hw\_num: 20

В данном задании вы продолжите совершенствовать REST API для фитнес клуба. Вы перенесёте логику работы в отдельные блюпринты Flask, добавите систему авторизации с API-ключами и разграничение прав доступа на основе ролей пользователей.

hw\_theme:

  - Flask

  - Blueprints

  - REST API

  - PeeWee

  - JSON

  - HTTP

  - python

**### Технологии: 🛠️**

- Python

- Flask

- Flask Blueprints

- PeeWee ORM

- JSON

- HTTP-методы (GET, POST, PUT, DELETE)

- API-ключи и авторизация

**## Задание 👷‍♂️**

**### Часть 1: Создание блюпринтов**

1. **\*\*Создание структуры каталогов для блюпринтов:\*\***

   - Создайте каталог `blueprints` в корне проекта

   - Внутри создайте два подкаталога:

     - `trainers` для маршрутов, связанных с тренерами

     - `appointments` для маршрутов, связанных с записями на тренировки

2. **\*\*Реорганизация кода:\*\***

   - В каждом из подкаталогов создайте файлы:

     - `\_\_init\_\_.py` — для корректной работы пакета

     - `routes.py` — для определения маршрутов

   - Перенесите соответствующие обработчики маршрутов из `app.py` в файлы `routes.py`, создав блюпринты:

**\*\*Пример для файла blueprints/trainers/routes.py:\*\***

from flask import Blueprint, request, jsonify

from models import Trainers # импортируйте модели из вашего модуля trainers

# Создание блюпринта с префиксом /trainers

trainers\_bp = Blueprint(' trainers', \_\_name\_\_, url\_prefix='/ trainers ')

# Перенесите сюда соответствующие маршруты

@ trainers \_bp.route('/', methods=['GET'])

def get\_trainers ():

    # логика получения списка мастеров

    pass

3. **\*\*Обновление главного файла приложения:\*\***

   - В файле `app.py` импортируйте созданные блюпринты и зарегистрируйте их в приложении:

**\*\*Пример файла app.py:\*\***

```python

from blueprints.trainers.routes import trainers \_bp

from blueprints.appointments.routes import appointments\_bp

# ... создание приложения Flask ...

# Регистрация блюпринтов

app.register\_blueprint(trainers\_bp)

app.register\_blueprint(appointments\_bp)

```

**### Часть 2: Добавление системы авторизации и разграничение прав доступа**

1. **\*\*Создание модуля для хранения ключей API:\*\***

   - Создайте файл `auth.py` в корне проекта

   - Определите в нём список пользователей с ключами API и ролями:

**# Пользователи с их API-ключами и ролями**

USERS = [

    {

        "username": "admin",

        "api\_key": "admin\_secret\_key\_123",

        "role": "admin"

    },

    {

        "username": "user",

        "api\_key": "user\_readonly\_key\_456",

        "role": "user"

    }

]

def is\_valid\_api\_key(api\_key):

    """

    Проверяет, является ли ключ API действительным

    """

    return any(user["api\_key"] == api\_key for user in USERS)

def is\_admin(api\_key):

    """

    Проверяет, имеет ли пользователь с данным ключом API права администратора

    """

    return any(user["api\_key"] == api\_key and user["role"] == "admin" for user in USERS)

```

2. \*\*Обновление маршрутов для проверки ключей API:\*\*

   - Добавьте проверку ключей API во все маршруты

   - Разграничьте права доступа:

     - Администраторы могут выполнять операции чтения, создания, обновления и удаления

     - Обычные пользователи могут только читать данные

   Пример добавления проверки для маршрута получения мастера по ID:

   ```python

   from auth import is\_valid\_api\_key, is\_admin

   import json

   @ trainers\_bp.route('/<int:id>', methods=['GET'])

   def get\_trainers (id):

       # Проверка API-ключа

       api\_key = request.headers.get('api\_key')

       if not is\_valid\_api\_key(api\_key):

           return Response(

               json.dumps({"error": "Неверный API-ключ"}, ensure\_ascii=False),

               status=403,

               mimetype="application/json; charset=utf-8"

           )

       # Логика получения мастера по ID

       try:

           # ... код для получения мастера ...

       except Exception as e:

           # ... обработка ошибок ...

   ```

   Пример добавления проверки прав для маршрута удаления мастера:

   ```python

   @ trainers\_bp.route('/<int:id>', methods=['DELETE'])

   def delete\_trainers(id):

