Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
(МАИ)

Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра 805 «Прикладная математика»

КУРСОВОЙ ПРОЕКТ

ПО
ИНСТРУМЕНТАЛЬНЫМ СРЕДСТВАМ РАЗРАБОТКИ ПРИКЛАДНЫХ ПРОГРАММНЫХ СИСТЕМ

ПО ТЕМЕ «ДОКУМЕНТИРОВАНИЕ И ТЕСТИРОВАНИЕ СЕРВИСОВ С ПРИМЕНЕНИЕМ SWAGGER»

3-й семестр

Выполнила: студентка группы
М8О-230Б-21
Быкова Е. Е.
Проверил(-а): Кузнецова С.В.

Москва, 2022

Тема работы

Документирование и тестирование сервисов с применением Swagger.

Описание используемых технологий разработки

1. **Flask** - фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности. Flask Framework использует Jinja2 — приложение для обработки шаблонов — и Werkzeug — инструмент для работы с WSGI (стандартом взаимодействия между Python-программой, которая выполняется на стороне сервера, и самим веб-сервером). Для создания изолированной среды в Python используется модуль Virtualenv.

2. **Swagger** - это набор инструментов, который позволяет автоматически описывать API на основе его кода. API — интерфейс для связи между разными программными продуктами, и у каждого проекта он свой. Документация, автоматически созданная через Swagger, облегчает понимание API для компьютеров и людей. На основе кода или набора правил Swagger автоматически генерирует документацию в формате JSON-файла. Ее можно встроить на страницу сайта или в приложение, чтобы пользователи могли интерактивно знакомиться с документацией, можно отправлять клиентам — сгенерировать такое описание намного быстрее, чем написать с нуля.
   Плюсы использования Swagger
   - Swagger упрощает и ускоряет написание документации, экономит время разработчиков и технических писателей.
   - Можно сократить рутинную работу и предоставить создание шаблонного кода Swagger Codegen: это опять же экономит время.
   - Документация получается подробной и ясной для всех категорий читателей: технических специалистов, обычных пользователей и роботов.
   - Пользоваться Swagger можно довольно гибко: создавать документацию на основе кода или самостоятельно, применять возможности визуального интерфейса для тестирования и быстрой навигации.
   - Документацию, написанную с помощью Swagger, можно легко встроить в страницу сайта или в приложение, и она будет интерактивной и полностью функциональной. Но лучше все же сверстать свой вариант отображения — стандартный интерфейс не слишком хорошо вписывается в дизайны реальных сайтов.

3. **Python** - высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как C или C++.

Структура проекта

В процессе работы над проектом был использован паттерн проектирования Singleton и принципы объектно-ориентированного программирования.

Файл DB.py

```python
from Car import Car
from typing import Optional

class DB:
    __instance = None
    carsDB = list()
    def __new__(cls):
        if not hasattr(cls, 'instance'):
            cls.instance = super(DB, cls).__new__(cls)
        return cls.instance

    def create(self, car: Car):
        self.carsDB.append(car)
        return True

    def read(self, car_id: int):
        for car in self.carsDB:
            if int(car.id) == int(car_id):
                return car
        return False


    def read_all(self):
        output = list()
        if len(self.carsDB) != 0:
            for car in self.carsDB:
                output.append(car.representation_car())
            return output

        return self.carsDB
```

```python
def update(self, car_id: int, manufacturer: Optional[str] = None, model: Optional[str] = None, car_type: Optional[str] = None,
           price: Optional[int] = None):
    for car in self.carsDB:
        if int(car.id) == int(car_id):
            car.manufacturer = manufacturer if manufacturer is not None else car.manufacturer
            car.model = model if model is not None else car.model
            car.car_type = car_type if car_type is not None else car.car_type
            car.price = price if price is not None else car.price

def delete(self, car_id: int):
    for car in self.carsDB:
        if int(car.id) == int(car_id):
            self.carsDB.remove(car)
            return True
        else:
            return False
```

Файл Car.py

```python
class Car():
    def __init__(self, id: int, manufacturer: str, model: str, car_type: str, price: int):
        self.id = id

        self.manufacturer = manufacturer
        self.model = model
        self.car_type = car_type
        self.price = price

    def representation_car(self):
        return {
            "id": self.id,
            "manufacturer": self.manufacturer,
            "model": self.model,
            "car_type": self.car_type,
            "price": self.price,
        }
```

Файл app.py

```python
from flask import Flask, request
from flask_swagger_ui import get_swaggerui_blueprint
from DB import DB
from Car import Car

app = Flask(__name__)

app = Flask(__name__)

### swagger specific ###
SWAGGER_URL = '/swagger'
API_URL = '/static/swagger.json'
SWAGGERUI_BLUEPRINT = get_swaggerui_blueprint(
    SWAGGER_URL,
    API_URL,
    config={
        'app_name': "KURSACH"
    }
)
app.register_blueprint(SWAGGERUI_BLUEPRINT, url_prefix=SWAGGER_URL)

db = DB()
```

```python
@app.route("/cars", methods=['GET', 'POST'])
def cars():
    if request.method == 'GET':
        return db.read_all()

    if request.method == 'POST':
        car = Car(
            id=request.json['id'],
            manufacturer=request.json['manufacturer'],
            model=request.json['model'],
            car_type=request.json['car_type'],
            price=request.json['price']
        )

        output = db.create(car=car)
        print(db.read_all())
        return str(output)

@app.route("/car/<id>", methods=['GET', 'PUT', 'DELETE'])
def car(id):
    if request.method == 'GET':
        car = db.read(car_id=id)
        if car is not False:
            return car.representation_car()
        return 'Not found car for this id'

    if request.method == 'PUT':
```

```python
    if request.method == 'PUT':
        car = db.read(car_id=id)
        if car is not None:
            car.id = request.json['id'] if request.json.get('id') is not None else car.id
            car.manufacturer = request.json['manufacturer'] if request.json.get('manufacturer') is not None else car.manufacturer
            car.model = request.json['model'] if request.json.get('model') is not None else car.model
            car.car_type = request.json['car_type'] if request.json.get('car_type') is not None else car.car_type
            car.price = request.json['price'] if request.json.get('price') is not None else car.price

            return 'True'
        else:
            return 'Not found car for this id'

    if request.method == 'DELETE':
        output = db.delete(car_id=id)
        if output:
            return str(True)
        else:
            return 'Not found car for this id'

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0")
```

Необходимое программное обеспечение

Для разработки приложения использовалась среда разработки PyCharm Community.

Выводы

В ходе работы над курсовым проектом я изучила технологию Swagger, с помощью которого можно автоматически создавать документацию, что значительно ускоряет работу.