SOFTWARE ENGINEERING MILESTONE 5

SUBMITTED IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE COURSE:

BSCSS3001: Software Engineering

By:
Arya Bhattacharyya (21f2000436)
Varun Venkatesh (21f1000743)
Chirag Goel (21f2000540)

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS
April, 2023

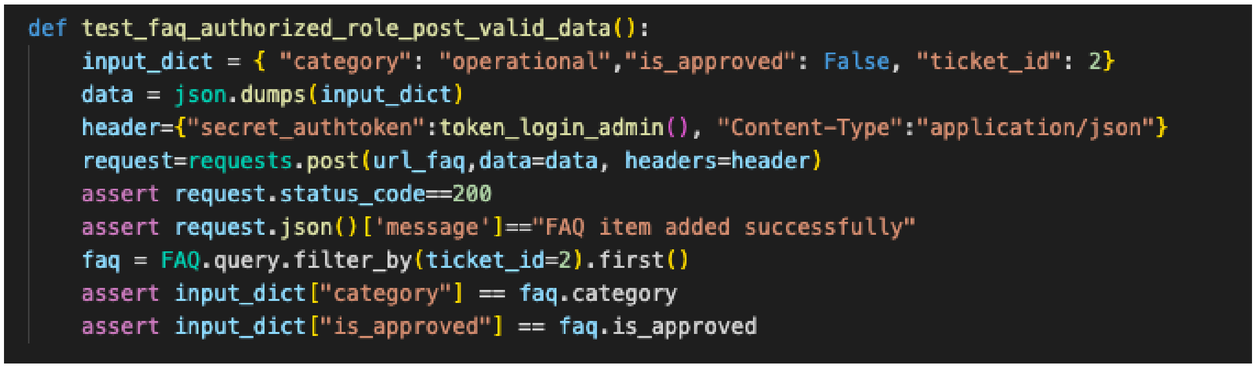
A new item suggested by a support agent is approved/rejected by the admin for FAQ

Page being tested: http://127.0.0.1:5000/api/faq
Inputs:
- Request Method: POST
- JSON: { "category": "operational", "is_approved": false, "ticket_id": 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item added successfully”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item added successfully”}

Result: Success



```python
def test_faq_authorized_role_post_valid_data():
    input_dict = { "category": "operational","is_approved": False, "ticket_id": 2}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.post(url_faq,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="FAQ item added successfully"
    faq = FAQ.query.filter_by(ticket_id=2).first()
    assert input_dict["category"] == faq.category
    assert input_dict["is_approved"] == faq.is_approved
```

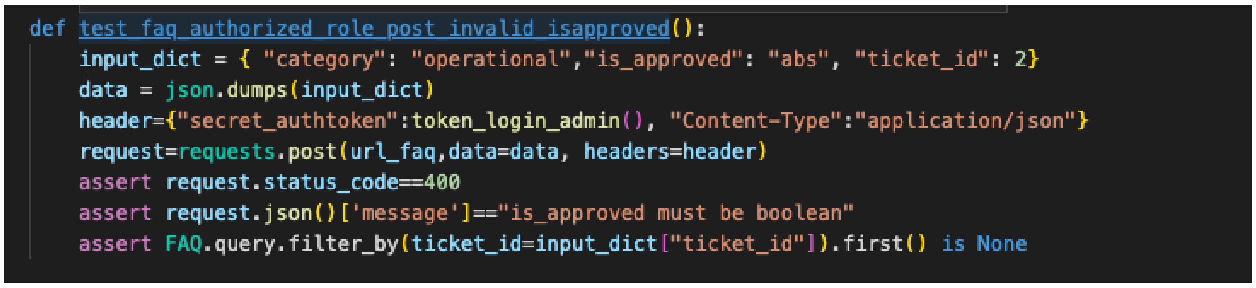
A new item suggested by a support agent is approved/rejected by the admin for FAQ but the request body has invalid data

Page being tested: http://127.0.0.1:5000/api/faq
Inputs:
- Request Method: POST
- Json body: { "category": "operational", "is_approved": “abcd”, "ticket_id": 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 400
- JSON: {“message”: “is_approved must be boolean”}

Actual Output:
- HTTP Status Code: 400
- JSON: {“message”: “is_approved must be boolean”}

Result: Success



```python
def test_faq_authorized_role_post_invalid_isapproved():
    input_dict = { "category": "operational","is_approved": "abs", "ticket_id": 2}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.post(url_faq,data=data, headers=header)
    assert request.status_code==400
    assert request.json()['message']=="is_approved must be boolean"
    assert FAQ.query.filter_by(ticket_id=input_dict["ticket_id"]).first() is None
```

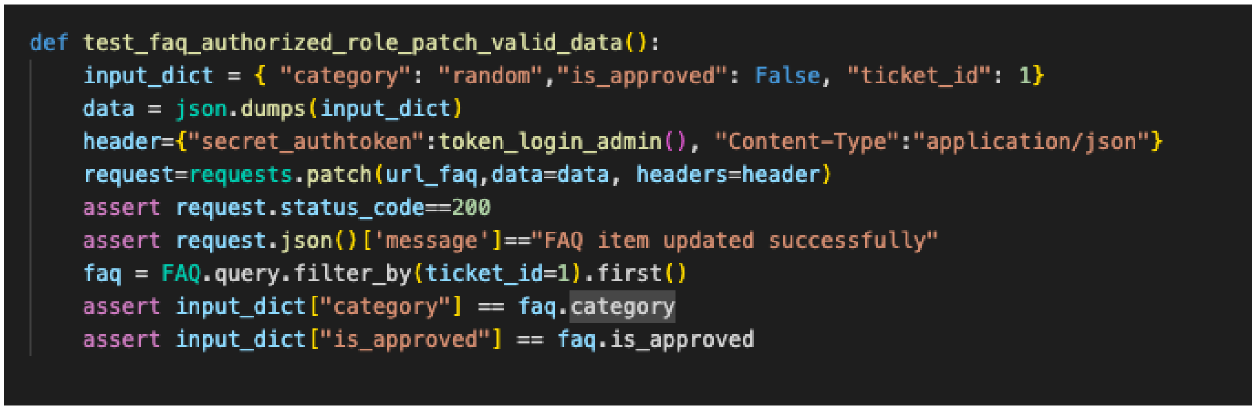
An existing ticket in the FAQ is updated with a new category

Page being tested: http://127.0.0.1:5000/api/faq
Inputs:
- Request Method: PATCH
- Json body: { "category": "random","is_approved": false, "ticket_id": 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item updated successfully”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item updated successfully”}

Result: Success



```python
def test_faq_authorized_role_patch_valid_data():
    input_dict = { "category": "random","is_approved": False, "ticket_id": 1}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.patch(url_faq,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="FAQ item updated successfully"
    faq = FAQ.query.filter_by(ticket_id=1).first()
    assert input_dict["category"] == faq.category
    assert input_dict["is_approved"] == faq.is_approved
```

