Python- Rest API integrations test

Description: Use pytest to test RestAPI test

YT Resource: https://www.youtube.com/watch?v=7dgQRVqF1N0

What is a RESTAPI?

It's an endpoint that allow user to call it and test it. You send a HTTP request and response back.

Prerequisite:

Python Library: requests, pytest

Endpoint: https://todo.pixegami.io

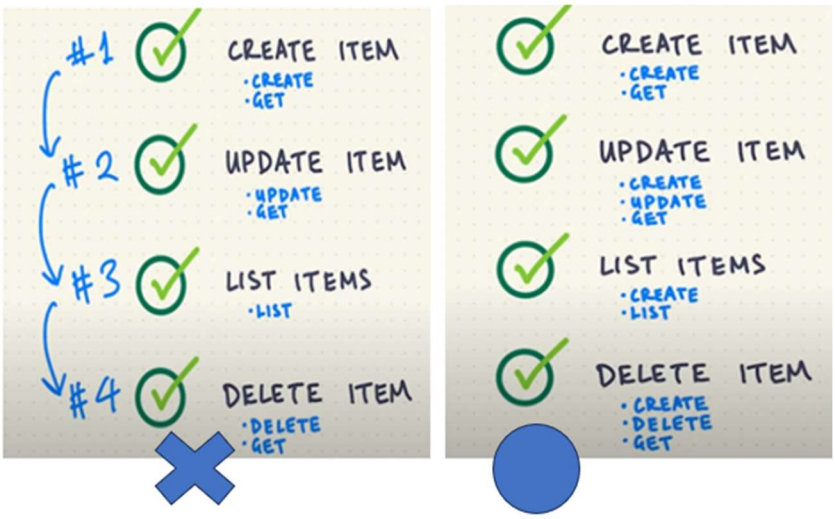
Test Case:

Case1: create item

Case2: update item

Case3: list items

Case4: delete item



You can **reuse across the testcase**, like if I have in test case1, in test case2 you can use testcase1, but **you shouldn't do that**. In this code all of the test will be test **independent**, so you can run individual test cases on their own without effect other condition. It will be easier to debug and many other reason.

Endpoint URL

API URL	
https://todo.pixegami.io	https://todo.pixegami.io/docs
{"message":"Hello World from Todo API"}	

Basic python syntax

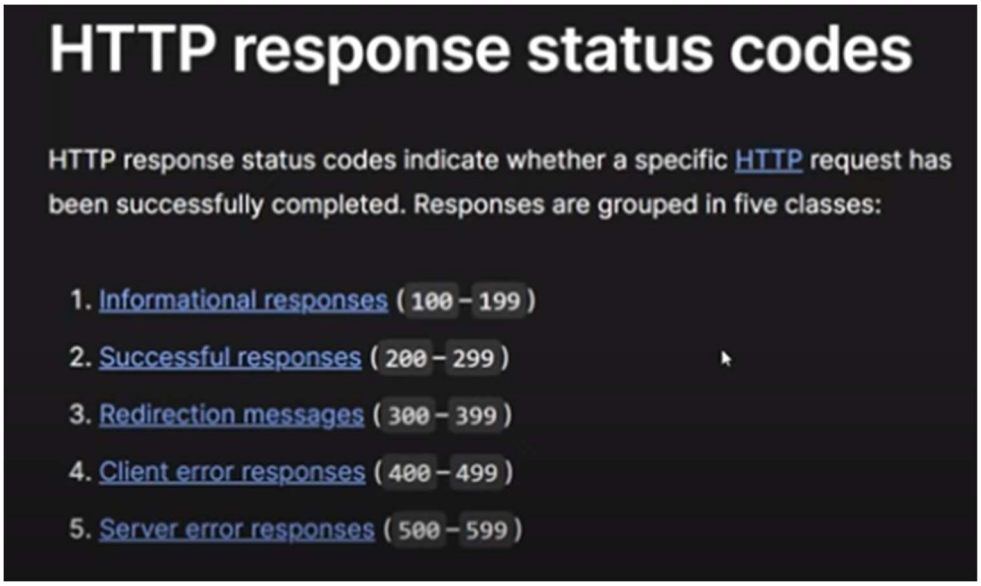
```
import requests
ENDPOINT= https://todo.pixegami.io/
response=requests.get(ENDPOINT)
print(response) # <Response [200]>
```

- Print browser's message :

```
#show website content
data= response.json()
print(data) #{'message': 'Hello World from Todo API'}
```

- Statuscode is important to check server up or down

```
status_code= response.status_code
print(status_code) #200
```



Case0 understand common code

Let add in to a function and check weather server is on or not, this is a good practice for sanity test:

```
import requests
ENDPOINT= "https://todo.pixegami.io/"

def test_can_Call_endpoint():
    response=requests.get(ENDPOINT)
    assert response.status_code == 200
```

How to run: pytest

You can run pytest to see if it work or not, If you run pytest with error even if you already install it, then you can run like this: python -m pytest

Run test case: pytest <test.py> -v -s

-v: verbose will show detail

If you run without -v option, then nothing will display, but if you add -v flag it will display result

```
(.venv) PS C:\pytest_apitest> pytest .\test_todo_api_draft.py
========================================== test session starts ==========
platform win32 -- Python 3.10.0, pytest-8.2.2, pluggy-1.5.0
rootdir: C:\pytest_apitest
collected 1 item

test_todo_api_draft.py .
```

No flag, nothing display

```
========================================== 1 passed in 1.76s ==========
(.venv) PS C:\pytest_apitest> pytest .\test_todo_api_draft.py -v
========================================== test session starts ==========
platform win32 -- Python 3.10.0, pytest-8.2.2, pluggy-1.5.0 -- C:\pytest_api
cachedir: .pytest_cache
rootdir: C:\pytest_apitest
collected 1 item

test_todo_api_draft.py::test_can_Call_endpoint PASSED

========================================== 1 passed in 1.62s ==========
```

-v flag
will display result

Pytest command

Run command will run all test file: py -m pytest or pytest

Run command with verbose more detail : py -m pytest code.py –v –s

Run command specific testcase: py -m pytest code.py –v –s :: testcase

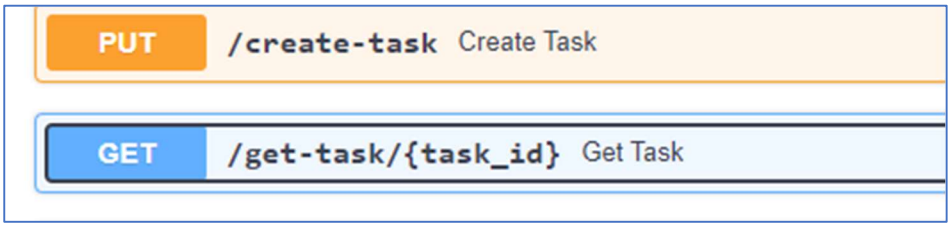
Run command: pytest -v -s <.\test_pythonfile>::<functiontest>

Case1: Create item

We will test two step:

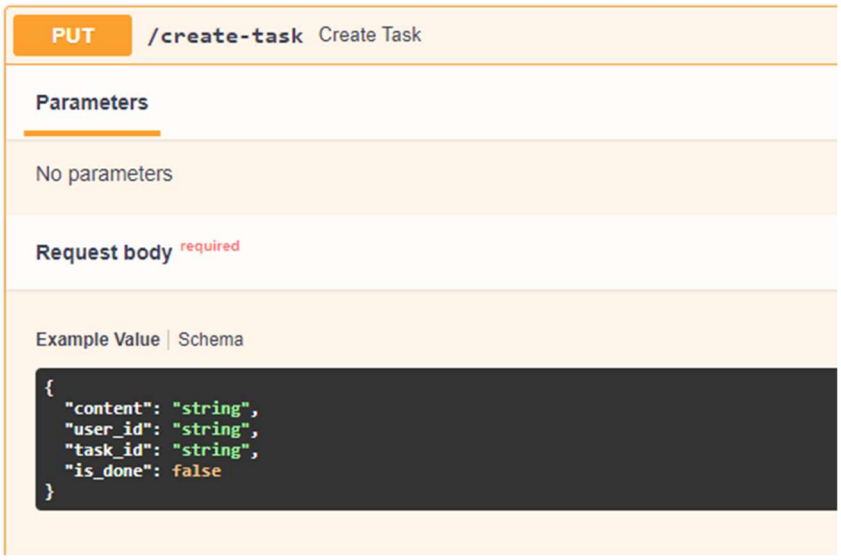
Create task

Get the task



Step1: create task

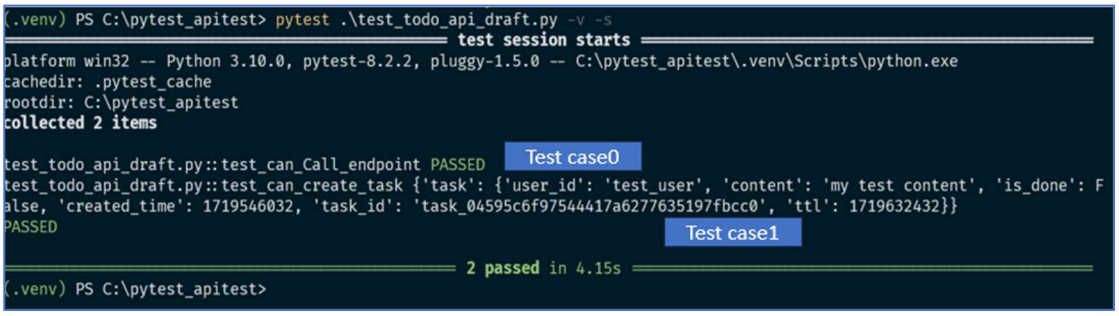
Copy the put's schema to create task




```
#CASE1
def test_can_create_task():
    payload={
        "content": "my test content",
        "user_id": "test_user",
        ~~"task_id": "task_task_id",~~
        "is_done": False,
    }
    create_task_response = requests.put(ENDPOINT + "/create-task",
json=payload)
    assert create_task_response.status_code == 200
    data=create_task_response.json()
    print(data)
```

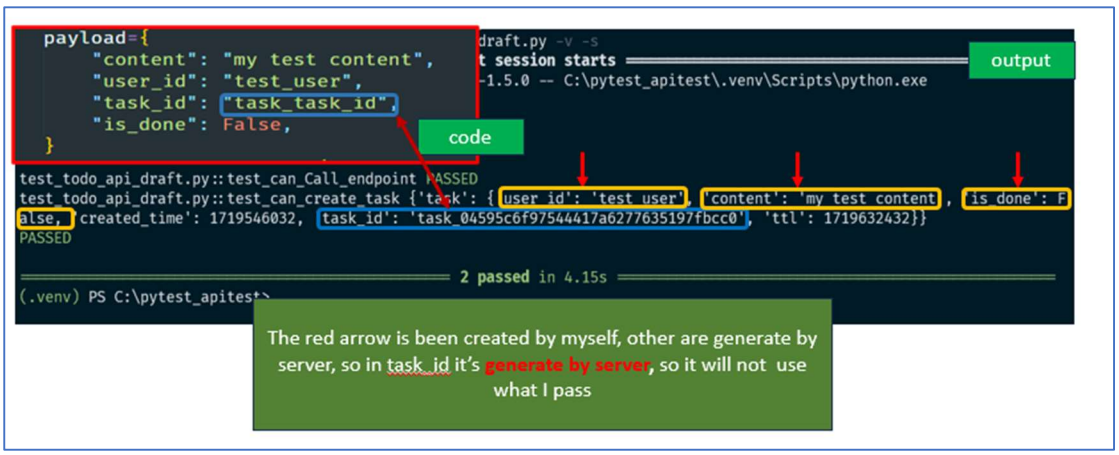
When you print will not display anything only result, so we need to add -s flag which will display output.

Command to run: pytest .\test_todo_api.py -v -s



As you can see the comparison if I create item in payload, it will pass in , and rest key

and value will use default generate by server. But the task_id will not update because this key is generated by server, so we can remove it (task_id).



Step2: get the task_id to get the task you created

We need top get the task to make sure we create success.

We need to get the **task_ID** first which is **API endpoint you're calling** (/get-task/{task_id}.

You will need to parse the data to get the task_id key like this:

task_id=data["task"]["task_id"] below is a diagram of output of getting the key

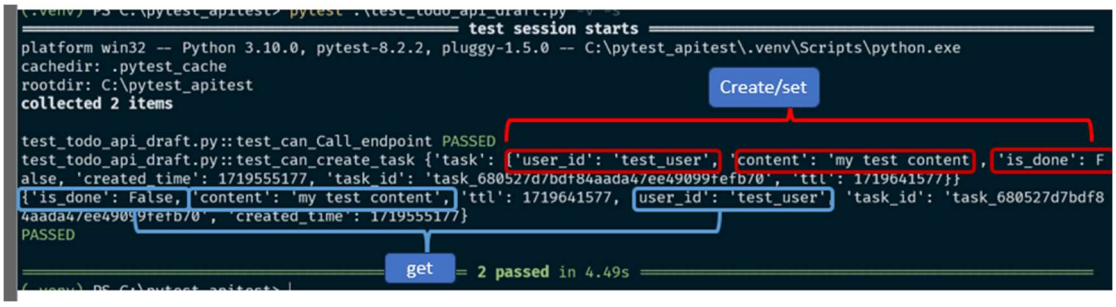

```
#CASE1
def test_can_create_task():
    #Step1
```

```
    payload={
        "content": "my test content",
        "user_id": "test_user",
        "is_done": False,
    }
    create_task_response = requests.put(ENDPOINT + "/create-task",
json=payload)
    assert create_task_response.status_code == 200
    data=create_task_response.json()
    print(data)

    #step2
    #get task_id
    task_id=data["task"]["task_id"]
    get_task_response = requests.get(ENDPOINT + f"/get-task/{task_id}")
    assert get_task_response.status_code==200
    get_task_data=get_task_response.json() #get value
    print(get_task_data)
```

Note and Description:

Let print the data and compare the set and get which will look like this:



Step3 check set and get the key and value to see if they are same or different

In above we print the whole data, let just compare user_id and content key and check the value are same or not.

Check data same or not

```
    #Step3 check data
    assert get_task_data["content"]==payload["content"]
    #assert get_task_data["user_id"]=="other content " #let it fail
    assert get_task_data["user_id"]==payload["user_id"
```

```
platform win32 -- Python 3.10.0, pytest-8.2.2, pluggy-1.5.0 -- C:\pytest_apitest\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\pytest_apitest
collected 2 items

test_todo_api_draft.py::test_can_Call_endpoint PASSED
test_todo_api_draft.py::test_can_create_task PASSED
```

Let make it fail to see will it will fail

```
    #Step3 check data
    #assert get_task_data["content"]==payload["content"]
    assert get_task_data["user_id"]=="other content " #let it fail
    assert get_task_data["user_id"]==payload["user_id"
```

```
27      get_task_data=get_task_reponse.json()
28      #   assert get_task_data["content"]==payload["content"]
29      assert get_task_data["user_id"]=="other content " #let it fail
30      assert get_task_data["user_id"]==payload["user_id"]
31    # print(get_task_data)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

        #copy the json code and paste below
        payload= {
            "content": "test content",
            "user_id": "test_user",
            "is_done": False,
      }
        create_task_reponse=requests.put(ENDPOINT + "/create-task", json=payload)
        assert create_task_reponse.status_code == 200
        data = create_task_reponse.json()


        print(data)

        task_id=data["task"]["task_id"]
        get_task_reponse=requests.get(ENDPOINT + f"/get-task/{task_id}")
        assert get_task_reponse.status_code == 200
        get_task_data=get_task_reponse.json()
        #   assert get_task_data["content"]==payload["content"]
>       assert get_task_data["user_id"]=="other content " #let it fail
E       AssertionError: assert 'test_user' == 'other content '
E
E         - other content
E         + test_user

test_tofo_api.py:29: AssertionError
========================================================================== short
FAILED test_tofo_api.py::test_can_create_task - AssertionError: assert 'test_user' == 'othe
========================================================================== 1 fail
```

As you can it tell you why it fail, the assert compare not match problem.

Full Code for test case1

```
def test_can_create_task():
    #Step1
    payload={
        "content": "my test content",
        "user_id": "test_user",
        "is_done": False,
    }
    create_task_response = requests.put(ENDPOINT + "/create-task",
json=payload)
```

```
    assert create_task_response.status_code == 200
    data=create_task_response.json()
    #print(data)

    #step2
    #get task_id
    task_id=data["task"]["task_id"]
    get_task_response = requests.get(ENDPOINT + f"/get-task/{task_id}")
    assert get_task_response.status_code==200
    get_task_data=get_task_response.json() #get value

    #Step3 check set and get match
    assert get_task_data["content"]==payload["content"]
    # assert get_task_data["user_id"]=="other content " #let it fail
    assert get_task_data["user_id"]==payload["user_id"]
    #print(data["task"]["task_id"])
```

Helper code and refactor code

Create some helper function, because most of the code will be use so creating helper function all test case can be use. So in Case1 the code will also be change.

