**ADHIYAMAAN COLLEGE OF ENGINEERING**

**(An Autonomous)**

**Dr. MGR Nagar – Hosur**

**Regulation – 2024**

**Batch 2024-2026**

**Semester – III**

**Mini Project**

**First Review**

**25th July 2025**

**TITLE:BLOCKCHAIN CLOUD DATA SECURITY USING DJANGO PYTHON**

**Dhanalakshmi . J**

**2403617662222006**

**II-MCA**

|  |  |
| --- | --- |
|  |  |

**1.PROBLEM STATEMENT**

Cloud storage platforms are widely used due to their convenience, scalability, and accessibility. However, they face critical issues related to data integrity, privacy, and security. Files stored in the cloud can be vulnerable to tampering, accidental modification, or unauthorized access—especially when there is no reliable method to verify if a file has been altered after upload.

This project proposes a solution by integrating blockchain technology with a Django-based web application. Blockchain offers a decentralized and immutable ledger, ideal for securely storing file hashes. When combined with file encryption, it ensures that:

* Each uploaded file is encrypted for confidentiality.
* A SHA-256 hash of each file is stored in a blockchain to maintain an unchangeable record.
* During file retrieval, the hash is verified to detect any tampering or data corruption.

By merging Django’s secure web framework with the trustless architecture of blockchain, this system enhances cloud data security, enforces file integrity checks, and provides users with a transparent and tamper-resistant environment for uploading and accessing their files.

**2. PROJECT OBJECTIVES (SMART)**

**Specific:**  
The project aims to build a secure file storage and verification system using a Django-based web application. Uploaded files will be encrypted using standard cryptographic techniques and their hashes stored in a custom blockchain. This ensures data confidentiality, immutability, and verifiability. Users can upload, download, and verify files while administrators manage the blockchain and monitor system integrity.

**Measurable:**

* 100% of files uploaded through the system will be encrypted using AES-256 before storage.
* Each file will generate a unique SHA-256 hash, stored in a blockchain block.
* Every download request will include an integrity check against the stored hash.
* Admin dashboard will log and display all blockchain entries, user uploads, and verification results.
* Performance will be tracked: each encryption and hash operation should complete within 3–5 seconds.

**Achievable:**  
The system leverages Django, a mature Python framework, along with Python’s built-in and third-party libraries such as:

* hashlib (for SHA-256 hashing),
* cryptography or pycryptodome (for AES encryption),
* datetime and uuid (for timestamping and file identification),
* SQLite or PostgreSQL for database management.

**Relevant:**

* Verifying file integrity,
* Preventing unauthorized modifications,
* Creating tamper-proof logs for audits and compliance.  
  This project is particularly relevant for sectors like healthcare, legal services, and finance where file authenticity and traceability are crucial.

**Time-bound:**  
The project is scheduled for completion within 12 weeks, with clearly defined milestones:

* Week 1–2: Requirement gathering, plannings.
* Week 3–4: User registration/login system and file upload module.
* Week 5–6: Encryption and hashing logic implementation.
* Week 7–8: Blockchain structure and chaining logic.
* Week 9–10: File verification module and admin dashboard.
* Week 11: Testing and bug fixes.
* Week 12: Final documentation and review presentation.

**3. WHAT WILL THIS PROJECT ACHIEVE?**

**End-to-End Encrypted File Storage and Retrieval:**  
All user-uploaded files will be encrypted using AES encryption before being stored in the backend. This ensures that sensitive data remains confidential and unreadable even if unauthorized access to the file system occurs. When a file is downloaded, it is decrypted securely and returned to the authorized user.

**Blockchain Structure for File Hash Storage:**  
Each uploaded file will be hashed using a cryptographic algorithm (SHA-256), and this hash will be recorded in a blockchain data structure. Every new hash is stored in a new block, which includes the previous block’s hash—creating a tamper-evident and immutable record of all uploaded files. This prevents undetected changes to data and preserves its authenticity.

**Hash Verification for Integrity Checks:**  
When a user requests a file download, the system recalculates the hash of the current file and compares it with the hash stored in the blockchain. If they match, the file is verified as original. If not, the system alerts the user about potential tampering or corruption. This ensures full trust in the file’s integrity and origin.

**Admin Dashboard for Monitoring and Control:**  
A secure admin interface will allow system administrators to:

* View all registered users and their upload activity.
* Monitor blockchain entries (file hashes and timestamps).
* Detect tampering through mismatch logs.
* Ensure system health and integrity through real-time status tracking.

**Audit Trail and Transparency:**  
Every file transaction is time-stamped and traceable through blockchain blocks, ensuring a clear audit trail. This makes the system suitable for industries where data integrity and auditability are essential, such as legal, financial, or medical record systems.

**User-Friendly Web Interface:**  
The application will include a responsive, intuitive UI built with Bootstrap and Django templates. Users can easily register, log in, upload files, check their status, and download verified content with integrity assurance—all through a simple web interface.

**Modular and Scalable Design:**  
The architecture will be modular, allowing for future enhancements such as multi-user roles, distributed blockchain nodes, cloud integrations (like AWS S3), and real-time alerts for file verification failures.

**4. SOFTWARE REQUIREMENTS SPECIFICATION (SRS)**

#### A. Functional Requirements

**1. User Authentication and Authorization**  
 - The system shall allow users to register with email, password, and role (admin/user).  
 - The system shall authenticate users at login and redirect based on their role.

**2.  File Upload**  
 - The system shall allow authenticated users to upload files.  
 - The system shall encrypt the file before storing it on the server.  
 - The system shall generate a SHA-256 hash of the encrypted file.

**3.  Blockchain Entry Creation**  
 - The system shall create a new block for each uploaded file.  
 - The block shall include file hash, previous block hash, and timestamp.  
 - The blockchain shall be immutable once blocks are added.

