Студент:

ФИО, группа

Руководитель:

Ученая степень, ФИО

Тема:

**Реализация и тестирование системы**

1. **Реализация компонентов системы (с примерами исходного кода)**

В реализации компонентов системы имеется сервис Data Management, который предоставляет функциональность для работы с информацией о дронах и камерах. Этот сервис состоит из следующих файлов:

uav/mod.rs - Основной файл, определяющий структуру Uav и методы для создания экземпляров дронов с заданными параметрами.

Пример структуры Uav:

#[derive(Debug, Deserialize, Serialize)]

pub struct Uav {

id: u64, // uav id

pub name: String, // uav name

pub max\_payload\_mass: u64, // maximum payload in grams

pub flight\_duration: u64, // average flight duration in seconds

pub takeoff\_speed: f64, // average takeoff speed in meters per second

pub flight\_speed: f64, // average flight speed in meters per second

pub min\_altitude: f64, // minimum safe flight altitude in meters

pub max\_altitude: f64, // maximum safe flight altitude in meters

}

**Рис. 1.** Листингфункции для определения количества кластеров

uav/uav\_handle.rs - Файл содержит функции для работы с дронами через Tauri, такие как создание, обновление и удаление дронов, а также получение списка всех дронов.

Пример функции для создания нового дрона:

#[tauri::command]

pub fn new\_uav(uav: Uav) -> String {

let conn = Connection::open("mydatabase.db").expect("Cant open base");

println!("Received new UAV: {:?}", uav);

match uav\_sql::insert(&uav, &conn) {

Ok(\_) => "Ok".to\_string(),

Err(e) => e.to\_string(),

}

}

**Рис. 1.** Листингфункции для определения количества кластеров

1. uav/uav\_sql.rs - Файл содержит функции для работы с базой данных, включая создание таблицы для хранения информации о дронах, а также вставка, обновление, удаление и получение списка всех дронов.

Пример функции для создания таблицы в базе данных:

pub fn create\_table(conn: &Connection) -> Result<usize> {

conn.execute(

"CREATE TABLE IF NOT EXISTS uav (

uav\_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,

uav\_name TEXT NOT NULL,

uav\_max\_payload\_mass INTEGER NOT NULL CHECK (uav\_max\_payload\_mass >= 0),

uav\_flight\_duration INTEGER NOT NULL CHECK (uav\_flight\_duration >= 0),

uav\_takeoff\_speed REAL NOT NULL CHECK (uav\_takeoff\_speed >= 0),

uav\_flight\_speed REAL NOT NULL CHECK (uav\_flight\_speed >= 0),

uav\_min\_altitude REAL DEFAULT 0 NOT NULL CHECK (uav\_min\_altitude >= 0),

uav\_max\_altitude REAL DEFAULT 0 NOT NULL CHECK (uav\_max\_altitude >= 0)

)",

(), // empty list of parameters.

)

}

**Рис. 1.** Листингфункции для определения количества кластеров

camera/mod.rs - Основной модуль, который содержит структуру и методы для работы с камерой. Структура Camera содержит различные свойства камеры, включая ее идентификатор, имя, массу, угол обзора по оси X и разрешение. Здесь также определены метод для создания новых камер.

Пример структуры Camera:

pub struct Camera {

id: u64, // id

pub name: String, // name

pub mass: u64, // mass in grams

pub fov\_x: f64, // x-axis viewing angle in degrees

pub resolution\_x: u16, // camera resolution x

pub resolution\_y: u16, // camera resolution y

}

**Рис. 1.** Листингфункции для определения количества кластеров

camera/camera\_handle.rs - Этот модуль предоставляет функции для обработки запросов к базе данных. Здесь определены функции для создания новой камеры, обновления существующей камеры, удаления камеры и получения всех камер из базы данных. Каждая из этих функций открывает соединение с базой данных и вызывает соответствующую функцию из модуля camera\_sql.

Пример функции для обновления данных о камере.

#[tauri::command]

pub fn update\_camera(camera: Camera) -> String {

let conn = Connection::open("mydatabase.db").expect("Cant open base");

println!("Received updated camera: {:?}", camera);

match camera\_sql::update(&camera, &conn) {

Ok(\_) => "Ok".to\_string(),

Err(e) => e.to\_string(),

}

}

**Рис. 1.** Листингфункции для определения количества кластеров

camera/camera\_sql.rs - Этот модуль содержит функции для взаимодействия с базой данных SQLite. Здесь определены функции для создания таблицы в базе данных, вставки новой камеры в базу данных, обновления существующей камеры, удаления камеры из базы данных и получения всех камер из базы данных. Все эти функции работают непосредственно с базой данных, выполняя соответствующие SQL-запросы.

Пример функции обновления данных о камере представлен на рисунке

Сервис Algorithms, описанный файлом algorithms.rs, представляет собой ключевую составляющую системы и выполняет алгоритмические функции. Он включает в себя следующие функции:

1. discretize\_area: Эта функция принимает вектор кортежей, представляющих собой координаты x и y полигона, и размеры фотографии (ширину и высоту). Функция начинает с инициализации минимальных и максимальных значений x и y. После этого функция применяет алгоритм поиска точек внутри полигона, основанный на лучевом методе. После того как все точки были проверены, функция возвращает вектор кортежей, представляющих собой координаты центров прямоугольников, которые пересекаются с полигоном. Реализация представлена в листинге во вложении
2. nearest\_neighbor: Эта функция принимает вектор точек и начальную точку, а затем использует алгоритм ближайшего соседа для определения оптимального пути через все точки. Функция инициализирует набор оставшихся точек и результатирующий вектор с начальной точкой. Функция затем входит в цикл, продолжающийся, пока все точки не будут обработаны. В каждой итерации цикла функция находит ближайшую точку к текущей точке, удаляет ее из оставшихся точек и добавляет ее в результатирующий вектор.
3. euclidean\_distance: Это вспомогательная функция, которая принимает две точки и возвращает евклидово расстояние между ними. Расстояние вычисляется как квадратный корень из суммы квадратов разностей координат x и y.
4. brute\_force: Эта функция использует алгоритм полного перебора для поиска оптимального пути через набор точек. Функция начинает с создания отдельного потока для каждой точки, каждый из которых запускает функцию brute\_force\_helper. Эта вспомогательная функция рекурсивно исследует все возможные пути через оставшиеся точки, обновляя лучший путь при обнаружении более короткого пути.
5. calculate\_distance: Эта функция вычисляет общую длину пути, проходящего через заданный набор точек
6. **Реализация интерфейса системы (пользовательский и/или программный интерфейс)**
7. **Тестирование системы**