## ЦЕЛЬ РАБОТЫ

Целью работы является изучение средств Java для разработки web-сервисов

## ХОД РАБОТЫ

Выбран фреймворк из представленной таблицы №19 Django (Python) – доп. информация: https://www.djangoproject.com.

Технология SOAP (Simple Object Access Protocol) является протоколом для обмена структурированными сообщениями между веб-сервисами. SOAP предоставляет стандартный способ форматирования сообщений и определения их структуры. Он основан на использовании XML для кодирования данных и расширяемых пространств имен для определения схемы данных.

WSDL (Web Services Description Language) - это язык описания веб-сервисов, который используется для описания доступных операций, формата сообщений и протокола обмена данными. WSDL файл определяет контракт между клиентом и сервером, указывая, какие операции могут быть выполнены, какие параметры они принимают и возвращают, а также какой протокол использовать для обмена сообщениями.

Вместе SOAP и WSDL образуют мощную комбинацию для разработки веб-сервисов. SOAP обеспечивает стандартизацию обмена сообщениями, а WSDL предоставляет формальное описание доступных операций и структуру данных. Это позволяет клиентам и серверам обмениваться данными в единообразном формате, независимо от используемых языков программирования или платформ.

Однако в сравнении с REST API, которые используют более легковесные и простые протоколы, такие как HTTP, SOAP/WSDL являются более сложными и тяжеловесными. Они обычно используются в случаях, когда требуется строгая типизация данных, расширяемость и поддержка различных протоколов обмена, таких как HTTP, SMTP и т. д.

В целом, технологии SOAP и WSDL являются стандартами для разработки веб-сервисов, которые обеспечивают гибкость, стандартизацию и расширяемость. Они имеют свои преимущества и недостатки, и выбор между ними и REST зависит от требований проекта и контекста использования.

Лабораторная работа номер три является продолжением первой и второй. SOAP сервер основан на технологии asmx и позволяет производить основные crud операции. Использовалась библиотека zeep и spyne. Для работы с SOAP сервером содержимое запроса должно быть указано как text/xml.

Zeep - это модуль Python, который предоставляет возможности работы с веб-сервисами на основе SOAP (Simple Object Access Protocol). SOAP является протоколом обмена структурированными сообщениями в распределенных вычислительных средах. Zeep позволяет создавать клиенты для веб-сервисов, отправлять SOAP-запросы и обрабатывать SOAP-ответы. Он автоматически генерирует классы Python на основе WSDL (Web Services Description Language) - описания веб-сервиса, что упрощает взаимодействие с ним.

Spyne (ранее известный как "pywebsvcs") - это еще одна библиотека на языке Python, предназначенная для разработки веб-сервисов. Она предоставляет простой и элегантный способ создания и экспонирования веб-сервисов с использованием различных протоколов, включая SOAP и JSON-RPC. Spyne упрощает создание серверов веб-сервисов и клиентов, обеспечивая автоматическую генерацию кода из спецификаций, таких как WSDL или OpenAPI.

