Music Storage System

Secure Software Development

SSD: Group 2

Table of Contents

[Introduction: 2](#_Toc579548400)

[Database Structure 2](#_Toc791853751)

[Database Creation 2](#_Toc983633513)

[Application Structure 3](#_Toc876375167)

[Libraries & Dependencies 3](#_Toc208064812)

[Development Dependencies 3](#_Toc1832044310)

[Testing Dependencies 4](#_Toc239635583)

[Functionality 7](#_Toc111517341)

[Security Measures & Tests 8](#_Toc1782034162)

[Code Structure 8](#_Toc2109544952)

[Security Implementations & Analysis 8](#_Toc1978479804)

[1. Login Validation Role 9](#_Toc1255743967)

[2. Music Entries 12](#_Toc1015651054)

[3. Read/List Music Entries: 13](#_Toc800175715)

[Automated Code Testing 17](#_Toc67614729)

[1. Ruff – Python Code Linter (to comply with PEP8 coding guidelines) 17](#_Toc766467461)

[2. Radon – Python Code for metrics to measure code Cyclomatic Complexity 18](#_Toc1226824219)

[3. Bandit – Python Code Security Linter 18](#_Toc1469756673)

[4. PyTest – Automated Testing for code errors 20](#_Toc501592705)

[5. Safety – Libraries Vulnerability Scanning 21](#_Toc1853316977)

[6. Pythonfuzz 22](#_Toc1082364028)

[Future Improvements: 24](#_Toc76088146)

[Conclusion 25](#_Toc1344374082)

[References 25](#_Toc799750377)

# Introduction:

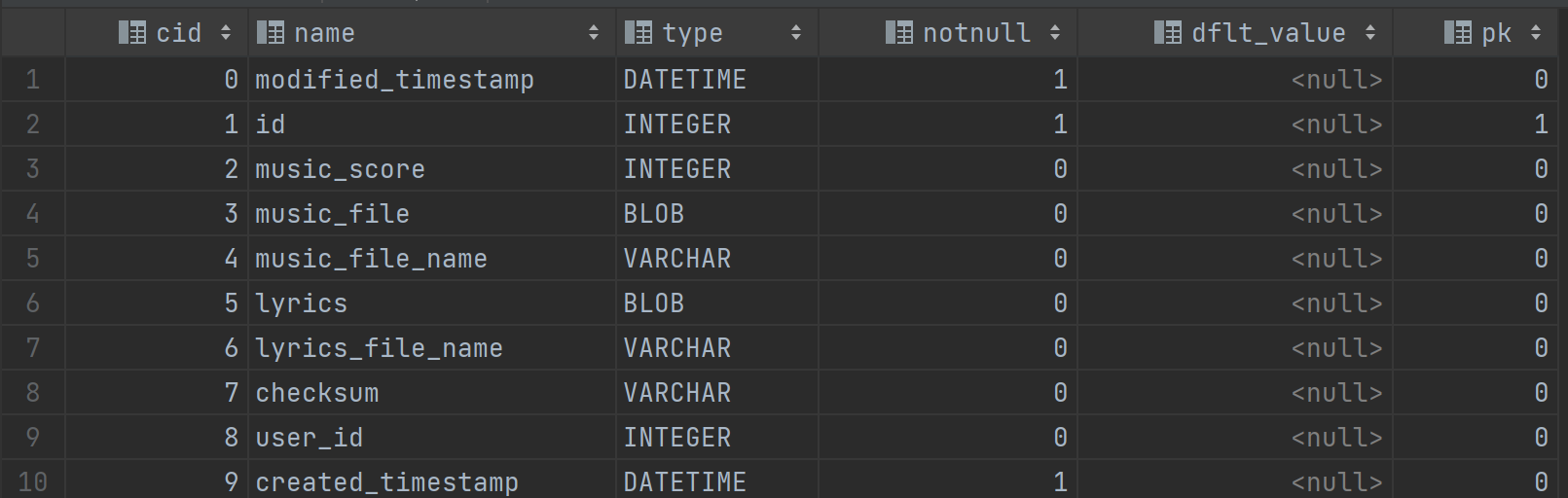
This comprehensive report presents a command-line application designed for a music storage system, empowering users to add and modify their entries seamlessly. Additionally, administrators are granted the ability to add, modify, and delete any entries alongside the functionality to calculate checksums. The primary objective of this application is to establish a robust platform that effectively prevents copyright infringements.

# Database Structure

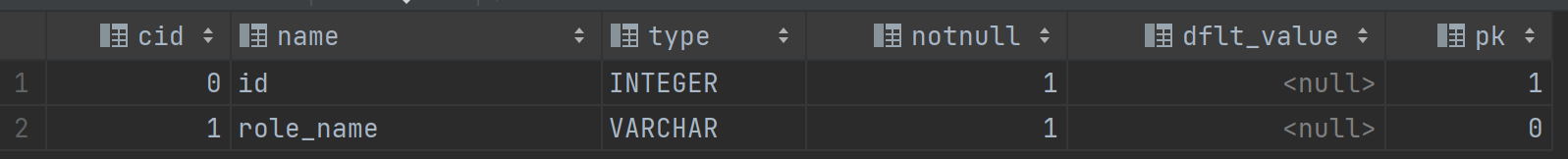
The SQL database of choice is SQLite, which implements a self-contained, serverless, zero-configuration, transactional SQL database engine and is lightweight (SQLite, n.d.). The data types is aligned with the design document proposed. By using the SQLite command (**PRAGMA (PRAGMA table\_info(table\_name);**), it allows to show information of the database tables, the Music-Storage-System contains visibly.

## Database Creation

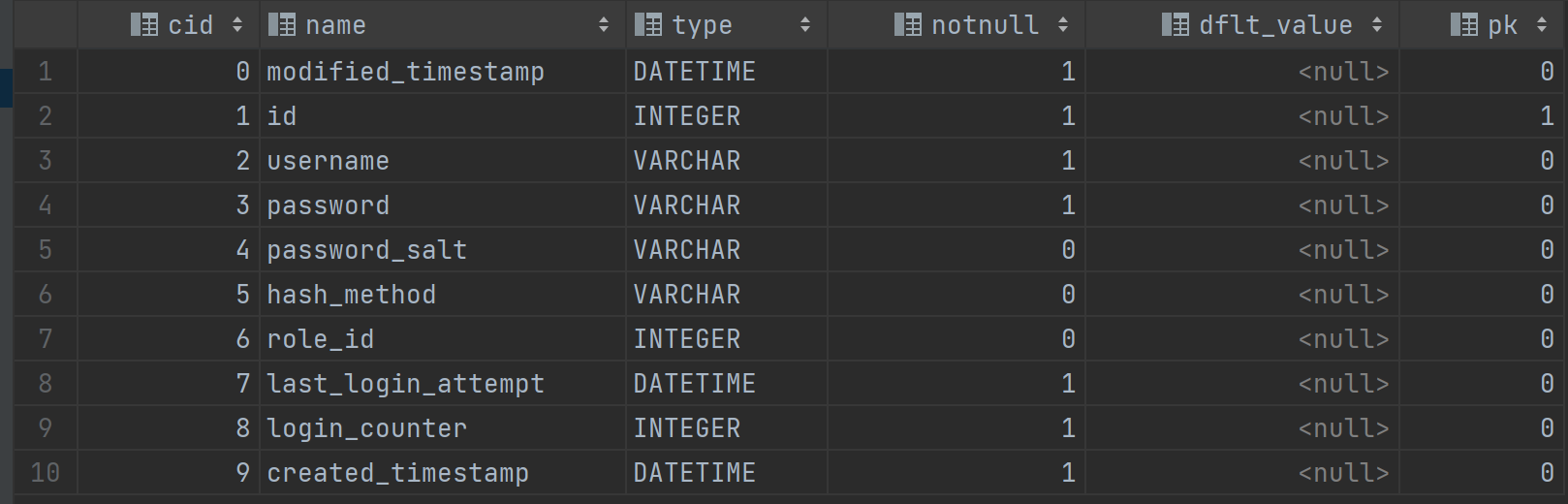
1. **‘musicdata’** Table



1. ‘**role’** Table



1. **‘user’** Table



# Application Structure

Libraries & Dependencies

In order to run the Music Storage System CLI application, the following libraries below must be installed. A description of each is provided to clarify its needs. All required libraries are stated in the application .zip file under **‘Music-Storage-System-main\requirements.txt’**.

The following libraries are required to develop the CLI application, including encryption, command creation, built-in virus scanning of uploaded data, session validation, and database communication.

|  |  |
| --- | --- |
| Development Dependencies | Purpose |
| 1. Typer - 0.9.0 | Library that simplifies the creation and parsing of command line arguments and commands, enhancing user experience in navigating CLI applications. (Typer, n.d.). |
| 1. Sqlmodel - 0.0.8 | Serves as an ORM (Object-Relational Mapper), facilitating interaction with databases through simplified SQL operations. |
| 1. PyJWT - 2.7.0 | Library which allows you to encode, decode, and verify the cryptographic signature of JSON Web Tokens (JWT). |
| 1. argon2-cffi - 21.3.0 | A secure password hashing library that implements the Argon2 algorithm for Python |
| 1. Pycryptodomex - 3.18.0 | To encrypt and decrypt data such as audio, lyrics, and JWT uploaded to the Music-Storage-System using AES-256 encryption. |
| 1. PyClamd - 0.4.0 | Open-source antivirus engine communicates with the ClamAV server to scan uploaded files and receive scan reports in return. |

To ensure the quality and detection of security flaws in the CLI Application, testing libraries have been used to perform security code scanning. The below are the libraries used for testing:

|  |  |
| --- | --- |
| Testing Dependencies | **Purpose** |
| 1. Bandit | Security linter that identifies potential security issues in Python code by flagging common vulnerabilities and exposing associated Common Weakness Enumerations (CWEs) (Bandit, n.d.). |
| 1. PyTest | The library use for simplifying the process of writing and running tests, promoting code quality and preventing the introduction of errors (pytest, n.d.). |
| 1. Ruff | A linter that helps to identify problematic areas in your code following PEP8 standards (Ruff.rs, n.d.). |
| 1. Safety | Safety is a Python library that checks your installed dependencies for known security vulnerabilities ("Safety," n.d.). |
| 1. Radon | Used to calculate the cyclomatic complexity of a program, offering developers insight into areas of their codebase that may be too complex and prone to errors ("Radon", n.d.). |
| 1. PythonFuzz | Open-source fuzzing tool utilised within GitLab's security products, which automatically feeds a program with random data to test for bugs. |

Using the command find **. -name '\*.py' | xargs wc -l** shows the lines of code of the directory.

Total line of development code without unit tests: 1629 lines (including Comments)

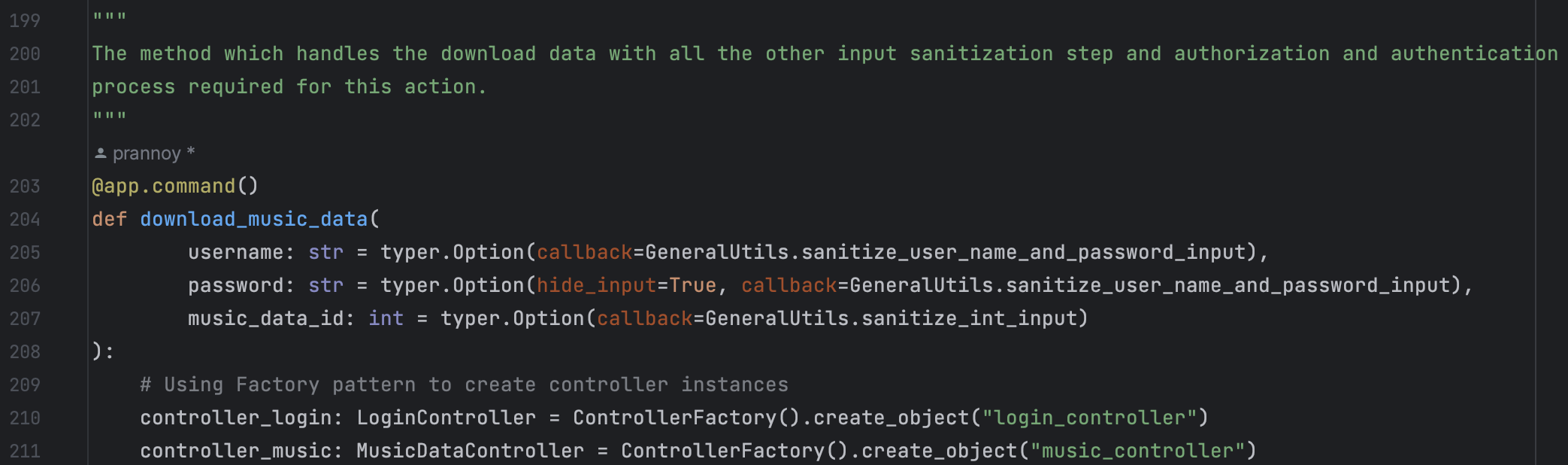
Total line of code using the development libraries: 335 lines (including Comments and boiler plate)

Therefore, the total percentage of external libaries used in the code is about 20.5% including some boiler plate code and comments.

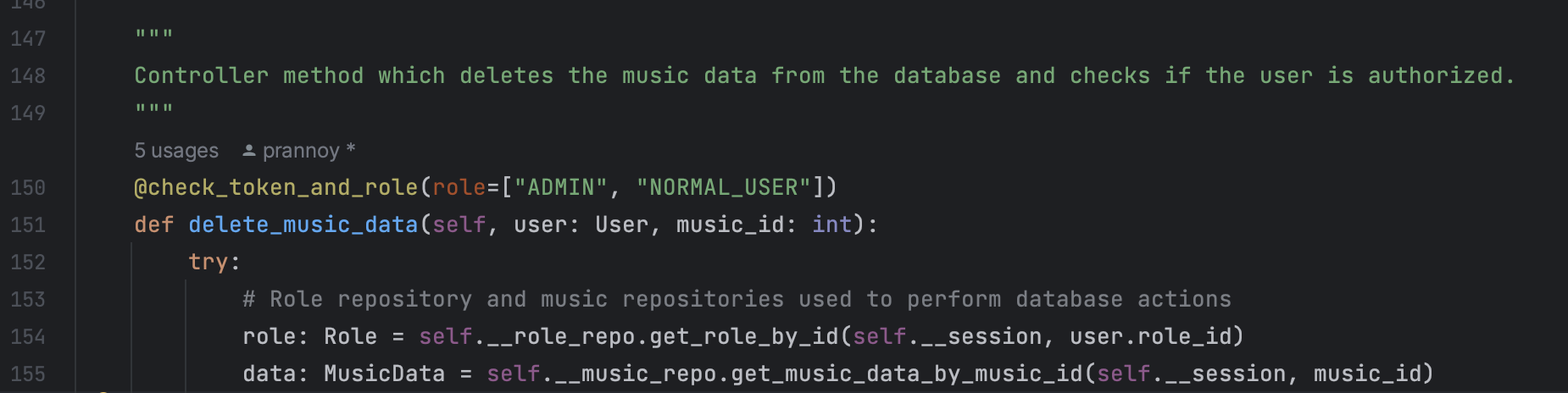
**Design Patterns implemented in the Application**

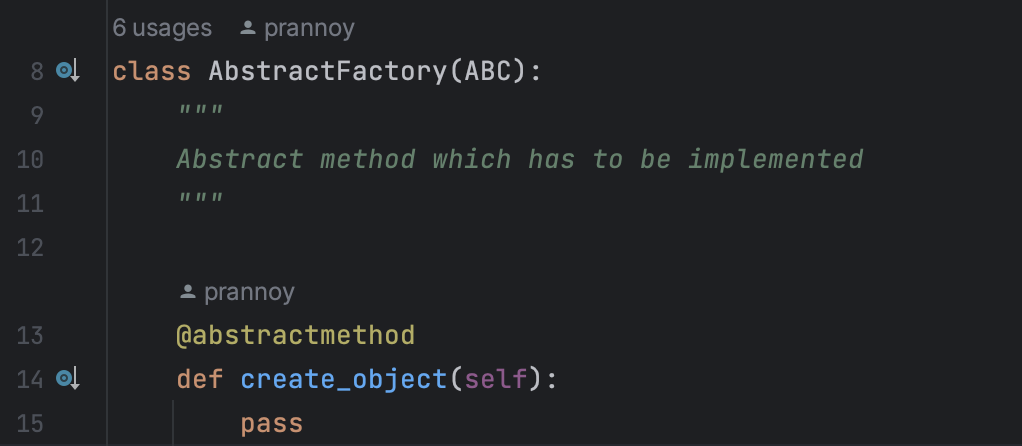
This section provides some code snippets of how the four patterns, Decorator, Singelton, Repository, and Abstract Factory, are applied to this application. All the applied design patterns are mentioned in the design document provided previously.

Factory pattern application in main.py within the code:

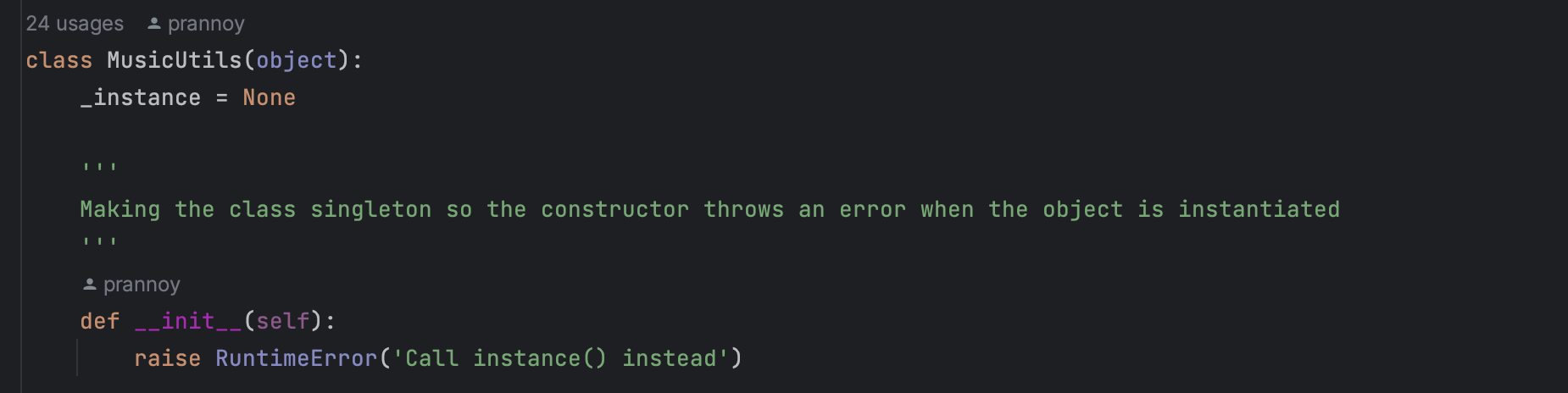


Repository pattern and use of decorators in music\_data\_controller.py



Abstract factory pattern in abstract\_factory.py:

Singelton pattern used in MusicUtils.py



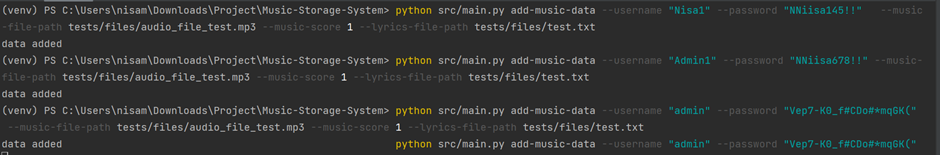
**Design Deviations**

|  |  |
| --- | --- |
| **Deviation** | **Reason** |
| The util class has been divided into many smaller classes, such as, password\_utils, encryption\_utils, jwt\_utils, and decorator utils. | As per the design, there were just two util classes, but the number of methods that were put in the two classes **music\_utils** and **general\_utils** would break the single responsibility principle (Martin, R.C., 2003.). Therefore, dedicated utils classes were added. |
| Download music data added | The previous design did not contain a description of the functionality of the download. As this function is required, it has been implemented in the application. |
| Logger class is not implemented | All the logging is done in the main.py, and it catches the errors thrown by the other classes and logs it there. |

Functionality

The CLI application's implementation includes support for CRUD functionality, which means it enables users to create, read, update, and delete both artefacts and music entries.