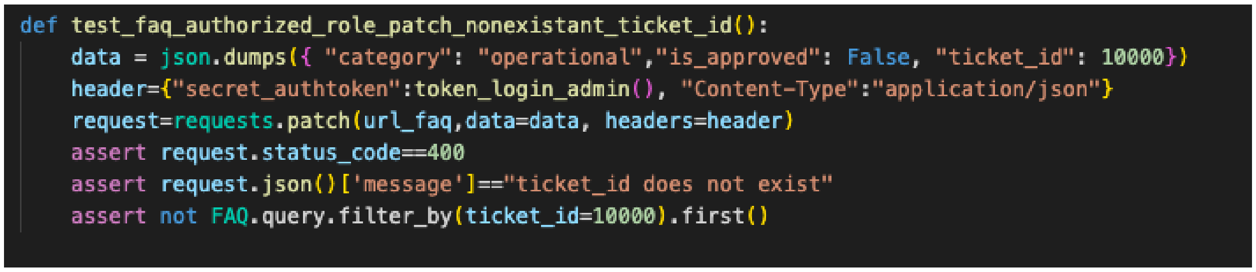
A request is sent to to update a non-existing ticket in the FAQ

Page being tested: http://127.0.0.1:5000/api/faq
Inputs:
- Request Method: PATCH
- Json body: { "category": "random","is_approved": false, "ticket_id": 1000}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 400
- JSON: {“message”: “ticket_id does not exist”}

Actual Output:
- HTTP Status Code: 400
- JSON: {“message”: “ticket_id does not exist”}

Result: Success



```python
def test_faq_authorized_role_patch_nonexistant_ticket_id():
    data = json.dumps({ "category": "operational","is_approved": False, "ticket_id": 10000})
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.patch(url_faq,data=data, headers=header)
    assert request.status_code==400
    assert request.json()['message']=="ticket_id does not exist"
    assert not FAQ.query.filter_by(ticket_id=10000).first()
```

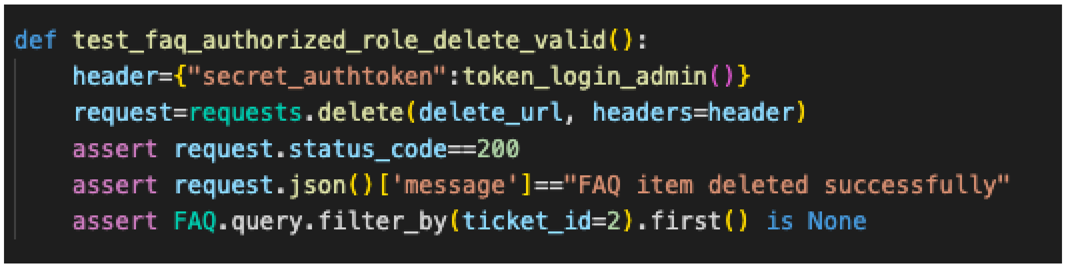
A ticket is removed from the FAQ Table

Page being tested: http://127.0.0.1:5000/api/faq/2
Inputs:
- Request Method: DELETE
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item deleted successfully”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “FAQ item deleted successfully”}

Result: Success



```python
def test_faq_authorized_role_delete_valid():
    header={"secret_authtoken":token_login_admin()}
    request=requests.delete(delete_url, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="FAQ item deleted successfully"
    assert FAQ.query.filter_by(ticket_id=2).first() is None
```

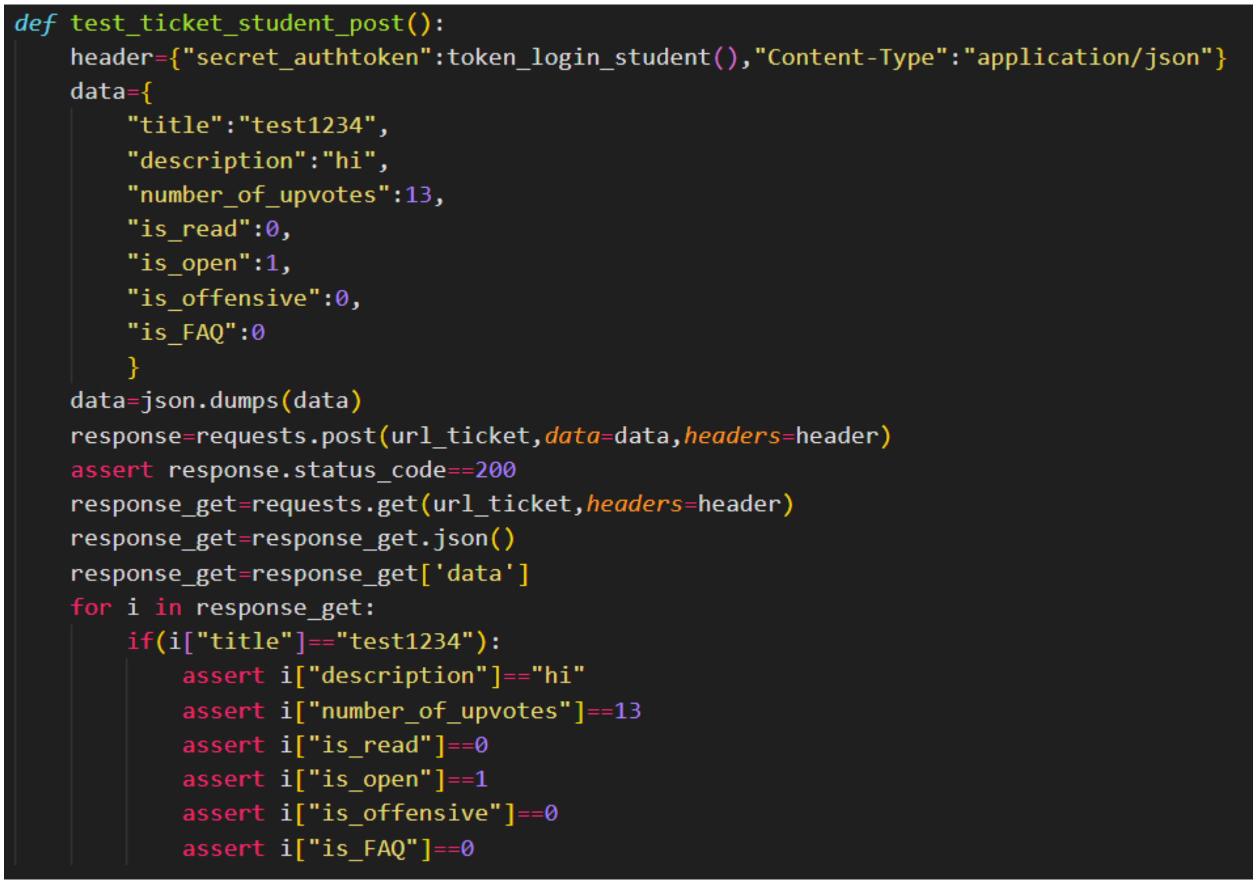
Creating a new ticket by student

Page being tested: http://127.0.0.1:5000/api/ticket
Inputs:
- Request Method: POST
- JSON: { "title":"test1234", "description":"hi", "number_of_upvotes":13, "is_read":0, "is_open":1, "is_offensive":0, "is_FAQ":0 }
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket created successfully'}

Actual Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket created successfully'}

Result: Success



```python
def test_ticket_student_post():
    header={"secret_authtoken":token_login_student(),"Content-Type":"application/json"}
    data={
        "title":"test1234",
        "description":"hi",
        "number_of_upvotes":13,
        "is_read":0,
        "is_open":1,
        "is_offensive":0,
        "is_FAQ":0
        }
    data=json.dumps(data)
    response=requests.post(url_ticket,data=data,headers=header)
    assert response.status_code==200
    response_get=requests.get(url_ticket,headers=header)
    response_get=response_get.json()
    response_get=response_get['data']
    for i in response_get:
        if(i["title"]=="test1234"):
            assert i["description"]=="hi"
            assert i["number_of_upvotes"]==13
            assert i["is_read"]==0
            assert i["is_open"]==1
            assert i["is_offensive"]==0
            assert i["is_FAQ"]==0
```

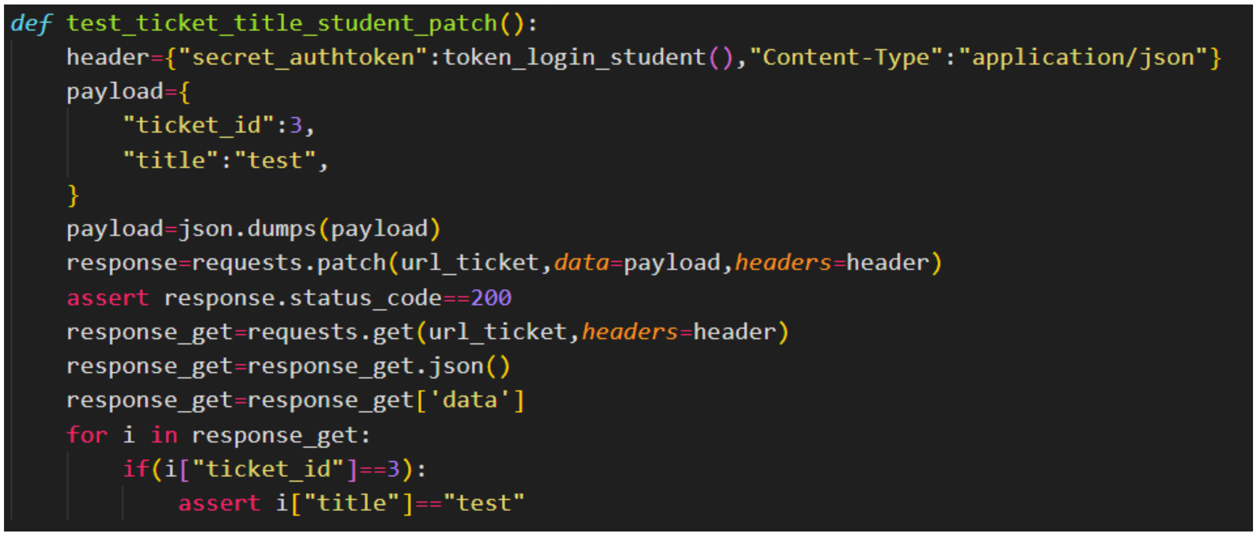
Editing a ticket by student

Page being tested: http://127.0.0.1:5000/api/ticket
Inputs:
- Request Method: PATCH
- JSON: { "ticket_id":3,"title":"test" }
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket updated successfully'}

Actual Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket updated successfully'}

Result: Success



```python
def test_ticket_title_student_patch():
    header={"secret_authtoken":token_login_student(),"Content-Type":"application/json"}
    payload={
        "ticket_id":3,
        "title":"test",
    }
    payload=json.dumps(payload)
    response=requests.patch(url_ticket,data=payload,headers=header)
    assert response.status_code==200
    response_get=requests.get(url_ticket,headers=header)
    response_get=response_get.json()
    response_get=response_get['data']
    for i in response_get:
        if(i["ticket_id"]==3):
            assert i["title"]=="test"
```

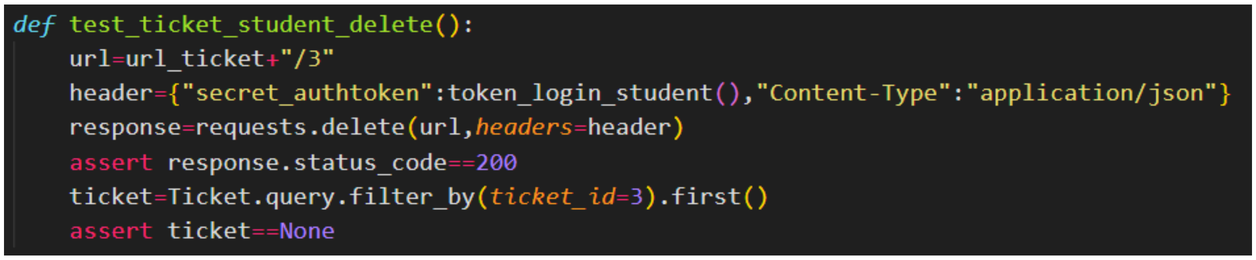
Deleting a ticket by student if resolved

Page being tested: http://127.0.0.1:5000/api/ticket/3
Inputs:
- Request Method: DELETE
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket deleted successfully'}

Actual Output:
- HTTP Status Code: 200
- JSON: {'message':'Ticket deleted successfully'}

Result: Success



```python
def test_ticket_student_delete():
    url=url_ticket+"/3"
    header={"secret_authtoken":token_login_student(),"Content-Type":"application/json"}
    response=requests.delete(url,headers=header)
    assert response.status_code==200
    ticket=Ticket.query.filter_by(ticket_id=3).first()
    assert ticket==None
```

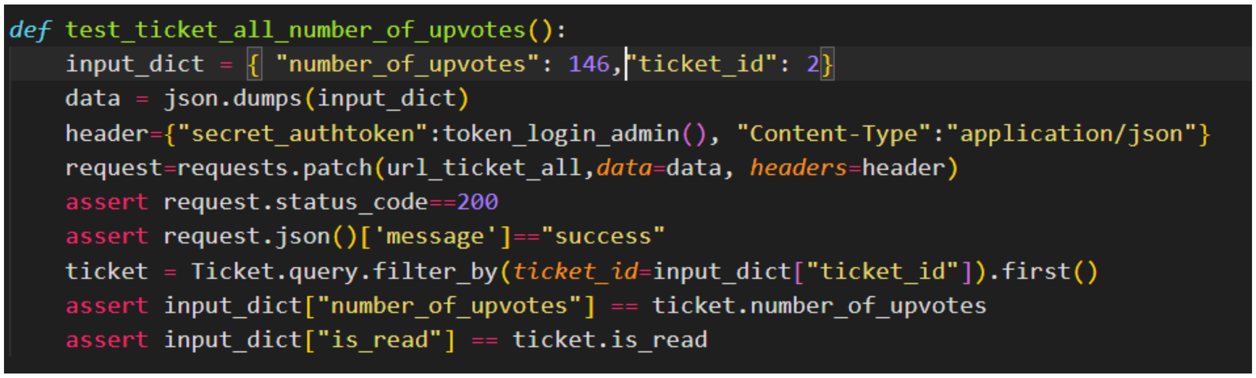
Upvoting ticket “+1” existing tickets created by other students

