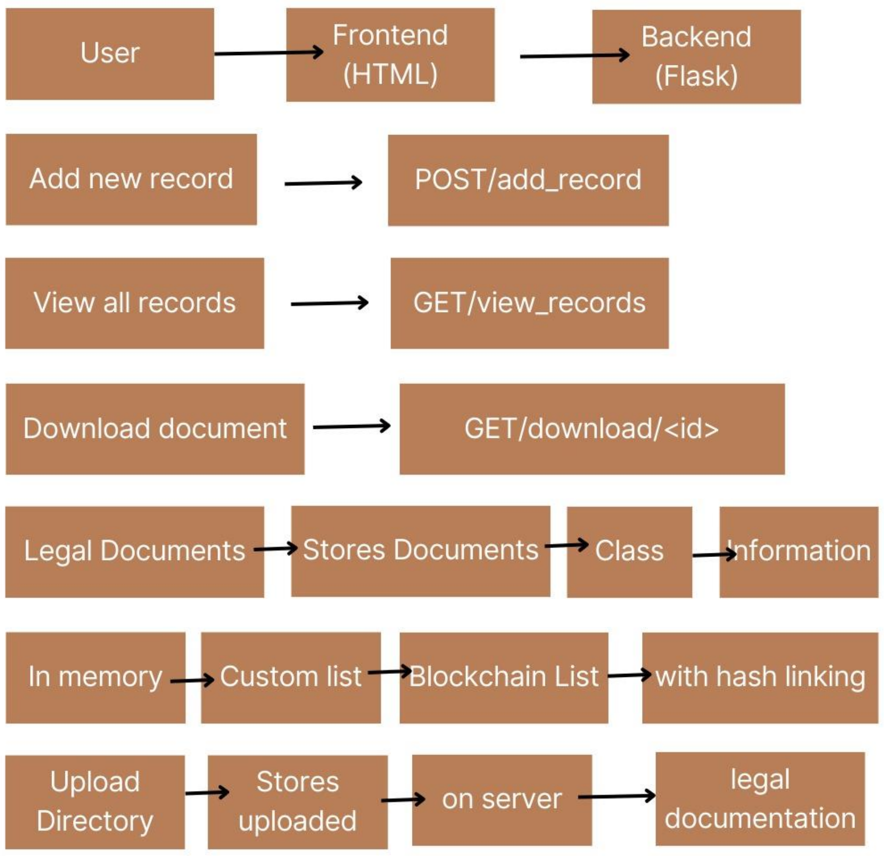
Developing a Blockchain-Based e-Vault for Legal Records

Architecture

1. **Flask Web Application**:
    - **Frontend**:
        - HTML templates for user interactions (adding records, viewing records, downloading documents).
    - **Backend**:
        - Flask routes for handling requests and responses.
        - Legal Documents class to represent legal document records.
2. **Data Storage**:
    - In-memory blockchain list to store instances of Legal Documents.
3. **File Upload**:
    - Uploaded legal documents are stored in an uploads directory on the server.
4. **Blockchain**:
    - Each Legal Documents instance contains:
        - Timestamp
        - Name
        - UID (Unique Identifier)
        - Age
        - Path to the legal document
        - Hash of the current record
        - Hash of the previous record

Block Diagram:



Explanation:

- The diagram shows three main components: User, Frontend (HTML), and Backend (Flask).
- The user interacts with the frontend through HTML templates.
- The frontend sends requests to the backend using Flask routes. These routes handle the user's actions and generate responses.
- The backend utilizes the Legal Documents class to represent legal document data.
- An in-memory blockchain list stores instances of Legal Document. This is a custom implementation that likely uses hashing to link documents together.
- Uploaded documents are stored in a directory named "uploads" on the server.

The arrows represent the flow of information between components. For instance:

- When a user adds a new record, the frontend sends a POST request to the /add record route.
- The backend processes this request, creates a new Legal Documents instance, and stores it in the in-memory blockchain list.
- Similarly, other routes handle viewing all records and downloading documents based on their unique identifier.

Algorithms

1. Record Creation Algorithm
2. Hash Calculation Algorithm
3. Blockchain Integrity Algorithm
4. File Handling Algorithm

1. Record Creation Algorithm

Purpose: To create a new record and add it to the blockchain.

Steps:

1. Collect user input (name, UID, age, and legal document).
2. Save the uploaded legal document to the file system.
3. Create a new instance of the Legal Documents class.
4. Calculate the hash of the new record.
5. Append the new record to the blockchain.

Pseudocode:

```
def add_record(name, uid, age, legal_document):
    evidence_path = save_file(legal_document)
    new_record = PatientRecord(name, uid, age, evidence_path)
    blockchain.append(new_record)
```

2. Hash Calculation Algorithm

Purpose: To generate a unique hash for each record using the SHA-256 algorithm.

Steps:

1. Concatenate the record's attributes (timestamp, name, UID, age, evidence path).
2. Encode the concatenated string.
3. Generate the SHA-256 hash of the encoded string.

Pseudocode:

```
def calculate_hash(record):
    hash_data =
f"{record.timestamp}{record.name}{record.uid}{record.age}{record.evidence_path}"
```

```
    return sha256(hash_data.encode()).hexdigest()
```

3. Blockchain Integrity Algorithm

Purpose: To ensure the integrity of the blockchain by linking each record to the previous one via hashes.

Steps:

1. When creating a new record, calculate the hash of the previous record.
2. Store the previous record's hash in the new record.

Pseudocode:

```
def calculate_previous_hash():

    if len(blockchain) > 0:
        previous_record = blockchain[-1]
        return previous_record.hash
    else:
        return None
```

4. File Handling Algorithm

Purpose: To manage the uploading, saving, and downloading of legal documents.

Steps:

Uploading and Saving:

1. Receive the uploaded file from the user.
2. Save the file to the designated upload directory.
3. Return the file path.

Pseudocode:

```
def save_file(file):
    file_path = os.path.join(UPLOAD_FOLDER, file.filename)
    file.save(file_path)
    return file_path
```

Downloading:

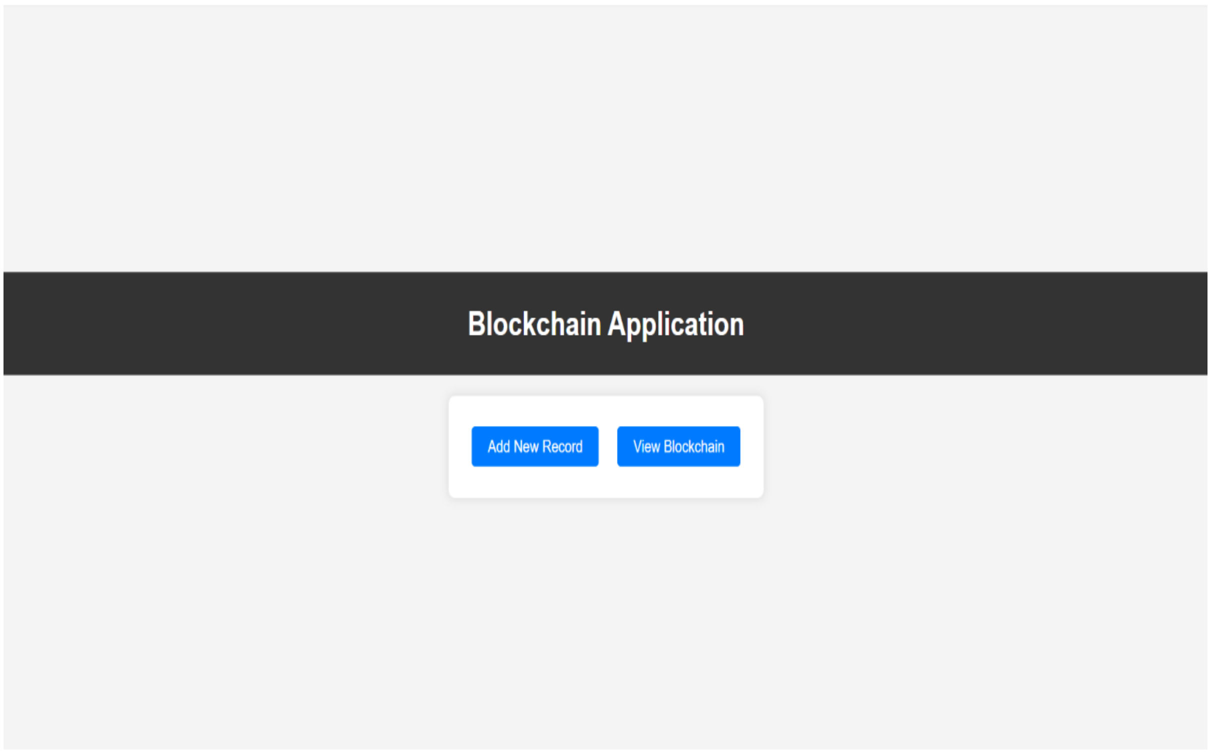
1. Find the record with the given UID.
2. Check if the file exists in the file system.
3. Send the file to the user for download.

