CSCI 531 PROJECT REPORT

Electronic Health Record (EHR) System

TEAM

Sr No	Name	Email	USC ID
1	Anish Rajesh Adnani	adnani@usc.edu	4092610491
2	Supriya Sambhaji Patil	patilsup@usc.edu	3384672333

Introduction

EHR stands for Electronic Health Record. An EHR system is a digital version of a patient's paper chart containing their medical history, diagnoses, medications, treatment plans, and test results. It is designed to provide healthcare professionals with immediate access to a patient's health information, regardless of location or time, allowing for more efficient and coordinated care. EHRs also help to reduce medical errors, improve patient safety, and streamline administrative tasks.

The EHR system developed as a part of this project aims to meet the below-mentioned goals:

- Privacy: Patient privacy will be maintained. Unauthorized entitites cannot access the audit records
- Identification and authorization: All requests to access the audit data are authorized through a backend verification server
- Queries: Only authorized authorities will be able to query the audit record
- Decentralization: The system does not rely only on a single trusted entity

Demo Video: https://youtu.be/1GfhUc9RbLg

Programming Langauge

- Python 3.9.12

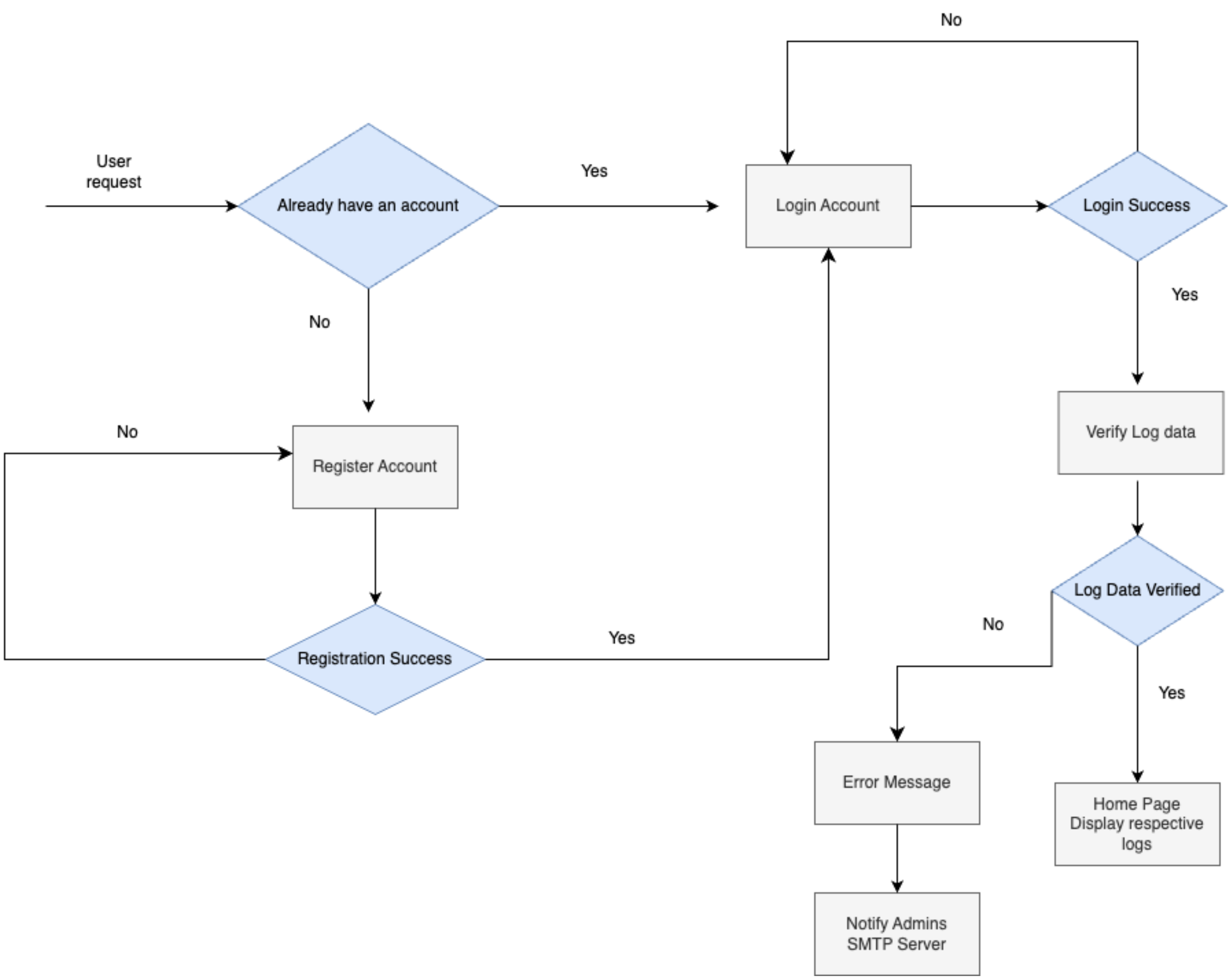