Step1. Add helper function

```
def create_task(payload):
    return requests.put(ENDPOINT + "/create-task", json=payload)

def get_task(task_id):
    return requests.get(ENDPOINT + f"/get-task/{task_id}")
def new_task_payload():
    return   {
        "content": "my test content",
       "user_id": "test_user",
        "is_done": False,
  }
```

Step2 Refactor the code in case1

original	refactor
#Step1	*payload=new_task_payload()*

payload={
"content": "my test content",
"user_id": "test_user",
"is_done": False,
}	

create_task_response = requests.put(ENDPOINT + "/create-task", json=payload)	*create_task_reponse=create_task(payload)*

#step2
get_task_response = requests.get(ENDPOINT + f"/get-task/{task_id}")
	get_task_response=get_task(task_id)

Case2 update items

We will test three step:

- Create task
- Update a task
- Get the task

Step1: add helper function

```
def update_task(payload):
    return requests.put(ENDPOINT + "/update-task", json=payload)
```

Step2 create a task

```
#CASE2
def test_can_update_task():
    #create a task
    payload=new_task_payload()
    assert create_task_response.status_code==200
    create_task_response=create_task(payload)
    task_id=create_task_response.json()["task"]["task_id"]
```

Step3 update new payroll

```
    #update the task
    new_payload={
```

```
        "user_id":payload["user_id"],
        "task_id": task_id,
        "content":"my update content",
        "is_done": True,
        }
    update_task_response=update_task(new_payload)
    assert update_task_response.status_code==200
```

Step4 validate and check is been update new payload

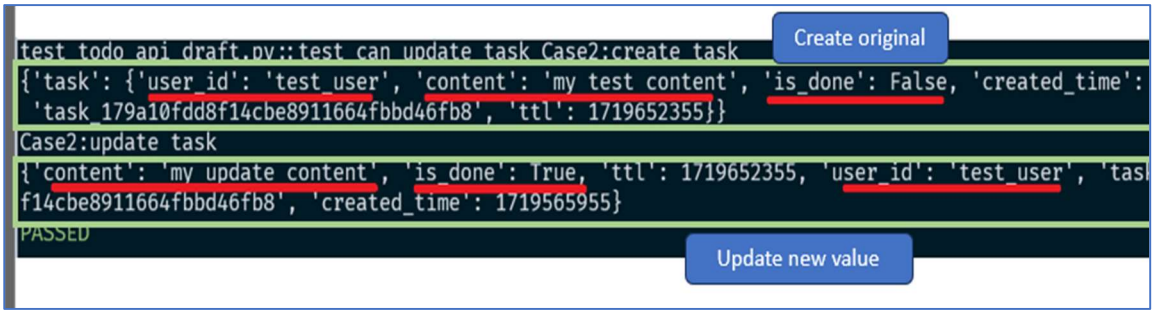
Continue with previous function

```
    get_task_response=get_task(task_id)
    assert get_task_response.status_code == 200
    get_task_data=get_task_response.json()
    assert get_task_data["content"]==new_payload["content"]
    assert get_task_data["is_done"]==new_payload["is_done"]
```

Let me make comparison when we create payroll

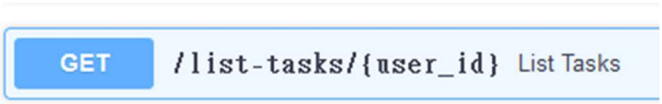
payload we create in step2	Update new payload in step3

payload={
"content": "my test content",
"user_id": "test_user",
"is_done": False,
}
	new_payload={
	"user_id":payload["user_id"],
	"task_id": task_id,
	"content":"my update content",
	"is_done": True,
	}



Case3 list items

List the task



In this test we will check the amount of task is same as what we created.

We will test three step:

- Create N task
- list tasks and check that there are N items

Step1 create test case for list task

```
def test_can_list_tasks():
    #create N tasks
     for _ in range(3):
            payload = new_task_payload()
            create_task_response=create_task(payload)
            assert create_task_response.status_code==200
     pass
```

Step2 Create helper function for list_tasks

```
def list_tasks(user_id):
     return requests.get(ENDPOINT + f"/list-tasks/{user_id}")
```

Step3 Go back to step1 again and add the list_tasks and print the data

```
def test_can_list_tasks():
    #create N tasks
     for _ in range(3):
            payload = new_task_payload()
            create_task_response=create_task(payload)
            assert create_task_response.status_code==200
     list_task_response = list_tasks("test_user")
     assert list_task_response.status_code==200
     data= list_task_response.json()
     print(data)
```

When you print the data it will occur 10 item (picture below) which in occur loop we only set to 3 times, due to some limitation. In this lesson will fix this issue.

Original data

- {'task': {'user_id': 'test_user', 'content': 'my test content', 'is_done': False, 'created_time': 1719650549, 'task_id': 'task_e1364a0b950d4effabcc491eab6856c2', 'ttl': 1719736949}}
- {'task': {'user_id': 'test_user', 'content': 'my test content', 'is_done': False, 'created_time': 1719650550, 'task_id': 'task_33423e1717254087a39b18f662545ad9', 'ttl': 1719736950}}
- {'task': {'user_id': 'test_user', 'content': 'my test content', 'is_done': False, 'created_time': 1719650552, 'task_id': 'task_f97fb0e44d8e4e79b989e0681e390e58', 'ttl': 1719736952}}

```
list_task_response = list_tasks("test_user")
assert list_task_response.status_code==200
data = list_task_response.json()
print(data)
```

```
def list_tasks(user_id):
    return requests.get(ENDPOINT + f"/list-tasks/{user_id}")
```