**4. File Download and Integrity Check**  
 - The system shall allow users to download their files.  
 - The system shall re-calculate the file's hash upon download.  
 - The system shall compare the new hash with the original stored in the blockchain.  
 - The system shall alert the user if a hash mismatch is detected.

**5. Admin Dashboard**  
 - The system shall allow admins to view all registered users.  
 - The system shall allow admins to view uploaded files and associated blockchain entries.  
 - The system shall present file integrity logs for auditing purposes.

**6. Audit Trail and Logging**  
 - The system shall log all file uploads, downloads, and verifications.  
 - The system shall provide time-stamped entries for each blockchain block and user action.

#### B. NON-FUNCTIONAL REQUIREMENTS

These define the quality attributes and constraints of the system’s performance and behavior.

**1. Performance**  
 - File encryption, hashing, and blockchain addition shall be completed within 5 seconds for files under 10MB.  
 - The dashboard shall load within 2 seconds under normal load.

**2. Security**  
 - Files shall be encrypted using AES-256 before being stored.  
 - Passwords shall be stored securely using bcrypt or PBKDF2 hashing.  
 - Access to dashboard and file data shall be restricted based on user roles.  
 - All communication shall use HTTPS in production.

**3. Usability**  
 - The UI shall be responsive and simple, using Bootstrap for consistent styling.  
 - Clear status messages and tooltips shall guide users through uploads and verification.

**4. Reliability**  
 - Blockchain structure shall preserve data integrity by preventing retroactive changes.  
 - If a hash mismatch occurs, the system shall flag it and prevent download.

**5. Scalability**  
 - The architecture shall support increasing numbers of users and files.  
 - Blockchain and database components shall be modular and easily upgradable.

**6. Maintainability**  
 - Code shall follow Django’s modular app structure.  
 - Components like file handling, encryption, and blockchain logic shall be separated for ease of updates.  
 - Proper documentation and in-code comments shall be maintained.

#### C. USER CHARACTERISTICS

**1.  End Users (Registered Users)**  
 - Basic computer literacy  
 - Able to upload/download files  
 - Rely on system to verify file authenticity

**2.  Admin Users**  
 - Have access to all user and file data  
 - Perform system monitoring and review blockchain logs  
 - Moderate the platform for misuse and manage file integrity

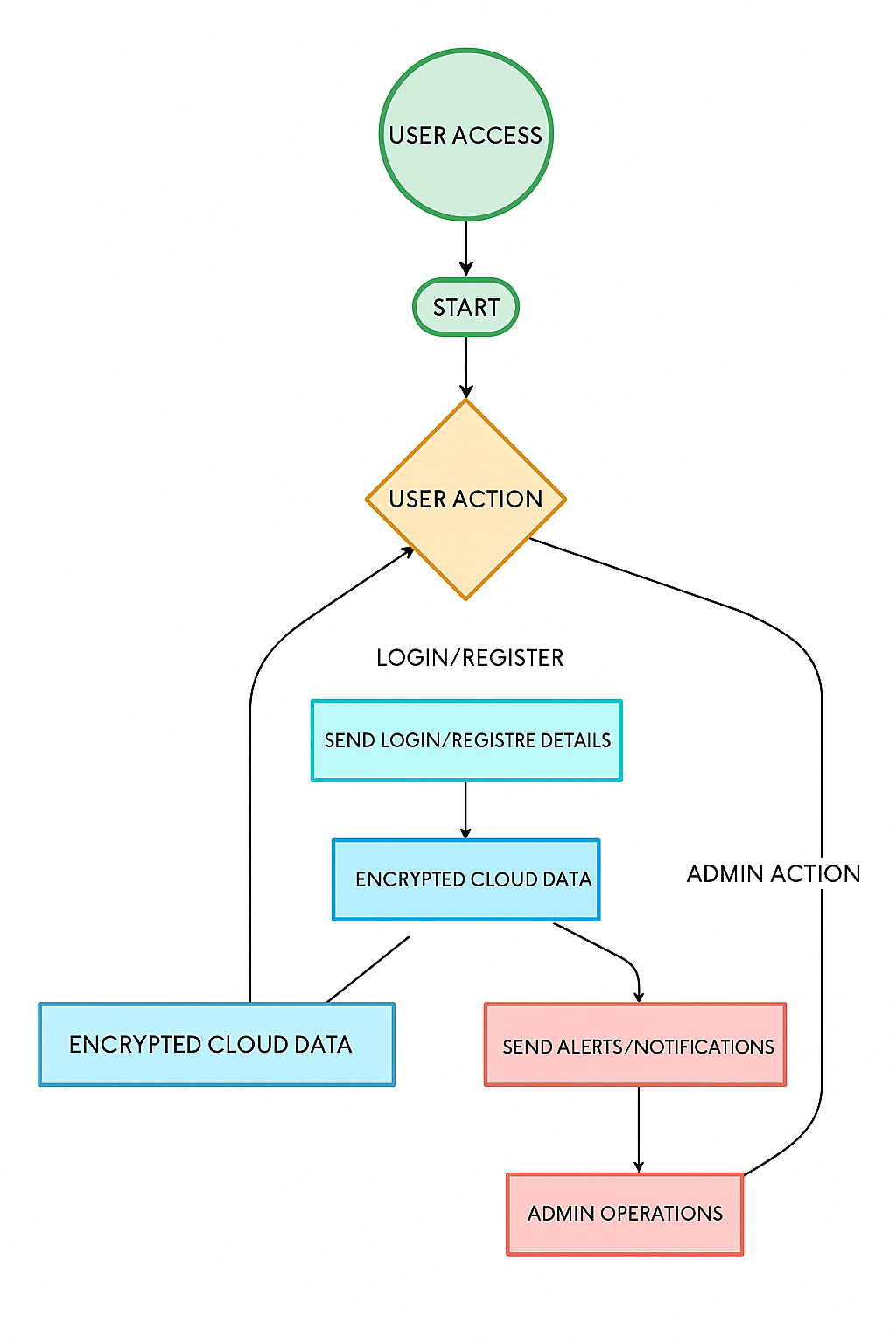
#### D. OPERATING ENVIRONMENT

**1.  Hardware Requirements**  
 - Client: Any device with a modern web browser (Chrome, Firefox, Edge)  
 - Server: Minimum 4GB RAM, 2-core CPU, 50GB SSD for file storage and database