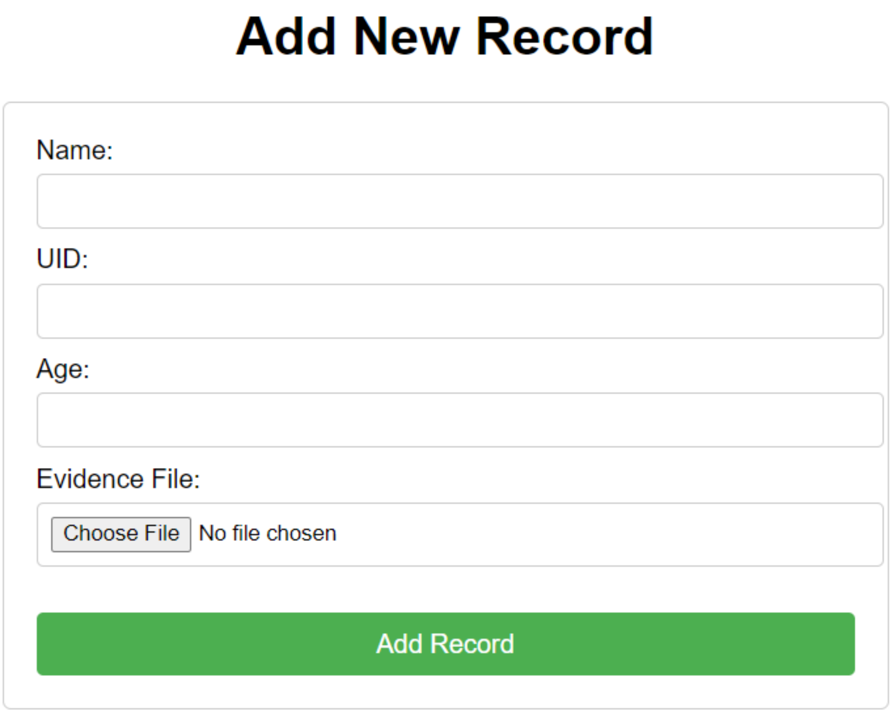
Pseudocode:

```
def download_file(uid):

    record = find_record_by_uid(uid)
    if record:
        file_path = os.path.join(UPLOAD_FOLDER, os.path.basename(record.evidence_path))
        if os.path.exists(file_path):
            return send_file(file_path, as_attachment=True)
        else:
            return 'File not found'
    else:
        return 'Record not found'
```

Interface Design:





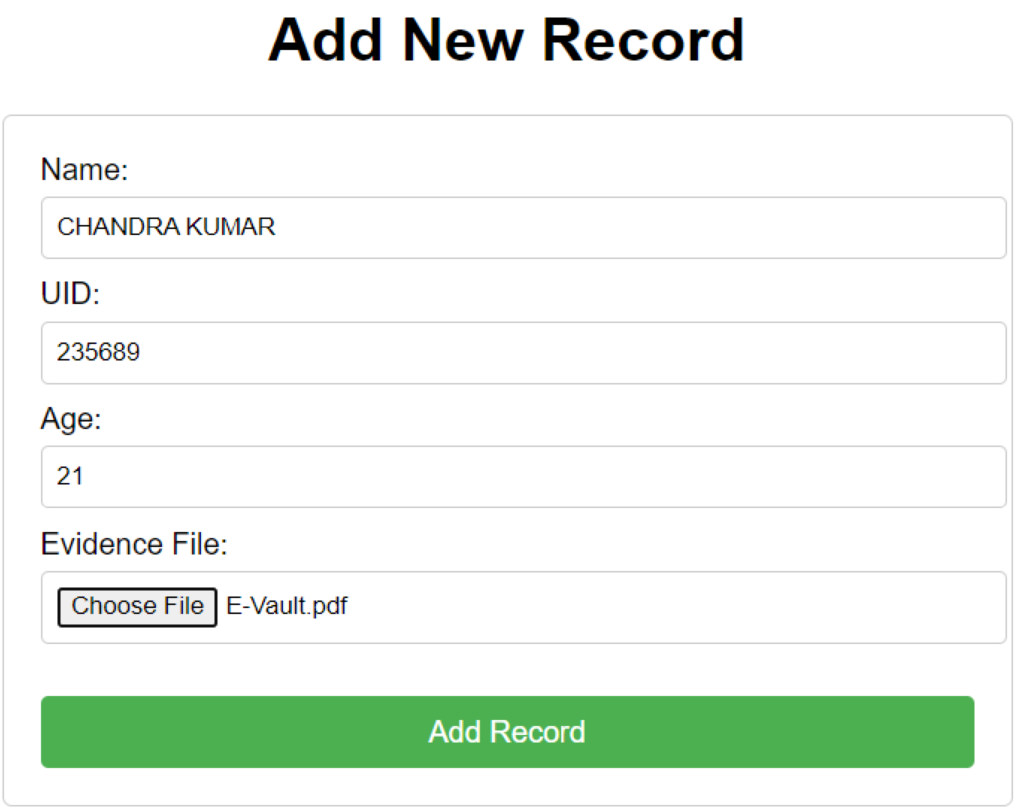
Add New Record

Name:

UID:

Age:

Evidence File:



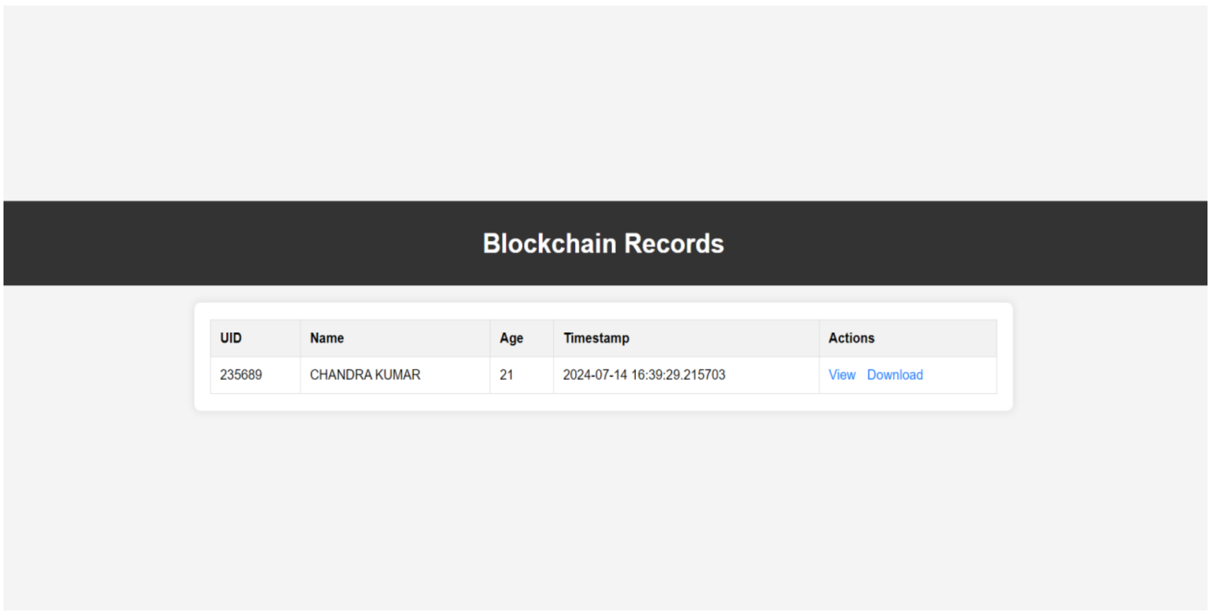
Add New Record

Name: CHANDRA KUMAR

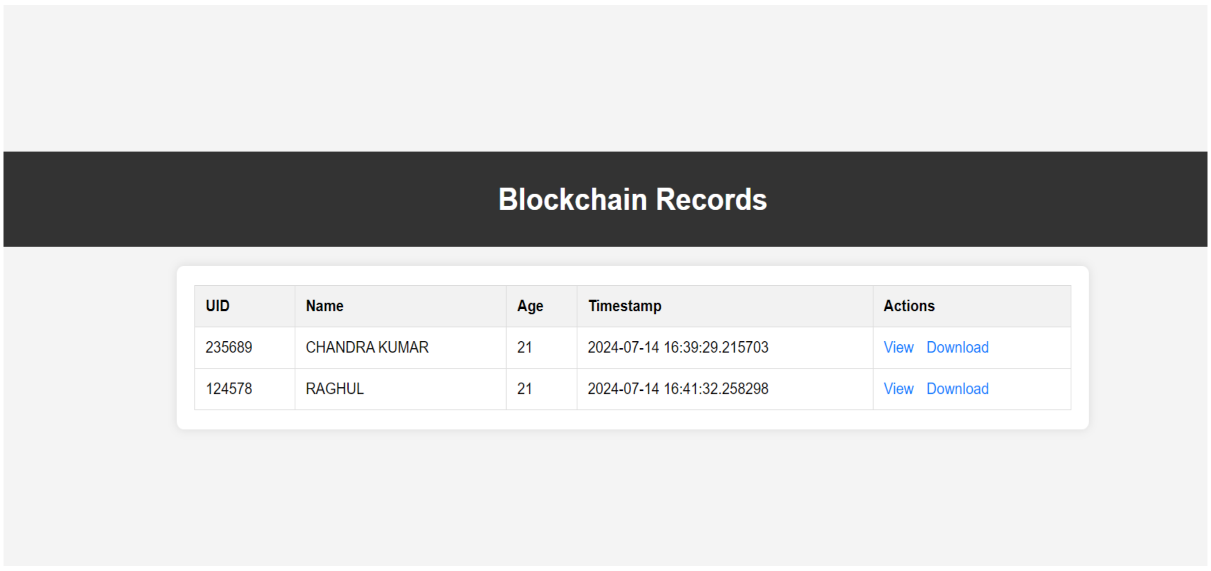
UID: 235689

Age: 21

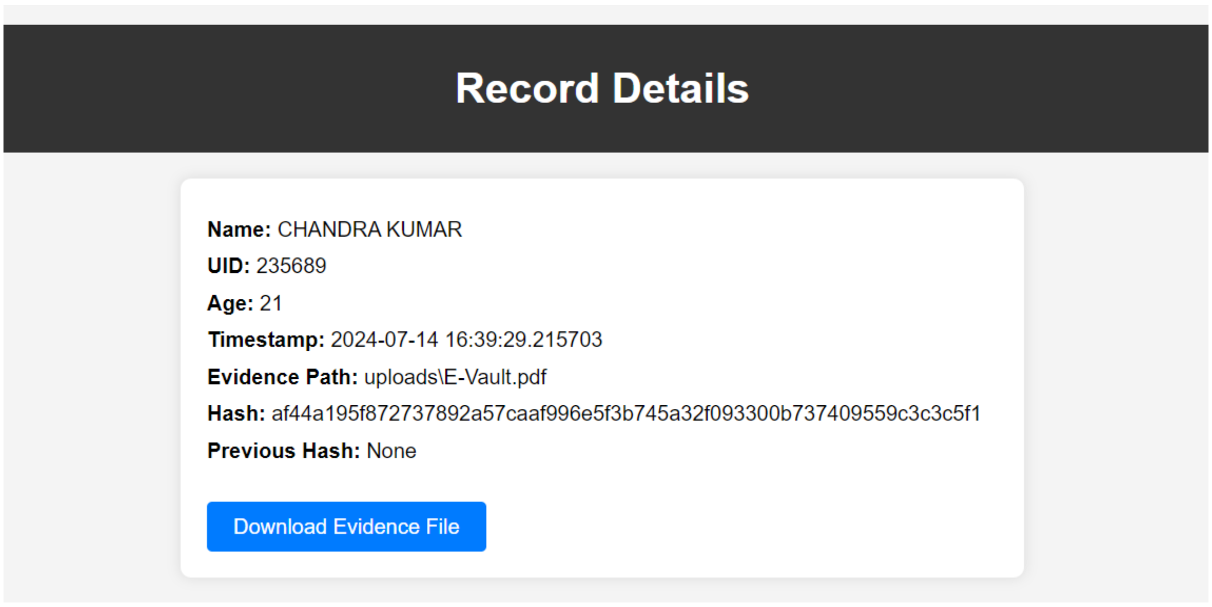
Evidence File: E-Vault.pdf


Blockchain Records

UID	Name	Age	Timestamp	Actions
235689	CHANDRA KUMAR	21	2024-07-14 16:39:29.215703	View Download


Blockchain Records

UID	Name	Age	Timestamp	Actions
235689	CHANDRA KUMAR	21	2024-07-14 16:39:29.215703	View Download
124578	RAGHUL	21	2024-07-14 16:41:32.258298	View Download


Record Details

Name: CHANDRA KUMAR
UID: 235689
Age: 21
Timestamp: 2024-07-14 16:39:29.215703
Evidence Path: uploads\E-Vault.pdf
Hash: af44a195f872737892a57caaf996e5f3b745a32f093300b737409559c3c3c5f1
Previous Hash: None

Conclusions:

In conclusion, the user interface design for the blockchain-based legal document management application successfully balances simplicity, clarity, and functionality. The intuitive layout ensures that users can easily navigate the application, add new records, and view detailed information about existing records. Key functionalities, such as uploading and downloading legal documents, are seamlessly integrated into the interface, promoting ease of use and accessibility. The structured presentation of data enhances user understanding and interaction, making the application a robust tool for managing sensitive legal documents. Overall, the design prioritizes user experience while leveraging the security and transparency of blockchain technology, resulting in an effective and user-friendly legal document management solution.