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- JSON: { "ticket_id":2,"number_of_upvotes":146 }
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Result: Success



```
def test_ticket_all_number_of_upvotes():
    input_dict = { "number_of_upvotes": 146,"ticket_id": 2}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_admin(), "Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="success"
    ticket = Ticket.query.filter_by(ticket_id=input_dict["ticket_id"]).first()
    assert input_dict["number_of_upvotes"] == ticket.number_of_upvotes
    assert input_dict["is_read"] == ticket.is_read
```

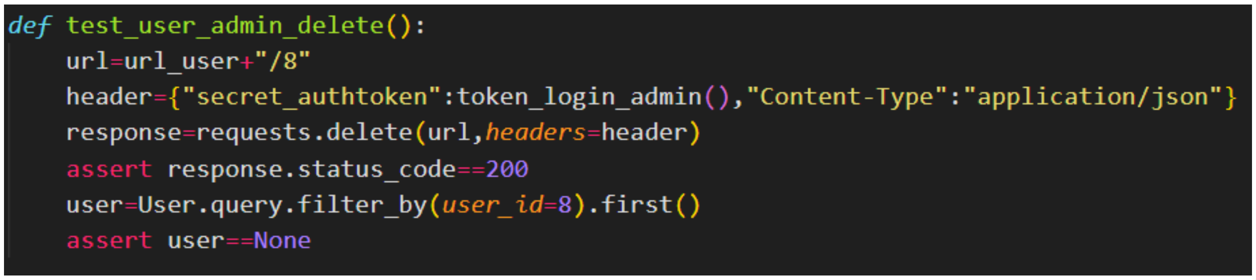
Remove a particular user

Page being tested: http://127.0.0.1:5000/api/user/3
Inputs:
- Request Method: DELETE
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {'message':'User deleted successfully'}

Actual Output:
- HTTP Status Code: 200
- JSON: {'message':'User deleted successfully'}

Result: Success



```
def test_user_admin_delete():
    url=url_user+"/8"
    header={"secret_authtoken":token_login_admin(),"Content-Type":"application/json"}
    response=requests.delete(url,headers=header)
    assert response.status_code==200
    user=User.query.filter_by(user_id=8).first()
    assert user==None
```

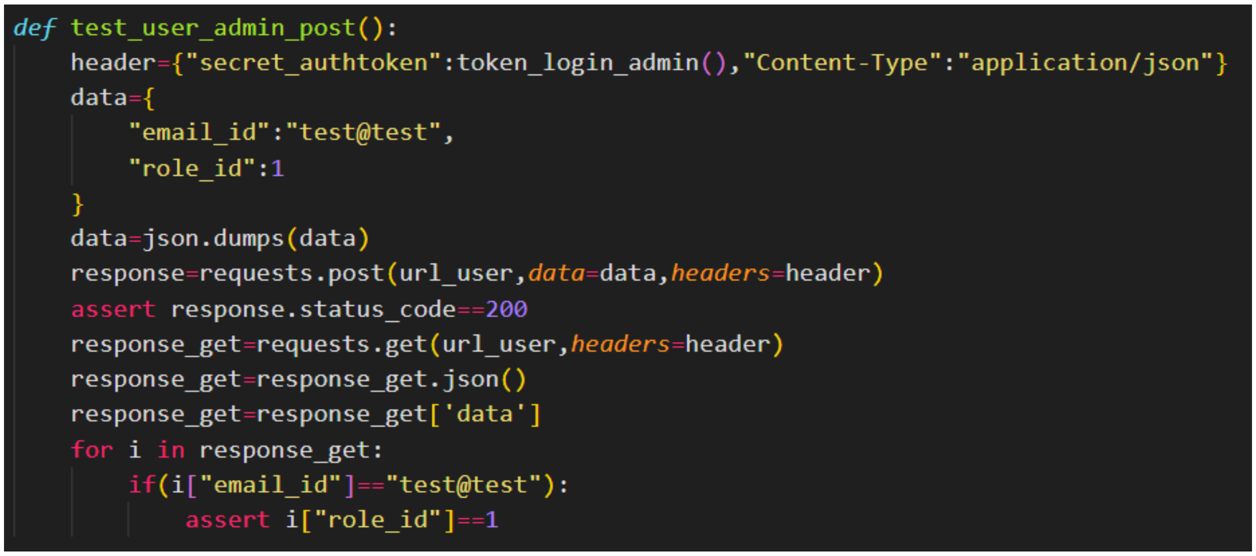
Add a particular user according to role by admin / manager

Page being tested: http://127.0.0.1:5000/api/user
Inputs:
- Request Method: POST
- JSON: {“email_id”:”test@test”,”role_id”:1}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {'message':'User created successfully'}

Actual Output:
- HTTP Status Code: 200
- JSON: {'message':'User created successfully'}

Result: Success



```python
def test_user_admin_post():
    header={"secret_authtoken":token_login_admin(),"Content-Type":"application/json"}
    data={
        "email_id":"test@test",
        "role_id":1
    }
    data=json.dumps(data)
    response=requests.post(url_user,data=data,headers=header)
    assert response.status_code==200
    response_get=requests.get(url_user,headers=header)
    response_get=response_get.json()
    response_get=response_get['data']
    for i in response_get:
        if(i["email_id"]=="test@test"):
            assert i["role_id"]==1
```

An existing ticket is marked closed/resolved by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_open": true, "ticket_id": 1}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Result: Success

```python
def test_ticket_all_patch():
    input_dict = { "is_open": True,"ticket_id": 1}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="success"
    ticket =
Ticket.query.filter_by(ticket_id=input_dict["ticket_id"]).first()
    assert input_dict["number_of_upvotes"] == ticket.number_of_upvotes
    assert input_dict["is_read"] == ticket.is_read
```

A non-existing ticket is attempted to be marked closed/resolved by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_open": true, "ticket_id": 10000}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Actual Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Result: Success

```python
def test_ticket_all_patch_ticket_not_found():
    input_dict = { "is_open": True,"ticket_id": 10000}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==404
    assert request.json()['message']=="There is no such ticket by that ID"
```

