Technical Documentation -

Societe Generale Hackathon

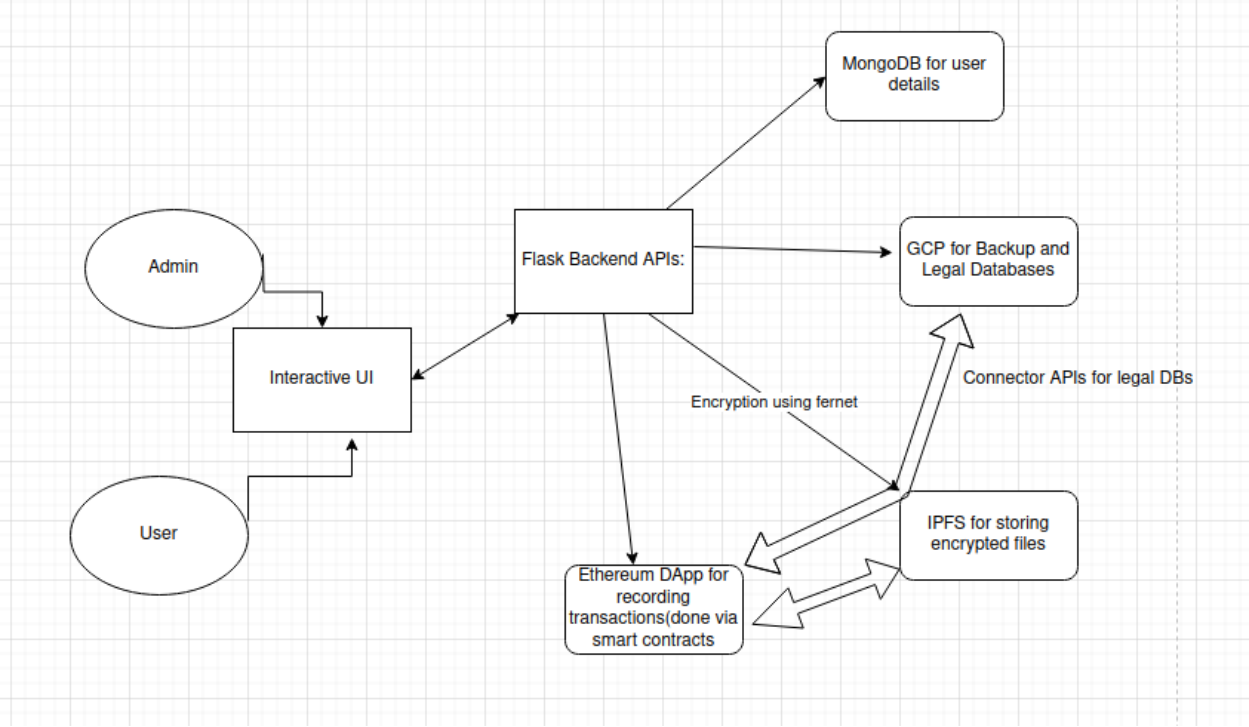
eVault Blockchain DApp Technical Documentation

Overview

The eVault DApp is a decentralized application designed to store and manage legal files securely. It leverages the Ethereum blockchain for immutability, IPFS for distributed file storage, and Flask for the backend. This system supports two types of users: Admin and regular Users. Admins have comprehensive control over the documents, including viewing, updating, adding, and generating audit logs. Regular Users can view, modify, and delete their own files. Additionally, a Google Cloud Platform (GCP) bucket is configured to serve as the legal database. Access control is enforced through smart contracts.

Architecture

1. **Frontend**: React.js
2. **Backend**: Flask (Python)
3. **Blockchain**: Ethereum (Solidity, Truffle for writing Smart contracts)
4. **File Storage**: IPFS
5. **Cloud Storage**: GCP Bucket for Legal Databases and backup
6. **Database**: GCP for Legal Database, MongoDB
7. **APIs**: Flask for backend logic, connector APIs for GCP integration
8. **Access Control** : Smart contract
9.



Components

1. Smart Contracts

Smart contracts are written in Solidity and deployed using Truffle. They handle the storage and retrieval, modification and deletion of document, ownership information, and enforce access control based on user roles.

2. Flask Backend

Flask is used to handle HTTP requests, interact with the smart contracts, and manage file operations in IPFS and GCP and MongoDB.

