

Manarat International University

Department of Computer Science & Engineering

Software Development (CSE 312)

Team Information

- Team Name- 1453
- Rahat Hasan Akash
- 1846CSE00694

Project Name: 1453_Blog (Web Based)

GitHub repository: https://github.com/Rahathasan1453/softDev_1453

Project Video: <https://drive.google.com/file/d/1whKiO6PgaNkm4Ng4-as9Kdq6Idek5snj/view?usp=sharing>

Objective

Our objective was to build up a personal Blogging Site. Where An admin can read, edit, update and add blog & a user can read the blogs.

Project Introduction

A personal blog is that sort of blog that has grown gradually over the span of years. A blog simply is a sort of website that mainly focuses on written content.

Personal blogs have gained a lot of popularity with time and have become a great source of sharing experiences, feelings, thoughts or even knowledge about anything and everything.

A blog is an online journal or informational website displaying information in reverse chronological order, with the latest posts appearing first, at the top. It is a platform where a writer or a group of writers share their views on an individual subject.

Feature Description

This is a personal blogging site. Here Admin can Add blog, Read Blog, Edit and Update the Blog. For login into Admin panel, admin have to log in into admin page with Email and Password.

A general viewer just can read the blog which are provided by the admin. But if he want he can sing in into the system and can add blog, he can write comment on specific blog, can share it on social media (future plan).

Language – Python3, HTML (Markup language)

Framework – Flask

Platform – Visual Studio Code

Libraries – Flask Libraries

Database – MySql Database

Problem Faced

We have started this project from the very beginning lavel knowledge of Flask framework . So we have faced many problem such as making environment, installing modules, add database with the main file etc. We got some help from The Internet & our friend Ahmed Zobayer helped a lot in this work. Hopefully those problem will help us to enrich our field of knowledge.

Future Plan

- Add the comment & Share section.
- Make an Android App of this Blog site.

Source Code

```
File Edit Selection View Go Terminal Help • venv.py - srcDir: 1453 - Visual Studio Code
Welcome | login.html | jgug.html | footer.html | _ht_py | venv.py • | resources.html | blog_admin.html | BlogEdit | ...
app > venv.py
1 from app import app
2 from flask import render_template, request, redirect, session, jsonify, make_response, send_from_directory
3 from flask_wtf import FlaskForm, form
4 from wtforms import StringField, StringField, StringField, StringField, StringField, StringField
5 from flask_mysqldb import MySQL
6 from flask import request
7 from flask import request
8 import os
9
10 app.config['SECRET_KEY'] = 'redishketch'
11 mySQL = MySQL(app)
12 bcrypt = bcrypt(app)
13
14 @app.route('/')
15 def index():
16     cur = mySQL.connection.cursor()
17     cur.execute("SELECT * FROM articles")
18     articles = cur.fetchall()
19     return render_template("index.html", articles = articles)
20
21 @app.route('/article')
22 def article():
23     cur = mySQL.connection.cursor()
24     cur.execute("SELECT * FROM articles")
25     articles = cur.fetchall()
26     return render_template("/Pages/Blog.html", articles = articles)
27
28 @app.route('/article/add')
29 def article_add():
30     return render_template("/Pages/Blog_add.html")
```

