WhistleVault

The Blind Ledgers

April , 2025

Team Information

Team Name: The Blind Ledgers

CS23B1047 Dhage Pratik Bhishmacharya
CS23B1055 Yadynesh D Sonale
CS23B1069 Ala Pavan Sai teja

Contents

1 Introduction **2**

2 System Architecture **2**

3 Technologies Used **2**

4 Database Design **3**
4.1 Table Structure 3
4.2 Relationships 4
4.3 Normalization 5
4.4 ER Diagram 5

5 Code Structure **6**

6 Installation and Setup **6**
6.1 Prerequisites 6
6.2 Local Installation 6
6.3 PythonAnywhere Deployment 7

7 Functionality **7**
7.1 User Features 7
7.2 Admin Features 8
7.3 Sorting Feature 8

8 Usage Instructions **8**
8.1 User Workflow 8
8.2 Admin Workflow 8

9 Future Improvements **8**

10 Technical Details **9**
10.1 Error Handling 9
10.2 Encryption Implementation 9

11 References **9**

1 Introduction

WhistleVault is a sophisticated web-based application developed to provide a secure platform for users to store, manage, and share sensitive information such as leaks, confidential data, or other private data. This project was undertaken as part of a course with a focus on integrating practical software development with database management system (DBMS) concepts. Leveraging a combination of Flask for the web framework, SQLite as the relational database, and the cryptography library for data security, WhistleVault offers a robust solution tailored to individual and organizational needs. The application supports features like user authentication, encrypted secret storage, categorization, photo attachments, and a sharing mechanism, making it a versatile tool for secure information management.

The primary objective of this documentation is to provide a comprehensive guide to WhistleVault's design, implementation, and usage. It explores the system's architecture, delves into the database structure with a slight emphasis on relational design principles, outlines the installation and deployment processes, provides detailed usage instructions, and proposes future enhancements.

2 System Architecture

WhistleVault is structured around the Model-View-Controller (MVC) architectural pattern, which separates concerns to enhance maintainability and scalability:

- **Model**: The data layer is powered by SQLite, a lightweight yet powerful relational database management system. It stores and manages entities such as users, secrets, categories, photos, shares, and view logs, ensuring efficient data retrieval and storage.
- **View**: The user interface is rendered using Jinja2 templates, which dynamically generate HTML pages (e.g., `view_all.html`, `submit.html`). These templates are designed to be responsive and user-friendly, providing an intuitive experience across devices.
- **Controller**: The logic layer, implemented in `app.py`, handles request routing, business logic, database interactions, and encryption/decryption processes using the Fernet symmetric encryption algorithm. This layer acts as the intermediary between the model and view, processing user inputs and updating the database accordingly.

The system also incorporates a static directory for storing uploaded photos, ensuring that multimedia content is securely managed alongside textual data. Encryption is a critical feature, protecting sensitive information from unauthorized access both at rest and during transmission.

3 Technologies Used

WhistleVault leverages a variety of technologies to ensure functionality, security, and scalability:

- **Flask**: A micro web framework written in Python, used for building the web application with a lightweight and modular structure.
- **SQLite**: A lightweight, serverless database engine that provides a reliable storage solution for the application's data.
- **Cryptography**: The Python cryptography library, specifically the Fernet module, is used for symmetric encryption to secure sensitive data.
- **Jinja2**: A templating engine for Python, utilized to render dynamic HTML pages with a clean and maintainable syntax.
- **Flask-Bcrypt**: A Flask extension that provides secure password hashing using the bcrypt algorithm to protect user credentials.
- **HTML/CSS**: Used for the frontend interface, ensuring a responsive and user-friendly design.
- **Git**: Version control system for managing code changes and collaboration among team members.

4 Database Design

The database serves as the backbone of WhistleVault, designed with relational principles to ensure data organization and integrity, with a slight nod to DBMS normalization techniques.

4.1 Table Structure

The database comprises six key tables, each with specific attributes and constraints:

- **Users**
  - `user_id` (INTEGER, PRIMARY KEY, AUTOINCREMENT): Unique identifier for each user.
  - `username` (VARCHAR(50), UNIQUE, NOT NULL): User's chosen username.
  - `email` (VARCHAR(100), UNIQUE, NOT NULL): User's email address.
  - `hashed_password` (TEXT, NOT NULL): Securely stored password hash.
  - `profile_picture` (VARCHAR(255)): Path to the user's profile image.
  - `is_admin` (INTEGER, DEFAULT 0): Flag indicating admin status.
  - `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP): Account creation timestamp.
- **Categories**
  - `category_id` (INTEGER, PRIMARY KEY, AUTOINCREMENT): Unique category identifier.
  - `name` (VARCHAR(50), UNIQUE, NOT NULL): Category name.

  - `is_custom` (INTEGER, DEFAULT 0): Indicates if the category is user-defined.

- **Secrets**
  - `secret_id` (INTEGER, PRIMARY KEY, AUTOINCREMENT): Unique secret identifier.
  - `user_id` (INTEGER, FOREIGN KEY to Users, NOT NULL): Owner of the secret.
  - `secret_text` (BLOB): Encrypted secret content.
  - `title` (VARCHAR(100), NOT NULL): Secret title.
  - `category_id` (INTEGER, FOREIGN KEY to Categories, NOT NULL): Associated category.
  - `priority` (TEXT, CHECK: 'Low', 'Medium', 'High', DEFAULT 'Medium'): Priority level.
  - `custom_category` (VARCHAR(50)): User-defined category if applicable.
  - `upload_date` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP): Date of submission.
  - `views` (INTEGER, DEFAULT 0): Number of times viewed.

- **Secret_Photos**
  - `photo_id` (INTEGER, PRIMARY KEY, AUTOINCREMENT): Unique photo identifier.
  - `secret_id` (INTEGER, FOREIGN KEY to Secrets, NOT NULL): Associated secret.
  - `photo_path` (VARCHAR(255), NOT NULL): File path of the photo.
  - `upload_date` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP): Upload timestamp.

- **Secret_Logs**
  - `log_id` (INTEGER, PRIMARY KEY, AUTOINCREMENT): Unique log identifier.
  - `secret_id` (INTEGER, FOREIGN KEY to Secrets, NOT NULL): Viewed secret.
  - `user_id` (INTEGER, FOREIGN KEY to Users): Viewing user.
  - `view_date` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP): View timestamp.

4.2 Relationships

The database features the following relationships:

- **Users to Secrets (1:N)**: A user can own multiple secrets, with `user_id` as the foreign key (ON DELETE SET NULL to allow orphaned secrets).

- **Categories to Secrets (1:N)**: A category can be assigned to multiple secrets, linked by `category_id`.
- **Secrets to Secret_Photos (1:N)**: A secret can have multiple photos, with `secret_id` (ON DELETE CASCADE to remove photos with secrets).
- **Secrets to Secret_Logs (1:N)**: A secret can have multiple view logs, linked by `secret_id` (ON DELETE CASCADE).
- **Users to Secret_Logs (N:1)**: Many users can view a secret; each log (if tracked) links to one user.

4.3 Normalization

- **1NF**: All attributes are atomic (e.g., no multivalued attributes; photos are in `Secret_Photos`).
- **2NF**: No partial dependency (all non-key attributes depend on the entire primary key).
- **3NF**: No transitive dependencies (e.g., `custom_category` is part of `Secrets`, not a separate table).

Thus, the database design achieves third normal form (3NF).

