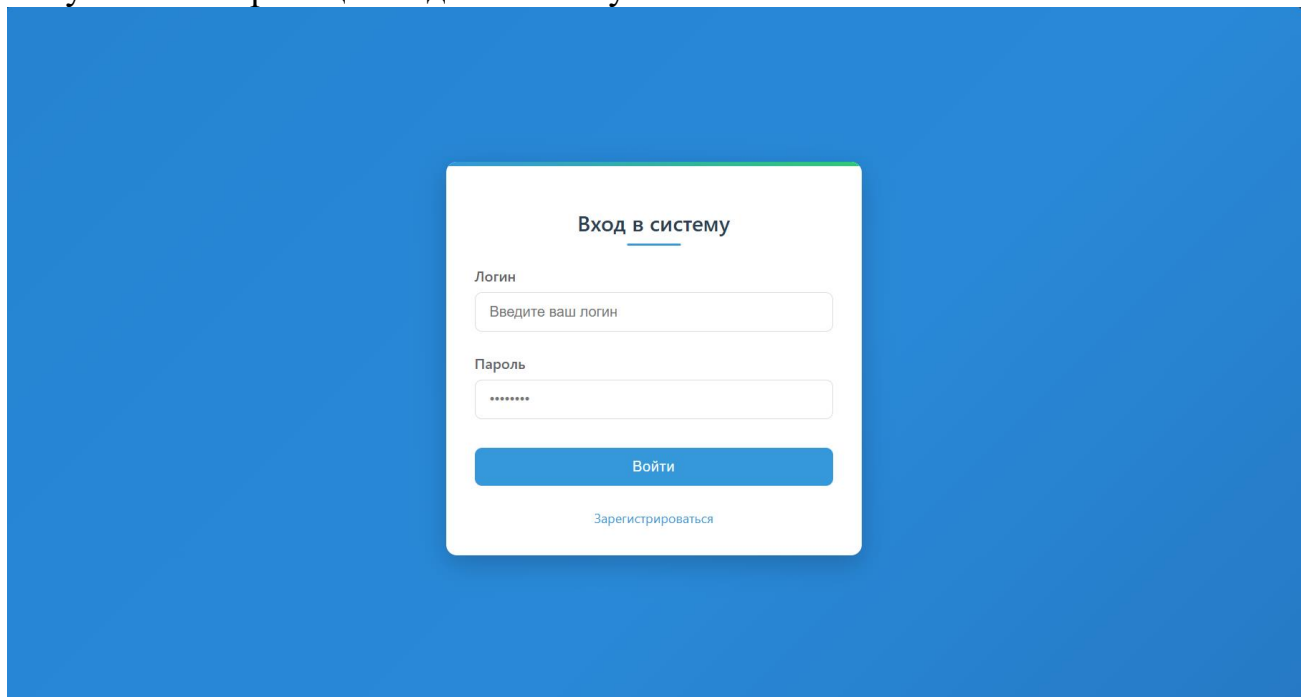
Рисунок 3 - Страница регистрации.



The wireframe shows a registration form titled "Регистрация нового пользователя" (Registration of a new user). The form is centered on a light gray background. It includes the following fields and elements:

- Фамилия*** (Surname): A text input field.
- Имя*** (Name): A text input field.
- Отчество** (Patronymic): A text input field.
- Логин*** (Login): A text input field with the placeholder "Уникальное имя пользователя" (Unique username).
- Дата рождения*** (Date of birth): A date picker field showing "ДД.ММ.ГГГГ" (DD.MM.YYYY).
- Документ*** (Document): Two adjacent text input fields labeled "Серия (4 цифры)" (Series (4 digits)) and "Номер (4 цифры)" (Number (4 digits)).
- Пароль*** (Password): A password input field with masked characters.
- Подтвердите пароль*** (Confirm password): A second password input field with masked characters.
- Зарегистрироваться** (Register): A blue button at the bottom of the form.

Рисунок 4 - Страница входа в систему.



The wireframe shows a login form titled "Вход в систему" (Login to the system). The form is centered on a solid blue background. It includes the following fields and elements:

- Логин** (Login): A text input field with the placeholder "Введите ваш логин" (Enter your login).
- Пароль** (Password): A password input field with masked characters.
- Войти** (Login): A blue button.
- Зарегистрироваться** (Register): A link below the login button.

Рисунок 5 - Главное меню пользователя.

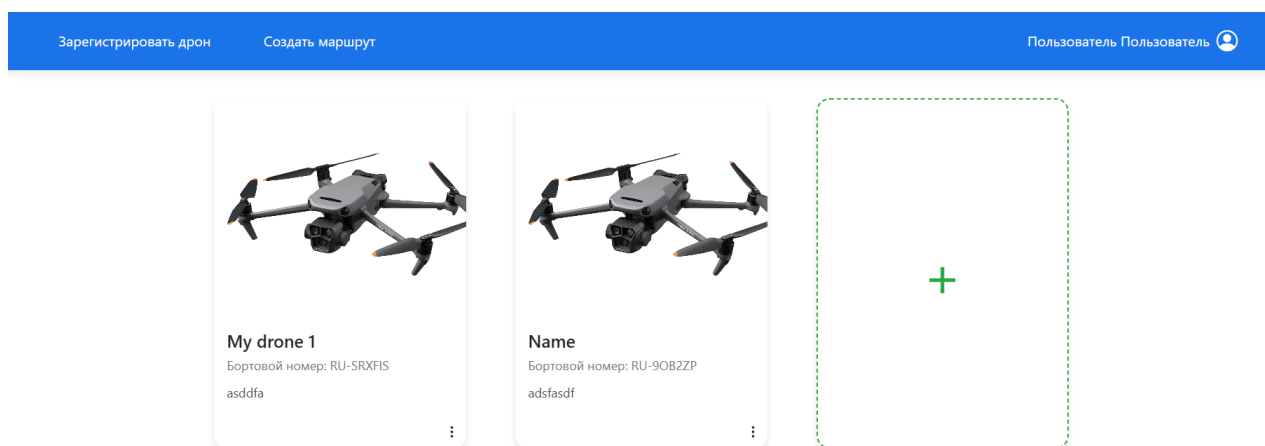


Рисунок 6 - Регистрация БПЛА.

The screenshot shows a registration form titled "Регистрация нового БПЛА" (Registration of a new UAV). The form is contained within a light gray box. At the top left of the box is a "Домой" (Home) link. At the top right is a "Пользователь Пользователь" (User User) link with a profile icon. The form fields are: "Фото дрона:" (Drone photo) with a dashed box and a camera icon; "Модель дрона:" (Drone model) with a dropdown menu labeled "Выберите модель"; "Именованье дрона:" (Drone name) with a text input field labeled "Введите уникальное имя"; and "Описание:" (Description) with a text area labeled "Опишите особенности вашего дрона (модификации, установленное оборудование)". A character count "0/300" is visible at the bottom right of the text area. At the bottom of the form is a blue button labeled "Зарегистрировать" (Register).

Рисунок 7 - Редактирование БПЛА.

Домой

Пользователь Пользователь

Редактирование БПЛА

Фото дрона:

Нажмите для изменения фото

Текущее фото загружено

Модель дрона:

Выберите модель

Именование дрона:

Введите уникальное имя

Описание:

Опишите особенности вашего дрона (модификации, установленное оборудование)

0/500

Обновить данные дрона

Рисунок 8 - Сертификат на БПЛА.



Рисунок 9 - Создание маршрута.