To add entry:



To list entries as admin:



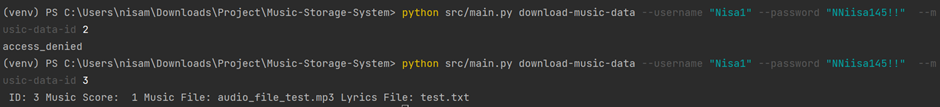
To update entry as admin:



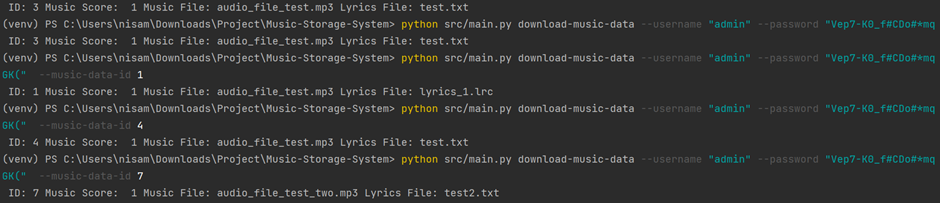
To delete entry as admin:



To download data (users can only download their entries as in the snippet):



To download data (admin can download all entries):



Note: The files are downloaded into the directory where the terminal is being executed from.

# 

# Security Measures & Tests

## Code Structure

In addition to proper coding style, the goal is to perform tests both automated and manual testing to detect vulnerabilities which mitigates the **A06:2021 OWASP Vulnerable Outdated Components** of the Music-Storage Application and ensure present threats mentioned CWEs are remediated in accordance to OWASP (Open Web Application Security Project).

## Security Implementations & Analysis

Based on the security tests performed, the security implementations has been applied based to refrain from CWEs highlighting threats that involves:

* Credential theft
* Data-tampering
* Injection
* Buffer overflow
* Denial Of Service (ReDos\DoS)
* Malware upload

Common vulnerabilities were identified, and then the OWASP guidelines and mitigations were applied in the application; the table shows the threats and the mitigations:

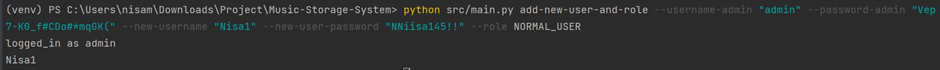
|  |  |  |
| --- | --- | --- |
| Threats | Mitigations | Framework |
| Credential theft | * Hashing passwords using Argon2id Algorithm (Biryukov, A et.al. 2016). * Prevent the reuse of compromised passwords and strong password policy (Pal, B et al. 2019). * Block user login for a while after five retries to prevent brute force. | A01 - Cryptographic failures (OWASP, 2021).  This framework is related to cryptographic failures and leads to sensitive data exposure. |
| Data-tampering | * Encryption for data in rest. * Signing tokens using cryptography techniques like Ed25519 (Bisheh-Niasar, M et al., 2021). * Using checksums for data integrity using SHA-256 (Rachmawati.D et al., 2018) | A01 – Cryptographic failures (OWASP, 2021).  This framework is related to cryptographic failures and leads to sensitive data exposure. |
| Injection/Maleware | * Input validation and sanitization (Ed, 2017). * File extension checks, file virus scanning checks. | A03 – Injection (OWASP, 2021).  This part of the framework relates to the data tempering through injection of malicious input. |
| Buffer overflow/ReDoS | * Fuzz testing (Tsankov, P et.al, 2012). * Input and file size limits. | CWE – 788 – Access of memory location after end of buffer.  This relates to the memory overflow and causes buffer overflow |
| Access Control | * Re-authentication for sensitive actions like Create/Delete/Update. * Short lived tokens (Chaturvedi, A., 2022). | A01 – Broken Access Control (OWASP, 2021)  This part of the framework relates to providing unauthorized access to attackers. |

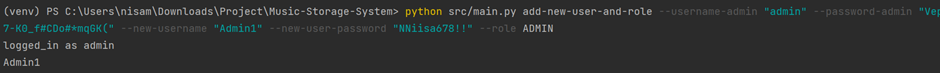
Each function of the Music-Storage-System has been equipped with security practices that address identified threats mentioned in the table. The following section provides evidence of the applied measures in the application.

## Login Validation Role

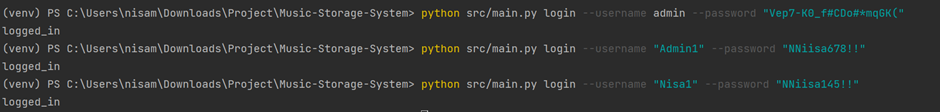
The music storage contains a Login Validation Role function where passwords are involved, the application follows OWASP guidelines to protect, to avoid Credential Thefts possibly leading to exposure of user accounts.

Add users:





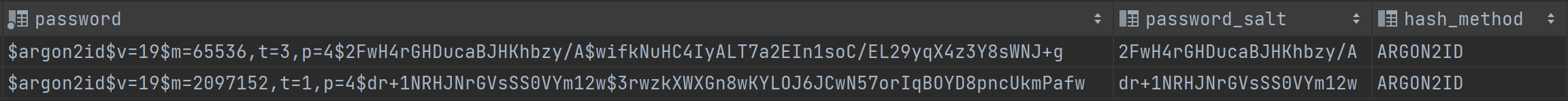
To login:



**Hashing passwords - A02:2021 - Cryptographic failures**

Using Argon2id Algorithm – it matches with the OWASP Password Storage Cheat Sheet recommendation, which suggests using Argon2, bcrypt or scrypt for password hashing (OWASP, 2021a).

In the application, we used Argon2id for hashing passwords, as seen in the screenshot below; a test is performed to ensure that once a password is stored in the database, it contains the Argon2ID:

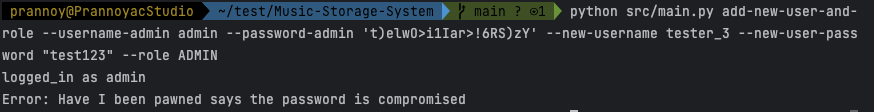


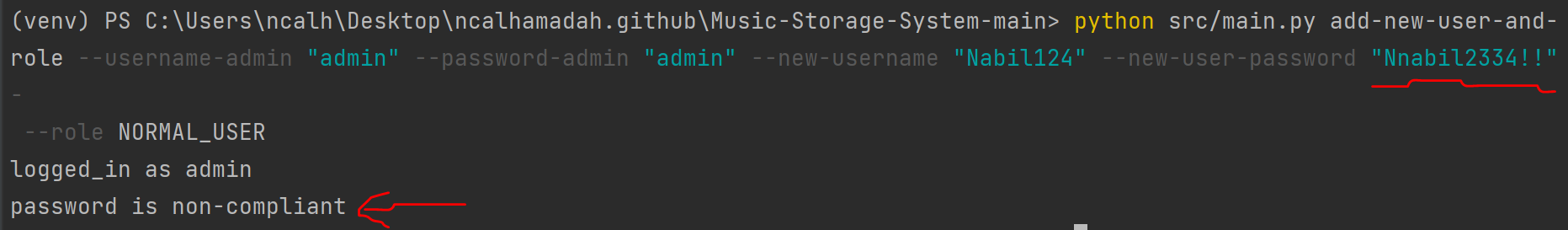
If required to check the configuration of the Argon2id algorithm used, please refer to the **README.md** file.

**Prevention of Reusing Passwords – OWASP Authentication General Guidelines**

Using **HaveIBeenPwned** to allows passwords to be validated if it was compromised or reused several times; in addition, we have also applied the password policy (policy configuration available in README.md), which aligns with OWASP Authentication General Guidelines for implementing proper password strength controls, which also aligns with **OWASP- A01:2021-Broken Access Control.**

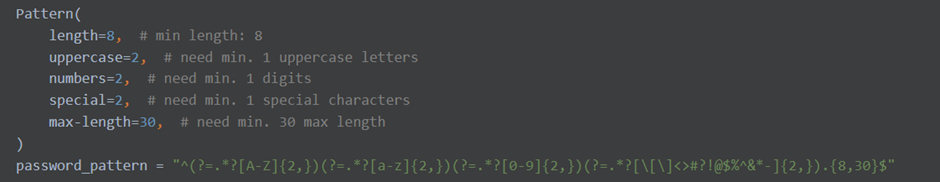
The below are the screenshot of the application preventing to use reused passwords and password policy in-place:





**Password Policy based on OWASP Authentication General Guidelines:**

Maximum and Minimum values range (Min 5 and Max 50 characters) for username and passwords.



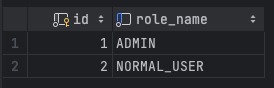
Password non-compliance special character snippets:



**A01:2021–Broken Access Control** **– Least Privilege Authorization**

Level of access is based on the user's assigned roles. It restricts actions to particular roles and denies them if they are not permitted. When a new user is created by the admin, it grabs the role type by **‘id’** from the **‘role’** table, there are two types of roles in the Music-Storage-System:

Role table values:



Role 1 – Admin

Role 2 – normal users (allowed to add, update and delete their entry)

The roles can be extended by modification of some code.

