User Manual

Web Application:

Run the command

python3 hello.py

```
(venv) kathu@kathu-Aspire-E5-575G:~/Desktop/project209$
(venv) kathu@kathu-Aspire-E5-575G:~/Desktop/project209$ python3 hello.py
 * Serving Flask app "hello" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 303-945-086
```

Centralized Authority

Run the command

python3 KDC.py

```
kathu@kathu-Aspire-E5-575G: ~/Desktop/project209
kathu@kathu-Aspire-E5-575G:~/Desktop/project209$ python3 KDC.py
KDC listening....
```

Database

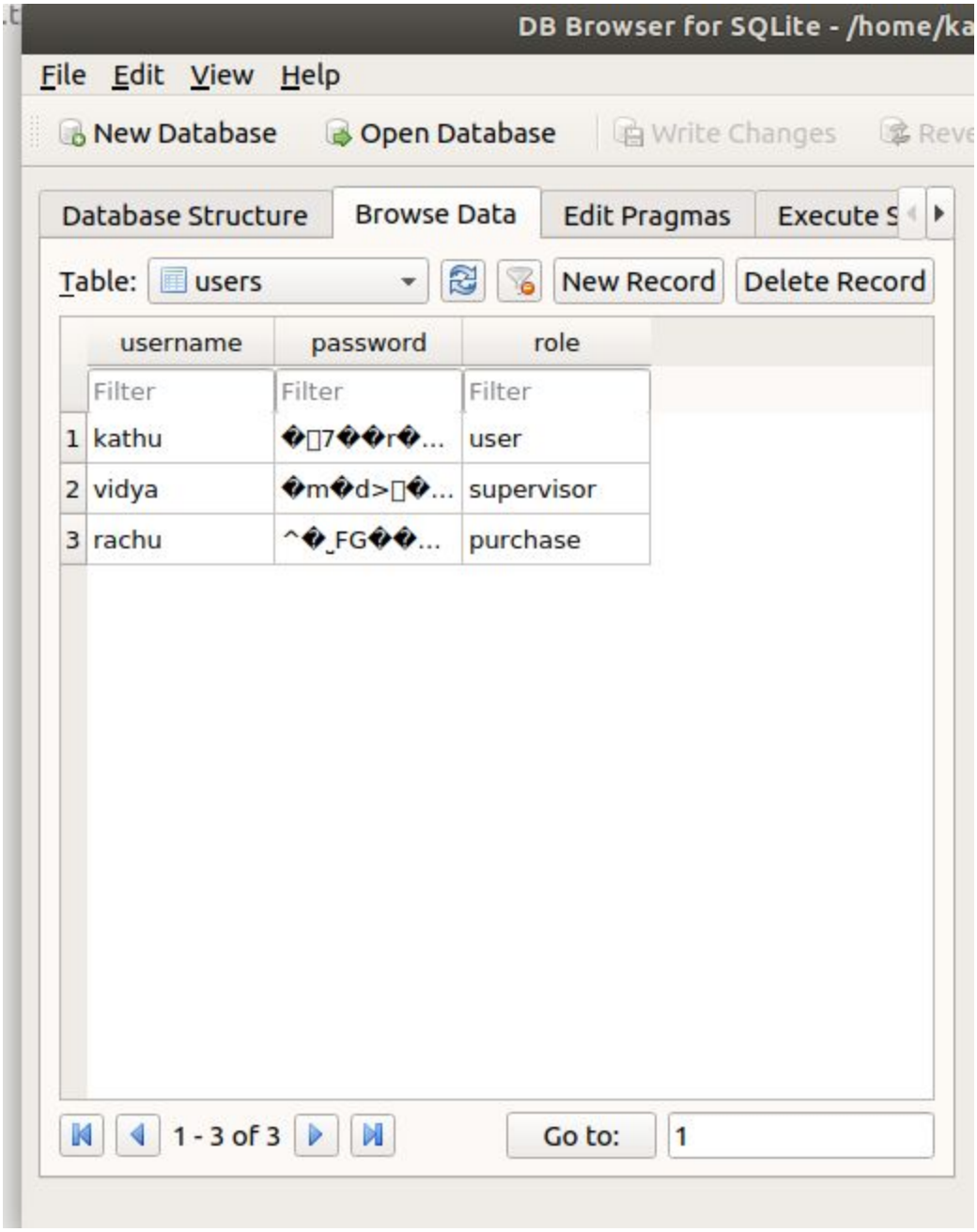
The sqilite3 browsers database.db and db1.db should be in same folder as hello.py and KDC.py

Role based login

The system has a role based log in. So the web pages appearing for each role will be different. The roles of each used can be verified using “users” table in database.db

Password of each entity

username: kathu	password: kathu123
username: vidya	password: vidya123
username: rachu	password: rachu123



Tables in database.db

Table name: users
Provides username, hash of password and role

Table name: orders
Provides the order details and its status

Table name: supervisor
Stores the order number, encrypted order details send by user and signature of user

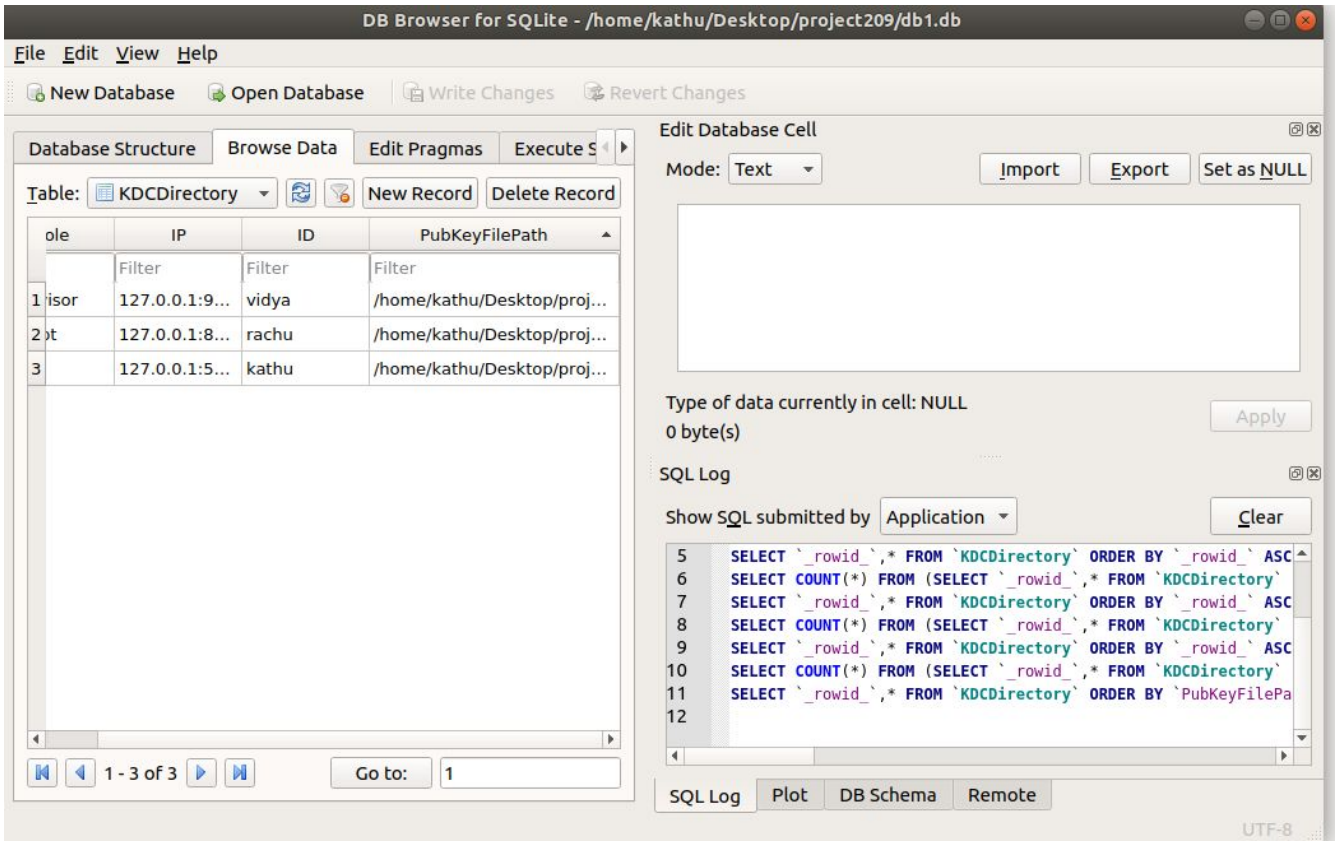
Table name: purchasedepartment
Stores the order numbers encrypted order details send by user and supervisor, and also their respective signatures

