DAILY SPORTS QUOTE APP DOCUMENTATION
ZUHAL DİDEM AYTAÇ

1. INTRODUCTION

The Daily Quote API is an API that provides you daily quotes about sports, in English. It uses the **Quotes Rest API** (https://quotes.rest/) as the remote API. The remote API provides a different quote for each date (quote of the day functionality). The Daily Quote API filters the quote category as sports and the quote language as English by making a get request to a remote API (see section 1.1)

The Daily Quote API is written in Django and is connected with a PostgreSQL database. The Daily Quote API is a RESTful API.

Each day's quote, author and date are saved to the database if it has not been saved before. The Daily Quote API adds a rating functionality to the remote API. It supports GET and POST methods. When a user makes a GET request to the API, they see the average rating of the quote and the number of ratings made so far, along with the quote and the author. Users can rate the quote with a point of their wish. Allowed points are 0,1,2,3,4 and 5. When a POST request is made with a valid point, the total points of the quote is incremented by the given point and the total ratings of the quote is incremented by one. These values are updated in the database. Then, the new average value and number of ratings are returned in the response.

The front-end is implemented using HTML. The HTML forms allows the API to get only valid inputs when taking a POST request. (see section 5) Unit tests are provided for the API. The details of the tests and functionality they test are explained in the corresponding section (see section 6).

1.1 Remote API Call

The Daily Quote API filters the quote category as sports and the quote language as English by making a GET request to a remote API on `https://quotes.rest/qod?category=sports&language=en`


```
{
    "success": { "total": 1},
    "contents": {
        "quotes": [
            {
                "quote": "Bodybuilding is much like any other sport. To be successful, you must
dedicate yourself 100% to your training, diet and mental approach.",
                "length": "136",
                "author": "Arnold Schwarzenegger",
                "tags": ["diet", "sports"],
                "category": "sports",
                "language": "en",
                "date": "2021-06-08",
                "permalink": "https://theysaidso.com/quote/arnold-schwarzenegger-bodybuilding-is-much-
like-any-other-sport-to-be-successful",
                "id": "LflBg5X1AMVlVOsgoso6aweF",
                "background": "https://theysaidso.com/img/qod/qod-sports.jpg",
                "title": "Sports Quote of the day"
            }
        ]
    },
    "baseurl": "https://theysaidso.com",
    "copyright": {"year": 2023, "url": "https://theysaidso.com"
    }
}
```

2. CODE DOCUMENTATION

models.py

```
class DailyQuote(models.Model):
    quote_text = models.CharField(max_length=1000)  # quote text
    author = models.CharField(max_length=100)       # author of the quote
    date = models.DateField(unique=True)            # date of the quote
    points = models.IntegerField()                  # total points given
    ratings = models.IntegerField()                 # total ratings made
```

DailyQuote is a django.db model that the app uses. The meanings of the fields are explained in the comments. The model is registered in admin.py

serializer.py

```
class DailyQuoteSerializer(serializers.ModelSerializer):
    class Meta:
        model = DailyQuote
        fields = ['quote_text', 'author', 'date', 'points', 'ratings']
```

DailyQuoteSerializer is a serializer (inheriting from rest_framework.serializers) for the DailyQuote class.

2.1 Handling the API requests

views.py

```
@api_view(['GET', 'POST'])
def show_quote(request):
    if request.method == 'GET':
        res = quote().data
        return render(request, 'dailyQuote/base.html', res)
    else:
        res = rate(request.POST.get("Points", '')).data
        return render(request, 'dailyQuote/base.html', res)
```

```
@api_view(['GET', 'POST'])
def quote_api(request):
    if request.method == 'GET':
        res = quote()
        return res
    else:
        res = rate(request.POST.get("Points", ''))
        return res
```

There are two api_views. Both supports GET and POST methods and provide the same functionality. The only difference is that the show_quote function renders the response data with the HTML, whereas the quote_api function provides a plain api response.

The GET method calls the quote() function and the POST method calls the rate() function with the input of “Points” in the request body. These functions are explained below.