An existing ticket is suggested for FAQ by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_FAQ": true, "ticket_id": 1}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Result: Success

```python
def test_ticket_all_patch():
    input_dict = { "is_FAQ": True,"ticket_id": 1}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="success"
    ticket =
Ticket.query.filter_by(ticket_id=input_dict["ticket_id"]).first()
    assert input_dict["number_of_upvotes"] == ticket.number_of_upvotes
    assert input_dict["is_read"] == ticket.is_read
```

A non-existing ticket is attempted to be suggested for FAQ by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_FAQ": true, "ticket_id": 10000}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Actual Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Result: Success

```
def test_ticket_all_patch_ticket_not_found():
    input_dict = { "is_FAQ": True,"ticket_id": 10000}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==404
    assert request.json()['message']=="There is no such ticket by that ID"
```

An existing ticket is marked as offensive by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_offensive": true, "ticket_id": 1}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“message”: “success”}

Result: Success

```
def test_ticket_all_patch():
    input_dict = { "is_offensive": True,"ticket_id": 1}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==200
    assert request.json()['message']=="success"
    ticket =
Ticket.query.filter_by(ticket_id=input_dict["ticket_id"]).first()
    assert input_dict["number_of_upvotes"] == ticket.number_of_upvotes
    assert input_dict["is_read"] == ticket.is_read
```

A non-existing ticket is attempted to be marked as offensive by support agent

Page being tested: http://127.0.0.1:5000/api/ticketAll
Page being tested: http://127.0.0.1:5000/api/ticketAll
Inputs:
- Request Method: PATCH
- Json body: { "is_offensive": true, "ticket_id": 10000}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Actual Output:
- HTTP Status Code: 404
- JSON: {“message”: “There is no such ticket by that ID”}

Result: Success

```
def test_ticket_all_patch_ticket_not_found():
    input_dict = { "is_offensive": True,"ticket_id": 10000}
    data = json.dumps(input_dict)
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    request=requests.patch(url_ticket_all,data=data, headers=header)
    assert request.status_code==404
    assert request.json()['message']=="There is no such ticket by that ID"
```

An existing ticket already marked as offensive is forwarded to admin by support agent

Page being tested: http://127.0.0.1:5000/api/flaggedPosts
Inputs:
- Request Method: POST
- Json body: { "creator_id": 1, "ticket_id": 1, “flagger_id”: 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON: {“status”: “success”}

Actual Output:
- HTTP Status Code: 200
- JSON: {“status”: “success”}

Result: Success

```
def test_post_flaggedPost():
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    input_dict = { "ticket_id": 1, "creator_id": 1,"flagger_id": 2 }
    data = json.dumps(input_dict)
    request=requests.post(url = url_flaggedPosts, headers=header, data =
data)
    response = request.json()
    assert request.status_code == 200
    assert response["status"] == "success"
    header2 = {"secret_authtoken":token_login_admin(),
"Content-Type":"application/json"}
    request2 = requests.get(url = url_flaggedPosts, headers=header2)
    response2 = request2.json()
    for item in response2["data"]:
        if item["ticket_id"] == input_dict["ticket_id"]:
            assert item["creator_id"] == input_dict["creator_id"]
            assert item["flagger_id"] == input_dict["flagger_id"]
```

A non existing ticket not already marked as offensive is tried to be forwarded to admin by support agent

Page being tested: http://127.0.0.1:5000/api/flaggedPosts

Inputs:
- Request Method: POST
- Json body: { "creator_id": 1, "ticket_id": 10000, “flagger_id”: 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 403
- JSON: {“message”: “The referenced ticket is not created by the referenced person/the ticket doesn’t exist in the first place.”}

Actual Output:
- HTTP Status Code: 403
- JSON: {“message”: “The referenced ticket is not created by the referenced person/the ticket doesn’t exist in the first place.”}

Result: Success

```
def test_post_flaggedPost_wrong_ticket_id():
    header={"secret_authtoken":token_login_support_agent(),
"Content-Type":"application/json"}
    input_dict = { "ticket_id": 10000, "creator_id": 1,"flagger_id": 2 }
    data = json.dumps(input_dict)
    request=requests.post(url = url_flaggedPosts, headers=header, data =
data)
    response = request.json()
    assert request.status_code == 403
    assert response["message"] == "The referenced ticket is not created by
the referenced person/ the ticket doesn't exist in the first place."
```

Manager obtaining resolution times of existing tickets

Page being tested: http://127.0.0.1:5000/api/getResolutionTimes
Inputs:
- Request Method: POST
- Json body: { "ticket_id": 2}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 200
- JSON:
```
{
 "data": [
  {
   "creation_time": "Fri, 10 Mar 2023 06:36:58 GMT",
   "days": 2,
   "microseconds": 583678,
   "resolution_time_datetime_format": "2 days, 21:40:12.583678",
   "response_time": "Fri, 10 Mar 2023 06:36:58 GMT",
   "seconds": 78012,
   "ticket_id": 2
  }
 ],
 "status": "success"
}
```

Actual Output:
- HTTP Status Code: 200
- JSON:
```
{
 "data": [
  {
   "creation_time": "Fri, 10 Mar 2023 06:36:58 GMT",
   "days": 2,
   "microseconds": 583678,
   "resolution_time_datetime_format": "2 days, 21:40:12.583678",
   "response_time": "Fri, 10 Mar 2023 06:36:58 GMT",
        "seconds": 78012,
        "ticket_id": 2
      }
    ],
    "status": "success"
  }
```