There's issue due to some limitation it will have 10 item in data

{'tasks': [{'is_done': False, 'content': 'my test content', 'ttl': 1719736028, 'user_id': 'test_user', 'task_id': 'task_f118a6bad9c24aba964e8c66f30449b3', 'created_time': 1719649628}, {'is_done': False, 'content': 'my test content', 'ttl': 1719736026, 'user_id': 'test_user', 'task_id': 'task_2cc5365f23d84e4c8c8ea1e08a84e15d', 'created_time': 1719649626}, {'is_done': False, 'content': 'my test content', 'ttl': 1719736025, 'user_id': 'test_user', 'task_id': 'task_57cb7741f48b4752a6a9804dd979c143', 'created_time': 1719649625}, {'is_done': True, 'content': 'my update content', 'ttl': 1719736018, 'user_id': 'test_user', 'task_id': 'task_7a07188fa65a477794e72ac516947128', 'created_time': 1719649620}, {'is_done': False, 'content': 'my test content', 'ttl': 1719736018, 'user_id': 'test_user' 'task_id': 'task_e9a605dd7c444d14aa718e2be9e23b2d', 'created_time': 1719649618}, {'is_done': False, 'content': 'my test content', 'ttl': 1719735987, 'user_id': 'test_user', 'task_id': 'task_e2dcac1 dbc439e9e94fe1c11f7801d', 'created_time': 1719649587}, {'is_done': False, 'content': 'my test content', 'ttl': 1719735986, 'user_id': 'test_user', 'task_id': 'task_fb7675ca582347879e0ab84a1da8d140' 'created_time': 1719649586}, {'is_done': False, 'content': 'my test content', 'ttl': 1719735985, 'user_id': 'test_user', 'task_id': 'task_57b746fb59a94fb3b5ee5dea86a365ab', 'created_time': 1719649 }, {'is_done': True, 'content': 'my update content', 'ttl': 1719735980, 'user_id': 'test_user', 'task_id': 'task_a27d70831bcd4101a6d5e14c3a83185f', 'created_time': 1719649580}, {'is_done': False, ' ntent': 'my test content', 'ttl': 1719735977, 'user_id': 'test_user', 'task_id': 'task_eac65f0d78934f9f9f0357e8885ecea4', 'created_time': 1719649577}]}

Now let show the looping data requesting data for more detail as below picture.

When we create new_task_payload(), and create_task(payload) will create a request and using the payload we add.

```
def test_can_list_tasks():
    #create N tasks
    for _ in range(3):
        payload = new_task_payload()
        create_task_response=create_task(payload)
        assert create_task_response.status_code==200
```

- Looping code
- create payload item
  - payload = new_task_payload()
    - {'content': 'my test content', 'user_id': 'test_user', 'is_done': False}
    - {'content': 'my test content', 'user_id': 'test_user', 'is_done': False}
    - {'content': 'my test content', 'user_id': 'test_user', 'is_done': False}

Set the items key and value

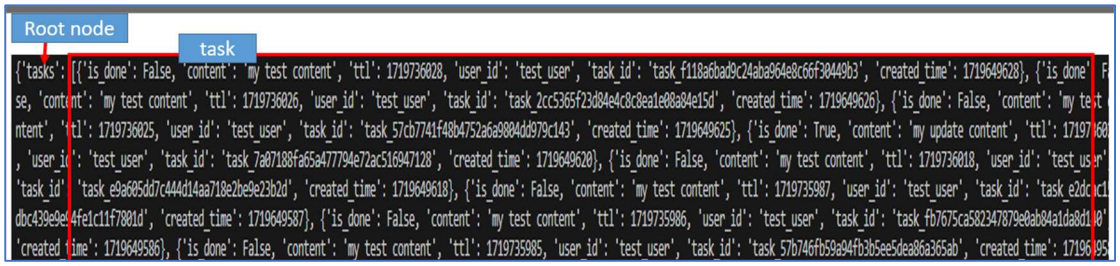
- Send request and response back(server response)
  - create_task_response.json

Send request and response

- {'task': {'user_id': 'test_user', 'content': 'my test content', 'is_done': False, 'created_time': 1719650549, 'task_id': 'task_e1364a0b950d4effabcc491eab6856c2', 'ttl': 1719736949}}
- {'task': {'user_id': 'test_user', 'content': 'my test content', 'is_done': False, 'created_time': 1719650550, 'task_id': 'task_33423e1717254087a39b18f662545ad9', 'ttl': 1719736950}}
- {'task': {'user_id': 'test_user', 'content': 'my test content', 'is_done': False, 'created_time': 1719650552, 'task_id': 'task_f97fb0e44d8e4e79b989e0681e390e58', 'ttl': 1719736952}}

Step4 let get the tasks item from response

The root node task have a list of task. So we need to get the root node as the key to get value of the task item.



Change helper variable n instead of hot code

Original	New
for _ in range(3)	n=3 for _ in range(n):

Add task item

tasks=data['tasks'] ➔ get the data and filter the items

```
#CASE3 LIST TASK_ID
def test_can_list_tasks():
    #create N tasks
     n=3
     for _ in range(n):
          payload = new_task_payload()
          create_task_response=create_task(payload)
          assert create_task_response.status_code==200
     #user_id=payload["user_id"]
     list_task_response = list_tasks("test_user")
     assert list_task_response.status_code==200
     data = list_task_response.json()
     tasks=data['tasks']
     assert len(tasks)==n
     print(data)
```

Now let run this code, but instead of running the whole suite, which is sending many request and take more time, so instead let just run this test.

```
Command syntax: pytest –b –v –s <test_script.sh>::functionname
Example: pytest -v -s .\test_todo_api_draft.py::test_can_list_tasks
```

When you run assert will failed, due to there's an server side limitation, as you can see above the data occur 10 item, which should be 3, please refer below outoput.

```
>       assert len(tasks)==n
E       AssertionError: assert 10 == 3
E        +  where 10 = len([{'content': 'my test content', 'created_time': 171971315
ted_time': 1719713158, 'is_done': False, 'task_id': 'task_39d769393e9c4275a483ce76c5
6f1dc24c05bb0d701df5747dd4', ...}, {'content': 'my test content', 'created_time': 171
, 'created_time': 1719712053, 'is_done': False, 'task_id': 'task_3952f9258cd940e19aee
```

The reason is because I have been running couples of time with the same user_id, so it will return the max of 10 items. So solve this we need to use a complete new user_id.