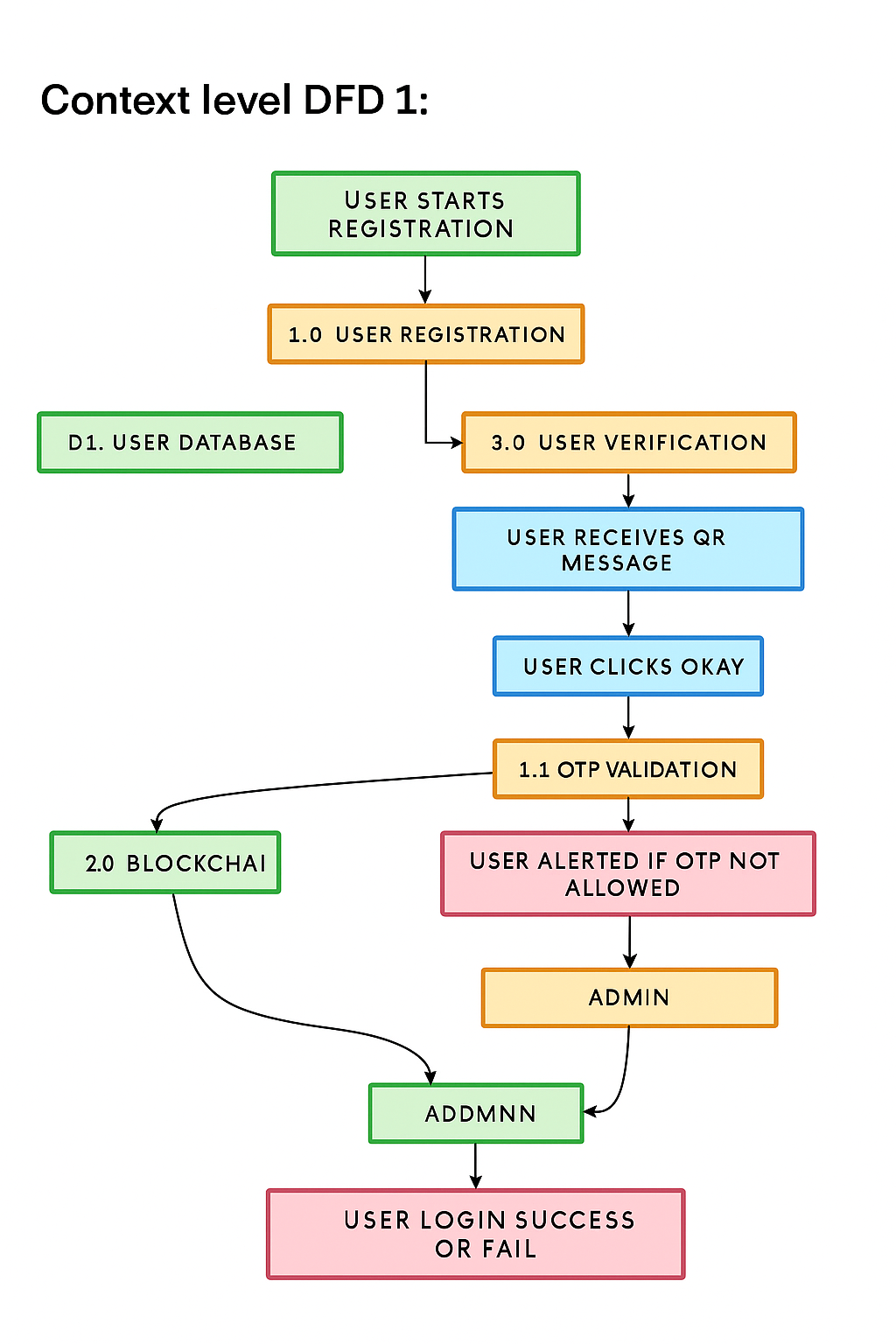
**2.  Software Requirements** - Backend: Python 3.10+, Django 4.x  
 - Frontend: HTML5, CSS3, Bootstrap 5  
 - Database: SQLite (local) or PostgreSQL (production)  
 - OS: Linux (Ubuntu 20.04+) or Windows 10+  
 - Libraries:  
  - cryptography (for AES encryption)  
  - hashlib (for hashing)  
  - datetime, uuid (for block metadata)

