Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Томский государственный университет систем
управления и радиоэлектроники»

Факультет систем управления (ФСУ)

Кафедра автоматизированных систем управления (АСУ)

&lab_name
отчет по лабораторной работе №& по дисциплине
«&course_name»

Обучающийся гр. 431-3
_____ Сергиевский Д.В.
«&__» &_____ 202& г.

Проверил: доцент каф. АСУ, к.т.н.
_____ &Фамилия И.О.
«&__» &_____ 202& г.

Томск 202&

Содержание

Введение.....3
1 Задание из методического пособия.....4
2 Задание по варианту.....6
2.1 Постановка задачи.....6
2.2 Структура проекта.....8
2.3 Описание работы приложения.....10
3 Вывод.....13
Приложение А.....14

Введение

В рамках данной лабораторной работы требуется ознакомиться с параграфом 3 методического пособия, с модулями для автоматической документации API и разработать небольшое веб-приложение.

Цели:

- Ознакомиться с инструментами автоматической документации.

Задания:

- Повторить действия, изложенные в методическом пособии.
- Разработать веб-приложение в соответствии с заданием по варианту.
- Выполнить индивидуальное задание при получении оного.

Задание по варианту (18):

- Создать свой собственный веб-сервис и API, реализующий возврат данных по заданию в соответствии с вариантом, где задана простая предметная область. Необходимо разработать простую модель данных, в соответствии с которой будут храниться данные, соответствующие предметной области. Количество полей должно быть более четырех, два-три поля могут быть строкового или другого типа, остальные – числового. Веб-сервис должен предоставлять возможность сортировки по всем полям записей, выдавать среднее, максимальное и минимальное значение по числовым полям, добавлять, удалять записи и обновлять записи, например, по идентификатору.
  Предметная область: спортивные состязания.

Индивидуальное задание:

-

1 Пример использования Flagger

Задание: выполнить изложенные в методическом пособии примеры для ознакомления с основами использования Swagger посредством модуля Flagger.

В ходе данного задания был разобран пример из методического пособия.

Пример работы приложения представлен на Рисунках 1-2.

Исходные файлы приложения были помещены в директорию manual_examples/flagger_example, размещенную в корневой директории проекта. Исходный код представлен в Приложении А.

Во время работы приложения из библиотеки Flagger выбрасывались исключения при обработке адреса /apidocs, что приводило к невозможности повторения действий из методического пособия. Причину подобного поведения обнаружить не удалось из-за недостаточного понимания внутренней работы данной библиотеки. Тем не менее, после добавления в один из файлов библиотеки нескольких проверок для обработки возникающих ситуаций, приложение показало стабильную работу во время ознакомления с интерфейсом Swagger. Поскольку данное приложение рассматривалось исключительно в качестве примера и в дальнейшем не подразумевалась работа с модулем Flagger, данный результат показался достаточным.

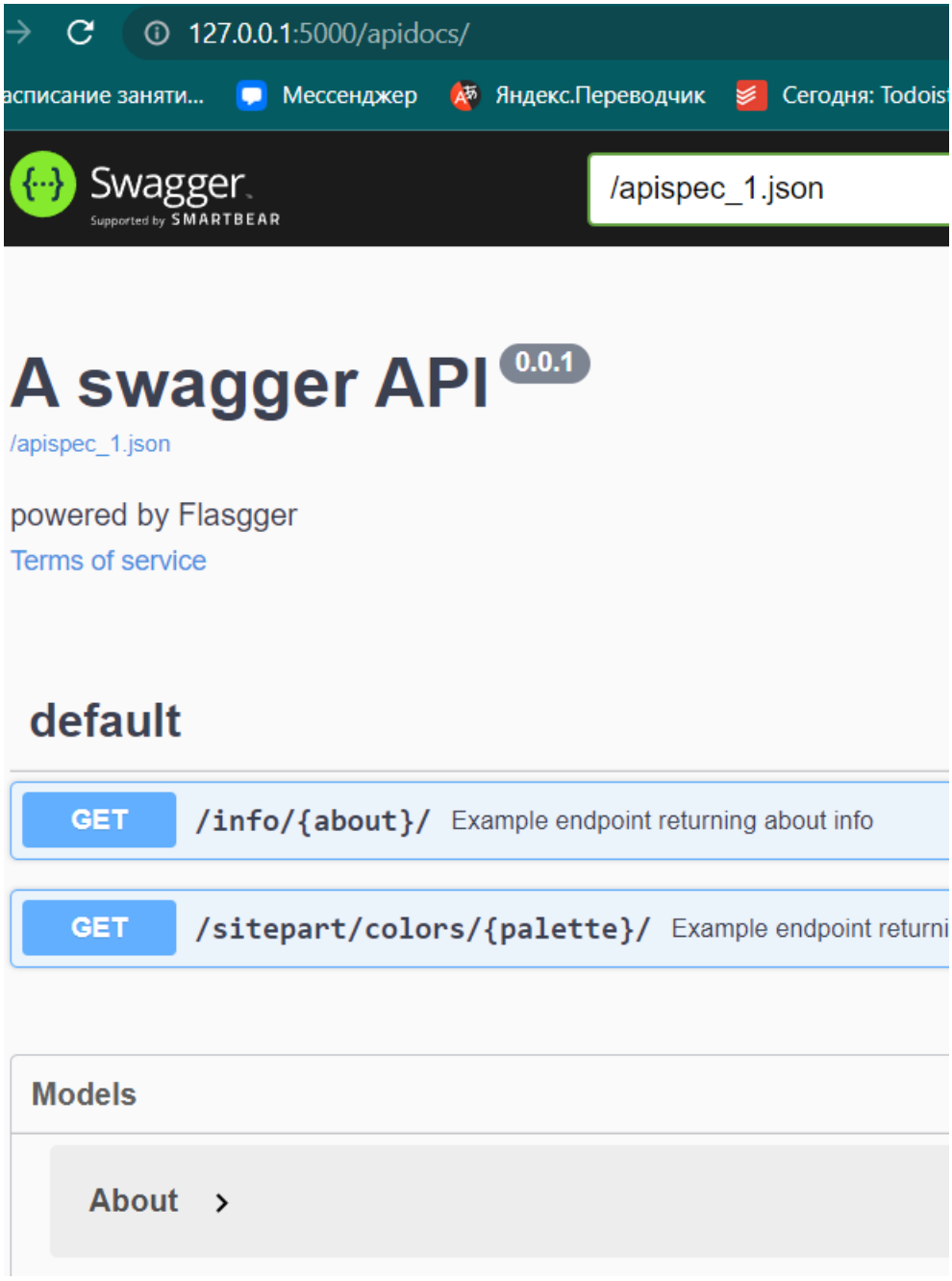

Рисунок 1.1 - пример отображения информации об API

Рисунок 1.2 - результата запроса

2 Пример использования flask_restplus

Задание: выполнить изложенные в методическом пособии примеры для ознакомления с основами использования Swagger посредством модуля flask_restplus.

В ходе данного задания был разобран пример из методического пособия.

В связи с многочисленными ошибками импорта, запустить данный пример без внесения изменений не удалось. Для решения данной проблемы был произведен откат нескольких модулей до более старых версий и был использован интерпретатор версии 3.9. Проблема с импортом пакета flask.scaffold была решена удалением строк с его использованием, что не повлекло заметных изменений в работе приложения.

Пример работы приложения представлен на Рисунках 1-2.

Исходные файлы приложения были помещены в директорию manual_examples/flask_restplus_example, размещенную в корневой директории проекта. Исходный код представлен в Приложении Б.