Technologies/ Libraries used

- Python
- Flask
- JSON
- SMTP
- AES

- RSA
- HTML
- CSS

How to run: python3 server.py

System Workflow



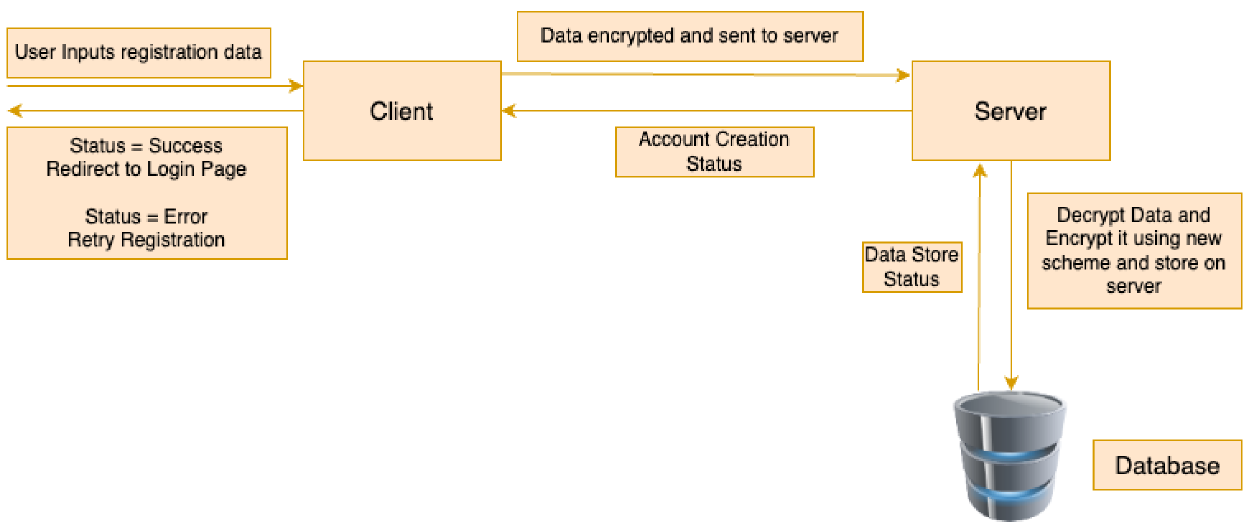
As seen in the above diagram user has two options to start using the application, either register a new account or login into a previously existing account. As a new registration request is received, the server stores the username, password, and other user details in the database, which is then used to authenticate users during login.

During login, every user enters their username and password, and the backend server checks whether the given username-password combination is correct. If the user is a 'normal user', all logs pertaining to the user are validated using the log's RSA signature. If the user is an 'audit company,' then all logs in the database are validated.

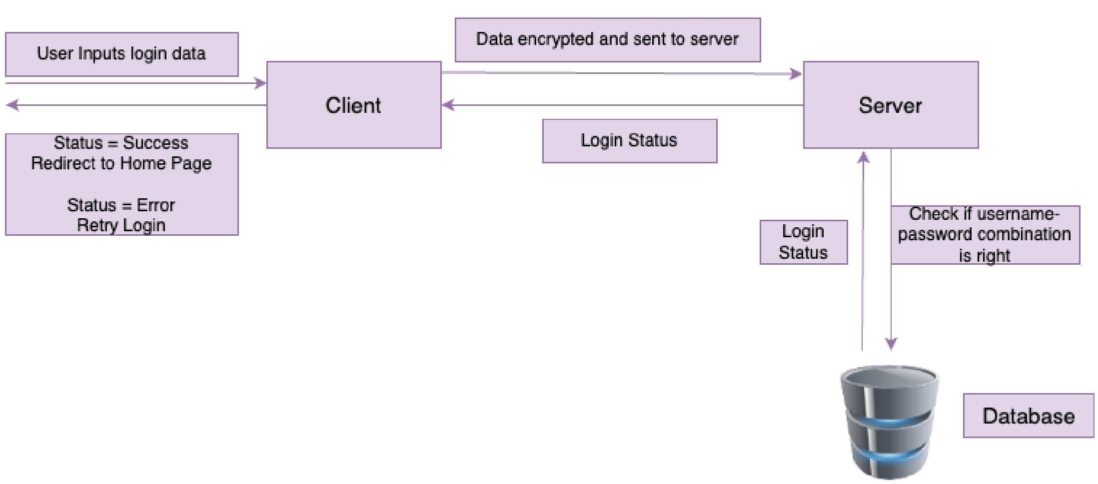
If, on checking, the backend server, finds data modification due to RSA signature mismatch, admins of the system are notified via an email about the affected logs.
Once the logs are validated, the user is redirected to the home page, where they can view their respective logs.
'Normal user': Can only view their audit logs
'Audit company': Can view all the logs