3. IPFS

IPFS is utilized for distributed file storage, ensuring documents are stored securely and can be accessed using unique IPFS hashes.

4. GCP Bucket

A GCP bucket is used as a centralized backup and storage solution for the legal documents, providing redundancy and additional security.

User Roles and Permissions

Admin

- **View Documents**: Access and view all documents stored in the system.
- **Update Documents**: Modify the contents of any document.
- **Add Documents**: Upload new documents to the system.
- **Delete Documents**: Remove documents from the system.
- **Backup Documents**: Backup all documents to the GCP bucket.
- **Generate Audit Logs**: Create logs of all activities performed in the system for auditing purposes.
- Import and Export any document from legal databases

User

- **View Documents**: Access and view documents they have uploaded.
- **Update Documents**: Modify the contents of their own documents.
- **Delete Documents**: Remove their own documents from the system.
- Export own data to legal databases

Unique Selling Points:

Connector APIs with legal Databases:

Connector APIs have been built using Flask to connect the IPFS data with GCP (in this context, the legal database). Users can directly directly export their data from vault to legal database and Admin can import/export data from vault.

Smart Contract - Role Based Access Control (SCRBAC):

We have Role Based Access Control enforced via blockchain.

Permission Enforcement:

- When a user attempts to perform an action (e.g., view, update, delete a document), the smart contract checks their role.
- The contract verifies whether the user’s role includes the permission to perform the requested action.
- If the user has the required permission, the action is allowed; otherwise, it is denied.
- This ensures transparency in the whole process.

Disaster Recovery and Backup:

Provisions have been made for the admin to take a full backup of the files in the vault. Mappings can be done from the MongoDB and the vault can be recovered if backup is present.

1. Admin triggers a backup operation.
2. Flask backend retrieves all documents from IPFS.
3. Documents are uploaded to the GCP bucket.
4. Backup completion status is logged.

Future work:

Run a cron job in GCP to store regular backups.

Version Control and Change Tracking System

The eVault DApp ensures transparency and accountability by tracking all changes and versions of legal records. Key features include:

Version Creation: Each document modification creates a new version stored in IPFS with a unique hash, logged on the blockchain with the timestamp and user details.

Metadata Tracking: Tracks metadata changes (e.g., title, description) along with document content, ensuring all modifications are visible.

Immutable History: Records all document versions on the blockchain, providing a complete, tamper-proof audit trail.

Security Considerations

- **Authentication**: Used secure authentication mechanisms to restrict access based on user roles.

  All Authentication is done by fernet cryptographic algorithm. It is a symmetric key alogorithm. Each user will be generated his own keys, and any document he uploads or modifies gets encrypted or decrypted using his key.

  - **Future work: Generate a master key for encrypting all keys and keep changing that master key for securing all keys**
- **Encryption**: Encrypted files before uploading to IPFS for additional security.
- **Audit Logs**: Maintain comprehensive logs for all actions to ensure traceability and accountability.
- **Backup**: Regularly backup documents to the GCP bucket to prevent data loss.

Workflow

Document Upload

1. User/Admin uploads a document through the frontend.
2. The document is stored in IPFS, and the IPFS hash is generated.
3. The IPFS hash and document metadata are sent to the Ethereum smart contract.
4. The smart contract stores the metadata and hash, associating it with the user's address.

Document Retrieval

1. User/Admin requests a document through the frontend.
2. The request is sent to the Flask backend, which interacts with the smart contract to retrieve the IPFS hash.
3. The document is fetched from IPFS using the hash and sent to the user.

Document Backup

1. Admin triggers a backup operation.
2. Flask backend retrieves all documents from IPFS.
3. Documents are uploaded to the GCP bucket.
4. Backup completion status is logged.

Deployment

1. **Smart Contracts**: Deploy using Truffle.
2. **Backend**: Flask application
3. **IPFS Node**: Set up and run an IPFS node.
4. **GCP Bucket**: Configure and secure the GCP bucket.
5. **Frontend**: Host on a web server or static site hosting service.

Future Enhancements

- **Multi-factor Authentication**: Enhance security with multi-factor authentication for Admins.
- **Scalable Architecture:** Group the API calls into different microservices and deploy using kubernetes