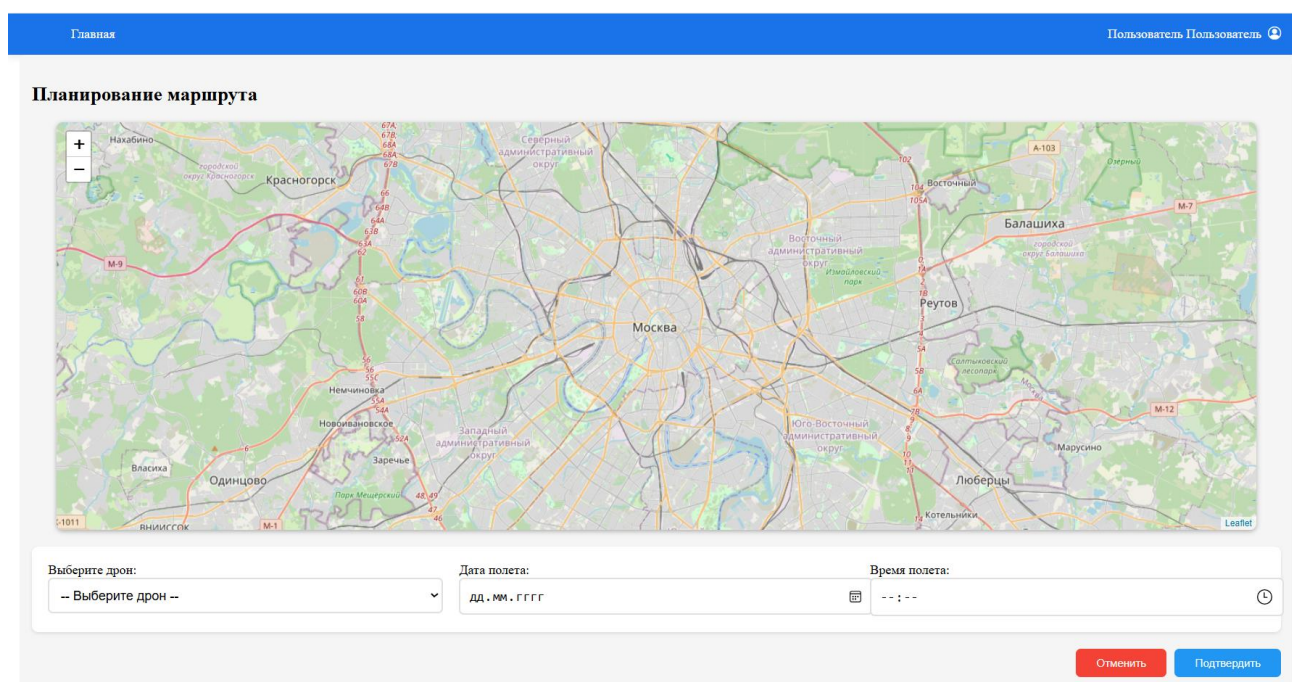
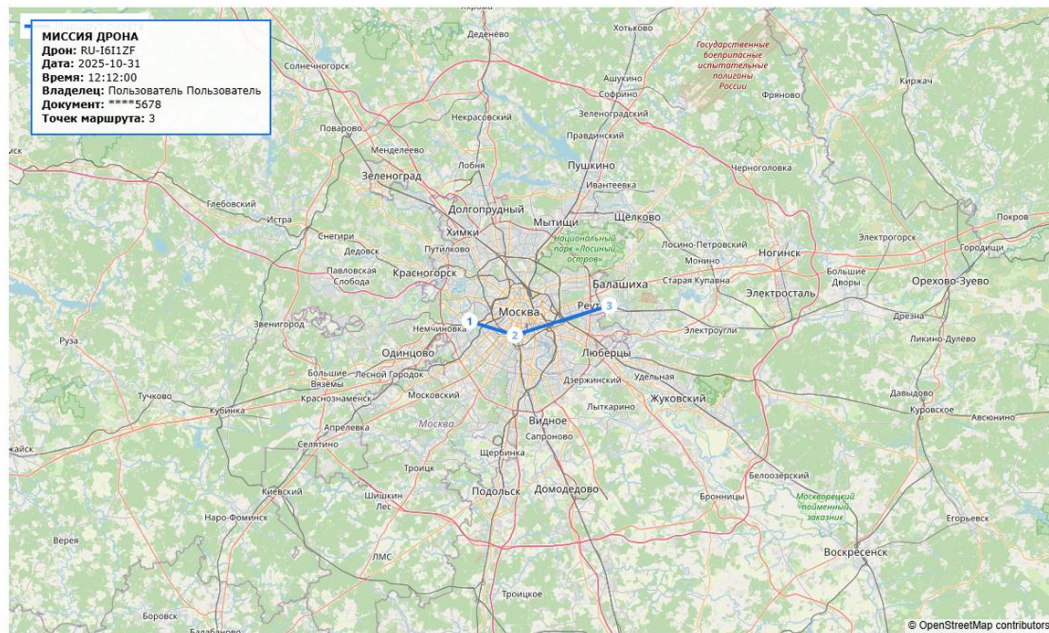


Рисунок 10 - Разрешение на полет.

Карта миссии дрона



Макеты администратора

Рисунок 11 - Главное меню администратора.

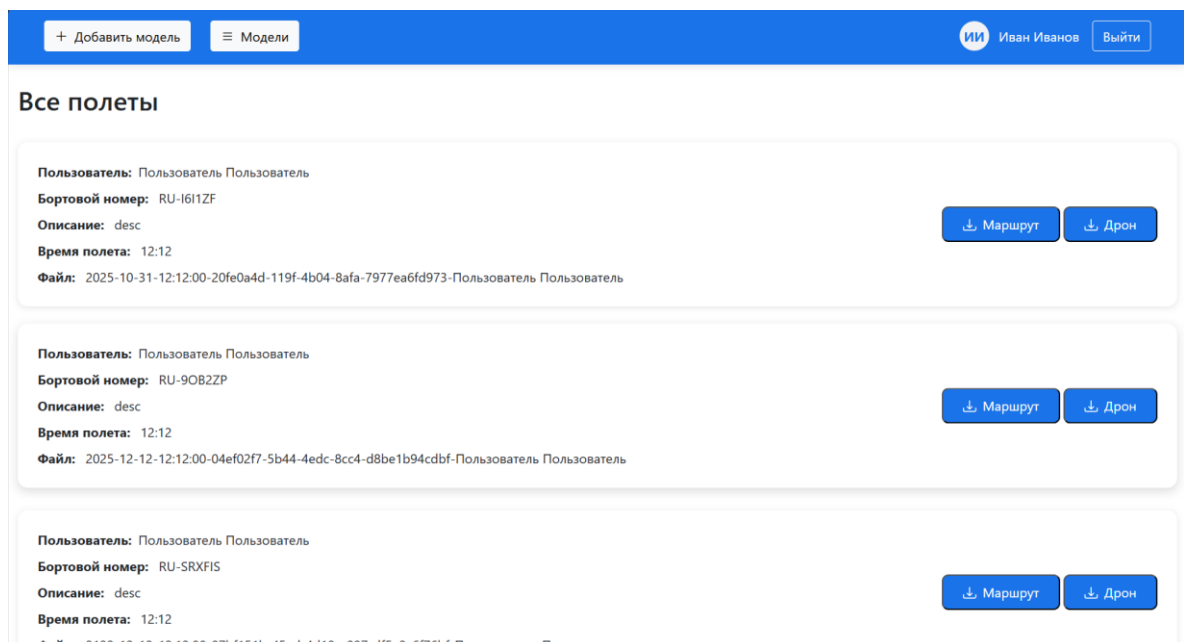


Рисунок 12 - Просмотр всех доступных моделей.

+

Добавить модель

☰Полеты

ИИИван Иванов

Выйти

Все модели

DJI MAVIC 3

Вес:

2.1 кг

Максимальная дальность:

10 км

Описание:

Особенности

Неустановленная

Вес:

0.5 кг

Максимальная дальность:

1 км

Описание:

Описание

Рисунок 13 - Создание модели.

←Назад

ИИИван Иванов

Выйти

Регистрация новой модели дрона

Название модели *

DJI Mavic 3 Pro

Производитель *

DJI

Вес (кг) *

1.2

Макс. дальность (км) *

25

Описание

Технические характеристики и особенности...

0/500

Зарегистрировать модель

Рисунок 14 - Просмотр маршрутов полетов.

