

# 缩小版fullStack project

- Frontend: 用tkinter来实现GUI
- Database: 利用json来代替数据库
- 加密: 简单的加密方法（ID加密-用的凯撒加密法）&hash  
SHA256加密法
- 简单实现聊天工具的一些功能

# 项目动机

- 想要了解微信或kakaotalk等聊天工具是如何进行注册、登录、登录后添加联系人，与联系人那些经年累月的聊天记录和联系人信息是存储在何处。还要了解软件公司是如何对用户的真实姓名以及设置的密码进行保密工作的。据此用Python尝试实现上述这几点。

# 项目目标

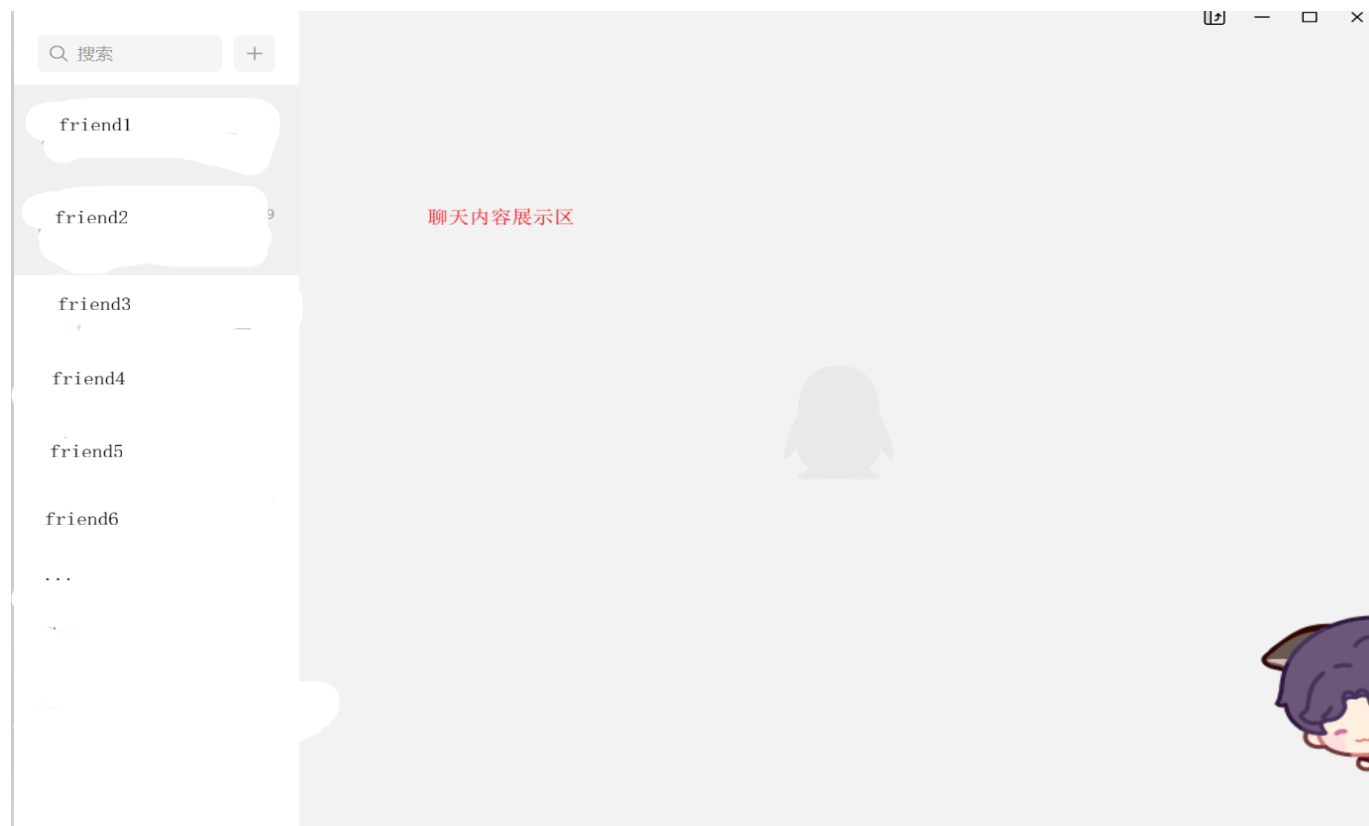
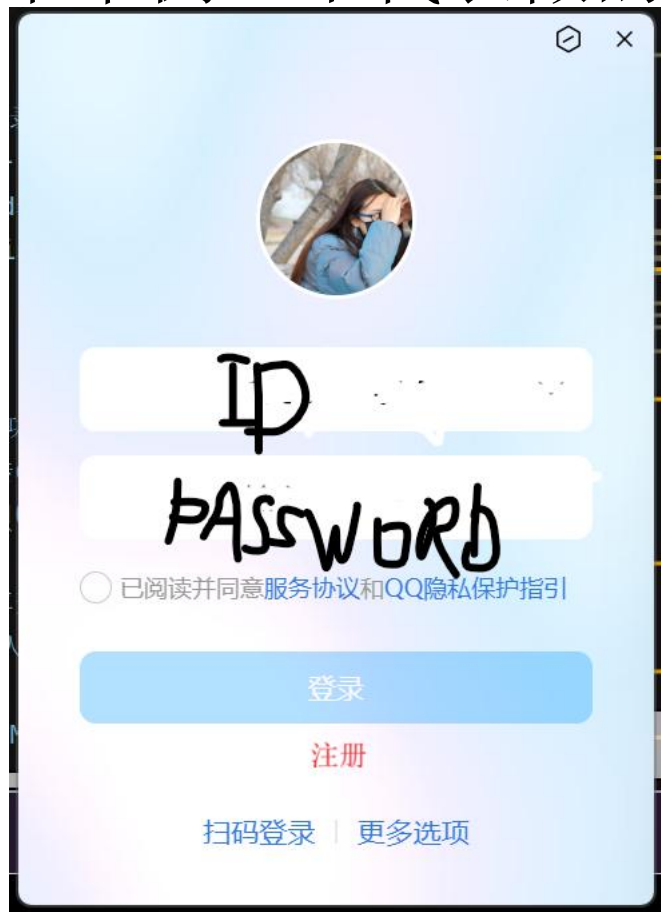
- 目标一：引入tkinter绘制出400x500的登录/注册界面，和登录后800x600的聊天界面。
- 目标二：用ID名test，密码1234，在登录界面，点击登录后能够跳到聊天界面。
- 目标三：将用户的ID和密码加密，ID用凯撒加密法，密码用SHA256 hash。
- 目标四：将不同用户的信息，也就是加密的ID和密码，以及与各自联系人的聊天记录，分别存储到json文件中，json文件在该项目中充当数据库
- 目标五：添加注册按钮和与其对应的功能，并实现用新ID和密码登录。
- 目标六：在聊天界面中增加一个可以添加新联系人的按钮，并实现与其对应的功能。

# 项目环境

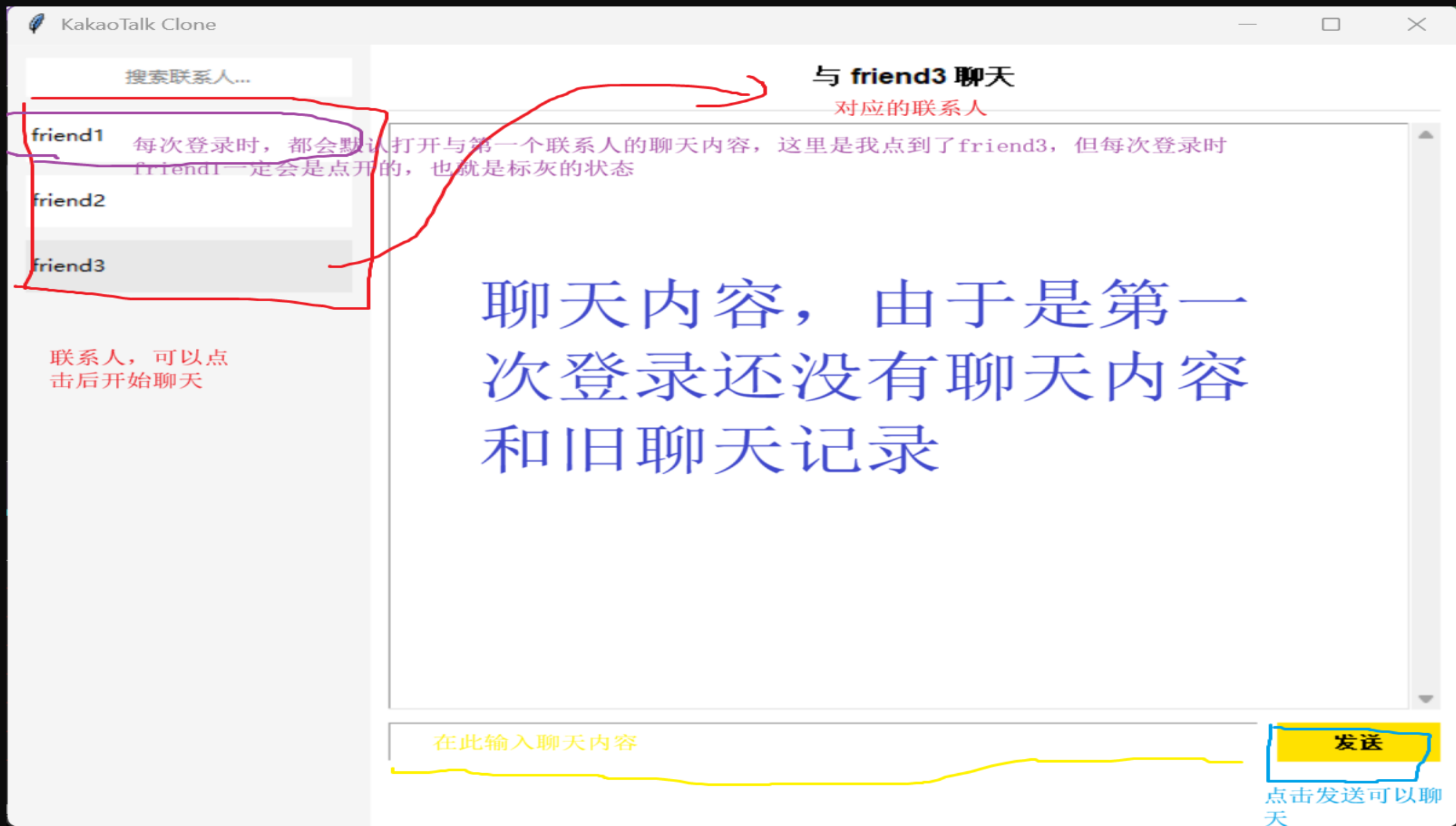
- Windows, Python3, tkinter (用于UI交互), hashlib (用于用户 password 的加密)

# 设计目标

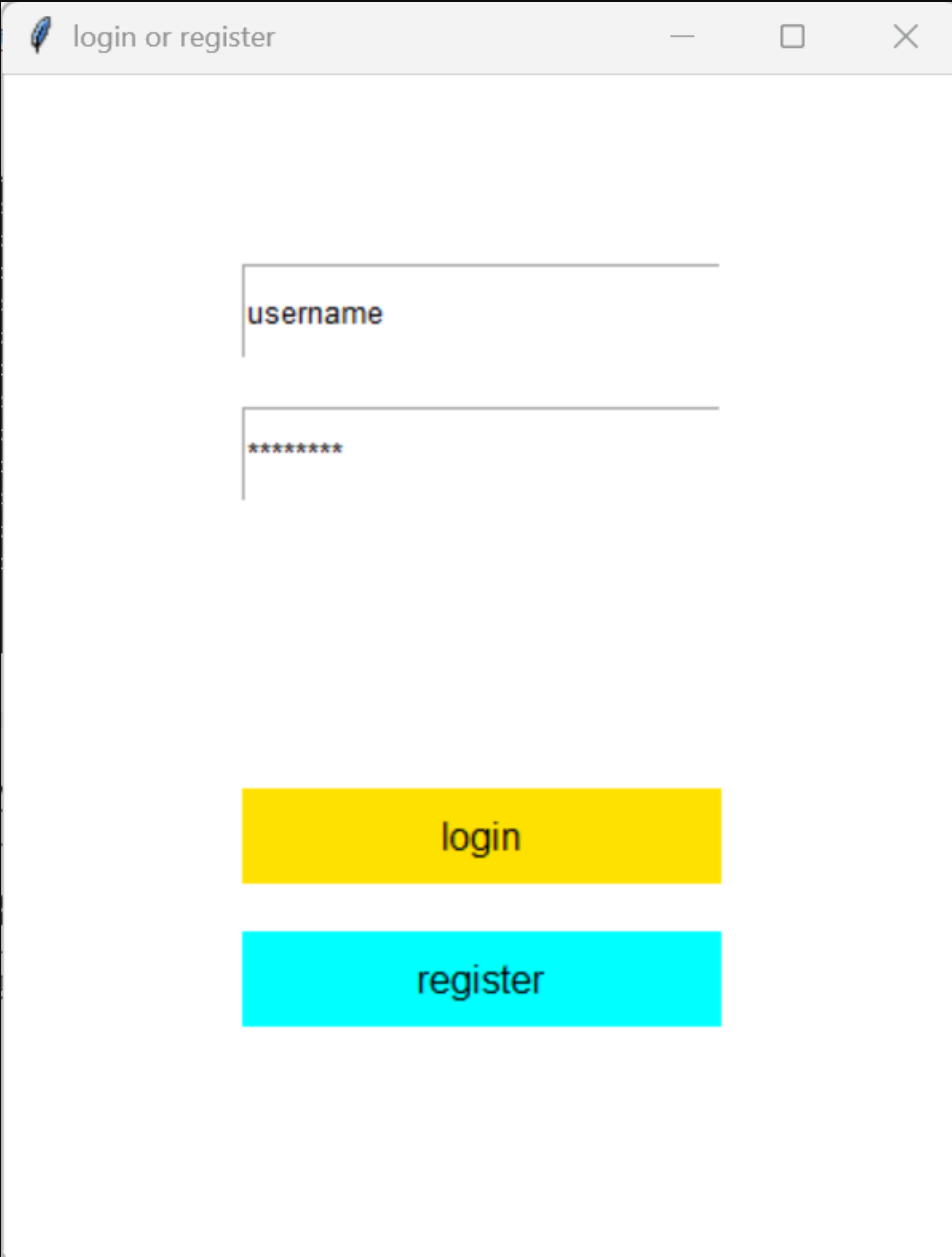
- 逐步实现项目目标的每一步，后续会在登录界面中添加注册按钮和相关功能，类似于下图，输入ID和密码，点击登录后，进入聊天界面。（以QQ简单举例，不代表做的就是QQ的）



