Ilian Ayoub (iliay038)
Christian Habib (chrha376) Date: 2019-05-22

REST VS SOAP Modifiability

Theory

REST

REST (Representational state transfer) or REST API is an architecture that describes the standard of which computer systems communicate on the web. A client trying to, for example access information from a system using REST (also known as RESTful systems) can expect the system to follow the constraints described by the REST architecture.

One of the main properties of REST is the separation of client and server, this means that the server and client can be implemented without each other in mind. The reason this is possible is because REST uncouples the two by defining the way information is shared between them.

Another main property is statelessness which means that a client does not need to know the current activity of the server that receives a message, and the server does not need to know the activity or state of a client that sends a message.(1)

REST is resource-oriented which means that a client either requests a resource or submits a resource. (3) Resources are accessed via HTTP requests (GET, POST, DELETE ...) which get processed by the server and returns a response.(4)

SOAP

Like REST, SOAP(Simple Object Access Protocol) is an architecture for web communication that is commonly used. SOAP does not have limited communication through HTTP, but can be sent through other communication protocols. On the other hand, with SOAP the messages need to use XML as format. The WSDL(Web Service Definition Language) is a document which describes everything a client needs to know about a SOAP service.(4)

Tools

Json

Json stands for JavaScript Object Notation that transmits data using only text(strings) with a specified format. Json is widely used in client-server communication.(2)

Zeep

Zeep is a python library that provides a simple way of implementing a SOAP client.(6)

Requests

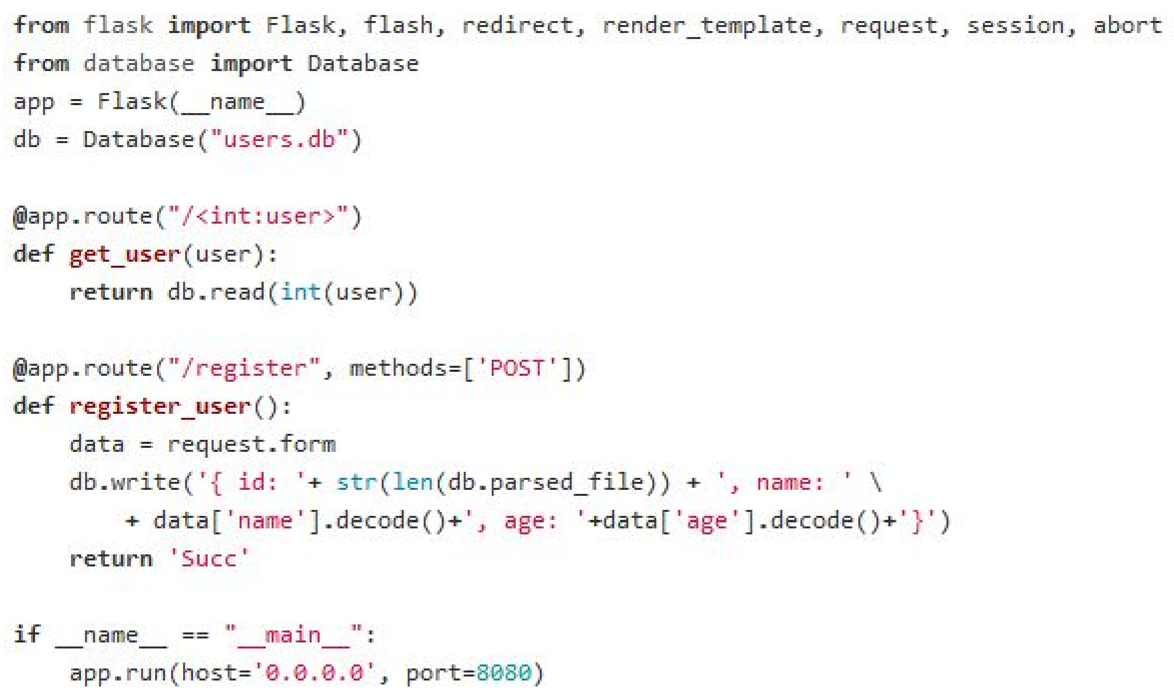
The requests library takes care of the intricate way of sending HTTP requests.(5)

Flask

Flask is a web framework for python. It's simple nature makes it popular to use.(7)