Карта миссии дрона

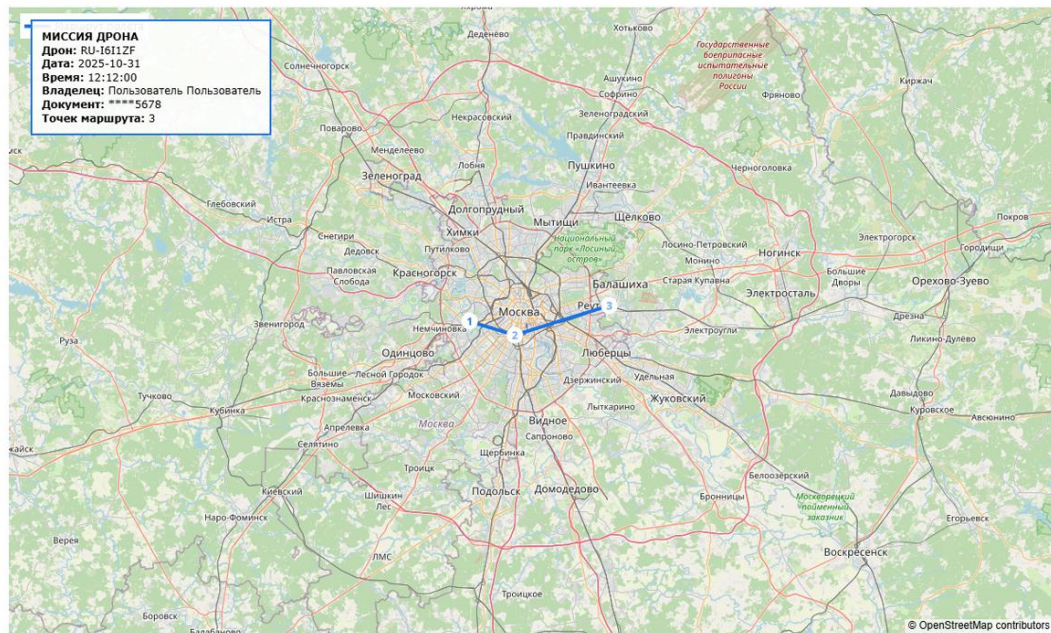


Рисунок 15 - Просмотр регистрации БПЛА.

СЕРТИФИКАТ РЕГИСТРАЦИИ БПЛА

Удостоверение регистрации беспилотного летательного аппарата



Бортовой номер:	RU-I611ZF
Название:	Дрон 3
Модель:	DJI MAVIC 3
Владелец:	Пользователь Пользователь Пользович
Дата выдачи:	31.10.2025

Данный сертификат подтверждает регистрацию беспилотного летательного аппарата
 в системе учета и управления дронами

Разработка веб-приложения

В ходе работы был создан сервис с использованием Python, FastApi, PostgreSQL.

Сервис в своей структуре реализует принципы чистой архитектуры, обеспечивая разграничение фреймворка, бизнес логики, слоя работы с данными, доменного слоя, слоя граничных моделей. Для уменьшения связанности компонентов между собой применяется инверсия зависимостей.

```
└─ src/  
    └─ adapter/  
    └─ domain/  
    └─ repository/  
    └─ router/
```

- |— *dependencies/*
- |— *schemas/*
- |— *service/*

На уровне фреймворка используется FastApi, это современный фреймворк для разработки бэкенд приложений. Его основными преимуществами является глубокая интеграция с граничными моделями на основе библиотеки Pydantic, а также поддержка инверсии зависимостей, инъекции зависимостей без установки сторонних библиотек. Для разграничения путей запросов применяются роутеры, которые позволяют настраивать гибкую маршрутизацию запросов.

Пример инициализации роутера, подключение зависимостей и их инъекция в обработчик запросов.

```
drones_router = APIRouter(prefix="/drones")
```

```
@drones_router.get("/")
```

```
async def get_all_drones(
```

```
    user: Annotated[CurrentUser, Depends(get_auth())],
```

```
    uow: UnitOfWork = Depends(get_uow),
```

```
) -> list[ResponseDrone]:
```

```
    return await drones.get_all_drones(user, uow=uow)
```

Роутеры подключаются на основное приложение FastApi.

```
app = FastAPI()
```

```
app.include_router(drones_router)
```