System Architecture

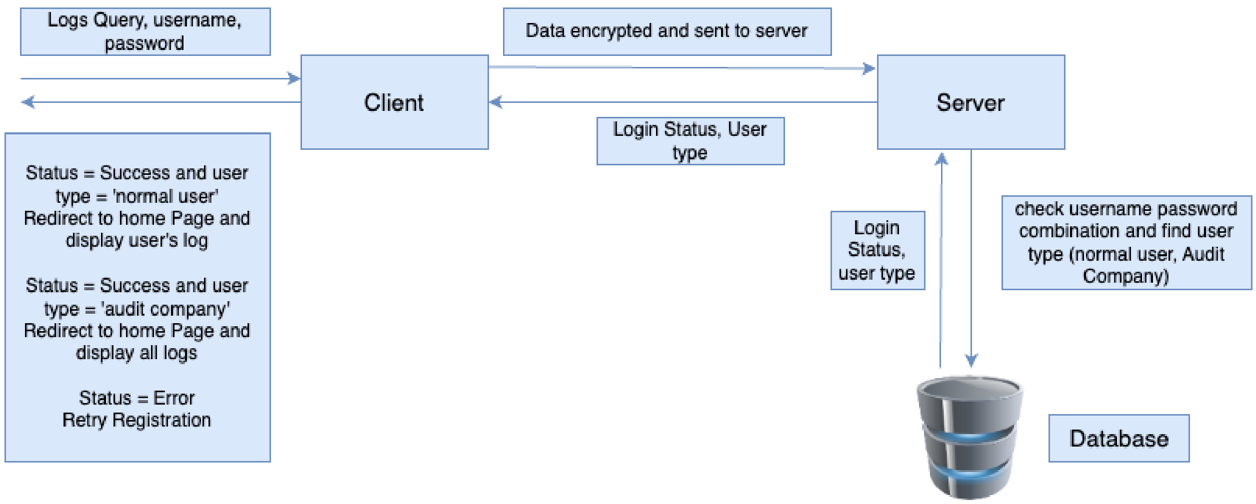
#1 Registration Architecture



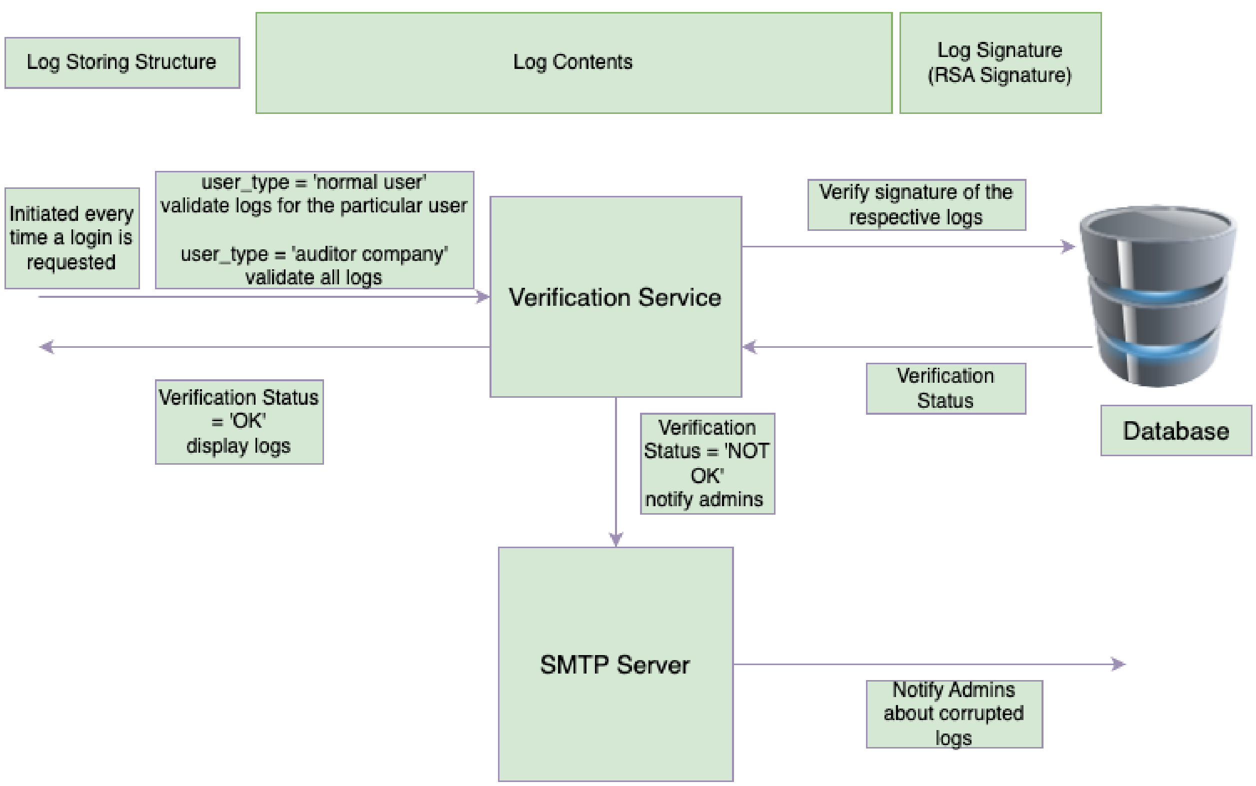
#2 Login Architecture



#3 Query Architecture



#4 Immutability Architecture (Data Tamper Prevention)



Implementation and Code Explanation

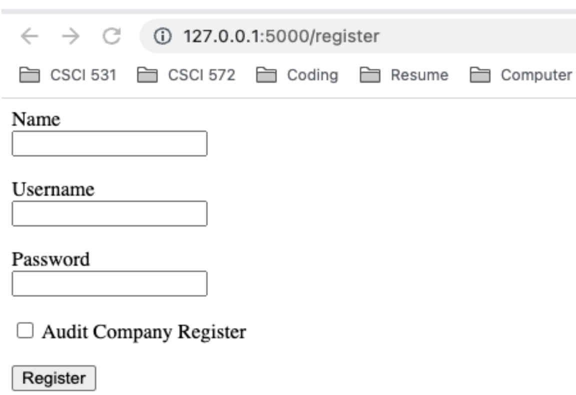
How to Start the Web Application

python3 server.py

http://127.0.0.1:5000/login
http://127.0.0.1:5000/register

User Registration

127.0.0.1:5000/register
Users can register a new account on the portal.
While registering user needs to input some basic information like (Name, username, and password), also the user needs to indicate whether they are a patient or an Audit company.
The passwords are then hashed, and further, all the user information is encrypted using AES encryption and stored in 'user_credentials.json'
Each user is also assigned a unique ID to identify each user uniquely.



The above depicts how the user information is stored in a JSON file.

The key in the JSON file is the encrypted username, and the array value has all the information about the user
Value array contains [user_id, name, user_name, password, company_login]
company_login = 'yes' means this user is an audit company

company_login = ‘no’ means this user is a normal patient

Following is the workflow for storing the user data during registration

Step 1: The user inputs the data, and a new user_id is generated for the user

Step 2 : [user_id, name, user_name, password, company_login] information needs to be stored

Step 3: Password is hashed (MD5 hashing)
[user_id, name, user_name, hash(password), company_login]

Step 4: Above array is then encrypted using AES encryption (ECB Mode)
[AES(user_id), AES(name), AES(user_name), AES(hash(password)), AES(company_login)]

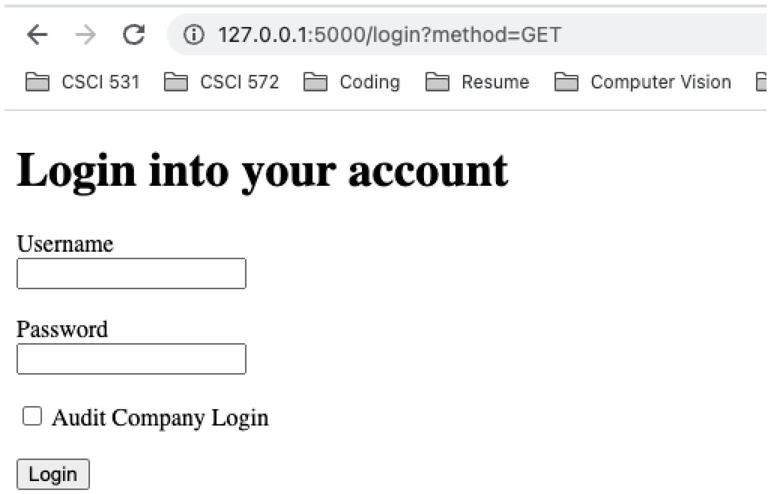
Step 5: The above information is then stored in the user_credentials.json file

```
{
AES(user_name) :  [AES(user_id), AES(name), AES(user_name),
AES(hash(password)), AES(company_login)]
}
```

On successful registration, the user is then redirected to the login page.

