Marcus Mallia
API Task 1

SpeakeasySounds API – Research and Planning Document

Introduction

This document outlines the research and planning process behind the development of a RESTful API for SpeakeasySounds, a community-driven platform that allows users to upload, share, and discover music mixes. Originally built as a PHP web application, the system will now be adapted into a modern API structure capable of serving multiple frontends, including a public-facing website, an administrative dashboard, and a future mobile application or game interface.

As user experiences become increasingly decentralized across devices and platforms, having a reliable and well-designed API is essential for maintaining consistency, security, and scalability. The goal of this API is to expose core functionalities of the original system such as user management, mix uploads, and content discovery through a standardized set of endpoints that can be easily consumed by external systems.

The research presented here focuses on three core areas that are essential when planning and designing an API system. Firstly, it explores the concept of RESTful APIs, detailing their architectural constraints and design principles. Secondly, it investigates OAuth 2.0 as a widely adopted authorization standard, examining how it can be implemented to secure API endpoints. Lastly, the document highlights potential security threats identified in the OWASP API Security Top 10 and outlines measures to mitigate those risks within the SpeakeasySounds API.

Part 1 – RESTful APIs

A RESTful API, or Representational State Transfer API, is a style of software architecture that allows different applications or platforms to communicate over the web using standard HTTP methods. It acts as a bridge between a client such as a web application, mobile app, or even a video game and the server where all the data and business logic are stored.

At its core, a RESTful API follows a number of guiding principles that make it simple, scalable, and consistent. One of the key principles is the client-server model, which separates the user interface (client) from the backend system (server). This separation allows different front ends to be developed independently while relying on the same backend API for data and logic.

Another important concept is statelessness. In a RESTful system, each API request from the client must contain all the information needed for the server to understand and fulfill it. The server does not store anything about the user session between requests. This improves scalability because the server can handle each request independently, without having to remember previous interactions.

RESTful APIs also embrace cacheability, which allows certain responses to be stored temporarily (cached) either by the client or by intermediaries like proxies. For example, a list of available music genres or trending mixes could be cached to reduce server load and improve response times.

In addition, a RESTful API enforces a uniform interface, meaning that it uses a consistent structure for all requests and responses. Typically, data is sent and received in JSON format, which is both lightweight and easy to parse across different platforms. REST also defines a clear way to structure URLs and actions using standard HTTP methods like GET, POST, PUT, PATCH, and DELETE.

The request-response cycle is fundamental to how REST works. A client sends a request to a specific URL (called an endpoint), optionally including parameters or a request body. The server processes the request and returns a response, usually in JSON, along with an appropriate HTTP status code to indicate success, failure, or errors.

In some cases, query string parameters are used to filter or sort the data in a GET request. For instance, /api/mixes?genre=house&sort=latest might be used to fetch the latest House mixes. On the other hand, request bodies are used primarily in POST, PUT, or PATCH requests to send data to the server, such as new mix information or updated user profile details.

The API we are designing will serve as the backbone for the SpeakeasySounds system, a music-sharing platform that supports a public-facing website, an admin dashboard, and a potential mobile application. Using the RESTful principles outlined above, we will expose the core functionalities of the system as clearly structured endpoints. For example, /api/mixes will be used to retrieve or submit user-generated music mixes. The GET method will allow clients to fetch lists of available content, while POST will be used to create new entries. Endpoints like /api/users/{id} will return user data, and /api/comments/{id} can be used to manage user comments.

Each request will be stateless — clients will authenticate using tokens (as explained in the next section), and all necessary data will be included in the request. Responses will follow a consistent JSON structure, and some endpoints may support cacheability for improved performance. By following RESTful architecture, the SpeakeasySounds API will provide a consistent and reliable interface for multiple frontends, ensuring that the system is scalable, flexible, and easy for developers to integrate with.

Part 2 – Authorisation

Security is a critical part of any API, especially when user data and content are involved. One of the most widely used methods of securing APIs today is OAuth 2.0, an industry-standard protocol for authorization. OAuth 2.0 allows third-party applications to obtain limited access to a user's resources without exposing the user's credentials.

At a basic level, OAuth 2.0 separates authentication (verifying a user's identity) from authorization (granting permission to access resources). Instead of logging in directly with a username and password on every request, the client obtains an access token from the authorization server. This token is then attached to API requests to prove the client has permission to access certain resources.

Scopes define what access the token provides. For example, one token may allow a client to upload mixes, while another only allows it to view mixes. Access tokens are usually short-lived and must be included in the Authorization header when making requests to protected endpoints. When a third-party app wants to access your API, it registers with the API provider and receives a unique Client ID and Client Secret to identify and authenticate itself.