# 目标设计：结果预示



# 目标设计：结果预示



The image shows a window titled "login or register" with a feather icon in the top-left corner. The window contains a form with two input fields. The first field is labeled "username" and the second field is masked with eight asterisks "\*\*\*\*\*". Below the input fields are two buttons: a yellow "login" button and a cyan "register" button.

login or register

username

\*\*\*\*\*

login

register

# 目标设计：功能介绍

- 如上两页展示可见，在第7页有一个用于测试的ID和密码，是test: 1234，输入该ID和密码，再点击login，即可跳到第6页的聊天界面。若点击register按钮，login按钮会暂时隐藏，而第二个输入框的正下方位置会出现需要确认密码的输入框，用于保证注册时用户输入了正确的密码。随后再次点击register按钮，表示成功注册新用户，此时第三个输入框会消失，login按钮会重新出现，用新的ID和密码即可实现登录。
- 除了上面这些，新用户成功注册登录后，需要联系人来进行联系，可以在左侧联系人部分的顶部搜索用户的左边添加一个加号，代表点击后通过输入用户ID或手机号等信息来添加新的联系人。



# 登录/login界面

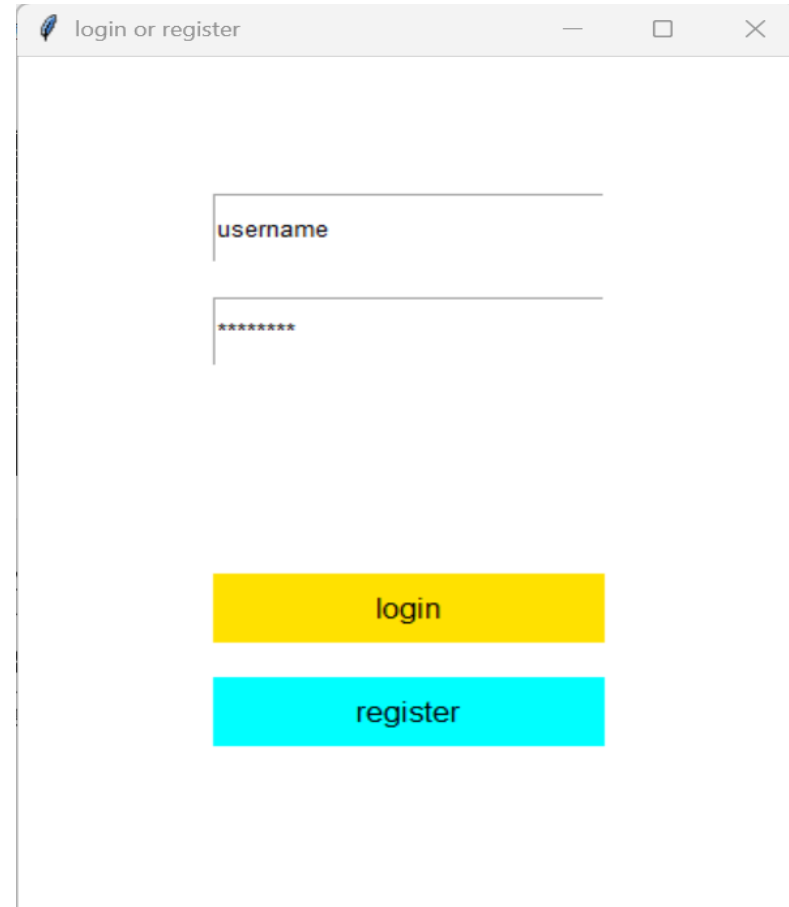
- 引入tkinter和tkinter中的scrolledtext模块，其实这两个模块不止用于登录界面，在下面的聊天界面中也会用到，在介绍登录界面时就一句带过了。创建名为LoginApp的类，在初始化方法中设置login界面里的所有样式，登录界面的大小，三个输入框的位置、大小，两个按钮的大小、位置、颜色等。
- 除了初始化上面的基础信息，还需要设置一个名为最高权限使用者administrator的ID和密码，这是因为在实际开发时，软件公司需要在被恶意攻击或为了在更新软件版本或修复bug时，需要启用最高权限来对代码进行一定的修改、修复和维护，就需要在一开始的初始化方法中设置好。
- 若是第一次登录或注册，都需要将用户的ID和密码存储到名为userInfo的json文件当中，且ID和密码需要进行加密，在这里ID用凯撒加密法，挪三格，密码用SHA256 hash加密法加密。

# 登录/login界面—用户名和密码输入框

- 用户名和密码输入框的大小和样式等，在下一页还有一个隐藏了的确认密码输入框，在下一页会详细说明。右侧为预示图。

```
# 用户名输入框
self.usernameEntry = tk.Entry(root)
self.usernameEntry['font'] = ('Arial', 10)
self.usernameEntry.place(x=100, y=80, width=200, height=40)
self.usernameEntry.insert(0, 'username')

# 密码输入框
self.passwordEntry = tk.Entry(root, show='*')
self.passwordEntry['font'] = ('Arial', 10)
self.passwordEntry.place(x=100, y=140, width=200, height=40)
self.passwordEntry.insert(0, 'password')
```



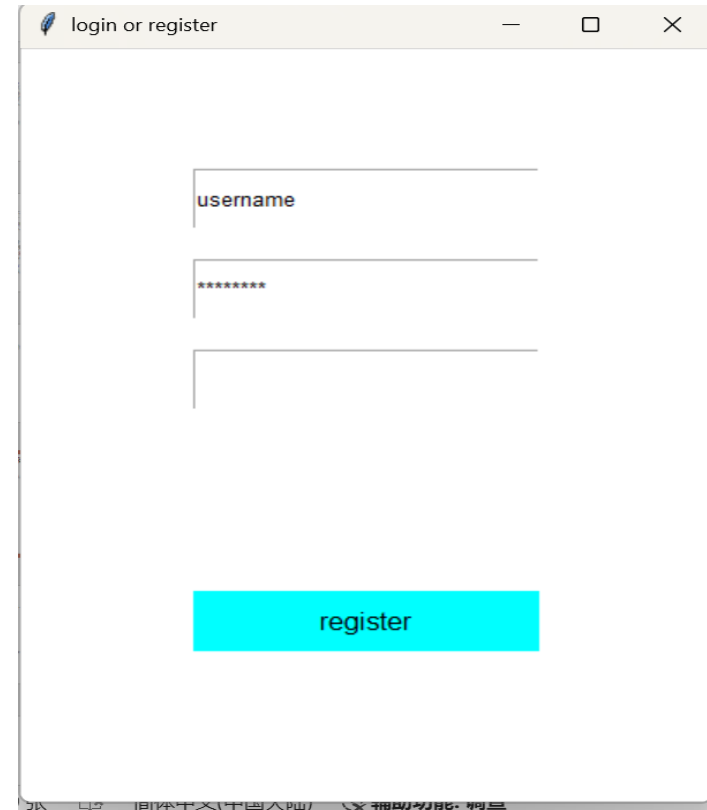
# 登录/login界面—注册和登录按钮及确认密码输入框

- 在上一页的右侧展示了预示图，在这里展示的是注册和登录，两个按钮和确认密码输入框的样式，一开始确认密码输入框是隐藏了的，点击register按钮后，触发open\_register函数，隐藏login按钮，显示确认密码输入框。open\_register函数，主要是点击register按钮后，隐藏login按钮，显示确认密码按钮，最后再将注册按钮和注册模块函数进行绑定初始化。因为注册后，想要用新ID和密码登录时，需要恢复login按钮。

```
# 登录按钮
self.loginBtn = tk.Label(root)
self.loginBtn['text'] = 'login'
self.loginBtn['bg'] = '#FFE100'
self.loginBtn['fg'] = 'black'
self.loginBtn['font'] = ('Arial', 12)
self.loginBtn.place(x=100, y=300, width=200, height=40)
self.loginBtn.bind('<Button-1>', self.login)

# 确认密码的输入框，就是需要二次输入密码的输入框，默认先不展示
self.passwordReEntry = tk.Entry(root, show='*')
self.passwordReEntry['font'] = ('Arial', 10)

# 注册按钮
self.registerBtn = tk.Label(root)
self.registerBtn['text'] = 'register'
self.registerBtn['bg'] = 'aqua'
self.registerBtn['fg'] = 'black'
self.registerBtn['font'] = ('Arial', 12)
self.registerBtn.place(x=100, y=360, width=200, height=40)
self.registerBtn.bind('<Button-1>', self.open_register)
```



# 登录/login界面—打开聊天框函数，SHA

- SHA256 hash，又称256位加密哈希算法，是密码学哈希函数家族SHA-2(Secure Hash Algorithm)中的一员，由美国国家安全局（NSA）设计并由美国国家标准与技术研究院（NIST）在 2001 年发布。它是当今最广泛使用的哈希算法之一，尤其在数据完整性验证、数字签名和密码存储等安全应用中。
- 其实在现实中开发聊天软件时，有的软件公司会使用SHA256 hash加密法来对用户设置的密码进行加密。对用户的密码进行加密，也并不是将设置的密码永远封存，这确实也做不到，有时用户需要修改密码的时候，还需要将原先的密码删除掉，再把新的密码加密重新进行保存，因此加密不是永久性行为，而是一种保护性行为。除了黑客用大量的zombie PC对软件公司进行强制性攻破以外，普通人无法破解密码，这也是该加密法的特点之一，就是一旦设置，就是不许修改，一般情况下，这种情况出现在用户许久没有登陆忘记了密码，此时就需要用户重新设置新的密码。

# 登录/login界面—打开聊天框函数，SHA

- SHA函数家族还有许多其他加密方法，SHA家族也经历了三个阶段，第一阶段为SHA-0和SHA-1，这两个版本已被攻破不再安全。SHA-2家族包含以下常见函数，SHA-224、SHA-256、SHA-512、SHA-284等，其中SHA-256是最常用的版本，目前SHA-2家族是被认为安全的并切实行业的黄金标准。
- SHA-3是当前最新、最先进的安全散列标准，也包含不同输出长度的版本，SHA-3子类中包含SHA3-224、SHA3-256、SHA3-384和SHA3-512. 虽然家族内的函数的应用速度不如SHA-2快，但也正在被越来越多的系统和协议所支持。其中SHAKE128和SHAKE256可以产生任意长度的输出，非常适合用于消息认证码、流密码或作为伪随机数生成器。

# 登录/login界面—打开聊天框函数，SHA

- 需要事先引入hashlib包，hashlib支持SHA256hash算法来对数据进行加密。用encrypt来命名加密密码的函数，该函数会用于登录(login函数)，注册(register函数)中，在register函数中会用到两次，因为在确认密码的时候还需要调用一次该函数，如图中所示，在需要该函数的时候，调用并将需要加密的密码作为参数传递进该函数即可。创建的几个用户可以在userinfo.json文件中查看加密后的ID和密码。

```
# 登录是密码加密，方法2
```

```
def encrypt(self, data): 4 usages  👤 goodMorning_pro@xxx.com <tianyuan990605@gmail.com>
    input_string=data
    encoded_string=input_string.encode('utf-8')
    sha256_hash=hashlib.sha256(encoded_string)
    hex_digest=sha256_hash.hexdigest()
    return hex_digest
```

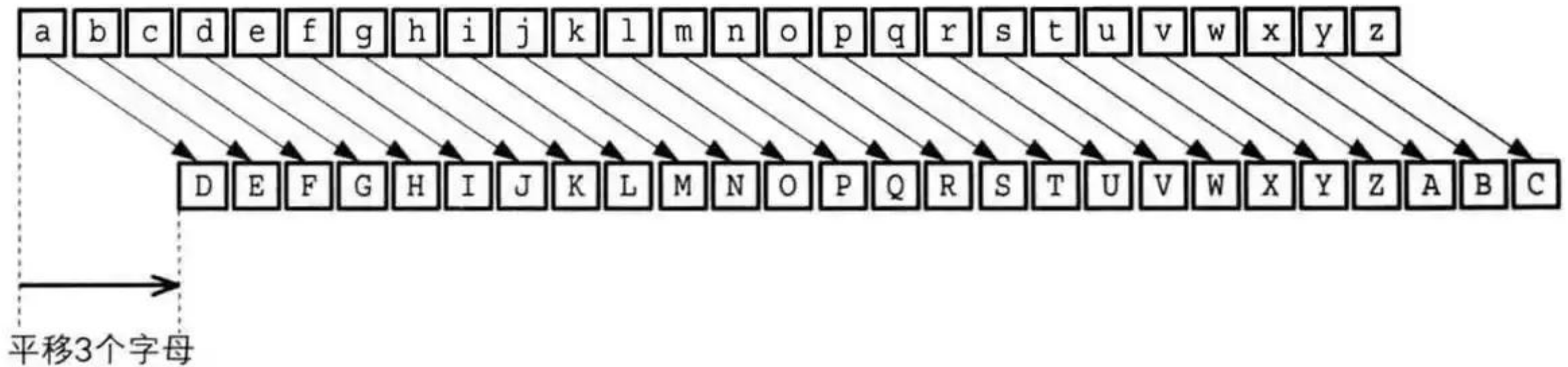
```
# 登录函数
```

```
def login(self, event=None): 1 usage  👤 goodMorning_pro@xxx.com <tianyuan990605@gmail.com>
    username = self.usernameEntry.get()
    password = self.passwordEntry.get()

    print(self.users)    #{'whvw': '03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4'}
    # 加密的ID
    encryptName=self.encryptID(username)
    print('encryptName-'+encryptName)    #encryptName-whvw
    encrypt=self.encrypt(password)
    print('encrypt-'+encrypt)    #encrypt-03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4
```

# 登录/login界面—打开聊天框函数，凯撒加密法

- 凯撒加密法，是一种替换加密的技术，明文中的所有字母都在字母表上向后（或向前）按照一个固定数目进行偏移后被替换成密文。例如，当偏移量是3的时候，所有的字母A将被替换成D，B变成E，以此类推。当然，偏移量并不是固定的，开发者可以根据个人习惯，偏移5个也是可以的，但在该项目和测试用中，还是使用了3个偏移量，下面是偏移3个量的简单介绍。



# 登录/login界面—打开聊天框函数，凯撒加密法

- 下面为该项目中使用凯撒加密法，并偏移3个量的代码。

```
# ID加密，用凯撒加密法，挪三格
def encryptID(self, data): 5 usages  👤 goodMorning_pro@xxx.com <tianyuan99060
    result=''
    for word in data:
        if 'a' <= word <= 'z':
            # 将字符转换为0-25的数字，加上偏移量3，然后取模26确保在字母范围内
            w = (ord(word) - ord('a') + 3) % 26
            encrypted_word = chr(w + ord('a'))
            result += encrypted_word
        else:
            # 非字母字符保持不变
            result += word
    return result
```



# 登录/login界面—打开聊天框函数，凯撒加密法

- 凯撒加密法测试用代码，如图所示，偏移了3个量，在控制台中输入用户的ID，可以得到和ID毫无关系的一串字符串，这就是加密后的ID。

```
# 把这段输出的结果截图插入到ppt中
```

```
words=input()
```

```
result=''
```

```
for word in words:
```

```
    if 'a'<=word<='z':
```

```
        w = (ord(word) - ord('a') + 3) % 26
```

```
        encryptWord=chr(w+ord('a'))
```

```
        result+=encryptWord
```

```
print(result)
```

```
C:\Users\田
```

```
admin
```

```
dgp1q
```

```
C:\Users\田
```

```
user
```

```
xvhu
```

```
C:\Users\田\App
```

```
catherine
```

```
fdwkhu1qh
```

# 登录/login界面—加密ID和密码并存储

- 设置两个用户ID和密码，创建userInfo.json文件，用于存储加密后的用户信息，在创建新账户时，该用户的信息也会被加密后存储到该文件中。

```
# 最高权限使用者
self.administrator = {'priorityRoot': '123456'}

# 加载用户数据
try:
    # 若第一次登录，则创建一个文件来存储用户数据
    with open('userInfo.json', 'r') as file:
        # with open(f"{self.encryptID('test')}_friends.json", 'r') as file:
            self.users = json.load(file)
except FileNotFoundError:
    # 若不是第一次登录，则用户的信息会存在，则直接打开已有文件，并继续向里存储后续数据
    with open('userInfo.json', 'w') as f:
        # with open(f"{self.encryptID('test')}_friends.json", 'w') as f:
            # self.users = {self.encryptID('test'):self.encrypt('1234')}
            # self.users={self.encryptID('makit'):self.encrypt('7890')}
            # 初始化users
            self.users={}
            # 向users中分别添加test和makit用户名及密码
            self.users[self.encryptID('test')]= self.encrypt('1234')
            self.users[self.encryptID('makit')]= self.encrypt('7890')
            json.dump(self.users, f)
```

```
main.js.py  usersInfo.json  test.py  readme.md
{"whvw": "03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4",
 "pdnlw": "6a95bbab63d587b596398c4bd7e91a132f24032d2007d107e5ea71967724b092",
 "xvhu1": "38083c7ee9121e17401883566a148aa5c2e2d55dc53bc4a94a026517dbff3c6b"}
```

# 登录/login界面—登录函数

- 登录函数当中，用get函数获取输入的用户ID和密码，可以通过encryptID函数和encrypt函数来获取当前登录时使用的ID和密码，再需要判断通过调用函数获得的ID和密码，是否与self.users的一致，是则予以登录，调用open\_chatApp函数，否则显示登录失败的提示。

```
def login(self, event=None): 1 usage  👤 goodMorning_pro@xxx.com <tianyuan990605@gmail.com>
    username = self.usernameEntry.get()
    password = self.passwordEntry.get()

    print(self.users)    #{'whvw': '03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7
    # 加密的ID
    encryptName=self.encryptID(username)
    print('encrypyName-'+encryptName)    #encrypyName-whvw
    encrypt=self.encrypt(password)
    print('encrypt-'+encrypt)    #encrypt-03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13

    # 若加密的id和self.users的id相同并且，加密的密码也进行比较，若相同，则予以登录
    # if encryptName in self.users and self.encrypt('1234')== '03ac674216f3e15c761ee1a5e255f06
    if encryptName in self.users and self.users[encryptName]==encrypt:
        self.open_chatApp(username)
    else:
        errorLabel = tk.Label(self.root)
        errorLabel['text'] = 'failed to login'
        errorLabel['bg'] = 'white'
        errorLabel['fg'] = 'red'
        errorLabel.place(x=150, y=250, width=100, height=20)
        self.root.after(2000, errorLabel.destroy)
```

failed to login

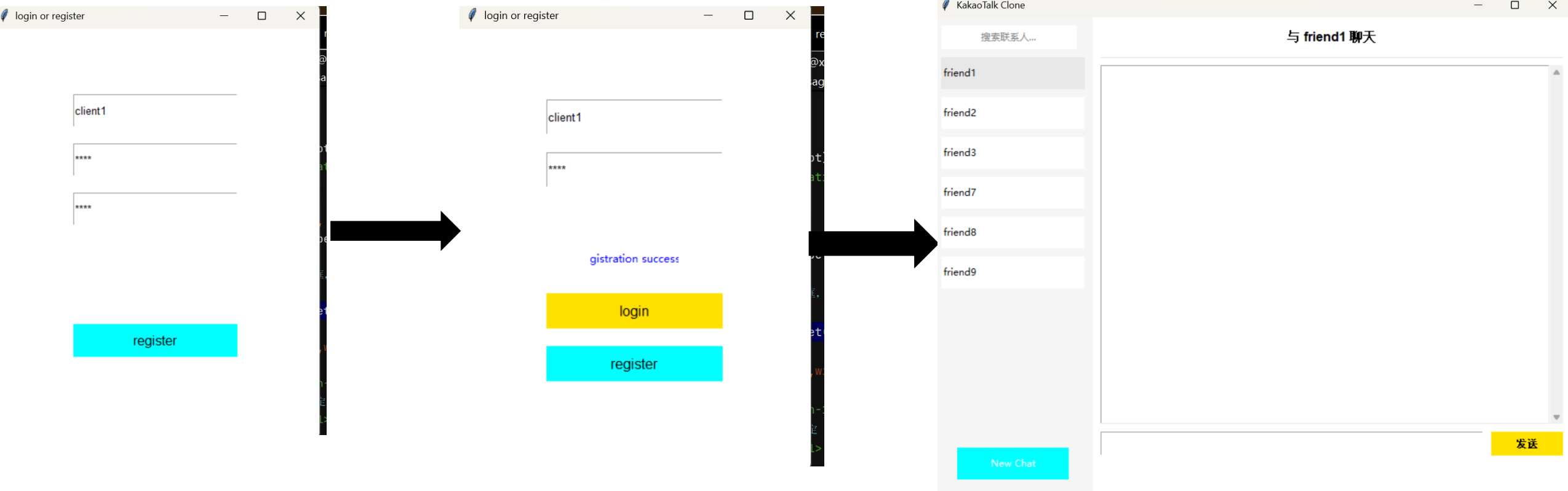
login

register

# 登录/login界面—注册函数

- 在注册函数中，与登录函数部分有区别的就是，会多出来一个确认密码的输入框。在register函数中，设置创建新用户时的条件，ID名长度不得小于1，已存在的账户不得重复注册，新注册时的密码长度不得小于4，新密码与确认密码必须保持一致，才予以注册，否则提示注册失败。满足以上条件时，将新用户的信息存储到userInfo.json文件中，并展示注册成功的提示。最后作为收尾，注册成功后，就需要隐藏确认密码输入框，恢复初始的登录页面，需要注册按钮和Button-1解绑后，又与open\_register按钮重新绑定。如下一页所示，用合法的ID和密码创建新用户，点击register按钮后，成功的话会出现‘Registration successful’的提示后，隐藏确认密码输入框，恢复login按钮，此时点击login按钮就可以登录，并跳转到该用户的登录页。

# 登录/login界面—注册函数



# 聊天界面

- ChatApp类中，除了和LoginApp的初始化方法中一样，接收root做参数以外，还要接收username作为参数，因为登录之后需要根据不同用户的信息，进行分门别类的数据保存和数据显示。除了聊天界面的大致框架外，从左边开始，依次联系人列表的框架、搜索框、聊天区域框架、聊天标题、聊天消息区域、输入框、信息样式、发送按钮和选中后信息人的颜色变化，以及不同用户与其联系人的聊天记录会存储在何处，这些都需要进行初始化设置。
- LoginApp中，是通过try...except...来存储用户的数据到名为userInfo的文件中，在ChatApp中同样也需要用到try...except...，这次是用来存储每个用户与其联系人的聊天记录的名称为{username}\_chatHistory的文件中。还需要遍历联系人，将遍历的结果与select\_contact函数绑定，这样可以点击联系人聊天。

# 聊天界面—联系人展示(测试contacts)



# 聊天界面—联系人展示(真实用contacts)

- 展示的是test的联系人，里面的friends1、2、3、4是提前创建{username}\_chatHistory.json时就写在文件里的，代表username自己的联系人。下面只展示test的联系人吧，makit的就不展示了





# 聊天界面—加载聊天记录和保存聊天记录函数

- 调用加载聊天记录函数，通过给self.chat\_history\_file赋值聊天记录来获取当前用户与其联系人的聊天记录。调用save\_chat\_history函数来将新增的聊天记录存储到用户与其联系人独有的存储聊天记录的文件中。

```
# 加载聊天记录
```

```
def load_chat_history(self): 1 usage  👤 goodMorning_pro
    try:
        with open(self.chat_history_file, 'r') as fi
            self.chat_history = json.load(file)
    except FileNotFoundError:
        self.chat_history = {}
```

```
# 保存聊天记录
```

```
def save_chat_history(self):  👤 goodMorning_pro@xxx.com
    with open(self.chat_history_file, 'w') as f:
        json.dump(self.chat_history, f, indent=4)
```

# 聊天界面—选择聊天对象函数

- 与选择的联系人第一次聊天的话，无从获取聊天记录，直接显示空白，但若不是第一次聊天，则将过往的聊天记录调取出来添加到聊天区域中。右图为makit与其联系人9月6号时的聊天记录。

```
# 点击选择聊天对象
def select_contact(self, contact): 3 usages  👤 goodMorning_pro@xxx.com <
    self.current_contact = contact
    self.chatTitle['text'] = f'与 {contact} 聊天'
    # 重置所有联系人的背景色
    for btn in self.contact_buttons:
        btn['bg'] = 'white'
    # 设置当前选中联系人的背景色
    for btn in self.contact_buttons:
        if btn['text'] == contact:
            btn['bg'] = '#E8E8E8'
            break
    # 显示聊天记录
    self.chat_area.config(state='normal')
    self.chat_area.delete(index1:1.0, tk.END)
    # 选中的联系人在之前有过聊天记录
    if contact in self.chat_history:
        # 获取到与该联系人的聊天记录
        for message in self.chat_history[contact]:
            # 向聊天区域添加新聊天记录中
            self.chat_area.insert(tk.END, message)
    self.chat_area.config(state='disabled')
    self.chat_area.yview(tk.END)
```



# 聊天界面—发送信息函数

- 将send\_message函数和发送按钮以及信息输入框绑定，而在send\_message中，首先保证选中了某一个联系人，并保证在信息输入框有新信息。随后获取当前时间，用于在聊天框中展示聊天时间和在聊天记录文件中存储信息。在每次点击发送按钮后，需要清空输入框，以便于下次或者想起他联系人发送信息。并且与联系人的聊天区域和存储聊天记录的文件也需要更新，在聊天区域需要将新的聊天内容展示出来，在存储聊天记录的文件中，需要将新的聊天内容和时间存储进去。最后因为该项目中没有真人版的联系人，发送信息后，得到的回复都是模拟回复列表当中随机回复的内容，因此发送的信息和收到的回复大概率驴唇不对马嘴，但在该项目中不会实现与真人的交互。
- 该函数的重点部分是，与联系人的新聊天内容该如何聊天框展示和存储到json文件当中的问题，在最后的模拟回复会创建另一个函数来解决模拟回复交互的问题。

# 聊天界面—发送信息函数

```
# 若与选中的联系人无聊天记录
if self.current_contact not in self.chat_history:
    # 则向聊天记录中添加选中的联系人
    self.chat_history[self.current_contact] = []
# 将与选中的联系人的聊天记录添加到chat_history中
self.chat_history[self.current_contact].append(formatted_message)

# 更新聊天区域
self.chat_area.config(state='normal')
self.chat_area.insert(tk.END, formatted_message)
self.chat_area.config(state='disabled')
self.chat_area.yview(tk.END)

# 清空输入框
self.message_entry.delete(0, tk.END)

# 将聊天记录存储到chatHistory.json文件中
with open(f"{self.username}_chatHistory.json", 'w') as f:
    json.dump(self.chat_history, f, indent=4)

# 模拟回复
self.root.after(1000, self.simulate_reply)
```

```
makit_chatHistory.json × usersInfo.json main.js.py test.py M↓ re
1 {
2     "friend5": [],
3     "friend6": [
4         "[13:30] \u6211: \u4f60\u597d\n",
5         "[13:30] friend6: \u5f88\u6709\u8da3\u7684\u60f3\u6cd5\n"
6     ],
7     "friend7": []
8 }
```

聊天内容也会加密

# 聊天界面—模拟回复信息函数

- 联系人回复信息的前提是‘我’选中了联系人后，向联系人发送了信息，像真实的聊天软件中会在聊天区域展示聊天时间，和send\_message函数一样，需要获取到当前时间，并列出会回复的消息列表，在‘我’给一位联系人，发送信息后，联系人那边会从列表中随机选中一条信息来回复‘我’，最后将这些内容添加到聊天区域，存储到聊天记录文件中。
- 该函数的重点部分依旧是如何将回复内容展示在聊天区域和存储到文件中，最后存储文件的部分没能截图，但可以参考send\_message函数的最后那部分代码，和它是完全一致的。且驴唇不对马嘴的回复依然可以参考makit与其联系人的聊天内容。

