Course: Module Lab Assessment Capstone

horizontal line

Author Bio

Brian Lipp is a full-time Data Scientist, Data Engineer and Software Developer with 5+ years of experience with software development.

Project Objectives

* Reinforce fundamental python programming skills.
* Introduce skills needed for testing python programs
* Introduce working with REST API’s in python

Activity Name: Unit Testing Rest API’s

Scenario

REST API’s are ubiquitous, and python is the go-to tool for testing REST APIs. The goal of this capstone is to mimic a real-world scenario of API testing. You must write several pytest unit tests that validates an example website’s endpoints. To guarantee stability over time, several JSON files will be accessible to represent the website under test. You will combine the skills learned in the course with some new skills. The Application will use the requests package and pytest for basic unit testing. Task 1 is to create class that will access a REST API using the requests package and perform basic functionality. Task 2 is to write several Unit tests with pytest , implementing the class you created in Task 1. Task 3 is to refactor 2 of the unit tests so they are parametrized.

**Pro Tips:**

The functions and tests folder should have an empty file names \_\_init\_\_.py in each folder.

To run your test you must install and use pytest and run the command:

python -m pytest <test file>

**Best Practices to Follow:**

* Use annotations to give type hints. Python is not a strongly typed language, but with version 3 a developer can give type hints that all IDE’s will use and mimic a strongly typed language. This increases readability, and reduces errors in your code with only minor functionality being given up.

Examples:

def function(arg: <type>) -> <return\_type>:

my\_variable: <type> = <value>

* Docstrings are a very helpful tool in increasing readability of code, which can come up after its been put down for some time. When possible always try to add useful information, but do not over document.

Project Tasks

1)  **Supporting Class**

**Scenario**: In order to access external REST API’s, you must use the requests package to communicate in the HTTP protocol.

Create a class called AccessApi with 4 main components.

1. A constructor that requires the developer to input a base URL as a string that will host the REST API endpoint. Example: “http://google.com”

2. A method to get the current URL base.

3. A method to set the current URL base.

4. A method to test that the URL is responding to GET requests to allow for a simple alive test.

5. A method to input an endpoint and have that endpoint concatenated to the base url and then send a GET request using the requests package to the combined string. Then return the JSON sent as a list.

6. A method to input an endpoint and have that endpoint concatenated to the base url and then send a GET request using the requests package to the combined string. Then return the status code.

7. A method to input an endpoint and have that endpoint concatenated to the base url and then send a GET request using the requests package to the combined string. Then return the total elapsed time used for the GET request.

.

2) **Testing Process**

**Scenario**: Now that the AccessApi class has been created we must create tests for each of the API endpoints.

Create the following 5 tests for each of the api endpoints.

1. valid the HTTP status code is 200

2. valid schema matches the one provided on a very simple level. Determine that the key’s are correct.

3. Validate the accuracy of the data by randomly picking an element for all the data and do a reasonable check. For example, a SSN should be all int’s, in a format of XXX-XX-XXXX.

4. validate that the response time should be less than one minute.

Endpoints:

• Billing

• Sites

• Customers

Resource: for now, please use: https://github.com/bclipp/APItesting

A simple pytest looks like:

def <test\_name>(<args>):

<setup code for test>

assert <an expression that tests and has a True or False outcome>

a simple example is:

def testing\_my\_class(age:int):

assert age <= 50

3) **Refactoring our tests**

**Scenario**: We now have a testing platform and some useful tests for our company’s website. The tests we have done are not maintainable and hard not very pythonic. The next task is to now refactor two of our tests, so they use the parametrize functionality of pytest.

Example:

import pytest

@pytest.mark.parametrize("<first\_input>,<second\_input>,", [

("String1", true),

("String2", false),

("String3",true),

])

def test\_eval((<first\_input>,<second\_input>):

….

Tests to Refactor:

1. valid the HTTP status code is 200

2. Validate that the response time should be less than one minute

Project Activity Solution

[…] Project Scaffolding:

Folder Structure:

MyApplication

├── requirements.txt

├── functions

├── \_\_init\_\_.py

│ └── AccessApi.py

└── tests

├── \_\_init\_\_.py

└── test\_company\_api.py

**requirements.txt:**

pytest

requests

**AccessApi.py:**

import requests  
import json  
  
  
class AccessApi:  
 *"""  
 Class AccessApi is used to abstract lower level access to course required API  
  
 Attributes  
 ----------  
 url : str  
 A valid website used to hold the courses json filesS  
  
 Methods  
 -------  
 url\_active()  
 returns True if the url is currently responding without errors, and False if not.  
  
 get\_end\_point(endpoint)  
 returns the json output of the GET request  
  
 """* def \_\_init\_\_(self, url):  
 *"""  
 Parameters  
 ----------  
 url: str  
 a valid website forexample: http://google.com  
 """* self.url: str = url  
  