In the SpeakeasySounds API, OAuth 2.0 will be used to protect sensitive endpoints such as posting content, editing profiles, or managing comments. When a user logs in, their credentials will be sent to an authorization endpoint like POST /api/auth/login. If the login is successful, the server returns an access token. This token will then be attached to future requests as a Bearer token in the Authorization header. The API will verify this token and allow access based on the user's role and scope.

Using OAuth 2.0 ensures that the SpeakeasySounds API is both secure and flexible. It protects user data from unauthorized access and provides a standardized system for managing access across multiple platforms like the website, admin dashboard, and mobile app.

Part 3 – Security Considerations

As APIs become a central part of modern applications, they also become prime targets for attackers. The OWASP API Security Top 10 is a globally recognized list of the most common and dangerous security risks for APIs. Understanding these risks is essential for building secure and reliable API systems.

Below is a summary of the most recent OWASP API Security Top 10 vulnerabilities:

1. Broken Object Level Authorization (BOLA) – Occurs when APIs do not properly verify if a user is allowed to access or manipulate a specific object (e.g., another user's data).
2. Broken Authentication – Weak or improperly implemented authentication mechanisms that allow attackers to compromise user accounts.
3. Broken Object Property Level Authorization – When users can access or modify sensitive object properties they should not have access to.
4. Unrestricted Resource Consumption – APIs that allow excessive use of system resources (e.g., CPU, memory, database calls), leading to performance issues or crashes.
5. Broken Function Level Authorization – When users can access privileged functionality (like admin actions) due to missing role checks.
6. Unrestricted Access to Sensitive Business Flows – When APIs expose critical business operations without proper access restrictions.
7. Server Side Request Forgery (SSRF) – When attackers manipulate API requests to force the server to make unauthorized requests to internal or external systems.
8. Security Misconfiguration – Default settings, overly permissive CORS policies, or missing TLS/HTTPS can all lead to vulnerabilities.
9. Improper Inventory Management – When older, undocumented, or unused API versions remain online, posing a hidden security risk.
10. Unsafe Consumption of APIs – Failing to validate or sanitize data received from third-party APIs can lead to injection attacks or data leakage.

Two of the most relevant threats to the SpeakeasySounds API are Broken Object Level Authorization (BOLA) and Security Misconfiguration. BOLA occurs when an API does not correctly verify that a user is allowed to access or manipulate a specific resource. For example, a user may try to edit another user's mix by guessing the ID in the URL.

To prevent this, the SpeakeasySounds API will always compare the user ID in the access token with the resource owner's ID in the database before allowing access.

Security Misconfiguration involves leaving default settings, verbose error messages, or insecure ports open to attackers. In the SpeakeasySounds API, HTTPS will be enforced for all connections. CORS settings will be restricted to trusted domains, and environment variables will be used to store sensitive credentials. Error messages returned to users will be generic, while detailed logs will be stored internally for debugging.

By applying the OWASP guidelines and anticipating common attack vectors, the SpeakeasySounds API will be built to safeguard user data and ensure secure interactions across all client applications.

Conclusion

This research document has laid out the theoretical foundation and practical considerations for developing a secure, scalable, and functional API for the SpeakeasySounds platform. Through a clear understanding of RESTful API principles, proper implementation of OAuth 2.0 authorization, and proactive attention to security concerns highlighted by the OWASP API Security Top 10, the proposed API will support a wide range of client systems while maintaining best practices in software development.

The insights and planning covered here will directly inform the technical execution in the upcoming development phase, guiding the structure of the API, shaping its security model, and ensuring that it is capable of supporting the evolving needs of both the system and its users.

Reference List

Auth0 (n.d.) *OAuth 2.0 Authorization Framework Overview*. Available at: https://auth0.com/docs/authenticate/protocols/oauth

Auth0 (n.d.) *What is OAuth 2.0 and what does it do for you?*. Available at: https://auth0.com/intro-to-iam/what-is-oauth-2

AWS (n.d.) *What is RESTful API?*. Available at: https://aws.amazon.com/what-is/restful-api/

Azure Architecture Center (n.d.) *Web API design best practices*. Available at: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

Curity Identity Server (n.d.) *OAuth 2.0 Overview*. Available at: https://curity.io/resources/learn/oauth-overview/

DigitalOcean (2014) *An Introduction to OAuth 2*. Available at: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

Fielding, R.T. (2000) *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis. University of California, Irvine. Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

IBM (n.d.) *What is a REST API?*. Available at: https://www.ibm.com/think/topics/rest-apis

Mozilla Developer Network (n.d.) *Working with REST APIs*. Available at: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction

OWASP Foundation (2023) *OWASP API Security Top 10 – 2023 Edition*. Available at: https://owasp.org/API-Security/editions/2023/en/0x00-header/

Postman (n.d.) *What is a REST API?*. Available at: https://www.postman.com/what-is-rest-api

Postman (n.d.) *What is API Design? Principles & Best Practices*. Available at: https://www.postman.com/api-platform/api-design/