Blockchain flow diagram
Login architecture diagram
Viva explanation for ONLY blockchain section
Code snippet breakdown with visuals

Just say **"give blockchain diagram"**.

> is there any code related to blockchain

YES — Your project *does* contain blockchain-related code.

It is located directly inside **`app.py`**, and that's all you need for blockchain integration in your project.

Let me show you **exactly** which parts of your code are blockchain-related.

Blockchain Code Inside `app.py`

Here are the **three main components**:

---

1. Web3 Connection (Ganache / Ethereum)

This connects your Flask app to the blockchain.

```
python

from web3 import Web3
from eth_account.messages import encode_defunct

INFURA_URL = "http://127.0.0.1:7545"  # Ganache local blockchain
web3 = Web3(Web3.HTTPProvider(INFURA_URL))
```

✔️ Connects to Ethereum blockchain
✔️ Uses Web3.py
✔️ Points to Ganache (your local blockchain network)

---

2. Signature Verification Function

This is the heart of blockchain authentication.
It verifies the MetaMask signature.

```
python

def verify_signature(address, message, signature):
    message_encoded = encode_defunct(text=message)
    recovered_address = web3.eth.account.recover_message(message_encoded, signature=signature)
    return recovered_address.lower() == address.lower()
```

✔️ Recovers Ethereum address from signature
✔️ Confirms the signed message was created by that wallet

✔️ Prevents fake login attempts

This is **true decentralized authentication**.

---

3. Flask Route for Blockchain Login

This is where MetaMask login is processed.

```
python

@app.route('/verify-login', methods=['POST'])
def verify_login():
    data = request.json
    user_address = data.get("address")
    message = data.get("message")
    signature = data.get("signature")

    if not user_address or not message or not signature:
        return jsonify({"error": "Missing parameters"}), 400

    is_valid = verify_signature(user_address, message, signature)
    if not is_valid:
        return jsonify({"error": "Invalid signature"}), 401

    return jsonify({"redirect": url_for('detection', address=user_address)})
```

✔️ Receives address + signature from frontend
✔️ Verifies Ethereum signature
✔️ If valid → logs user in
✔️ If invalid → rejects login

---

4. Frontend HTML (MetaMask Login Button)

Inside **`login.html`** (your MetaMask screen), JavaScript handles:

- Connecting to MetaMask
- Signing message
- Sending signature to Flask backend

Example code your file should contain:

```
javascript

const accounts = await ethereum.request({ method: 'eth_requestAccounts' });
const message = "Login authentication";
const signature = await ethereum.request({
    method: 'personal_sign',
    params: [message, accounts[0]],
});
```

✔️ This signs the login message
✔️ Sends to **`/verify-login`**
✔️ Backend verifies using Web3

Conclusion

YES blockchain is integrated

YES code exists (in `app.py` and `login.html`)

It uses:

- MetaMask
- Web3.py
- Ethereum signatures
- Ganache

Purpose:

Decentralized authentication for your web app.

If you want

I can also extract the blockchain parts and create a **blockchain-only code file**, or generate a **blockchain-only flowchart** for your presentation.

Just tell me:
👉 **"Give blockchain-only breakdown"**

> "Give blockchain-only breakdown"

Here is a **clean, complete, easy-to-present** breakdown of ONLY the blockchain portion of your project — perfect for viva, slides, documentation, and Q&A.

BLOCKCHAIN-ONLY BREAKDOWN

(Clear, short, and professional — exactly what examiners expect.)

---

1. Purpose of Blockchain in This Project

Your blockchain component is used for:

Decentralized Authentication

Instead of usernames and passwords, users log in using their **MetaMask Ethereum wallet**.

Secure Identity Verification

A user cannot fake login without the private key.

No password storage

No database required.
No password leaks.
Only wallet signatures.

Perfect for medical privacy

Fits the privacy goals of skin cancer detection systems.

---

2. How Blockchain Login Works (Simple Explanation)

This is exactly how you should explain it:

> "When the user clicks Login, MetaMask asks them to sign a message using their private key.
> This signed message is sent to our Flask backend.
> The backend uses Web3.py to recover the signer's Ethereum address, and if the recovered address matches the user address, login is approved.
> No passwords, no data storage — authentication is decentralized."

---

3. Blockchain Flow Diagram

```
pgsql

User clicks "Login"
        |
        v
 MetaMask opens
        |
User signs a message (using private key)
        |
        v
Signature + Address ----> Flask Backend
                          (verify-login route)
        |
        v
 Web3.py recovers signer address
        |
Compare to user address
        |
     Valid?
   /        \
Yes          No
|            |
Grant Access  Reject
```

---

4. Blockchain Code (Backend — Flask / Web3.py)

1. Connect to Ethereum Blockchain (Ganache)

```
python

web3 = Web3(Web3.HTTPProvider("http://127.0.0.1:7545"))
```

2. Signature Verification Function

This is the heart of your blockchain logic:

```
python

from eth_account.messages import encode_defunct

def verify_signature(address, message, signature):
    message_encoded = encode_defunct(text=message)
    recovered_address = web3.eth.account.recover_message(message_encoded, signature=signature)
    return recovered_address.lower() == address.lower()
```

✔️ Guarantees only the real wallet owner can log in

3. Flask Route That Uses Blockchain

```
python

@app.route('/verify-login', methods=['POST'])
def verify_login():
    data = request.json
    user_address = data['address']
    message = data['message']
    signature = data['signature']

    if verify_signature(user_address, message, signature):
        return jsonify({"redirect": url_for('detection', address=user_address)})
    else:
        return jsonify({"error": "Invalid signature"}), 401
```

✔️ If signature is valid → login successful
✔️ If not → unauthorized access

5. Blockchain Code (Frontend — MetaMask / JavaScript)

Inside **`login.html`**, MetaMask signs a message:

```
javascript

const accounts = await ethereum.request({ method: 'eth_requestAccounts' });
const user = accounts[0];

const message = "Login authentication";
const signature = await ethereum.request({
    method: "personal_sign",
    params: [message, user]
});

// send to Flask
fetch("/verify-login", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
```

```
    body: JSON.stringify({ address: user, message: message, signature: signature })
});
```

✔️ MetaMask signs message
✔️ Signature sent to backend for verification

---

6. Why Blockchain Is Used (Examiner-Friendly Answer)

Say this clearly in viva:

> "We used blockchain for decentralized authentication.
> Instead of storing usernames and passwords, the user logs in using MetaMask.
> The user signs a message with their private key, and we verify the signature in Flask using Web3.py.
> This ensures secure access without storing any sensitive credentials, making it ideal for medical systems."

---

7. Output of Blockchain System

Secure Login

No password database

Authentication tied to blockchain identity

Protected access to skin cancer detection system

Want a PPT slide for blockchain?

I can generate:

- A blockchain authentication slide
- A MetaMask flow diagram
- Code snippet slide
- Viva note slide

Just say **"give blockchain PPT slides"**.

> wts the role of metamask

MetaMask plays a **very important and clear role** in your project.
Here is the perfect explanation for **viva, presentation, and documentation**.

Role of MetaMask in Your Project