Step 5 create a unique user_id UUID

To fix above problem on user_id, we need to generate a random number and string then append to the user_id. We can use UUID this module support it, so need to first import it.

This is what UUID look like have string and number, which will big enough that won't have duplicate value

```
8be4df61-93ca-11d2-aa0d-00e098032b8c-dbxDefault
8be4df61-93ca-11d2-aa0d-00e098032b8c-dbDefault
8be4df61-93ca-11d2-aa0d-00e098032b8c-KEKDefault
8be4df61-93ca-11d2-aa0d-00e098032b8c-PKDefault
07a66697-d400-4903-b3da-67a61d2b7058-Tcg2ConfigInfo
aa1305b9-01f3-4afb-920e-c9b979a852fd-SecureBootData
1f2d63e1-febd-4dc7-9cc5-ba2b1cef9c5b-FeData
3a997502-647a-4c82-998e-52ef9486a247-AmdSetup
5bce4c83-6a97-444b-63b4-672c014742ff-IrsiInfo
146b234d-4052-4e07-b326-11220f8e1fe8-lBoot0002
```

Import uuid

Create uuid method in uuid it supports 5 version, you can use version4 is enough:

```
user_id=uuid.uuid4().hex
```

Let create uuid for user_id, and content_string,

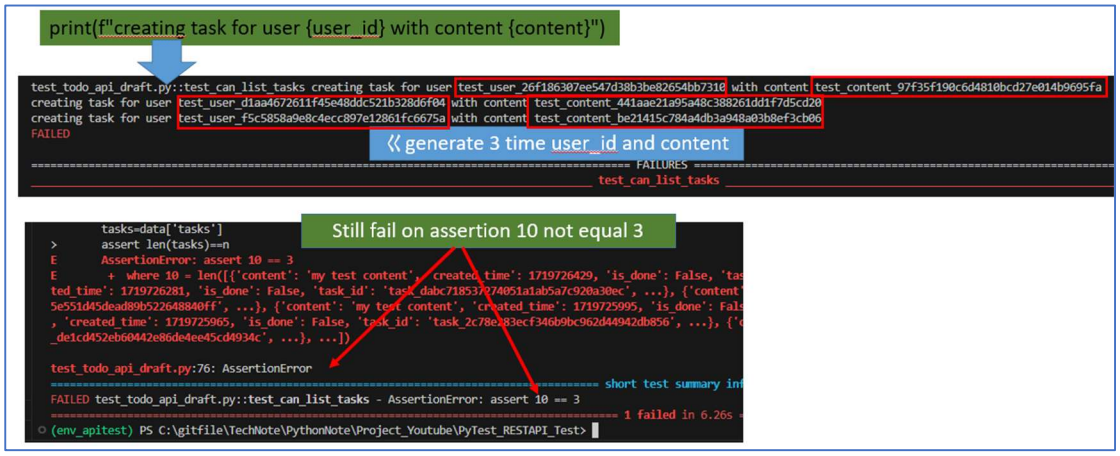
```
user_id=f"test_user_{uuid.uuid4().hex}"
content=f"test_content_{uuid.uuid4().hex}"
```

Refactor our new_task_payload

original	Refactor

return {
 "content": "my test content",
 "user_id": "test_user",
 "is_done": False,
 }
	user_id=f"test_user_{uuid.uuid4().hex}"
	 content=f"test_content_{uuid.uuid4().hex}"
	
	 #print the user_id and content from UUID
	 print(f"creating task for user {user_id} with content {content}")
	
	 return {
	 #"content": "my test content",
	 "content": content,
	 "user_id": user_id,
	 "is_done": False,
	 }

Let see below picture it generate three different user_id and content_id.



Now let run again, it will still fail same problem, it because not update the use list_task("test_user")

Step6: update the user_id and refactor some code

Move the generate payload out of the loop so we only generate once, so we will create 3 task with only one user_id. The task_id will be different because the server will generate.

Generate 1 single userid, and three task and task_id which gernate by server

original	def test_can_list_tasks():
	#create N tasks
	n=3
	for _ in range(n):
	payload = new_task_payload()

Refactor	#CASE3 LIST TASK_ID
	def test_can_list_tasks():
	#create N tasks
	n=3
	payload = new_task_payload()
	for _ in range(n):
	create_task_response=create_task(payload)
	assert create_task_response.status_code==200
	# list tasks and check that there are N items
	user_id=payload["user_id"]
	list_task_response = list_tasks(user_id)

```
#CASE3 LIST TASK_ID
def test_can_list_tasks():
    #create N tasks
    n=3
    for _ in range(n):
        payload = new_task_payload()
        create_task_response=create_task(payload)
        assert create_task_response.status_code==200
    #user_id=payload["user_id"]
    list_task_response = list_tasks("test_user")

    assert list_task_response.status_code==200
    data = list_task_response.json()
    tasks=data['tasks']
    assert len(tasks)==n
    print(data)
```

