Project Report: Time-Bound Digital Access Vault

1. Project Overview

The **Time-Bound Digital Access Vault** is a full-stack web application designed to allow users to securely store sensitive text data and share it through temporary, rule-governed access links. The system ensures that access constraints are strictly enforced on the backend, making it a reliable "Single Source of Truth".

Core Functionalities:

- **Secure Vault Creation:** Authenticated users can create items with titles and sensitive content.
- **Rule-Based Sharing:** Generation of links with specific expiration times, maximum view limits, and optional passwords.
- **Automated Enforcement:** Instant revocation of access once a link expires or the view limit is reached.
- **Audit Trail:** Detailed logging of every successful and denied access attempt, including IP addresses.

---

2. System Architecture

The application follows a classic Client-Server architecture using a RESTful API design.

Technical Stack:

- **Backend:** Python with **Flask** framework.
- **Database: SQLAlchemy (ORM)** with SQLite for persistent storage.
- **Security: Flask-Bcrypt** for password hashing and **Flask-JWT-Extended** for token-based authentication.
- **Frontend:** HTML5 and Vanilla JavaScript using the Fetch API for asynchronous communication.

---

3. Detailed Design & Implementation

A. Database Schema (`models.py`)

The system utilizes four primary relational tables:

1. **User:** Stores email and hashed passwords for authentication.
2. **VaultItem:** Stores the sensitive text, title, and a foreign key to the owner.
3. **ShareLink:** Contains the rules for access, including `expires_at`, `remaining_views`, and a hashed `password`.
4. **AccessLog:** An immutable record of every access attempt with status (allowed/denied) and source IP.

B. Access Validation Logic (`app.py`)

The backend performs a triple-check validation on every access request:

1. **Temporal Validation:** Checks if current UTC time is within the `expires_at` window.
2. **Quantitative Validation:** Checks if `remaining_views` is greater than zero.
3. **Credential Validation:** Verifies the provided password against the stored hash if a password was configured.

---

4. Security Considerations

- **Preventing Unauthorized Access:** All vault management routes are protected by JWT tokens. Without a valid "Bearer" token, the backend returns a 401 Unauthorized status.
- **Link Brute-Forcing:** Share links use **UUID4** (128-bit random strings) instead of incremental IDs, making it computationally impossible to guess active links.
- **Data Integrity:** Passwords (both for users and shared links) are never stored in plain text; they are hashed using the Bcrypt algorithm.
- **Race Conditions:** The backend updates view counts and active status within a single database transaction to handle simultaneous access attempts safely.

5. Edge Case Handling

- **Server Restarts:** Since the system uses a persistent SQLite database, all vault items and active links remain valid after a server reboot.
- **Link Exhaustion:** The moment `remaining_views` hits zero, the `active` flag is set to `False`, ensuring the link can never be reused.
- **Empty/Invalid Input:** The frontend and backend both handle empty strings or non-numeric view counts to prevent server crashes.