-

Result: Success

```python
def test_getResolutionTimes_post():
    #Only checks if days, seconds, microseconds and ticket IDs match
    header={"secret_authtoken":token_login_manager(),
"Content-Type":"application/json"}
    input_dict = {"ticket_id": 2}
    data = json.dumps(input_dict)
    request=requests.post(url = url_getResolutionTimes,data = data,
headers=header)
    response = request.json()
    assert request.status_code == 200
    if isinstance(input_dict["ticket_id"], int):
        responses = Response.query.filter_by(ticket_id =
input_dict["ticket_id"]).all()
        responses = list(responses)
        ticket = Ticket.query.filter_by(ticket_id =
input_dict["ticket_id"]).first()
        a = {}
        response_times = []
        for thing in responses:
            if isinstance(thing.response_timestamp, datetime):
                #print("Here 1")
                response_times.append(thing.response_timestamp)
            elif isinstance(thing.response_timestamp, str):
                #print("Here 2")

response_times.append(datetime.strptime(thing.response_timestamp,'%Y-%m-%d
%H:%M:%S.%f'))
            response_time = max(response_times)
            a["creation_time"] = None
            if isinstance(ticket.creation_date, str):
                a["creation_time"] =
datetime.strptime(ticket.creation_date, '%Y-%m-%d %H:%M:%S.%f')
            elif isinstance(ticket.creation_date, datetime):
                a["creation_time"] = ticket.creation_date
            a["response_time"] = response_time
            a["resolution_time_datetime_format"] = a["response_time"] -
a["creation_time"]
            a["days"] = a["resolution_time_datetime_format"].days
            a["seconds"] = a["resolution_time_datetime_format"].seconds
            a["microseconds"] =
a["resolution_time_datetime_format"].microseconds
            a["resolution_time_datetime_format"] =
str(a["resolution_time_datetime_format"])
            a["creation_time"] = a["creation_time"]
            a["ticket_id"] = input_dict["ticket_id"]
            a["response_time"] = None
            a["resolution_time_datetime_format"] = None
            a["creation_time"] = None
        d = response["data"]
        for keys in a:
            if a[keys] is not None:
                assert a[keys] == d[keys]
    elif isinstance(input_dict["ticket_id"], list):
        data = []
        for item in input_dict["ticket_id"]:
            d = {}
            ticket = None
            ticket = Ticket.query.filter_by(ticket_id = item).first()
            if ticket is None:
                continue
            if isinstance(ticket.creation_date, str):
                d["creation_time"] =
datetime.strptime(ticket.creation_date, '%Y-%m-%d %H:%M:%S.%f')
            elif isinstance(ticket.creation_date, datetime):
                d["creation_time"] = ticket.creation_date
            responses = Response.query.filter_by(ticket_id = item).all()
            if ticket.is_open == False:
                responses = list(responses)
                response_times = []
                for thing in responses:
                    if isinstance(thing.response_timestamp, datetime):
                        response_times.append(thing.response_timestamp)
                    elif isinstance(thing.response_timestamp, str):
                        #print("Here 2")
```

```
response_times.append(datetime.strptime(thing.response_timestamp,'%Y-%m-%d
%H:%M:%S.%f'))

                response_time = max(response_times)
                d["response_time"] = response_time
                d["resolution_time_datetime_format"] = d["response_time"]
- d["creation_time"]
                d["days"] = d["resolution_time_datetime_format"].days
                d["seconds"] =
d["resolution_time_datetime_format"].seconds
                d["microseconds"] =
d["resolution_time_datetime_format"].microseconds
                d["response_time"] = d["response_time"]
                d["resolution_time_datetime_format"] =
str(d["resolution_time_datetime_format"])
                d["creation_time"] = d["creation_time"]
                d["ticket_id"] = item
                d["response_time"] = None
                d["resolution_time_datetime_format"] = None
                d["creation_time"] = None
                data.append(d)
        x = response["data"]
        for item in x:
            for thing in data:
                if item["ticket_id"] == thing["ticket_id"]:
                    for keys in thing:
                        if thing[keys] is not None:
                            assert thing[keys] == item[keys]
```

Manager trying to obtain resolution times of non-existing tickets

Page being tested: http://127.0.0.1:5000/api/getResolutionTimes
Inputs:
- Request Method: POST
- Json body: { "ticket_id": 1000}
- Header: secret_authtoken: abcxyz

Expected Output:
- HTTP Status Code: 404
- JSON: {“message”: “No such ticket exists by the given ticket ID.”}

Actual Output:

- HTTP Status Code: 404
- JSON: {“message”: “No such ticket exists by the given ticket ID.”}

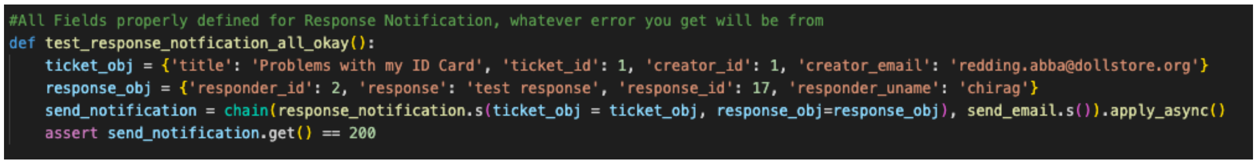
Result: Success

```
def test_getResolutionTimes_post_wrong_ticket_id():
    header={"secret_authtoken":token_login_manager(),
"Content-Type":"application/json"}
    input_dict = {"ticket_id": 1000}
    data = json.dumps(input_dict)
    request=requests.post(url = url_getResolutionTimes,data = data,
headers=header)
    response = request.json()
    assert request.status_code == 404
    assert response["message"] == "No such ticket exists by the given
ticket ID."
```

Unit Tests on Celery tasks

Email Notification for a new response sent when inputs properly supplied

Page being tested: Not an API Endpoint. This is a celery task triggered internally when a new response is added
Inputs: (ticket_obj = {'title': 'Problems with my ID Card', 'ticket_id': 1, 'creator_id': 1, 'creator_email': 'redding.abba@dollstore.org'}, response_obj = {'responder_id': 2, 'response': 'test response', 'response_id': 17, 'responder_uname': 'chirag'})
Expected Output: 200
Actual Output: 200
Result: Success

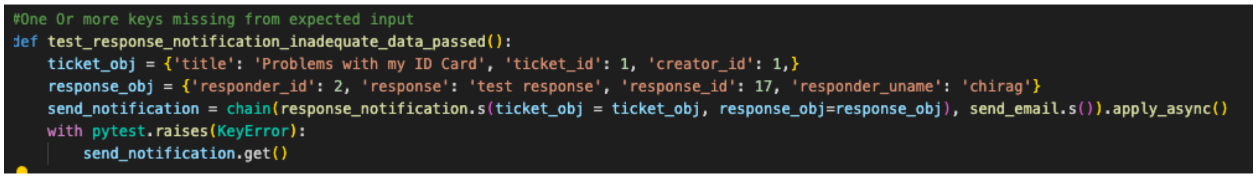


```
#All Fields properly defined for Response Notification, whatever error you get will be from
def test_response_notfication_all_okay():
    ticket_obj = {'title': 'Problems with my ID Card', 'ticket_id': 1, 'creator_id': 1, 'creator_email': 'redding.abba@dollstore.org'}
    response_obj = {'responder_id': 2, 'response': 'test response', 'response_id': 17, 'responder_uname': 'chirag'}
    send_notification = chain(response_notification.s(ticket_obj = ticket_obj, response_obj=response_obj), send_email.s()).apply_async()
    assert send_notification.get() == 200
```

Email Notification for a new response not sent when inputs improperly specified

Page being tested: Not an API Endpoint. This is a celery task triggered internally when a new response is added
Inputs: (ticket_obj = {'title': 'Problems with my ID Card', 'ticket_id': 1, 'creator_id': 1}, response_obj = {'responder_id': 2, 'response': 'test response', 'response_id': 17, 'responder_uname': 'chirag'})