Move outside loop

```
def test_can_list_tasks():
    #create N tasks
    n=3
    #only generate payload once
    #generate once with only one user_id
    payload = new_task_payload()
    for _ in range(n):
        create_task_response=create_task(payload)
        assert create_task_response.status_code==200
    user_id=payload["user_id"]
    list_task_response = list_tasks(user_id)

    assert list_task_response.status_code==200
    data = list_task_response.json()
    tasks=data['tasks']
    assert len(tasks)==n
    print(data)
```

When running the code will show three item, each user_id,content will use the same UUID, and task_id will have three different ID. Please refer the result data as below:

```
(env_apitest) PS C:\gitfile\TechNote\PythonNote\Project_Youtube\PyTest_RESTAPI_Test> pytest -v -s .\test_todo_api_draft.py::test_can_list_tasks
========================================================================== test session starts ==========================================
platform win32 -- Python 3.12.0, pytest-8.2.2, pluggy-1.5.0 -- C:\gitfile\TechNote\PythonNote\Project_Youtube\PyTest_RESTAPI_Test\env_apitest\Scripts\python.
cachedir: .pytest_cache
rootdir: C:\gitfile\TechNote\PythonNote\Project_Youtube\PyTest_RESTAPI_Test
collected 1 item

test_todo_api_draft.py::test_can_list_tasks creating task for user test_user_4899aa59b53a44bab1dc2f9fbd57fe33 with content test_content_f6cc3026ad844f02b164
{'tasks': [{'is_done': False, 'content': 'test_content_f6cc3026ad844f02b1649fd3b4bdb90a', 'ttl': 1719815524, 'user_id': 'test_user_4899aa59b53a44bab1dc2f9fbd
984691809905d70c', 'created_time': 1719729124}, {'is_done': False, 'content': 'test_content_f6cc3026ad844f02b1649fd3b4bdb90a', 'ttl': 1719815523, 'user_id':
3', 'task_id': 'task_1ca7e433e7c0413ebaf40e67b2974b52', 'created_time': 1719729123}, {'is_done': False, 'content': 'test_content_f6cc3026ad844f02b1649fd3b4bd
_user_4899aa59b53a44bab1dc2f9fbd57fe33', 'task_id': 'task_b576a0fc35054c4c9122684ccb3769a7', 'created_time': 1719729122}]}
PASSED

========================================================================== 1 passed in 4.14s ============================================
```

user_id and content

Let me show the item as below, the red color is where we use UUID to generate User_ID and content. And the orange color is generate by server when response

back,

```
test_todo_api_draft.py::test_can_list_tasks creating task for user
test_user_4899aa59b53a44bab1dc2f9fbd57fe33
with content test_content_f6cc3026ad844f02b1649fd3b4bdb90a

{'tasks': [{'is_done': False, 'content': 'test_content_f6cc3026ad844f02b1649fd3b4bdb90a', 'ttl':
1719815524, 'user_id': 'test_user_4899aa59b53a44bab1dc2f9fbd57fe33', 'task_id':
'task_67c75288686e4e74984691809905d70c', 'created_time': 1719729124},

{'is_done': False, 'content': 'test_content_f6cc3026ad844f02b1649fd3b4bdb90a', 'ttl': 1719815523,
'user_id': 'test_user_4899aa59b53a44bab1dc2f9fbd57fe33', 'task_id':
task_1ca7e433e7c0413ebaf40e67b2974b52', 'created_time': 1719729123},

{'is_done': False, 'content': 'test_content_f6cc3026ad844f02b1649fd3b4bdb90a', 'ttl': 1719815522,
'user_id': 'test_user_4899aa59b53a44bab1dc2f9fbd57fe33', 'task_id':
'task_b576a0fc35054c4c9122684ccb3769a7', 'created_time': 1719729122}]}
```

Generate each new task_id

Summary full code in case 3:

```
import uuid

#CASE3 LIST TASK_ID
def test_can_list_tasks():
    #1. create N tasks
      n=3
      #only generate payload once
      #generate once with only one user_id
      payload = new_task_payload()
      for _ in range(n):
            create_task_response=create_task(payload)
            assert create_task_response.status_code==200
      user_id=payload["user_id"]
      #2. list tasks, and check that there are N items
      list_task_response = list_tasks(user_id)

      assert list_task_response.status_code==200
      data = list_task_response.json()
      tasks=data['tasks']
      assert len(tasks)==n
      #print(data)

def create_task(payload):
```

```
        return requests.put(ENDPOINT + "/create-task", json=payload)


def new_task_payload():
        #limitation add uuid
        user_id=f"test_user_{uuid.uuid4().hex}"
        content=f"test_content_{uuid.uuid4().hex}"
        #there are 5 version, 4 version is enough, and it's an object
        #print generate uuid's userid and content
        print(f"creating task for user {user_id} with content {content}")

        return    {
                #"content": "my test content",
                "content": content,
                "user_id": user_id,
                "is_done": False,
    }

def list_tasks(user_id):
        return requests.get(ENDPOINT + f"/list-tasks/{user_id}")

def new_task_payload():
        #limitation add uuid
        user_id=f"test_user_{uuid.uuid4().hex}"
        content=f"test_content_{uuid.uuid4().hex}"
        #there are 5 version, 4 version is enough, and it's an object
        #print generate uuid's userid and content
        #print(f"creating task for user {user_id} with content {content}")

        return    {
                #"content": "my test content",
                "content": content,
                "user_id": user_id,
                "is_done": False,
    }
```

Case4 delete items

In this test we will delete the task and get the task again

We will test three step:

- Create task
- Delete task
- Get task an check that it's not found

Step1 create test_delete task and delete helper function

```
…
#case4 delete task
def test_can_delete_tasks():
        pass
….
def delete_task(task_id):
        return requests.delete(ENDPOINT + f"/delete-task/{task_id}")
```