## ХОД РАБОТЫ:

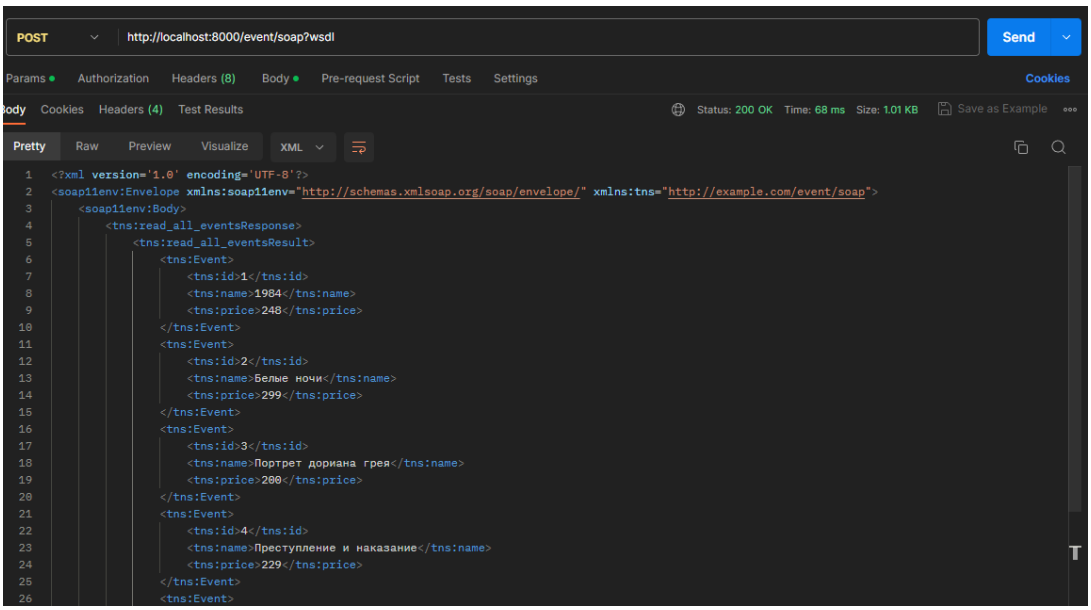


Рисунок 1 - Запрос на получение полного списка книг

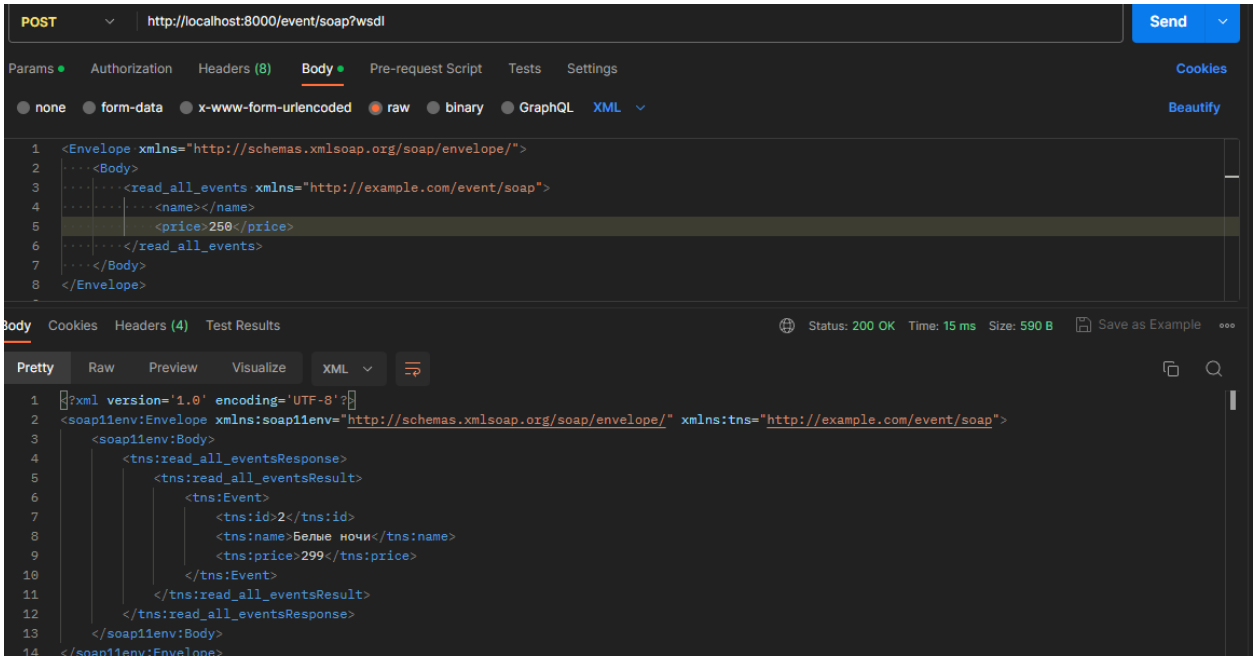


Рисунок 2 - Запрос на получение книг со стоимостью выше 250

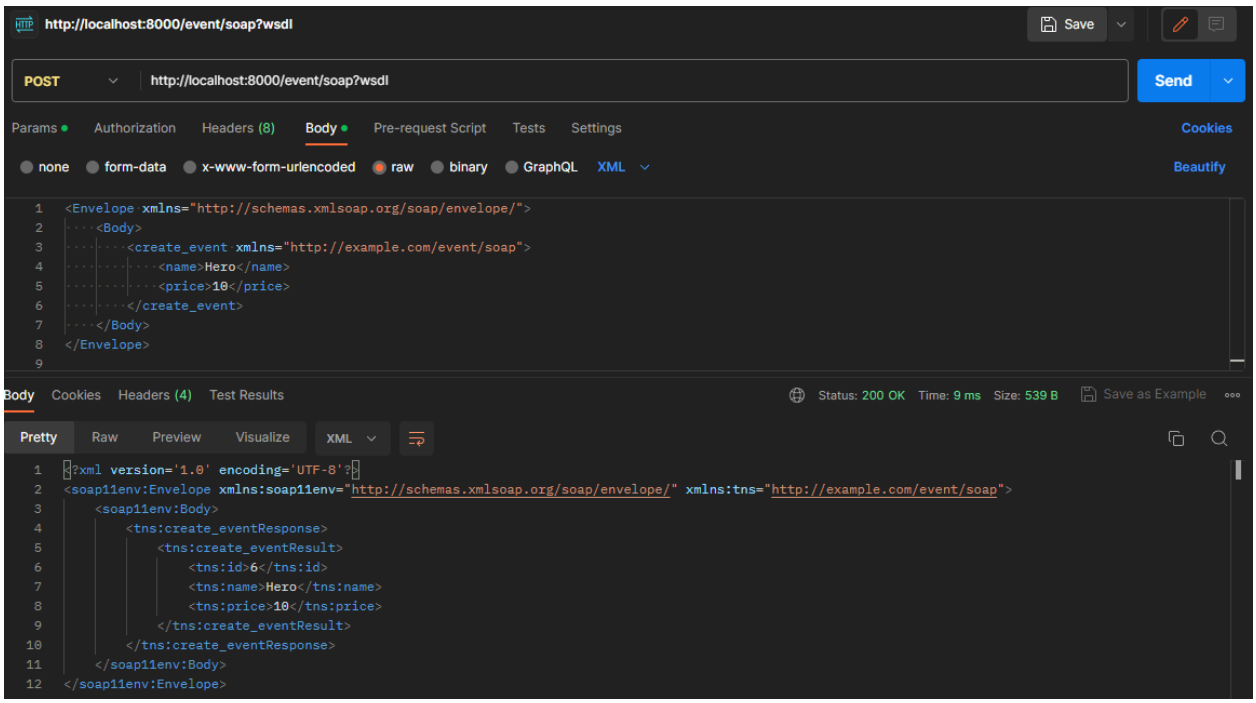


Рисунок 3 - Запрос на добавление книги

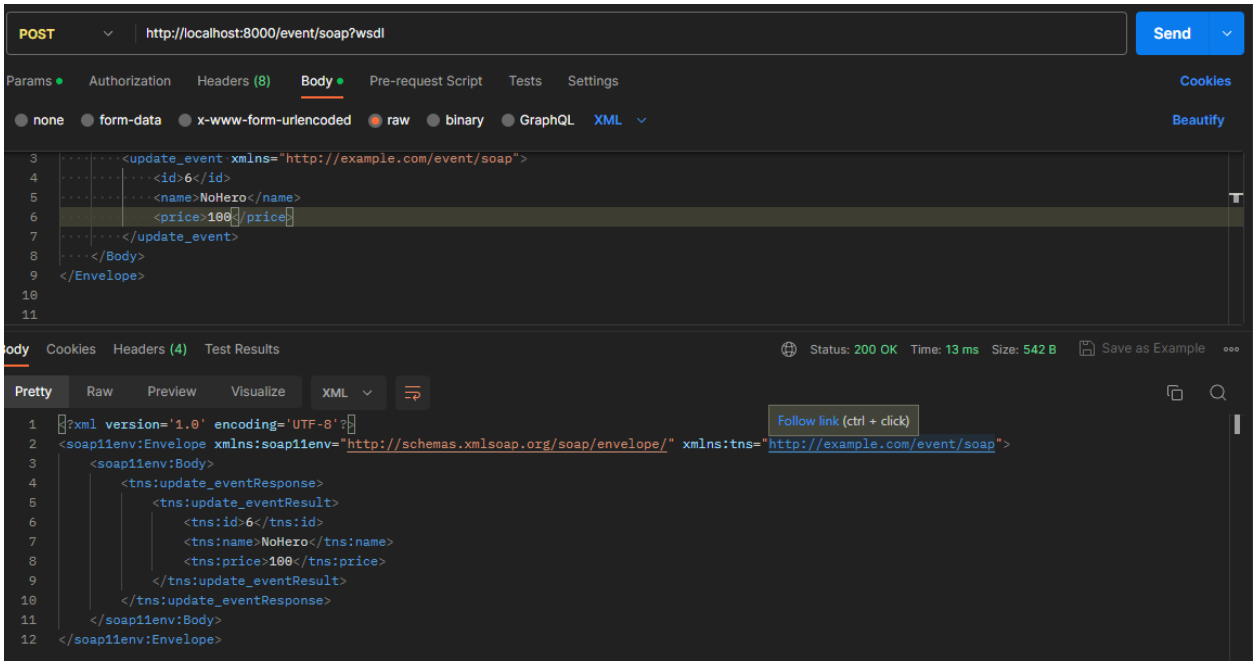


Рисунок 4 - Запрос на изменение книги с id: 6

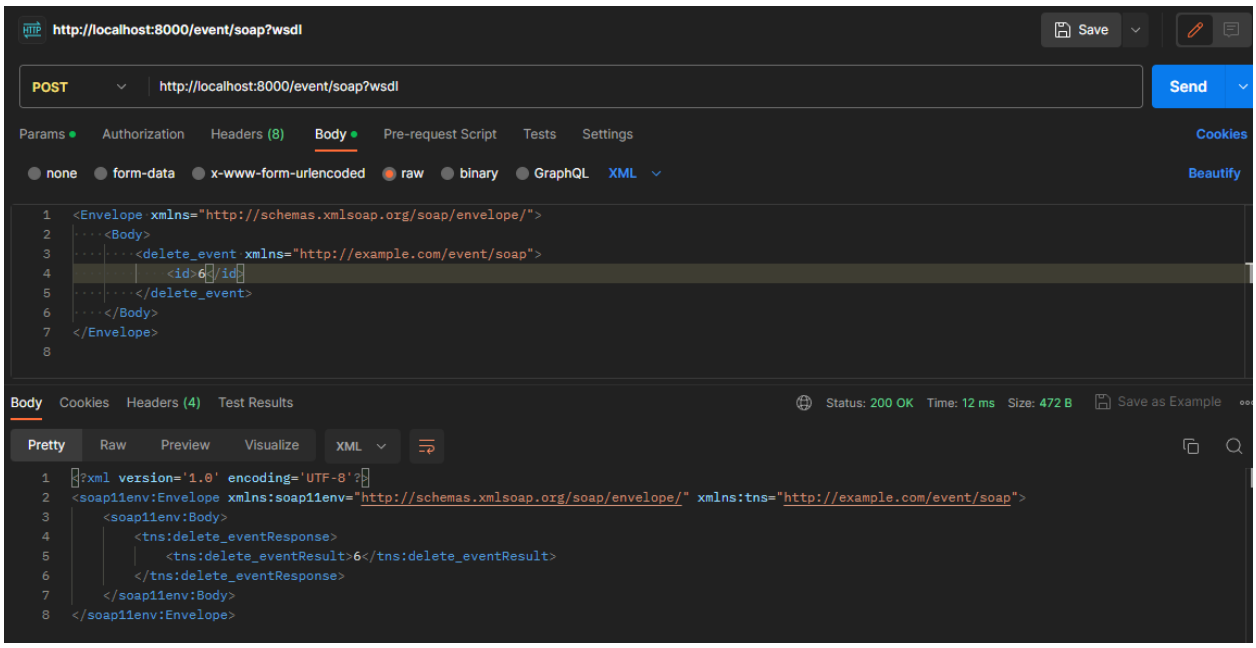


Рисунок 5 - Запрос на удаление книги