Доменный слой представлен классами отображающими модель данных каждой представленной сущности. Все классы наследуются от базового класса с универсальными полями для всех классов. Базовый класс в свою очередь наследуется от класса SQLAlchemy.

```
from sqlalchemy.orm import DeclarativeBase
```

```
class Base(DeclarativeBase):
```

```
    id: Mapped[UUID] = mapped_column(default=uuid.uuid4,  
primary_key=True)
```

```
    created_at: Mapped[datetime.datetime] =  
mapped_column(default=datetime.datetime.utcnow)
```

```
    def to_dict(self) -> dict:
```

```
        return {  
            c.name: str(getattr(self, c.name)) for c in  
self.__table__.columns  
        }
```

```
class Role(Base):
```

```
    __tablename__ = 'role'
```

```
    title: Mapped[str]
```

```
    users: Mapped[list['User']] =  
relationship(back_populates="role")
```

```

class User(Base):
    __tablename__ = 'user'

    first_name: Mapped[str]
    middle_name: Mapped[str]
    last_name: Mapped[str]
    login: Mapped[str]
    password: Mapped[str]
    birth_date: Mapped[datetime.date]
    role_id: Mapped[UUID] = mapped_column(ForeignKey('role.id'))
    role: Mapped['Role'] = relationship(back_populates='users')
    id_card_series: Mapped[int]
    id_card_number: Mapped[int]

    drones: Mapped[list['Drone']] =
relationship(back_populates='owner')

```

```

class Model(Base):
    __tablename__ = 'model'

    title: Mapped[str]
    factory: Mapped[str]

```

description: Mapped[str]

weight: Mapped[float]

max_range: Mapped[float]

drones: Mapped[list['Drone']] =
relationship(back_populates='model')

class File(Base):

__tablename__ = 'file'

title: Mapped[str]

base64_data: Mapped[str]

mime_type: Mapped[str]

drones: Mapped[list['Drone']] =
relationship(back_populates='file')

missions: Mapped[list['Mission']] =
relationship(back_populates='file')

class Drone(Base):

__tablename__ = 'drone'

model_id: Mapped[UUID] =
mapped_column(ForeignKey('model.id'))

model: Mapped['Model'] =
relationship(back_populates='drones')


```

    title: Mapped[str]

    photo: Mapped[bytes]

    description: Mapped[str | None]

    hull_number: Mapped[str]

    file_id: Mapped[UUID] = mapped_column(ForeignKey('file.id'))

    file: Mapped['File'] = relationship(back_populates='drones')


    owner_id: Mapped[UUID] = mapped_column(ForeignKey('user.id'))

    owner: Mapped['User'] = relationship(back_populates='drones')

    is_deleted: Mapped[bool]

    missions: Mapped[list['Mission']] = relationship(back_populates='drone')


class Mission(Base):

    __tablename__ = 'mission'

    drone_id: Mapped[UUID] = mapped_column(ForeignKey('drone.id'))

    drone: Mapped['Drone'] = relationship(back_populates='missions')

    file_id: Mapped[UUID] = mapped_column(ForeignKey('file.id'))

    file: Mapped['File'] = relationship(back_populates='missions')

```

```
mission_time: Mapped[datetime.time]  
description: Mapped[str]  
map_data: Mapped[dict] = mapped_column(JSON())
```

Мы рассмотрим один из наиболее важных архитектурных паттернов в данном сервисе, под названием Unit of Work или единица работы. Данный паттерн активно используется для доступа к данным, открытию и закрытию транзакций к базе данных. Совместно с ним активно используется паттерн Репозиторий, для непосредственной работы с базой данных, реализации конкретных запросов взаимодействия с ней. Для составления SQL запросов используется популярный ORM фреймворк SQLAlchemy.

Класс UnitOfWork реализует управление транзакциями: их открытие, закрытие, отмену транзакции. Данные методы называются `__aenter__`, `__aexit__`, `rollback` соответственно. Для фиксирования изменений в транзакции используется метод `commit`. В классе присутствуют репозитории являющиеся экземплярами класса `BaseRepo`, при вызове метода `commit` происходит фиксирование всех изменений во всех репозиториях которые были вызваны в рамках открытой транзакции. Это позволяет использовать одну транзакцию на фиксацию изменений в нескольких моделях принадлежащим разным репозиториям, что гарантирует атомарность бизнес операции.