2.2 Introducing API Functionalities

Below, the quote() function is given in 2 parts. The quote function basically returns a response with a body and a status. It is the code that runs when a GET request is made.

views.py

```
def quote():
    today = str(datetime.utcnow().date())
    points, ratings, value = 0, 0 , 0

    # get the quote with todays date from the database, if exists
    try:
        quote_in_db = DailyQuoteSerializer(DailyQuote.objects.filter(date=today)[0]).data
    except:
        quote_in_db = None

    # if the quote of today has been saved before, take the points and ratings
    if quote_in_db is not None:
        points = quote_in_db["points"]
        ratings = quote_in_db["ratings"]

    # send get request to the remote api
    api = 'https://quotes.rest/qod?category=sports&language=en'
    response = requests.get(api)

    # if the response is successful, process the response of the remote api
    if response.status_code == 200:
        res = response.json()["contents"]["quotes"][0]
        res["quote_text"] = res['quote']
    # due to request limit per hour, the response may return 429, too many requests
    # if so, process the response of the database
    else:
        res = quote_in_db
```

The general logical flow is as follows: The aim is to provide the quote of the day, along with its points and number of ratings. So, it first checks the database and tries to get the row with today's date. If a row is found, this means somebody has made a GET request before and the quote is saved to the database. Maybe some POST request has been made, so the points and number of ratings are read from the database response. Then, a request to the remote API is send.

```
    # make sure that the res is not None
    # it can be None if the remote api did not return 200 and the database is empty
    if res is not None:
        quote_text = res["quote_text"]
        author = res["author"]
        date = res["date"]
        # if there is not a row in the database for today, create a new object and save it to the db
        if quote_in_db is None:
            new_quote = DailyQuote(quote_text=quote_text, author=author, date=date,
                                   points=0, ratings=0)
            new_quote.save()
            status = s.HTTP_201_CREATED

        # if there exists a row in the database for today, then calculate the value to be shown
        else:
            if not ratings == 0:
                value = points * 1.0 / ratings
                value = "{:.1f}".format(value)
            status = s.HTTP_200_OK

        # send quote text, author, calculated value and number of ratings as the response
        dictionary = {"quote_text": quote_text, "author": author, "value": value, "ratings": ratings}

    # if res is None (the remote api did not return 200 and the database is empty), return 404
    else:
        dictionary = False
        status = s.HTTP_404_NOT_FOUND

    return Response({"response": dictionary}, status=status)
```

The remote API returns the sports quote of the day along with many other information, but the app just needs the quote and the author. Sometimes the remote API does not return due to too many requests. If so, it uses the database's return value to reach the quote and author. Either way, unless a very unlikely error occurs, the code reaches a quote and author. If the database does not contain a row for today, then the quote has 0 points and 0 ratings. It is saved to the database. If so, the return status becomes 201. If a new row for today hasn't been created, it returns with HTTP 200. If a quote from neither of the database and remote API could be reached, it returns with HTTP 404. If there is a successful return, the response has the keys quote_text, author, value and ratings. Value is the total points divided by number of ratings rounded to have 1 digit after decimal.

views.py

```
def rate(points):

    # make sure that the input is valid, if not, return 400
    if points is None or (not str(points).isdecimal()) or (not int(points) in [0, 1, 2, 3, 4, 5]):
        dictionary = False
        status = s.HTTP_400_BAD_REQUEST
        return Response({"response": dictionary}, status=status)

    today = str(datetime.utcnow().date())
    # if somebody has rated, the quote they liked is today's quote, which is in the database
    try:
        quote_in_db = DailyQuoteSerializer(DailyQuote.objects.filter(date=today)[0]).data
    except:
        quote_in_db = None

    # if we could get the row from the database successfully, process it
    if quote_in_db is not None:
        points = int(points)
        # calculate points, ratings and value
        points_total = quote_in_db["points"] + points
        ratings_total = quote_in_db["ratings"] + 1
        value = points_total * 1.0 / ratings_total
        value = "{:.1f}".format(value)
        dictionary = {"quote_text": quote_in_db["quote_text"],
                      "author": quote_in_db["author"],
                      "value": value,
                      "ratings": ratings_total}
        DailyQuote.objects.filter(date=quote_in_db["date"]).update(points=points_total)
        DailyQuote.objects.filter(date=quote_in_db["date"]).update(ratings=ratings_total)
        status = s.HTTP_200_OK

    # if we could not get the row from the database successfully (unlikely) return 404
    else:
        dictionary = False
        status = s.HTTP_404_NOT_FOUND

    return Response({"response": dictionary}, status=status)
```

Above, the rate() function is given. The quote function basically returns a response with a body and a status. It is the code that runs when a POST request is made. Rows are explained in detail in the comments. The general logical flow is as follows: First the input is validated to be 0,1,2,3,4 or 5. If not, HTTP 400 is returned. If the input is valid, the row with today's date is read from the database. The points and ratings are updated and updated in the database. Then, a success response is returned with the quote, author, value and ratings. If due to some unlikely error, the data could not be read from the database, HTTP 404 is returned.

3. API DOCUMENTATION

Endpoint

```
GET http://localhost:8000/dailyQuote/api
```

This endpoint retrieves the sports quote of today, it's author, the average of ratings and the number of ratings of that quote.

Request Parameters

None

Response

```
Response({"response": <dictionary>}, status=<HTTP status>)
```

Sample Call

```
{
    "response": {
        "quote_text": "Bodybuilding is much like any other sport. To be
successful, you must dedicate yourself 100% to your training, diet and mental
approach.",
        "author": "Arnold Schwarzenegger",
        "value": "5.0",
        "ratings": 1
    }
}
```

This is a successful response with status code 200.

Endpoint

```
POST http://localhost:8000/dailyQuote/api/
```

This endpoint allows rating the quote between 0 and 5. The response is the quote of, it's author, the new average of ratings and the new number of ratings of that quote.

Request Parameters

Points: Taken as form parameter. Stands for the points the user gives to the quote. The API only allows values 0,1,2,3,4 and 5 as legal points.

Response

```
Response({"response": <dictionary>}, status=<HTTP status>)
```

Sample Call (Points = 4)

```
{
    "response": {
        "quote_text": "Bodybuilding is much like any other sport. To be
successful, you must dedicate yourself 100% to your training, diet and mental
approach.",
        "author": "Arnold Schwarzenegger",
        "value": "4.5",
        "ratings": 2
    }
}
```

This is a successful response with status code 200.

Sample Call (Points = 6)

```
{
    "response": False
}
```

This is an unsuccessful response with status code 400 as the value is greater than 5.

Sample Call (Points = 'bad_request')

```
{
    "response": False
}
```

This is an unsuccessful response with status code 400 as the value is not an integer.

4. RUNNING THE APP

- Clone the repository
- Create and activate a virtual environment
- Install the requirements
- Set up a postgresql database named group5db
- Run the following command to create the tables in the database:

```
python3 ./manage.py migrate
```

- Run the following command to start the app:

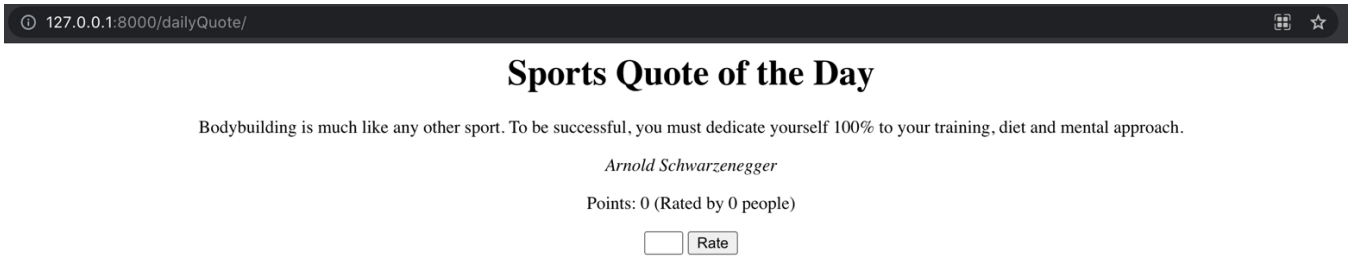
```
python3 ./manage.py runserver
```

- Open the browser (http://127.0.0.1:8000/)
- Enjoy the app
- Quit the app using Ctrl+C
- You can test the app using the following command (see section 6 for test details):

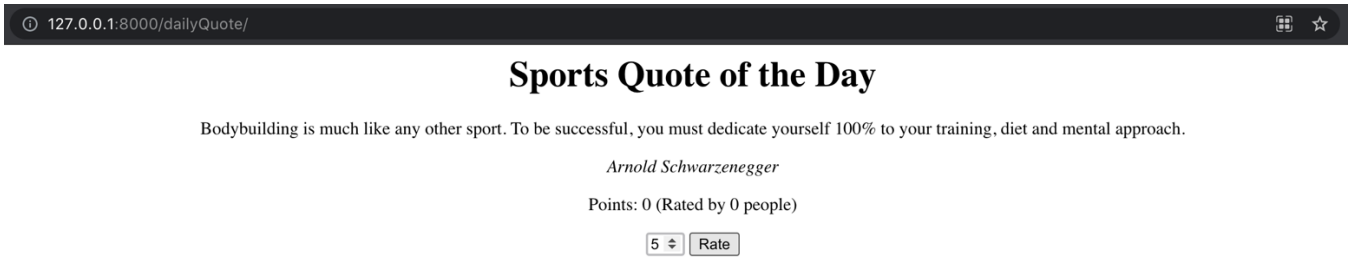
```
python3 ./manage.py test
```

5. THE USER INTERFACE

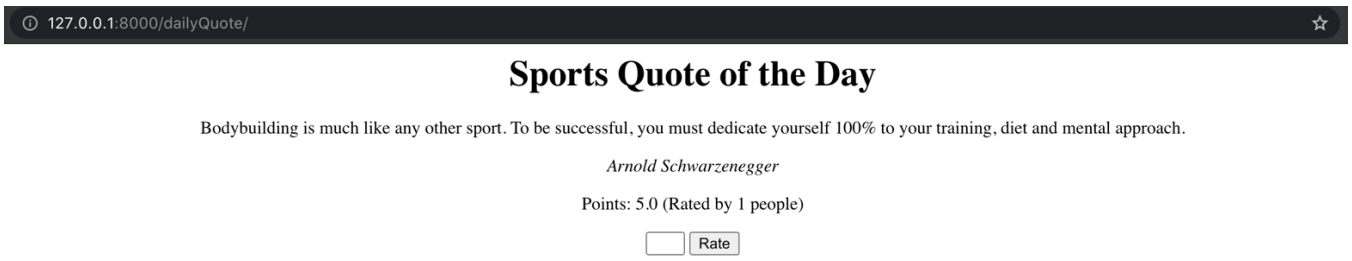
The API functionality can be tested via the user interface or tools like Postman. Below, some screenshots are attached that show the app works logically.



(the sports quote of the day is shown as a result of the GET request)



(the form input is filled)



(when the rate button is clicked, a POST request is made, and the page is updated)

The HTML Code:

```
<!Doctype html5>
<html>
    <head> <title>Group5</title> </head>
    <body>
    <center>
        <p>
            <h1>Sports Quote of the Day</h1>
            {% if response %}
                <p> {{response.quote_text}}</p>
                <p><i>{{response.author}}</i></p>
                <p> Points: {{response.value}} (Rated by {{response.ratings}} people)</p>
                <form id='rateQuote' action="" method='post'>
                    <input type="number" name="Points" required min="0" max="5" step="1">
                    <button type='Rate'>Rate</button>
                </form>
            {% else %}
                <p>Quote not found</p>
            {% endif %}
        </p>
    </center>
    </body>
</html>
```

6. UNIT TESTS

I grouped the tests into 3, test GET request, test unsuccessful POST request and test successful POST request. Note that the remote API allows up to a limit of requests per hour. This does not affect the functionality of the API as we use the quote in database if that happens. However, this causes the tests to fail (returning 404) if that happens. So, the tests should be run making sure that limit is not exceeded.

