

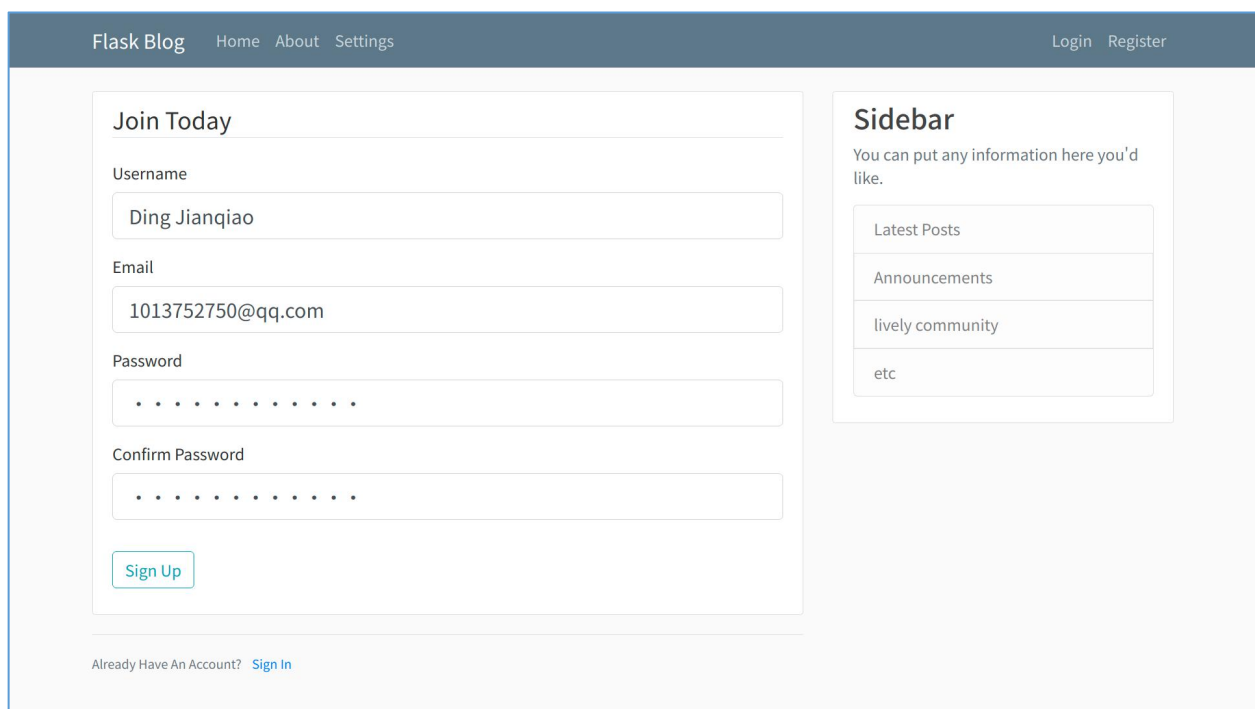
# Project Brief Description:

In this coursework, one blog application called “Flask Blog” is developed to let users create posts and make comments or give likes to posts.

## ● Features:

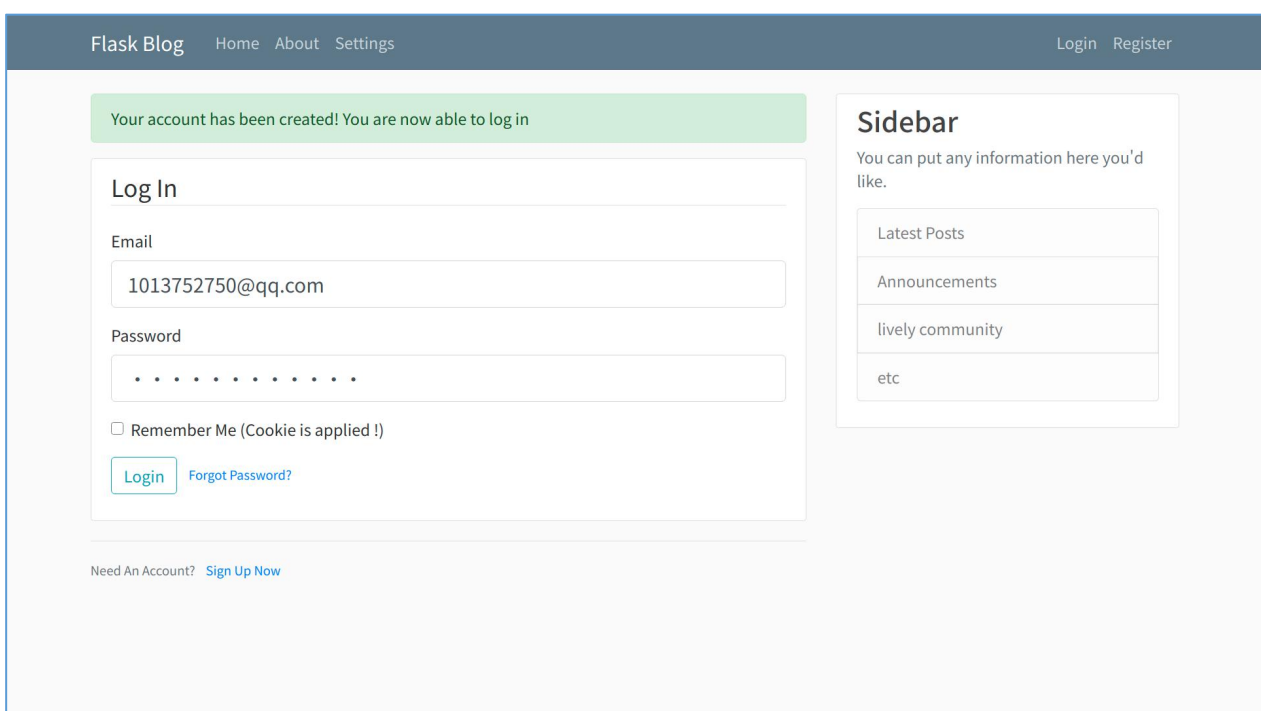
### 1. Basic User Operations:

Users can easily register an account with appropriate email address, username and password. When the registration is finished, users will be able to login the blog application with registered account. After logged in, users can also click the “Logout” button to logout the system.



The screenshot shows the registration page of the Flask Blog application. The header includes the site name "Flask Blog" and navigation links "Home", "About", and "Settings". On the right, there are links for "Login" and "Register". The main content area is titled "Join Today" and contains a registration form with the following fields: "Username" (with the value "Ding Jianqiao"), "Email" (with the value "1013752750@qq.com"), "Password", and "Confirm Password". Each password field is masked with dots. A "Sign Up" button is located at the bottom of the form. Below the form, there is a link "Already Have An Account? Sign In". To the right of the form is a "Sidebar" section with the text "You can put any information here you'd like." and a list of items: "Latest Posts", "Announcements", "lively community", and "etc".

( register one account )



The screenshot shows the login page of the Flask Blog application after a successful registration. The header is identical to the previous page. A green success message banner at the top reads "Your account has been created! You are now able to log in". The main content area is titled "Log In" and contains a login form with the following fields: "Email" (with the value "1013752750@qq.com") and "Password" (masked with dots). Below the password field is a checkbox labeled "Remember Me (Cookie is applied !)". A "Login" button and a "Forgot Password?" link are at the bottom of the form. Below the form, there is a link "Need An Account? Sign Up Now". To the right of the form is a "Sidebar" section, identical to the one in the previous screenshot.

( register finished, use the account to login )

Flask Blog

HomeAboutSettings

LoginRegister

Your account has been created! You are now able to log in

Log In

Email

1013752750@qq.com

Password

.....

☐ Remember Me (Cookie is applied !)

LoginForgot Password?

Need An Account? [Sign Up Now](#)

Sidebar

You can put any information here you'd like.

Latest Posts

Announcements

lively community

etc

( log in successfully )

Flask Blog

HomeAboutSettings

LoginRegister

Your account has been created! You are now able to log in

Log In

Email

1013752750@qq.com

Password

.....

☐ Remember Me (Cookie is applied !)

LoginForgot Password?

Need An Account? [Sign Up Now](#)

Sidebar

You can put any information here you'd like.

Latest Posts

Announcements

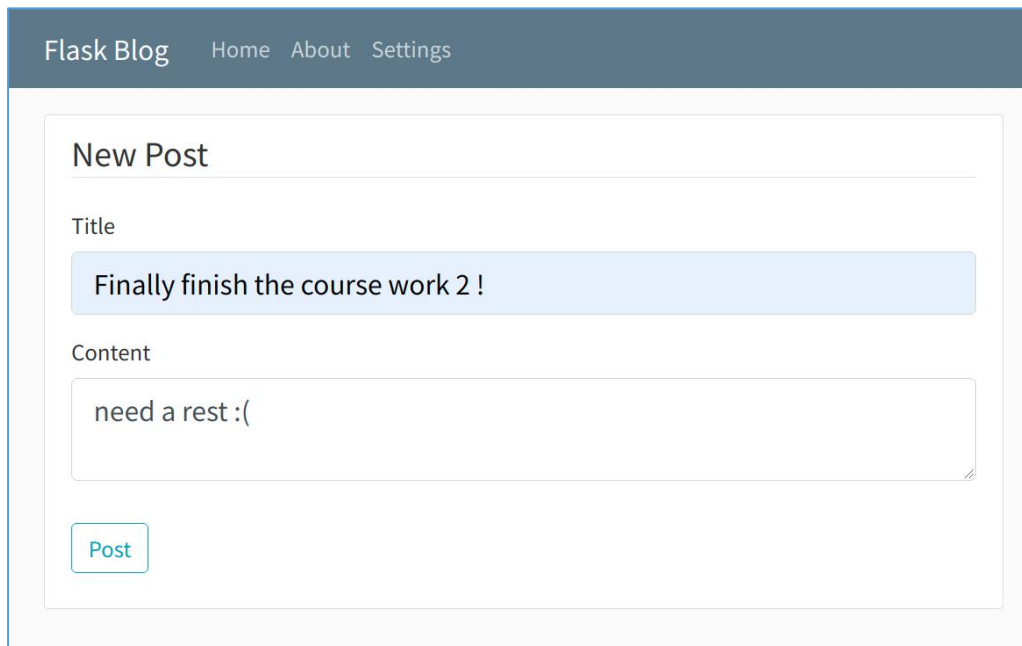
lively community

etc

( log out )

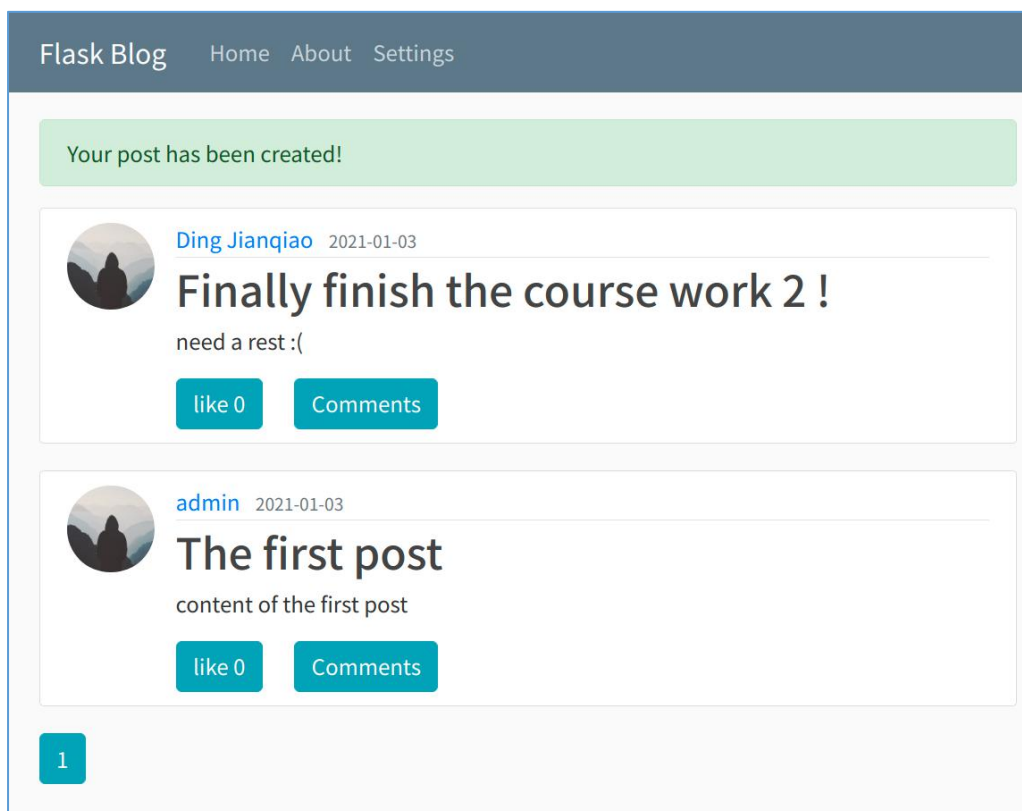
## 2. Post Operations:

In this application, posts from all users are displayed in home page. To create a post, users can click the “New Post” button and start up a form for a new post.



The screenshot shows the 'New Post' form in the Flask Blog application. The form is titled 'New Post' and contains two input fields: 'Title' and 'Content'. The 'Title' field contains the text 'Finally finish the course work 2 !' and the 'Content' field contains the text 'need a rest :('. Below the input fields is a 'Post' button.

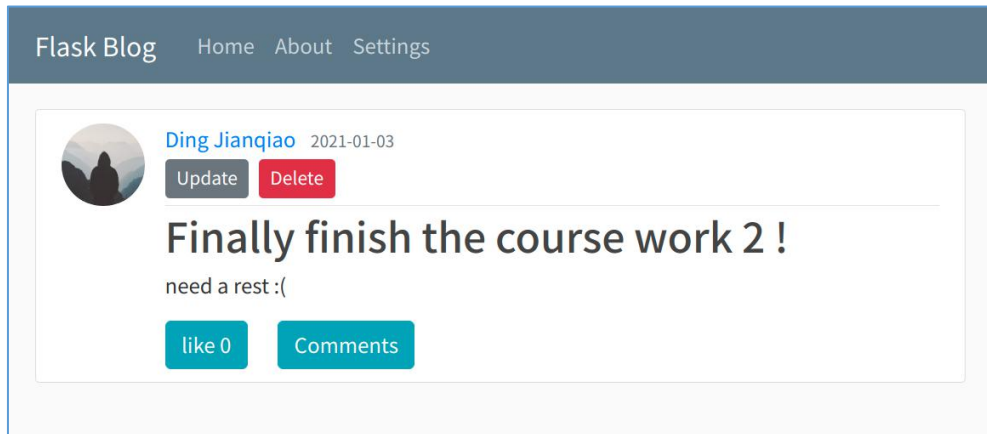
( create a new post )



The screenshot shows the Flask Blog home page. At the top, there is a navigation bar with the text 'Flask Blog' and links for 'Home', 'About', and 'Settings'. Below the navigation bar, there is a green banner that says 'Your post has been created!'. The main content area displays two posts. The first post is by 'Ding Jianqiao' and is dated '2021-01-03'. It has a title 'Finally finish the course work 2 !' and a content 'need a rest :('. Below the title and content, there are two buttons: 'like 0' and 'Comments'. The second post is by 'admin' and is dated '2021-01-03'. It has a title 'The first post' and a content 'content of the first post'. Below the title and content, there are two buttons: 'like 0' and 'Comments'. At the bottom of the page, there is a pagination button labeled '1'.

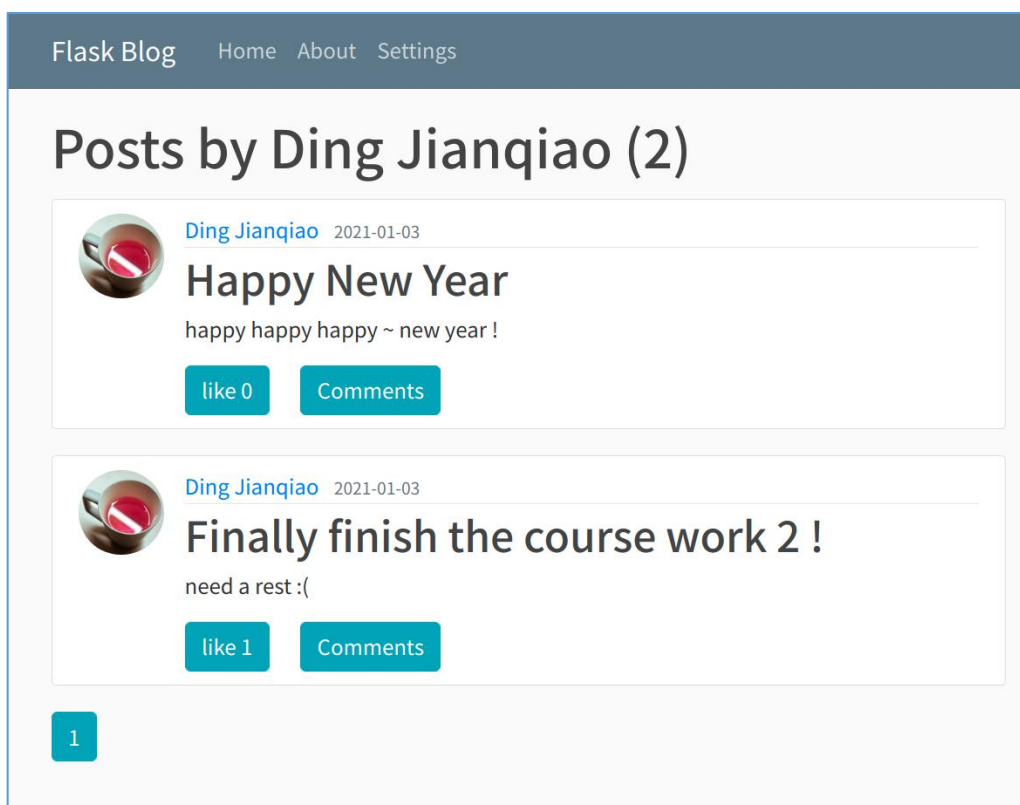
( new post created )

Enter the page for specific post, user can manipulate the post if he/she is the author of the post, such as deleting or updating.



( manipulate a post )

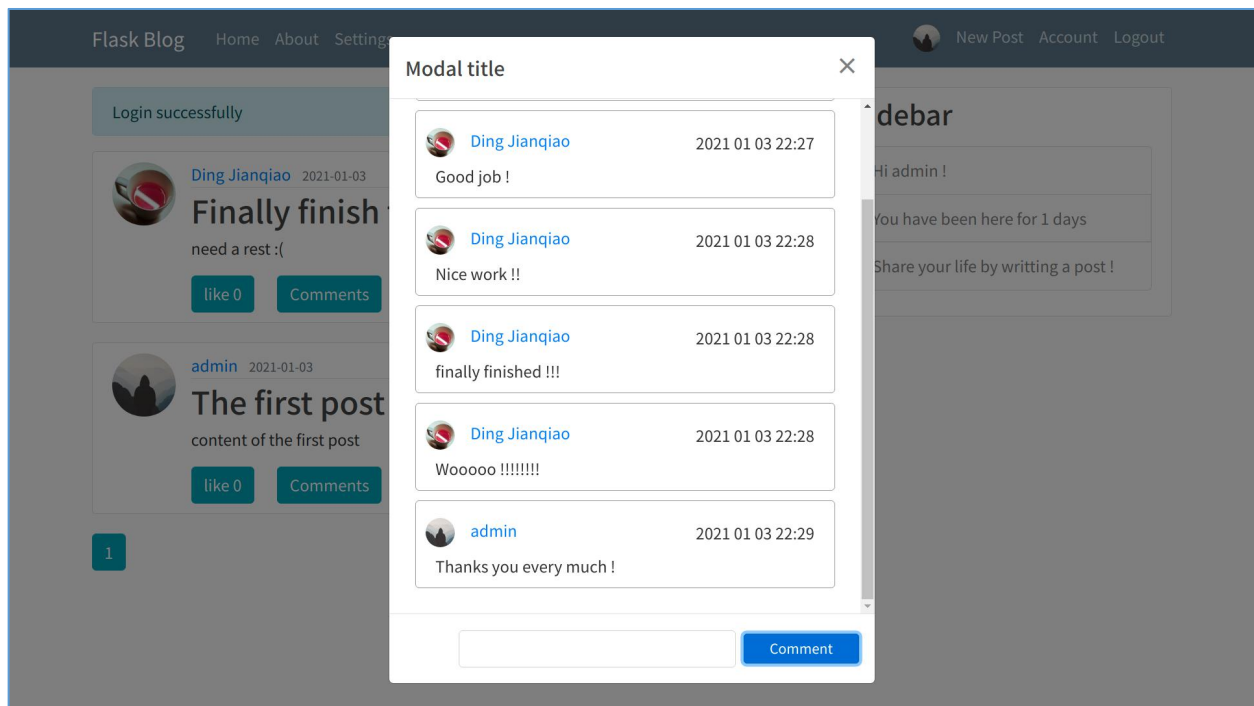
Each user has its own page displaying all his/her posts.



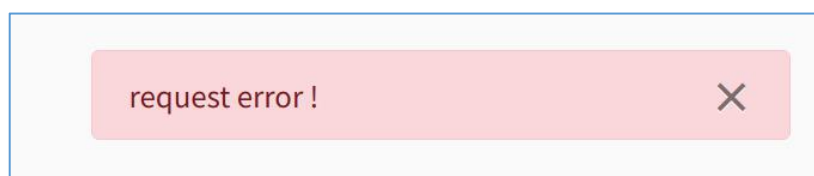
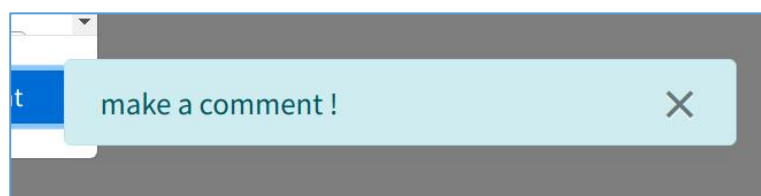
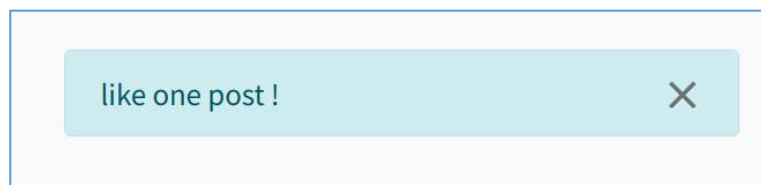
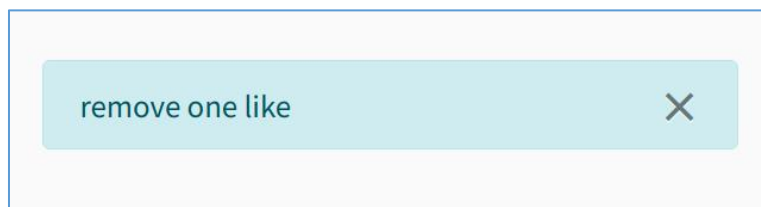
( personal posts )

### 3. Social Media:

In this blog application, one user can make comments under other users' posts. The basic information of a comment, such as its content, author and time are displayed in good styling. Moreover, to observe the popularity of one post, "like" is appended to show how many users like it. All of these operations have a clear notification feedback at right bottom corner.



( comments )




( various notification )

#### 4. Account Information Modification:

In “Flask Blog”, it is convenient for users to view or update their basic account information by clicking “Account” button. In “Account” page, basic information are displayed as the placeholders of input forms. By changing them and click “submit” button, new information will be updated immediately.

Flask Blog Home About Settings

 **Ding Jianqiao**  
1013752750@qq.com

Account Info

Username  
Ding Jianqiao

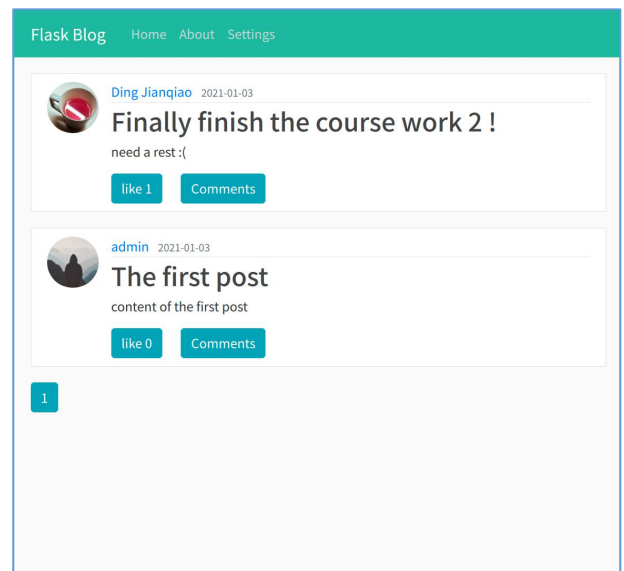
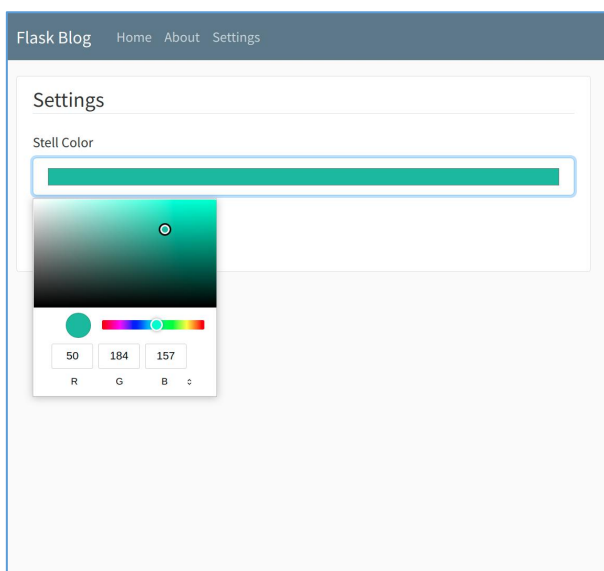
Email  
1013752750@qq.com

Update Profile Picture  
选择文件 未选择文件

Update

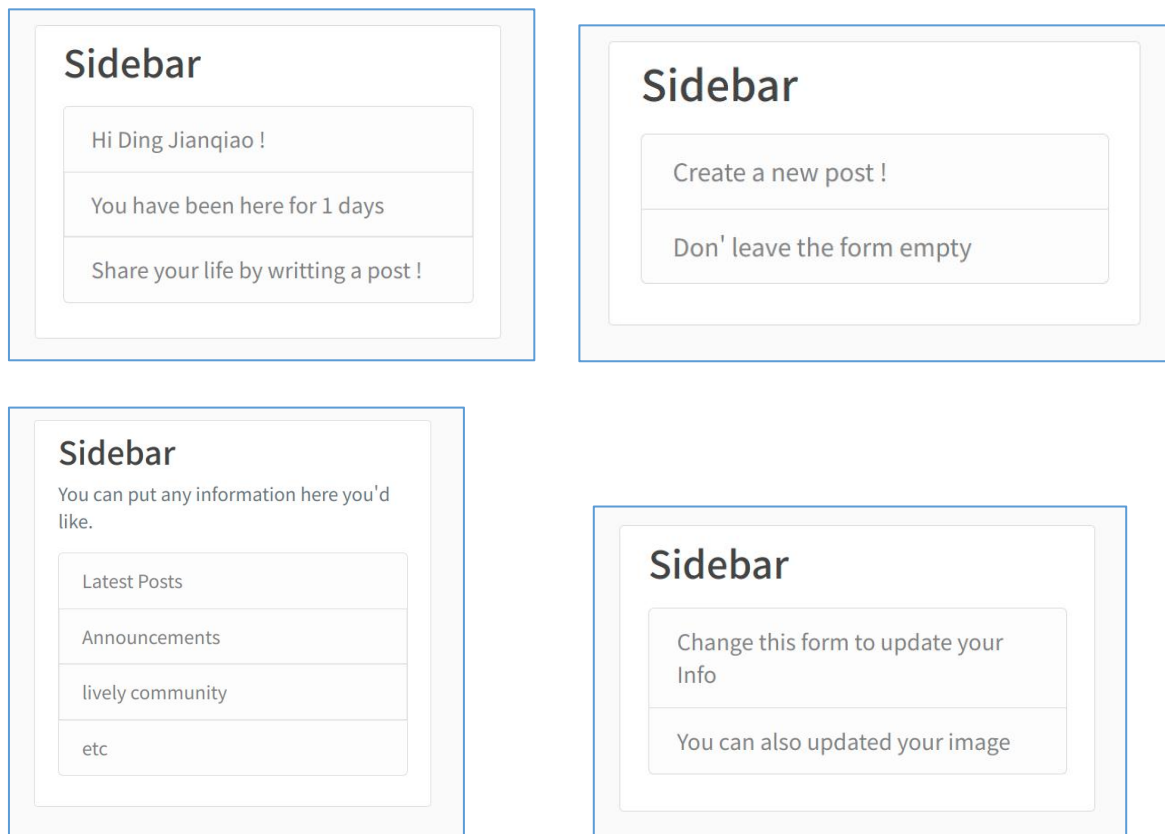
#### 5. Browsing Setting:

To satisfy user’s customized browsing requirements, “Settings” page is developed to store and change browser’s local feature. In this application, an option to change application’s main color is provided for users, in case of aesthetic fatigue.



## 6. Sidebar

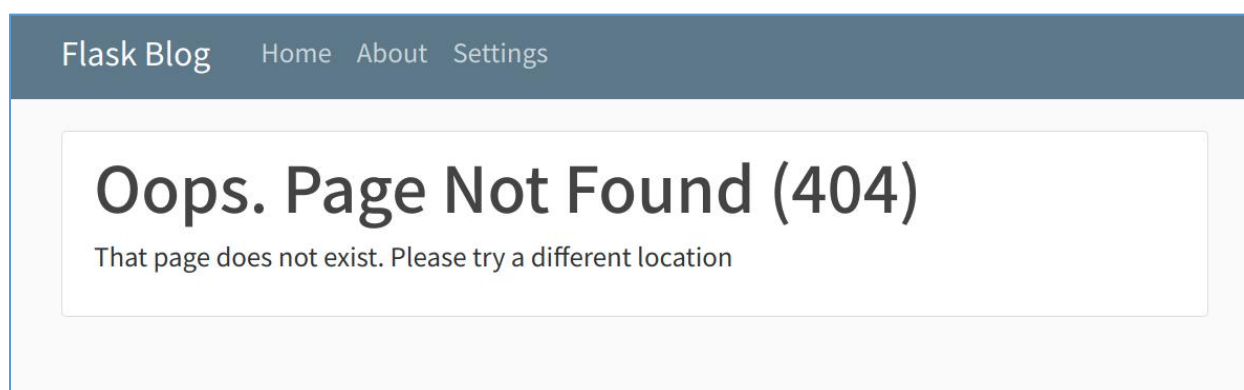
To enhance users' browsing experience, a guiding sidebar is built in order to give essential information or guidance.



( various sidebars )

## 7. Customized Error Pages:

Instead of pale "404" or "403" notifications, "Flask Blog" developed its own error pages, which keeps the browsing experience consistent and smooth.



## 8. Email Password Retrieve

If one user forget the password, then he/she can click “forget password” in login page. By entering user’s email address, the system will send a email to this address which contains one link to modify user’s password.

Flask Blog Home About Settings

An email has been sent with instructions to reset your password.

### Log In

Email

Password

☐ Remember Me

[Forgot Password?](#)

**Password Reset Request** 1013752750@qq.com 我<olym\_ding@163.com> 2021年01月03日 23:12 (星期日)

To reset your password, visit the following link:  
[http://localhost:5000/reset\\_password/eyJhbGciOiJIUzUxMiIsImhhdCI6MTYwOTY4NjczOCwiZXhwIjoxNjA5Njg4NTM4fQ.eyJ1c2VyX2lkIjo0fQ.Dpo0KqIXrFi-arrRv3sDA3REDu6wukBz0d-ST6P\\_6W0fxXeFp5-TR-80QGQoTUsIdG8m4G4LYjF6eILU52i\\_lg](http://localhost:5000/reset_password/eyJhbGciOiJIUzUxMiIsImhhdCI6MTYwOTY4NjczOCwiZXhwIjoxNjA5Njg4NTM4fQ.eyJ1c2VyX2lkIjo0fQ.Dpo0KqIXrFi-arrRv3sDA3REDu6wukBz0d-ST6P_6W0fxXeFp5-TR-80QGQoTUsIdG8m4G4LYjF6eILU52i_lg)

If you did not make this request then simply ignore this email and no changes will be made.

Flask Blog Home About Settings

### Reset Password

Password

Confirm Password



Flask Blog   Home   About   Settings

Your password has been updated! You are now able to log in

### Log In

Email

Password

☐ Remember Me

Login   [Forgot Password?](#)

## 9. Logging:

To better manager the application while running, application log is applied. In “Flask Blog”, logging is taken place with two handlers: one for terminal and one for static file.

Specific information is recorded with various level of logging:

routes.py   login.html   models.py   record.log   forms.py   Untitled-1

```
flaskblog > static > record.log
1 2021-01-03 15:57:51,822 - ERROR - 404 error
2 2021-01-03 17:38:26,921 - ERROR - 404 error
3 2021-01-03 19:28:04,373 - WARNING - unsuccessful login admin@qq.com
4 2021-01-03 19:28:09,290 - WARNING - unsuccessful login admin@qq.com
5 2021-01-03 19:28:19,720 - WARNING - unsuccessful login olym_ding@163.com
6 2021-01-03 19:28:45,722 - INFO - Account created: User('admin', 'admin@qq.com', 'default.jpg')
7 2021-01-03 19:28:51,926 - WARNING - unsuccessful login admin@qq.com
8 2021-01-03 19:28:56,217 - INFO - user log in: User('admin', 'admin@qq.com', 'default.jpg')
9 2021-01-03 19:31:28,660 - INFO - Post created: Post('The first post', '2021-01-03 19:31:28.639215')
10 2021-01-03 19:32:57,676 - ERROR - 404 error
11 2021-01-03 20:11:57,371 - INFO - user log out: User('admin', 'admin@qq.com', 'default.jpg')
12 2021-01-03 21:46:36,397 - ERROR - 404 error
13 2021-01-03 21:46:38,541 - ERROR - 404 error
14 2021-01-03 21:49:43,142 - INFO - Account created: User('Ding Jianqiao', '1013752750@qq.com', 'default.jpg')
15 2021-01-03 21:51:50,123 - INFO - user log in: User('Ding Jianqiao', '1013752750@qq.com', 'default.jpg')
16 2021-01-03 21:52:16,644 - INFO - user log out: User('Ding Jianqiao', '1013752750@qq.com', 'default.jpg')
17 2021-01-03 21:58:50,945 - INFO - user log in: User('Ding Jianqiao', '1013752750@qq.com', 'default.jpg')
18 2021-01-03 21:59:59,882 - INFO - Post created: Post('Finally finish the course work 2 !', '2021-01-03 21:59:59.868062')
19 2021-01-03 22:00:08,391 - INFO - Post deleted: Post('Finally finish the course work 2 !', '2021-01-03 21:59:59.868062')
20 2021-01-03 22:02:13,544 - INFO - user log out: User('Ding Jianqiao', '1013752750@qq.com', 'default.jpg')
21 2021-01-03 22:02:16,904 - ERROR - 404 error
22 2021-01-03 22:06:18,644 - ERROR - 404 error
23 2021-01-03 22:08:21,332 - ERROR - 404 error
24 2021-01-03 22:09:59,425 - ERROR - 404 error
25 2021-01-03 22:10:16,476 - ERROR - 404 error
26 2021-01-03 22:20:19,850 - INFO - user log in: User('Ding Jianqiao', '1013752750@qq.com', 'default.jpg')
27 2021-01-03 22:20:40,138 - INFO - Post created: Post('Finally finish the course work 2 !', '2021-01-03 22:20:40.131685')
28 2021-01-03 22:20:46,705 - ERROR - 404 error
29 2021-01-03 22:20:51,870 - ERROR - 404 error
30 2021-01-03 22:20:59,653 - INFO - Post deleted: Post('Finally finish the course work 2 !', '2021-01-03 22:20:40.131685')
31 2021-01-03 22:22:36,267 - INFO - Post created: Post('Finally finish the course work 2 !', '2021-01-03 22:22:36.260461')
32 2021-01-03 22:23:34,497 - ERROR - 404 error
33 2021-01-03 22:26:56,448 - INFO - Comment made: Comment('id: 1, user: 2, post: 1,
content: wow !')
34
35 2021-01-03 22:27:08,560 - INFO - Comment made: Comment('id: 2, user: 2, post: 1,
content: Good job !')
36
37 2021-01-03 22:28:25,077 - INFO - Comment made: Comment('id: 3, user: 2, post: 1,
content: Nice work !!')
38
39 2021-01-03 22:28:32,207 - INFO - Comment made: Comment('id: 4, user: 2, post: 1,
content: finally finished !!!')
40
41 2021-01-03 22:28:39,110 - INFO - Comment made: Comment('id: 5, user: 2, post: 1,
```

unittest\*   Python 3.6.9 64-bit   Server not selected

# Deployment

This application has been deployed on Tencent Cloud Server, and its address is:

[140.143.18.41:8000](http://140.143.18.41:8000)

In convenience of course work examination, one tester account is provided as below:

Email: [test@demo.com](mailto:test@demo.com)

Password: testing

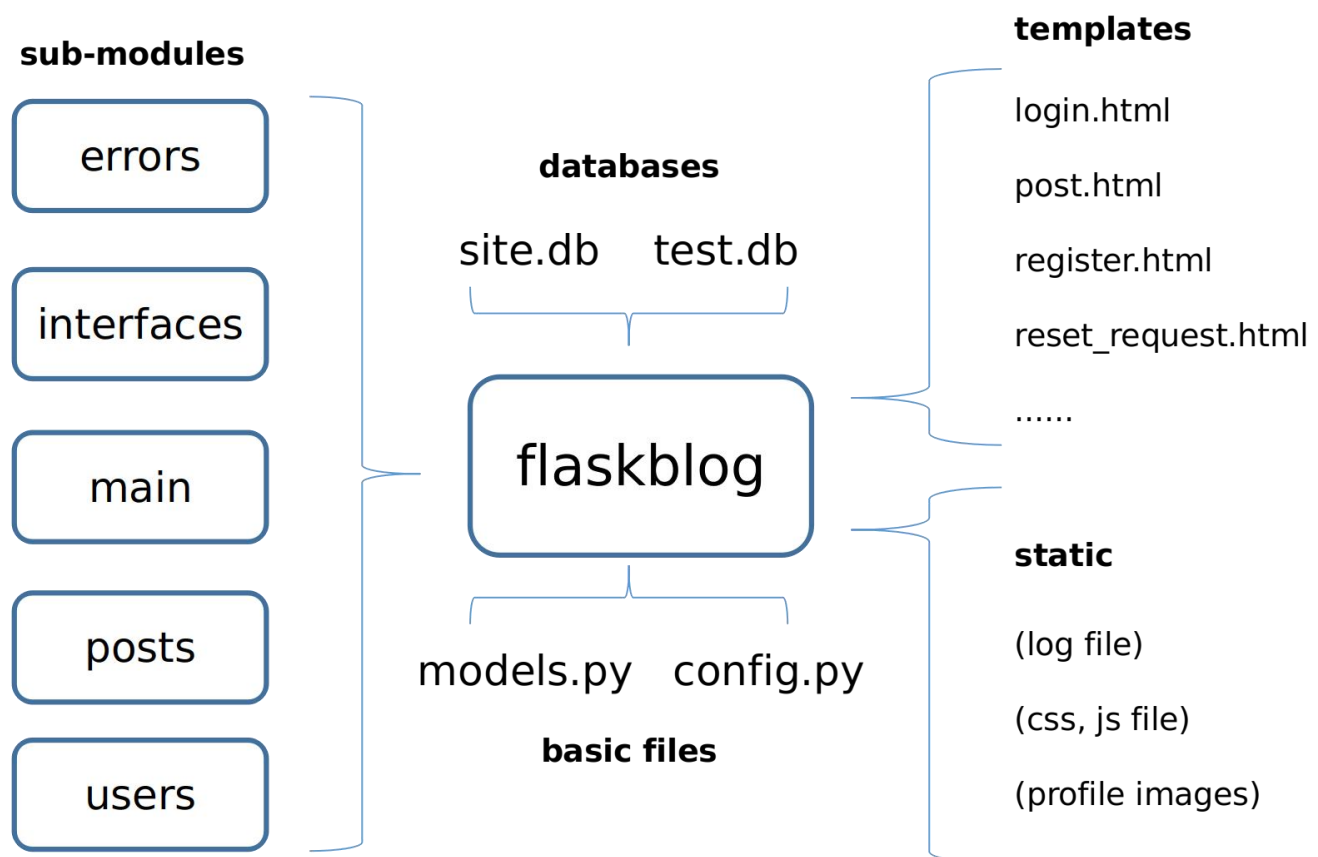
However, to experience the whole functionality of this application, a registration with correct email address is strongly recommended.

( Note: this server is free and accessible from 2021.1.1 to 2021.1.7. If the application failed due to server limitation, please contact me ASAP to restart a new server.)

## Analysis and Evaluation of Application Architecture:

### ● File Structure

For “Flask Blog”, its application architecture can be described as this picture below:



According to this image, “Flask Blog” application architecture can be divided into five parts, which are:

#### 1. databases:

Two databases with the same structure.

“site.db” is used as the main database for application, and test.db serves when unit tests are applied.

## 2. static:

“static” is a folder storing static files, such as Javascript dependencies, CSS files, and user’s profile images.

## 3. templates:

“template” is another folder containing HTML files for Jinja2 engine to render, which can be considered as the front end part.

## 4. basic files:

Two python files which store significant application information.

For models.py file, it contains the structure of database applied, and for config.py, there is one object (or dictionary) variable which contains all configuration needed when a Flask application is created, such as secret key, or mail settings.

## 5. sub-modules:

This is the most important part of the whole application.

At the beginning of development, all routes and forms are constructed inside one single file respectively, that are routes.py and forms.py. However, as development carried on, multiple functions piled up, making it difficult to manage source code.

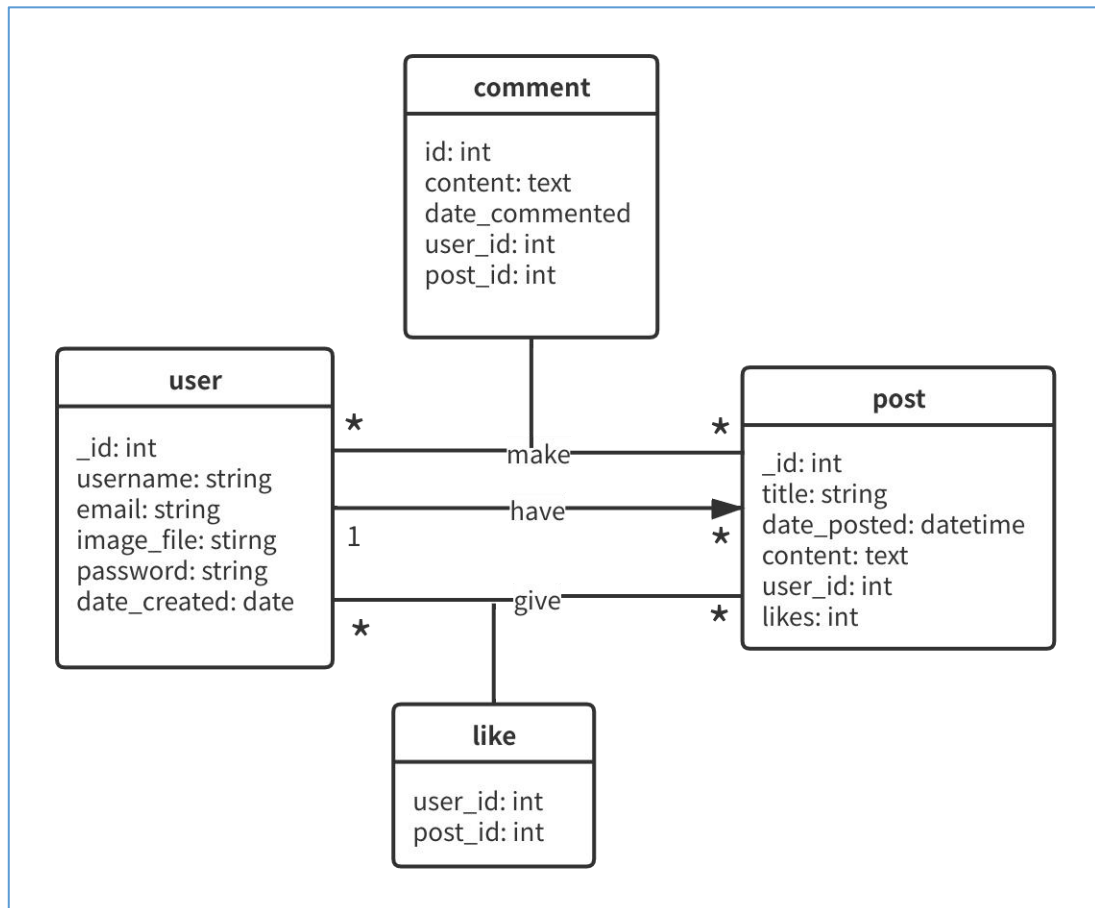
By dividing original “flaskblog” module into five sub-modules, the code structure is optimized to be clear. Each sub-module is registered in “flaskblog” module, and imported using “Blueprint”, which is provided by flask to better manage code structure.

```
from flaskblog.users.routes import users
from flaskblog.posts.routes import posts
from flaskblog.main.routes import main
from flaskblog.errors.handlers import errors
from flaskblog.interface.interfaces import interfaces
app.register_blueprint(users)
app.register_blueprint(posts)
app.register_blueprint(main)
app.register_blueprint(errors)
app.register_blueprint(interfaces)
```

“main” sub-module is used to store the routes handling register, login, logout, and a form of local settings. For “users” and “posts” sub-modules, they store routes and forms relevant with users and posts respectively. “errors” sub-module contains multiple handlers for errors (403, 404, 500), and “interfaces” sub-module mainly handles AJAX requests.

## ● Database Structure:

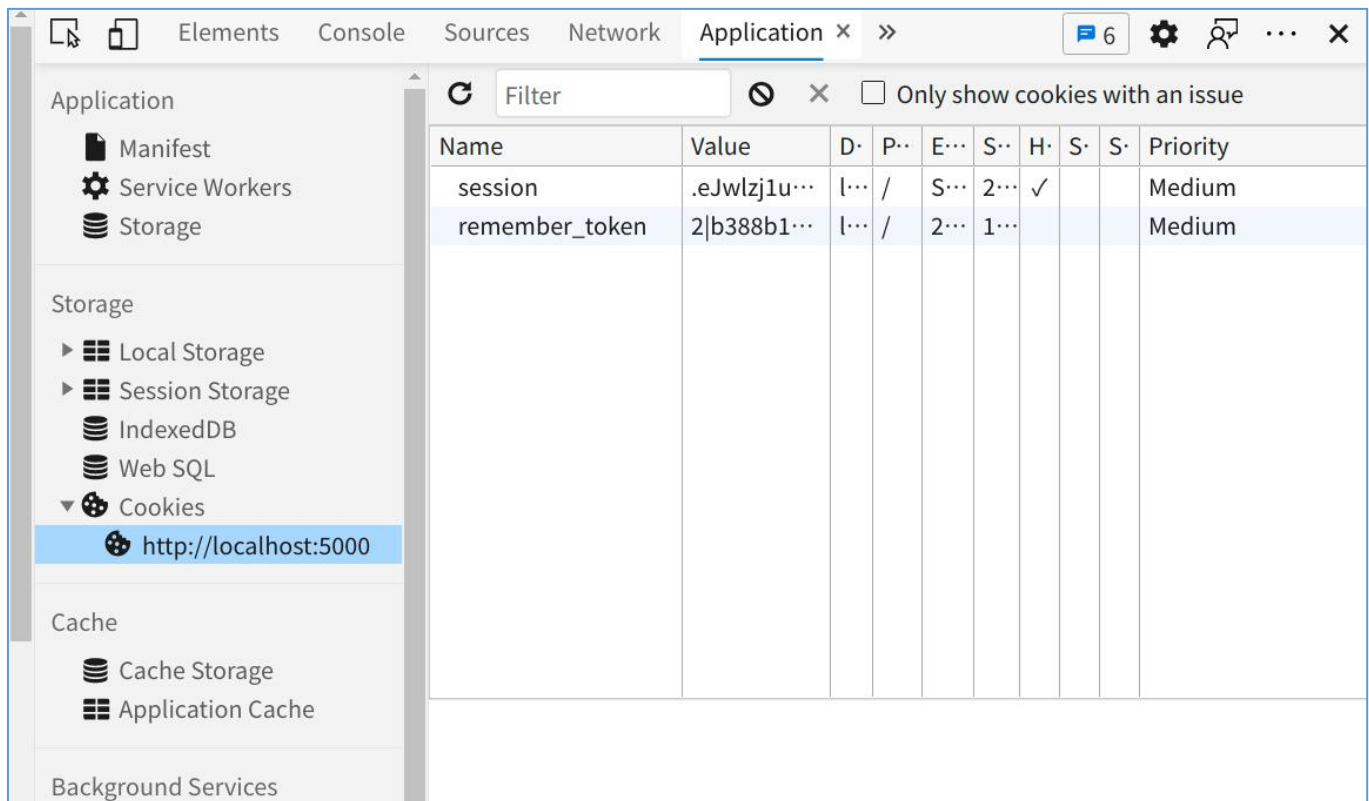
In this application, four tables are designed to represent user, post, like and comment. Like and comment are two Many-to-Many relationships between user and post. Detailed information can be viewed in models.py, and the UML diagram of this database is drawn as below:



## ● Advanced Features:

In this application, to improve user's experience, multiple advanced features are implemented.

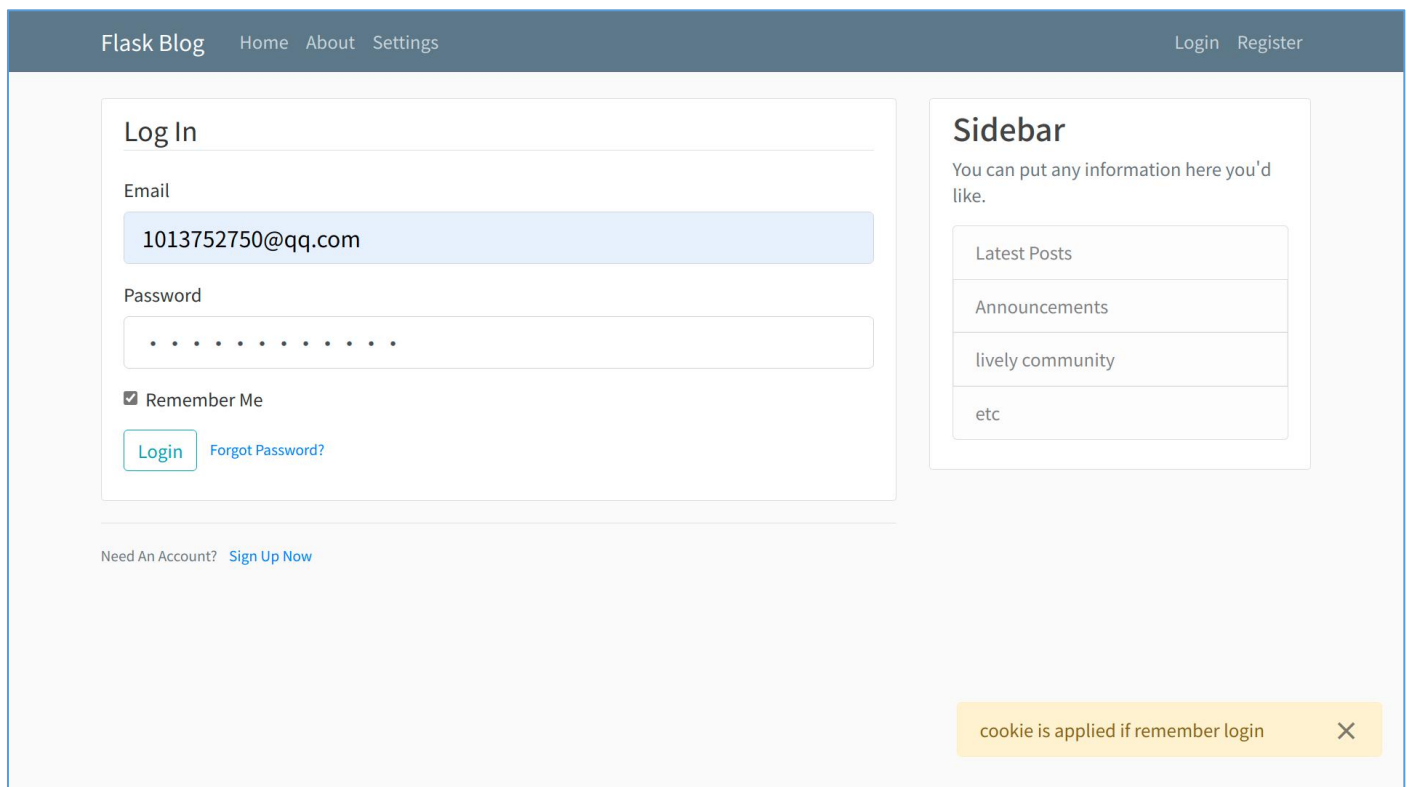
### 1. Cookie and session are applied for authentication:



The screenshot shows the Chrome DevTools Application tab. The left sidebar lists various storage areas: Application (Manifest, Service Workers, Storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), Cache (Cache Storage, Application Cache), and Background Services. The 'Cookies' section under 'Storage' is expanded, showing the URL 'http://localhost:5000'. The main pane displays a table of cookies with the following data:

Name	Value	D	P	E	S	H	S	S	Priority
session	.eJwlzj1u...	l...	/	S...	2...	✓			Medium
remember_token	2 b388b1...	l...	/	2...	1...				Medium

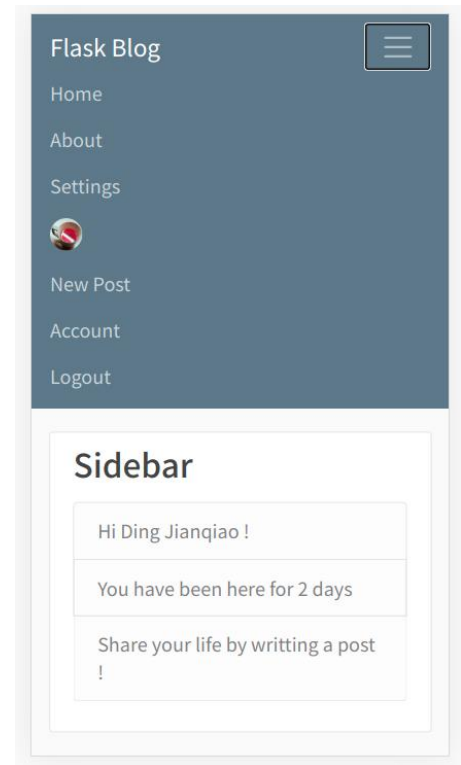
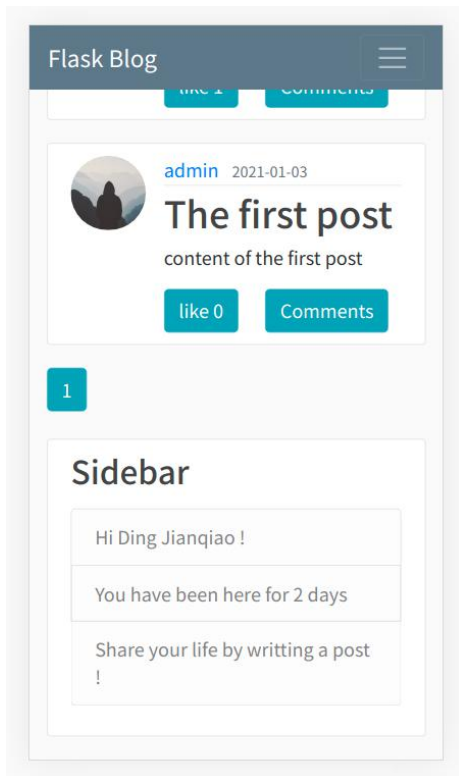
As shown above, session is responsible for authenticating and cookie is used for remembering user's login. Each time user clicks on the "Remember Me" checkbox, a notification will pump up to inform user that cookie is collected.



The screenshot shows the Flask Blog login page. The header includes 'Flask Blog' and navigation links 'Home', 'About', 'Settings', 'Login', and 'Register'. The main content area has a 'Log In' form with fields for 'Email' (1013752750@qq.com) and 'Password' (masked with dots). There is a checked 'Remember Me' checkbox and 'Login' and 'Forgot Password?' buttons. A 'Sidebar' section on the right contains 'Latest Posts', 'Announcements', 'lively community', and 'etc'. At the bottom, there is a notification bar that says 'cookie is applied if remember login' with a close button.

## 2. Bootstrap Beautify and Responsive Layout

Bootstrap is imported and applied to beautify this application. Moreover, it provides an easy access to responsive layout, which increase mobile browsing experience.



( mobile responsive layout )



### 3. Jquery and AJAX request

Jquery provides much conciser operations than Javascript itself. Also, it's encapsulation of AJAX make it easy to send asynchronous requests.

In this application, to achieve features such as "like" or "comment", it is no doubt a bad option to refresh the page each time a request is sent, since the **user's experience will be inconsistent and awful**. Thus, AJAX is required to make dynamic requests without refreshing the page.

```
// 点赞
$like_btns = $(".like") // 为了防止下面回调函数中的this指向错误;
$like_btns.on('click', function () {

    $like_btn = $(this)
    $.ajax({
        type: "POST",
        url: '/like/' + $(this).attr('ref'),
        success: function(data) {
            // 还没点赞
            if (data.logged) {
                if (!data.founded) {
                    notify("like one post !", "info")
                    $like_btn.children().html(data.count)
                }
                // 点过赞了
            } else {
                notify("remove one like", "info")
                $like_btn.children().html(data.count)
            }
        } else {
            notify("please log in first !", "danger")
            $like_btn.children().html(data.count)
        }
    },
    error: () => {
        notify("request error !", 'danger')
    }
})
})
```

( code of giving a like using AJAX )

## 4. Local Storage:

In the “settings” page, user’s option is stored within HTML local storage, which avoids storing data in server side and sending requests.

Application	Filter	
	Key	Value
Manifest	color	#cf3030
Service Workers		
Storage		
Storage		
Local Storage		
http://localhost:5000		
Session Storage		
IndexedDB		
Web SQL		

## ● Unit tests:

Instead of manually test functions one by one, automatic unit tests are better choices. In test.py file, a group of unit tests are written to check multiple functionality of this application.

```
#####
#### tests ####
#####

# main page
def test_main_page(self):
    response = self.app.get('/', follow_redirects=True) # 发送请求
    self.assertEqual(response.status_code, 200) # 接收信号

# register
def test_valid_user_registration(self): # 正确登录
    response = self.register('Tom', '1013752750@qq.com', 'TheFirstPassword', 'TheFirstPassword')
    self.assertEqual(response.status_code, 200)
    self.assertIn(b'Your account has been created!', response.data)

def test_invalid_user_registration_different_passwords(self): # 前后密码不一致
    response = self.register('Tom', '1013752750@qq.com', 'TheFirstPassword', 'TheDifferentPassword')
    self.assertEqual(response.status_code, 200)
    self.assertIn(b'Field must be equal to password.', response.data)

def test_invalid_user_registration_duplicate_email(self):
    response = self.register('Tom', '1013752750@qq.com', 'TheFirstPassword', 'TheFirstPassword')
    self.assertEqual(response.status_code, 200)
    response = self.register('AnotherTom', '1013752750@qq.com', 'TheFirstPassword', 'TheFirstPassword')
    self.assertIn(b'That email is taken. Please choose a different one.', response.data)

def test_invalid_user_registration_duplicate_username(self):
    response = self.register('Tom', '1013752750@qq.com', 'TheFirstPassword', 'TheFirstPassword')
    self.assertEqual(response.status_code, 200)
    response = self.register('Tom', '1013752750@gmail.com', 'TheFirstPassword', 'TheFirstPassword')
    self.assertIn(b'That username is taken. Please choose a different one.', response.data)

# login
def test_valid_login(self): # 正确登录
    response = self.register('Tom', '1013752750@qq.com', 'TheFirstPassword', 'TheFirstPassword')
    self.assertEqual(response.status_code, 200)
    response = self.login('1013752750@qq.com', 'TheFirstPassword')
    self.assertEqual(response.status_code, 200)
    self.assertIn(b'Login successfully', response.data)
```



By running test.py, the result will be rapidly given:

```
python test.py
/home/olym/.local/lib/python3.6/site-packages/flask_sqlalchemy/_init_.py:834: SQLAlchemyWarning: SQLAlchemy_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future. Set it to True or False to suppress this warning.
'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
/home/olym/projects/flask_blog/web-cw2/flaskblog/users/routes.py:44: DeprecationWarning: The 'warn' method is deprecated, use 'warning' instead
    current_app.logger.warn(f"unsuccessful login {form.email.data}")
[2021-01-04 00:50:08,642] WARNING in routes: unsuccessful login Null@null.com
..[2021-01-04 00:50:08,937] INFO in routes: Account created: User('Tom', '1013752750@qq.com', 'default.jpg')
[2021-01-04 00:50:09,160] WARNING in routes: unsuccessful login 1013752750@qq.com
..[2021-01-04 00:50:09,463] INFO in routes: Account created: User('Tom', '1013752750@qq.com', 'default.jpg')
.[2021-01-04 00:50:09,733] INFO in routes: Account created: User('Tom', '1013752750@qq.com', 'default.jpg')
..[2021-01-04 00:50:10,159] INFO in routes: Account created: User('Tom', '1013752750@qq.com', 'default.jpg')
[2021-01-04 00:50:10,430] INFO in routes: user log in: User('Tom', '1013752750@qq.com', 'default.jpg')
[2021-01-04 00:50:10,447] INFO in routes: user log out: User('Tom', '1013752750@qq.com', 'default.jpg')
.[2021-01-04 00:50:10,848] INFO in routes: Account created: User('Tom', '1013752750@qq.com', 'default.jpg')
[2021-01-04 00:50:11,080] INFO in routes: user log in: User('Tom', '1013752750@qq.com', 'default.jpg')
.[2021-01-04 00:50:11,338] INFO in routes: Account created: User('Tom', '1013752750@qq.com', 'default.jpg')
.
-----
Ran 10 tests in 2.795s

OK
```

## Potential Security Problem:

At the beginning of this application's development, user's password is stored without encryption, which raised high security risk when there is a leak of database's information.

To avoid this problem, one python encrypt library “BCrypt” is imported to encrypt user’s password inside the database.

数据库结构

浏览数据

编辑杂注

执行 SQL

表(I):

user

新建记录



删除记录

	id	username	email	image_file	password	date_created
	过滤	过滤	过滤	过滤	过滤	过滤
1	1	admin	admin@qq.c...	default.jpg	\$2b\$12\$zvPRlzz1Xtf...	2021-01-03
2	2	Ding Jianqiao	1013752750...	4f67d5e7365...	\$2b\$12\$qybex/u2L1...	2021-01-03
3	3	Tester	test@demo.c...	default.jpg	\$2b\$12\$w8HEpTFqc...	2021-01-03
4	4	163	olym_ding@...	default.jpg	\$2b\$12\$rkPfzd1STH...	2021-01-03

( encrypted passwords )

Moreover, when some users are updating their new profile\_file, the image file may be too large that it occupies a large amount of memory, raise the server storage load.

Once again, another module called “Pillow” is used to compress these image files to store storage.

	名称(N): <b>header.jpg</b>		名称(N): <b>4f67d5e7365c3eec.jpg</b>
	类型: JPEG 图像 (image/jpeg)		类型: JPEG 图像 (image/jpeg)
	大小: 2.4 MB (2,354,584 字节)		大小: 3.3 KB (3,273 字节)
			父文件夹: /home/olym/projects/flask_b...

( comparison, left is original, right is the compressed one )

## Reference:

1. PATKENNEDY79@GMAIL.COM (2016, November 22). Unit Testing a Flask Application [Blog post]. Retrieved from <http://www.patricksoftwareblog.com/unit-testing-a-flask-application/>
2. Julian Nash (2019, Mar 05). Password hashing in Python with Bcrypt[Blog post]. Retrieved from <https://pythonise.com/categories/python/python-password-hashing-bcrypt>