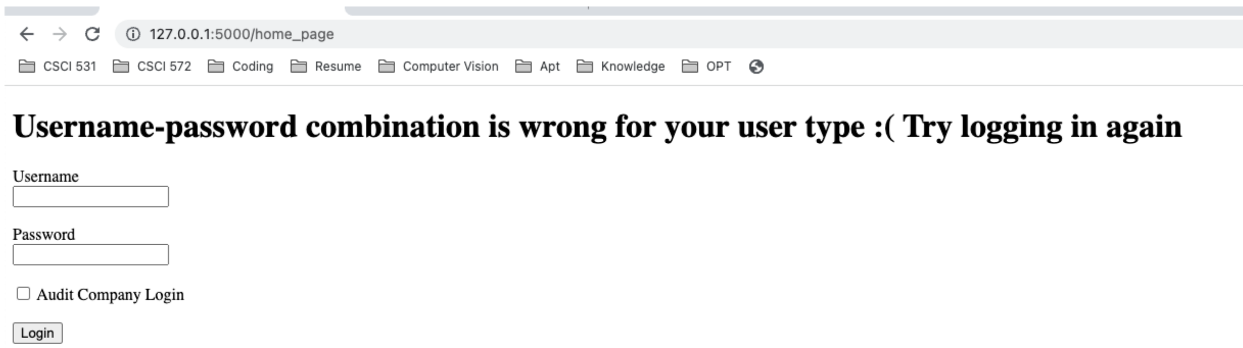
User Login

127.0.0.1:5000/login
If a user has an existing account on the portal, they can log in to view the logs using the login page.



For user login, username, password, and user_type are taken as input. These user input values are then validated through the backend server to check whether the details entered are legitimate.
Following are the steps for an account login verification

Step1 : Client enters [username, password, user_type] through the web interface
Step2: These fields are then passed to the server for verification
Step3 : The password is then hashed [username, hash(password), user_type]
Step4: All the fields are then encrypted to check in user_credentials.json
[aes(username), aes(hash(password)), aes(user_type)]
Step5 : Then aes(username) is searched in the user_credentials.json
Step6: If the user is found its password and user_type is checked to match the entered values
Step7: If the username does not exist, or password does not match, or the user_type is incorrect, the user is asked to enter the details again



Once the user logins successfully, they can view their respective logs depending on the user type.

User Home Page

http://127.0.0.1:5000/home_page

User Home Page displays the logs respective their user_type. If user is a patient they can only view their logs, while if the user is the audit company they can see all the logs in the system.

Storing of Logs

Logs are stored in user_logs.json
All the logs are encrypted using AES encryption

Key of the json file is the encrypted username
The value is a logs dictionary that has an array of logs for that user

username : {logs : [contains all the logs for this user] }
Each log has multiple information fields associated with it.
Each log has the following information stored [log_id, user_id, user_name, action_type, table_affected in EHR system, timestamp, notes, log signature]
All the log fields are concatenated and a signature (RSA signature) is produced for the log. This signature can be used to verify that the contents of the logs are not changed by any unauthorized entity.

```python
def compute_sign(message):
    pvt_key = rsa.PrivateKey.load_pkcs1(read_contents('privatekey.key'))
    signature = rsa.sign(message.encode(), pvt_key, 'SHA-512')
    ans = signature.hex()
    return ans
```


```json
  },
  "1761fa529dcc13337ce8b0207df030b2": {
      "logs": [
          [
              "851c1d815b1d4aba8dced3d8f8fbcdbf",
              "1761fa529dcc13337ce8b0207df030b2",
              "71367c3f787eacf617b5c53b91162b95",
              "6358797bf3953eb3dc7bff1ba4182974",
              "b83d69ec3ae04f0aac9bd9219381b5a3",
              "98e16c4cf968736cedc4f09c4235ff9f4a008740cf6f54ba74f5bca88241abf4",
              "155c90436583d975b928d87c7066cf870a4e29ca2d199040854a9c3d0637eff8",
              "b548467d7ddc5686657b6cfa2f4e6ef8ea18c67e19cd042a6a343382e750ab58",
              "1ad38ddce1538b6b55e3a37535a2f803ab4219176755e37a5b504e6da547580f8ee89418a8a7a5b4eea13e4af4bc3e0b922a95a216a5cc7a8b8c45abe0
          ],
          [
              "1f0c30aea947d04632998626c3ab0c0e",
              "1761fa529dcc13337ce8b0207df030b2",
              "71367c3f787eacf617b5c53b91162b95",
              "9e23c70066d91dcfc9a0e6b6a33fc552",
              "f509c2a2e50e0e0ed983971122b75254",
              "9988b64a49cd58f51775133a150ee3e3d41e0ea75f96052f6e47c047868a3b8e",
              "5b2cd13a1986ecb6bc3a7d88526672974d02389b5ff358f1d22aa6a530ce9554",
              "f0a9566d92f9ee7e34079d59ee9c4c278e7a9d2d0ebc6144f0990d6d4bed2c0d",
              "029f7d094a67a08d57e1606130b2c371cc80e1d1eb4c68ac70977c096bffc150ff075c6c27aa0e7f7b36c62380f5c91e945beff7f62391382fae4a53bc
          ],
          [
              "9e23c70066d91dcfc9a0e6b6a33fc552",
              "1761fa529dcc13337ce8b0207df030b2",
              "71367c3f787eacf617b5c53b91162b95",
              "9e23c70066d91dcfc9a0e6b6a33fc552",
```


The above diagram shows how the logs are stored in user_logs.json

The signature of the entire user_logs.json is stored in overall_sign.json (This also helps us verify no unauthorized change has been made in the database, it also helps us identify if any log record has been deleted from the user_logs.json)

overall_sign.json — project

verification_backend.py | user_logs.json | email_notifier.py | overall_sign.json

overall_sign.json > ...


```json
{
    "overall_sign": "7089ada2fce5aeecc1f914bbdfa4e2e34688f2921d18d3fd9b42bd089e8098557faccac9e2217574f89c515159c035ee86c23c240cba31f67f21d3
}
```