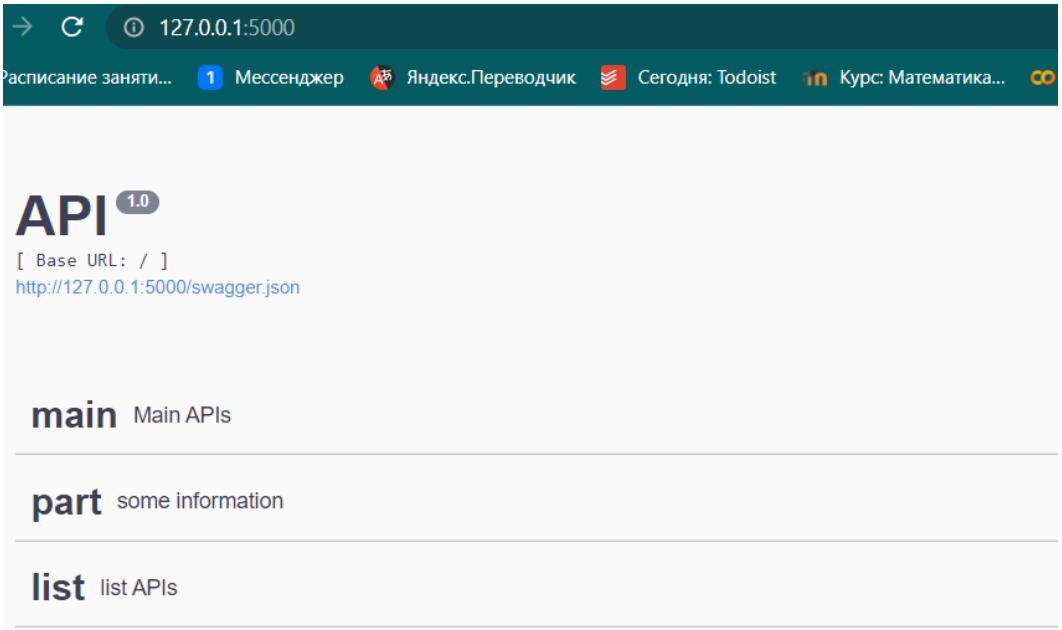

Рисунок 2.1 - страница документации

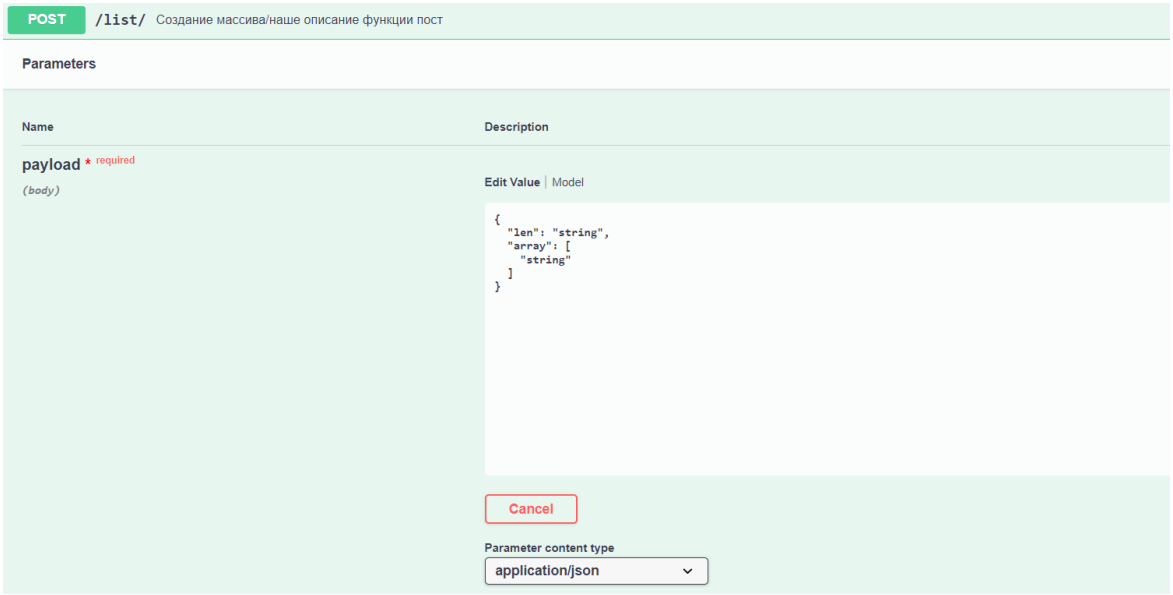

Рисунок 2.2 - пример ввода данных для запроса

3 Задание по варианту

Задание (вариант 18): Создать свой собственный веб-сервис и API, реализующий возврат данных по заданию в соответствии с вариантом, где задана простая предметная область. Необходимо разработать простую модель данных, в соответствии с которой будут храниться данные, соответствующие предметной области. Количество полей должно быть более четырех, два-три поля могут быть строкового или другого типа, остальные – числового. Веб-сервис должен предоставлять возможность сортировки по всем полям записей, выдавать среднее, максимальное и минимальное значение по числовым полям, добавлять, удалять записи и обновлять записи, например, по идентификатору. Предметная область: спортивные состязания.

В качестве модели данных был выбран участник некого спортивного состязания. В качестве полей были выбраны следующие: имя, страна, категория, количество очков и количество медалей. Первые три принимают произвольное строковое значение, остальные — числовое. Имя также выступает в качестве идентификатора.

В качестве фреймворка, осуществляющего документацию, был выбран FastAPI.

Результаты работы приложения представлены на Рисунках 1-3.

Исходный код приведен в Приложении В.

