Development Team Project Design Document

Team Members
1. Syed Imran Ali
2. Ilyas Demirtas
3. Ayo Adeniran
4. Oluwatosin Ibisola
5. Ibrahim Abukar

Assignment Breakdown

1. Introduction and Requirements Analysis

Introduction: In today's music industry, copying and copyright infringement pose significant challenges. Our team has designed a digital application to address these issues, safeguarding the rights of writers and performers. This application will securely store and manage musical artefacts, including lyrics, music scores, and recordings.

Requirements Analysis:

Secure Storage: The application must securely store digital artefacts, such as lyrics, music scores, and audio recordings, protecting them from unauthorized access.

Checksum Generation: Checksums will be generated to maintain data integrity, acting as unique digital fingerprints for each artefact.

Encryption: All artefacts will be encrypted, ensuring that only authorized users can access the data.

Time Stamping: The application will generate and store timestamps to record creation and modification dates for each artefact.

Roles and Permissions:

Administrators: Administrators can create, delete, and manage artefacts and user roles.

Users: Users can add, view, and modify their own entries and view others' artefacts without modifying them.

By integrating these features, the application provides a robust solution for managing and protecting musical artefacts.

2. Database Design and Data Structures

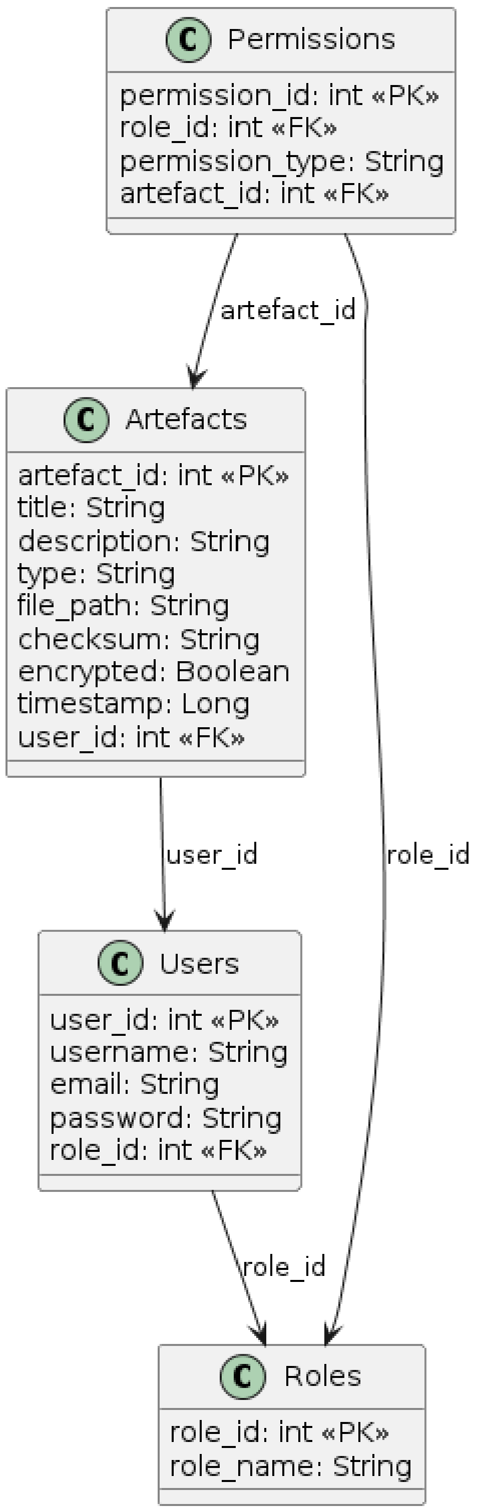
Database Schema
In order to design the database schema for storing lyrics, music scores, and recordings, it is necessary to create several tables. The key tables will include Users, Artefacts, Roles, and Permissions. Each table will have specific attributes and relationships in order to ensure the integrity and security of the data.

Tables and Relationships

1. **Permissions Table**
    - **permission_id** (Primary Key): Unique identifier for each permission.
    - **role_id** (Foreign Key): Links to the Roles table
    - **permission_type**: Type of permission (e.g., create, read, update, delete)
    - **artefact_id** (Foreign Key): Links to the Artefacts table to define permissions for specific artefacts
2. **Artefacts Table**
    - **artefact_id** (Primary Key): Unique identifier for each artefact
    - **title**: Title of the artefact (e.g., song title)
    - **description**: Brief description of the artefact
    - **type**: Type of artefact (e.g., lyrics, music score, recording)
    - **file_path**: Location of the stored file

- **checksum**: Unique checksum to verify the integrity of the file
- **encrypted**: Indicates whether the file is encrypted
- **timestamp**: Timestamp of the artefact's creation or modification
- **user_id** (Foreign Key): Links to the Users table to identify the owner of the artefact

3. **Users Table**
   - **user_id** (Primary Key): Unique identifier for each user
   - **username**: The name chosen by the user
   - **email**: User's email address
   - **password**: Encrypted password for user authentication
   - **role_id** (Foreign Key): Links to the Roles table to define the user's role
4. **Roles Table**
   - **role_id** (Primary Key): Unique identifier for each role
   - **role_name**: Name of the role (e.g., Admin, User)



The relationships between these tables ensure clarity and functionality. The `role_id` foreign key in the Users table assigns roles to users, determining permissions. The `user_id` foreign key in the Artefacts table links artefacts to their owners. The Permissions table connects roles to specific permissions, indicating user actions on artefacts. This structure ensures accurate management of roles and permissions across all artefacts.

Data Structure: To manage artefacts effectively, specific data structures ensure integrity, security, and timestamp management.

1. **Checksums:**
   - **SHA-256:** This algorithm generates a unique checksum for each file, stored in the Artefacts table to verify integrity.
2. **Encryption:**
   - **AES (Advanced Encryption Standard):** This standard encrypts artefacts, with encryption status indicated in the Artefacts table.
3. **Time Stamping:**
   - **UNIX Timestamps:** These record creation and modification times, stored in the Artefacts table. This provides a precise way to track dates and times.

The implementation of these data structures ensures the secure and efficient management of artefacts.

3. UML Diagrams: Use Case and Sequence Diagrams

Use Case Diagram: Identifies the actors and their interactions with the system.

Actors:

- Administrator: Can create and delete artefacts, manage users.
- User: Can add, view, and modify their own entries, view others' entries.

Use Cases:

1. Login
2. Create Artefact
3. Delete Artefact
4. Add Artefact
5. View Artefact
6. Modify Artefact
7. Generate Checksum
8. Encrypt Artefact
9. Generate Timestamp

The main actors **Actors:**

- **Administrator:** Can create and delete artefacts, manage users.
- **User:** Can add, view, and modify their own entries, view others' entries.

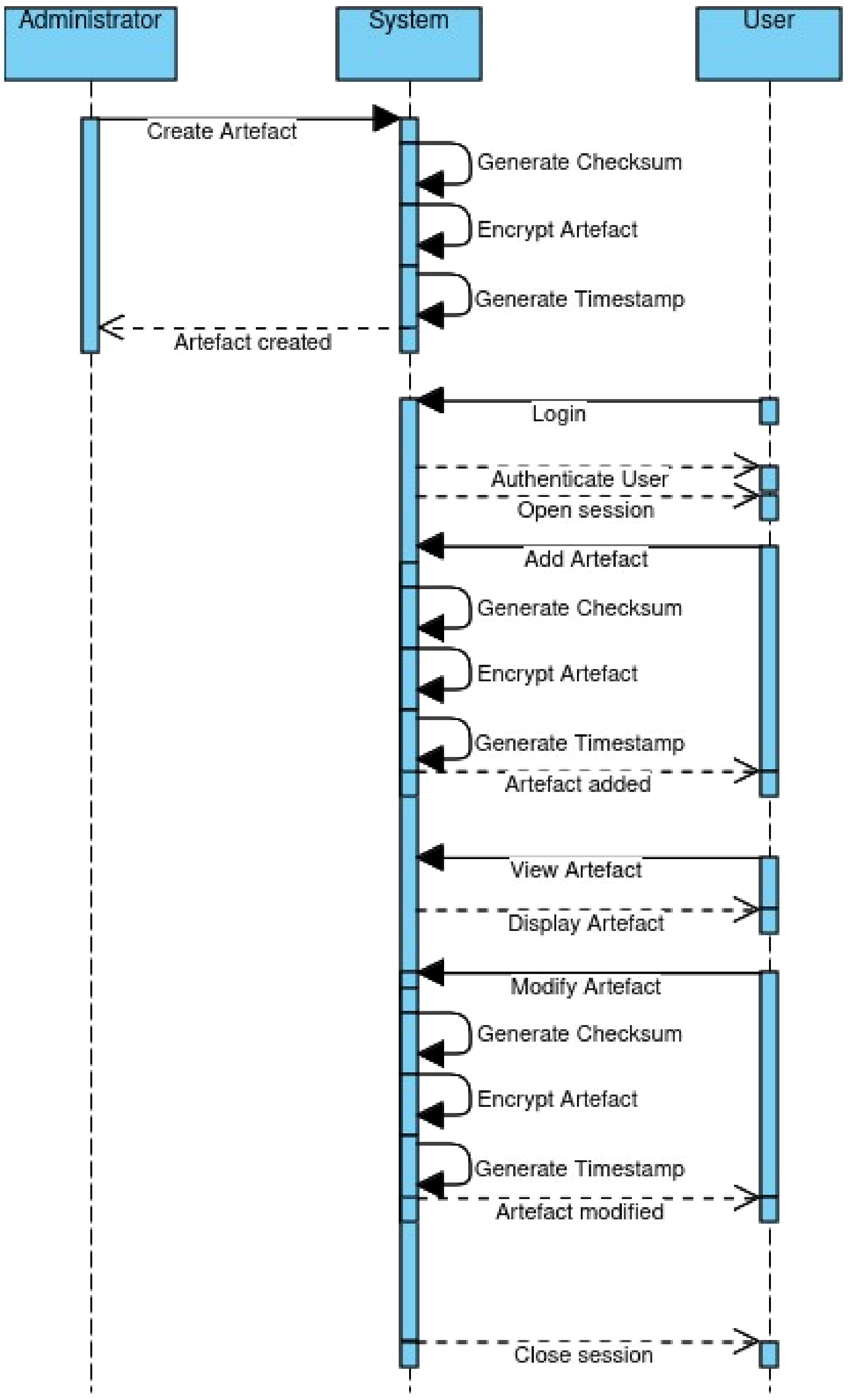


Fig: Administrator User System Sequence diagram

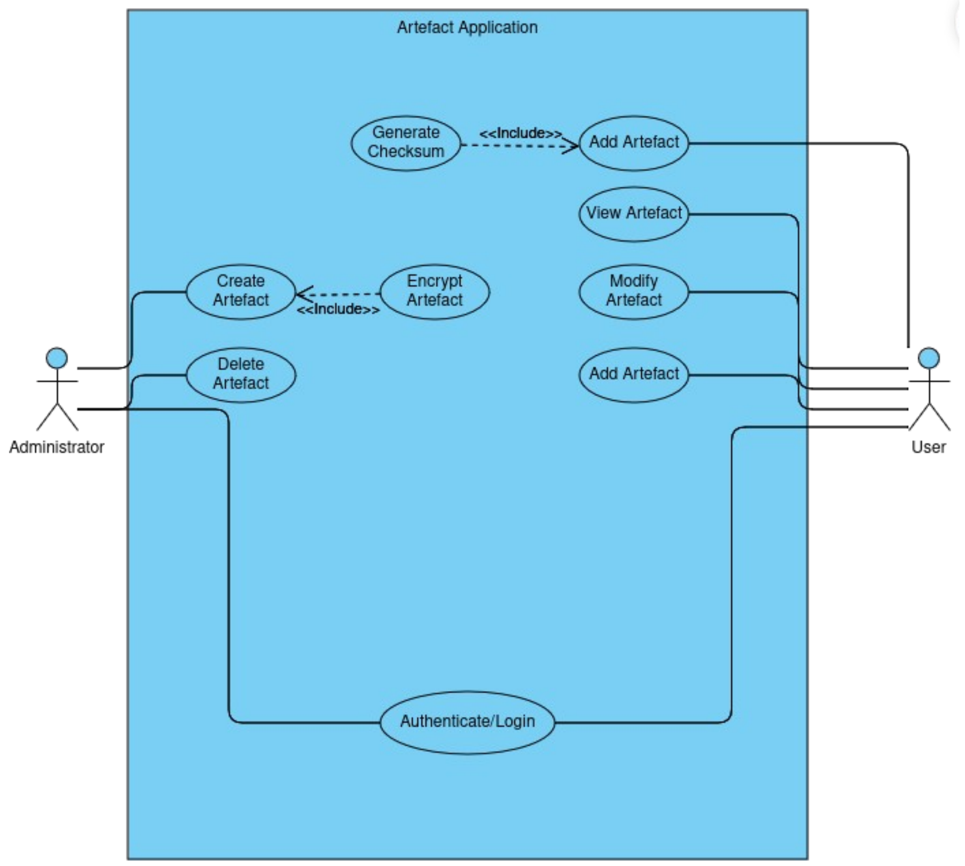


Fig: Artefact Application Use case diagram

4. UML Diagrams: Class Diagram and Data Flow

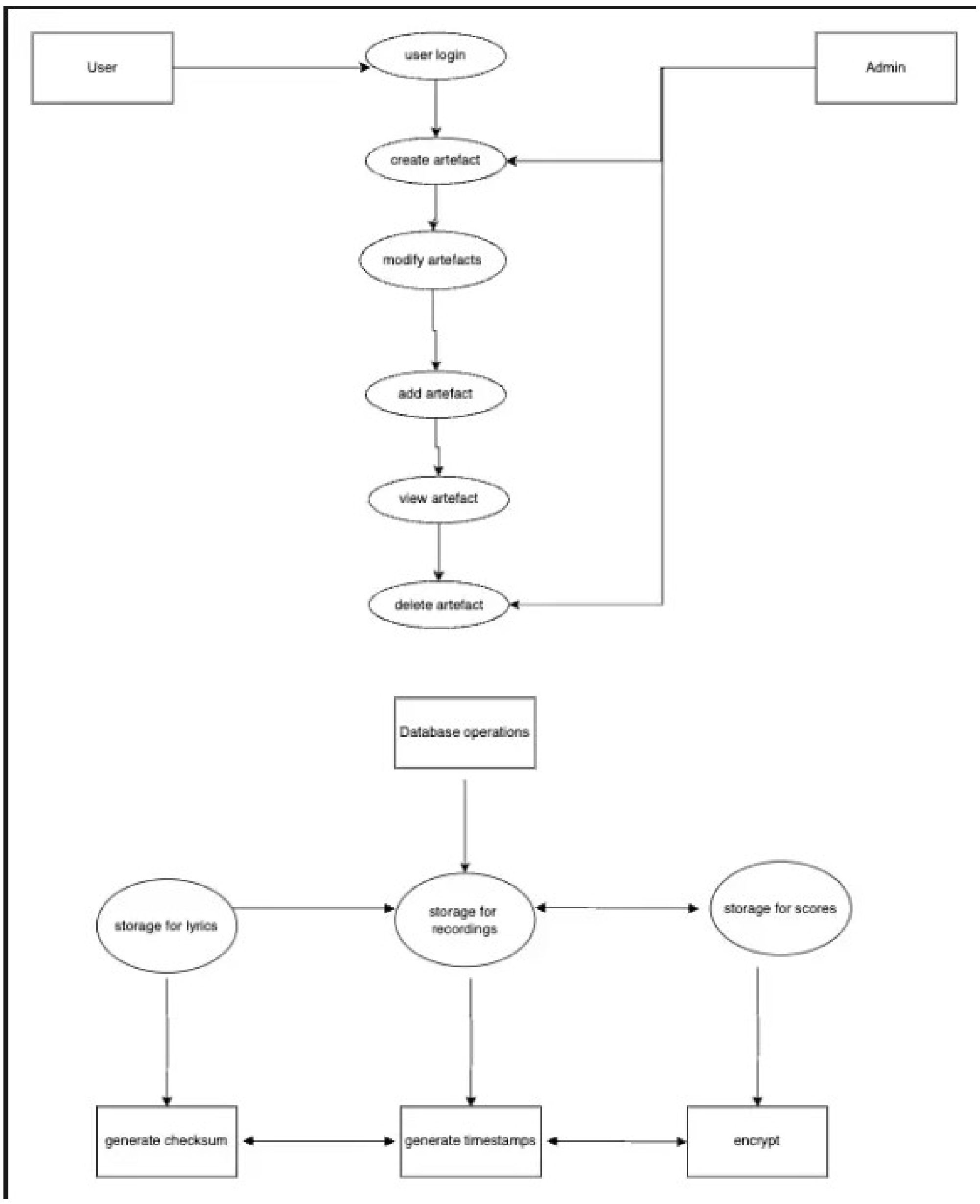


Fig: artefact management system data flow

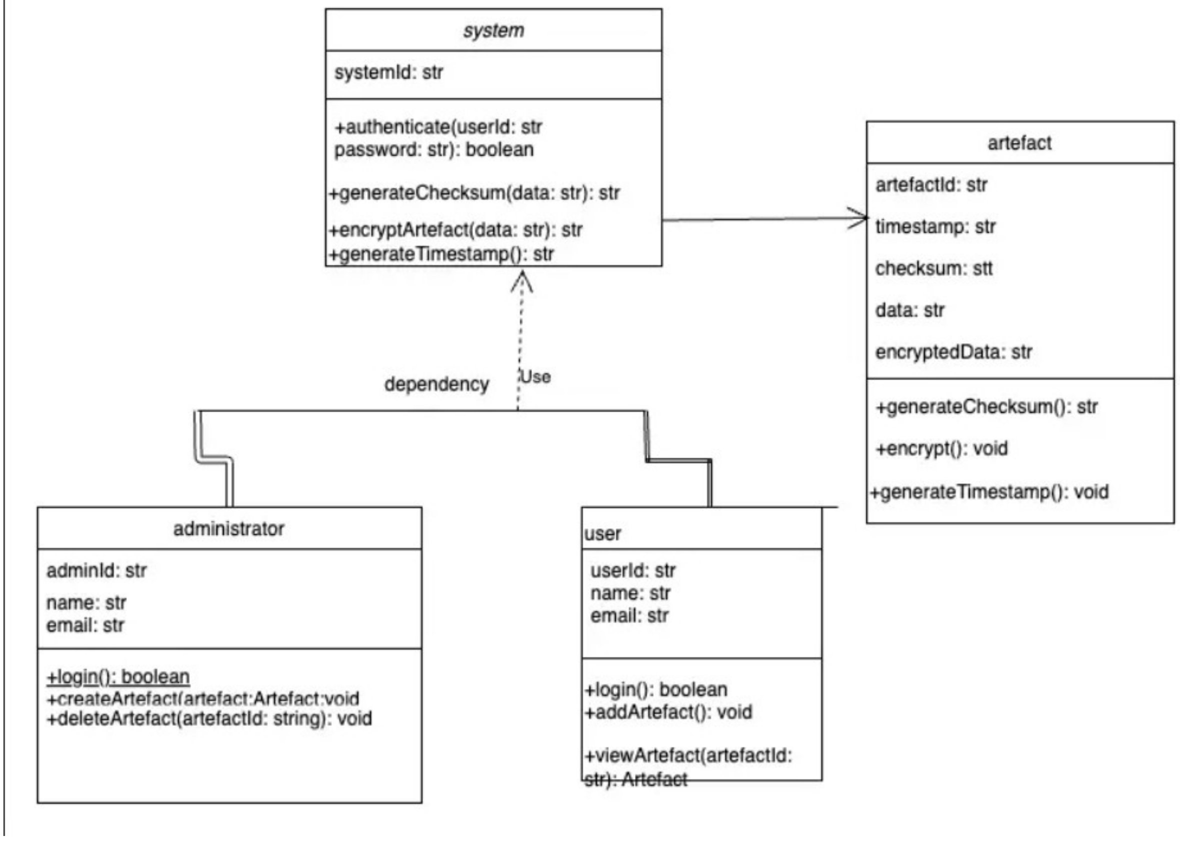

system
systemId: str
+authenticate(userId: str password: str): boolean
+generateChecksum(data: str): str
+encryptArtefact(data: str): str
+generateTimestamp(): str
artefact
artefactId: str
timestamp: str
checksum: stt
data: str
encryptedData: str
+generateChecksum(): str
+encrypt(): void
+generateTimestamp(): void
dependency
Use
administrator
adminId: str
name: str
email: str
+login(): boolean
+createArtefact(artefact:Artefact:void
+deleteArtefact(artefactId: string): void
user
userId: str
name: str
email: str
+login(): boolean
+addArtefact(): void
+viewArtefact(artefactId: str): Artefact

Fig: Artefact management class diagram

OOP Principles: Encapsulation: Each class contains its own data and methods. Inheritance: The artefact class generalizes all artefacts. Abstraction: Hides implementation details, providing simple interfaces. Polymorphism: Allows creation of various artefacts using a shared interface.

5. Encryption, Checksums, and Time Stamping Mechanisms

In relation to the project description, there are different elements and functions for each level of encryption, checksums, and time stamping that need to be described and integrated into the application.

Encryption: Encryption ensures the security of artefacts (lyrics, music scores, and recordings) during storage and transmission. Below are the methods and steps for implementing encryption.

1. **Encryption Methods:**
    - **Symmetric Encryption:** Uses the same key for encryption and decryption. AES (Advanced Encryption Standard) will be used for its speed and efficiency with large data sets.
    - **Asymmetric Encryption:** Uses different keys for encryption and decryption. RSA (Rivest-Shamir-Adleman) is suitable for encrypting small data blocks like encryption keys.
2. **Key Management:**
    - **Key Generation:** Secure random number generators will produce keys.
    - **Key Storage:** Store keys in a key management service to prevent hard-coding in the application.
3. **Encryption Process:**

    - Encrypt artefacts before storing them in the database and during transmission using protocols like TLS (Transport Layer Security).
4. **Integration:**
    - Use libraries like PyCryptodome for Python and Crypto++ for C++ during upload and retrieval to ensure data security.

Checksums: Checksums verify the authenticity of artefacts and ensure data integrity.

1. **Generation:**
    - Use SHA-256 to generate checksums when an artefact is created or modified.
2. **Verification:**
    - Store checksums with artefacts and recalculate them when accessed to ensure integrity.
3. **Integration:**
    - Include checksum generation and verification functions during artefact upload and retrieval.

Time Stamping: Time stamping tracks the creation and modification times of artefacts, forming historical records.

1. **Mechanism:**
    - Use the system clock to generate timestamps, stored in a designated field in the database for each artefact.
2. **Process:**
    - Add timestamps when artefacts are created or modified, updating the field each time.
3. **Integration:**
    - Employ libraries like `datetime` in Python to retrieve the current time and ensure timestamps are included in the creation and update processes.

Integration into the Project:

- **Encryption:** Integrate both encryption and decryption features into the artefact storage and retrieval mechanism.
- **Checksums:** Implement processes to create and update checksums, verifying them whenever artefacts are accessed.
- **Time Stamping:** Include timestamps throughout the creation and modification of artefacts.

Incorporating these security measures will maintain data integrity, confidentiality, and traceability. It's critical to thoroughly examine each step to ensure no flaws or vulnerabilities surface.

Reference:

Fernández-Medina, E. & Piattini, M. (2005) Designing secure databases. *Information and software technology*. [Online] 47 (7), 463–477.

Schneier, B. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C. John Wiley & Sons.

Stallings, W. (2016). Cryptography and Network Security: Principles and Practice. Pearson.

Ferraiolo, D., Kuhn, R., & Chandramouli, R. (2003). Role-Based Access Control. Artech House.

Rumbaugh, J., Jacobson, I., & Booch, G. (2004). The Unified Modeling Language Reference Manual. Addison-Wesley.