Реализация класса Unit of Work.

```
class UnitOfWork:  
  
    def __init__(self, session_factory):  
  
        self._session_factory = session_factory
```

```

async def __aenter__(self):

    self._session: AsyncSession= self._session_factory()

    self._user = BaseRepo[User](session=self._session,
model=User)

    self._model = BaseRepo[Model](session=self._session,
model=Model)

    self._mission =
MissionRepo[Mission](session=self._session, model=Mission)

    self._file = BaseRepo[File](session=self._session,
model=File)

    self._drone = DroneRepo[Drone](session=self._session,
model=Drone)

    self._role = BaseRepo[Role](session=self._session,
model=Role)

    return

async def __aexit__(self, exc_type, exc_val, exc_tb):

    self._session.expunge_all()

    asyncio.shield(self._session.close())

    self._session = None

async def commit(self):

    await self._session.commit()

async def rollback(self):

```

```
await self._session.rollback()
```

```
async def refresh(self, model):
```

```
    await self._session.refresh(model)
```

```
@property
```

```
def user(self):
```

```
    return self._user
```

```
@property
```

```
def file(self):
```

```
    return self._file
```

```
@property
```

```
def drone(self):
```

```
    return self._drone
```

```
@property
```

```
def model(self):
```

```
    return self._model
```

```
@property
```

```
def mission(self):
```

```
    return self._mission
```

```

@property

def role(self):

    return self._role

def add(self, model):

    self._session.add(model)

```

Класс BaseRepo представляет собой универсальную реализацию паттерна Repository с использованием generic. Паттерн репозиторий позволяет инкапсулировать логику доступа к данным для конкретной сущности. Однако использование generic в коде позволяет создать абстрактный репозиторий, который будет работать с любой моделью данных которую в него передали. Таким образом обеспечивается пере использование кода и избегание создание классов с одинаковой функциональностью, но разным наименованием.

Основной функционал класса BaseRepo это обеспечить доступ к данным, поэтому в нем присутствуют различные методы для выбора данных, управлением политикой загрузки данных, чтобы выгружать не только запрашиваемый объект, но и связанные с ним. Метод add позволяет добавлять к транзакции новый объект, метод get реализует логику получения модели из базы данных по выбранному полю и условию. Метод get_list позволяет выбирать группы данных по заданному полю и условию.

```

class BaseRepo[T]:

    def __init__(self, session: AsyncSession, model: T):

```

```

    self._session = session

    self._model = model

def add(self, model: T) -> T:
    self._session.add(model)

    return model

async def get(
    self, field: str, value: Any,
    joined_load: list[str] | None = None
) -> T | None:
    q = select(self._model).where(getattr(self._model,
field) == value)

    if joined_load:
        q = q.options(*(joinedload(getattr(self._model,
field))

                                for field in joined_load))

    return await self._session.execute(q).scalars().first()

async def delete(self, model: T):
    return await self._session.delete(model)

async def get_or_error(
    self, field: str, value: Any,

```

```

        joined_load: list[str] | None = None) -> T:

    if res := await self.get(field=field, value=value,
                             joined_load=joined_load):

        return res

    raise Exception(f"{self._model.__name__} with field"
                    f" {field} == {value} not found")

def update(self, model: T):

    self._session.add(model)

async def get_list(self, field: str | None = None,
                   value: Any | None = None) -> list[T]:

    q = select(self._model)

    if field:

        q = q.where(getattr(self._model, field) == value)

    return (await self._session.execute(q)).scalars().all()

```

Устройство слоя работы с базой данных позволяет представлять класс как структуру хранения данных, объект класса как единичную запись в таблице. Данный функционал позволяет гибко менять отдельные поля в объекте класса обеспечивая их отображение таблице данных. Рассмотрим пример где происходит обновление сущности Drone. После получения экземпляра данной сущности мы получаем возможность заменить обновленные поля в конкретном экземпляре, а после добавить изменения в базу с помощью метода add и зарегистрировать их методом commit, оба этих метода относятся к классу UnitOfWork.