**5.DATA FLOW DIAGRAM:**

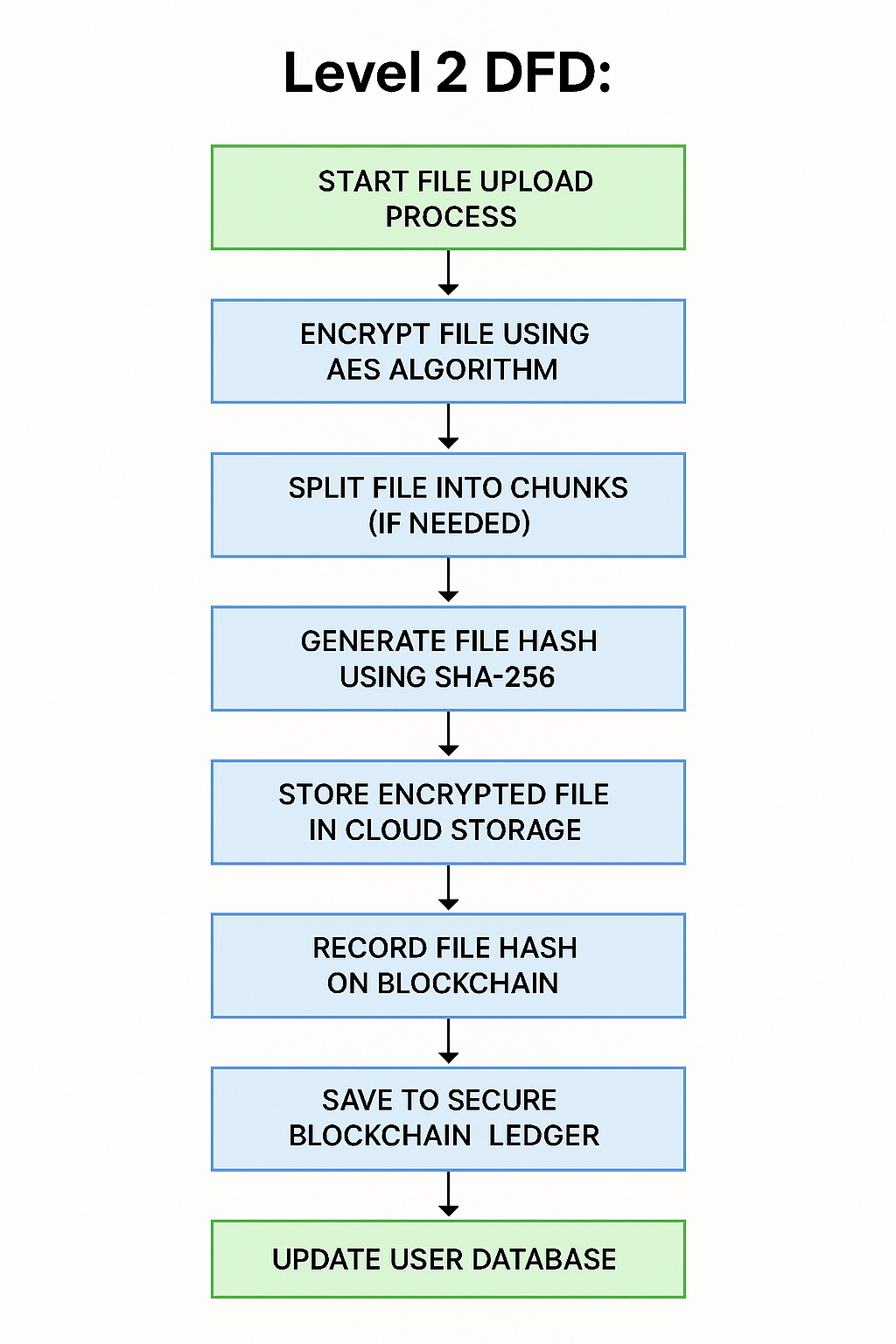
**Level 0 DFD:**

****

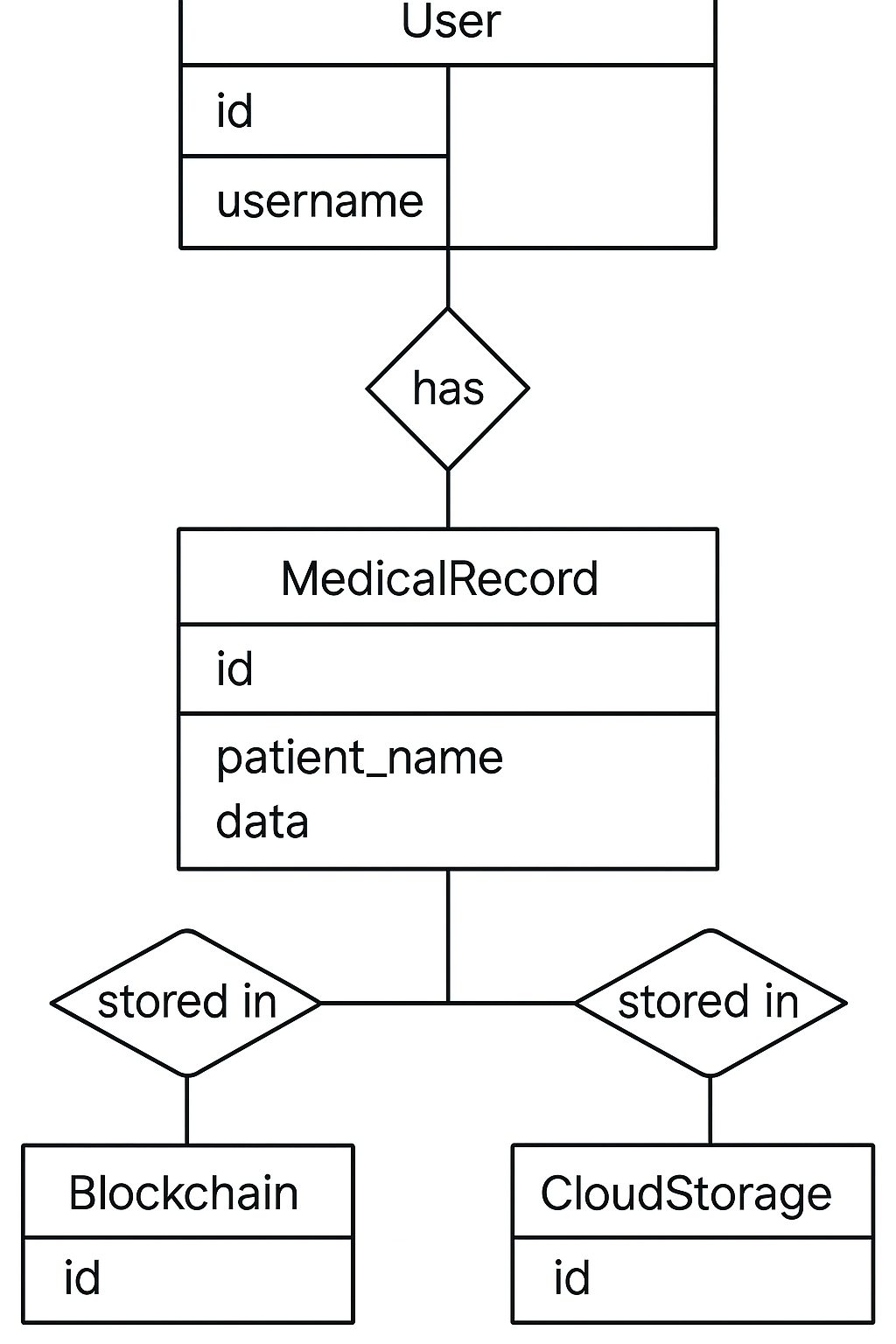
**Level 1 DFD:**

****

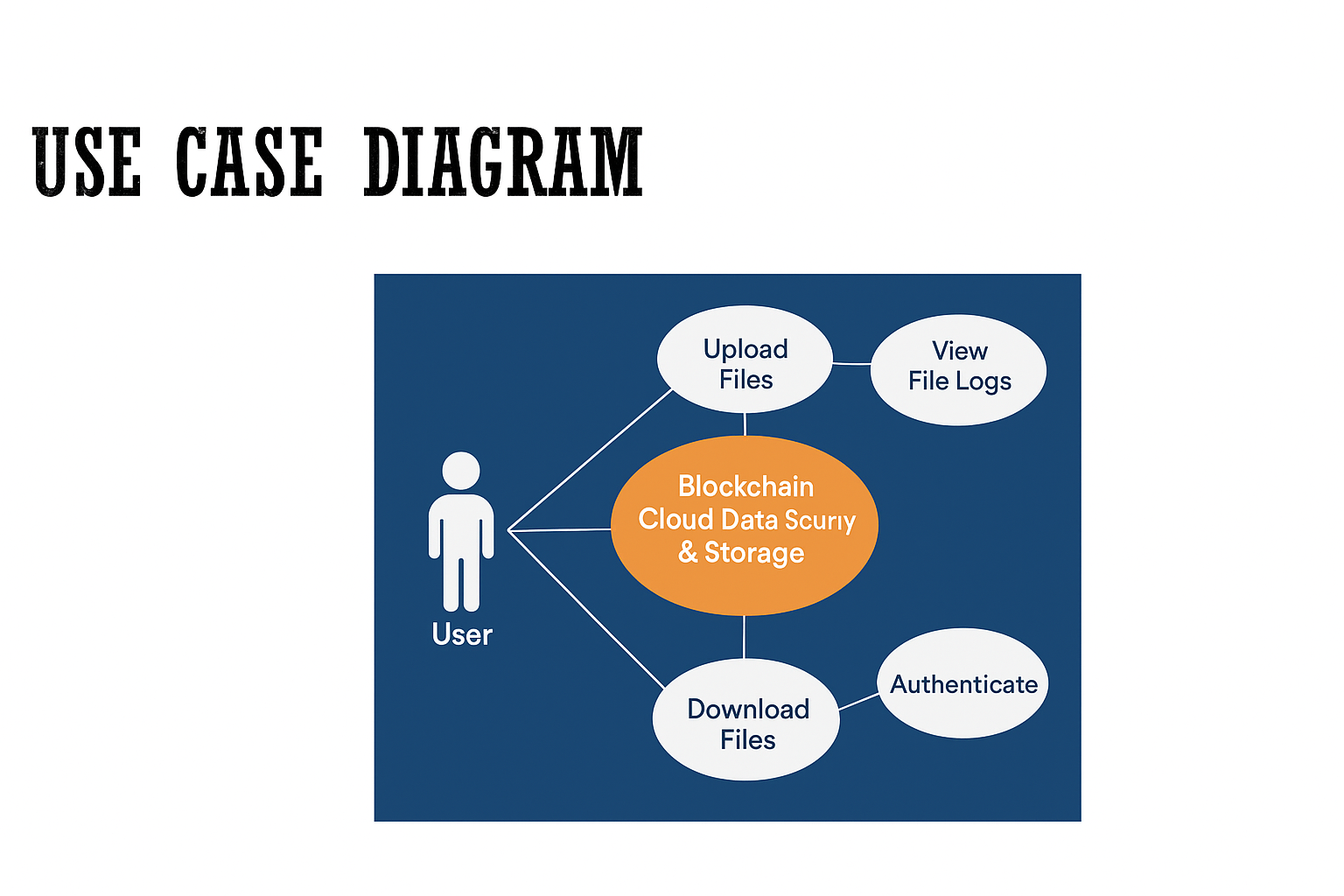
**Level 2 DFD:**

****

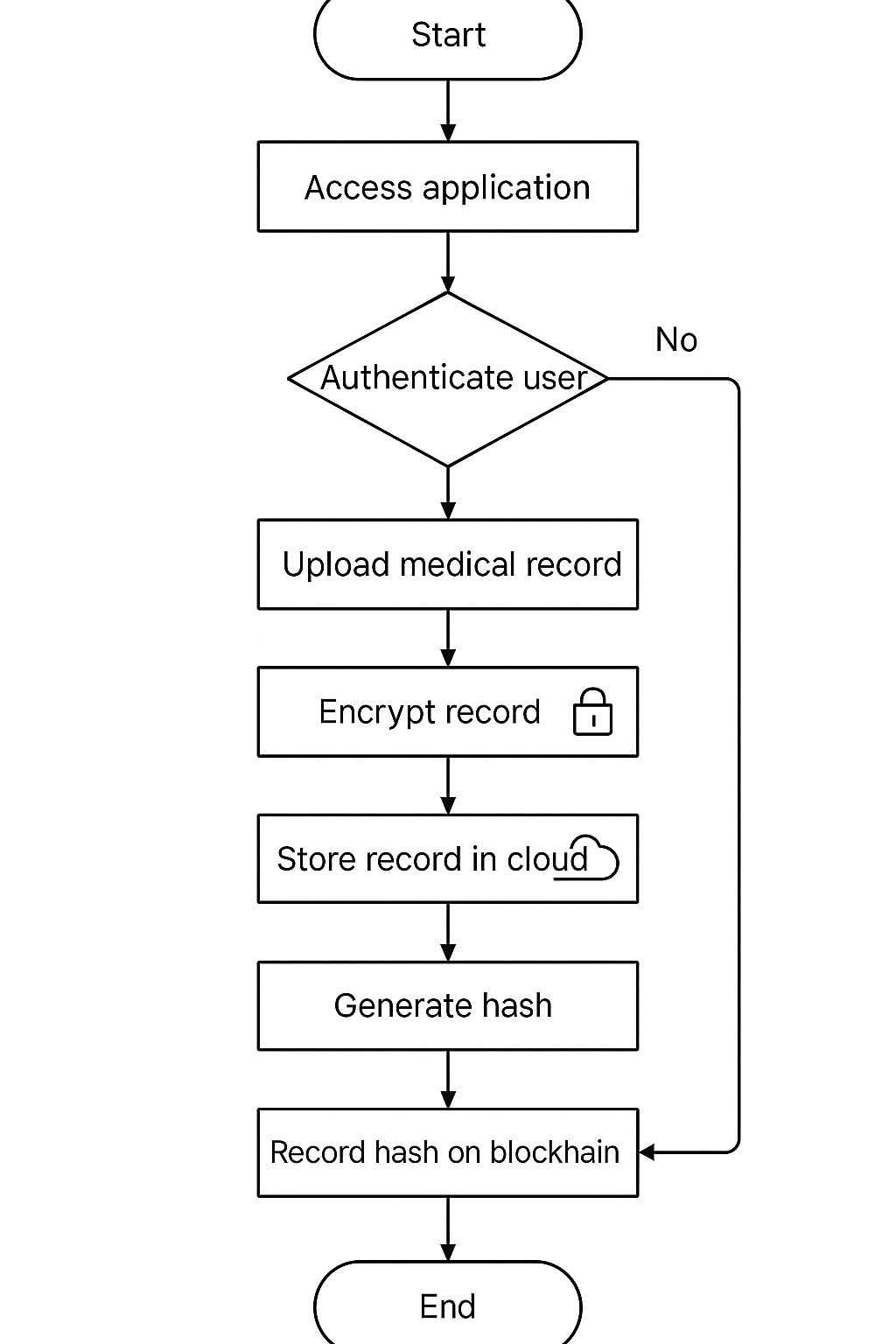
**ER-DIAGRAM:**

****

**USE CASE DIAGRAM:**

****

**ACTIVITY DIAGRAM:**

****

|  |  |
| --- | --- |
|  |  |

**CLASS DIAGRAM:**

**DATABASE DESIGN:**

**1.USER:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Size** | **Description** | **Constraints** |
| Id | Integer | – | Primary key (auto increment) | Primary Key, Auto-Inc |
| Name | Varchar | 100 | Full name of the user | Not Null |
| Email | Varchar | 100 | Unique email address | Unique, Not Null |
| Password | Varchar | 255 | Hashed password | Not Null |
| Role | Enum | – | User role: 'admin' or 'user' | Default: 'user' |
| created\_at | Datetime | – | Registration timestamp | Default: now() |

**2.FILES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Size** | **Description** | **Constraints** |
| file\_id | Integer | – | Primary key for the file | Primary Key, Auto-Inc |
| filename | Varchar | 255 | Original name of uploaded file | Not Null |
| file\_path | Text | – | Server path to encrypted file | Not Null |
| encrypted\_hash | Varchar | 64 | SHA-256 hash of encrypted file | Not Null, Unique |
| user\_id | Integer | – | Foreign key to users table | Foreign Key(users.id) |
| uploaded\_at | Datetime | – | Timestamp of file upload | Default: now() |
| is\_verified | Boolean | – | Status of integrity check | Default: true |