4.4 ER Diagram

A visual representation of the entity-relationship (ER) diagram would enhance understanding of the database structure:

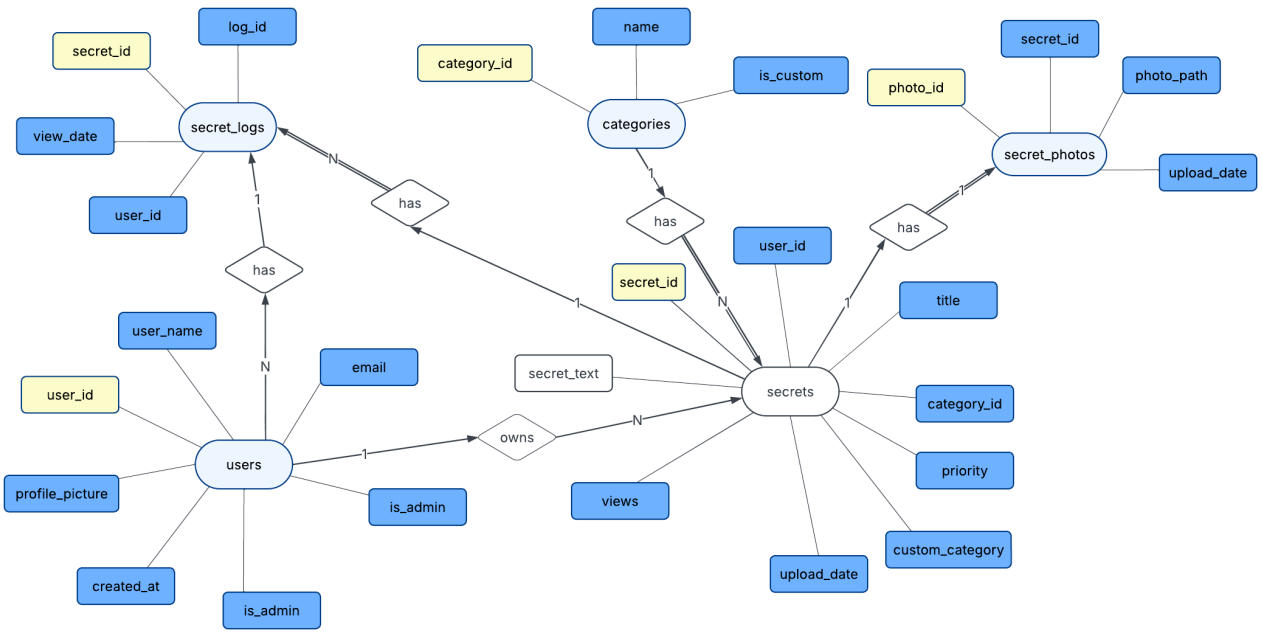

Figure 1: Entity-Relationship Diagram of WhistleVault Database

5 Code Structure

The WhistleVault application is organized into a modular code structure to ensure maintainability and scalability:

- **app.py**: The main application file containing Flask routes, business logic, database interactions, and encryption/decryption functions. Key routes include:
  - `/login`: Handles user authentication.
  - `/submit`: Processes secret submission with encryption.
  - `/view_all`: Displays all secrets with sorting functionality.
- **templates/**: Directory containing Jinja2 templates:
  - `view_all.html`: Lists all secrets with a sortable table.
  - `submit.html`: Form for submitting new secrets.
  - `login.html`: User login interface.
  -
  - `profile.html`: User profile management page.
- **static/**: Stores uploaded photos and static assets like CSS files for styling.
- **scripts/setup.sql**: SQL script to initialize the SQLite database with table structures and constraints.

The code uses Python 3.8+ and follows PEP 8 style guidelines for consistency.

6 Installation and Setup

6.1 Prerequisites

To run WhistleVault, the following are required:

- Python 3.8 or higher with the Flask framework, flask-bcrypt for password hashing, and cryptography for encryption.
- SQLite3 for database management.
- A web server environment, either locally (e.g., Flask development server) or on a platform like PythonAnywhere.

6.2 Local Installation

Follow these steps for a local setup:

1. Install the necessary Python packages using: `pip install flask flask-bcrypt cryptography`.
2. Create a virtual environment to isolate dependencies: `python3 -m venv venv`.

