Dhirubhai Ambani
Institute of Information and Communication Technology

Software Engineering (IT314)

Course instructors

Prof. Saurabh Tiwari

Prof. Manish Khare

Lab 8

Group submission (Grp-10)

Date

19th April 2023

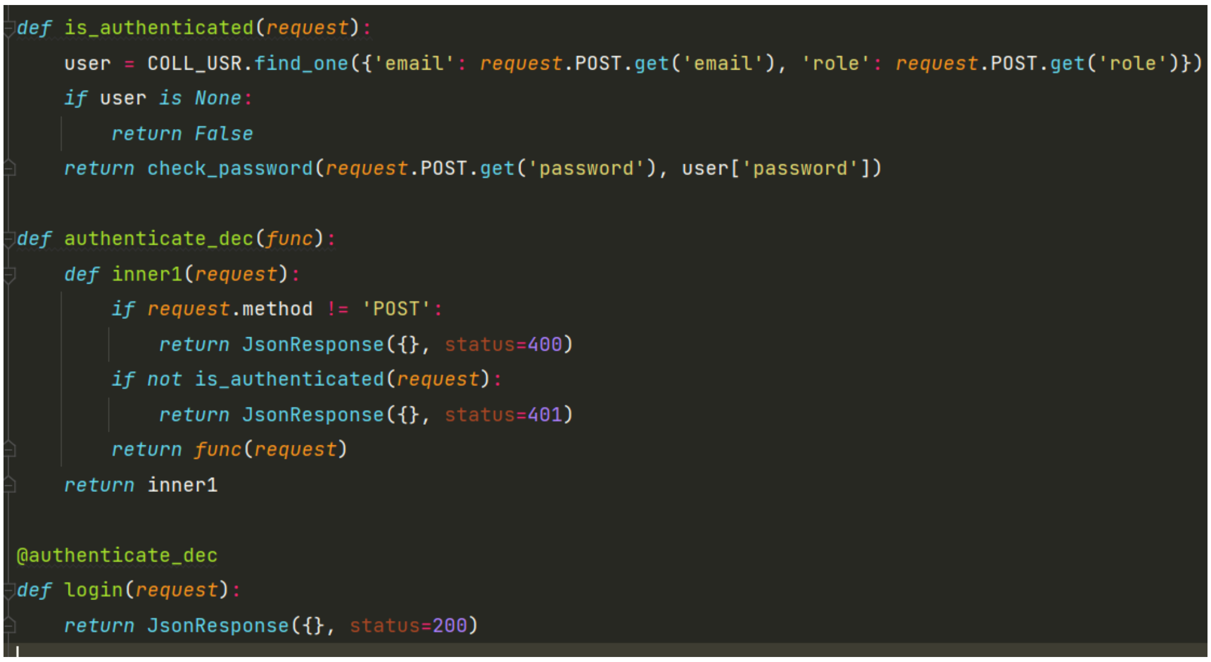
table of contents

Choosing the module

The module we are using to write the tests is our own backend code we've written as it will help test the project and we can use it to progress our project.

Writing the test cases

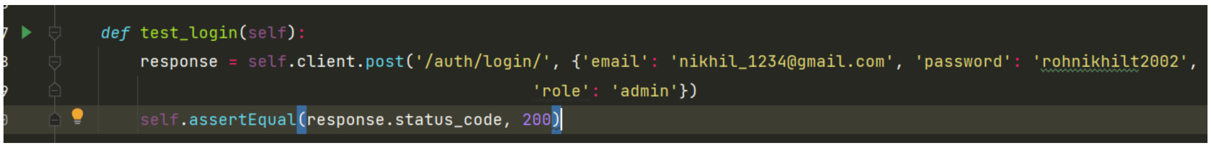
Test case #1:



```python
def is_authenticated(request):
    user = COLL_USR.find_one({'email': request.POST.get('email'), 'role': request.POST.get('role')})
    if user is None:
        return False
    return check_password(request.POST.get('password'), user['password'])

def authenticate_dec(func):
    def inner1(request):
        if request.method != 'POST':
            return JsonResponse({}, status=400)
        if not is_authenticated(request):
            return JsonResponse({}, status=401)
        return func(request)
    return inner1

@authenticate_dec
def login(request):
    return JsonResponse({}, status=200)
```

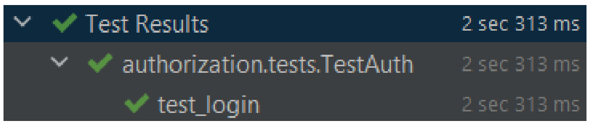


```python
    def test_login(self):
        response = self.client.post('/auth/login/', {'email': 'nikhil_1234@gmail.com', 'password': 'rohnikhilt2002',
                                                     'role': 'admin'})
        self.assertEqual(response.status_code, 200)
```

This test case is testing the view "login". When the user writes their mail, password and role, this view is run by backend and test_login is the function to test this code.

The test will send the post request to the client and get the response and checks its response code which would run perfectly fine as the parameters in the payload are sent correctly.

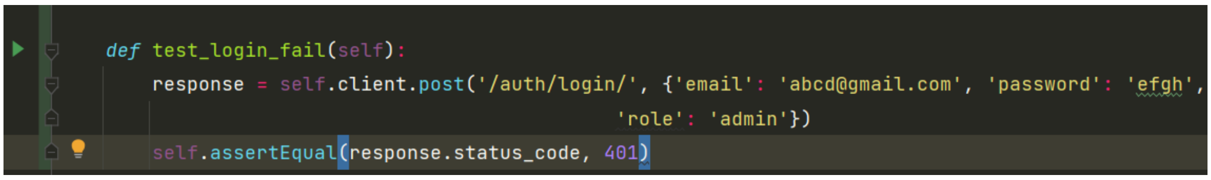
Output on running the test:



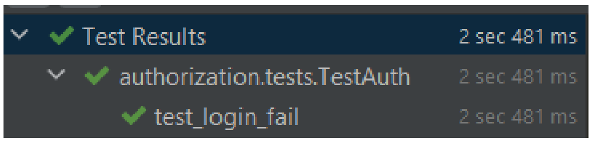
```
Test Results                        2 sec 313 ms
  authorization.tests.TestAuth      2 sec 313 ms
    test_login                      2 sec 313 ms
```

Testcase #2:

This test case will be testing the same previous view but with wrong parameters in the payload which would return unauthorized as response.

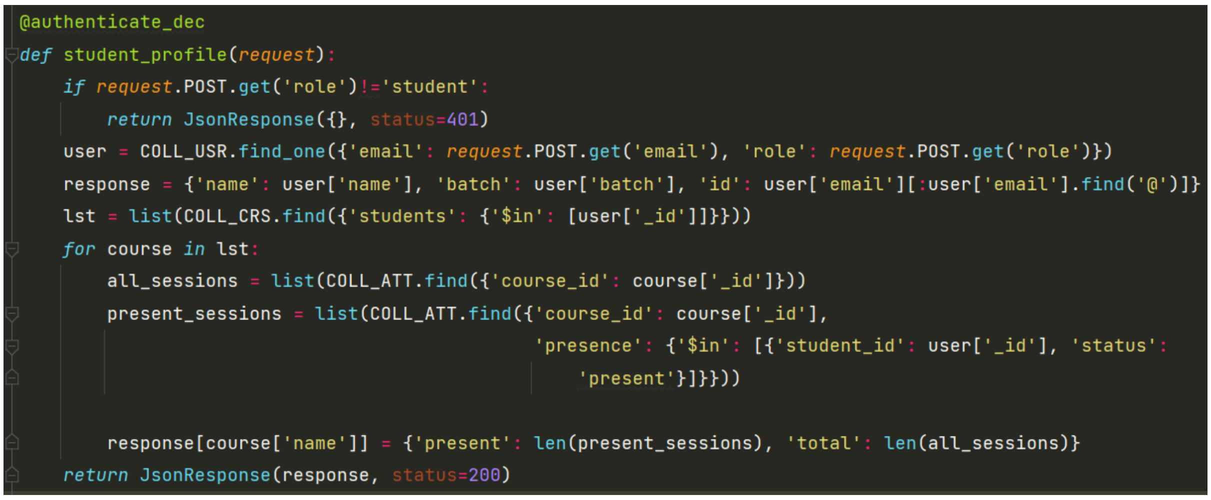


```
def test_login_fail(self):
    response = self.client.post('/auth/login/', {'email': 'abcd@gmail.com', 'password': 'efgh',
                                                 'role': 'admin'})
    self.assertEqual(response.status_code, 401)
```