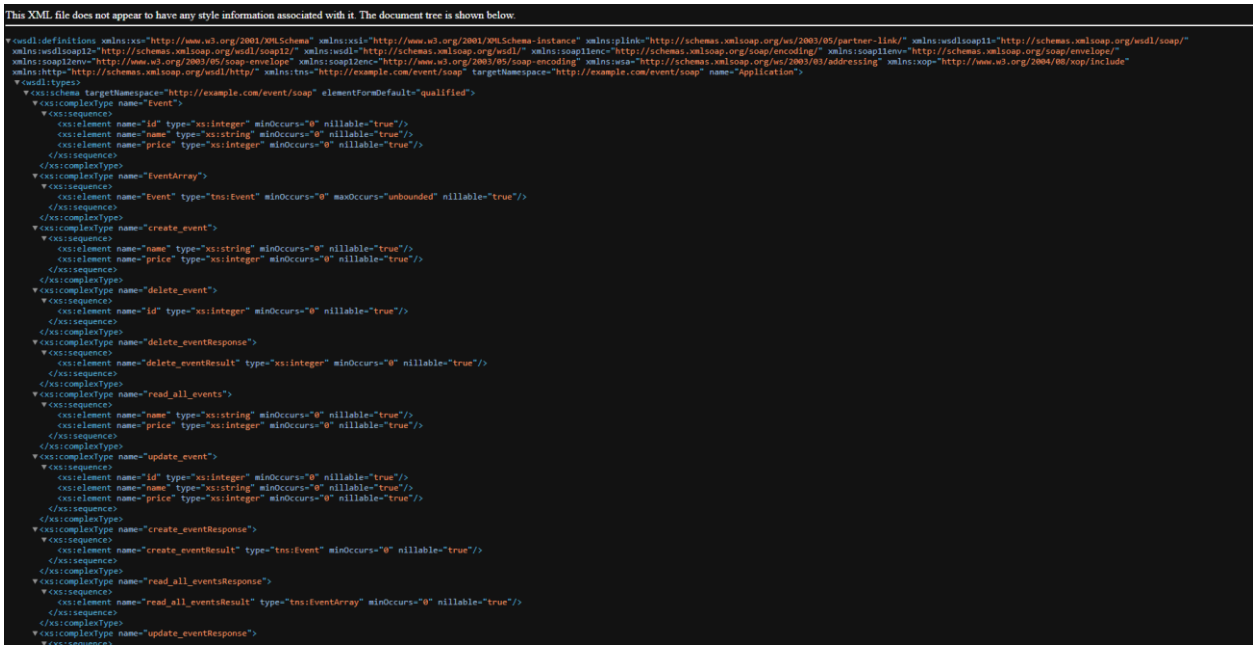


Рисунок 6 - Wsdl-файл

# ЗАКЛЮЧЕНИЕ

В заключении лабораторной работы по реализации веб-сервиса с использованием библиотек Zeep и Spyne можно отметить следующее:

Выполнение данной лабораторной работы позволило ознакомиться с основами разработки веб-сервисов на основе протокола SOAP. Я познакомился с концепцией SOAP, WSDL и типами данных, используемыми в SOAP-сообщениях.

Использование библиотеки Zeep позволило мне легко создавать, отправлять и получать SOAP-сообщения. Zeep предоставил удобные инструменты для работы с WSDL-файлами, автоматического создания клиентов и серверов веб-сервисов.

Библиотека Spyne позволила определить структуру веб-сервиса и его операции. Я создал WSDL-файл, определил тип данных и операции, что было необходимо для правильного функционирования веб-сервиса.

В ходе выполнения лабораторной работы успешно разработали веб-сервис на основе протокола SOAP с использованием Zeep и Spyne. Создали клиентскую и серверную стороны веб-сервиса, протестировали его работу, отправляя SOAP-запросы и получая SOAP-ответы.

В целом, лабораторная работа предоставила вам практический опыт в разработке веб-сервисов на основе SOAP. Я получил навыки работы с библиотеками Zeep и Spyne, а также понимание основных концепций SOAP-сервисов. Это является ценным дополнением к моим навыкам веб-разработки и может быть полезным при создании и интеграции веб-сервисов в различные приложения и системы.