 @property  
 def url(self) -> str:  
 return self.\_\_url  
  
 @url.setter  
 def url(self, url: str):  
 self.\_\_url: str = url  
  
 def url\_active(self) -> bool:  
 response: requests.models.Response = requests.get(self.\_\_url)  
 if response:  
 return True  
 else:  
 return False  
  
 def get\_end\_point(self, end\_point:str) -> dict:  
 *"""  
 Parameters  
 ----------  
 end\_point: str  
 a valid endpoint on a website "api/sites/master.json"  
 """* response: requests.models.Response = requests.get(self.\_\_url + end\_point)  
 content: dict = json.loads(response.text)[0]  
 return content  
  
 def get\_status\_code(self, end\_point:str) -> int:  
 *"""  
 Parameters  
 ----------  
 end\_point: str  
 a valid endpoint on a website "api/sites/master.json"  
 """* response: requests.models.Response = requests.get(self.\_\_url + end\_point)  
 status\_code: int = response.status\_code  
 return status\_code  
  
 def get\_elapsed\_time(self, end\_point:str) -> float:  
 *"""  
 Parameters  
 ----------  
 end\_point: str  
 a valid endpoint on a website "api/sites/master.json"  
 """* response: requests.models.Response = requests.get(self.\_\_url + end\_point)  
 elapsed\_time: float = response.elapsed.total\_seconds()  
 return elapsed\_time

**test\_company\_api.py:**

from functions import AccessApi as mws  
import pytest  
  
base\_url: str = "https://raw.githubusercontent.com/.."  
billing\_end\_point: str = "getBillingInfo.json"  
customer\_end\_point: str = " getCustomers.json"  
site\_end\_point: str = "getSites.json"  
  
# TASK 2  
  
# billing  
def test\_billing\_status\_code():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 status\_code: int = api.get\_status\_code(billing\_end\_point)  
 assert status\_code == 200  
  
  
def test\_billing\_validate\_schema():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 billing: dict = api.get\_end\_point(billing\_end\_point)  
 billing\_schema : list = ['id', 'FirstName', 'LastName', 'city', 'state', 'Lang', 'SSN']  
 assert billing\_schema == list(billing.keys())  
  
  
def test\_billing\_validate\_ssn():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 billing: dict = api.get\_end\_point(billing\_end\_point)  
 chunk: list = billing["SSN"].split('-')  
 is\_ssn: bool = False  
 if len(chunk) ==3:  
 if len(chunk[0]) == 3 and len(chunk[1]) == 2 and len(chunk[2]) == 4:  
 is\_ssn: bool = True  
 assert is\_ssn == True  
  
  
def test\_billing\_validate\_time():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 elapsed\_time: float = api.get\_elapsed\_time(billing\_end\_point)  
 assert elapsed\_time <= 3  
  
  
# customers  
  
  
def test\_customers\_status\_code():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 status\_code: int = api.get\_status\_code(customer\_end\_point)  
 assert status\_code == 200  
  
  
def test\_customers\_validate\_schema():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 customers: dict = api.get\_end\_point(customer\_end\_point)  
 customers\_schema: list = ['id', 'first\_name', 'last\_name', 'email', 'ip\_address', 'address']  
 assert customers\_schema == list(customers.keys())  
  
  
def test\_customers\_validate\_ssn():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 customers: dict = api.get\_end\_point(customer\_end\_point)  
 chunk: list = customers["ip\_address"].split('.')  
 is\_ip\_address: bool = False  
 if len(chunk) == 4:  
 if len(chunk[0]) == 3 and len(chunk[1]) == 3 and len(chunk[2] ) == 3 and len(chunk[3] ) == 3:  
 is\_ip\_address: bool = True  
 assert is\_ip\_address == True  
  
  
def test\_customers\_validate\_time():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 elapsed\_time: float = api.get\_elapsed\_time(customer\_end\_point)  
 assert elapsed\_time <= 3  
  
  
# site  
  
def test\_site\_status\_code():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 status\_code: int = api.get\_status\_code(site\_end\_point)  
 assert status\_code == 200  
  
  
def test\_site\_validate\_schema():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 site: dict = api.get\_end\_point(site\_end\_point)  
 site\_schema: list = ['id', 'address', 'ThirdParty', 'admin']  
 assert site\_schema == list(site.keys())  
  
  
def test\_site\_validate\_ssn():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 site: dict = api.get\_end\_point(site\_end\_point)  
 id: int = site["id"]  
 assert isinstance(id, int)  
  
  
def test\_site\_validate\_time():  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 elapsed\_time: float = api.get\_elapsed\_time(site\_end\_point)  
 assert elapsed\_time <= 3  
  
   
# task 3  
  
  
@pytest.mark.parametrize('base\_url', [base\_url])  
@pytest.mark.parametrize('billing\_end\_point', [billing\_end\_point, customer\_end\_point, site\_end\_point])  
def test\_billing\_status\_code(base\_url, billing\_end\_point):  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 status\_code: int = api.get\_status\_code(billing\_end\_point)  
 assert status\_code == 200  
  
  
@pytest.mark.parametrize('base\_url', [base\_url])  
@pytest.mark.parametrize('billing\_end\_point',[billing\_end\_point,customer\_end\_point,site\_end\_point])  
def test\_billing\_validate\_time(base\_url,billing\_end\_point):  
 api: mws.AccessApi = mws.AccessApi(base\_url)  
 elapsed\_time: float = api.get\_elapsed\_time(billing\_end\_point)  
 assert elapsed\_time <= 3