API Systems

Task 1 Research Document

Louisann Borg
BA IDM 6.1

1. Introduction

In today's music industry, streaming platforms and recommendation systems play a large role in discovering new artists and albums. However, metal music fans frequently have difficulty using mainstream music streaming services due to insufficient subgenre-specific filtering, inaccurate metadata, and poor discovery algorithms. Most major platforms, including Spotify, Apple Music, and Deezer, use generic recommendation models that fail to distinguish between the many different metal subgenres.

Metalurgy is an API that aims to transform metal music discovery by providing a genre-specific, AI-based recommendation engine. This API will be useful for music streaming services, independent developers, bands, and metal fans looking for new ways to discover and interact with metal music. This document delves into the research behind RESTful APIs, authorization mechanisms, and security considerations before presenting the Metalurgy API concept in detail.

---

2. RESTful APIs

2.1 What is a RESTful API?

A RESTful API (Representational State Transfer) is an architectural style that facilitates communication between clients (such as web applications) and servers. RESTful APIs are built on standard HTTP methods and stick to stateless principles, making them scalable and easy to integrate across platforms.

2.2 Core RESTful API Principles

- ◊ **Client-Server Architecture***:* Separates front-end applications from the back-end API, allowing for more flexible integration.
- ◊ **Cacheability**: Responses can be cached for improved performance.
- ◊ **Statelessness**: Every API request has to contain all the necessary information, as servers do not store client session data.
- ◊ **Layered Systems**: Multiple layers between the client and the server can exist, these include load balancers or security layers.
- ◊ **Uniform Interface**: This ensures consistent communication using standard HTTP methods and JSON formatting.

2.3 How do API requests work?

When a client (e.g. Spotify) requests data from an API the following process occurs;

- ◊ The client sends an HTTP request (e.g. GET bands?genre=death_metal)
- ◊ The API processes the request, retrieving data from it’s database.
- ◊ The API sends a response in JSON format with the requested information.
- ◊ The client displays the data (e.g. listing bands, albums etc)

2.4 HTTP Methods

◊ **GET**: This is used to retrieve data from the API.
◊ **POST**: This is used to submit new data to the API.
◊ **PUT**: This is used to update existing data within API.
◊ **PATCH**: This is used to partially update an existing resource.
◊ **DELETE**: This is used to remove data from the API.

---

3. Authorization & OAuth 2.0

3.1 *What is OAuth 2.0*

OAuth 2.0 is an industry-standard protocol for securing API access without disclosing user passwords. It allows users to grant restricted access to their accounts on external services without sharing credentials.

3.2 *Benefits of OAuth 2.0*

◊ **Secure Authentication**: Prevents unauthorized API usage.
◊ **Token-Based Access**: Users receive an access token with an expiration period.
◊ **Third-Party Integration**: Allows to connect with services like Spotify etc.

3.3 *OAuth 2.0 Components*

◊ **Scopes**: Define the level of access (e.g. read_bands etc.)
◊ **Access Tokens**: Short-lived tokens that authenticate API requests.
◊ **Client ID & Client Secret**: Credentials issued to third-party applications using the API.

---

4. API Security Considerations

4.1 *Overview of OWASP API Security Top 10*

The OWASP API Security Top Ten is a comprehensive list of the most serious security threats that modern APIs face. These flaws can result in serious consequences such as data breaches, unauthorised access, and service disruptions. APIs continue to play an important role in connecting applications and facilitating data exchange, so developers, security professionals, and organisations must be aware of the risks. To protect sensitive information and prevent cyber threats, API designers must follow industry best practices such as strong authentication and authorisation mechanisms, validating user inputs, and securing data in transit and at rest. Businesses can improve API resilience and protect themselves from potential attacks by proactively addressing these security concerns.

4.2 *Two Major Security Threats & Mitigation Strategies*

1. **Broken Object Level Authorization (BOLA)**

   ◊ **Threat**: Attackers could manipulate API requests to gain access to or modify other users' data.
   ◊ **Solution**: Implement role-based access control (RBAC) to limit access to specific actions.