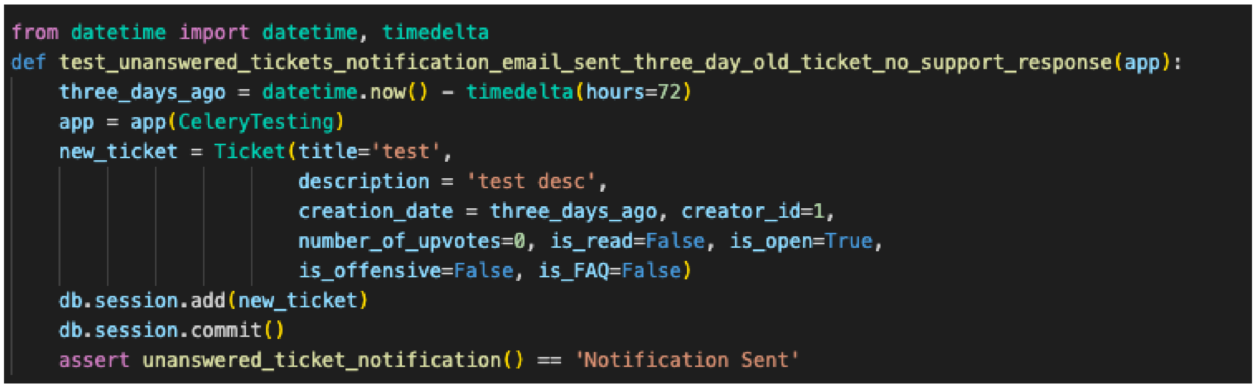
Expected Output: KeyError
Actual Output: KeyError
Result: Success



```python
#One Or more keys missing from expected input
def test_response_notification_inadequate_data_passed():
    ticket_obj = {'title': 'Problems with my ID Card', 'ticket_id': 1, 'creator_id': 1,}
    response_obj = {'responder_id': 2, 'response': 'test response', 'response_id': 17, 'responder_uname': 'chirag'}
    send_notification = chain(response_notification.s(ticket_obj = ticket_obj, response_obj=response_obj), send_email.s()).apply_async()
    with pytest.raises(KeyError):
        send_notification.get()
```

Manager is sent an email notification for open tickets created over 72 hours ago that are still unanswered

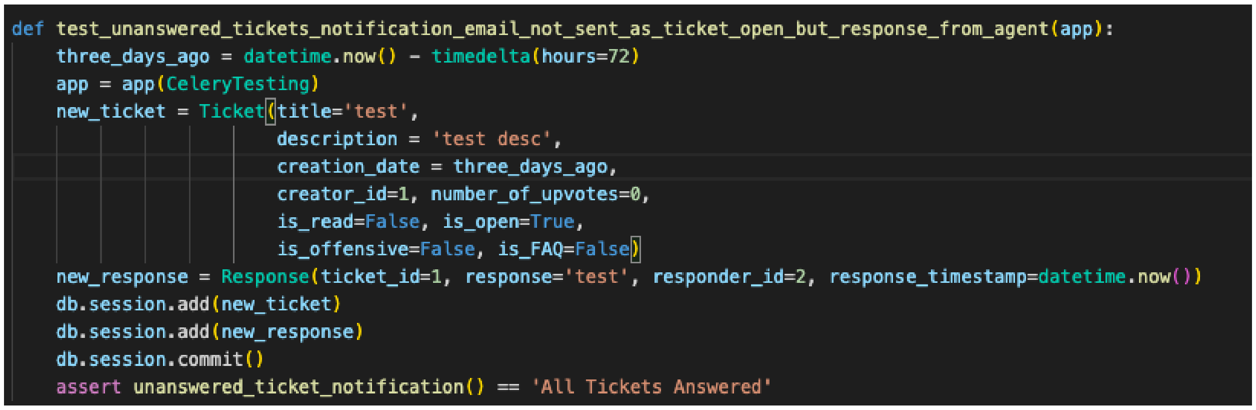
Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: Notification Sent
Actual Output: Notification Sent
Result: Success



```python
from datetime import datetime, timedelta
def test_unanswered_tickets_notification_email_sent_three_day_old_ticket_no_support_response(app):
    three_days_ago = datetime.now() - timedelta(hours=72)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = three_days_ago, creator_id=1,
                        number_of_upvotes=0, is_read=False, is_open=True,
                        is_offensive=False, is_FAQ=False)
    db.session.add(new_ticket)
    db.session.commit()
    assert unanswered_ticket_notification() == 'Notification Sent'
```

Manager is not sent an email notification for open tickets created over 72 hours ago that have been responded to by a support agent

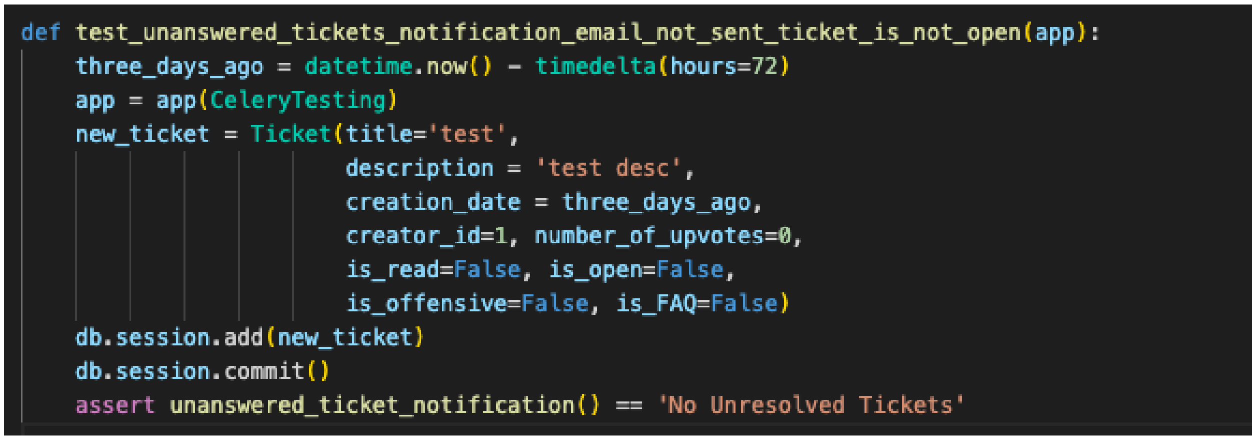
Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: All Tickets Answered
Actual Output: All Tickets Answered
Result: Success



```python
def test_unanswered_tickets_notification_email_not_sent_as_ticket_open_but_response_from_agent(app):
    three_days_ago = datetime.now() - timedelta(hours=72)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = three_days_ago,
                        creator_id=1, number_of_upvotes=0,
                        is_read=False, is_open=True,
                        is_offensive=False, is_FAQ=False)
    new_response = Response(ticket_id=1, response='test', responder_id=2, response_timestamp=datetime.now())
    db.session.add(new_ticket)
    db.session.add(new_response)
    db.session.commit()
    assert unanswered_ticket_notification() == 'All Tickets Answered'
```