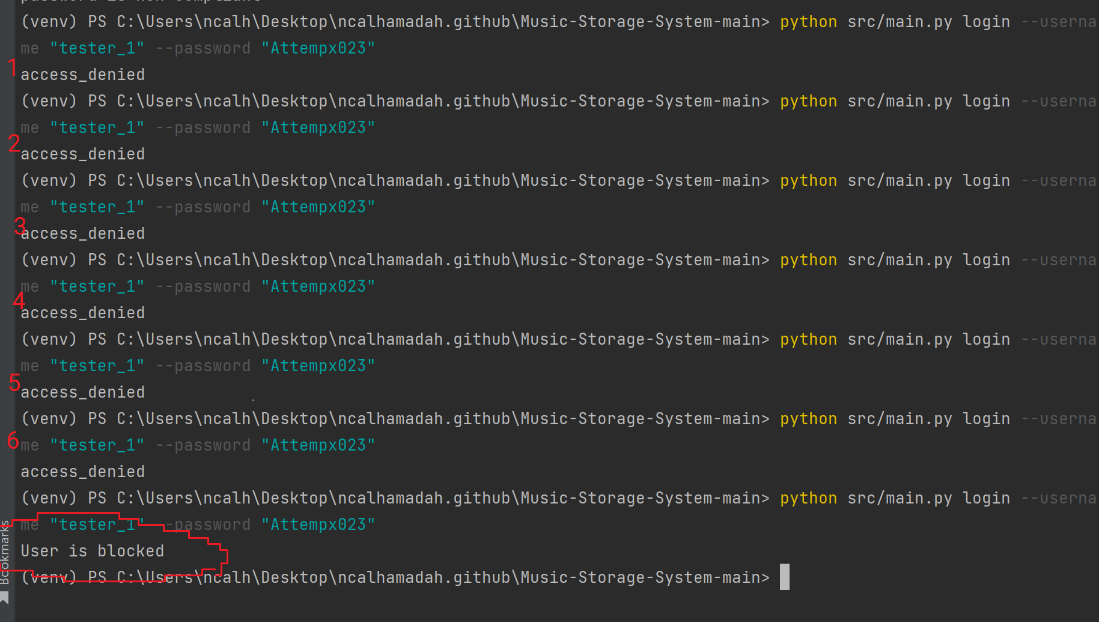
Access denied by least privilege (non-admin):



To delete entry as a user (access denied as user not privileged to delete other’s entries):



As part of guidelines for authentication, to prevent from brute-force attacks on the login, a policy of 5 attempts has been set before getting blocked from logging in as suggested also in **A01:2021-Broken Access Control**. The following screenshot is a test:



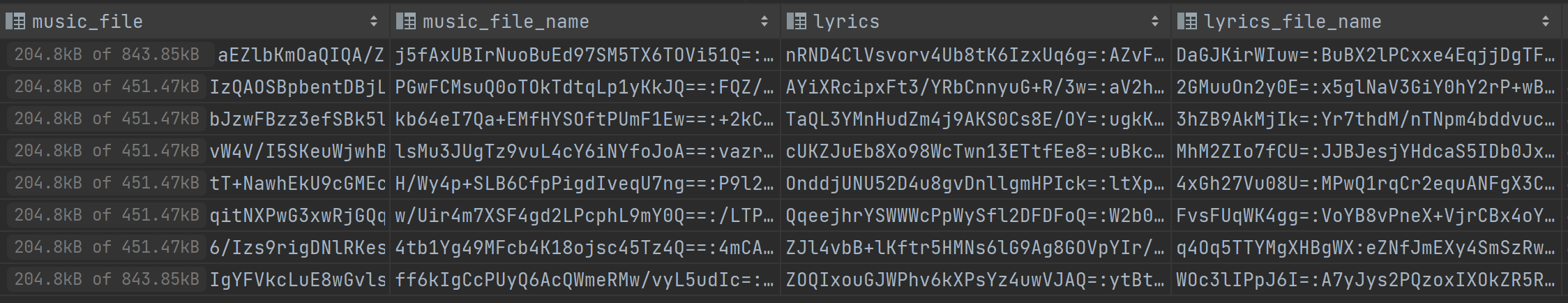
## Music Entries

To secure data confidentiality of the music uploaded to the system, the guidelines related to data exposure have been applied where all data must be encrypted. While ensuring the data integrity of the music, ensuring user sessions are legitimate when performing CRUD (create, read, update and delete) operations is also applied as per guidelines. The following has been applied to refrain from Data Tampering.

**AES256 Encryption (A02:2021-Cryptographic Failure)**

As per the guidelines of OWASP for protecting sensitive data, it is advised to encrypt all sensitive data at rest and ensure up-to-date and robust standard algorithms, protocols, and keys are in place; use proper key management (OWASP, 2021). All related music data is stored in AES-256 encryption **(for a detailed explanation of choice, refer to README.md).**

The below screenshot is a test of uploading data and stored in the database as encrypted data:

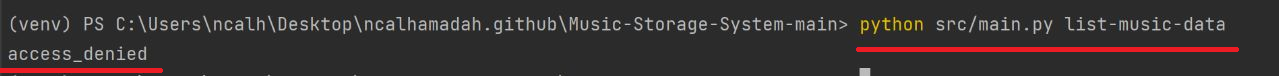


## Read/List Music Entries:

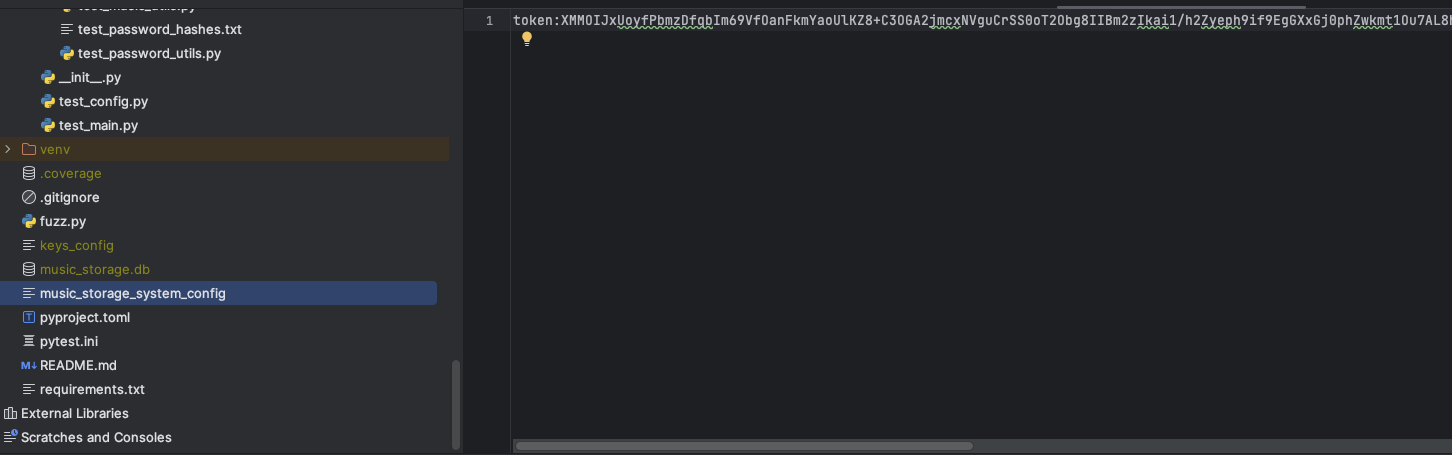
**JWT Tokens (A02:2021- Cryptographic Failure)**

The use of encrypted JWT Tokens signed using the Edd25519 digital signature algorithm **(for more details of the choice, refer to README.md)** is present when attempting to view music records. The tokens act as a session validation of the current user logged in; attempting to run the list function without a valid token denies access. The screenshot below is a test of the session validation when listing music entries:

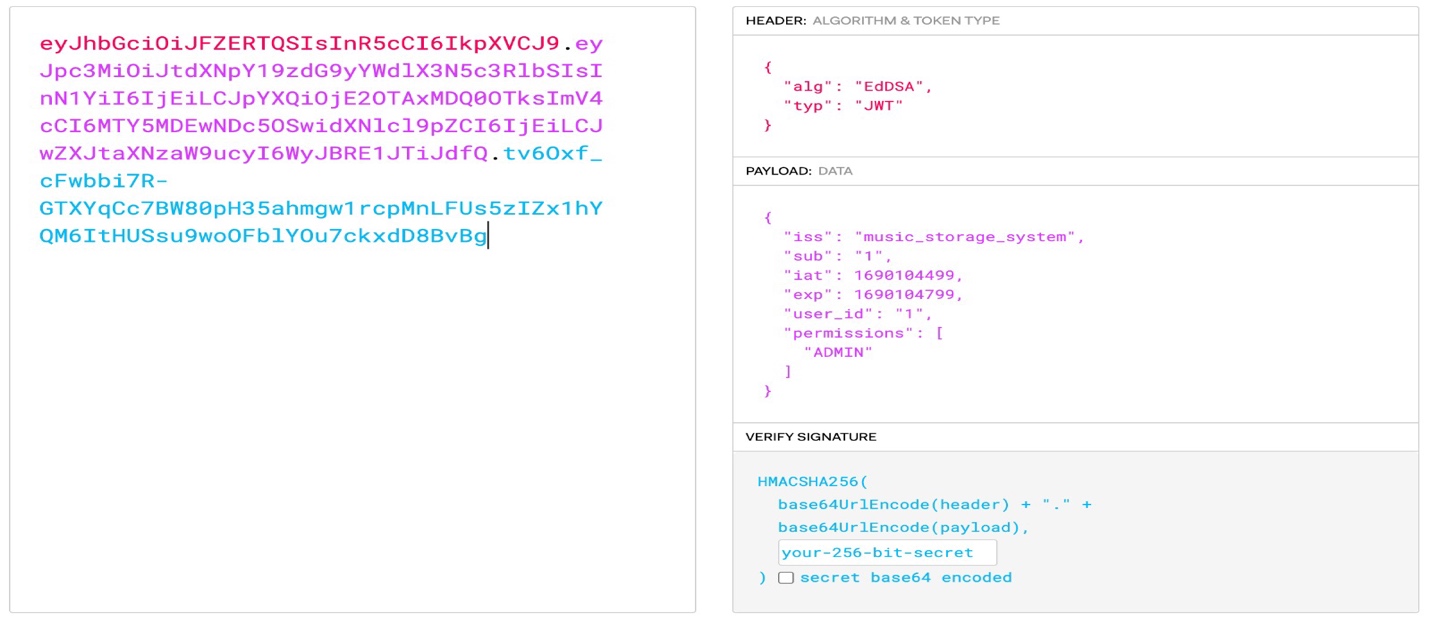
Screenshot attempt to run list-music-data with no user login:



Encrypted Token generated and stored, once user successfully logs in for session validation when performing listing/viewing command:



The token shown above is decrypted using the secret key, and below, the token signed using the EdDSA signature algorithm for the JWT token is presented.