tests.py

```
  class DailyQuoteTests(APITestCase):

    # test if get method works correctly
    def test_get(self):
        client = APIClient()
        response = client.get("/dailyQuote/api/")
        remote_response = requests.get('https://quotes.rest/qod?category=sports&language=en')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

        # response body containts quote_text, author, value and ratings
        body = response.json()["response"]
        self.assertIn("quote_text", body)
        self.assertIn("author", body)
        self.assertIn("value", body)
        self.assertIn("ratings", body)
        # as the quote is just created, value and ratings are both 0
        self.assertEqual(body["value"], 0)
        self.assertEqual(body["ratings"], 0)

        # the response's quote_text and author should match with remote api's
        remote_body = remote_response.json()["contents"]["quotes"][0]
        self.assertEqual(remote_body['quote'], body['quote_text'])
        self.assertEqual(remote_body['author'], body['author'])
```

This part makes a GET request to both the remote API and the Daily Quote API. After making sure that the Daily Quote API returns successfully with 201, it checks whether it includes the intended keys in the response (quote_text, author, value, ratings). After that it makes sure the value and ratings are both 0 as the object is just created. Then it checks the equality of quote text and author with the remote API.

```
    def test_post(self):
        # first, send a get request to create a quote in the database
        client = APIClient()
        response = client.get("/dailyQuote/api/")
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

        # send a post request
        points_1 = 5
        data = {"Points": points_1}
        client = APIClient()
        response = client.post("/dailyQuote/api/", data)
        self.assertEqual(response.status_code, status.HTTP_200_OK)

        # as only one post request is made, the ratings should equal 1 and the value should equal the
calculated value
        body = response.json()["response"]
        self.assertIn("value", body)
        self.assertIn("ratings", body)
        self.assertEqual(body["value"], "{:.1f}".format(points_1 * 1.0 / 1))
        self.assertEqual(body["ratings"], 1)

        # send a second post request
        points_2 = 3
        data = {"Points": points_2}
        client = APIClient()
        response = client.post("/dailyQuote/api/", data)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
```

```
        # as two post requests are made, the ratings should equal 2 and the value should equal the
calculated average
        body = response.json()["response"]
        self.assertIn("value", body)
        self.assertIn("ratings", body)
        self.assertEqual(body["value"], "{:.1f}".format((points_1 + points_2) * 1.0 / 2))
        self.assertEqual(body["ratings"], 2)
```

Above part checks a successful response of POST request. First, a GET request is made to make sure a row exists in the database. Then, a POST request is made with the request data in desired form. The response's status code is checked to be 200. Then, it asserts the number of ratings in the response is 1 and the value equals the points sent in the request. Then a second POST request is sent. Same checks of responses status and keys are done. This time it asserts the number of ratings became 2 and the value returned is the average of the points sent in the first and second POST requests.

```
    def test_post_invalid_input(self):
        # first, send a get request to create a quote in the database
        client = APIClient()
        response = client.get("/dailyQuote/api/")
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

        # the points can't be greater than 5
        data = {"Points": 10}
        client = APIClient()
        response = client.post("/dailyQuote/api/", data)
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

        # the points can't be lower than 5
        data = {"Points": -1}
        client = APIClient()
        response = client.post("/dailyQuote/api/", data)
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

        # the points can't be real numbers
        data = {"Points": 1.5}
        client = APIClient()
        response = client.post("/dailyQuote/api/", data)
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

        # the points can't be strings
        data = {"Points": 'points'}
        client = APIClient()
        response = client.post("/dailyQuote/api/", data)
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)

        # the request body should contain the key 'Points'
        data = {}
        client = APIClient()
        response = client.post("/dailyQuote/api/", data)
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
```

Above part checks if the invalid responses are handled during POST requests. First, a GET request is made and then 5 different illegal POST requests are made. The request should contain the key "Points". The values allowed for "Points" are integer values 0,1,2,3,4 and 5. Any other value is responsed with 400, Bad Request.