       # Проверка API-ключа

       api\_key = request.headers.get('api\_key')

       if not is\_valid\_api\_key(api\_key):

           return Response(

               json.dumps({"error": "Неверный API-ключ"}, ensure\_ascii=False),

               status=403,

               mimetype="application/json; charset=utf-8"

           )

       # Проверка прав администратора

       if not is\_admin(api\_key):

           return Response(

               json.dumps({"error": "Отказано в доступе. Требуются права администратора"}, ensure\_ascii=False),

               status=403,

               mimetype="application/json; charset=utf-8"

           )

       # Логика удаления тренера

       # ...

   ```

3. **\*\*Таблица прав доступа для маршрутов:\*\***

| Маршрут | Метод | Администратор | Обычный пользователь |

|---------|-------|---------------|---------------------|

| `/ trainers` | GET | ✅ | ✅ |

| `/ trainers/<id>` | GET | ✅ | ✅ |

| `/ trainers` | POST | ✅ | ❌ |

| `/masters/<id>` | PUT | ✅ | ❌ |

| `/ trainers/<id>` | DELETE | ✅ | ❌ |

| `/appointments` | GET | ✅ | ✅ |

| `/appointments/<id>` | GET | ✅ | ✅ |

| `/appointments/ trainers/< trainers\_id>` | GET | ✅ | ✅ |

| `/appointments` | POST | ✅ | ❌ |

| `/appointments/<id>` | PUT | ✅ | ❌ |

| `/appointments/<id>` | DELETE | ✅ | ❌ |

**### Часть 3: Тестирование API с использованием ключей и документация**

1. **\*\*Протестируйте работу API:\*\***

   - Используйте Postman или аналогичный инструмент для тестирования маршрутов с передачей API-ключей в заголовках запросов

   - Выполните тесты с ключами администратора и обычного пользователя

2. **\*\*Документирование API:\*\***

   - Обновите существующую документацию API (если она была создана ранее) или создайте новую, указав:

     - Информацию о необходимости передачи API-ключа в заголовке `api\_key`

     - Подробности о правах доступа для разных типов пользователей

3. **\*\*Сделайте скриншоты для подтверждения работы системы авторизации:\*\***

   - Запрос с корректным ключом администратора (успешный ответ)

   - Запрос с корректным ключом обычного пользователя (успешный для GET, отказ для POST/PUT/DELETE)

   - Запрос с некорректным ключом (отказ в доступе)

[!info]

**### Передача API-ключа в заголовке запроса**

В инструментах вроде Postman добавьте заголовок `api\_key` со значением ключа:

```http

api\_key: admin\_secret\_key\_123

```

В Python-запросах с библиотекой `requests`:

```python

response = requests.get(

    "http://localhost:5000/ trainers",

    headers={"api\_key": "admin\_secret\_key\_123"}

)

```

**### Функциональная структура проекта после выполнения задания**

```text

├── app.py                   # Основной файл приложения

├── models.py                # Модели PeeWee

├── auth.py                  # Функции авторизации и данные пользователей

├── blueprints/

│   ├── \_\_init\_\_.py

│   ├── trainers /

│   │   ├── \_\_init\_\_.py

│   │   └── routes.py        # Маршруты для мастеров

│   ├── appointments/

│   │   ├── \_\_init\_\_.py

│   │   └── routes.py        # Маршруты для записей

```

**## Критерии оценки 👌**

1. **\*\*Структура проекта и блюпринты: 6 баллов\*\***

   - Корректно созданы блюпринты с соответствующими url\_prefix

   - Логика маршрутов успешно перенесена из основного файла в блюпринты

   - Блюпринты правильно импортированы и зарегистрированы в основном приложении

2. **\*\*Система авторизации: 3 балла\*\***

   - Корректная проверка API-ключей в заголовках запросов

   - Правильная реализация проверки ролей пользователей

   - Соблюдение разграничения прав доступа согласно таблице

3. **\*\*Тестирование и документация: 2 балла\*\***

   - Предоставлены скриншоты с тестами, подтверждающими работу системы авторизации

   - Запросы, требующие прав администратора, недоступны для обычных пользователей

   - Запросы с некорректными ключами отклоняются с подходящим статус-кодом

4. **\*\*Кодирование и структура: 1 балл\*\***

   - Код соответствует стилю Python (PEP 8)

   - Корректная обработка ошибок и возврат соответствующих статус-кодов

   - Правильная обработка кириллических символов в ответах