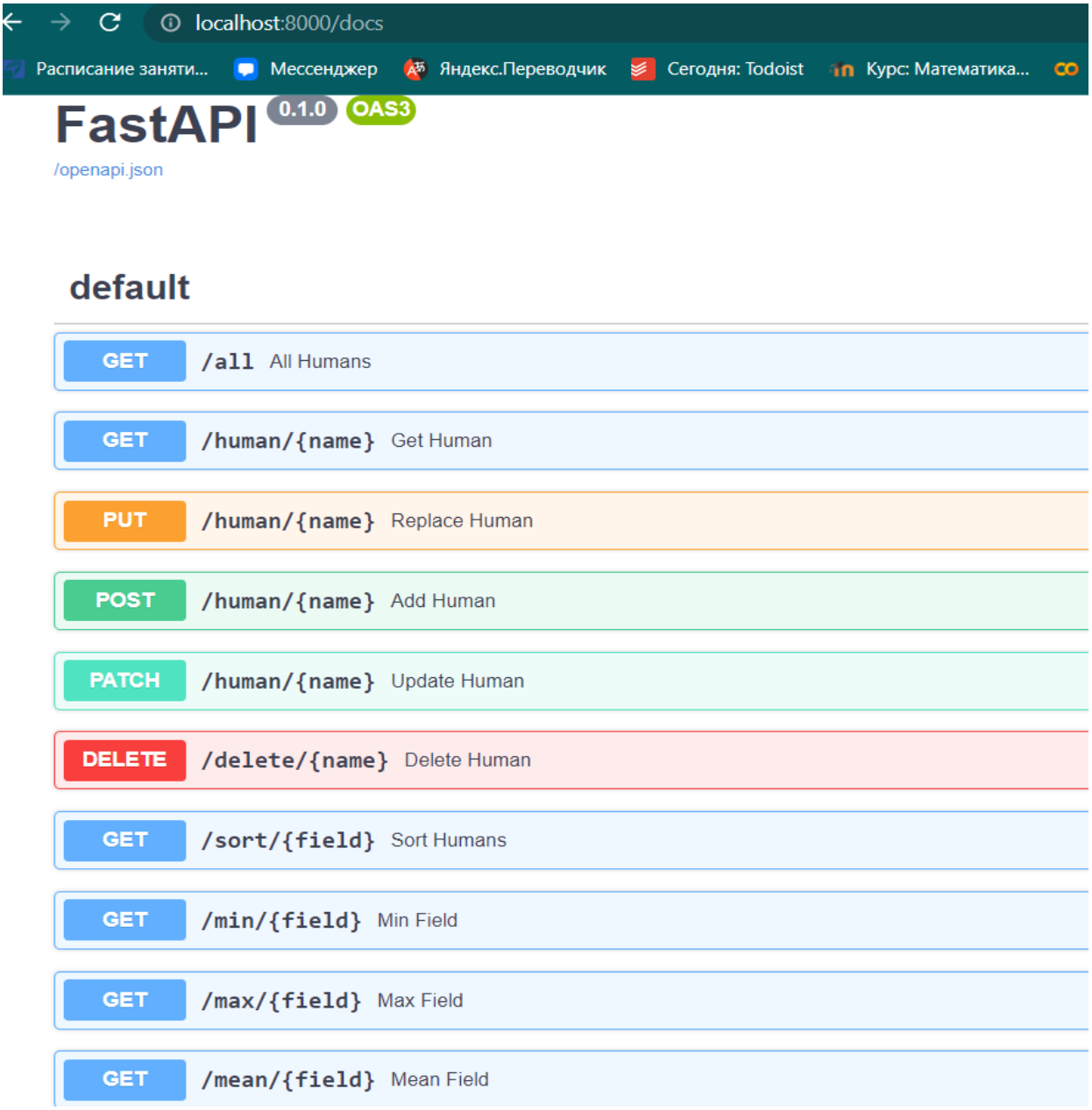

Рисунок 3.1 - страница API

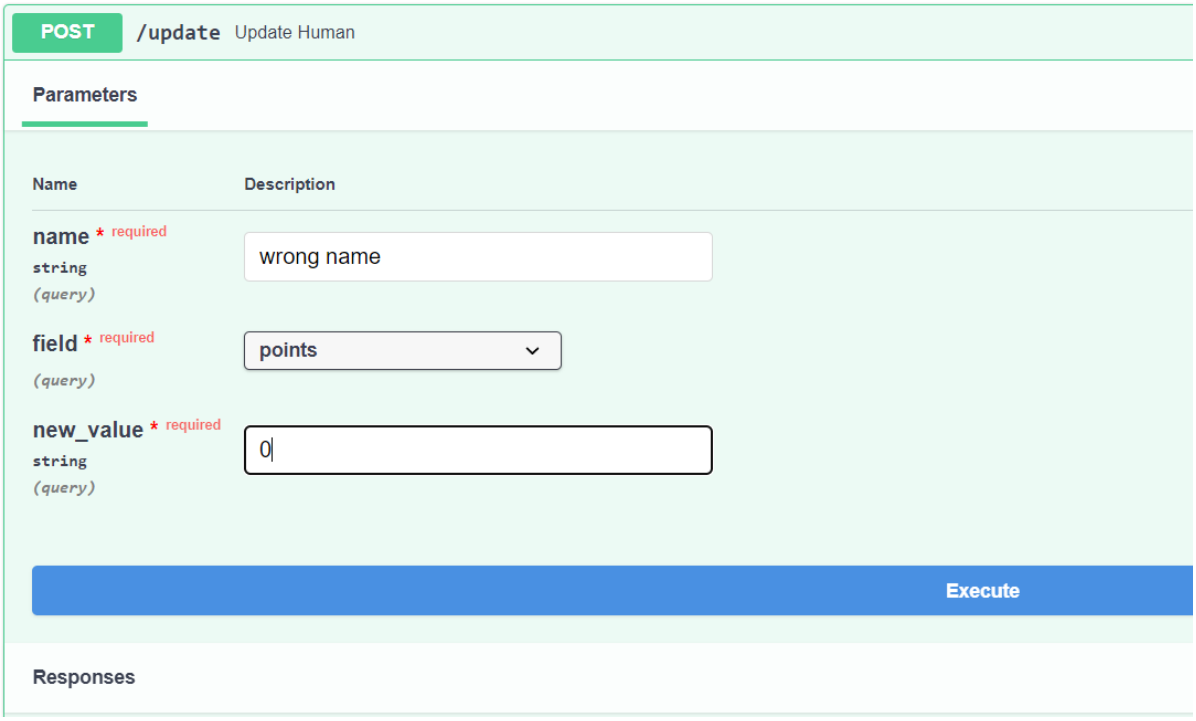

Рисунок 3.2 - пример запроса

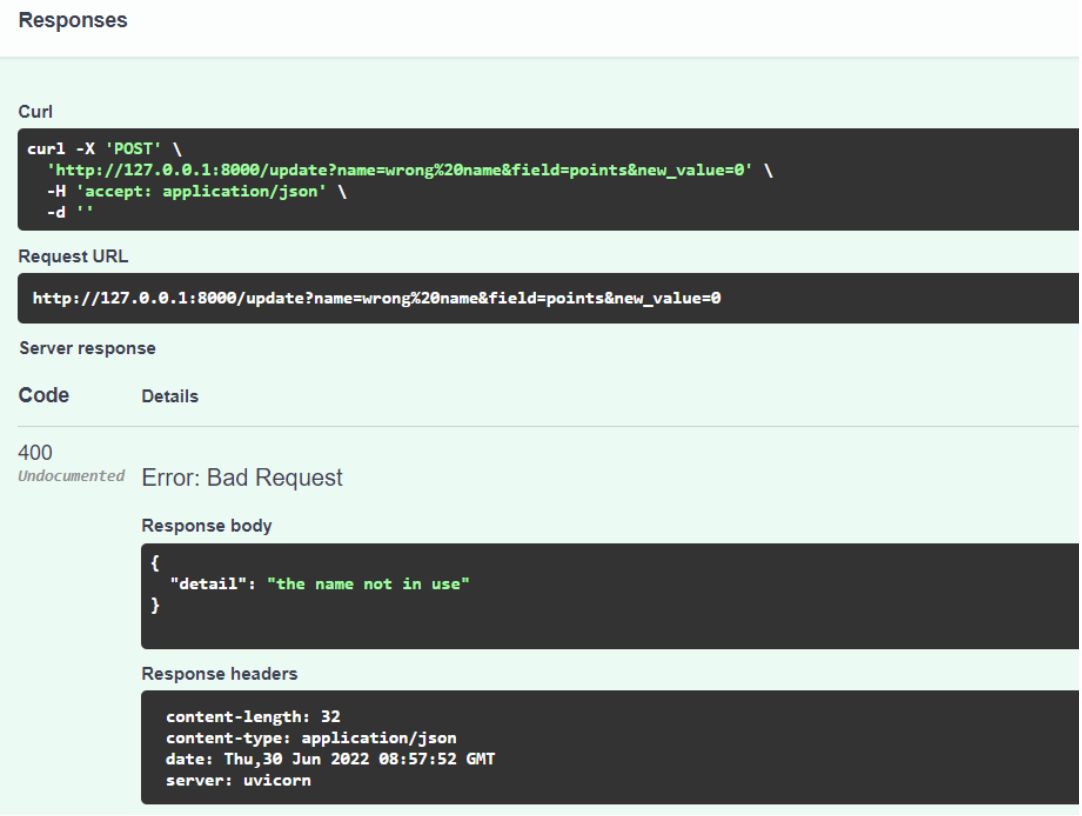

Рисунок 3.3 - пример ответа

4 Вывод

В ходе выполнения данной лабораторной работы было проведено знакомство с фреймворками Flagger, Flask-Restplus, FastAPI и получены базовые навыки документации веб-приложений и разработки API.

Также полученные навыки были применены на практике в ходе выполнения заданий.

Приложение А

Исходный код веб-приложения, созданного в соответствии с первой частью заданий методического пособия представлен по ссылке github.com/serdenvl/s2_dws_lab4 в директории manual_examples/flagger_example.

