itucsdb Documentation

Release 1.0

Team Name

Dec 11, 2017

Contents

Team itucsdb1741

Members

- Muhammet Uçan

This project provides a API service for generating quotes. Quotes are limited by the number of quotes that are currently exists in database. API provides users two option; they can either send a keyword in order to take a quote generated with provided keyword or they can generate quotes without sending any keywords. If users do not want to use API service they can generate quotes within the web application and use whereever they like.

For more detailed description of API usage please look at the User Guide section.

Contents:

CHAPTER 1

User Guide

1.1 Quote with Keyword

In order to get a quote with given keyword, simply type the keyword in the box and hit generate button. If the quote with given keyword exists in database then result will be successful. If there is no quote with given keyword then rather showing a 404 error or not found error, randomly chosen "404" quote will be displayed.

Below, there is an example of quote generating quote with keyword.

1.1.1 Rating

After generating quote user can rate a quote if the given keyword is related with the generated quote or they can give rating because they like the quote. These rating data will be used later in order to determine whether generated quote is good or not.

1.1.2 Give Feedback

If user want to share opinion about quotes or they want to share what is their thought about that they can share using Give Feedback button. Any type of comments can be send using that button. However in order to send comments users have to log in. Otherwise Give Feedback button will not appear.

1.2 Randomly Chosen Quote

If users don't have any particular keyword, still they can generate quotes. This functionality can be reached from the menu.

1.3 API Service

Users who need a service for generating quotes, can be use this service. In order to use this service they have to create an account using the menu link. Afterwards, with the given API key they can generate thier quotes either randomly or with given keyword.

1.3.1 API Usage

Before making a request user has to create credentials. In order to that, **Auth** menu link should be used. After creating and user account `API KEY` will be given to user.

Base Link

`http://itucsdb1741.mybluemix.net/quote/api/v1.0/`

Authentication

Add following key-pairs to the Header. Basic Auth requires user to send Username and Password.

```
ApiKey = API_KEY_HERE
For Authorization use Basic Auth
```

GET /quote/api/v1.0/random

Below request will return randomly generated quote.

```
GET /quote/api/v1.0/random HTTP/1.1
Host: itucsdb1741.mybluemix.net
ApiKey: API_KEY
Authorization: Basic user-password
```

GET /quote/api/v1.0/{?keyword}

In order to get quote with keyword, it should be added at the and of URL. In the example below quote with `funny` will be returned.

```
GET /quote/api/v1.0/funny?ApiKey=API_KEY_HERE HTTP/1.1
Authorization: Basic user-password
ApiKey: API_KEY_HERE
```

Error Codes and Meanings

Code	Text	Description
400	ApiKey Missing or Wrong	Api Key is wrong
403	Add ApiKey to header	Api Key should be added to Header
403	Unauthorized access	Basic Auth is missing in Header

CHAPTER 2

Developer Guide

2.1 Database Design

The database includes six table. The ‘users’ table is for the users who want to use API service. The ‘users’ table holds the data for username, password and API key.

The ‘categories’ table holds the available categories in the database at that time. The id variable is unique for each category.

The ‘writers’ table holds the name of the the writers and the id variable is unique for each one.

The ‘quotes’ table is the main table for database. It holds the value quotes, votes and rates also two foreign key for categories and writers.

The ‘comments’ table holds the data for comments sent by users. It has four rows, which are id as primary key, user_id for pointing out which user sent the comment, quote_id for referencing for commented quote and finally comment itself.

The ‘user_quotes’ table holds the quotes which sent by user in order to be added to actual quotes database. These quotes should be examined by the admin and after they can be added to actual quote database manually.

2.2 Code

In order to built application below programming languages and libraries are used.

- Python 3.6.2
- PostgreSQL
- Flask

Since application both provides API service and user interface for web client, it can be analyzed with 2 part.

1. Web Client and backend
2. API service

2.2.1 Web Client and backend

```
def get_quote_random():
    SQL = "SELECT quote,writer FROM quotes,writers,categories WHERE (quotes.writer_
↪id = writers.id) \
            AND (quotes.category_id = categories.id) AND \
            (categories.keyword NOT IN ('notfound')) ORDER BY random() LIMIT 1"
    curr.execute(SQL)
    quote = curr.fetchone()
    return quote
def get_quote_with_keyword(keyword):
    SQL = "SELECT EXISTS (SELECT 1 FROM categories,quotes \
        WHERE (quotes.category_id = categories.id) AND (keyword = '{}'))".format(
        keyword)
    curr.execute(SQL)
    category_exists = curr.fetchone()[0]
    if not category_exists:
        keyword = "notfound"
        session['404'] = True
    else:
        session['404'] = False
    SQL = "SELECT quotes.id,quote,writer FROM quotes,categories,writers \
        WHERE (quotes.category_id = categories.id) AND \
        (quotes.writer_id = writers.id) AND (keyword = '{}') ORDER BY random()␣
↪LIMIT 1".format(
        keyword)
    curr.execute(SQL)
    quote = curr.fetchone()
    return quote
```

The core of the applicaiton is served by these two function. These functions are called in `home` and `random` routes in order to generate quotes. `get_quote_random()` functions returns a randomly chosen quote from database. `get_quote_with_keyword(keyword)` function takes keyword as a parameter and afterwards it checks whether keyword exists in database or not. If exists, it creates a query with the keyword and returns data from database. If not than a key is held in the session in order to specify that quote with the keyword is not exists in database. Also, it makes a quote request with keyword `notfound`. This query returns one of the quotes which infroms users that given keyword is not exists.

```
@app.route('/', methods=['GET', 'POST'])
def keyword():
    keyword = ""
    if request.method == 'POST':
        keyword = request.form.get('keyword')
    elif request.method == 'GET':
        SQL = "SELECT keyword FROM categories WHERE \
        (keyword NOT IN ('notfound'))ORDER BY random() LIMIT 1"
        curr.execute(SQL)
        keyword = curr.fetchone()[0]
    data = get_quote_with_keyword(keyword)
    is_hidden = ''
    is_logged = ''
    if session.get('404'):
        if session['404']:
            is_hidden = 'hidden'
        else:
            is_hidden = ''
    if session.get('user_logged'):
        is_logged = ''
    else:
        is_logged = 'hidden'
```

```
    return render_template('home.html', writer=data[2], quote=data[1], keyword_
↪value=keyword,
                           quote_id=data[0], isHidden=is_hidden, islogged=is_logged)
```

This code block executed when user reach the home page of the website. If it is the first time that user enters the site, since it is a GET request, function will choose randomly keyword excluding `notfound` category. After that a quote with keyword will be generated. If user generating quotes with keyword using button on the home page, then it will be a POST request. Because of this, rather than generating new keyword, keyword in the form field will be used. Also `is_hidden` and `is_logged` session booleans are used for detecting whether user logged in or not. These way users will be prevented to send comments without logging in.

```
@app.route('/authentication', methods=['GET', 'POST'])
def auth_page():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        if request.form['btn'] == 'Login':
            # do login
            SQL = "SELECT * FROM users WHERE username='{}'".format(username)
            curr.execute(SQL)
            user = curr.fetchone()

            if username == "" or password == "":
                return render_template('auth.html', prompt="Form field should be␣
↪filled")
            elif user is None:
                return render_template('auth.html', prompt="Are you sure about␣
↪that username?")
            else:
                if password == user[2]:
                    session['api_key'] = user[3]
                    session['username'] = user[1]
                    session['user_logged'] = True
                    return redirect(url_for('generateKey'))
                else:
                    return render_template('auth.html', prompt="Password invalid.")

        elif request.form['btn'] == 'Create':
            # do create
            if username == "":
                return render_template('auth.html', prompt="An user without name,␣
↪are you robot?")
            elif password == "":
                return render_template('auth.html', prompt="Enter a pass, that␣
↪might be helpful.")
            else:
                apikey = binascii.hexlify(os.urandom(12)).decode('utf-8')
                SQL = "INSERT INTO users (username, password, api_key) " \
                      "SELECT '{}', '{}', '{}' WHERE NOT EXISTS \
                      (SELECT id FROM users WHERE username='{}') \
                      RETURNING id;".format(username, password, apikey, username)
                curr.execute(SQL)
                conn.commit()
                id = curr.fetchone()
                if id is not None:
                    session['api_key'] = apikey
                    session['user_logged'] = True
                    session['username'] = username
                    return redirect(url_for('generateKey'))
                else:
                    prompt = "Username '{}' exist, try another.".format(username)
```

```
                    return render_template('auth.html', prompt=prompt)

    elif request.method == 'GET':
        # if user logged in direct to generateKey
        if session.get('user_logged'):
            if session['user_logged']:
                return render_template('generateKey.html', apikey=session['api_key
↪'])
        else:
            return render_template('auth.html')
```

This code block provides function to create account or login. If user created an account it will redirect user to authentication page. When user try to create an account if the input fields are valid, function will make a request to database in order to insert username. If username exists than it will prompt some error. It will also create an unique api_key for that specific user.

After creating an account user can enter using same username and password value. If error exists, than it will prompt some warnings in order to inform user. Function will also keep some session variables in order to remember that user logged in.

```
@app.route('/giveRating', methods=['POST'])
def giveRating():
    star = request.form.get('rating')
    quote_id = request.form.get('quote_id')
    # keyword = request.form.get('keyword')
    SQL = "SELECT * FROM quotes WHERE id={}".format(quote_id)
    curr.execute(SQL)
    data = curr.fetchone()
    votes = data[2]
    rate = data[3]
    new_rate = 0
    if star == "star-5":
        new_rate = 5
    elif star == "star-4":
        new_rate = 4
    elif star == "star-3":
        new_rate = 3
    elif star == "star-2":
        new_rate = 2
    elif star == "star-1":
        new_rate = 1

    rate = float(rate * votes + new_rate) / float(votes + 1)
    votes = votes + 1

    SQL = "UPDATE quotes SET rate={}, votes={} WHERE id={}".format(rate, votes,␣
↪quote_id)
    curr.execute(SQL)
    conn.commit()
    return jsonify({
        'status': 'OK',
        'rating': star
    })
```

```
$(function () {
        $('.star').click(function (input) {
            if ($(this).is(':checked')) {
                var star = input.target.id;
                var quote_id = $('#quote-id').html();
                var keyword = '{{ keyword_value }}';
```

```
                $.post('/giveRating', {rating: star, quote_id: quote_id, keyword:␣
↪keyword},
                function (result) {
                    setTimeout(function () {
                        $('#thankYou').fadeIn(4000);
                    }, 1500);
                    $('#ratingFrom').fadeOut(1500);
                })
            }
        });
    });
```

In this code blocks giving rating for quotes handled. With jquery post request the rating clicked by user will sent to `giveRating()` function. This function will use this information and update the votes and rating in the database for that specific quteo.

```
@app.route('/addNew', methods=['GET', 'POST'])
def addNew():
    SQL = "SELECT keyword FROM categories"
    curr.execute(SQL)
    categories = ''

    if session.get('user_logged'):
        prompt_hidden = "hidden"
        btn_hidden = ""
    else:
        prompt_hidden = ""
        btn_hidden = "hidden"
    for x in curr.fetchall():
        if x[0] != "notfound":
            categories += "<option value=" + "{}>".format(x[0]) + x[0] + "</option>
↪"

    if request.method == "POST":
        quote = request.form.get('quote')
        writer = request.form.get('writer')
        keyword = request.form.get('sel1')
        if session.get('username'):
            username = session['username']
            # if user is admin he can directly add to the main quotes table
            if username == "admin":
                SQL = "SELECT id FROM writers WHERE writer = '{}'".format(writer)
                curr.execute(SQL)
                writer_id = curr.fetchone()

                SQL = "SELECT id from categories WHERE keyword = '{}'".
↪format(keyword)
                curr.execute(SQL)
                category_id = curr.fetchone()[0]

                if writer_id:
                    # writer is in database
                    writer_id = writer_id[0]
                else:
                    # writer is not in database
                    # insert writer than return id
                    SQL = "INSERT INTO writers(writer) VALUES ('{}') RETURNING id".
↪format(writer)
                    curr.execute(SQL)
                    conn.commit()
                    writer_id = curr.fetchone()[0]
```

```
                SQL = "INSERT INTO quotes(quote, category_id, writer_id) \
                VALUES ('{}', {}, {})".format(quote, category_id, writer_id)

            else:
                SQL = "INSERT INTO user_quotes(user_id, quote, writer, category_
↪id) VALUES  \
                  ((SELECT id FROM users WHERE username = '{}'),'{}','{}',\
                  (SELECT id FROM categories WHERE keyword = '{}'))".format(
                    username, quote, writer, keyword)

            curr.execute(SQL)
            conn.commit()

    return render_template("addNew.html", categoryList=categories,␣
↪promptHidden=prompt_hidden,
                           btnHidden=btn_hidden, prompt='')
```

This code block handles adding new quotes to database using web interface. It will give different behaviours whether the user `admin` or not. If it is admin then the quotes will be directly added to quotes database. Otherwise they will be added to `user_quotes` table.

2.2.2 API Service

API service require authentication and also gives some errors if some credential is absent. Errors and necessity of authentication provided using these functions

```
@app.route('/quote/api/v1.0/random', methods=['GET'])
@auth.login_required
def get_random():
    quote = get_quote_random()
    if 'ApiKey' not in request.headers:
        return make_response(jsonify({'error': 'Add ApiKey to header'}), 400)
    api_key = request.headers['ApiKey']
    api_key_exist_in_db = False
    SQL = "SELECT id from users WHERE api_key='{}'".format(api_key)

    if api_key != "":
        curr.execute(SQL)
        if curr.fetchone() is not None: api_key_exist_in_db = True

    if api_key == "" or not api_key_exist_in_db:
        return make_response(jsonify({'error': 'ApiKey Missing or Wrong'}), 400)
    else:
        return jsonify({
            'quote': quote[1],
            'writer': quote[2]
        })


@app.route('/quote/api/v1.0/<string:keyword>', methods=['GET'])
@auth.login_required
def get_with_keyword(keyword):
    quote = get_quote_with_keyword(keyword=keyword)

    if 'ApiKey' not in request.headers:
        return make_response(jsonify({'error': 'Add ApiKey to header'}), 400)
    api_key = request.headers['ApiKey']
    api_key_exist_in_db = False
    SQL = "SELECT id from users WHERE api_key='{}'".format(api_key)
```

```
    if api_key != "":
        curr.execute(SQL)
        if curr.fetchone() is not None: api_key_exist_in_db = True

    if api_key == "" or not api_key_exist_in_db:
        return make_response(jsonify({'error': 'ApiKey Missing or Wrong'}), 400)
    else:
        if len(quote) == 0:
            abort(404)
        return jsonify({
            'quote': quote[1],
            'writer': quote[2]
        })
```

If user enters credential properly, these functions check for `api_key` and return quotes either with keyword or randomly choosen as before. The `get_quote_random()` and `get_quote_with_keyword()` functions works as described in the web client part.