2. **Security Misconfiguration**

   ◊ **Threat**: API default settings or misconfigured endpoints may expose confidential data.
   ◊ **Solution**
      - Enforce **HTTPS** encryption.
      - Store API keys as **environment variables** instead of hardcoding them.
      - Disable debug endpoints in production environments.

---

5. Proposed API Concept: Metalurgy

5.1 *Overview*

Metalurgy is an API that enhances the discovery and classification of metal music. Unlike other APIs, Metalurgy offers deep metadata tagging, AI-powered recommendations, and community-driven insights.

5.2 *Key Features*

◊ **Band & Album Search**: Helps discover new artists by genre, country, or popularity
◊ **Subgenre Classification**: Categorizes bands based on niche metal styles (e.g. Blackened Death Metal, Technical Trash)
◊ **AI-Powered Recommendations**: Suggest artists based on user behaviour.
◊ **Community Driven Tagging**: The user contributes metadata and genre labels.

5.3 *Unique Selling Points (USPs)*

◊ **Genre-Specific Accuracy**: Unlike Spotify, Metalurgy focuses solely on metal subgenres.
◊ **Developer-Friendly**: RESTful API with clear documentation.
◊ **AI-Powered Personalization**: Uses machine learning to refine recommendations.
◊ **Open-Source Contribution**: Community-driven improvements to API datasets.

5.4 *Target Audience*

◊ **Metal Music Fans**: Looking for better discovery options.
◊ **Indie Bands and Labels**: Seeking exposure for underground artists.
◊ **Developers**: Creating bots, websites, or streaming applications.

5.5 *Example API Requests*

1. **Retrieve Bands by Sub-Genres**

   ◊ GET / bands?genre=doom_metal

2. **Fetch Recent Album Releases**

   ◊ GET /albums?year=2024

3. **Post a new album review**

   ◊ POST /reviews { “albums”: “Ghost Reveries” , “rating”: 9}

5.6 *Implementation Plan*

1. **Phase 1: Research and Planning**

◊ Conduct market analysis on metal music discovery gaps.
◊ Define API endpoints, authentication mechanisms, and security measures.

2. **Phase 2: Development**

◊ Build a prototype of the Metalurgy API.
◊ Implement OAuth 2.0 authentication and API security measures.
◊ Integrate AI-powered recommendation features.

3. **Phase 3: Testing & Refinement**

◊ Conduct beta testing with early adopters (metal music fans, developers, and bands).
◊ Gather feedback on usability, speed, and accuracy of recommendations.
◊ Optimize API response times and improve metadata categorization.

5.7 *API Workflow*

1. **Client App (User Interaction)** – The user interacts with Spotify, Bandcamp, or a Discord bot.
2. **API Request** – The app sends a request, e.g., GET /recommendations?genre=doom.
3. **Metalurgy API (Processing)** – The API filters and processes the request.
4. **Database & AI Engine** – It retrieves metadata, applies AI-driven filtering, and refines recommendations.
5. **API Response** – The API sends a JSON response (e.g., {band: ‘Gojira})

---

6. Conclusion

Metalurgy will bridge the gap in metal music discovery by providing a RESTful API with secure OAuth 2.0 authentication and strong security measures. This API, which uses AI-powered recommendations and community-driven metadata, will empower developers,

bands, and fans alike. The following steps include development and testing to ensure a consistent user experience.

References

- ◊ Fielding, R.T. (2000) *Architectural styles and the design of network-based software architectures*. University of California, Irvine.
- ◊ Hardt, D. (2012) *The OAuth 2.0 Authorization Framework*. Internet Engineering Task Force (IETF). Available at: https://tools.ietf.org/html/rfc6749 (Accessed: 10 March 2024).
- ◊ OWASP (2023) *OWASP API Security Top 10*. Open Web Application Security Project. Available at: https://owasp.org/www-project-api-security/ (Accessed: 10 March 2024).
- ◊ Richardson, L. and Ruby, S. (2007) *RESTful Web Services*. O'Reilly Media.