# ПРИЛОЖЕНИЕ

## Листинг программы

### server.py

```
from spyne import Application, rpc, ServiceBase, Integer, Unicode, Array, ComplexModel
from spyne.protocol.soap import Soap11
from spyne.server.wsgi import WsgiApplication
# '''
# ServiceBase Определяется класс сервиса, который наследуется от ServiceBase. В этом классе определяются методы, которые будут предоставляться клиентам через веб-сервис
# Создается экземпляр класса Application с указанием сервиса и протокола
# Создается объект WsgiApplication, который инкапсулирует Application и предоставляет его через WSGI-интерфейс
#
# '''

# Модель Event
class Event(ComplexModel):
    id = Integer
    name = Unicode
    price = Integer

    def __init__(self, id, name, price):
        self.id = id
        self.name = name
        self.price = price


# База данных
events = [Event(1, '1984', 248),
          Event(2, 'Белые ночи', 299),
          Event(3, 'Портрет дориана грея', 200),
          Event(4, 'Преступление и наказание', 229),
          ]


# Сервис Event
class EventService(ServiceBase): # класс EventService, который наследуется от ServiceBase из библиотеки Spyne. Внутри класса определен метод create_event, который будет предоставлен клиентам через веб-сервис.
    # Создание события
    @rpc(Unicode, Integer, _returns=Event) # @rpc - декоратор, указывающий, что этот метод является операцией RPC для веб-сервиса. (Unicode, Integer, _returns=Event) - аннотации типов для аргументов и возвращаемого значения метода. В данном случае, метод принимает аргументы
    def create_event(ctx, name, price): #name (тип Unicode) и price (тип Integer), а возвращает объект типа Event. ctx - аргумент, представляющий контекст выполнения метода.
        new_event_id = max([event.id for event in events]) + 1
        new_event = Event(new_event_id, name, price)
        events.append(new_event)
        return new_event
# '''
# Внутри метода create_event происходит следующее:
# Определяется новый идентификатор события (new_event_id) путем нахождения максимального идентификатора среди всех событий (events) и увеличения его на единицу.
# Создается новый объект Event с помощью полученного идентификатора, имени и цены.
# Созданное событие добавляется в список events.
# Возвращается созданное событие в качестве результата.
# '''
    # Получение всех событий
    @rpc(Unicode, Integer, _returns=Array(Event))
```

#@rpc - декоратор, указывающий, что метод является операцией RPC для веб-сервиса. (Unicode, Integer, _returns=Array(Event)) - аннотации типов для аргументов и возвращаемого значения метода. Метод принимает аргументы name (тип Unicode) и price (тип Integer), и возвращает список объектов типа Event (Array(Event)).

```
        def read_all_events(ctx, name=None, price=None):
            if name or price is not None:
                filtered_events = []
                for event in events:
                    if name and name in event.name:
                        filtered_events.append(event)
                    elif price is not None and price < event.price:
                        filtered_events.append(event)
                return filtered_events
            return events
        # '''
        # Внутри метода read_all_events происходит следующее:
        # Проверяется, указаны ли аргументы name или price для фильтрации событий.
        # Если указан хотя бы один из аргументов (name или price), создается пустой список filtered_events, который будет содержать отфильтрованные события.
        # Происходит итерация по всем событиям в списке events.
        # Для каждого события проверяется, соответствует ли его имя критерию name (если указан) или цена события больше критерия price (если указан).
        # Если событие удовлетворяет критериям фильтрации, оно добавляется в список filtered_events.
        # Если был указан хотя бы один из аргументов фильтрации, возвращается список filtered_events. Иначе, возвращается список всех событий (events).
        # '''
            # Обновление события по ID
            @rpc(Integer, Unicode, Integer, _returns=Event)
            def update_event(ctx, id, name=None, price=None):
                for event in events:
                    if event.id == id:
                        if name:
                            event.name = name
                        if price:
                            event.price = price
                        return event
                return None
        # '''
        # Внутри метода update_event происходит следующее:
        # Происходит итерация по всем событиям в списке events.
        # Для каждого события проверяется, совпадает ли его идентификатор с указанным id.
        # Если идентификатор события совпадает, то проверяются указанные новые значения name и price. Если указано новое имя (name), то оно присваивается атрибуту name события. Если указана новая цена (price), то она присваивается атрибуту price события.
        # Возвращается обновленное событие.
        # Если не найдено событие с указанным идентификатором, возвращается None.
        # '''


            # Удаление события по ID
            @rpc(Integer, _returns=Integer)
            def delete_event(ctx, id):
                for i, event in enumerate(events):
                    if event.id == id:
                        del events[i]
                        return id
                return None


        # Создание приложения и добавление сервиса Event
        application = Application([EventService], 'http://example.com/event/soap',
                                  in_protocol=Soap11(validator='lxml'),
```

```
                    out_protocol=Soap11())
    # '''
    # [EventService] - список сервисов, которые будут связаны с веб-приложением. В данном случае, в списке содержится только один сервис EventService.
    # 'http://example.com/event/soap' - строка, представляющая базовый URL веб-приложения. В данном случае, указан URL 'http://example.com/event/soap'.
    # in_protocol=Soap11(validator='lxml') - указывает протокол входящих сообщений для приложения. В данном случае, используется протокол SOAP 1.1 (Soap11), а валидатором для сообщений указан 'lxml'.
    # out_protocol=Soap11() - указывает протокол исходящих сообщений для приложения. В данном случае, используется протокол SOAP 1.1 (Soap11).
    # '''
    # Запуск сервера
    if __name__ == '__main__':
        from wsgiref.simple_server import make_server

        server = make_server('localhost', 8000, WsgiApplication(application))
        server.serve_forever()

    # '''
    # Импортируется функция make_server из модуля wsgiref.simple_server.
    # Создается объект сервера с использованием функции make_server. Сервер будет слушать на локальном хосте (localhost) и порту 8000. В качестве WSGI-приложения указывается WsgiApplication(application), где application - объект Application выше.
    # Запуск сервера с помощью метода serve_forever(). Сервер будет слушать входящие запросы и обрабатывать их с использованием WSGI-приложения.
    # '''
```