Manager is not sent an email notification as all tickets are marked as closed

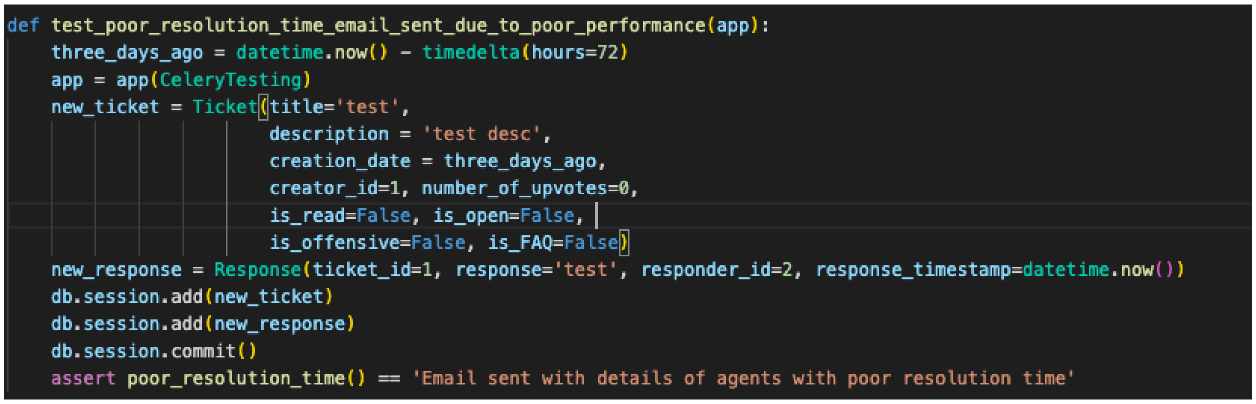
Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: No Unresolved Tickets
Actual Output: No Unresolved Tickets
Result: Success



```python
def test_unanswered_tickets_notification_email_not_sent_ticket_is_not_open(app):
    three_days_ago = datetime.now() - timedelta(hours=72)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = three_days_ago,
                        creator_id=1, number_of_upvotes=0,
                        is_read=False, is_open=False,
                        is_offensive=False, is_FAQ=False)
    db.session.add(new_ticket)
    db.session.commit()
    assert unanswered_ticket_notification() == 'No Unresolved Tickets'
```

Manager is sent an email report of all the agents who have a resolution time of over 48 hours for tickets created in the past 30 days

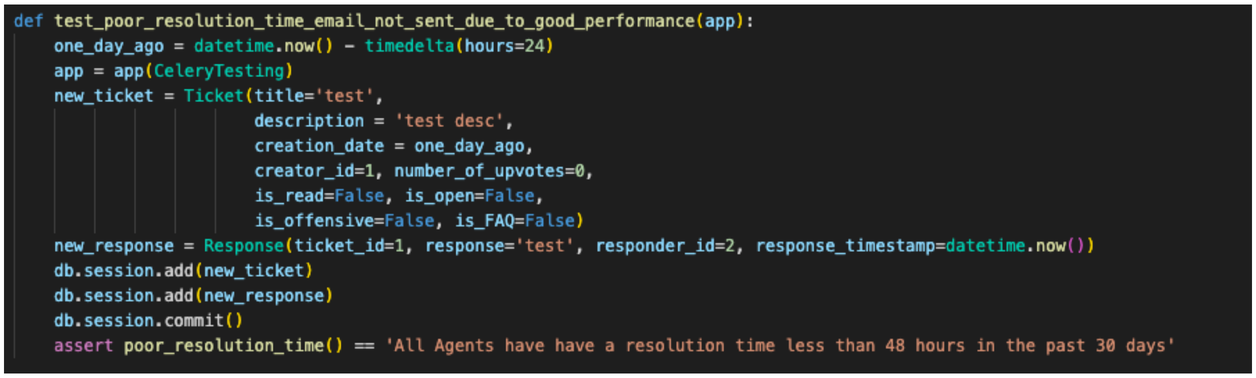
Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: Email sent with details of agents with poor resolution time
Actual Output: Email sent with details of agents with poor resolution time
Result: Success



```python
def test_poor_resolution_time_email_sent_due_to_poor_performance(app):
    three_days_ago = datetime.now() - timedelta(hours=72)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = three_days_ago,
                        creator_id=1, number_of_upvotes=0,
                        is_read=False, is_open=False,
                        is_offensive=False, is_FAQ=False)
    new_response = Response(ticket_id=1, response='test', responder_id=2, response_timestamp=datetime.now())
    db.session.add(new_ticket)
    db.session.add(new_response)
    db.session.commit()
    assert poor_resolution_time() == 'Email sent with details of agents with poor resolution time'
```

Manager is not sent an email report since all agents have a resolution time under 48 hours for tickets created in the past 30 days

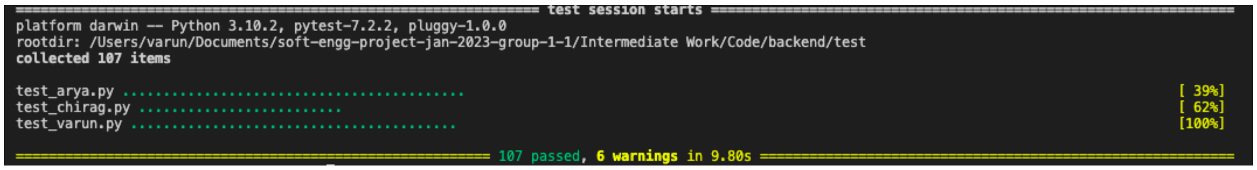
Page being tested: Not an API Endpoint. This is a celery task triggered internally by cron
Inputs: None (The output of the task is dependent on the state of the db not by any inputs)
Expected Output: All Agents have have a resolution time less than 48 hours in the past 30 days
Actual Output: All Agents have have a resolution time less than 48 hours in the past 30 days
Result: Success



```python
def test_poor_resolution_time_email_not_sent_due_to_good_performance(app):
    one_day_ago = datetime.now() - timedelta(hours=24)
    app = app(CeleryTesting)
    new_ticket = Ticket(title='test',
                        description = 'test desc',
                        creation_date = one_day_ago,
                        creator_id=1, number_of_upvotes=0,
                        is_read=False, is_open=False,
                        is_offensive=False, is_FAQ=False)
    new_response = Response(ticket_id=1, response='test', responder_id=2, response_timestamp=datetime.now())
    db.session.add(new_ticket)
    db.session.add(new_response)
    db.session.commit()
    assert poor_resolution_time() == 'All Agents have have a resolution time less than 48 hours in the past 30 days'
```

Test Results

Apart from the tests explicitly mentioned above, we had written many more tests. All of the tests passed. There were 6 warnings pertaining to soon to be deprecated methods used by sqlalchemy/celery. All the tests reside in the following directory: .\Intermediate Work\Code\backend\test



```
================================================================= test session starts =================================================================
platform darwin -- Python 3.10.2, pytest-7.2.2, pluggy-1.0.0
rootdir: /Users/varun/Documents/soft-engg-project-jan-2023-group-1-1/Intermediate Work/Code/backend/test
collected 107 items

test_arya.py ..........................................                                                                                         [ 39%]
test_chirag.py .........................                                                                                                        [ 62%]
test_varun.py ........................................                                                                                          [100%]

========================================================== 107 passed, 6 warnings in 9.80s ===========================================================
```