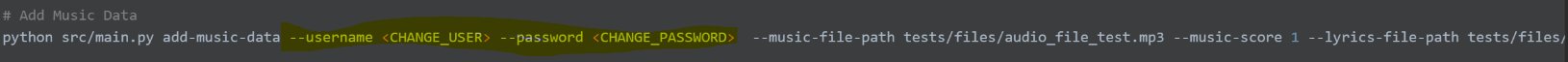


**Credential Revalidation (A01:2021-Broken Access Control)**

The CRUD operation is controlled with validating procedures as per OWASP guidelines to avoid Broken Access Control related threats. The operations follow different validating procedures stated below:

**Creating/Update/Deleting Music Operations – Credential Revalidate:**

To avoid broken access control, OWASP recommends implementing access control mechanisms once and reusing them throughout the application (OWASP, 2021). The screenshot below shows that commands for creating and deleting require re-use of credentials:



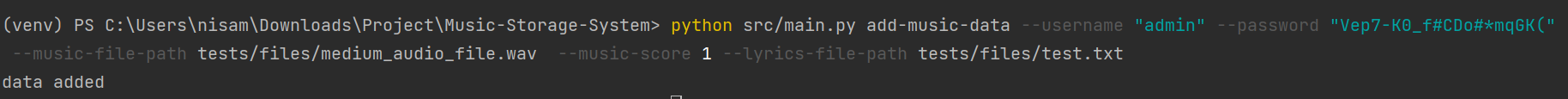
**File Upload Protection:**

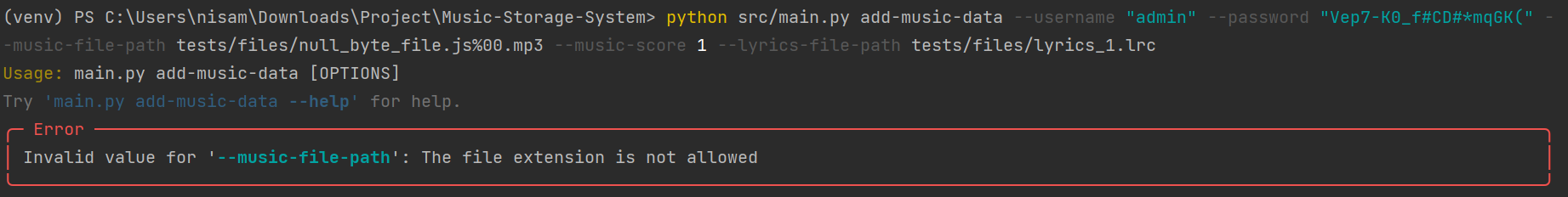
Following OWASP File Upload protection (File upload cheat sheet, OWASP, n.d) the below mentioned measures are implemented:

* Maximum size for upload of audio file (35 MB) and max file for Lyrics(2MB).
* Virus scanning for file using ClamAV (ClamAV, 2021) before file upload. The Virus scanner is tested using an ECIAR file, which is a benign-infected file for AV testing (Harley, D., Myers, L. and Willems, E., 2011).
* Double extensions, such as **double\_extension\_file.mp3.js** are not allowed and are validated.

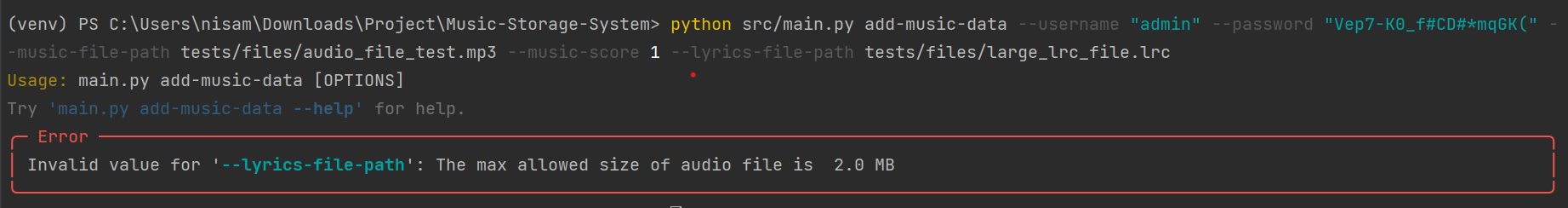
File Upload tests:

Allow types of Audio files such as (.mp3, .wav, aac, wma, ogg, and flac):

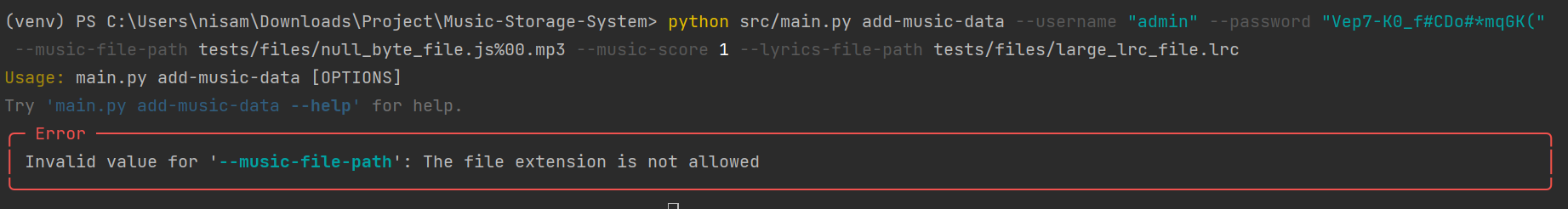


Null byte file:

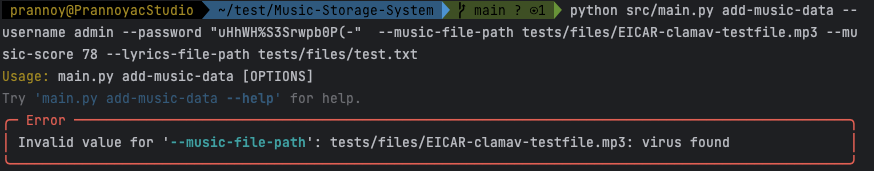
Large files:

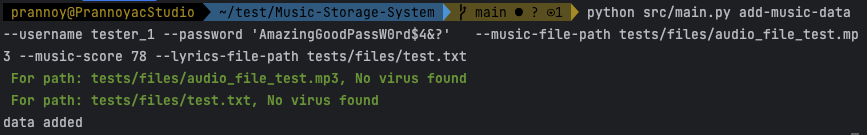


Extensions denied:



A sample file provided by ClamAV to simulate a present virus is tested to ensure its working, below screenshot is the ECIAR file test:

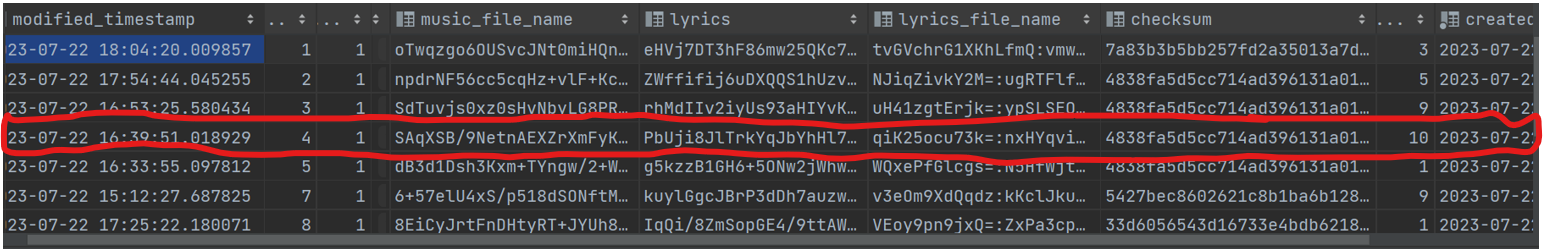
Screenshot below is a successful scan with no viruses:



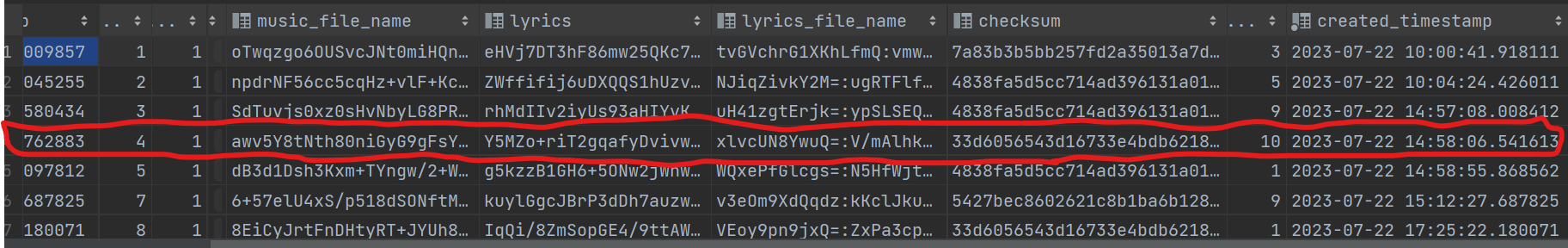
**Calculating checksum for Data integrity:**

According to OWASP (**A08: Data Integrity Failures**), checksums are employed to guarantee data integrity during transmission and storage. As shown in the example below, each time a user updates the music data, the file name, lyrics, and checksum are updated to maintain data integrity.

The highlighted section below shows timestamp, music file name, lyrics and checksum before updating the music entry.



The highlighted section below shows changes in timestamp, music file name, lyrics and checksum after updating the music entry.



# Automated Code Testing

### Ruff – Python Code Linter (to comply with PEP8 coding guidelines)

The command **‘ruff check src\.’** is initiated to scan all python files in Music-Storage-Application directory.



To remediate errors, the command **'ruff check –fix'** corrected errors; some have been corrected by manual coding. After remediation, the test results achieved 0 errors after re-running the Ruff linting command.

### Radon – Python Code for metrics to measure code Cyclomatic Complexity

The cyclomatic complexity of the code has been calculated using the **‘radon cc -a src’** command, where it –a provides the result of the average complexity at the end of the report. At the same time, ‘**src’** is the directory where the codes are located. As per radon, the following is the scale of complexity:

**A:** Cyclomatic Complexity between 1 and 5 (very low complexity)

**B:** Cyclomatic Complexity between 6 and 10 (low complexity)

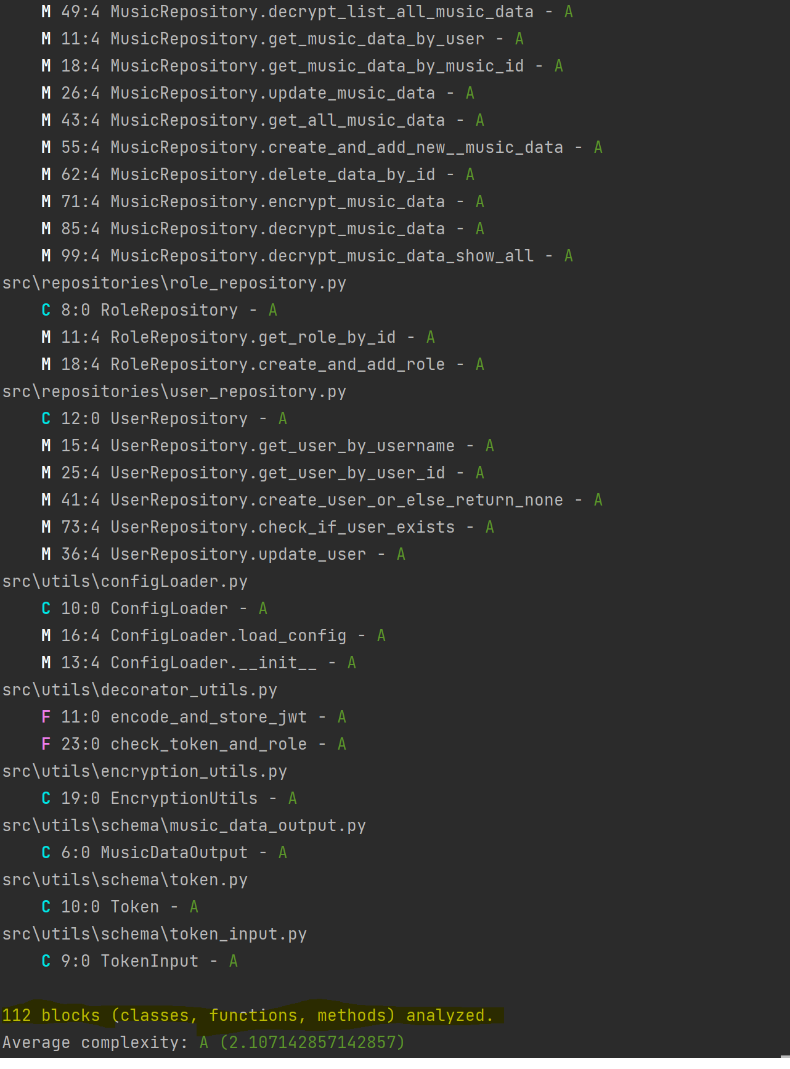
**C:** Cyclomatic Complexity between 11 and 20 (moderate complexity)

**D:** Cyclomatic Complexity between 21 and 30 (more than moderate complexity)

**E:** Cyclomatic Complexity between 31 and 40 (high complexity)

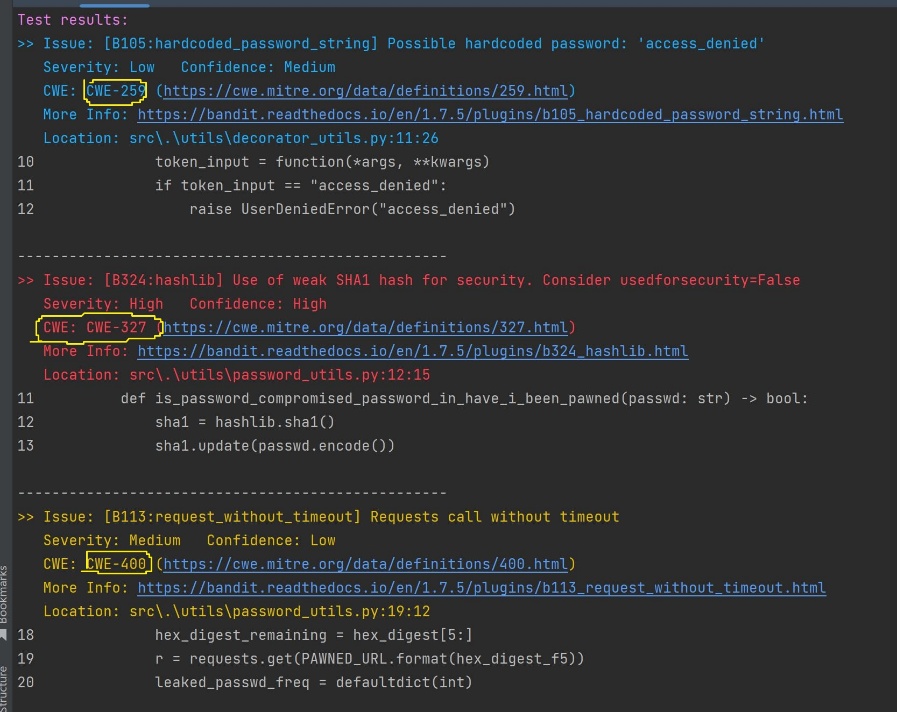
**F:** Cyclomatic Complexity over 40 (very high complexity)

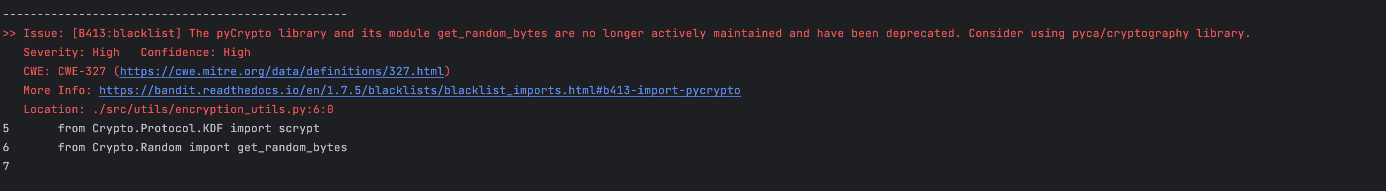
The screenshot below is the result of overall Cyclomatic Complexity scale of the Music-Storage-System, in which it was scaled at **A (2.107142857142857)**.



### Bandit – Python Code Security Linter

The **'bandit -r -l src\.'** command directs the Bandit security tool to inspect the Python files in the **‘src’** directory and its subdirectories for common security issues. The **-r** flag triggers recursive scanning, while **-l** instructs bandit to list the IDs of the security tests in the output. A screenshot is below, followed by a table for reviews concerning OWASP and its CWE.



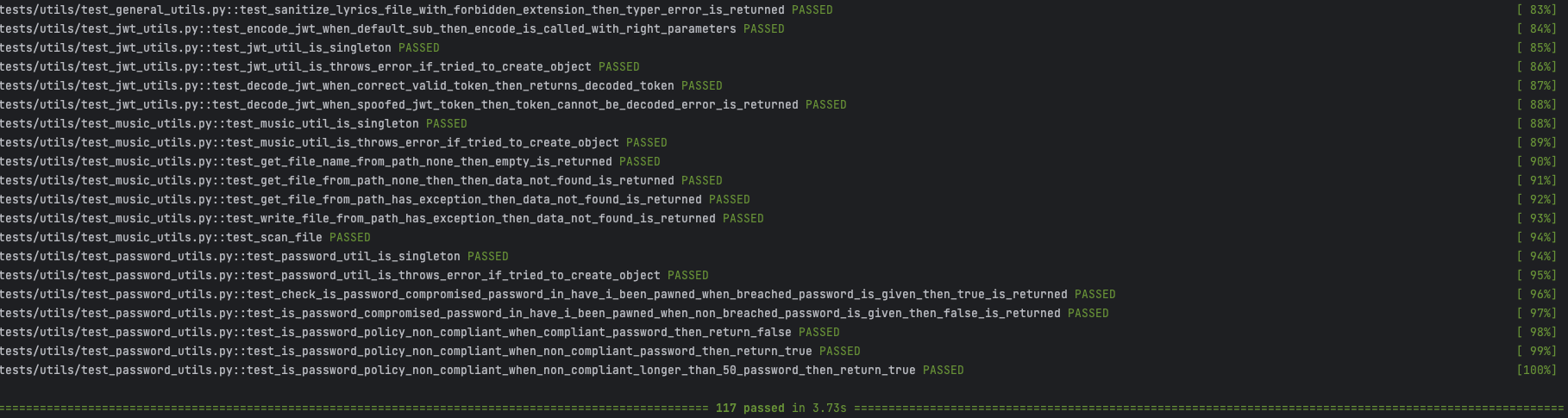


**Vulnerabilities identified by the bandit test**

|  |  |  |  |
| --- | --- | --- | --- |
| Vulnerabilities | Framework | Severity | Mitigations Applied |
| Authentication failure due to hard-coded password | CWE – 259  A07 – Identification and authentication failure - OWASP, 2021 | Medium | * Store passwords in protected and encrypted file or database. * Unique strong passwords. * Access control checks (CWE, 2023). |
| Cryptographic failures | CWE – 327  A02 – Cryptographic failures – OWASP, 2021 | High | * Cipher text and encrypted passwords. * Strong encryption method or Algorithm. * Use lengthy salts for sensitive data (Omar, 2022). |
| Uncontrolled resource consumption | CWE – 400  A9 – Denial of service – OWASP, 2004 | Low | * Strong authentication and access controls. * Protect login application. * Limit database access. |

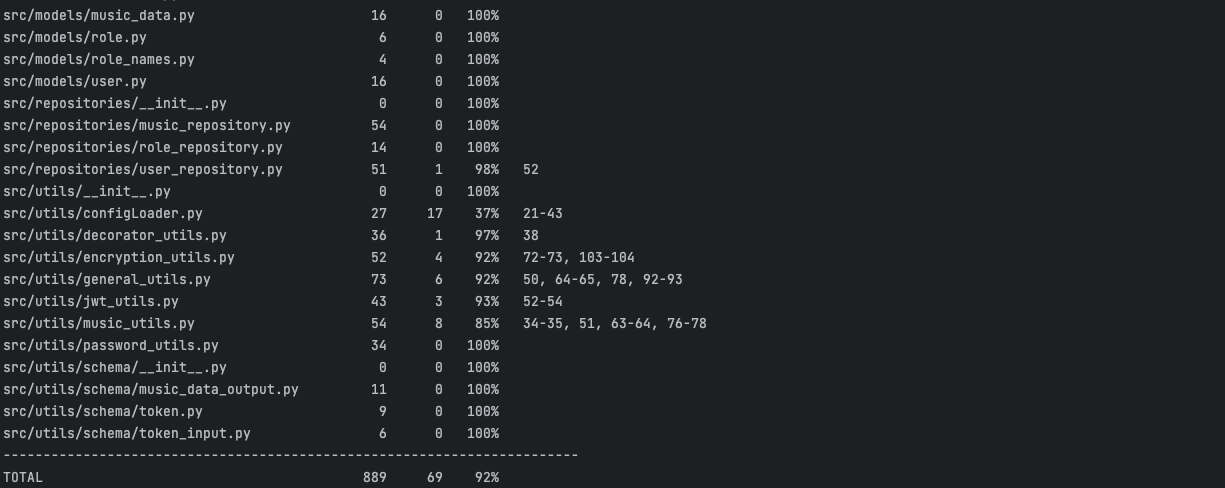
### PyTest – Automated Testing for code errors

The command ‘**coverage run** **--source=src -m pytest -v tests’** initiates the Coverage.py tool to measure the code coverage of Python programs, specifically within the **'src'** directory. The screenshot below is the test performed using PyTest.



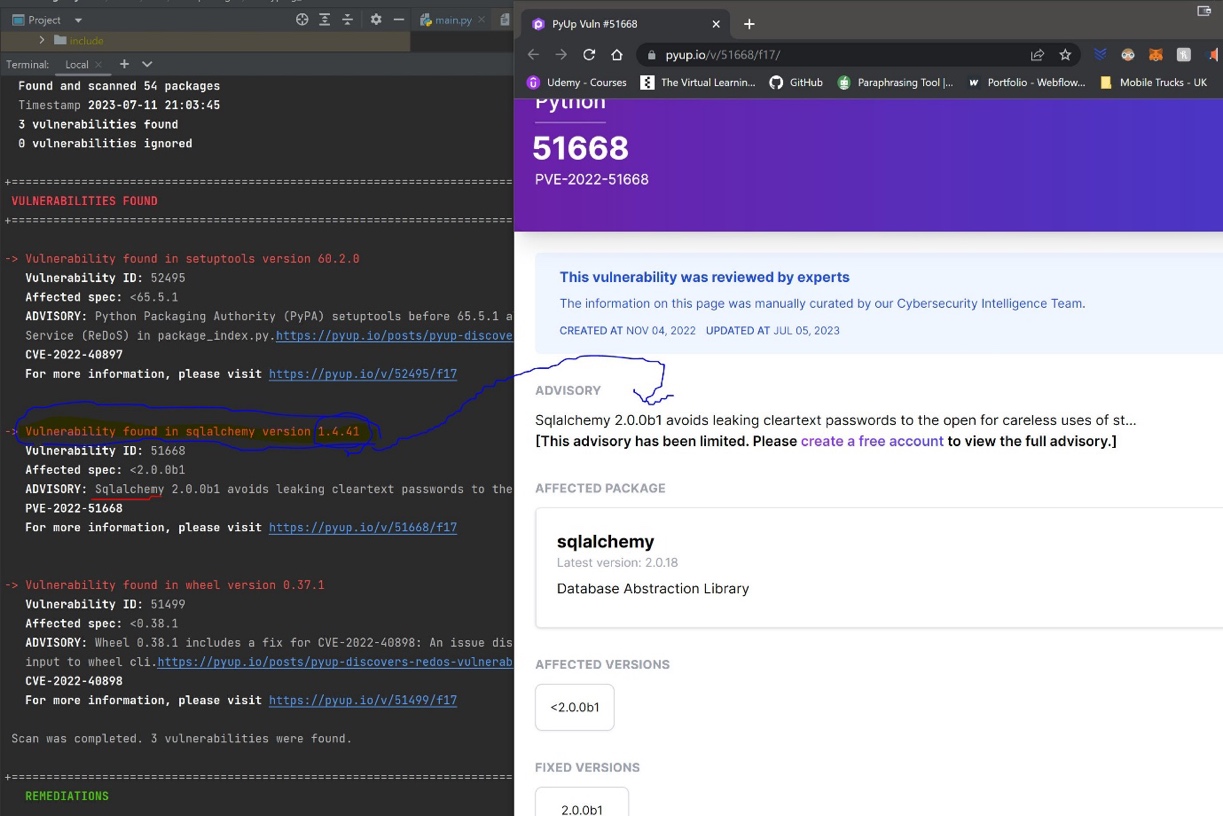
The command **‘coverage report -m'** generates a detailed coverage report, listing untested parts of the code. The **-m** flag specifies inclusion of line numbers for the missed code statements.

The overall test coverage for the application is **92%**, with 69 out of 889 statements not covered by the tests.



1. Safety – Libraries Vulnerability Scanning

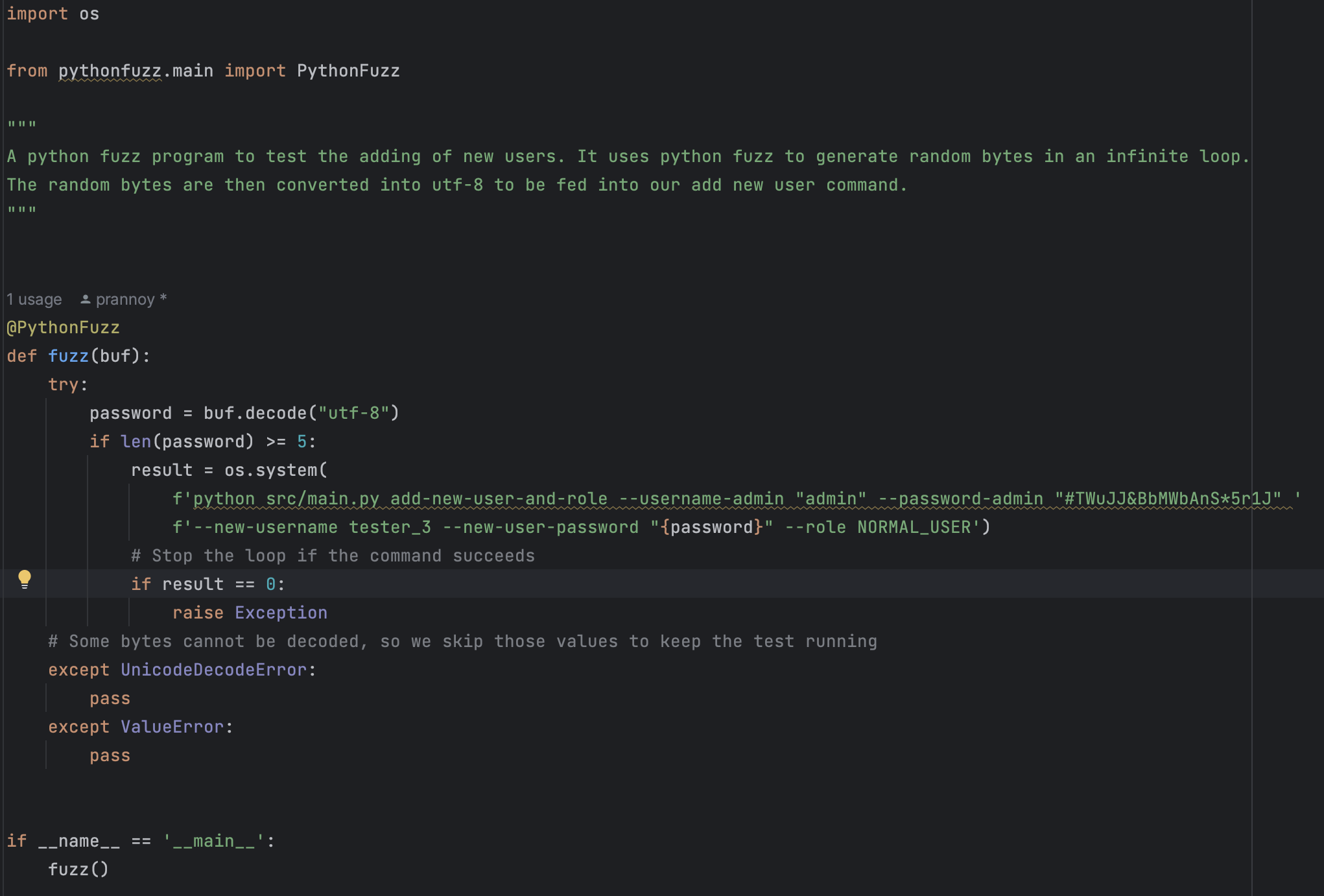
The **safety check --full-report** command is used with the Safety Python package, a tool that checks a project's installed dependencies for known security vulnerabilities. The **--full-report** option provides a complete rundown of the safety check, detailing all the installed packages and the specific vulnerabilities identified.

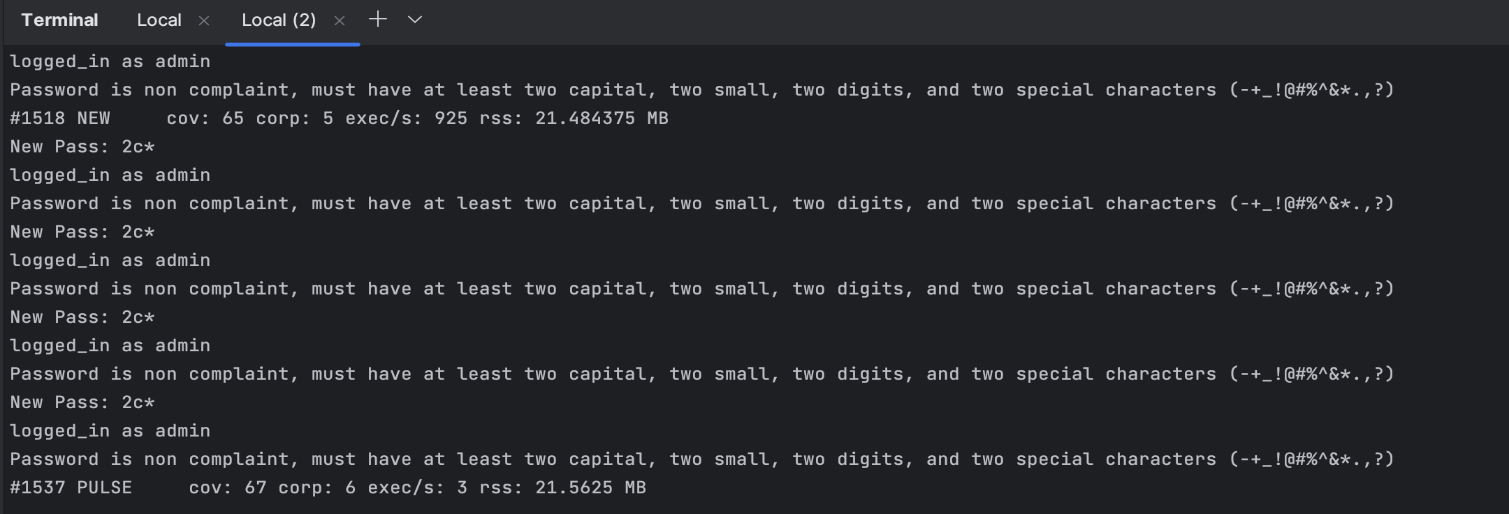


**Vulnerabilities identified by safety:**

|  |  |  |  |
| --- | --- | --- | --- |
| Vulnerabilities | Framework | Severity | Mitigations Applied |
| Denial of service (ReDoS) | CVE 2022-40897  Common Vulnerabilities exposure | Medium | Update to version 65.5.1 |
| Plain text passwords  Database abstraction library | PVE 2022-51668 | High | Update SQL Alchemy to version 2.0.0b1 |
| Denial of service attacks | CVE 2022-40898 | High | Apply patch for version 0.38.1 fix |

### Pythonfuzz





A fuzz test was performed to add new users with roles where a random string that the fuzz library generates is inputted. This fuzz test discovered two issues that were present in the application.

|  |  |
| --- | --- |
| Issue | Remediation |
| The regex, which was checking the password policy, allowed certain passwords that were not supposed to be compliant (2 digits, 2 Uppercase, 2 Lowercase, 2 specials) | The problem arose as the regex was evaluated using the function re.match(), which only ensured that the string begins with the pattern rather than the whole pattern being checked. Using the re.fullmatch() function solved this issue. |
| The application does not work if there is a token config file and the token is not present. | There was no check when the file was present if the Token field was also present in the file. Therefore, the parsing of the field broke. Now error handling when no field is added. |

# Future Improvements:

* Replacing the numeric Id value with a Universally Unique Identifier (UUID) is essential to mitigate enumeration attacks. The risk of attackers guessing valid numeric IDs is significantly higher than UUIDs, 128-bit random values with an extremely low likelihood of guessing. Utilizing UUIDs instead of numeric values enhances security and prevents unauthorized access attempts, making it exceptionally difficult for attackers to infer or guess valid user identities (Kusum, 2023).

* To enhance security, it is essential to validate audio files for authenticity beyond relying on their extensions. Attackers can easily disguise malicious content by spoofing or manipulating file extensions. Therefore, it is recommended to implement file validity to ensure the legitimacy of the audio files (logsign, 2021).

* By employing more threads to handle encryption and decryption tasks, By employing more threads to handle encryption and decryption tasks, the application can take advantage of multi-core processors and parallel processing capabilities. As a result, the encryption and decryption operations can be completed more quickly, improving overall application performance (Shadi et al., 2018).
* To fully comply with the GDPR rights of users, the option to delete accounts is required. However, due to time constraints, the feature has yet to be implemented.

* Fuzz Testing has been conducted with adding new user function only, and due to the complexity of use cases in Fuzz Testing, it required additional time to perform it further on other sections of the application.

* To protect the private key from unauthorized access, it is essential to store it securely through a secret manager to store and manage sensitive data, such as passwords, API keys, and private cryptographic keys, in a secure and controlled manner (Shadi, Muneer & We’am, 2018).

* Key rotation enhances overall security and ensures compliance with security standards. Its implementation reduces the risk of compromised keys being used to decrypt encrypted data and limits the potential impact of any critical breaches (Cryptomathic, 2020).

# Conclusion

In conclusion, the command-line application introduced in this report offers a music storage system, enabling users to effortlessly manage their entries. By focusing on the prevention of copyright infringements, this application serves as a robust platform, ensuring data integrity and safeguarding against unauthorized use of copyrighted content by implementing strong security measures.

# References

Bandit (n.d.) Bandit - Python Security Linter. Available from: https://bandit.readthedocs.io/en/latest/ [Accessed: 18 July 2023].

Biryukov, A., Dinu, D., & Khovratovich, D. (2021). The Memory-Hard Argon2 Password Hash Function. RFC 9106. IETF. Available from: https://datatracker.ietf.org/doc/rfc9106/ [Accessed 18 July 2023].

Biryukov, A., Dinu, D. and Khovratovich, D., 2016, March. Argon2: new generation of memory-hard functions for password hashing and other applications. In 2016 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 292-302). IEEE.

Bisheh-Niasar, M., Azarderakhsh, R. and Mozaffari-Kermani, M., 2021. Cryptographic accelerators for digital signature based on Ed25519. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 29(7):1297-1305.

Chaturvedi, A., 2022, June. Comparison of Different Authentication Techniques and Steps to Implement Robust JWT Authentication. In 2022 7th International Conference on Communication and Electronics Systems (ICCES) (pp. 772-779). IEEE.

Cryptomathic, 2020. Symmetric Cryptography and Key Management: Key Exhaustion, Rotation and Security Models. Available from: https://www.cryptomathic.com/news-events/blog/symmetric-cryptography-and-key-management-considerations-on-key-exhaustion-rotation-and-security-models [Accessed 22 July 2023].

CWE (2023). CWE-259: Use of Hard Coded Password. Available from: https://cwe.mitre.org/data/definitions/259.html [Accessed 14 July 2023].

CWE (2023). CWE-327: Use of Broken or Risky Cryptographic Algorithm. Available from: https://cwe.mitre.org/data/definitions/327.html [Accessed 14 July 2023].

CWE (2023). CWE-400: Uncontrolled Resource Consumption. Available from: https://cwe.mitre.org/data/definitions/400.html [Accessed 14 July 2023].

Ed, P. (2017). Sanitizing Inputs: Avoiding Security and Usability Disaster. Available from: https://www.sqlshack.com/sanitizing-inputs-avoiding-security-usability-disasters/ [Accessed 14 July 2023].

GitLab. (n.d.) 'PythonFuzz', GitLab Security Products Analyzers Fuzzers. Available at: https://gitlab.com/gitlab-org/security-products/analyzers/fuzzers/pythonfuzz [Accessed 22 July 2023].

Harley, D., Myers, L. & Willems, E., 2011. Test Files and Product Evaluation: the Case for and against Malware Simulation. In AVAR Conference.

Kusum, J. (2023). UUID in Python: How to Generate random IDs? (with code). Available from: https://favtutor.com/blogs/uuid-python [Accessed 22 July 2023].

Logsign. (2019). How to Check the Integrity of a File? Available from: https://www.logsign.com/blog/how-to-check-the-integrity-of-a-file/ [Accessed 22 July 2023].

"Radon" (n.d.) Radon. Available from: https://radon.readthedocs.io [Accessed 18 July 2023].

SQLite. (n.d.). About SQLite. Available from: https://www.sqlite.org/about.html [Accessed 18 July 2023].

Van, R. G. and Drake, F. L. (2001). 'PEP 8 -- Style Guide for Python Code'. Available from: https://peps.python.org/pep-0008/ [Accessed18 July 2023].

Ruff. (n.d.). Documentation. Available from: https://beta.ruff.rs/docs/ [Accessed 18 July 2023].

Safety. (n.d.). PyUp. Available from: https://pyup.io/safety/ [Accessed: 18 July 2023].

Shadi, A., Muneer, B.Y. & We’am, A.T. (2018). A Multithreaded Programming Approach for Multimedia Big Data: Encryption System, Multimedia Tools and Application, 77(6), pp.10997-11016.

OWASP. (2021). OWASP Top 10. Available from: https://owasp.org/www-project-top-ten/ [Accessed 14 July 2023].

OWASP. (n.d). 'Unrestricted File Upload', OWASP. Available from: https://owasp.org/www-community/vulnerabilities/Unrestricted\_File\_Upload [Accessed 22 July 2023].

OWASP. (2021). 'Authentication General Guidelines', OWASP Cheat Sheet Series. Available from: https://cheatsheetseries.owasp.org/cheatsheets/Authentication\_Cheat\_Sheet.html [Accessed 22 July 2023].

Omar. H. (2022). Introduction to Cryptographic Failures. Available from: https://www.softwaresecured.com/introduction-to-cryptographic-failures/ [Accessed 19 July 2023].

OWASP. (2023). Password Storage Cheat Sheet. OWASP Cheat Sheet Series. Available from: https://cheatsheetseries.owasp.org/cheatsheets/Password\_Storage\_Cheat\_Sheet.html [Accessed 18 July 2023].

Pal at el. 2022. Might i get pwned: A second generation compromised credential checking service. In 31st USENIX Security Symposium (USENIX Security 22) (pp. 1831-1848).

Rachmawati, D., Tarigan, J.T. & Ginting, A.B.C., (2018). A comparative study of Message Digest 5 (MD5) and SHA256 algorithm. In Journal of Physics: Conference Series, 978, 012116.

Pytest (n.d.). The pytest framework makes it easy to write small tests, yet scales to support complex functional testing. Available from: https://docs.pytest.org/en/latest/ [Accessed 18 July 2023].

Typer (n.d.). Typer - 0.9.0. Available from: https://typer.tiangolo.com/ [Accessed 18 July 2023].