Приложение Б

Исходный код веб-приложения, созданного в соответствии с первой частью заданий методического пособия представлен по ссылке github.com/serdenvl/s2_dws_lab4 в директории manual_examples/flask_restplus_example.

Приложение В

Исходный код веб-приложения, созданного в соответствии с заданием по варианту.

Также доступен по ссылке github.com/serdenvl/s2_dws_lab3.

```
import pydantic
from fastapi import FastAPI, HTTPException


import random
from enum import Enum
from typing import Any
from dataclasses import dataclass



class NEnum(Enum):
    @staticmethod
    def _generate_next_value_(name: str, start: int, count: int, last_values: list[Any]) -> Any:
        return name

    @classmethod
    def names(cls):
        return {a.name for a in cls}



StrFields = NEnum('StrFields', 'name country category')
```

```
IntFields = NEnum('IntFields', 'points medals')
AnyFields = NEnum('AnyFields', 'name country category points medals')



class Human(pydantic.BaseModel):
    name: str
    country: str
    category: str
    points: int
    medals: int

    def get_attr_by_field(self, field: AnyFields):
        return getattr(self, field.name)

    @classmethod
    def sort_key(cls, field: AnyFields):
        def key(human: cls):
            return human.get_attr_by_field(field)
        return key



humans = {
    name: Human(
        name=name,
```

```
        country=random.choice(('country', 'COUNTRY', 'COunTRy')),
        category=random.choice(('super', 'norm', 'bad')),
        points=random.randint(0, 999),
        medals=random.randint(0, 9)
    )
    for name in "ABCDEFG"
}



app = FastAPI()



@app.get("/all")
def all_humans() -> list[Human]:
    """
    Returns all humans
    """
    return list(humans.values())



@app.get("/human/{name}")
def get_human(name: str) -> Human:
    try:
        return humans[name]
```

```
    except KeyError:
        raise HTTPException(status_code=404, detail='there is no human with that name')



@app.post("/human/{name}", status_code=201)
def add_human(name: str, human: Human) -> Human:
    human.name = name
    if name in humans:
        raise HTTPException(status_code=400, detail='there is already a human with that name')
    humans[name] = human
    return human



@app.put("/human/{name}")
def replace_human(name: str, human: Human) -> Human:
    human.name = name
    if name not in humans:
        raise HTTPException(status_code=400, detail='there is no human with that name')
    humans[name] = human
    return human



@app.patch("/human/{name}")
def update_human(name: str, field: AnyFields, new_value: str) -> Human:
```

```
    if field != StrFields.name:
        if name not in humans:
            raise HTTPException(status_code=400, detail='there is no human with that name')
    else:
        if new_value in humans:
            raise HTTPException(status_code=400, detail='there is already a human with that name')
        humans[new_value] = humans[name]
        name = new_value


    print(type(field))
    if field.name in IntFields.names():
        try:
            new_value = int(new_value)
        except ValueError:
            raise HTTPException(status_code=400, detail='wrong type of value')


    setattr(humans[name], field.name, new_value)
    return humans[name]



@app.delete("/delete/{name}")
def delete_human(name: str) -> Human:
    try:
        human = humans[name]
```

```
        del humans[name]
    except KeyError:
        raise HTTPException(status_code=404, detail='there is no human with that name')

    return human



@app.get("/sort/{field}")
def sort_humans(field: AnyFields) -> list[Human]:
    return sorted(humans.values(), key=Human.sort_key(field))



@app.get("/min/{field}")
def min_field(field: AnyFields) -> Human:
    return min(humans.values(), key=Human.sort_key(field))



@app.get("/max/{field}")
def max_field(field: AnyFields) -> Human:
    return max(humans.values(), key=Human.sort_key(field))



@app.get("/mean/{field}")
def mean_field(field: IntFields) -> float:
```

```
return sum(a.get_attr_by_field(field) for a in humans.values()) / len(humans)
```