# 聊天界面—模拟回复信息函数

```
def simulate_reply(self): 1 usage 1 goodMorning_pro@xxx.com <tianyuan990605@gmail.com>
# 模拟回复信息
replies = [
    "好的, 我明白了",
    "很有趣的想法",
    "我需要考虑一下",
    "明天再聊吧",
    "谢谢你的消息"
]

# 随机发送上面replies的内容
import random
reply = random.choice(replies)
formatted_reply = f"[{time_str}] {self.current_contact}: {reply}\n"

# 添加到聊天记录
self.chat_history[self.current_contact].append(formatted_reply)

# 更新聊天区域
self.chat_area.config(state='normal')
self.chat_area.insert(tk.END, formatted_reply)
self.chat_area.config(state='disabled')
self.chat_area.yview(tk.END)
```



# 聊天界面-好友添加和管理

- 除了现已有联系人，增添一个添加好友的按钮，点击该按钮可以在左侧列表中添加新联系人，还可以与新添加的联系人进行聊天交互。联系人框的样式设置可以从上面拷过来，按照要求修改一下按钮的内容和颜色样式，并将其与addContacts函数进行绑定。在该函数中需要注意的就是，保证添加了新联系人后关闭窗口，在下一次用同一ID登录时，依旧有新添加的联系人以及与其的聊天记录。这是kakaotalk的特点，同样是用手机号进行的绑定，国内的主要聊天软件，如微信和QQ，在添加好友时，需要发送添加好友的请求，再对方验证并同意后才可以开始聊天，这是为了保护每个用户的安全，等完成kakaotalk特点的添加好友的操作后，尝试需要发送验证信息，等待对方同意后的操作。

# 聊天界面-好友添加和管理(不需要验证)

- 如下图所示, 这部分的重点除了将用户与新联系人的聊天内容存储到文件中以外, 还要保证下一次用同一ID登录时, 与新联系人的聊天内容依旧会展示在聊天区域, 这样就代表数据也被存储在了存储聊天记录的文件中。

```
class ChatApp:
    def __init__(self, username, password):
        self.username = username
        self.password = password
        self.contacts = []
        self.chat_history = {}

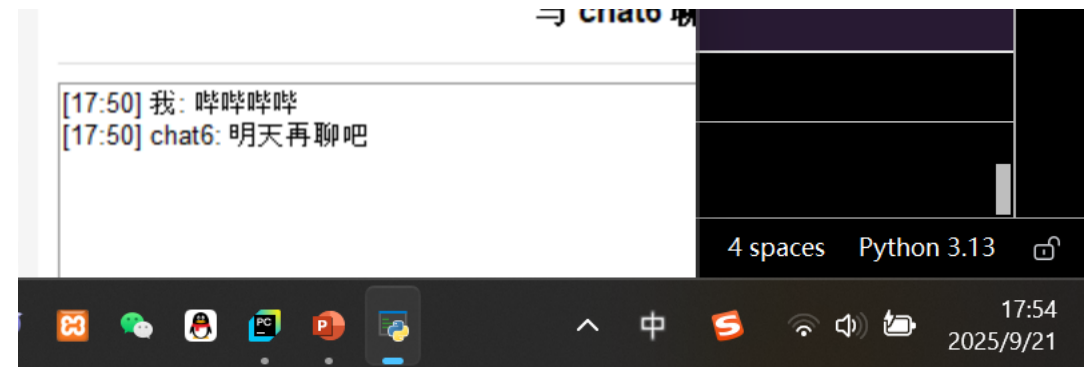
    def addContacts(self, event=None):
        # self.contacts.append(newContact)

        # 设置联系人框的样式
        btn = tk.Label(self.contacts_frame)
        btn['text'] = 'chat'+str(len(self.contact_buttons)+1)
        btn['bg'] = 'white'
        btn['fg'] = 'black'
        btn['anchor'] = 'w'
        btn.place(x=10, y=50 + len(self.contact_buttons) * 50, width=180, height=40)

        # 将选中联系人的函数self.select_contact()和联系人按钮绑定在一起
        btn.bind('<Button-1>', lambda e, c='chat'+str(len(self.contact_buttons)+1): self.select_contact(c))
        self.contact_buttons.append(btn)

        # 用同一ID, 保证每次登录时, 上一次登录的记录还在, 比如聊天记录和聊天对象
        self.chat_history[btn['text']] = []

        with open(f'{self.username}_chatHistory.json', 'w') as f:
            json.dump(self.chat_history, f, indent=4)
```





# 聊天界面-好友添加和管理(需要验证)

- 这一页展示的是添加好友时需要验证和对方同意版本的addContacts函数，是有别于韩国的kakaotalk不需要发送好友申请，在手机通讯录有了电话号，在kakaotalk软件中就会默认已添加联系人，可以无需验证，通过通讯录中的电话号码就可以开始和指定联系人聊天。虽然这样并没有什么大碍，但国内最常用和流行的软件，想要添加联系人的话，基本都需要向联系人发送添加请求，并在对方同意后，才可以开始聊天。个人认为，这样更有利于保护个人隐私，在不法分子通过得到的电话号码或个人二维码后，就可以随意对用户进行骚扰或对用户的安全造成不利。
- 在这部分省去了向对方发送请求和对方同意验证的步骤，只添加了点击New Chat按钮后，弹出一个对话框，在输入框输入联系人的ID后点击添加，即可在联系人列表中显示新的联系人，同时在用户的chatHistory文件中也会新增刚加的联系人。点击新增的联系人，也可以正常聊天。

# 聊天界面-好友添加和管理(需要验证)

- 因为是需要验证因此，需要在初始化函数中增加获取联系人的加载方式，通过调用loadContacts函数来实现，先添加联系人的加载方式，再通过调用load\_chatHistory函数来加载聊天记录。
- 事前，在同一目录下创建名为contacts\_database.json的文件，这个loadContacts函数中，首先声明一个变量来接收该文件，并用打开该文件，从文件中获取每个用户的联系人数据。但若没有，则写入默认联系人来打开。

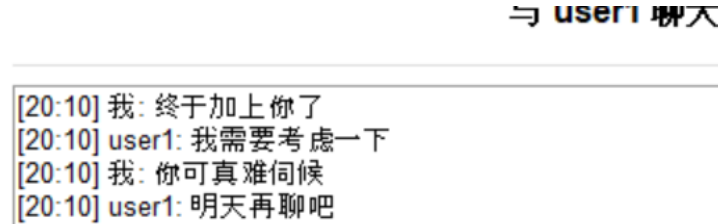
```
class ChatApp: 1 usage  👤 goodMorning_pro@xxx.com <tiar  🚨 14 🚨 39 🟢 4 ^ v
# 从文件中获取联系人列表
def loadContacts(self,event=None): 1 usage  👤 goodMorning_pro@xxx.com <t
# 从contacts_database.json中获取用户各自的联系人列表
contactsFile="contacts_database.json"
# 打开后将文件中的数据赋值给变量all_contacts
try:
    with open(contactsFile, 'r', encoding='utf-8') as f:
        all_contacts = json.load(f)
        return all_contacts.get(self.username, [])
except FileNotFoundError:
    # 如果文件不存在，使用默认联系人
    default_contacts = {
        'test': ["friend1", "friend2", "friend3", "friend4"],
        'makit': ["friend5", "friend6", "friend7"]
    }
    return default_contacts.get(self.username, [])
```

```
1 {
2     "makit": [
3         "friend5",
4         "friend6",
5         "friend7",
6         "admin1"
7     ],
8     "test": [
9         "friend1",
10        "friend2",
11        "friend3",
12        "friend4",
13        "user1"
14    ]
15 }
```