Доменная модель Drone.

```
class Drone(Base):  
  
    __tablename__ = 'drone'  
  
    model_id: Mapped[UUID] = mapped_column(ForeignKey('model.id'))  
  
    model: Mapped['Model'] = relationship(back_populates='drones')  
  
    title: Mapped[str]  
  
    photo: Mapped[bytes]  
  
    description: Mapped[str | None]  
  
    hull_number: Mapped[str]  
  
    file_id: Mapped[UUID] = mapped_column(ForeignKey('file.id'))  
  
    file: Mapped['File'] = relationship(back_populates='drones')  
  
    owner_id: Mapped[UUID] = mapped_column(ForeignKey('user.id'))  
  
    owner: Mapped['User'] = relationship(back_populates='drones')  
  
    is_deleted: Mapped[bool]  
  
    missions: Mapped[list['Mission']] = relationship(back_populates='drone')
```

Блок кода где показан пример обновления полей.


```

async def update_drone(
    uow: UnitOfWork,
    drone_id: str,
    drone: CreateDrone,
    image: UploadFile
):

    async with uow:

        drone_db = await uow.drone.get_or_error("id", drone_id)
        model = await uow.model.get("id", drone.model_id)
        user = await uow.user.get("id", drone_db.owner_id)
        file = await uow.file.get("id", drone_db.file_id)
        temp_file = None
        hull_number = drone_db.hull_number
        try:

            content = await image.read()

            with tempfile.NamedTemporaryFile(delete=False,
suffix=os.path.splitext(image.filename)[1]) as tmp:

                tmp.write(content)

                temp_file = tmp.name

        owner_name = f"{user.last_name} {user.first_name}"
        if user.middle_name:

            owner_name += f" {user.middle_name}"

```

```

        base64_pdf = generate_drone_cert_simple(
            image_path=temp_file,
            hull_number=drone_db.hull_number,
            owner=owner_name,
            drone_title=drone.title,
            model_title=model.title if model else None
        )

```

finally:

```

        if temp_file and os.path.exists(temp_file):
            os.unlink(temp_file)

```

```

file.base64_data = base64_pdf
drone_db.file = file
drone_db.title = drone.title
drone_db.description = drone.description
drone_db.model_id = drone.model_id
drone_db.photo = content

```

```

uow.add(drone_db)

```

```

await uow.commit()

```

```

return DroneCert(
    title=f"drone_certificate_{hull_number}.pdf",
    content=base64_pdf,
    mime_type="application/pdf"
)

```

)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мартин Фаулер, 2002 г , Patterns of Enterprise Application Architecture – URL: <https://martinfowler.com/books/eaa.html>.
2. Роберт Мартин, 2008 г , Clean Code: A Handbook of Agile Software Craftsmanship – URL: <https://www.oreilly.com/library/view/clean-code/9780136083238/>
3. Официальная документация Python – URL: <https://docs.python.org/3/>
4. Боб Грегори , Cosmic Python – URL: <https://www.cosmicpython.com/>
5. Pydantic: документация - URL: <https://docs.pydantic.dev/>
6. PostgreSQL: Руководство администратора
URL: <https://www.postgresql.org/docs/>
7. JavaScript: современные стандарты - URL:
<https://developer.mozilla.org/ru/docs/Web/JavaScript>
8. HTML: спецификации и руководства -
URL: <https://html.spec.whatwg.org/>
9. SQLAlchemy: официальная документация -
URL: <https://docs.sqlalchemy.org/>
10. Alembic: документация по миграциям базы данных -
URL: <https://alembic.sqlalchemy.org/>
11. FastAPI: современный веб-фреймворк – URL: <https://fastapi.tiangolo.com/>
12. Asyncio: документация по асинхронному программированию – URL:
<https://docs.python.org/3/library/asyncio.html>

13. Docker: документация по контейнеризации -
URL: <https://docs.docker.com/>
14. Git: система контроля версий - URL: <https://git-scm.com/doc>