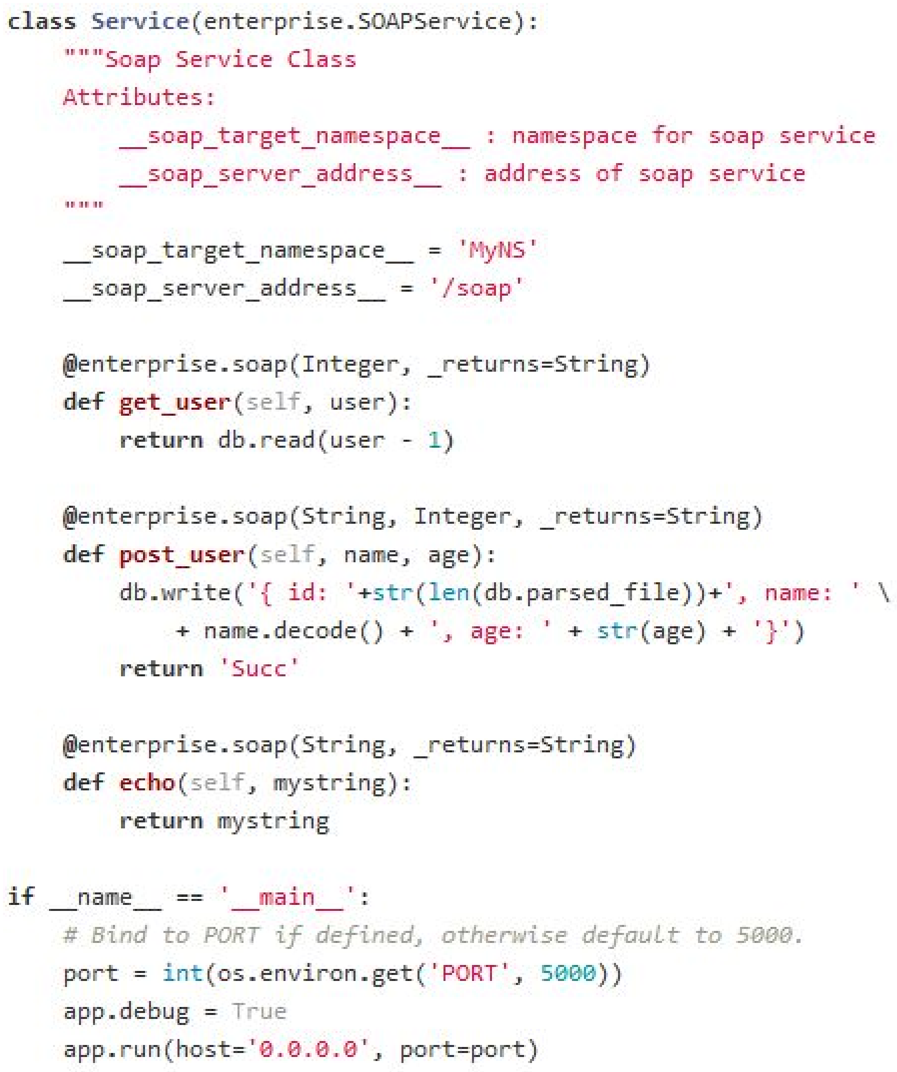
Method

We implemented two web clients,one a SOAP client and one a REST client, where both interface with respective SOAP server and REST server. Both clients where implemented in python while the SOAP client was using the zeep library and the REST client used the requests library. Both the REST and the SOAP server were implemented using the python web framework Flask as shown in the figures below.



```python
from flask import Flask, flash, redirect, render_template, request, session, abort
from database import Database
app = Flask(__name__)
db = Database("users.db")

@app.route("/<int:user>")
def get_user(user):
    return db.read(int(user))

@app.route("/register", methods=['POST'])
def register_user():
    data = request.form
    db.write('{ id: '+ str(len(db.parsed_file)) + ', name: ' \
        + data['name'].decode()+', age: '+data['age'].decode()+'}')
    return 'Succ'

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8080)
```

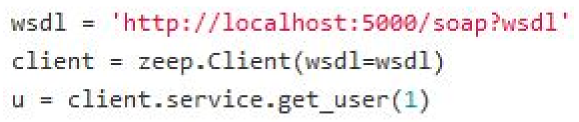
Figure 1: Implementation of REST server.



```python
class Service(enterprise.SOAPService):
    """Soap Service Class
    Attributes:
        __soap_target_namespace__ : namespace for soap service
        __soap_server_address__ : address of soap service
    """
    __soap_target_namespace__ = 'MyNS'
    __soap_server_address__ = '/soap'

    @enterprise.soap(Integer, _returns=String)
    def get_user(self, user):
        return db.read(user - 1)

    @enterprise.soap(String, Integer, _returns=String)
    def post_user(self, name, age):
        db.write('{ id: '+str(len(db.parsed_file))+', name: ' \
            + name.decode() + ', age: ' + str(age) + '}')
        return 'Succ'

    @enterprise.soap(String, _returns=String)
    def echo(self, mystring):
        return mystring

if __name__ == '__main__':
    # Bind to PORT if defined, otherwise default to 5000.
    port = int(os.environ.get('PORT', 5000))
    app.debug = True
    app.run(host='0.0.0.0', port=port)
```

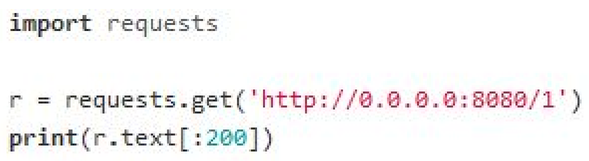
Figure 2: Implementation of SOAP server.