# 聊天界面-好友添加和管理(需要验证)

- 随后需要创建名为addConact，在函数中调用addContactDialog函数，当点击New Chat按钮时，这个对话框就会弹出。同时在addContact中需要检查输入的用户ID不是用户本人，也要保证添加的用户不是已加的联系人。通过调用append方法来将新的联系人添加到contacts中，并向chat\_history中添加新联系人的键值对。随后保存联系人并更新联系人界面。

```
def addContact(self, new_contact):  
    self.addContactDialog()  
    # 检查是否已添加  
    if new_contact in self.contacts:  
        self.show_message('该联系人已存在')  
        return False  
  
    # 检查是否为自己  
    if new_contact == self.username:  
        self.show_message('不能添加自己为联系人')  
        return False  
  
    # 添加联系人  
    self.contacts.append(new_contact)  
    # 在聊天记录中也添加这个联系人的空记录  
    self.chat_history[new_contact] = []  
  
    # 保存联系人并更新联系人界面  
    self.saveContacts()  
    self.updateContactList()  
  
    self.show_message(f"成功添加联系人: {new_contact}")  
    return True
```



# 聊天界面-好友添加和管理(需要验证)

- 接上一页代码和说明，除了在{username}\_chatHistory.json文件中更新联系人信息以及保存用户与联系人聊天的内容，并在saveContacts函数中会创建contacts\_database文件，用于存储用户与其各自联系人，包括默认联系人和新添加的联系人。

```
def saveContacts(self, event=None):
    contactsFile = 'contacts_database.json'

    try:
        with open(contactsFile, 'r', encoding='utf-8') as f:
            allContacts = json.load(f)
    except FileNotFoundError:
        allContacts = {}

    # 更新当前用户的联系人
    allContacts[self.username] = self.contacts

    # 保存回文件
    with open(contactsFile, 'w', encoding='utf-8') as f:
        json.dump(allContacts, f, indent=4, ensure_ascii=False)
```



The screenshot shows a web application interface. On the left, there is a vertical list of contact names: friend1, friend2, friend3, friend4, user1, and user2. The 'user2' entry is highlighted. On the right, a modal dialog box titled 'add contact' is open. It contains a text input field with the value 'user2' and a yellow button labeled '添加' (Add).

[illegible]

# 总结

- 分为两大部分进行总结，登录界面和聊天界面。
- 在登录界面，需要着重注意的就是用户ID和密码的加密方式，目前密码的加密方式使用的最普遍的就是SHA256 hash加密方式，它是SHA家族第二代的加密方式之一，也是使用最广泛的方式之一。目前SHA家族已更新到3.0版本，加密方式也在不断地更新迭代中，该项目中使用256加密的密码，一般不可能随意破解，也就是一旦加密，就无法恢复成原样，这也是他们家族的特点。而ID的加密用的是凯撒加密法，凯撒加密法作为世界上最古老的加密方式之一，也是最广为人知的，除了它也有其他加密方式，ID就可以用。
- 凯撒加密法可以通过算法和规律，对加密的数据进行恢复，但用SHA加密的数据，除非暴力破解或用大量zombie PC进行强行破解外无解法，所以SHA的加密方式确实是最方便，也最安全的加密方式之一。
- 在聊天界面，重点部分为用户数据的存储。在添加新联系人时，要保证原先联系人的情况下，安全地存储和在联系人列表中展示出来。

# 难点与重点

- 难点：
  - 用SHA256 hash加密数据；用凯撒加密法加密用户数据时，需要对加密的数据进行复原，以便于核查是否按照正确的偏移量进行加密。
  - 点击New Chat按钮添加新联系人时，会弹出新对话框，输入联系人ID后点击添加按钮即可，如何做到点击添加按钮后不会反复弹出新对话框，目前尚在解决中。
- 重点：
  - 了解和掌握如何使用SHA256 hash和凯撒加密法对数据进行加密。
  - 在注册和登录时，在对话框中，根据点击的按钮(login或register)的不同，输入框和按钮不同的展示。
  - 存储用户的个人数据，包括用户与联系人的聊天内容，新添加的联系人、与其的数据。
  - 如何妥善存储和处理新添加的联系人，例如，新添加后，在联系人列表中是否会出现只展示新联系人，而丢失先前联系人的情况。

# 感受

- 需要能熟练使用用户UI交互的tkinter。
- 了解了对数据加密的方法和途径，例如SHA家族已将加密方法更新到第三代，第二代的256位虽然还是目前使用最广泛，也最安全的加密方式之一，但还需要继续学习和了解更新迭代中的方法。

# 关于PyQt

- PyQt一组用于创建图形用户界面应用程序的Python绑定库，它成功地将 Qt 库(一个功能极其强大的 C++ 跨平台应用开发框架)与 Python 语言(一种简洁、易学且高效的编程语言)融合在一起。PyQt就是可以让程序员可以用 Python语言来调用Qt框架的功能，从而快速、轻松地开发出专业的桌面应用程序。
- 在该项目中我使用的是名为TkinterUI交互工具，查阅资料后得知，Tkinter的外观比较老旧，功能相对PyQt较少，但优点就是无需安装。

```
C:\Users\田园\AppData\Local\Programs\Python\Python313>pip install PyQt6
Collecting PyQt6
  Downloading pyqt6-6.9.1-cp39-abi3-win_amd64.whl.metadata (2.2 kB)
Collecting PyQt6-sip<14,>=13.8 (from PyQt6)
  Downloading pyqt6_sip-13.10.2-cp313-cp313-win_amd64.whl.metadata (515 bytes)
Collecting PyQt6-Qt6<6.10.0,>=6.9.0 (from PyQt6)
  Downloading pyqt6_qt6-6.9.2-py3-none-win_amd64.whl.metadata (551 bytes)
Downloading pyqt6-6.9.1-cp39-abi3-win_amd64.whl (25.7 MB)
  25.7/25.7 MB 15.2 MB/s eta 0:00:00
Downloading pyqt6_qt6-6.9.2-py3-none-win_amd64.whl (74.1 MB)
  74.1/74.1 MB 14.3 MB/s eta 0:00:00
Downloading pyqt6_sip-13.10.2-cp313-cp313-win_amd64.whl (53 kB)
Installing collected packages: PyQt6-Qt6, PyQt6-sip, PyQt6
Successfully installed PyQt6-6.9.1 PyQt6-Qt6-6.9.2 PyQt6-sip-13.10.2

[notice] A new release of pip is available: 25.0.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```



# 关于PyQt

- 如上一页图片所示，安装PyQt后就可以使用了，下面展示一个简单案例。通过在引入时，引入各种各样的GUI组件，并进行实例化后，灵活运用起来，GUI组件里有QPushButton、Qlabel、QVBoxLayout等。