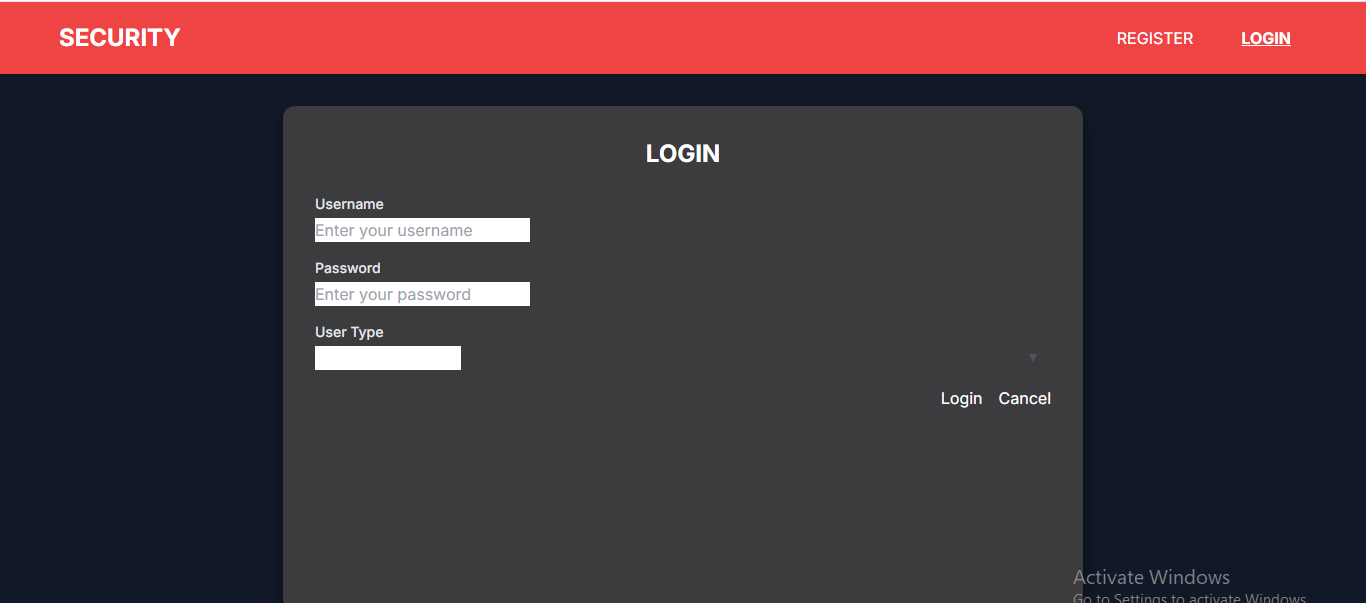
**3.BLOCKCHAIN**

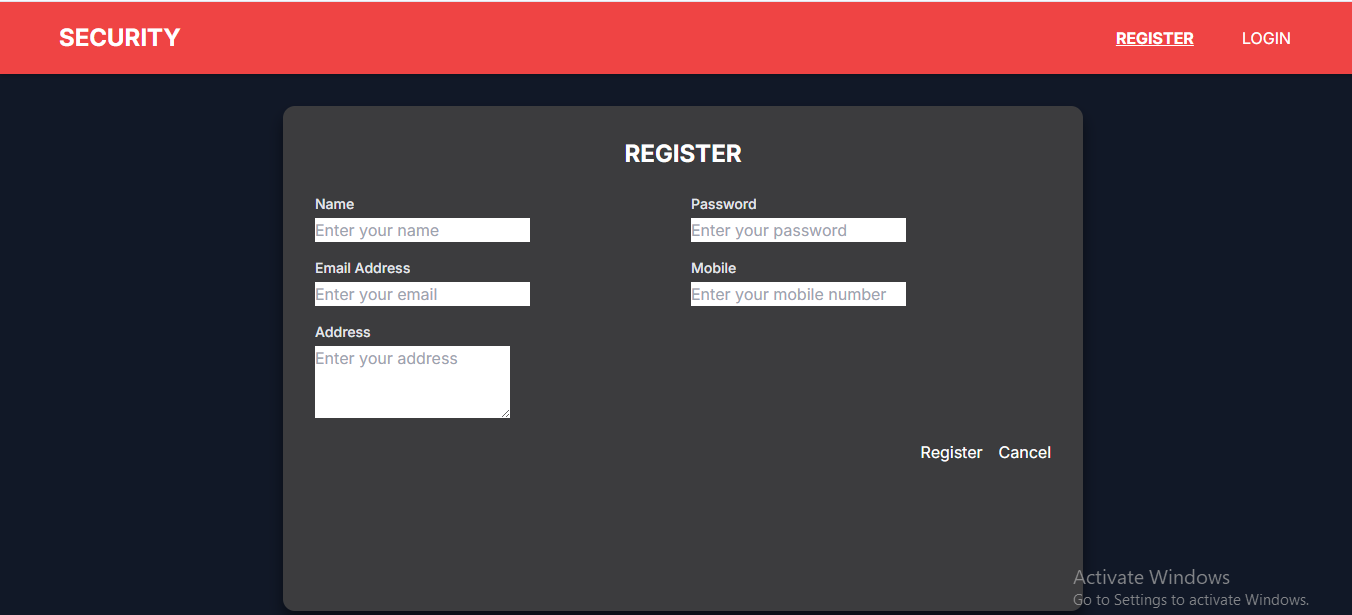
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Size** | **Description** | **Constraints** |
| block\_id | Integer | – | Unique identifier for each block | Primary Key, Auto-Inc |
| file\_id | Integer | – | Reference to file in the system | Foreign Key(files.file\_id) |
| hash | Varchar | 64 | Current SHA-256 hash | Not Null, Unique |
| previous\_hash | Varchar | 64 | Previous block's hash (chain link) | Nullable (for first block) |
| timestamp | Datetime | – | Time of block creation | Default: now() |
| nonce | Integer | – | Proof-of-Work value (if applied) | Default: 0 |

