Secure Media Manager

Overview

Secure Media Manager is a command-line interface (CLI) application developed in Python 3.x that enables users to securely store and manage multimedia files (e.g., MP3s, lyric sheets, scores). It supports role-based access control with admin and user privileges and implements secure coding practices including encryption, integrity checking, and timestamping.

Features

- Command-line interface
- Role-based access control (Admin/User)
- Create, Read, Delete (CRUD) operations
- AES encryption of all media files using Fernet
- Checksum (SHA-256) for file integrity
- Timestamping for creation and modification
- Singleton pattern for database access

Setup Instructions

Requirements

- Python 3.x
- Dependencies:
    - cryptography

Install dependencies:

```
pip install cryptography
```

Running the Application

1. Place the media files you want to add in the project folder.
2. Run the application:

```
python secure_media_manager.py
```

3. Login with a role (admin or user) when prompted.

Usage

- **Admin Role:** Can add, view, and delete media files.
- **User Role:** Can only add and view metadata.

Example Actions

- Add file: Input path to a valid MP3 or text document.
- View metadata: Enter the exact filename.

- Delete file: Only available to admins.

Design Patterns Used

- **Singleton Pattern** is applied in the MediaDatabase class to ensure a single point of interaction with the database.
- **Separation of Concerns** is used to split logic among encryption, database handling, and user interaction.

External Libraries Justification

- **cryptography (Fernet):** Used for encryption and decryption. Only this library is used externally, and its use accounts for <20% of the total codebase. It is essential for secure storage.

Security Features

- **File Encryption:** Ensures confidentiality using AES-based encryption (Fernet).
- **Checksums:** SHA-256 hashes are calculated to verify file integrity.
- **Timestamps:** Creation and modification dates are recorded.
- **Role Restriction:** Users are limited to certain operations.
- **Testing Tools:**
  - **Linting:** flake8 used to ensure code quality.
  - **Security:** bandit -r . run to check for common Python security issues.

Testing Evidence

- Application was tested with lyric sheet .txt and .mp3 files under both roles.
- Metadata retrieval verified by checksum and timestamp logging.
- Bandit and linting reports included in the project folder (test_files).

Deviations from Unit 3 Design

- Replaced SQL database with JSON for simplicity and to ensure full control over encryption and structure.
- CLI modified to handle basic terminal input without external CLI frameworks.

Academic Integrity

All external sources are cited. The code follows the University of Essex referencing standard.

References

- Python Cryptography Library: https://cryptography.io/en/latest/
- Python hashlib: https://docs.python.org/3/library/hashlib.html
- Python json and os: https://docs.python.org/3/

Files Created:

- lyrics/test_lyric.txt – sample lyric file (encrypted)
- audio/test_audio.mp3 – sample audio file (encrypted)

Operations Performed:

- SHA-256 checksum calculated for each file
- Files encrypted using cryptography.fernet
- Metadata saved in metadata.json: