几个问题：

1.模块，函数命名不规范，参考下python pep8规范文档。

2.用户上传头像和修改资料没有登录和身份验证。

3.cookie设置的email地址，不安全。

4.email没有设置unique key，判断用户是否存在没有考虑并发问题。

5.mysql语句用了拼接，有sql注入风险。

6.数据库用户名密码配置不能直接提交到（代码）仓库，需要单独管理或者加密处理。

7.密码用md5处理不是特别安全

8.存储头像考虑下如果web服务不是部署的单机，用什么方式存储才能保证不同的web机器都能访问到

学习点：

1.session和cookie原理

2.csrf和xss注入

3.sql注入

https://tech.meituan.com/2018/09/27/fe-security.html

https://tech.meituan.com/2018/10/11/fe-security-csrf.html

https://dropbox.tech/security/how-dropbox-securely-stores-your-passwords

有几个很基础的问题，都是学习任务里面的

所以学习的时候不要偷工减料，要学透彻，不是浮于表面

你们入职的时候还要做一个项目的，那个项目很重要，如果是现在这样，那基本没法及格的，可以抓紧时间多学习，夯实基础。

资源：

1、https://fasionchan.com/network/

问题修改：

1.模块，函数命名不规范，参考下python pep8规范文档。

根据pep8规范，

（1）模块命名尽量短小，使用全部小写的方式，可以使用下划线。

（2）函数命名使用全部小写的方式，可以使用下划线。

2.用户上传头像和修改资料没有登录和身份验证。√

修改用户个人信息时如何校验身份：

通过客户端携带Token认证

将Token存放在cookie中，使用以下方法可以防止CSRF攻击。

双重提交Cookie。利用CSRF攻击不能获取到用户Cookie的特点，我们可以要求Ajax和表单请求携带一个Cookie中的值。这是一个比较有效的CSRF防护方法，只要页面没有XSS漏洞泄露Token，那么接口的CSRF攻击就无法成功。


```
        data: {
            "image":data.toString(),
            "email":email,
            //提交表单时要求加入token，用于后端校验，防止CSRF攻击
            "token":getCookie( name: "token")
        },
```

```
# 获取数据
image   = form.getvalue('image')
email   = form.getvalue('email')
form_token   = form.getvalue('token')

# 防止csrf攻击，校验表单提交的cookie值是否符合要求
if token != "" and token != form_token:
    print("Content-type:text/html")
    print()
```

Token是目前广泛使用的一种保持会话状态的技术，与以前的cookie、session共同存在于如今各大网站架构中。

```
1 pyjwt提供的jwt.encode(payload,key,algorithm)方法可以让我们快速的生成token；需要
```

```
payload      公有声明和私有声明组成的字典，根据需要进行添加
key          自定义的加密key。重要，不能外泄
algorithm      声明需要使用的加密算法，如’HS256’
```

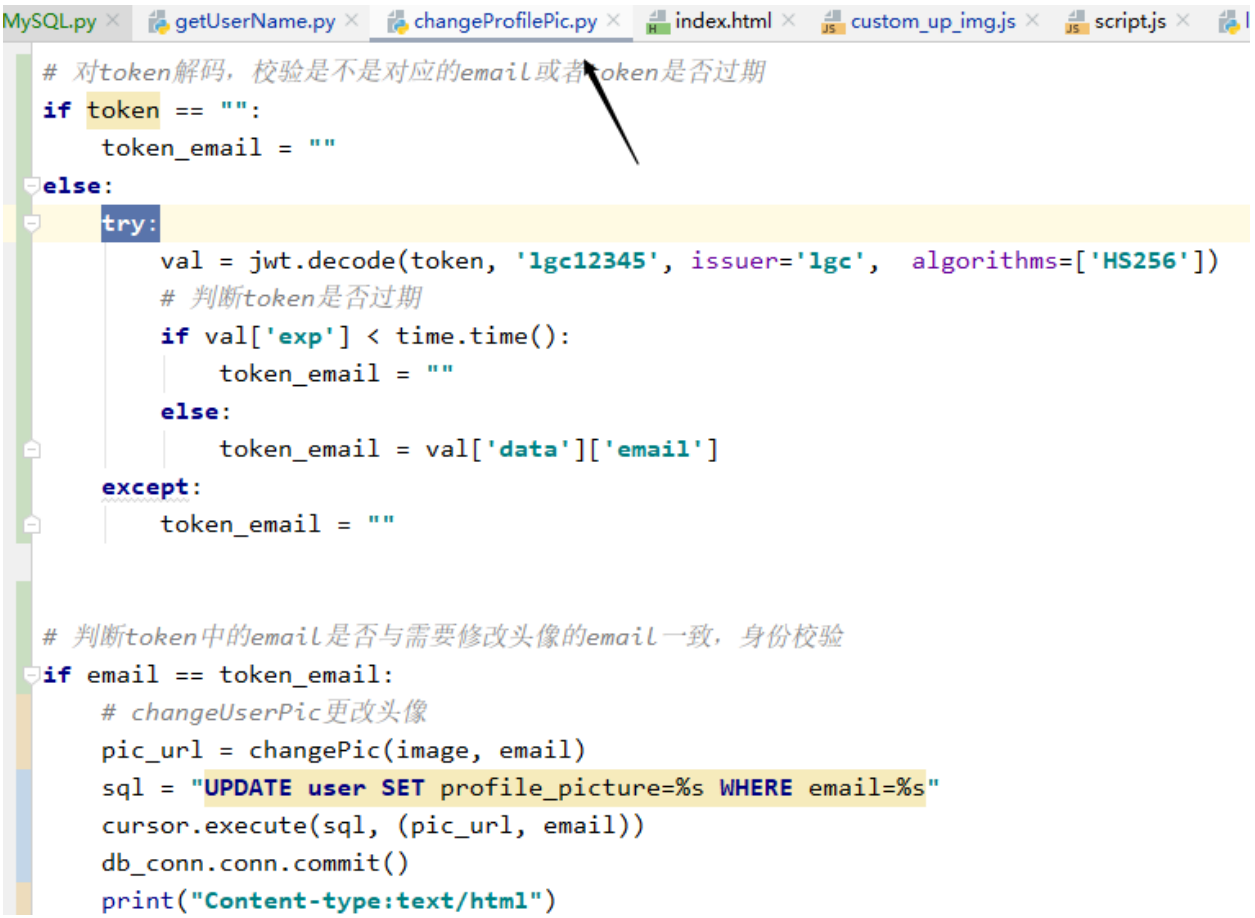
登录验证：

前端查看cookie中储存的token是否有效，因为cookie的有效期设置了一天，且退出登录时会清除cookie，这样可以判断是否当前是登录状态

身份验证：

获取前端传回来的token，解码，判断token中的个人信息，且判断token是否过期（token有效期为一天）

修改头像：



```
    print()
    # print(code)
    print(image)
else:
    print("Content-type:text/html")
    print()
    # print(code)
```

前端校验登录身份过期或者身份校验失败：

```
            success: function(data, textStatus){
                // 身份验证错误或者cookie失效
                if (data === "" || getCookie( name: "token") === "") {
                    if (getCookie( name: "token") === "") {
                        alert("身份过期，请重新登录!");
                    } else {
                        alert("身份校验失败，无法更改头像，请重新登录!");
                    }
                     window.location.href = "../register";
                } else {
```

3.cookie设置的email地址，不安全。 √

cookie里不设置email地址，设置服务端传过来的token

（1）客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 里或者 Local Storage 里;

（2）客户端每次向服务端请求资源的时候需要带着服务端签发的 Token;

（3）服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据。

请求 Cookie　☐ 显示滤除的请求 Cookie

名称	值	Domain	Path
Pycharm-900df24b	6a7135d9-137f-4a8c-8929-25c57b296266	localhost	/cgi-bin
token	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2...	localhost	/cgi-bin
Pycharm-900df24b	6a7135d9-137f-4a8c-8929-25c57b296266	localhost	/

4.email没有设置unique key，判断用户是否存在没有考虑并发问题。

（1）email设置unique key √

```sql
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(64) COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
  `email` varchar(64) COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
  `profile_picture` varchar(256) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `email` (`email`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=29 DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

（2）判断用户存在考虑并发问题：

可能性：两个用户使用同一个，多线程操作，判断用户是否存在那部分的代码加锁即可

Docs » 第十二章：并发编程 » 12.4 给关键部分加锁 Edit on GitHub

12.4 给关键部分加锁

问题

你需要对多线程程序中的临界区加锁以避免竞争条件。

解决方案

要在多线程程序中安全使用可变对象，你需要使用 threading 库中的 `Lock` 对象，就像下边这个例子这样：

```python
import threading

class SharedCounter:
    '''
    A counter object that can be shared by multiple threads.
    '''
    def __init__(self, initial_value = 0):
        self._value = initial_value
        self._value_lock = threading.Lock()

    def incr(self,delta=1):
        '''
        Increment the counter with locking
        '''
        with self._value_lock:
             self._value += delta

    def decr(self,delta=1):
        '''
        Decrement the counter with locking
        '''
        with self._value_lock:
             self._value -= delta
```

`Lock` 对象和 `with` 语句块一起使用可以保证互斥执行，就是每次只有一个线程可以执行 with 语句包含的代码块。with 语句会在这个代码块执行前自动获取锁，在执行结束后自动释放锁。

```python
lock = threading.Lock()
```

```
with lock:
    flag = checkUserExist(email)
    if not flag:
        pass
```

5.mysql语句用了拼接，有sql注入风险。 √

取消SQL拼接，通过以下方式防止SQL注入：

mysql.connector用 %s 作为占位符

```
cursor.execute('insert into user (name,password) value (%s,%s)',(name,passwor
```

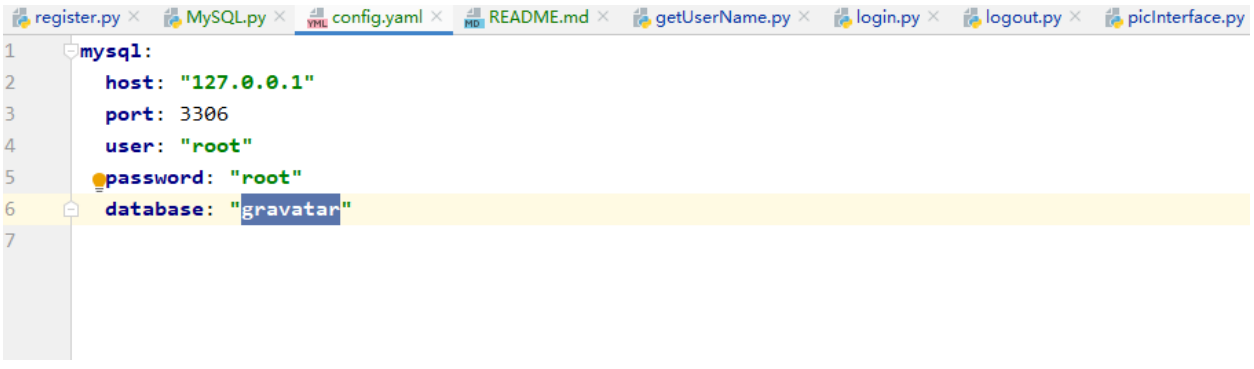
而不能把参数拼接到sql语句中，这样数据库就容易被sql注入攻击，比如

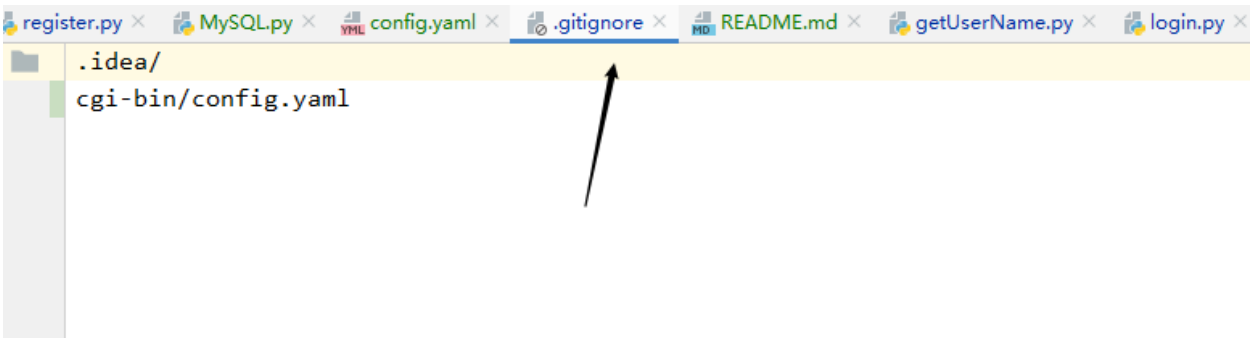
```
cursor.execute('select * from user where user=%s and password=%s'%(name,passw
```

原理： python并不支持mysql预编译语句，其实“参数化”是在MySQLdb中通过转义字符串然后直接将它们插入到查询中而不是使用MYSQL_STMT API来完成的。因此，unicode字符串必须经过两个中间表示（编码字符串，转义编码字符串）才能被数据库接收。也就是说，我们传入的参数并不是直接拼接到sql语句中，而是经过了字符转换，因此参数化查询可以有效防止sql注入

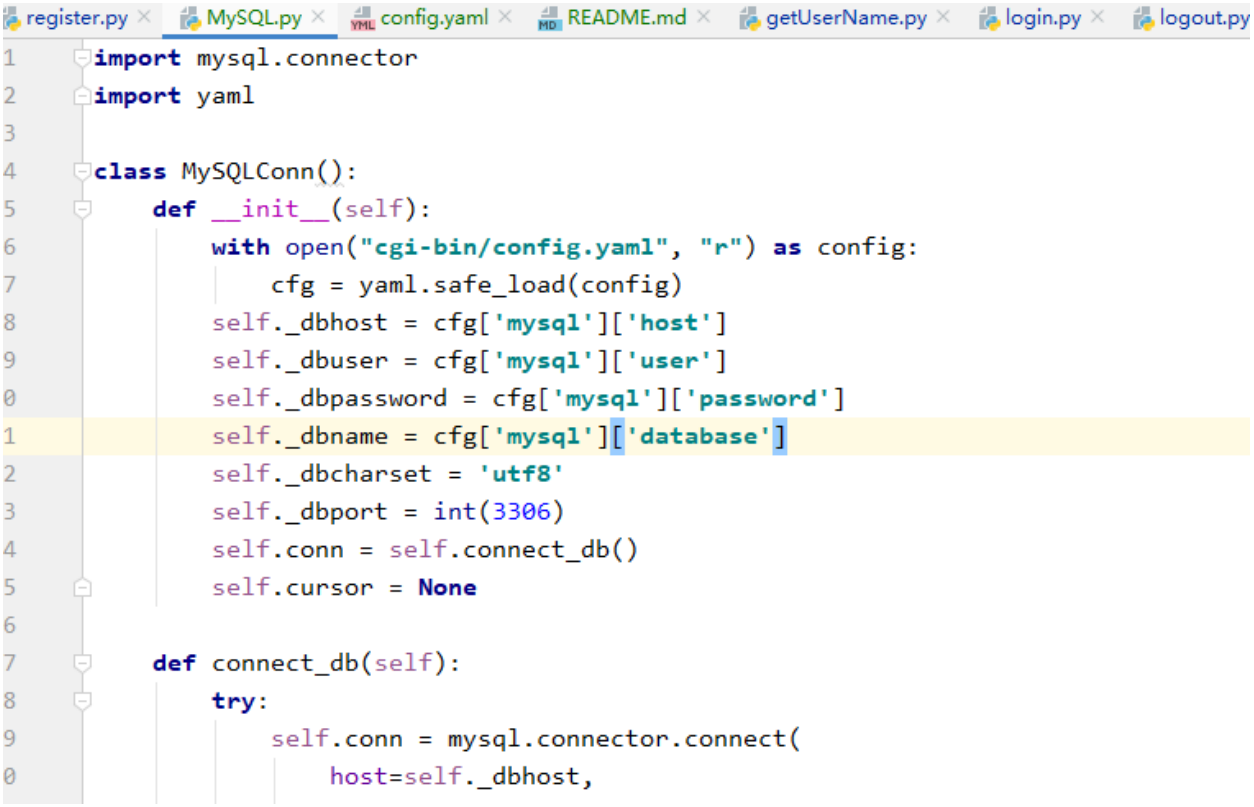
6.数据库用户名密码配置不能直接提交到仓库，需要单独管理或者加密处理。 √

config.yaml存放数据库配置信息，且不上传github





数据库连接单独管理，方便配置信息（未设置数据库连接池）：




```
db_conn = MySQL.MySQLConn()
cursor = db_conn.get_cursor()
```

7. 密码用md5处理不是特别安全 √

密码存储：

```
    # 对密码进行加密，插入密码表
    # 1. 准备一个字符集用于生成随机字符串，字符集只包含字母和数字
    charset = string.ascii_letters + string.digits
    # 2. 生成一个随机字符串作为盐，长度为32
    salt = ''.join(random.sample(charset, 32))
    # 3. 将盐和密码拼在一起算哈希值
    salted_hash = hashlib.sha256((salt + password).encode(encoding='UTF-8')).hexdigest()
    # 4. 将盐和哈希值拼在一起保存到数据库
    salt_password = '{}-{}'.format(salt, salted_hash)
```

密码校验：

```python
# 从数据库取出密码，盐值，加盐哈希值分离，对登录的密码加盐哈希，判断密码是否相同
salt_password = [x[0] for x in cursor.fetchall()][0]
salt, salted_hash = salt_password.split('-')
password_encode = hashlib.sha256((salt + password).encode(encoding='UTF-8')).hexdigest()

if password_encode == salted_hash:
    return True
else:
    return False
```

8.存储头像考虑下如果web服务不是部署的单机，用什么方式存储才能保证不同的web机器都能访问到

抽象问题：分布式访问修改同一数据问题

web服务如果不是部署的单机有以下方式解决不同的web机器都能访问到头像：

把用户头像单独存储为头像数据库，不同的web服务都去访问同一个头像数据库，但是这样，在修改用户的头像时，需要加分布式锁（redis、zookeeper、数据库等实现），避免出现并发修改问题。

当头像数据库不足以存放所有头像，即用户量较大时，需要扩容（添加多个头像数据库）时，可能需要通过一致性哈希算法或者其他方法解决头像存储位置问题，另外衍生出分布式问题可能需要结合CAP、BASE理论等解决。

分布式访问时，用户访问路由到Nginx，通过适当的路由算法路由到不同的web机器，避免单一机器负载过大。

访问数据库量太大、数据库扩容时，通过主从复制、分库分表、读写分离、添加前置缓存（redis）等。

其他问题总结：

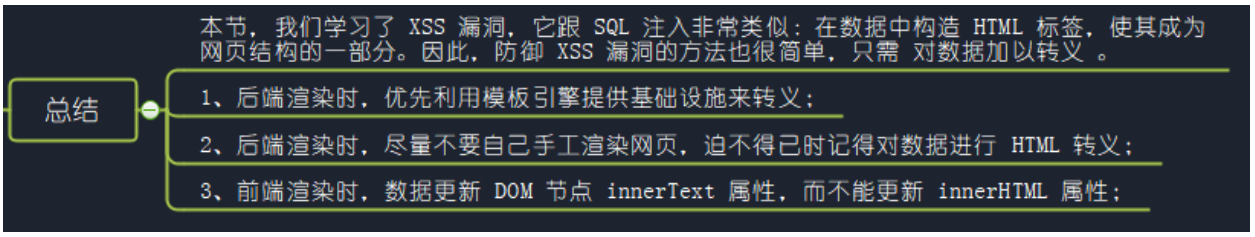
数据库连接，不要连接太多，数据库账号密码保存位置，统一配置。数据库连接池

数据库连接、用户信息抽象orm

块注释

设置cgi-bin为source root

防止XSS、CSRF攻击（防止xss攻击，对用户输入都进行校验）


总结
本节，我们学习了 XSS 漏洞，它跟 SQL 注入非常类似：在数据中构造 HTML 标签，使其成为网页结构的一部分。因此，防御 XSS 漏洞的方法也很简单，只需 对数据加以转义 。
1、后端渲染时，优先利用模板引擎提供基础设施来转义；
2、后端渲染时，尽量不要自己手工渲染网页，迫不得已时记得对数据进行 HTML 转义；
3、前端渲染时，数据更新 DOM 节点 innerText 属性，而不能更新 innerHTML 属性；