

Описание проекта

демонстрации системы управления дроном для создания полноценного Web-сайта и публикация его на GitHub

Постановка задачи и порядок реализации проекта приложения.

****1. Обзор проекта****

Задача — разработать клиент-серверное приложение для моделирования системы управления квадрокоптером (дроном) с веб-интерфейсом, позволяющим управлять устройством и выводить данные.

Среда разработки включает Windows 11, PyCharm IDE и Python 3.10.11. Основные используемые библиотеки и API — Flask, NumPy, OpenCV, Airsim и Mavlink.

****2. Определение требований к системе управления дроном****

- А. авторизованный доступ к дрону, подключение дрона к системе управления и отключение от нее,
- Б. проверку подключения и состояния его оборудования (двигатели, батарея, камера, высотомер)
- В. запуск и остановку двигателей
- Г. выполнение полетных функций (взлет, движение, поворот, посадка)
- Д. использование сенсоров (альтиметр, GPS), фиксирования телеметрии (положение, высота) и обновление маршрута
- Е. подключение и использование камеры при выполнении полетной миссии
- Ж. выполнение различных миссий из фабрики миссий: патрулирование (демонстрация полетных функций движения по заданному маршруту, записи и графической обработки данных маршрута) и разведка (передвижение к заданной точке и задействование камеры)
- З. выбор способов управления – автоматическое (взлет и изменение высоты при взлете и посадке) и по координатам GPS (координаты и высоты точек полета задаются через веб-интерфейс и через POSTзапрос возвращаются для исполнения.
- И. Все основные операции от момента авторизации до момента посадки дрона и отключения должны логгироваться, исключения обрабатываться, рабочие процессы профилироваться.

****3. Основные требования к проекту****

****Паттерны проектирования**:**

- На серверной стороне реализовать паттерны: Декоратор, Абстрактный метод и RESTful API.
- На стороне клиента применить паттерны: Abstract Factory, Command, Strategy, Facade и Logging.

****Интерфейсы API**:**

- Использовать API, такие как Airsim и Mavlink, для взаимодействия с внешними системами управления и моделирования дронов.

****Библиотеки оптимизации**:**

- Применять библиотеки, такие как NumPy и OpenCV, для оптимизации задач обработки данных.

****Тестирование и оптимизация**:**

- Реализовать процессы логирования, обработки исключений и модульного тестирования.
- Проводить профилирование для анализа производительности и оптимизации кода с целью повышения скорости и эффективности.
- Управлять ресурсами и энергопотреблением для повышения эффективности работы программы.

****4. Структура проекта:****

****4.1. Основное приложение:****

4.1.1. **Server Module (Серверный модуль):**

- ****Назначение:**** - Обработка бизнес-логики и предоставление RESTful API Decorator, Abstract Method и RESTful API для управления дроном и извлечения данных.
- ****Функциональность:**** обеспечение конечных точек для управления дроном и получения данных - авторизация, маршрутизация запросов, взаимодействие с базой данных, логгирование и обработка исключений.

/project-root

|

|— /server

| |— __init__.py

| |— main.py

| |— helpers.py # объединяет decorators и abstract_methods

| |— drone_routes.py

| |— logging.py

****Описание файлов:****

****`__init__.py`**:**

Инициализирует пакет `server`, упрощает импортирование модулей внутри пакета `server`.

****`main.py`**:**

Основной файл для запуска серверного приложения. Инициализация и запуск веб-сервера, настройка маршрутизации и подключения к базе данных. Использован паттерн "Фасад" для предоставления единственной точки доступа к запуску приложения.

****`helpers.py`**:**

Содержит вспомогательные функции, такие как декораторы и абстрактные методы. Обеспечивает переиспользуемый код для валидации, авторизации и других аспектов бизнес-логики. Использует паттерны "Декоратор" для модификации поведения функций, "Шаблонный метод" для определения алгоритмической структуры.

****`drone_routes.py`**:**

Определяет маршруты API для управления дроном. Обработка HTTP-запросов, маршрутизация их к соответствующим методам и возврат ответов. Использует паттерны "Контроллер" в рамках MVC для обработки запросов и формирования ответов.

****`logging.py`**:**

Настройка и управление логированием в приложении. Обеспечивает конфигурацию логов, записи ошибок, информационных сообщений и отладочных данных. Использует паттерн "Одиночка" для обеспечения единственного экземпляра логгера в приложении.

4.1.2. **Drone Module (Модуль дрона):**

- ****Назначение:**** Управление операциями дрона и интеграция с API Airsim и Mavlink.
- ****Функциональность:**** Контроль полетов, выполнение миссий, взаимодействие с сенсорами.

/project-root

```
|
|
| — /drone
|   | — __init__.py
|   | — drone_controller.py
|   | — mission_manage.py
|   | — drone_manager.py
|   | — database_access.py
|   | — security.py
```

****Описание файлов:****

A) ****drone_controller.py****

- **** Назначение:**** Модуль `drone_controller.py` является частью системы управления дроном, реализующей клиентское и серверное взаимодействие. Основная задача модуля — предоставлять интерфейс для управления дроном, позволяя выполнять базовые команды, такие как взлет, движение вперед и повороты.

- ****Функциональность:****

****Управление дроном**:**

- Модуль предоставляет класс ``DroneController``, содержащий асинхронные методы для выполнения команд взлета, движения вперед и поворотов. Это позволяет эффективно управлять дроном, используя задержки для имитации реального времени выполнения команд.

****Командный интерфейс**:**

- Реализован интерфейс ``ICommand``, который определяет асинхронный метод ``execute`` для выполнения команд.

- Конкретные команды взлета (``Takeoff``), движения вперед (``MoveForward``) и поворота (``Turn``) реализуют интерфейс ``ICommand``, предоставляя конкретные реализации метода ``execute``.

**** Примененные паттерны проектирования****

- ****Команда****: Паттерн команды реализован через интерфейс ``ICommand`` и его конкретные реализации. Это позволяет инкапсулировать запросы в объекты, облегчая управление командами и их выполнение.

****Методы оптимизации****

- ****Асинхронное выполнение****: Использование асинхронных функций для выполнения команд позволяет оптимизировать работу, обеспечивая многозадачность и уменьшение времени ожидания между выполнением команд.

Методы тестирования

- Модуль включает функцию ``main``, которая демонстрирует пример использования команд для управления дроном. Это позволяет проводить базовые проверки корректности выполнения команд. Также в структуре проекта предусмотрено отдельное приложение для тестирования, содержащее модули тестирования всех критических функций.

Б) ****mission_manager.py****

- **** Назначение:**** Модуль является ключевым компонентом системы управления дронами, предназначенным для назначения и управления миссиями дронов. Он взаимодействует с менеджером дронов `/drone/drone_manager.py` и контроллером дронов `/drone/drone_controller.py`, предоставляющим управляющие команды пилотирования, получает списки валидированных дронов и назначает им соответствующие миссии и полетные задания.

- ****Используемые паттерны:****

- **Стратегия**: Паттерн стратегии реализован через интерфейс `IFlightStrategy`` и его конкретные реализации (`ReconMissionStrategy`` и `PatrolMissionStrategy``). Это позволяет гибко менять стратегию полета дронов без изменения контекста.

- **Контекст**: `DroneContext`` выступает в роли контекста, который управляет стратегиями полета и выполняет команды в зависимости от выбранной стратегии.

- **Функциональность**:

Управление стратегиями полета:

- Модуль предоставляет возможность управления различными стратегиями полета дронов, такими как разведывательные и патрульные миссии.

Для этого используется интерфейс `IFlightStrategy``, который определяет метод `execute`` для выполнения команд.

Контекст управления:

`DroneContext`` позволяет устанавливать стратегии полета и управлять списком команд, которые необходимо выполнить.