Step2 go back to test_delete_task

Create a task

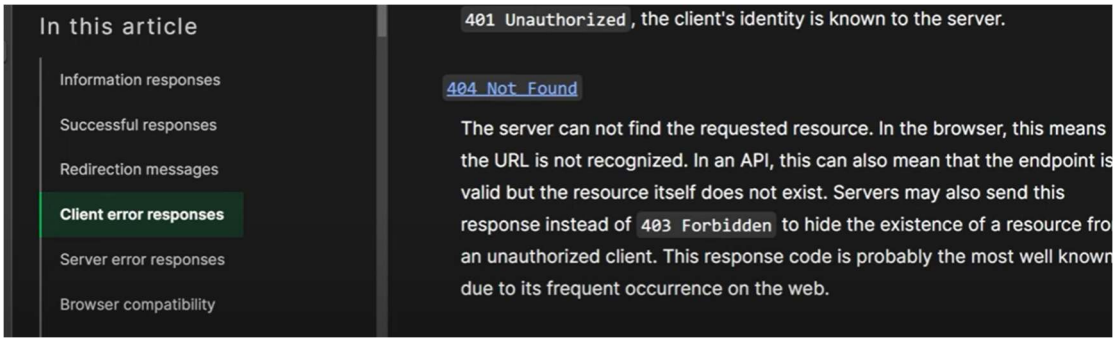
```
def test_can_delete_tasks():
        #create a task
        payload=new_task_payload()
        #create_task_response=requests.put(ENDPOINT + "/create-task",
json=payload)
        create_task_response=create_task(payload)
        assert create_task_response.status_code == 200
        task_id = create_task_response.json()["task"]["task_id"]
```

```
#delete a task
        delete_task_response=delete_task(task_id)
        assert delete_task_response.status_code==200
```

```
#get task
        get_task_response=get_task(task_id)
        print(get_task_response.status_code) #404 mean not found
        #assert get_task_response.status_code
```

How do we know is it found? We use the status code, then how to know which?

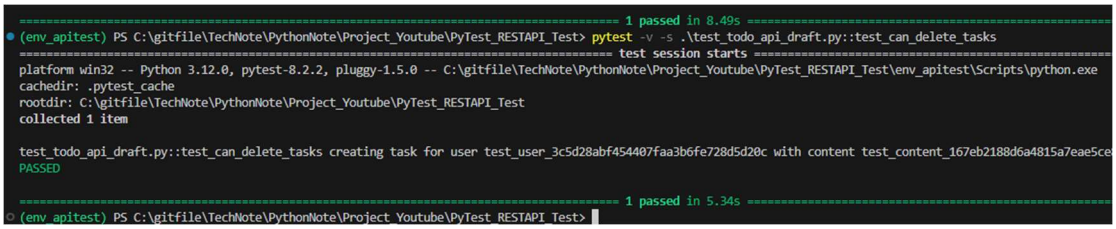
When it start with 4XX mean client error. Let run this code and print status code to see result, it will print 404 meaning not found



Now we can comment the print or remove and add assert with it.

```
#get task
    get_task_response=get_task(task_id)
    #print(get_task_response.status_code) #404 mean not found
    assert get_task_response.status_code
```

Let run again and result as below



Summary full code case4

```
def test_can_delete_tasks():
    #create a task
    payload=new_task_payload()
    create_task_response=create_task(payload)
    assert create_task_response.status_code == 200
    task_id = create_task_response.json()["task"]["task_id"]
    #delete the task
    delete_task_response=delete_task(task_id)
    assert delete_task_response.status_code==200
    # get the task and    check that it's not found
    get_task_response=get_task(task_id)
    assert get_task_response.status_code

def create_task(payload):
    return requests.put(ENDPOINT + "/create-task", json=payload)
```

```
def get_task(task_id):
    return requests.get(ENDPOINT + f"/get-task/{task_id}")
def delete_task(task_id):
    return requests.delete(ENDPOINT + f"/delete-task/{task_id}")
def new_task_payload():
    #limitation add uuid
    user_id=f"test_user_{uuid.uuid4().hex}"
    content=f"test_content_{uuid.uuid4().hex}"
    #there are 5 version, 4 version is enough, and it's an object
    #print generate uuid's userid and content
    print(f"creating task for user {user_id} with content {content}")

    return    {
        #"content": "my test content",
        "content": content,
        "user_id": user_id,
        "is_done": False,
  }
```

Run all test case

Now go to top we can now remove below

```
def test_can_Call_endpoint():
    response=requests.get(ENDPOINT)
    assert response.status_code == 200
```

since already have enough detail in below already, it just for beginning to understand to to send request. I will remove all the print, to make code and result more near.

Now let run all the code, all 4 test case pass

```
(env_apitest) PS C:\gitfile\TechNote\PythonNote\Project_Youtube\PyTest_RESTAPI_Test> pytest -v -s .\test_todo_api_draft.py
============================================================================= test session starts =============================================================================
platform win32 -- Python 3.12.0, pytest-8.2.2, pluggy-1.5.0 -- C:\gitfile\TechNote\PythonNote\Project_Youtube\PyTest_RESTAPI_Test\env_apitest\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\gitfile\TechNote\PythonNote\Project_Youtube\PyTest_RESTAPI_Test
collected 4 items

test_todo_api_draft.py::test_can_create_task PASSED
test_todo_api_draft.py::test_can_update_task PASSED
test_todo_api_draft.py::test_can_list_tasks PASSED
test_todo_api_draft.py::test_can_delete_tasks PASSED

============================================================================== 4 passed in 18.62s ==============================================================================
(env_apitest) PS C:\gitfile\TechNote\PythonNote\Project_Youtube\PyTest_RESTAPI_Test>
```