Tables in db1.db

Table name: purchasedepartment
Stores the role, user id , ip and public key path

Public Key and Private Key Storage

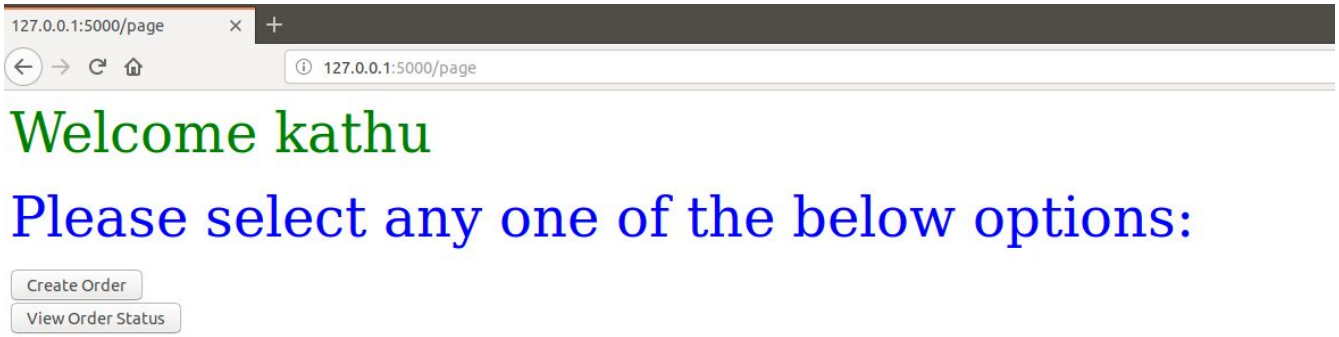
The public and private keys are stored as .pem files in the system. The Centralized Authority stores the file path of the keys in database db1.db. So while running the web application please change the public key path to the appropriate path inside the system in the table KDCDirectory



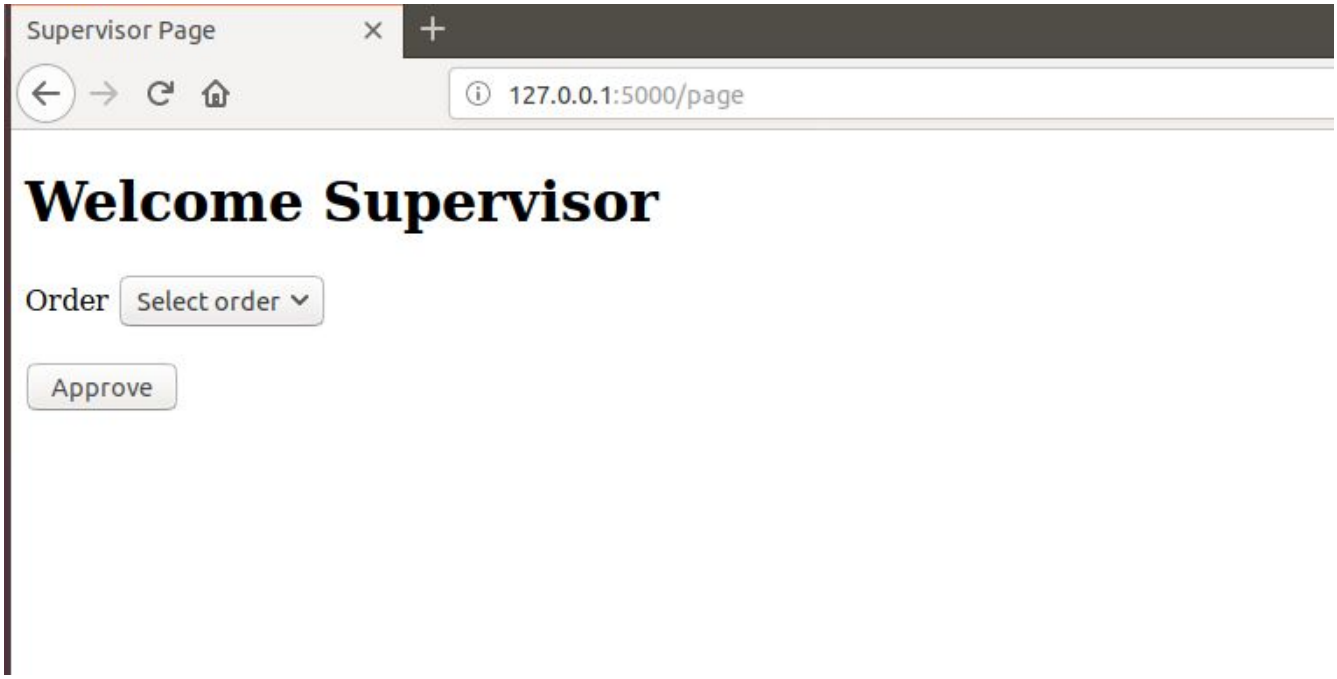
HTML Page storage
All html pages are stored in templates folder. So if you wish to make any edits or verify any page, that's the folder to look in to.