Обработка и валидация данных:

Класс `MissionManager`` отвечает за прием и валидацию списка дронов, а также за симуляцию назначения миссий.

Метод `check_completeness`` проверяет полноту переданного списка дронов.

- **Методы оптимизации и тестирования**

Логирование: В модуле применено логирование для отслеживания выполнения миссий и выявления ошибок, что облегчает отладку и мониторинг системы.

- **Очистка команд**: После выполнения миссии список команд очищается, что освобождает память и предотвращает повторное использование.

- Модуль включает пример использования, который демонстрирует передачу данных и проверку полноты списка дронов. Это позволяет проводить базовые проверки корректности работы методов.

В) **drone_manager.py**:

- **Назначение**:

Предназначен для управления дронами различных производителей, таких как DJI и AirSim, в процессе выполнения миссий. Он обеспечивает выбор, проверку и подключение дронов, а также взаимодействие с менеджером миссий для передачи валидированных дронов.

- ****Примененные паттерны проектирования****
 - **Легковес****: Для управления экземплярами дронов.
 - ****Фабрика****: Используется для создания объектов дронов различных производителей.
 - ****Абстрактная фабрика****: Для создания API управления дронами в зависимости от производителя.
 - ****Логирование****: Применяется для отслеживания и записи действий в системе.

- ****Функциональность:****
 - ****Создание дронов****: Используются фабрики ``DJIDroneFactory`` и ``AirSimDroneFactory`` для создания объектов дронов с использованием паттерна проектирования "Фабрика".
 - ****API Управление****: Предоставляются интерфейсы для подключения и отправки команд дронам через ``IDroneAPI``, ``AirSimAPI`` и ``DJIDroneAPI``. Эти классы реализуют паттерн "Абстрактная фабрика", обеспечивая создание соответствующего API в зависимости от производителя.
 - ****Логирование****: Класс ``DroneLogger`` отвечает за логирование действий, таких как одобрение и выбор дронов для миссий.
 - ****Выбор и проверка дронов для миссии****: Функции ``select_drone_for_mission`` и ``approve_drone_for_mission`` отвечают за выбор дронов на основе их характеристик, таких как емкость батареи, и проверку их готовности к миссии.
 - ****Интеграция с менеджером миссий****: Функция ``send_validated_drones_to_mission_manager`` передает валидированные дроны в модуль менеджера миссий для дальнейшего управления.

Методы оптимизации и тестирования

- Паттерн Легковес позволяет эффективно управлять памятью, уменьшая количество экземпляров объектов за счет разделения их состояния на общее и уникальное. В данном случае, общий объект дрона будет содержать неизменяемые данные (например, модель, производитель и т.д.), а уникальное состояние (например, заряд батареи) будет передаваться в методы в виде аргументов.
- Реализованы проверки на поддержку API для указанных производителей, обеспечивающие устойчивую работу системы при различных условиях.
- Логирование действий позволяет отслеживать состояние дронов и оперативно выявлять проблемы.
- Показан пример создания дронов и выбора подходящего для миссии.

Допущения

- Данные о дронах представлены в виде списка, а не через подключение к базе данных, для упрощения демонстрации и тестирования. В реальной системе предполагается интеграция с полноценной базой данных для хранения и управления информацией о дронах.

Г) ****database_access.py****

- **** Назначение:**** Доступ к базе данных с информацией о дронах.
- ****Функциональность:**** Управление доступом к базе данных, включая получение списка дронов, ввод параметров миссии и получение обратной связи от дрона.

Д) ****security.py****

- **** Назначение:**** - обеспечение безопасного доступа к дрону
- ****Функциональность:****
 - **Аутентификация**:** Подтверждение подлинности пользователей
 - **Авторизация**:** Определение прав доступа пользователей к различным частям приложения
 - **Шифрование данных**:** Защита данных от несанкционированного доступа
 - **Управление сессиями**:** Обеспечение безопасного управления пользовательскими сессиями, включая их создание, хранение и завершение. –
 - **Журналирование и мониторинг**:** Ведение журнала действий пользователей и мониторинг подозрительной активности.

Применяемые паттерны:

- **Singleton**:** обеспечение единственного доступа к управлению аутентификацией и сессиями.
- **Role-Based Access Control (RBAC)**:** управление правами доступа на основе ролей
- **Strategy**:** Для возможности использования различных алгоритмов шифрования.
- **Observer**:** Для мониторинга и реагирования на события безопасности.

****Описание работы приложения****

****Паттерн «Команда»****

****Командный интерфейс (ICommand)**:**

- абстрактный базовый класс, определяющий единственный метод `execute()`. Любая команда, которая может быть отдана дрону, будет реализована в этом интерфейсе.

****Классы конкретных команд**:**

- ``Takeoff``: Реализует метод ``execute()`` для вызова метода ``takeoff()`` на ``DroneController``.
- ``MoveForward``: Реализует метод ``execute()`` для вызова метода ``move_forward()`` на ``DroneController`` с заданным расстоянием.
- ``Поворот``: Реализует метод ``execute()`` для вызова метода ``turn()`` на ``DroneController`` с заданным градусом.

****Шаблон стратегии****

****Интерфейс стратегии (``IFlightStrategy``)**:**

- Это абстрактный базовый класс, определяющий метод ``execute()``, который принимает список команд. Различные стратегии будут реализовывать этот интерфейс для выполнения команд по-разному.

****Классы конкретных стратегий**:**

- ``ReconMissionStrategy``: Выполняет список команд в рамках разведывательной миссии. Метод ``execute()`` украшен ``SafetyCheck``, предлагая шаг безопасности или проверки перед выполнением.
- ``PatrolMissionStrategy``: Выполняет список команд в цикле для указанного количества патрулей. Также использует декоратор ``SafetyCheck``.

Декоратор `SafetyCheck**`:**

- декоратор выполняет некоторые проверки безопасности или валидации с использованием ``secret_key``, прежде чем разрешить выполнение команд.

Эта система позволяет гибко управлять беспилотными летательными аппаратами, инкапсулируя команды и стратегии, что позволяет легко расширять ее новыми командами или стратегиями без изменения существующего кода.

4.1.3. `Client Module (Клиентский модуль)**`**

- ****Назначение:**** Веб-интерфейс для взаимодействия пользователя с системой с использованием HTML, Bootstrap, Jinja2.
- ****Функциональность:**** Отображение UI, обработка пользовательских команд, интеграция с сервером. Использовать паттерны, такие как Abstract Factory, Command, Strategy, Facade и Logging для реализации компонентов пользовательского интерфейса и взаимодействий.

/project-root

|

|— /client

| |— __init__.py

| |— app.py


```
|  └─ /templates
|  |  └─ index.html
|  |  └─ control_panel.html
|  └─ device_manager.py
|  └─ sensor_manager.py
|  └─ user_interface.py
|  └─ logging_config.py
```

****Описание модулей****

A) ****app.py****

- ****Назначение****: Основной модуль для запуска и конфигурации приложения, обеспечивает основную инфраструктуру для работы клиентской части системы управления дроном, предоставляя удобный интерфейс для взаимодействия и управления компонентами системы.

****Функциональность****: Инициализация компонентов и запуск веб-приложения. Использует паттерны Facade для упрощенного доступа к подсистемам приложения.

- ****Инициализация компонентов****: Модуль создает и инициализирует экземпляры менеджеров устройств, датчиков и пользовательского интерфейса.

- ****Настройка и запуск веб-приложения****: Используя Flask, модуль настраивает маршруты для главной страницы и страницы панели управления, обеспечивая базовый интерфейс для взаимодействия с системой.

- ****Логирование****: Включает настройку логирования для отслеживания событий и упрощения диагностики.

****Примененные паттерны проектирования****

- ****Фасад (Facade)****: Модуль использует паттерн фасада через класс `ApplicationFacade`, который инкапсулирует инициализацию и управление различными компонентами системы, упрощая взаимодействие с ними, изолирует код клиента от сложных деталей реализации системы, изменения в этих деталях не должны затрагивать клиентский код.

****Методы оптимизации****

- ****Инкапсуляция логики инициализации****: Весь процесс инициализации компонентов и настройки приложения вынесен в отдельные методы (`initialize_components`, `run`), что упрощает поддержку и расширение кода.

- ****Модульное разделение ответственности****: Каждый компонент (менеджеры устройств, датчиков и интерфейса) инкапсулирован в собственный модуль, что улучшает читаемость и поддерживаемость кода.

Методы тестирования

- ****Интеграционное тестирование****: Проверка взаимодействия между компонентами через тестирование фасада приложения, чтобы удостовериться в корректности их взаимодействия и интеграции.

Б) **device_manager.py**

Назначение модуля

Модуль `device_manager.py` предназначен для управления устройствами, связанными с дроном, в частности для работы с устройствами, не используемыми непрерывно. Он отвечает за инициализацию, управление и обработку данных от таких устройств, а также за интеграцию с веб-интерфейсом для управления видеопотоком.

Функциональность

- **Регистрация и инициализация устройств**: Модуль позволяет регистрировать новые устройства и инициализировать все зарегистрированные устройства.
- **Обработка данных устройств**: Обеспечивается обработка данных от всех зарегистрированных устройств.
- **Управление видеопотоком**: Включает в себя функции для начала и остановки захвата видео, а также для трансляции видеопотока через веб-интерфейс.
- **Веб-интерфейс**: Использует Flask для создания маршрутов, позволяющих управлять видеопотоком через веб-страницу.

Примененные паттерны проектирования

- **Фабричный метод**: Используется для создания и инициализации устройств через `DeviceManager`.
- **Шаблон проектирования "Стратегия"**: Абстрактный класс `Device` и его конкретная реализация `Camera` позволяют изменять реализацию устройств без изменения кода, который их использует.

Оптимизация

- **Логирование**: Используется модуль `logging` для детального отслеживания работы приложения и упрощения диагностики ошибок.
- **Обработка ошибок**: Все ключевые методы содержат обработку исключений для предотвращения сбоев в работе системы.
- **Ресурсоэффективность**: Остановка видеопотока включает освобождение ресурсов, таких как закрытие захвата камеры.

Методы тестирования

- **Модульное тестирование**: Для каждого класса и метода могут быть написаны тесты, проверяющие корректность их работы в изолированных условиях.

- ****Интеграционное тестирование****: Проверка взаимодействия модуля ``device_manager.py`` с другим программным обеспечением, таким как серверное приложение и приложение для работы с дронами.

- ****Функциональное тестирование****: Тестирование через веб-интерфейс для проверки корректности работы функций управления видеопотоком.

Этот модуль является важной частью системы управления дроном, обеспечивая гибкость и надежность в работе с устройствами, необходимыми для выполнения миссий.

- ****Модульное тестирование****: Для каждого класса и метода могут быть написаны тесты, проверяющие корректность их работы в изолированных условиях.

- ****Интеграционное тестирование****: Проверка взаимодействия модуля ``device_manager.py`` с другим программным обеспечением, таким как серверное приложение и приложение для работы с дронами.

- ****Функциональное тестирование****: Тестирование через веб-интерфейс для проверки корректности работы функций управления видеопотоком.

Этот модуль является важной частью системы управления дроном, обеспечивая гибкость и надежность в работе с устройствами, необходимыми для выполнения миссий.

В) ****sensor_manager.py****

****Назначение ****: Модуль ``sensor_manager.py`` предназначен для управления сенсорами дрона, которые обеспечивают навигацию и телеметрию. Он отвечает за инициализацию, хранение и работу с устройствами, обеспечивающими поведение дрона и регулирование его положения в полете. Модуль также взаимодействует с менеджером дронов и менеджером миссий.

****Функциональность****

- Модуль предоставляет класс ``SensorManager``, который позволяет добавлять и удалять сенсоры, а также читать данные с них.

- Сенсоры, такие как ``Altimeter`` и ``GPSSensor``, реализуют интерфейс ``Sensor`` и предоставляют методы для чтения данных с устройств.

****Паттерн Наблюдатель****:

- Модуль позволяет добавлять и удалять наблюдателей, которые получают обновления при изменении данных сенсоров.

- Реализован абстрактный класс ``SensorObserver``, определяющий метод ``update``, который вызывается для передачи новых данных наблюдателям.

****Паттерны проектирования****

- **Команда**: позволяет инкапсулировать запросы в объекты, что помогает абстрагировать выполнение команд и упрощает управление ими. В контексте модуля ``sensor_manager.py`` этот паттерн может быть применен следующим образом:

Инкапсуляция действий: Каждый сенсор, такой как ``Altimeter`` или ``GPSSensor``, может быть рассмотрен как исполнение команды, которая запрашивает данные с устройства. Это позволяет легко добавлять новые типы сенсоров, просто реализуя интерфейс ``Sensor``, который содержит методы для выполнения команд (например, ``read_data``).

Наблюдатель: Используется для уведомления различных частей системы о новых данных сенсоров, что обеспечивает реакцию на изменения в реальном времени.

Модуль взаимодействует с остальной системой, применяя паттерн Команда, что позволяет абстрагировать выполнение команд дрона. Используется для организации взаимодействия между объектами, когда один объект уведомляет другие о произошедших изменениях. В ``sensor_manager.py`` это реализовано следующим образом:

Абстрактный класс ``SensorObserver``: Этот класс определяет интерфейс для наблюдателей, который включает метод ``update(data)``. Наблюдатели реализуют этот интерфейс и определяют, как они будут обрабатывать новые данные.

Регистрация и уведомление наблюдателей: Класс ``SensorManager`` поддерживает список наблюдателей, которые подписываются на обновления. Когда данные сенсора обновляются, ``SensorManager`` вызывает метод ``update`` у каждого наблюдателя, передавая ему новые данные.

Изоляция изменений: Благодаря использованию паттерна "Наблюдатель", изменения в данных сенсоров автоматически передаются всем заинтересованным частям системы, что упрощает реакцию на события в реальном времени и снижает связанность между компонентами.

Методы оптимизации

- **Асинхронное взаимодействие**: Модуль подразумевает использование асинхронного режима для повышения эффективности обработки данных и многопоточности, хотя в данном фрагменте кода это не представлено явно.

Методы тестирования

- Для тестирования функциональности модуля в коде предоставлен пример использования, где создается экземпляр ``SensorManager``, добавляются сенсоры и наблюдатели, а также демонстрируется чтение данных сенсоров и уведомление наблюдателей.

Г) **``user_interface.py``**

- **Назначение**: Управление компонентами пользовательского интерфейса и взаимодействием с пользователем.

- **Функциональность**: Отображение UI и обработка пользовательских команд. Использует паттерны Abstract Factory для создания UI-компонентов, Command для обработки пользовательских команд.

Д) **logging_config.py**

- **Назначение**: Конфигурация системы логирования для клиентского модуля.

- **Функциональность**: Настройка логирования и управление логами приложения. Использует паттерн Logging для организации и записи логов.

Е) Директория **/templates**

- **Назначение**: Хранение HTML-шаблонов для веб-интерфейса.

- **Функциональность**: Поддержка рендеринга страниц для пользовательского интерфейса с использованием Jinja2 и стилей Bootstrap.

4.2. Вспомогательные приложения:

А) **User Manager (Управление пользователями)**:

- **Назначение**: создание и администрирование базы пользователей, управление пользователями, регистрация и авторизация.

- **Функциональность**: Аутентификация и управление данными пользователей хэширование паролей пользователей, предоставить услуги аутентификации для основного приложения.

Б) **Drone Manager (Управление дронами)**:

- **Назначение**: Управление конфигурациями дронов – создавать и вести базу данных дронов и их конфигураций.

- **Функциональность**: Взаимодействие с API для управления дронами и предоставление данных основной системе через абстрактную фабрику API на примере API Airsim и Mavlink

/project-root

|

|— /support

| |— /user_manager

| | |— __init__.py

| | |— models.py

| | |— auth.py

| | |— database.py

| |— /drone_manager

```
|   ├── __init__.py
|   ├── models.py
|   ├── api_factory.py
|   └── database.py
```

4.3. Тестирование

```
/project-root
|
└── /tests
    ├── __init__.py
    ├── test_server.py
    ├── test_drone.py
    ├── test_client.py
    ├── test_user_manager.py
    └── test_drone_manager.py
```

****Примечание:**** Полная файловая структура проекта в приложении 1.

****5. Порядок приоритетов разработки модулей****

****Server Module (Модуль сервера)****

****Drone Module (Модуль дрона)****

****User Manager (Управление пользователями)****

****Drone Manager (Управление дронами)****

****Client Module (Клиентский модуль)****

****Tests (Тестирование)**** - по мере разработки каждого из предыдущих модулей.

Предложенная структура обеспечивает модульность и расширяемость системы. Разработка начнется с серверной логики и управления дроном, что позволит быстро развернуть основные функциональные возможности системы управления квадрокоптером.

5. Этапы выполнения задачи:

А) ****Дизайн и архитектура****:

- Определить архитектуру серверной, клиентской и вспомогательных частей приложения - программное обеспечение для управления полетом, интерфейсы для взаимодействия с пользователем, системы обработки данных и связи.

- Установить модули и их обязанности.

Б) ****Реализация****:

- Разработать серверную логику с использованием Flask и реализовать RESTful API конечные точки. Начать с самых критичных компонентов, таких как системы безопасности и основные функции управления. Постепенно переходите к второстепенным функциям, таким как пользовательские интерфейсы и дополнительные возможности.

- Создать веб-интерфейс и интегрировать его с серверным API.

- Реализовать логику управления дроном и взаимодействие с API.

В) ****Интеграция****:

- Подключить User Manager для управления потоками аутентификации и авторизации.

- Интегрировать Drone Manager для доступа к конфигурациям дронов и API.

Г) ****Оптимизация и тестирование****:

- Реализовать логирование и обработку исключений во всех модулях.

- Разработать и выполнить модульные тесты для ключевых компонентов.

- Анализировать приложение на наличие узких мест в производительности и применять оптимизацию.

Д) ****Управление ресурсами****:

- Осуществлять мониторинг и управление потреблением ресурсов для обеспечения эффективности.

- Реализовать стратегии управления энергопотреблением при работе с дронами.

Файловая структура проекта

/project-root

|

└─ /server

| └─ __init__.py

| └─ app.py

| └─ decorators.py

| └─ abstract_methods.py

| └─ /routes

| | └─ __init__.py

| | └─ drone_routes.py

| └─ /utils

| └─ __init__.py

| └─ logging.py

|

└─ /drone

| └─ __init__.py

| └─ drone_control.py

| └─ airsim_interface.py

| └─ mavlink_interface.py

|

└─ /client

| └─ __init__.py

| └─ app.py

| └─ /templates

| | └─ index.html

| | └─ control_panel.html

| └─ /static

| | └─ /css

| | └─ /js

| └─ abstract_factory.py


```
| | └─ command.py
| | └─ strategy.py
| | └─ facade.py
|
└─ /support
| | └─ /user_manager
| | | └─ __init__.py
| | | └─ models.py
| | | └─ auth.py
| | | └─ database.py
| | └─ /drone_manager
| | | └─ __init__.py
| | | └─ models.py
| | | └─ api_factory.py
| | | └─ database.py
|
└─ /tests
    └─ __init__.py
    └─ test_server.py
    └─ test_drone.py
    └─ test_client.py
    └─ test_user_manager.py
    └─ test_drone_manager.py
```