## client.py

```
    from zeep import Client # Этот класс предоставляет возможности для работы с веб-сервисами на основе WSDL.
    def get_documentation(wsdl):
        import operator #Импортируется модуль operator, который используется для сортировки операций по их имени.
            for service in wsdl.services.values(): #обход всех сервисов, доступных в WSDL-схеме, через объект wsdl.services.values()
                print(str(service))
                for port in service.ports.values(): # Внутри вложенного цикла for происходит обход всех портов, доступных в текущем сервисе, через объект service.ports.values().
                    print("  " * 4, str(port)) #Выводится информация о текущем порте, добавляя отступы для наглядности.
                    print("  " * 8, "Operations:") #Выводится заголовок "Operations:", с добавлением отступа.
                    operations = sorted(port.binding._operations.values(), key=operator.attrgetter("name")) #Получаются все операции, доступные в текущем порте, и сортируются по имени операции.
                    for operation in operations: #  обход отсортированных операций и выводится информация о каждой операции с отступами для наглядности.
                        print("%s%s" % ("  " * 12, str(operation)))
                    print("") #После вывода информации о текущем порте, выводится пустая строка для создания разделителя между портами.


    if __name__ == '__main__':
        url = 'http://localhost:8000/event/soap?wsdl' #URL-адресом WSDL-сервиса
        client = Client(url)
        client.wsdl.dump()
    #В итоге, создаем объект Client с помощью указанного URL-адреса WSDL-сервиса и вызывает метод dump() для вывода информации о WSDL-схеме.
```

## Postman.md

Get - Отправить новую информацию

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <create_event xmlns="http://example.com/event/soap">
         <name>Hero</name>
         <price>10</price>
      </create_event>
   </Body>
</Envelope>
```

Delete- удаление по id

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <delete_event xmlns="http://example.com/event/soap">
         <id>[integer?]</id>
      </delete_event>
   </Body>
</Envelope>
```

Read  POST указывает список книг со стоимостью больше price

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <read_all_events xmlns="http://example.com/event/soap">
         <name>[string?]</name>
         <price>[integer?]</price>
      </read_all_events>
   </Body>
</Envelope>
```

Update – обновление информации по id

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Body>
      <update_event xmlns="http://example.com/event/soap">
         <id>[integer?]</id>
         <name>[string?]</name>
         <price>[integer?]</price>
      </update_event>
   </Body>
</Envelope>
```