**Документация проекта**

# Описание проекта

Этот проект представляет собой простое серверное приложение на Flask с использованием базы данных PostgreSQL. Приложение предоставляет функции регистрации и аутентификации пользователей. Только администраторы могут получать список пользователей, добавлять новых пользователей, обновлять информацию о пользователях и удалять пользователей.

# Структура проекта

```  
project/  
│  
├── app/  
│ ├── \_\_init\_\_.py  
│ ├── models.py  
│ ├── routes.py  
│ ├── utils.py  
│  
├── migrations/  
│  
├── add\_admin.py  
├── config.py  
├── requirements.txt  
├── run.py  
```

# Описание файлов

- `app/\_\_init\_\_.py`: Инициализация приложения Flask и настройка расширений.  
- `app/models.py`: Определение модели пользователя.  
- `app/routes.py`: Определение маршрутов и логики обработки запросов.  
- `app/utils.py`: Утилиты для работы с паролями.  
- `migrations/`: Папка для хранения миграций базы данных.  
- `add\_admin.py`: Скрипт для добавления администратора вручную.  
- `config.py`: Конфигурация приложения.  
- `requirements.txt`: Список зависимостей проекта.  
- `run.py`: Точка входа для запуска приложения.

# Инструкции по установке и запуску

1. Клонируйте репозиторий и перейдите в директорию проекта.

2. Создайте и активируйте виртуальное окружение:

```sh  
python -m venv .venv  
.venv\Scripts\activate  
```

3. Установите зависимости:

```sh  
pip install -r requirements.txt  
```

4. Установите PostgreSQL и создайте базу данных:

```sql  
CREATE DATABASE myapp;  
CREATE USER cofedish WITH PASSWORD '123444';  
GRANT ALL PRIVILEGES ON DATABASE myapp TO cofedish;  
```

5. Установите переменные окружения:

```sh  
set DATABASE\_URL=postgresql://cofedish:123444@localhost/myapp  
set SECRET\_KEY=your\_secret\_key  
```

6. Инициализируйте и примените миграции базы данных:

```sh  
flask db init  
flask db migrate -m "Initial migration"  
flask db upgrade  
```

7. Запустите приложение:

```sh  
flask run  
```

# Описание API маршрутов

## Регистрация пользователя

`POST /register`

Регистрирует нового пользователя.

Пример тела запроса:

```json  
{  
 "username": "newuser",  
 "password": "newpassword"  
}  
```

Ответ при успешной регистрации (201):

```json  
{  
 "message": "User created successfully"  
}  
```

## Авторизация пользователя

`POST /login`

Авторизует пользователя.

Пример тела запроса:

```json  
{  
 "username": "user",  
 "password": "password"  
}  
```

Ответ при успешной авторизации (200):

```json  
{  
 "message": "Logged in successfully",  
 "is\_admin": true  
}  
```

## Получение списка пользователей

`GET /users`

Возвращает список всех пользователей (доступно только для администраторов).

Пример запроса с использованием curl:

```sh  
curl -X GET http://localhost:5000/users -u admin:adminpassword  
```

Ответ при успешном запросе (200):

```json  
[  
 { "id": 1, "username": "user1" },  
 { "id": 2, "username": "user2" },  
 { "id": 3, "username": "admin" }  
]  
```

## Обновление информации о пользователе

`PUT /users/<int:id>`

Обновляет информацию о пользователе (доступно только для администраторов).

Пример тела запроса:

```json  
{  
 "username": "updateduser",  
 "password": "updatedpassword"  
}  
```

Ответ при успешном обновлении (200):

```json  
{  
 "message": "User updated successfully"  
}  
```

## Удаление пользователя

`DELETE /users/<int:id>`

Удаляет пользователя (доступно только для администраторов).

Пример запроса с использованием curl:

```sh  
curl -X DELETE http://localhost:5000/users/1 -u admin:adminpassword  
```

Ответ при успешном удалении (200):

```json  
{  
 "message": "User deleted successfully"  
}  
```

## Добавление нового пользователя вручную

`POST /add\_user`

Добавляет нового пользователя вручную (доступно только для администраторов).

Пример тела запроса:

```json  
{  
 "username": "newadmin",  
 "password": "newpassword",  
 "is\_admin": true  
}  
```

Ответ при успешном добавлении (201):

```json  
{  
 "message": "User added successfully"  
}  
```