1

```

1  # -*- coding: utf-8 -*-
2
3  import os
4  import sys
5  import random
6  import string
7  import time
8  import datetime
9  import hashlib
10 import json
11
12 from flask import Flask, request, session, g, redirect, url_for, \
13     abort, render_template, current_app
14 from flask_sqlalchemy import SQLAlchemy
15 from flask_migrate import Migrate
16 from flask_login import LoginManager, login_required, \
17     login_user, logout_user
18
19 from werkzeug.security import generate_password_hash, \
20     check_password_hash
21
22 from itsdangerous import URLSafeTimedSerializer as Serializer
23
24 from extensions import db, migrate, login_manager
25
26 app = Flask(__name__)
27 app.config['SECRET_KEY'] = 'secretkey'
28 app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:123456@localhost:3306/test'
29 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
30 db.init_app(app)
31 migrate.init_app(app, db)
32 login_manager.init_app(app)
33
34 # 用户登录
35 @app.route('/login', methods=['POST'])
36 def login():
37     input = request.form
38     v = Validator()
39     schema = {
40         'email': {'type': 'string', 'required': True, 'empty': False},
41         'password': {'type': 'string', 'required': True, 'empty': False},
42     }
43     if v.validate(input, schema) == False:
44         return {'status': 'error', 'msg': 'validation error', 'errors': v.errors}
45
46     try:
47         cur = mysql.connection.cursor()
48         sql = 'select * from users where email = %s'
49         values = (input['email'],)
50         cur.execute(sql, values)
51         result = cur.fetchall()
52         if len(result) == 1:
53             user = result[0]
54             password = user['password']
55             if check_password_hash(password, input['password']):
56                 login_user(user)
57                 return redirect(url_for('index'))
58             else:
59                 return {'status': 'error', 'msg': 'password error'}
60         else:
61             return {'status': 'error', 'msg': 'user not exist'}
62     except:
63         return {'status': 'error', 'msg': 'database error'}
64
65 # 用户注册
66 @app.route('/register', methods=['POST'])
67 def register():
68     input = request.form
69     v = Validator()
70     schema = {
71         'email': {'type': 'string', 'required': True, 'empty': False},
72         'password': {'type': 'string', 'required': True, 'empty': False},
73     }
74     if v.validate(input, schema) == False:
75         return {'status': 'error', 'msg': 'validation error', 'errors': v.errors}
76
77     try:
78         cur = mysql.connection.cursor()
79         sql = 'select * from users where email = %s'
80         values = (input['email'],)
81         cur.execute(sql, values)
82         result = cur.fetchall()
83         if len(result) == 1:
84             return {'status': 'error', 'msg': 'user already exist'}
85         else:
86             password = generate_password_hash(input['password'])
87             sql = 'insert into users(email, password) values(%s, %s)'
88             values = (input['email'], password)
89             cur.execute(sql, values)
90             db.session.commit()
91             return {'status': 'success', 'msg': 'register success'}
92     except:
93         return {'status': 'error', 'msg': 'database error'}
94
95 # 文章编辑
96 @app.route('/article/edit/', methods=['POST'])
97 def article_edit(id):
98     cur = mysql.connection.cursor()
99     sql = 'select * from articles where id = %s'
100    values = (id,)
101    cur.execute(sql, values)
102    result = cur.fetchall()
103    if len(result) == 1:
104        article = result[0]
105        return render_template('/Pages/blog_edit.html', article=article)
106    else:
107        return {'status': 'error', 'msg': 'article not exist'}
108
109 # 文章发布
110 @app.route('/article/vin/', methods=['POST'])
111 def article_vin(id):
112     cur = mysql.connection.cursor()
113     sql = 'select * from articles where id = %s'
114     values = (id,)
115     cur.execute(sql, values)
116     result = cur.fetchall()
117     if len(result) == 1:
118         article = result[0]
119         return render_template('/Pages/blog_vin.html', article=article)
120     else:
121         return {'status': 'error', 'msg': 'article not exist'}
122
123 # 博客编辑
124 @app.route('/blog/edit', methods=['POST'])
125 def blog_edit():
126     cur = mysql.connection.cursor()
127     sql = 'select * from blogs where id = %s'
128     values = (request.form['id'],)
129     cur.execute(sql, values)
130     result = cur.fetchall()
131     if len(result) == 1:
132         blog = result[0]
133         return render_template('/Pages/blog_edit.html', blog=blog)
134     else:
135         return {'status': 'error', 'msg': 'blog not exist'}
136
137 # 博客发布
138 @app.route('/blog/vin', methods=['POST'])
139 def blog_vin():
140     cur = mysql.connection.cursor()
141     sql = 'select * from blogs where id = %s'
142     values = (request.form['id'],)
143     cur.execute(sql, values)
144     result = cur.fetchall()
145     if len(result) == 1:
146         blog = result[0]
147         return render_template('/Pages/blog_vin.html', blog=blog)
148     else:
149         return {'status': 'error', 'msg': 'blog not exist'}
150
151 # 首页
152 @app.route('/', methods=['GET'])
153 def index():
154     cur = mysql.connection.cursor()
155     sql = 'select * from articles'
156     cur.execute(sql)
157     result = cur.fetchall()
158     return render_template('index.html', articles=result)
159
160 # 关于
161 @app.route('/about', methods=['GET'])
162 def about():
163     return render_template('about.html')
164
165 # 联系我们
166 @app.route('/contact', methods=['GET'])
167 def contact():
168     return render_template('contact.html')
169
170 # 404
171 @app.route('/404', methods=['GET'])
172 def 404():
173     return render_template('404.html')
174
175 # 500
176 @app.route('/500', methods=['GET'])
177 def 500():
178     return render_template('500.html')
179
180 # 测试
181 @app.route('/test', methods=['GET'])
182 def test():
183     return 'test'
184
185 if __name__ == '__main__':
186     app.run(debug=True)

```

2

```

1  # Django REST Framework view
2  from django.contrib.auth import authenticate, login
3  from django.contrib.auth.models import User
4  from django.http import JsonResponse
5  from django.shortcuts import get_object_or_404
6  from django.urls import reverse
7  from django.utils.decorators import method_decorator
8  from django.views.decorators.csrf import csrf_exempt
9  from django.views.generic import View
10 from rest_framework import status
11 from rest_framework.decorators import api_view
12 from rest_framework.response import Response
13 from rest_framework.views import APIView
14
15 # Create a new user
16 def register_user(request):
17     """
18     Register a new user
19     """
20     if request.method == 'POST':
21         username = request.data.get('username')
22         password = request.data.get('password')
23         email = request.data.get('email')
24
25         # Check if user already exists
26         if User.objects.filter(username=username).exists():
27             return JsonResponse({
28                 'status': 400,
29                 'msg': 'User already exists'
30             }, status=status.HTTP_400_BAD_REQUEST)
31
32         # Create new user
33         user = User.objects.create_user(
34             username=username,
35             password=password,
36             email=email
37         )
38         user.set_password(password)
39         user.save()
40
41         # Login the user
42         login(request, user)
43
44         return JsonResponse({
45             'status': 200,
46             'msg': 'User registered successfully'
47         }, status=status.HTTP_200_OK)
48     return JsonResponse({
49         'status': 400,
50         'msg': 'Invalid request'
51     }, status=status.HTTP_400_BAD_REQUEST)
52
53 # Login user
54 def login_user(request):
55     """
56     Login a user
57     """
58     if request.method == 'POST':
59         email = request.data.get('email')
60         password = request.data.get('password')
61
62         # Authenticate user
63         user = authenticate(email=email, password=password)
64
65         if user is None:
66             return JsonResponse({
67                 'status': 400,
68                 'msg': 'Invalid credentials'
69             }, status=status.HTTP_400_BAD_REQUEST)
70
71         # Login user
72         login(request, user)
73
74         # Create session
75         session = request.session
76         session['user_id'] = user.id
77         session.save()
78
79         return JsonResponse({
80             'status': 200,
81             'msg': 'User logged in successfully'
82         }, status=status.HTTP_200_OK)
83     return JsonResponse({
84         'status': 400,
85         'msg': 'Invalid request'
86     }, status=status.HTTP_400_BAD_REQUEST)
87
88 # Logout user
89 def logout_user(request):
90     """
91     Logout a user
92     """
93     if request.method == 'POST':
94         # Destroy session
95         request.session.flush()
96
97         return JsonResponse({
98             'status': 200,
99             'msg': 'User logged out successfully'
100        }, status=status.HTTP_200_OK)
101    return JsonResponse({
102        'status': 400,
103        'msg': 'Invalid request'
104    }, status=status.HTTP_400_BAD_REQUEST)
105
106 # Register schema
107 class RegisterSchema(APIView):
108     """
109     Register schema
110     """
111     def post(self, request):
112         """
113         Register a new user
114         """
115         register_user(request)
116
117     def get(self, request):
118         """
119         Get all users
120         """
121         users = User.objects.all()
122         return JsonResponse({
123             'status': 200,
124             'users': users
125         }, status=status.HTTP_200_OK)
126
127 # Login schema
128 class LoginSchema(APIView):
129     """
130     Login schema
131     """
132     def post(self, request):
133         """
134         Login a user
135         """
136         login_user(request)
137
138     def get(self, request):
139         """
140         Get all users
141         """
142         users = User.objects.all()
143         return JsonResponse({
144             'status': 200,
145             'users': users
146         }, status=status.HTTP_200_OK)
147
148 # Logout schema
149 class LogoutSchema(APIView):
150     """
151     Logout schema
152     """
153     def post(self, request):
154         """
155         Logout a user
156         """
157         logout_user(request)
158
159     def get(self, request):
160         """
161         Get all users
162         """
163         users = User.objects.all()
164         return JsonResponse({
165             'status': 200,
166             'users': users
167         }, status=status.HTTP_200_OK)
168
169 # Register view
170 class RegisterView(APIView):
171     """
172     Register view
173     """
174     def post(self, request):
175         """
176         Register a new user
177         """
178         register_user(request)
179
180     def get(self, request):
181         """
182         Get all users
183         """
184         users = User.objects.all()
185         return JsonResponse({
186             'status': 200,
187             'users': users
188         }, status=status.HTTP_200_OK)
189
190 # Login view
191 class LoginView(APIView):
192     """
193     Login view
194     """
195     def post(self, request):
196         """
197         Login a user
198         """
199         login_user(request)
200
201     def get(self, request):
202         """
203         Get all users
204         """
205         users = User.objects.all()
206         return JsonResponse({
207             'status': 200,
208             'users': users
209         }, status=status.HTTP_200_OK)
210
211 # Logout view
212 class LogoutView(APIView):
213     """
214     Logout view
215     """
216     def post(self, request):
217         """
218         Logout a user
219         """
220         logout_user(request)
221
222     def get(self, request):
223         """
224         Get all users
225         """
226         users = User.objects.all()
227         return JsonResponse({
228             'status': 200,
229             'users': users
230         }, status=status.HTTP_200_OK)
231
232 # Register schema
233 class RegisterSchema(APIView):
234     """
235     Register schema
236     """
237     def post(self, request):
238         """
239         Register a new user
240         """
241         register_user(request)
242
243     def get(self, request):
244         """
245         Get all users
246         """
247         users = User.objects.all()
248         return JsonResponse({
249             'status': 200,
250             'users': users
251         }, status=status.HTTP_200_OK)
252
253 # Login schema
254 class LoginSchema(APIView):
255     """
256     Login schema
257     """
258     def post(self, request):
259         """
260         Login a user
261         """
262         login_user(request)
263
264     def get(self, request):
265         """
266         Get all users
267         """
268         users = User.objects.all()
269         return JsonResponse({
270             'status': 200,
271             'users': users
272         }, status=status.HTTP_200_OK)
273
274 # Logout schema
275 class LogoutSchema(APIView):
276     """
277     Logout schema
278     """
279     def post(self, request):
280         """
281         Logout a user
282         """
283         logout_user(request)
284
285     def get(self, request):
286         """
287         Get all users
288         """
289         users = User.objects.all()
290         return JsonResponse({
291             'status': 200,
292             'users': users
293         }, status=status.HTTP_200_OK)
294
295 # Register view
296 class RegisterView(APIView):
297     """
298     Register view
299     """
300     def post(self, request):
301         """
302         Register a new user
303         """
304         register_user(request)
305
306     def get(self, request):
307         """
308         Get all users
309         """
310         users = User.objects.all()
311         return JsonResponse({
312             'status': 200,
313             'users': users
314         }, status=status.HTTP_200_OK)
315
316 # Login view
317 class LoginView(APIView):
318     """
319     Login view
320     """
321     def post(self, request):
322         """
323         Login a user
324         """
325         login_user(request)
326
327     def get(self, request):
328         """
329         Get all users
330         """
331         users = User.objects.all()
332         return JsonResponse({
333             'status': 200,
334             'users': users
335         }, status=status.HTTP_200_OK)
336
337 # Logout view
338 class LogoutView(APIView):
339     """
340     Logout view
341     """
342     def post(self, request):
343         """
344         Logout a user
345         """
346         logout_user(request)
347
348     def get(self, request):
349         """
350         Get all users
351         """
352         users = User.objects.all()
353         return JsonResponse({
354             'status': 200,
355             'users': users
356         }, status=status.HTTP_200_OK)
357
358 # Register schema
359 class RegisterSchema(APIView):
360     """
361     Register schema
362     """
363     def post(self, request):
364         """
365         Register a new user
366         """
367         register_user(request)
368
369     def get(self, request):
370         """
371         Get all users
372         """
373         users = User.objects.all()
374         return JsonResponse({
375             'status': 200,
376             'users': users
377         }, status=status.HTTP_200_OK)
378
379 # Login schema
380 class LoginSchema(APIView):
381     """
382     Login schema
383     """
384     def post(self, request):
385         """
386         Login a user
387         """
388         login_user(request)
389
390     def get(self, request):
391         """
392         Get all users
393         """
394         users = User.objects.all()
395         return JsonResponse({
396             'status': 200,
397             'users': users
398         }, status=status.HTTP_200_OK)
399
400 # Logout schema
401 class LogoutSchema(APIView):
402     """
403     Logout schema
404     """
405     def post(self, request):
406         """
407         Logout a user
408         """
409         logout_user(request)
410
411     def get(self, request):
412         """
413         Get all users
414         """
415         users = User.objects.all()
416         return JsonResponse({
417             'status': 200,
418             'users': users
419         }, status=status.HTTP_200_OK)
420
421 # Register view
422 class RegisterView(APIView):
423     """
424     Register view
425     """
426     def post(self, request):
427         """
428         Register a new user
429         """
430         register_user(request)
431
432     def get(self, request):
433         """
434         Get all users
435         """
436         users = User.objects.all()
437         return JsonResponse({
438             'status': 200,
439             'users': users
440         }, status=status.HTTP_200_OK)
441
442 # Login view
443 class LoginView(APIView):
444     """
445     Login view
446     """
447     def post(self, request):
448         """
449         Login a user
450         """
451         login_user(request)
452
453     def get(self, request):
454         """
455         Get all users
456         """
457         users = User.objects.all()
458         return JsonResponse({
459             'status': 200,
460             'users': users
461         }, status=status.HTTP_200_OK)
462
463 # Logout view
464 class LogoutView(APIView):
465     """
466     Logout view
467     """
468     def post(self, request):
469         """
470         Logout a user
471         """
472         logout_user(request)
473
474     def get(self, request):
475         """
476         Get all users
477         """
478         users = User.objects.all()
479         return JsonResponse({
480             'status': 200,
481             'users': users
482         },
```

3

```

104 if INPUT['password'] != INPUT['confirm_password']:
105     return ("status": 5000, "msg": "validation error", "errors": {"password": ["confirm password did not match"]})
106
107 cur = mysql.connection.cursor()
108 cur.execute("select * FROM users where email = %s", [INPUT['email']])
109 user = cur.fetchone()
110
111 if user is not None:
112     return ("status": 5000, "msg": "validation error", "errors": {"email": ["Email address already exist"]})
113
114 password_hashed = bcrypt.generate_password_hash(INPUT['password'])
115 try:
116     cur.execute("insert INTO users (name, username, institute, phone, email, password) VALUES (%s,%s,%s,%s,%s,%s)", [INPUT['name'],
117     mysql.connection.commit()
118     return ("status": 2000, "msg": "successfully registered")
119 except Exception as e:
120     return ("status": 5000, "msg": "something went wrong", "error": str(e))
121
122 @app.route('/api/article/add', methods=['POST'])
123 def article_add_api():
124     INPUT = request.form
125     v = validator
126     schema = {
127         'title': {'type': 'string', 'required': True, 'empty': False},
128         'content': {'type': 'string', 'required': True, 'empty': False},
129     }
130     v.allow_unknown = True
131     if v.validate(INPUT, schema) == False:
132         return ("status": 5000, "msg": "validation error", "errors": v.errors)
133
134     if request.files.get('cover') == None:
135         return ("status": 5000, "msg": "validation error", "errors": {"cover": ["cover is required"]})
136
137     try:

```

4

[illegible]

5

```

173 new_cover_name = None
174 if request.files.get('cover') != None:
175     cover = request.files['cover']
176     cover_file_ext = cover.filename.split('.')[-1]
177     if cover_file_ext in ALLOWED_EXTENSIONS_COVER:
178         cover_unique_filename = str(uuid.uuid4())
179         new_cover_name = cover.filename.replace('.' + cover_file_ext
180         cover.save(os.path.join app.config['UPLOAD_FOLDER'], new_cover_name))
181     else:
182         return jsonify({'status': '500', 'msg': 'validation error', 'errors': ['cover', '[cover file format is invalid]']})
183
184
185
186 if new_cover_name == None:
187     cur = mysql.connection.cursor()
188     cur.execute("INSERT INTO articles SET title = %s, contents = %s, where_id = %s", [INPUT['title'], INPUT['contents'], INPUT['id']])
189     mysql.connection.commit()
190 else:
191     cur = mysql.connection.cursor()
192     cur.execute("UPDATE articles SET title = %s, contents = %s, cover = %s WHERE id = %s", [INPUT['title'], INPUT['contents'],
193     mysql.connection.commit()
194     return jsonify({'status': '2000', 'msg': 'successfully updated'})
195 except Exception as e:
196     return jsonify({'status': '5000', 'msg': 'something went wrong', 'error': str(e)})
197
198
199 @app.route('/top/article/get/single', methods=['POST'])
200 def article_get_single_sql():
201     INPUT = request.form
202     v = validator()
203     schema = {
204         'id': (Type: 'string', required: True, empty: false),
205     }
206     if v.validate(INPUT, schema) == False:
207         return jsonify({'status': '5000', 'msg': 'validation error', 'errors': v.errors})

```

f

```

201 cur = mysql.cursor(cursor)
202 cur.execute("SELECT * FROM library WHERE id = %s", (IDOUT[14]))
203 row = cur.fetchall()
204 return ("Status": 2000, "data": row)
205
206 @app.route("/api/articles/get/all", methods=["POST"])
207 def article_get_all(api):
208     cur = mysql.cursor(cursor)
209     cur.execute("SELECT * FROM library")
210     rows = cur.fetchall()
211     return ("Status": 2000, "data": rows)
212
213

```

7

Conclusion

A personal Blog is one of the platforms where a person can share their views and knowledge and opinion. It can also help him to gather his knowledge, thoughts.

To build up this site, we have used Python language for the operations and functions, Flask framework and MySql Database. To develop this site, we have faced many difficulties which 'll help us in near future.