Web Page Logins

Role: user



Role: Supervisor



Role: Purchase Department

Understanding the source code

hello.py

```
44 #Validation of the user
45 @app.route('/page', methods = ['POST', 'GET'])
46 def role():
47         if request.method == 'POST':
48                 session['user'] = request.form['username']
49                 pwd = request.form['password']
50                 print(pwd)
51         #hashing the password and comparing it with the value in database
52         pwd = hashFile(pwd)
53         conn = connectionOpen()
54         cur = conn.cursor()
55         print(session['user'])
56         print(pwd)
57         #Rows will only be returned if the hash value of password matches the value in databases
58         cur.execute("SELECT * FROM users WHERE username=? AND password=?",(session['user'],pwd,))
59         rows = cur.fetchall()
60         if rows == []:
61                 print("hash value of password not matching or username not valid")
62                 abort(500, 'Invalid login credentials')
63         role  = rows[0][2]
64         print(role)
65         #displaying web pages based on roles
66         if role == 'user':
67                 return render_template('page.html',msg="Welcome "+ session['user'])
68         elif role == 'supervisor':
69                 cur.execute("SELECT * FROM supervisor WHERE orderstatus=?",('CREATED',))
70                 rows = cur.fetchall()
71                 return render_template('supervisor.html', rows = rows)
72         elif role == 'purchase':
73                 cur.execute("SELECT * FROM orders WHERE status=?",('PROCESSING',))
74                 rows = cur.fetchall()
75                 return render_template('purchase.html', rows=rows)
76         connectionClose(conn)
77
78 #Displaying Order Creation Page
```

Line 45: @app.route shows the html page extension and **def role()** is the function the application will execute when *page is called.*

generateRSAkeys.py
command: python3 generateRSAkeys.py

This file is generates 1024-bit RSA public and private keys and saves as .pem files

```
import rsa
(pubkey, pvtkey) = rsa.newkeys(1024)
#Saving public key and private key in PEM format
exppub =pubkey.save_pkcs1(format='PEM')
exppriv = pvtkey.save_pkcs1(format='PEM')
f1 = open("pubpurDept.txt", "wb")
f2 = open("prvpurDept.txt","wb")
f1.write(exppub)
f2.write(exppriv)
f1.close()
f2.close()
```

hash.py
command python3 hash.py

On prompting please input username and also the new password

```
import hashlib
import sqlite3

hashObject = hashlib.sha3_224()
username = input("enter the username")
password = input("password")
conn = sqlite3.connect('database.db')
cur = conn.cursor()

for i in range (len(password)):
        hashObject.update(password[i].encode("utf8"))
        digest = hashObject.digest()
        cur.execute("UPDATE users SET password=? WHERE username=?", (digest,username))
        conn.commit()

conn.close()
```

Please view the demo video provided to know more about the operation.