The above file stores the RSA signature of the entire user_logs.json file

All the requests from the login page are redirected to the home page
Any request directly coming to the home page will not display any result

127.0.0.1:5000/home_page

CSCI 531 | CSCI 572 | Coding | Resume | Computer Vision | Apt | Knowledge | OPT

BadRequestKeyError

```
werkzeug.exceptions.BadRequestKeyError: 400 Bad Request: The browser (or proxy) sent a request that this server could not understand.
KeyError: 'username'
```

Traceback (most recent call last)

```
File "/Users/anishadnani/opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/flask/app.py", line 2548, in __call__
    return self.wsgi_app(environ, start_response)
File "/Users/anishadnani/opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/flask/app.py", line 2528, in wsgi_app
    response = self.handle_exception(e)
File "/Users/anishadnani/opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/flask/app.py", line 2525, in wsgi_app
    response = self.full_dispatch_request()
File "/Users/anishadnani/opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/flask/app.py", line 1822, in full_dispatch_request
    rv = self.handle_user_exception(e)
File "/Users/anishadnani/opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/flask/app.py", line 1820, in full_dispatch_request
    rv = self.dispatch_request()
File "/Users/anishadnani/opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/flask/app.py", line 1796, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
File "/Users/anishadnani/Desktop/USC/CSCI 531/project/server.py", line 49, in home_page_user
    user_name = request.form['username']
File "/Users/anishadnani/opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/werkzeug/datastructures.py", line 375, in __getitem__
    raise exceptions.BadRequestKeyError(key)
```

werkzeug.exceptions.BadRequestKeyError: 400 Bad Request: The browser (or proxy) sent a request that this server could not understand.
KeyError: 'username'

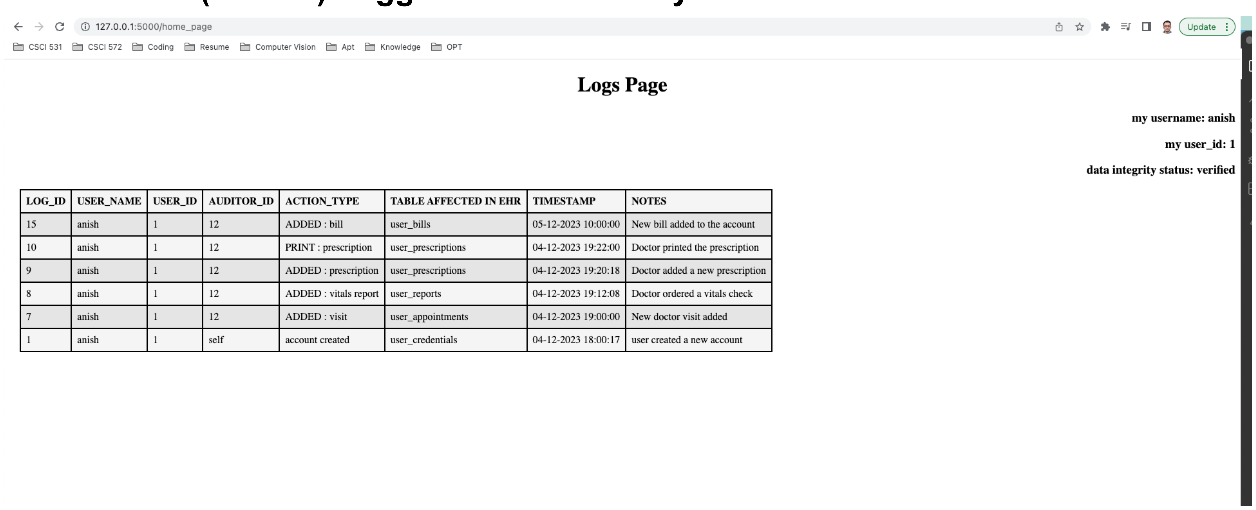
Once the user logs in to their account, all the logs in the database are verified before showing it to the user. Also the signature of the entire log file is verified to make sure there have not been any unauthorized log deletions.

```python
def verify_sign(message, sign):
    pub_key = rsa.PublicKey.load_pkcs1(read_contents('publickey.key'))

    try:
        rsa.verify(message.encode(), bytearray.fromhex(sign), pub_key)
        return True
    except:
        return False
```

```python
def verify_complete_file_sign():
    with open('user_logs.json', 'r+') as f:
        lines = f.readlines()

    all_logs = ",".join(lines)

    with open('overall_sign.json', 'r+') as f:
        data = json.load(f)
        stored_sign = data['overall_sign']

    verified = verify_sign(all_logs, stored_sign)
    if verified:
        return True
    else:
        subject = 'Data integrity compromised'
        content = '[Unauthorized Log Deletions] Some Logs missing in the audit database' + '\
        content+= 'Unauthorized deletions have been made to the audit records' + '\n'
        content+= 'Immediate action needs to be taken' + '\n'
        msg_sent = send_email_notification(subject, content)
        return False
```

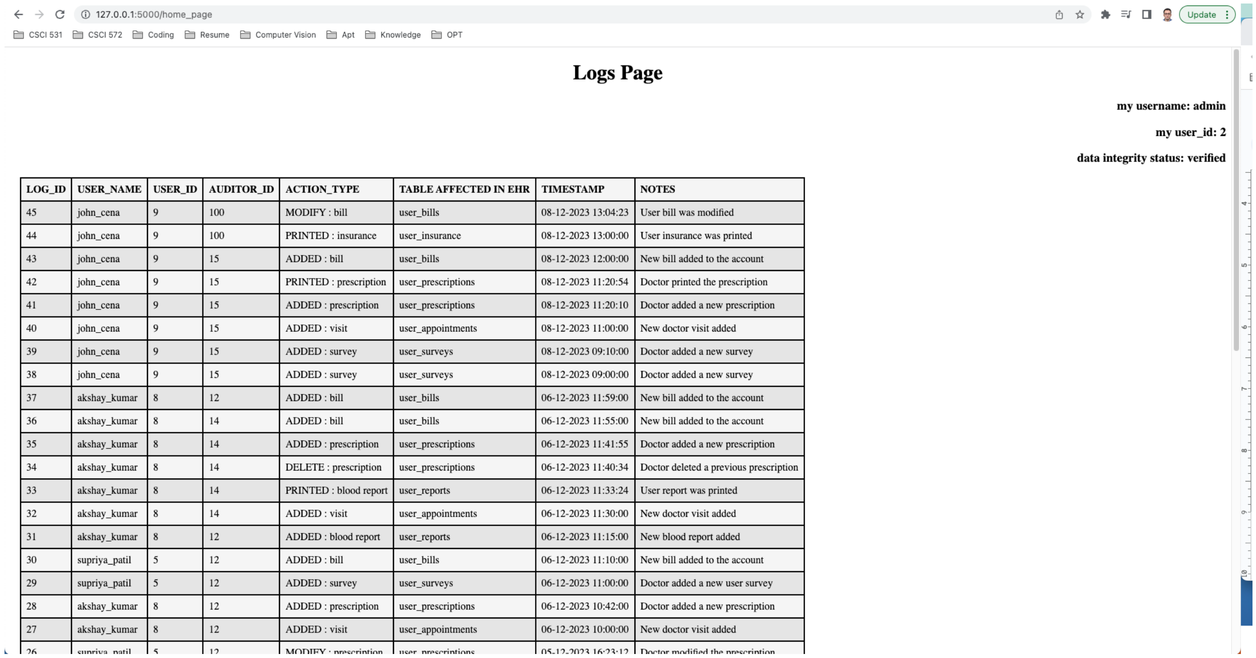
If all the logs are successfully verified, they are displayed to the user, else an error message is displayed, and admins are notified via mail about the unauthorized changed logs.

Normal User (Patient) Logged in successfully.



LOG_ID	USER_NAME	USER_ID	AUDITOR_ID	ACTION_TYPE	TABLE AFFECTED IN EHR	TIMESTAMP	NOTES
15	anish	1	12	ADDED : bill	user_bills	05-12-2023 10:00:00	New bill added to the account
10	anish	1	12	PRINT : prescription	user_prescriptions	04-12-2023 19:22:00	Doctor printed the prescription
9	anish	1	12	ADDED : prescription	user_prescriptions	04-12-2023 19:20:18	Doctor added a new prescription
8	anish	1	12	ADDED : vitals report	user_reports	04-12-2023 19:12:08	Doctor ordered a vitals check
7	anish	1	12	ADDED : visit	user_appointments	04-12-2023 19:00:00	New doctor visit added
1	anish	1	self	account created	user_credentials	04-12-2023 18:00:17	user created a new account

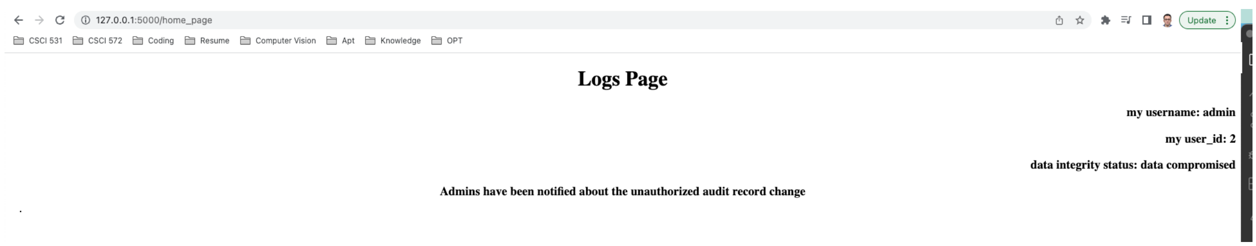
Audit Company Logged in Successfully



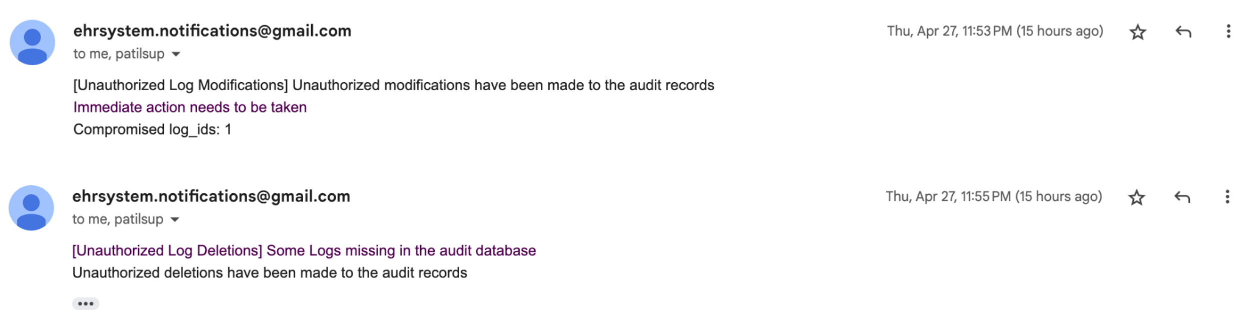
LOG_ID	USER_NAME	USER_ID	AUDITOR_ID	ACTION_TYPE	TABLE AFFECTED IN EHR	TIMESTAMP	NOTES
45	john_cena	9	100	MODIFY : bill	user_bills	08-12-2023 13:04:23	User bill was modified
44	john_cena	9	100	PRINTED : insurance	user_insurance	08-12-2023 13:00:00	User insurance was printed
43	john_cena	9	15	ADDED : bill	user_bills	08-12-2023 12:00:00	New bill added to the account
42	john_cena	9	15	PRINTED : prescription	user_prescriptions	08-12-2023 11:20:54	Doctor printed the prescription
41	john_cena	9	15	ADDED : prescription	user_prescriptions	08-12-2023 11:20:10	Doctor added a new prescription
40	john_cena	9	15	ADDED : visit	user_appointments	08-12-2023 11:00:00	New doctor visit added
39	john_cena	9	15	ADDED : survey	user_surveys	08-12-2023 09:10:00	Doctor added a new survey
38	john_cena	9	15	ADDED : survey	user_surveys	08-12-2023 09:00:00	Doctor added a new survey
37	akshay_kumar	8	12	ADDED : bill	user_bills	06-12-2023 11:59:00	New bill added to the account
36	akshay_kumar	8	14	ADDED : bill	user_bills	06-12-2023 11:55:00	New bill added to the account
35	akshay_kumar	8	14	ADDED : prescription	user_prescriptions	06-12-2023 11:41:55	Doctor added a new prescription
34	akshay_kumar	8	14	DELETE : prescription	user_prescriptions	06-12-2023 11:40:34	Doctor deleted a previous prescription
33	akshay_kumar	8	14	PRINTED : blood report	user_reports	06-12-2023 11:33:24	User report was printed
32	akshay_kumar	8	14	ADDED : visit	user_appointments	06-12-2023 11:30:00	New doctor visit added
31	akshay_kumar	8	12	ADDED : blood report	user_reports	06-12-2023 11:15:00	New blood report added
30	supriya_patil	5	12	ADDED : bill	user_bills	06-12-2023 11:10:00	New bill added to the account
29	supriya_patil	5	12	ADDED : survey	user_surveys	06-12-2023 11:00:00	Doctor added a new user survey
28	akshay_kumar	8	12	ADDED : prescription	user_prescriptions	06-12-2023 10:42:00	Doctor added a new prescription
27	akshay_kumar	8	12	ADDED : visit	user_appointments	06-12-2023 10:00:00	New doctor visit added
26	supriya_patil	5	12	MODIFY : prescription	user_prescriptions	05-12-2023 16:23:12	Doctor modified the prescription

A user is trying to log in, and logs cannot be verified.

The data integrity status shows the data is compromised, and the logs will not be displayed to the user.



Admins of the application are notified via email about the unauthorized change.



Cryptographic Components

There are various cryptographic components used in the application

- **AES (Encryption and Decryption)**
  All the data in the database is encrypted before storing. We have used AES encryption with ECB mode for the purpose of encryption
  Following are screenshots of the encrypted data
  user_logs.json


```
{} user_logs.json > {} 1761fa529dcc13337ce8b0207df030b2 > [ ] logs > [ ] 0
    },
    "1761fa529dcc13337ce8b0207df030b2": {
        "logs": [
            [
                "851c1d815b1d4aba8dced3d8f8fbcdbf",
                "1761fa529dcc13337ce8b0207df030b2",
                "71367c3f787eacf617b5c53b91162b95",
                "6358797bf3953eb3dc7bff1ba4182974",
                "b83d69ec3ae04f0aac9bd9219381b5a3",
                "98e16c4cf968736cedc4f09c4235ff9f4a008740cf6f54ba74f5bca88241abf4",
                "155c90436583d975b928d87c7066cf870a4e29ca2d199040854a9c3d0637eff8",
                "b548467d7ddc5686657b6cfa2f4e6ef8ea18c67e19cd042a6a343382e750ab58",
                "1ad38ddce1538b6b55e3a37535a2f803ab4219176755e37a5b504e6da547580f8ee89418a8a7a5b4eea13e4af4bc3e0b922a95a216a5cc7a8b8c45abe0
            ],
            [
                "1f0c30aea947d04632998626c3ab0c0e",
                "1761fa529dcc13337ce8b0207df030b2",
                "71367c3f787eacf617b5c53b91162b95",
                "9e23c70066d91dcfc9a0e6b6a33fc552",
                "f509c2a2e50e0e0ed983971122b75254",
                "9988b64a49cd58f51775133a150ee3e3d41e0ea75f96052f6e47c047868a3b8e",
                "5b2cd13a1986ecb6bc3a7d88526672974d02389b5ff358f1d22aa6a530ce9554",
                "f0a9566d92f9ee7e34079d59ee9c4c278e7a9d2d0ebc6144f0990d6d4bed2c0d",
                "029f7d094a67a08d57e1606130b2c371cc80e1d1eb4c68ac70977c096bffc150ff075c6c27aa0e7f7b36c62380f5c91e945beff7f62391382fae4a53bc
            ],
            [
                "9e23c70066d91dcfc9a0e6b6a33fc552",
                "1761fa529dcc13337ce8b0207df030b2",
                "71367c3f787eacf617b5c53b91162b95",
```


  user_credential.json


```
{} user_credentials.json > {} username_password
                "71367c3f787eacf617b5c53b91162b95",
                "0216b03a8db3b10f34933d6c30b30109",
                "1761fa529dcc13337ce8b0207df030b2",
                "2c7bab8b0089b4cffa041d9ce7ac372ec41004a71369f9afab78a48d01234ce28fe3af622bbf7728e3ecd250fd7df68bb3e4e5a2d6a33905897c45aeaf2a50
                "9da19db82af8f512f85cb5be770ce2a0"
            ],
            "035815825930b373591147d6929ebd75": [
                "fab9b8c24f8dd9da3c562fb9652b2f50",
                "24f21ea78470ecdf3bbe48236b5ce88f",
                "035815825930b373591147d6929ebd75",
                "59d849f510e1c2732647dd65c289c440003a341ef4978ada25c921645ef874512229d732a38c14279d236e2d262cb2aa65aa1f161060a5d2a1f3747ccfa9a7
                "9da19db82af8f512f85cb5be770ce2a0"
            ],
            "f93fe560ed692bef94a62ddeb3af6e4a": [
                "765df0129f48a54c91a7ceef26c1ea2a",
                "f786736a94fb84a1b44f3f44e16806c4",
                "f93fe560ed692bef94a62ddeb3af6e4a",
                "9b28835a517e5de8904bc89c740d66e25bcef177bc7bbf8e493c7e887b12171f06a829361e43b09dac1271981a9ba21c96ddb0634a40d7e620b9d847d6edbd
                "9da19db82af8f512f85cb5be770ce2a0"
            ],
            "1026082bfb24801a92af2950696b4b7b": [
                "a3f7415bd736acd710418de7ea3caa8f",
                "6ff23ee81f91a29d0f679faef905a5a7",
                "1026082bfb24801a92af2950696b4b7b",
                "49c9c652a2e4b21be9b236ad8c30f5238ea25a6fd34b5d18617a459b2a8ce809496a90f52dd06f49254620e20caaea711e2d3a809bad72db02128c1a4c1040
                "09c0c2e17637520aa1f403db4afe8401"
            ],
            "88e0baf04c117a0936cbfb9204627111": [
                "8fa663507e2e322c622e891a381d68dd",
```

- **MD5 hash**
  All user passwords are hashed and encrypted before storing in the user_credentials.json. This improves user privacy as the password hash cannot be reverse-engineered back to get back the original password.

```python
def create_new_user(name, user_name, user_password, company_login):

    name = aes_encrypt(name)  ## encrypt
    user_name = aes_encrypt(user_name)  ## encrypt
    user_password = hashlib.sha256(user_password.encode('utf-8')).hexdigest() ## hash the password
    user_password = aes_encrypt(user_password)  ## encrpyt
    company_login = aes_encrypt(company_login)  ## encrypt
```


```json
    ],
    "f878ad837628c6bc2914d93392bee840": [
        "851c1d815b1d4aba8dced3d8f8fbcdbf",
        "e1f2b34f203a3ff0751c3db49c9a3256",
        "f878ad837628c6bc2914d93392bee840",
        "49c9c652a2e4b21be9b236ad8c30f5238ea25a6fd34b5d18617a459b2a8ce809496a90f52dd06f49254620e20caaea711e2d3a809bad72db02128c1a4c1
        "09c0c2e17637520aa1f403db4afe8401"
    ],
```


- **RSA Signature and Verification**
  All logs in the audit system are stored with their RSA signature, which can be used to verify unauthorized changes in the database.

```python
def compute_sign(message):
    pvt_key = rsa.PrivateKey.load_pkcs1(read_contents('privatekey.key'))
    signature = rsa.sign(message.encode(), pvt_key, 'SHA-512')
    ans = signature.hex()
    return ans

def verify_sign(message, sign):
    pub_key = rsa.PublicKey.load_pkcs1(read_contents('publickey.key'))

    try:
        rsa.verify(message.encode(), bytearray.fromhex(sign), pub_key)
        return True
    except:
        return False
```

System Goals

1. Privacy Goal
   Patient Privacy is maintained by implementing the below-mentioned steps
   - Patients can only view their logs
   - All the Data in the database is encrypted
   - Passwords are hashed and encrypted before storing in the database

   Unauthorized entities cannot edit the audit log data
   - All logs are encrypted and stored with their RSA signature. The signature is verified at each login, and in case the verification fails, the admins are notified via email the compromised log id.

2. Identification and Authorization
   All users are authenticated. All the requests to access the audit data are authorized by implementing the below-mentioned functionality.
    - All users have a unique username.
    - All users have a username-password combination to login into the system
    - If the user's username, password, and user type are successfully verified in the database, they are directed to the home_page, where they can see their respective logs
    - All the logs are verified before displaying them to the user

3. Queries
   Only authorized users should be able to query the database. No unauthorized entity should be able to edit the audit logs.
    - The system is designed such that patients can only query their own logs, and the audit company can query all the logs in the database.
    - If any user directly tries to open home_page without logging in to the system, an error message will appear, and no logs will be shown.
    - All logs have an RSA signature stored in the database. All the logs are verified before displaying them to the user. Hence if there is any unauthorized change in the database, it will be immediately notified to the admins via email.

Assumptions and Limitations

Some of the assumptions considered while developing this project are as follows.

- Public and private keys are already shared between entities using a secured medium, and the attacker does not have access to the keys.
- The Backend server here is a Python file but practically, it would be hosted on a different machine (Hosted on a machine other than on the client's machine)

Some Limitations of the project are

- As the current database for the system is a JSON file, we can only protect data changes in the JSON file; there might be a possibility that someone deletes the entire JSON file. Since the current build of the project works locally, we face the limitation; if the backend server was running on the cloud, we could have created an AWS instance and stored all the data in AWS S3.

Additional Improvements (Project Option 1)

1> Zero Knowledge Proofs

Zero knowledge proofs (ZKP) can be used to authenticate user login by proving that a user knows a secret without actually revealing the secret itself. This can be done using a ZKP protocol called a password-authenticated key agreement (PAKE).

In a PAKE protocol, the user and the server both start with the same value, which is usually the user's password. Using cryptographic algorithms, the user and the server then exchange a series of messages to verify that they both have the same password without actually revealing the password to each other.

One example of a PAKE protocol is the Secure Remote Password (SRP) protocol, which uses a combination of cryptographic hash functions and modular exponentiation to authenticate the user without revealing their password. In the SRP protocol, the user and the server each generate a random value, which is then combined with the user's password to generate a shared secret. This shared secret is then used to generate a cryptographic key, which is used to authenticate the user's login.

PAKE protocols are designed to be resistant to attacks such as dictionary attacks, where an attacker tries to guess the user's password by trying a large number of possible passwords. By using a ZKP, such as a PAKE protocol, to authenticate user login, the system can achieve a higher level of security and privacy than traditional password-based authentication methods.

Overall, zero-knowledge proofs provide a powerful tool for achieving secure and private authentication and can be used in a variety of applications, including user login.

2> Homomorphic encryption

Homomorphic encryption is a form of encryption that allows computations to be performed on ciphertext (encrypted data) without requiring the data to be decrypted first. This means that the data can remain encrypted throughout the computation process, providing a higher level of privacy and security.

In other words, homomorphic encryption allows computations to be performed on data while it is still encrypted without revealing the underlying data to the computation system. This can be useful in situations where data privacy and security are important, such as in cloud computing or data analysis.

There are two types of homomorphic encryption: fully homomorphic encryption (FHE) and partially homomorphic encryption (PHE).

FHE allows any computation to be performed on encrypted data, including addition, multiplication, and more complex operations. This makes FHE very powerful, but it is also very computationally expensive and requires a lot of resources.

PHE, on the other hand, only allows for specific computations to be performed on encrypted data, such as addition or multiplication. While less powerful than FHE, PHE is much more efficient and practical for real-world applications.

Homomorphic encryption is still an active area of research.

References

[1] https://en.wikipedia.org/wiki/Zero-knowledge_proof

[2] https://blog.chain.link/what-is-a-zero-knowledge-proof-zkp/

[3] https://en.wikipedia.org/wiki/Homomorphic_encryption

[4] https://brilliant.org/wiki/homomorphic-encryption/

[5] https://www.tutorialspoint.com/python/python_sending_email.htm