We implemented a get_user() and a register_user function in the SOAP server and equivalent POST and GET methods in the REST server. We modified the APIs by moving resources in the REST API and renaming functions in the SOAP API. We also removed one parameter on the server side for the register function in both API. We then studied what effects it had on the clients.



```python
wsdl = 'http://localhost:5000/soap?wsdl'
client = zeep.Client(wsdl=wsdl)
u = client.service.get_user(1)
```

Figure 3: Implementation of SOAP client.



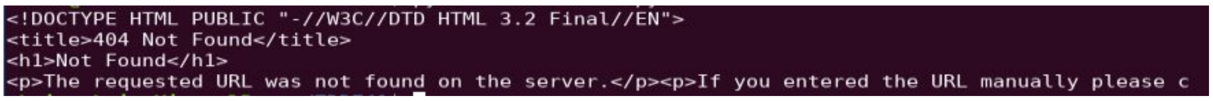
```python
import requests

r = requests.get('http://0.0.0.0:8080/1')
print(r.text[:200])
```

Figure 4: Implementation of REST client.

This study compares the two API implementations in terms of modifiability. In other words, what issues will the client have when the service API is changed and how hard is it for the client to fix these issues. The experiment will consist of changing the API and see what ramifications it will have for the clients and what it takes to fix the issues.

Results

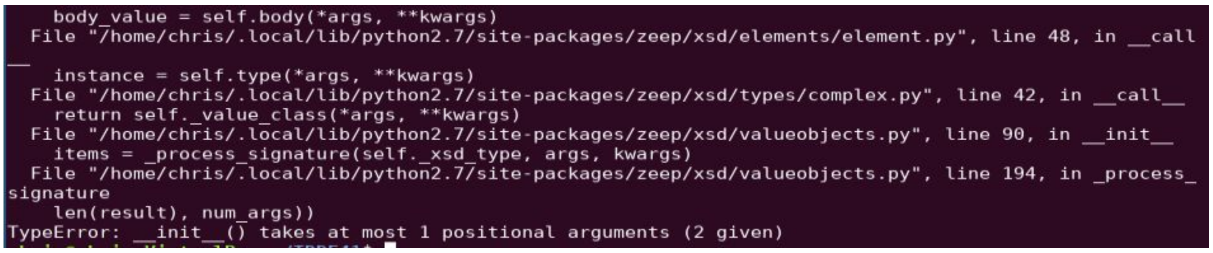
When we changed the location of a resource in the REST server the client received a 404 not found. The client had no way of remedy this because it has no information on where the resource is anymore. When we renamed the SOAP function the client could not invoke that function however the client could at least see in the WSDL file what functions are available to see what functions are available.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server.</p><p>If you entered the URL manually please c
```

Figure 5: 404 response from REST server.

When changing a POST method in the REST API by removing a parameter the client using the old API will receive a 400 bad request. When removing a parameter in SOAP we get a error message that more resembles a local error which gives us what the issue is. From the WSDL document we can see what parameters a given function takes.



```
    body_value = self.body(*args, **kwargs)
  File "/home/chris/.local/lib/python2.7/site-packages/zeep/xsd/elements/element.py", line 48, in __call
__
    instance = self.type(*args, **kwargs)
  File "/home/chris/.local/lib/python2.7/site-packages/zeep/xsd/types/complex.py", line 42, in __call__
    return self._value_class(*args, **kwargs)
  File "/home/chris/.local/lib/python2.7/site-packages/zeep/xsd/valueobjects.py", line 90, in __init__
    items = _process_signature(self._xsd_type, args, kwargs)
  File "/home/chris/.local/lib/python2.7/site-packages/zeep/xsd/valueobjects.py", line 194, in _process_
signature
    len(result), num_args))
TypeError: __init__() takes at most 1 positional arguments (2 given)
```

Figure 6: SOAP error informing us that we have one to many arguments.

Discussion

Our test focused on a bare-boned implementation of a REST and SOAP API and how a script-based client using these services would be affected by the API changing. we found that in this scenario a SOAP API would benefit the client the most.

REST relies heavily on HATEOAS to navigate the user though resources, however when the client is not a browser where users can simply click on a hyperlink browsing resources becomes difficult. Since SOAP gives you all the functions and everything you need to know about the service to be able to interface with it.

For a REST API to be useful to a client extensive documentation about the resources and where to find them is needed for the API to have value to the client. We find that in the script-based client's point of view the easiest code to maintain would be the one interfacing with a SOAP service.

References

(1) https://www.codecademy.com/articles/what-is-rest
(2) https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON
(3) https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
(4) https://www.soapui.org/learn/api/soap-vs-rest-api.html
(5) https://realpython.com/python-requests/
(6) https://python-zeep.readthedocs.io/en/master/
(7) Vogel, Patrick, et al. "A Low-Effort Analytics Platform for Visualizing Evolving Flask-Based Python Web Services." *2017 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2017