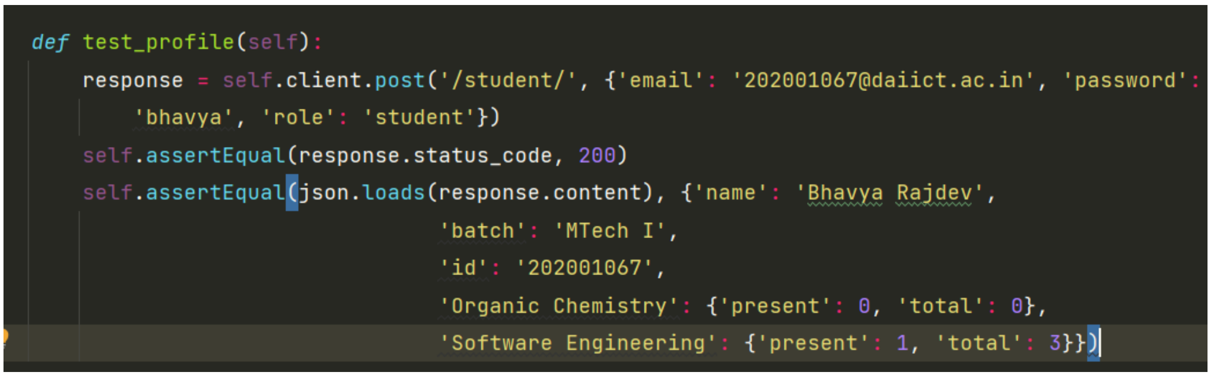
```
Test Results                        2 sec 481 ms
  authorization.tests.TestAuth      2 sec 481 ms
    test_login_fail                 2 sec 481 ms
```

Testcase #3:



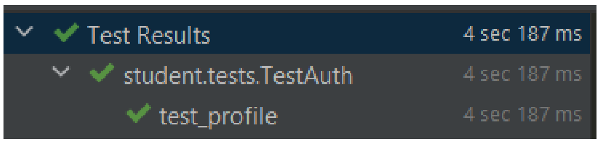
```
@authenticate_dec
def student_profile(request):
    if request.POST.get('role')!='student':
        return JsonResponse({}, status=401)
    user = COLL_USR.find_one({'email': request.POST.get('email'), 'role': request.POST.get('role')})
    response = {'name': user['name'], 'batch': user['batch'], 'id': user['email'][:user['email'].find('@')]}
    lst = list(COLL_CRS.find({'students': {'$in': [user['_id']]}}))
    for course in lst:
        all_sessions = list(COLL_ATT.find({'course_id': course['_id']}))
        present_sessions = list(COLL_ATT.find({'course_id': course['_id'],
                                               'presence': {'$in': [{'student_id': user['_id'], 'status':
                                                   'present'}]}}))

        response[course['name']] = {'present': len(present_sessions), 'total': len(all_sessions)}
    return JsonResponse(response, status=200)
```



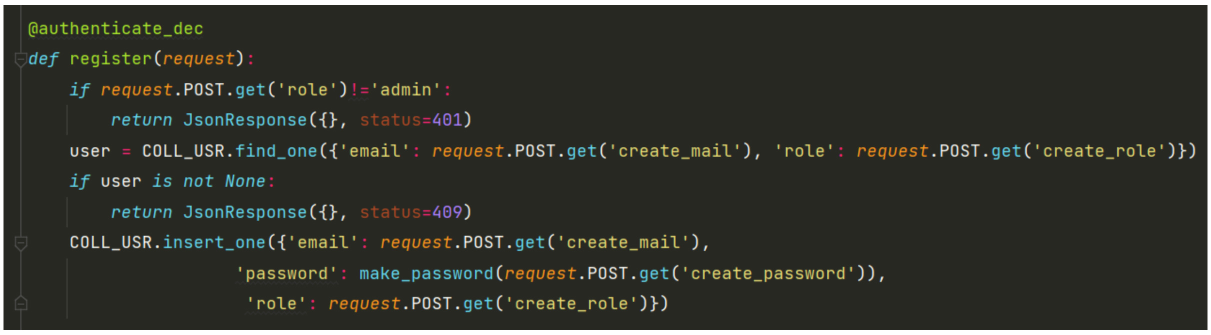
```
def test_profile(self):
    response = self.client.post('/student/', {'email': '202001067@daiict.ac.in', 'password':
        'bhavya', 'role': 'student'})
    self.assertEqual(response.status_code, 200)
    self.assertEqual(json.loads(response.content), {'name': 'Bhavya Rajdev',
                                                    'batch': 'MTech I',
                                                    'id': '202001067',
                                                    'Organic Chemistry': {'present': 0, 'total': 0},
                                                    'Software Engineering': {'present': 1, 'total': 3}})
```

Here the test_profile function is testing the student_profile view which is invoked when the student clicks on the profile. The response object is being checked with all its parameters along with the response code.

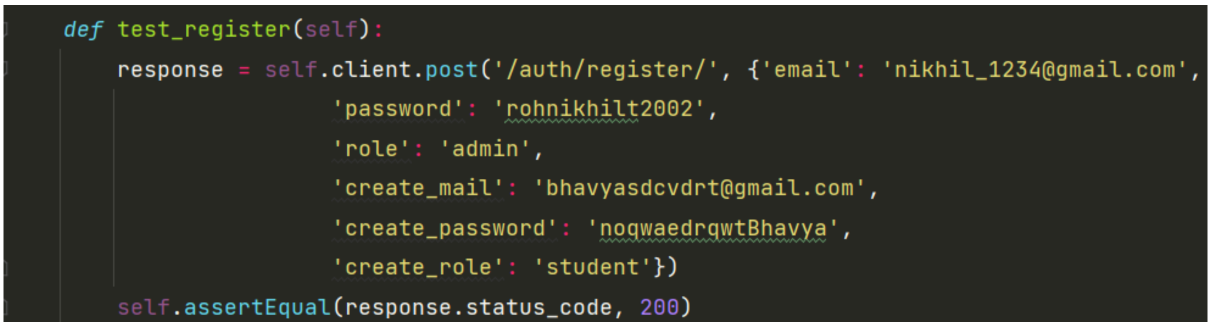


```
✓ Test Results                    4 sec 187 ms
  ✓ student.tests.TestAuth        4 sec 187 ms
    ✓ test_profile                4 sec 187 ms
```

Testcase #4:

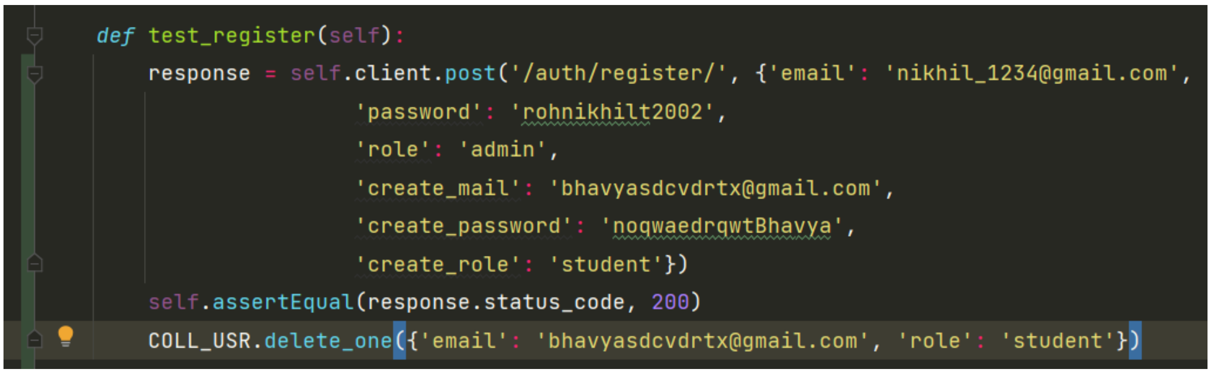


```python
@authenticate_dec
def register(request):
    if request.POST.get('role')!='admin':
        return JsonResponse({}, status=401)
    user = COLL_USR.find_one({'email': request.POST.get('create_mail'), 'role': request.POST.get('create_role')})
    if user is not None:
        return JsonResponse({}, status=409)
    COLL_USR.insert_one({'email': request.POST.get('create_mail'),
                'password': make_password(request.POST.get('create_password')),
                 'role': request.POST.get('create_role')})
```



```python
    def test_register(self):
        response = self.client.post('/auth/register/', {'email': 'nikhil_1234@gmail.com',
                            'password': 'rohnikhilt2002',
                            'role': 'admin',
                            'create_mail': 'bhavyasdcvdrt@gmail.com',
                            'create_password': 'noqwaedrqwtBhavya',
                            'create_role': 'student'})
        self.assertEqual(response.status_code, 200)
```

This test correctly returns the 200 code but on executing the same test twice the test will return the code 409 saying that the data already exists in the code. **The problem here is** that we can't run this test twice and to resolve the error we must delete the data that was inserted by calling the register function

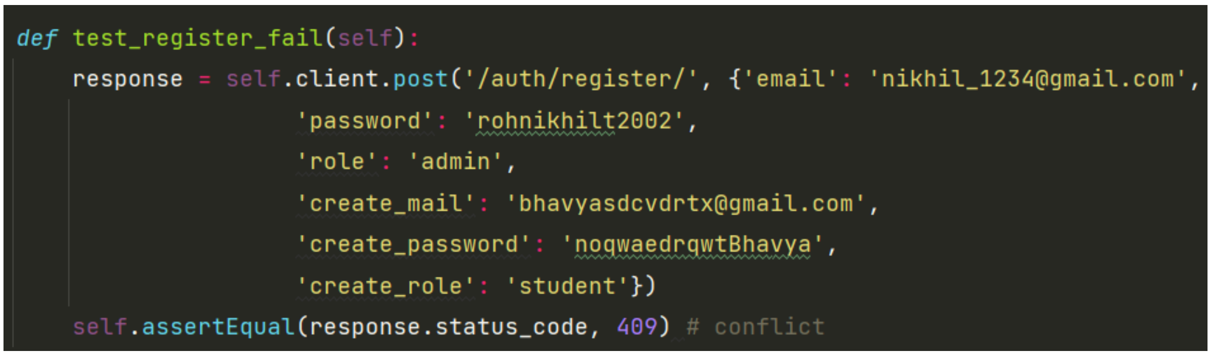


```python
    def test_register(self):
        response = self.client.post('/auth/register/', {'email': 'nikhil_1234@gmail.com',
                            'password': 'rohnikhilt2002',
                            'role': 'admin',
                            'create_mail': 'bhavyasdcvdrtx@gmail.com',
                            'create_password': 'noqwaedrqwtBhavya',
                            'create_role': 'student'})
        self.assertEqual(response.status_code, 200)
        COLL_USR.delete_one({'email': 'bhavyasdcvdrtx@gmail.com', 'role': 'student'})
```

The above function is the correct implementation of the test function of register view which deletes the data that was inserted in the database by calling the post method.

This resolves the error and now we can call the same test function multiple times.

Testcase #5



```python
def test_register_fail(self):
    response = self.client.post('/auth/register/', {'email': 'nikhil_1234@gmail.com',
                                'password': 'rohnikhilt2002',
                                'role': 'admin',
                                'create_mail': 'bhavyasdcvdrtx@gmail.com',
                                'create_password': 'noqwaedrqwtBhavya',
                                'create_role': 'student'})
    self.assertEqual(response.status_code, 409) # conflict
```

Above test function tests for the failed registration as the user already exists in the database and view returns the status code 409 which refers to conflict in the database.

Learnings

Django by default provides the ‘unittest’ module inside it for unit testing. This module can be used for unit test the urls, forms, views in Django. In our work, the module is used to test the different views that are implemented in different apps. The test functions send the post request to one of the views, records the response and do checks if the response is as intended or not.

One thing to take care while testing is that the tests should not be doing any changes to the database and if it does for the testing, then the changes must be reverted as seen in **Testcase #4** as making changes to database should not be part of unit testing.

Testing the failing test cases are equally import as testing for successful scenarios as seen in **Testcase #5** otherwise the code might give errors in frontend and code might also be prone to attacks by hackers.