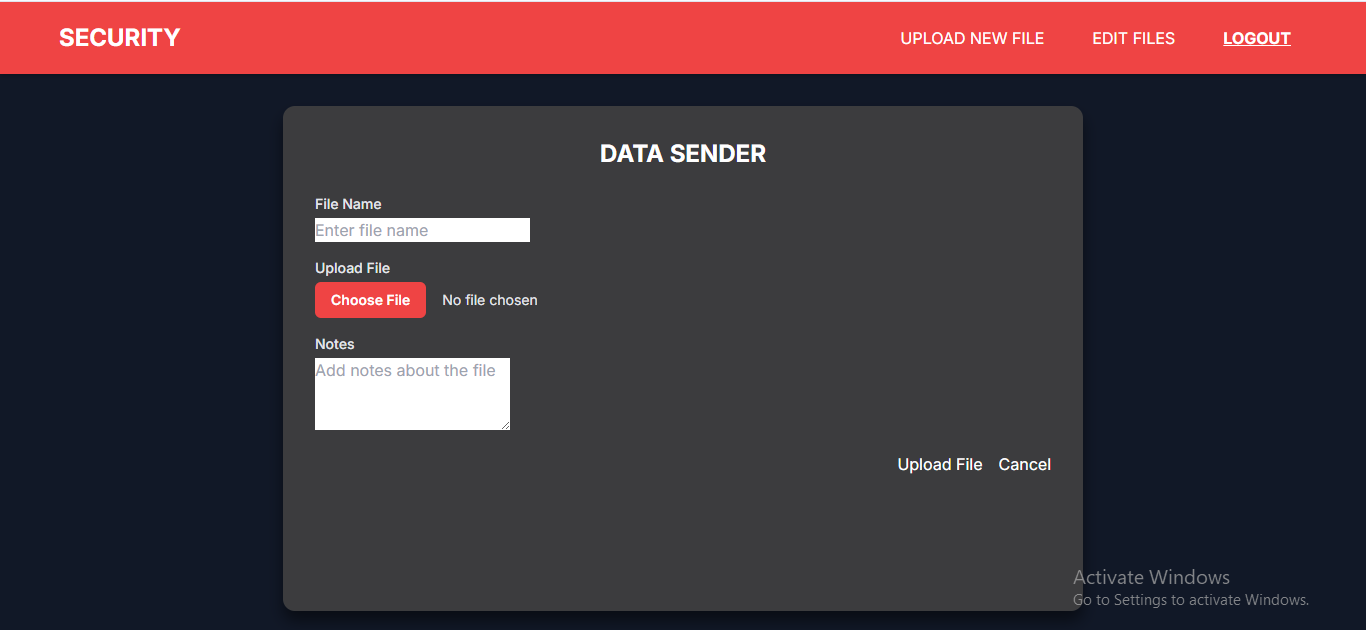
**4.LOG**

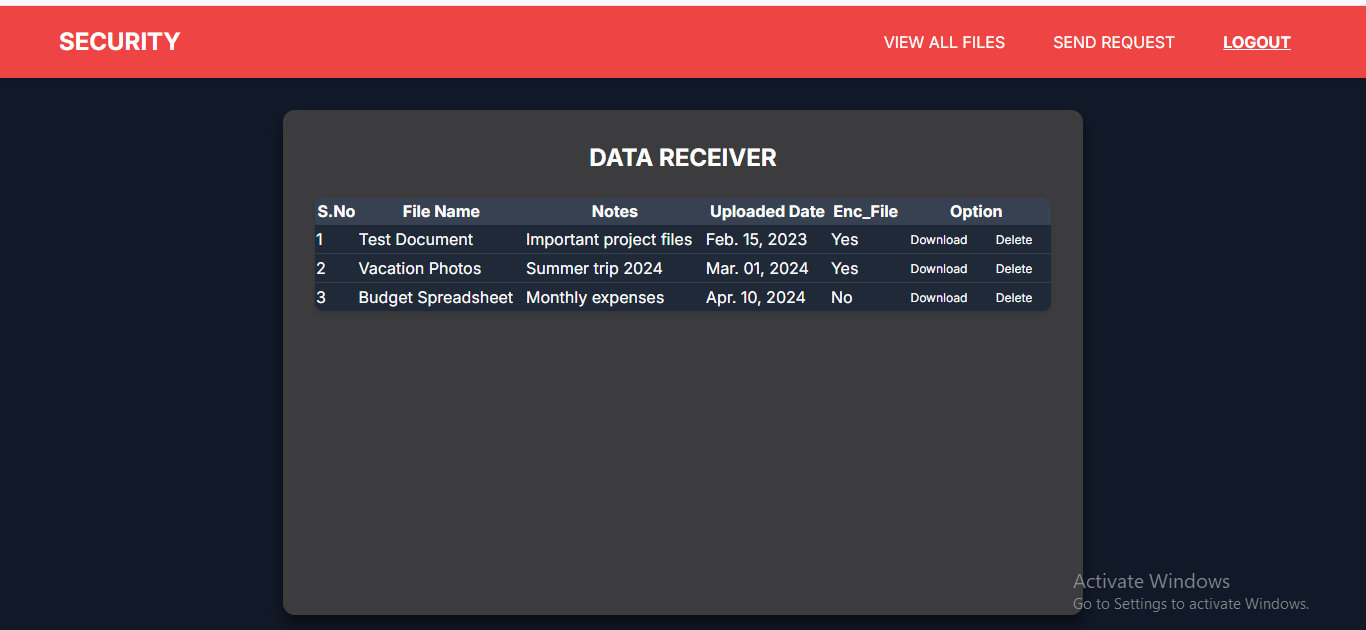
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Size** | **Description** | **Constraints** |
| log\_id | Integer | – | Unique ID of each log entry | Primary Key, Auto-Inc |
| user\_id | Integer | – | Reference to user | Foreign Key(users.id) |
| action | Varchar | 100 | Action performed (upload, verify, etc) | Not Null |
| file\_id | Integer | – | Related file ID (nullable) | Foreign Key(files.file\_id) |
| timestamp | Datetime | – | Date and time of action | Default: now() |

**USER INTERFACE (UI) REQUIREMENTS / MOCKUPS (PRELIMINARY):**

****

****

****

****