3. Activate the virtual environment:
    - On Linux/Mac: `source venv/bin/activate`.
    - On Windows: `venv`.
4. Initialize the SQLite database by running: `sqlite3 whistlevault.db < scripts/setup.sql`.
5. Modify `app.py` to point to the correct database path (e.g., `/home/pratikdhage/WhistleVault/wh`
6. Start the application: `python app.py`.

6.3 PythonAnywhere Deployment

For deployment on PythonAnywhere:

1. Upload the project as a ZIP file (e.g., `WhistleVault.zip`) to your PythonAnywhere account under `/home/pratikdhage/`.
2. Extract the files: `unzip WhistleVault.zip -d /home/pratikdhage/`.
3. Configure a new web app: Set the source directory to `/home/pratikdhage/WhistleVault` and update the WSGI configuration file to point to your application.
4. Install dependencies within a virtual environment created on PythonAnywhere.
5. Reload the web app to apply changes and make it live.

7 Functionality

7.1 User Features

WhistleVault offers a rich set of features for regular users:

- **Signup/Login**: Users can register with a username, email, and password, and log in securely using hashed passwords.
- **Submit Secret**: Allows users to create secrets with a title, category selection, priority level, and an optional photo attachment.
- **View Secrets**: Provides access to both secrets owned by the user and those shared with them.
- **View All**: Displays a comprehensive list of all secrets, with a sortable interface for columns like title and date.
- **Profile**: Allows users to upload or remove a profile picture and view account creation details.
- **Delete Account**: Provides an option to permanently delete the user account and associated data.

7.2 Admin Features

Administrators have elevated privileges:

- **Admin Dashboard**: Offers a centralized view of all secrets with the ability to delete any entry.
- **Admin Profile**: Allows admins to manage their profile picture and account settings.

7.3 Sorting Feature

The `view_all.html` template includes a dropdown menu enabling users to sort the secret list by various columns (e.g., Title, Date, Views) in ascending or descending order. This functionality is implemented in `app.py` using dynamic SQL queries that leverage the database's ORDER BY clause, enhancing user interaction with the data.

8 Usage Instructions

8.1 User Workflow

To use WhistleVault as a regular user:

1. Navigate to `/login` to log in or `/signup` to create a new account.
2. Submit a new secret by visiting `/submit` and filling out the required fields (title, category, etc.).
3. View personal or shared secrets at `/view_secrets`, or explore all secrets at `/view_all`.
4. Manage your profile settings at `/profile`, including photo updates.
5. Delete your account permanently at `/delete_account` if desired.

8.2 Admin Workflow

For administrators:

1. Log in using the admin credentials at `/admin_login`.
2. Access the admin dashboard at `/admin_dashboard` to view and delete secrets.
3. Update profile settings at `/admin_profile`.

9 Future Improvements

To enhance WhistleVault, the following improvements are proposed:

- **Enhanced Security**: Integrate two-factor authentication (2FA) to add an extra layer of protection.

- **Search Functionality**: Develop a search bar to quickly locate secrets by keyword or category.
- **User Roles**: Introduce moderator roles to assist admins in managing content.
- **Backup System**: Implement automated database backups to prevent data loss.
- **Mobile Support**: Create a mobile-friendly interface or a dedicated app for on-the-go access.

10 Technical Details

10.1 Error Handling

The application employs Flask's `flash` system to display error messages to users, such as database connection failures or invalid inputs. Robust try-except blocks are used throughout `app.py` to catch and handle exceptions, ensuring the system remains stable during operations like database queries or file uploads.

10.2 Encryption Implementation

The Fernet symmetric encryption from the cryptography library is used to encrypt secret data. A unique encryption key is generated and stored securely, ensuring that only authorized users can decrypt their data. The encryption process is integrated into the `submit` route in `app.py`.

11 References

- Flask Documentation. Available at: `https://flask.palletsprojects.com/`.
- SQLite Documentation. Available at: `https://www.sqlite.org/docs.html`.
- Cryptography Library. Available at: `https://cryptography.io/en/latest/`.
- Elmasri, R., Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson.