Music360

A 360 DEGREES PERSPECTIVE ON THE VALUE OF MUSIC

Deliverable 2.3

Secure and trusted sharing of music data – version 1

Disclaimer
The Music360 project has received funding from the European Union's Horizon Europe research and innovation action under grant agreement number 101094872.

Version history			
Ver.	Date	Comments/Changes	Author/Reviewer
0.1	05/03/2024	Initial version	Yulu Wang
0.2	24/03/2024	Merged with OAuth 2.0 / OpenID Connect	Ed Green
0.3	26/03/2024	Final draft	Jaap Gordijn
1.0	07/04/2024	Feedback of all partners processed	Jaap Gordijn

Project Acronym	Music360	
Project Title	360 DEGREES PERSPECTIVE ON THE VALUE OF MUSIC	
Project Number	101094872	
Instrument	Research and Innovation Action (RIA)	
Topic	HORIZON-CL2-2022-HERITAGE-01-05	
Project Start Date	01/03/2023	
Project Duration	36 months	
Work Package	WP2 - Standardized, trusted and unified collection of music metadata	
Task	T2.3a Designing and implementing security mechanisms for controlled data access	
Deliverable	D2.3. Secure and trusted sharing of music data – version 1	
Due Date	31/03/2024	
Submission Date	08/04/2024	
Dissemination Level[1]	Public	
Deliverable Responsible	VU	
Version	1.0	
Status	Final	
Author(s)	Yulu Wang	VU
	Ed Green	VU
	Jaap Gordijn	VU
Reviewer(s)	Gonçal Calvo	BMAT
	Bruno Gaminha	GDA
	Giovanni Giachetti	UPV
	Jaco Koen	BMAT
	Pau Riera	BMAT

[1] PU= Public, CO=Confidential, only for members of the consortium (including the Commission Services), CL=Classified, as referred to in Commission Decision 2001/844/EC

Disclaimer/Acknowledgement:

“The project Music360 has received funding from the European Union’s Horizon Europe research and innovation programme under grant agreement No 101094872.

The opinions expressed in this document reflect only the author’s view and in no way reflect the European Commission’s opinions. The European Commission is not responsible for any use that may be made of the information it contains.”

Contents

1 Introduction 6
2 Requirements 7
3 Architecture 10
3.1 Overview 10
3.2 Components 10
3.2.1 Data Providers 10
3.2.2 Client(s) 11
3.2.3 Authorization Server 11
3.2.4 Resource Server 11
3.2.5 Message Bus 11
3.2.6 API Gateway/Reverse Proxy 11
3.2.7 Plugins 11
4 Database 13
4.1 Overview 13
4.1.1 Data model 13
4.1.2 Access control strategies 13
4.1.3 Database Encryption 14
4.2 Privacy computation for second phase 14
4.3 Queries collection 15
4.4 Secure data aggregation and distribution 20
4.5 Scope Verification and Row Level Security 22
5 Data Import 25
6 OAuth 2.0 & OpenID Connect 26
6.1 Overview 26
6.2 Definitions 26
6.2.1 OAuth 2.0 26
6.2.2 OpenID Connect 27
6.3 OAuth Flows & Disambiguation 27
6.3.1 OAuth Grant Disambiguation 27
6.3.2 Router and Aggregation Service Client Disambiguation 28
6.4 Objectives 29
6.4.1 Proximate Objectives 29

6.4.2 Distal Objectives 30
6.5 OAuth Integration in Music360 31
6.5.1 Registration Provisions 31
6.5.2 Discovery Provisions 32
6.5.3 Authorization Request Provisions 33
6.5.4 Authorization Request 34
6.5.5 Access Token Acquisition 37
6.5.6 Access Token Refreshing 39
6.5.7 Querying 40
6.5.8 Token Revocation 40
7 View Applications 42
8 Data Import 43
9 Conclusion 44
10 Standards 45

1 Introduction

This deliverable presents the design of the security system for the Music360 platform. As with many deliverables of the Music360 project, this deliverable has two versions. This first version has the goal to define the security baseline. The second version will focus on advanced use cases, specifically the case that data, or computations based on the data, can be done by untrusted parties.

The fundamental trust assumption for this first version of the Music360 security architecture is that data processors themselves are trusted by everyone. In other words: they behave as agreed and must comply with the GDPR and other relevant European and national legislation.. This can be arranged in multiple ways, for example by legal contracts.

In section 2, we summarize the security requirements, as elicited by the Music360 consortium.

Section 3 provides an overview of the security architecture, plus an explanation of its components.

The Music360 has the capability to host a significant amount of data in a distributed and sometimes replicated manner. Access to this data needs to be secured, and access controls should be delegated to the database management systems as much as possible. There are a number of reasons for this, one of them is that there will be multiple backends using the data, and we do not want to rely on the correct implementation of access controls by backends (as often is the case), but rather on the component that hosts the data. All this is detailed in section 4.

The Music360 platform is primarily a query-only system. However, data must be able to enter the platform.This is the task of the data providers, such as CMOs and BMAT in Music360, as well as project partners who intend to make their data accessible to the various clients which run in the Music360 ecosystem. Security implications of importing data by the data providers are discussed in section 5.

An important part of any security architecture is authentication of users (is someone the person they claim to be) and authorization (what is someone allowed to do). We have chosen to use OAuth 2.0 and OpenID Connect to solve authentication and authorization. This is discussed in detail in section 6.

Applications that use the data must comply with certain security requirements. These requirements are discussed in section 7.

Section 8 presents the conclusions.

2 Requirements

This section summarizes the requirements with respect to security about the Music360 platform. It is based on a series of workshops with the data owners, namely the CMOs and BMAT.

Index	Description
R1	Data custody **OUGHT** to follow the status quo where feasible. Data providers (e.g., CMOs), thus, ought to retain management of the data they currently have custody and control over as to prevent data duplication in domains which are, as of yet, not production tested.
R2	Data providers **OUGHT** to make data in their custody, but owned by other parties, available to the ecosystem, within the reasonable scope of the project's data accessibility requirements.
R3	Data providers **OUGHT** to be empowered to choose which applications (e.g., the dashboard) they serve data to and receive data requests from via a whitelist. N.B., consensus among data providers upon a general whitelist would create the natural boundaries of the Music360 ecosystem but limits its scope until such a whitelist becomes expanded. Such consensus is not necessary but does limit attack vectors.
R4	Data providers **MUST**, where reasonably possible, integrate the software packages provided henceforth with their own systems to make a scope of data in their custody available to the ecosystem where such scope is reasonably required to meet the goals of the research project.
R5	CMOs **MUST** provide means for the following user groups to manage their data: creatives (e.g., artists), venues and policy makers.
R6	Data made available to the ecosystem **MUST** be done so in a secure manner in non-prototype systems (i.e., systems not using mocked data).
R7	Data owner(s) **MUST** be empowered to grant data access to requesting third parties within the ecosystem.
R8	Data owner(s) **MUST** be empowered to revoke data access to third parties.
R9	Data owner(s) **MUST** be warned about the risks of sharing data with third parties in the Music360 ecosystem **PRIOR TO** granting access to such parties.
R10	Data owner(s) **OUGHT** to give their consent and take full liability of the risks associated with data theft from third parties in the Music360 ecosystem.
R11	Access or revocation of data to and from the various parties of the ecosystem **MUST** be done in a timely and consistent manner.
R12	Communication between services in non-prototype systems **MUST** be done over TLS 1.2 or higher.
R13	Provisions against common web attacks (e.g., XSS, CSRF, Injections such as (but not necessarily limited to) SQL Injections, DoS/DDoS) **MUST** be taken by the various components of the ecosystem and documented at a later date in a comprehensive security policy.
R14	Ecosystem database(s), notably, but not limited to, described in D2.1 **OUGHT** to be partitioned in accordance with **R1**.
R15	Ecosystem database(s), notably, but not limited to, described in D2.1 **OUGHT** to be backed up and replicated at a reasonable frequency by the various data providing parties of the ecosystem.
R16	Authorization policies of the ecosystem database(s), notably, but not limited to, described in D2.1 **MUST** be as fine-grained in scope as reasonably possible given the current technological landscape and resource/computation budget.

R17 Ecosystem database(s), notably, but not limited to, described in D2.1 **MUST** be encrypted using cryptographically secure symmetric or asymmetric algorithms with sufficient collision entropy for the lifespan of the ecosystem; e.g., in the case that at year “n” RSA2048 is not predicted to provide sufficient collision resistance, RSA3072 must be phased in at year “n - (some reasonable time period)”.

R18 Ecosystem database(s) **OUGHT** to phase in homomorphic encryption at such a time when it becomes strategically feasible to increase the security guarantees that the Music360 ecosystem can make.

R19 Encrypted ecosystem database(s) **OUGHT** to rotate keys at certain intervals under a strict policy in order to prevent data leaks and generally harden security.

R20 A penetration test **OUGHT** to be taken out by a sufficiently capable actor on a production system with mocked data such that any weaknesses or flaws may be identified and fixed proactively.

R21 Authorization and authentication of the ecosystem **MUST** be stateless and accomplished in accordance with the JWT standard (*RFC 7519, RFC 8725*).

R22 For all data providers in the ecosystem, JWTs generated **MUST** be signed with an asymmetric private key.

R23 For all data providers in the ecosystem, JWTs generated and used by requesting parties **MUST** provide details of the scope of data access, or can be used to derive scope of data access.

R24 All non-prototype services in the ecosystem **MUST** be highly available.

R25 All services implementing or working with standards **MUST** make a best effort to implement best common practices (BCP) where available, moreover, such services **MUST** also be compliant with data privacy and protection legislation within the jurisdictions they are available in (e.g., GDPR).

R26 Data providers **MUST** reach consensus on a policy for user lifecycle management e.g., Annex A ISO/IEC 27001:2013 section 9.2.

R27 Data providers **MUST** make data accessible in a manner compliant with the FAIR principles.

3 Architecture

3.1 Overview

Our predicted architecture to minimally meet the requirements above is depicted in *figure 1* below and may be subject to minor changes depending on currently unforeseeable implementation details. It is composed of five major independent deliverables described in the subsequent sections and is largely centered around implementing the OAuth 2.0 and OpenID Connect protocols to facilitate the secure and distributed sharing of data between data providers, aggregator, resource owners and clients. We provide comprehensive definitions of the components and actors in the subsequent OAuth 2.0 and OpenID Connect section along with the use-case flows which we expect in the ecosystem. Accordingly, this section serves as a minimal high-level overview.

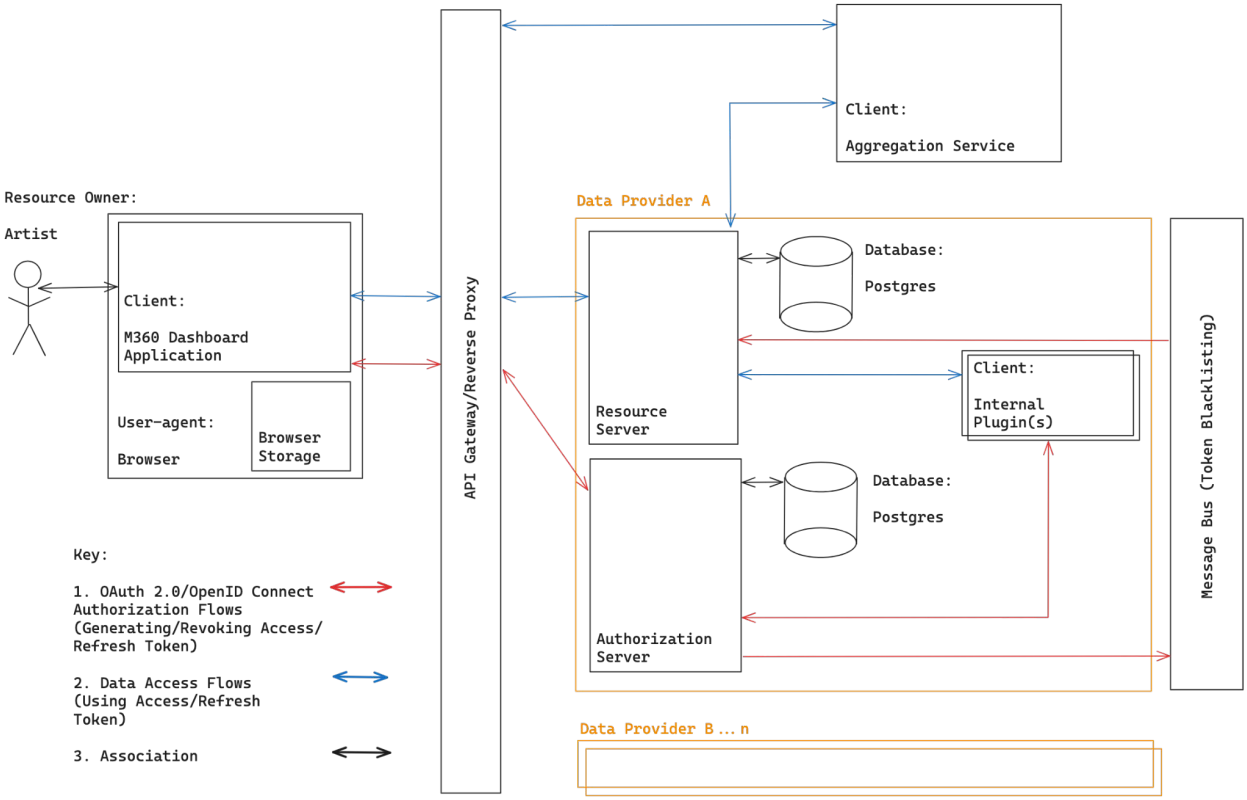

Figure 1. Abstract architecture of security components. N.B., aggregation service and dashboard application are interpreted as a distributed client, see details in OAuth and OpenID section.

3.2 Components

3.2.1 Data Providers

Data providers are a set of, but not limited to, project partners who intend to make their data accessible to the various clients which run in the Music360 ecosystem. Examples being BMAT, various CMOs, etc. We envision that the data providers implement their own OAuth 2.0 services or utilize the deliverables that shall be provided henceforth.

3.2.2 Client(s)

Currently, the Music360 project has two software bundles which may be classified as OAuth clients (defined below), the first being the dashboard application; the second, the aggregation service used to compose complex queries. It is worthy to note that the aggregation androuter services may possibly be considered as a backend feature of the dashboard application (thus the dashboard application becomes a distributed client).

Due to the flexibility of the proposed architecture, it is possible to add new clients to expand the feature-set of the system in part or as a whole; examples may include third-party plugin components which work as part of the dashboard application or as part of the data provider's software bundle.

3.2.3 Authorization Server

The authorization server provides service responsible for issuing access and refreshing tokens to clients on behalf of the resource owners. Each party providing data must offer and maintain a service that allows clients to access the data on their associated resource servers promptly on behalf of the resource owner. Such a server can be bridged with the existing authorization and authentication system of the data provider, acting as an identity provider. Identity provision is discussed in depth in the OAuth 2.0 and OpenID Connect section.

3.2.4 Resource Server

The resource server provides the service(s) responsible for providing the resource owner's data to an authorized client in its desired format. Each party providing data must offer and maintain a service that allows clients to access the data on behalf of the resource owner promptly. Such a server can be bridged with the existing various resource systems of the data provider.

3.2.5 Message Bus

The message bus is an event stream that is easily accessible to various components. In the case of the authorization and resource servers, it is necessary to push and subscribe to the event stream. We mandate the need for the latter services to push and subscribe, respectively, such that access token blacklisting may be expedited (instead of waiting for token expiration).

3.2.6 API Gateway/Reverse Proxy

We eventually envision the possible need for, without defining an exact scope or specification, a reverse proxy may act as an API gateway to and from, either the ecosystem as a whole, or each data provider as a singular unit, in order to coordinate requests and responses in an efficient manner. Furthermore, the service may perform additional security measures, such as limiting the rate of spam requests, verifying tokens, or rejecting invalid requests, to mitigate potential attacks on the data provider(s), the ecosystem, or both.

3.2.7 Plugins

Music360 platform plugins can be seen and organised as multiple functional modules. An important part of these plugins is the user management, session management plugin and aggregation plugin, which is responsible for data security perspective.

- **User Management:** The user management aspect of the plugin encompasses various functionalities aimed at handling user accounts and their associated data securely. This section includes user authentication, registration, profile management, authorization mechanisms, and more. By implementing OAuth2 and scope mechanisms, the platform ensures secure authentication of user accounts, while fine-grained authorization mechanisms at the database level will strictly control access rights.
- **Session Management:** The session management functionality of the plugin is essential for maintaining the integrity and security of user sessions within the Music360 platform. It involves the creation, monitoring and termination of user sessions to regulate access to platform resources and prevent unauthorised sessions from compromising sensitive data. Technologies such as session tokenization, encryption and session expiration mechanisms can ensure that user sessions are protected from attacks and other security threats. A user may have multiple sessions, and a session always corresponds to one user. Sessions represents an authenticated user, is created if the user logs on, and disappears if the user logs off. As the database management system safeguards data access, the session identifier must be created by the database management system.
- **Aggregation plugin:** The Music360 aggregation service plugin will focus on collecting, integrating and organising data from different data owners (CMOs) and processing the data to provide users/stakeholders with comprehensive, unified access to information based on their query requirements.

4 Database

4.1 Overview

For the Music360 platform, formulating its security requirements requires a comprehensive understanding of the domain and the specific risks and challenges associated with it. As the platform involves sensitive data such as assets and copyrights, we would consider the requirements related to data security as follows.

4.1.1 Data model

The data model for the Music Ecosystem Platform's database, the Music360 database, has been specifically described in Deliverable D2.1 - An ontology of the value of music - version 1. Ideally, this database will be partitioned and, due to the general security needs of the database, database backup and replication will also be considered. That is, different platform data providers will host their corresponding Music360 platform data partitions. The partitions may overlap because the database contents and even the organization of the internal data providers are not necessarily completely separated. This setup also allows data owners to maintain maximum ownership of access to their data. Therefore, a unified, semantically agreed Music360 data model is a must in this data architecture design, which facilitates the correlation of related data across different partitions and the aggregation and manipulation of data.

4.1.2 Access control strategies

With the increase in cyber threats and data breaches, ensuring the security and integrity of stored data itself has become critical. The choice of an appropriate authorization mechanism will contribute to the secure storage and access of sensitive data. Access control is fundamental to any security solution. It involves defining who can access what resources and under what circumstances. It also requires a fine grained access control mechanism, which ensures that only authorized users can view, modify, or delete specific data.

For the first version of the secure data architecture, we choose for a database-based strategy for realizing authorization and access control for sensitive data content, and speciallly Row Level Secutity (RLS). RLS policies will be considered to control user access to database viewing privileges, simplify application design and coding, and help enable access control of database row-level data. Some current basic authorization justifications and strategies are listed as follows:

- **Database-based security:** Since the data is stored in the database, if the data is protected by the database itself, the application can only access what is allowed by the database, and security errors in the back-end of the application will not affect the data security. In addition, the query optimizer of the database management system can more efficiently handle the combination of data content tables and tables containing access rights. Finally, pagination (returning a required number of rows rather than all of them) can be easier and more efficiently implemented if the database handles which rows are returned given security constraints. Consequently, we choose to implement data security at the database level as much as possible (and only implement security in backends if it can not be accomplished by the database).
- **Row-Level security (RLS):** A specific way of implementing database-based security is RLS. Each attempt to access data is restricted by RLS, thus ensuring that different users

access different data content. In PostgreSQL,rules enforces Row Security Policies (RSPs) in addition to the permissions that users can obtain through GRANT, the standard permissions system for SQL. Row-level security restricts which rows can be returned by each user's normal queries or modified by data modification commands (INSERT, UPDATE, DELETE).

- **Multiple sources of access control rights:** Access control rights come from multiple sources. First, the Music360 data model itself implicitly defines access controls. For example, an artist who has a claim on recording should at least be able to see the properties of the claim, the recording, and possibly more (e.g. other artists having a claim on the recording). All this information can be found in the Music360 database itself. We refer to this as Domain-based Access Control (DBAC). However, in some cases, it is needed to set access controls based on roles (Role-based Access Control (RBAC)), based on attributes of elements in the Music360 data model (Attribute-based Access Control (ABAC), effectively a generalization of DBAC), and even based on Access Control Lists (ACLs). The latter is needed to allow for granting access to other users than only those based on the Music360 data model, roles, or attributes of data stored.
- **Combine Access Control mechanisms (DBAC, RBAC, ABAC and ACL):** Integrating Domain-based access control, Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC) and Access Control Lists (ACLs) will be applied in an integrated way to find the access rights for RLS.
- **Disadvantage of database-based security:** A disadvantage of database-based security is that the programming language of the chosen database management system should be used, which affects portability to some extent. However, RLS is found in multiple (open source) database management systems, so we consider this not as a significant shortcoming.

4.1.3 Database Encryption

For now, the assumption is that securing a row containing data (e.g. properties or cells) is sufficient. However, in the future, there might be a need to control access to particular properties. Depending on the use case, this can be controlled by security measures in the database, or it may require encryption of the relevant properties,

Also, we assume that the storage provider of the database is trusted by the data owner. However, in the second version of this deliverable, we relax that constraint, and we assume that the storage provider is not necessarily to be trusted.

In sum, in this first version, we do not employ encryption to secure data in the database. This will be revisited in version 2 of this deliverable.

4.2 Privacy computation for second phase

An important principle for Music360 platform data is that the owner of the data always maintains ultimate ownership of that data and will enforce fine-grained access control permissions through authorization mechanisms. However, data contains important insights about the value of music that benefit holders may wish to receive, and these data assets are often only accessible after some data aggregation and computation. Privacy Computing provides a partial solution by allowing relevant stakeholders/platform users to access and obtain the valuable information they need (aggregations or calculations of data) without

revealing the raw data being used. In the second phase, we focus on the application of Secure Data Aggregation (SDA) and related cryptographic computation techniques for privacy data. Ideally, we would build a decentralized and distributed platform to ensure that these computations can be performed without revealing the confidential original data on which they are based, thus addressing the need for stakeholders to share unclassified results (e.g., valuable information about changes in the marketplace for music distribution) with other parties. Therefore, we will compare and apply relevant technologies in the field of privacy computing based on a better understanding of the music ecosystem and the value exchange between its participants. Again, this will be the focus of the second version of this deliverable.

4.3 Queries collection

Obviously, we need to know what data should be protected, and the other way around, which users need to have access to data. Therefore, we have done several workshops with the project partners to find out what queries can be asked, and by whom.

Query collection in the Music360 music ecosystem involves collecting information that different stakeholders/system users want to access. Query collection provides insights into how users interact with the database, including the types of queries they expect to perform, the frequency of use, and the data they access. In this project, the data information/assets that can be accessed and need to be protected are divided into two main categories:

1. Information that can be directly extracted from the distributed database through simple SQL statements according to the schema definitions.
2. Data analytics results that involve multi-party privacy data and ideally need to be obtained by applying appropriate privacy computing techniques.

The system will strictly combine user access requirements and security protection strategies for private data, and further refine the database access control policy based on a thorough investigation of the expectations of different stakeholders.

In the process of collecting queries in the database, prioritizing queries is critical to optimize system resources, ensure timely response and meet user expectations. Certain data information may need to be accessed in a timely manner, while other queries may be less time-sensitive or tolerate longer response times.

We could also use SQL tuning techniques to manage the prioritization process of the queries. For example, define the target tuning queys as the ones with a high level of priority (table partition or temporary tables as a replacement to manage subqueries).

Also, in order to better identify the priority of the collected queries, it is necessary to interact with the stakeholders corresponding to the queries, which can further identify their key needs and preferences from the user's perspective. In addition, certain query prioritization criteria need to be defined as a reference based on factors such as the importance of the data information itself, the frequency of use, the impact on system resources and user expectations. These criteria will guide the prioritization process and help determine the relative importance of each query. We plan to define the priority of the query list with reference to the following aspects based on full consideration of user expectations:

1. **Performance metrics:** performance metrics such as query response time, throughput and resource utilization are considered when prioritizing queries. Queries that are resource-

intensive or have a significant impact on system performance may require higher priority to ensure overall system stability and responsiveness.

2. **Workload Analysis:** Identify queries that are executed frequently, have high resource consumption, or have a significant impact on system load through user requirement reports and possibly historical query logs. Queries that have the highest impact on system performance or are most frequently used by users need to be given a higher priority.
3. **Dynamic Prioritization:** In the second phase, consider implementing a dynamic prioritisation mechanism that adjusts query priorities based on real-time factors such as system load, user activity, and resource availability.
4. **Iteration:** From the second phase, continuously monitor query performance and user feedback to refine the prioritisation strategy based on changing requirements.

Our initial list of user query collection results is shown below from perspectives of different stakeholders:

Table 1: Initial query collection for stakeholder: CMOs

Stakeholder: CMOs			
Query Entity (Access needs)	**Description(scale)[2]**	**Related class (data model)**	**Operation**
Recordings used	Show by location[3]	Recording, MusicWork, Region, Location, MusicVenue, Play, CMOs, Rightholder (eg. producer, performer,...)	C,R,U,D
	Show by timeframe[4]		
	Show by CMO		
	Show by rightsholder's identifier		
	Show by recordings' characteristics[5]		
	Show detailed information about each recording		
Revenues generated[6]	Show by location	Recording, MusicWork, Region, Location, MusicVenue, Play, CMOs, Rightholder (eg. producer,	C,R,U,D
	Show by timeframe		
	Show by CMO[7]		

[2] The detailed information listed in column "Description (scale)" can be considered and demonstrated in combination with multiple dimensions.

[3] Ideally, location here can be detailed as: per city/region/country.

[4] Timeframe here can be detailed as: per day/week/month/year.

[5] Characteristics of recordings/music work could be seen as genre, tempo, language and so on.

[6] According to the general business scope of partner CMOs, revenues we mentioned in these query collection tables mostly refers to royalties that come from neighbouring rights (from recording plays in venues).

[7] CMOs can publicly view the overall revenue that other CMOs (identified by different countries/regions) have generated in a period of time (usually, one year).

	Show by performer	performer,...), Claim, Mandate, Beneficiary	
	Show by music creation name (recording /music work)		
	Show by recordings' characteristics		
	Show by rightsholder's identifier (overall)[8]		
Forecasted usage data (eg. revenue, play counts…) on individual rightsholder/recording's level[9]	Show by location	Recording, MusicWork, Region, Location, Play, Rightholder (eg. producer, performer,...), Claim	R
	Show by timeframe		
	Show by performer		
	Show by music creation name (recording /music work)		
	Show by recordings' characteristics		
	Show by rightsholder's identifier		
Audience profile[10]	Show by venue type	Recording, MusicWork, Region, Location, MusicVenue, VenueType, Play, EndUser	C,R,U,D
	Show by timeframe		
	Show by location		

Table 2: Initial query collection for stakeholder: Rightsholders

Stakeholder: Rightsholders[11]

[8] CMOs can access one specific rightsholder's overall revenue number and the part of revenue number comes from its own datasource. However, CMOs cannot access the detailed distribution of one specific rightsholder's revenue of different countries/regions.

[9] CMOs should not be able to access the forecasted usage data (eg. revenue, play counts…) of other CMOs'.

[10] Possible audience profile could be: gender, age group, customer card information, etc.

[11] Rightsholder scope in Music360 project:

1. The rightsholder can be generally classified into two groups: recording creators (performers, producers) and music work creators (composers, lyricists, publishers).
2. The right regarding recordings is neighbouring right while the right relating to musicworks is copyright in the usual sense.
3. In some CMO context, the Performers also can be addressed as musicians. The role of musicians can be subdivided into main artist, featured artist, un-featured artist and conductors. Composers and lyricists can be known collectively as authors.
4. Heirs is one of a special type of music work rightsholder which means in the event of the author's death, the heirs will be registered and paid for their rights during the 70-year copyright term.

Query Entity (Access needs)	Description(scale)	Related class (data model)	Operation
Personally relevant recordings used	Show by timeframe	Recording, MusicWork, Region, Location, MusicVenue, VenueType, Play, CMOs, Rightholder (eg. producer, performer,...)	R
	Show by location		
	Show by venue (type)		
	Show by recordings' characteristics		
	Show by music creation name (recording /music work)		
Venues (where recording was played)	Show by timeframe	Recording, MusicWork, Region, Location, MusicVenue, VenueType, Play	R
	Show by location		
	Show by music creation name (recording /music work)		
	Show by venue (type)		
	Show by recordings' characteristics		
Most used recordings in whole database (overall)	Show by location	Region, Location, MusicVenue, VenueType, Play	R
	Show by timeframe		
	Show by venue type		
	Show by individual level[12]		
Personal revenues	Show by location	Recording, MusicWork, Region, Location, MusicVenue, VenueType, Play, Claim, Rightsholder (eg. producer, performer,...)	R
	Show by music creation name (recording /music work)		
	Show by venue type		
	Show distribution situation according to different participation types		

5. According to the reality business process in the right clearing process, agents also can be seen as a separate roletype. Generally speaking, the agents will have the same access controls as the multiple direct rightsholders associated with him/her.

[12] For example, one rightsholder can access and get the information of "My top **X** played recordings in **[possible constraints]**".

Forecasted usage data (revenues, play counts)	*Same as above [Revenues]*	Recording, MusicWork, Region, Location, MusicVenue, VenueType, Play, Rightsholder (eg. producer, performer,...), Claim	
Special user query requirement			
As a performer I want to delegate to my agent/heirs the privilege to access my information.		Rightsholder, Agent, Mandate	C,R,U,D

Table 3: Initial query collection for stakeholder: Policy makers

Stakeholder: Policy makers (researchers, government)			
Query Entity (Access needs)	Description(scale)	Related class (data model)	Operation
Recordings used	Show by location	Region, Location, MusicVenue, Play, CMOs	R
	Show by timeframe		
Revenues	Show by location (minimum level: city)	Recording, MusicWork, Region, Location, MusicVenue, ValueType, Play, CMOs, Claim	R
	Show by timeframe		
	Show by venue type		
	Show by recording characteristics		
Most popular[13](played) recording (type)	Show by genre	Recording, MusicWork, Region, Location, MusicVenue, VenueType, Play	R
	Show by lyrics language		
	Show by tempo		
	Show by timeframe		
	Show by location		
	Show by venue (type)		
Audience profile	Show by venue type	Recording, MusicWork, Region, Location, MusicVenue, VenueType, Play, EndUser	R
	Show by timeframe		
	Show by location		
Special user query requirement			
In the hospital scenario, recordings' performance on patients need to be measured and represented in some dimensions with some quantitative criterias.		Recording, MusicWork, Play, EndUser, MusicValue (NoEconomic), RelatedImpact	R

Table 4: Initial query collection for stakeholder: Music Venues

[13] In the initial proposal, the determination of "popular" will be based primarily on the number of times a recording has been played.

Stakeholder: Music Venues			
Query Entity (Access needs)	Description(scale)	Related class (data model)	Operation
Most popular (played) recording (type)	Show by genre	Recording, MusicWork, Region, Location, MusicVenue, VenueType, Play	R
	Show by lyrics language		
	Show by tempo		
	Show by timeframe		
	Show by location		
	Show by venue (type)		
Most played artists	Same as above *[Most popular (played) recording (type)]*	Recording, MusicWork, Region, Location, MusicVenue, VenueType, Play, Rightholder (eg. performer,...)	R
Detailed value indicators[14] for personal venue (e.g. revenue, customer satisfaction, employee satisfaction)	Show by timeframe[15]	Recording, MusicWork, Play, Location, Region, EndUser, MusicValue (Economic), RelatedImpact	R
	Show by concrete location (chain commercial venues)		
Venue playlist[16]	Show by timeframe	Recording, MusicWork, Play, Location, Region, VenuePlayList	R,U
	Show by recordings' characteristics		

4.4 Secure data aggregation and distribution

Data investigation results should be shown precisely and appropriately on the dashboard of the music ecosystem. As mentioned before, the two types of data information should be processed in different approaches. For mostly user management data or information that can be directly extracted from the distributed database, simple SQL statements and the corresponding schema definitions can be applied in the database query language. While some general investigation results from statistics or certain computation which involve multi-party privacy data will ideally need to be obtained by utilizing some appropriate privacy computing techniques.

[14] This query entity is related to the music value from the perspective of venues, certain value models need to be constructed based on the collected music play information and financial situation and the correlation between them will be investigated. During this phase, different levels of a business need to be taken into account, e.g. ICI Paris as a whole, or individual stores.

[15] For a concrete venue that plays music/recordings, music play situations during certain times on a day can be explored with the busyness level.

[16] Venues can choose whether or not to publicly demonstrate their music/recordings playlists.

Sensitive data should be protected from unauthorized parties while still enabling controlled access, as data could contain important insights on topics that stakeholders may want to address to other parties. Computations on data offer a partial solution to this by allowing access to results of aggregations on data without necessarily revealing the data that has been used. During the data aggregation phase, several steps will be deployed:

- **Data collection and transformation:** In the general Extract Transform and Load (ETL) phase, each data provider (CMOs) will extract data from its private database, transform it into a unified Music360 data model and then load it to the Music360 database architecture.
- **Data cleaning:** Collected and transformed data needs to be properly cleansed, which may include deduplication, data integrity assessment, anomaly detection and exclusion, or data classification.
- **Secure data aggregation and information fusion:** Dataset from multiple parties/owners (CMOs) will be aggregated securely with some appropriate privacy computing architecture and homomorphic techniques, some unified views need to be provided. Aggregation may involve statistical calculations, summarization or data merging operations based on predefined queries and criteria. Filtering, sorting and displaying of information will be the most important features of the Music360 platform to satisfy stakeholders' / end users' needs.
- **Performance optimization:** When dealing with data aggregation, consider using caching mechanisms, data indexing and parallel processing techniques. Since the Music360 platform may involve application scenarios with large amounts of data, this ensures efficient data retrieval and processing.

For example, in query scenarios, as shown in the table 5, the query submission and distribution can be categorized into two cases: Simple aggregation SQL queries and Multi-dimensional aggregation SQL queries. The stakeholder/user requesting the data information actually performs the operation of submitting and distributing the aggregated SQL query.

- **Simple aggregation SQL queries:** Assume that the data/information requester u want to query about Q_1 and Q_1 does not involve multi-party data owners/sources, the corresponding aggregate SQL will be submitted according to some mechanism, in the form of "select $\alpha(A_i)$ from R" to the resource servers. R is the denotation of Relation while A can be seen as extracted attributes from R. This query will be distributed to every individual server without any changes.
- **Multi-dimensional aggregation SQL queries:** In the case that the specific query from data/information requester u involves multiple data sources from different data owners. Some specific sharing scheme like Shamir's secret sharing scheme[17] could be utilized, the data owner can split relation R (i.e., including data and associated access control policies) into c sub-relations R_1, R_2, ..., and R_c, and sends each relation to the corresponding resource server. Assume that the submitted aggregation SQL query Q_2 will illustrate in the form of "select $\alpha(A_i)$ from R where$(A_k = C_k)$ OP ... OP $(A_l = C_l)$ ", C_k and C_l are specific query condition values.A set of c secret-shares will be created for each bit of them by the data requester side, where c is the number of resource servers involved. Assume that the p^{th} set of secret-shares for all bits of the values of

[17] https://en.wikipedia.org/wiki/Shamir%27s_secret_sharing, Mirabi, M., & Binnig, C. (2023). QFilter: Towards a Fine-Grained Access Control for Aggregation Query Processing over Secret Shared Data.

C_k and C_l is denoted as $S(C_k)p$ and $S(C_l)p$, respectively. Then, these secret-shares are replaced by the actual values in the query conditions to hide the query pattern from every individual resource server. Thus, the whole query Q_2 will be distributed to each p^{th} server in the form of "select $\alpha(A_i)$ from R where $(A_k = S(C_k)p\ )\ OP \ldots OP\ (A_l = S(C_l)p)$ " .

Table 5: Types of possible aggregation queries[18]

Aggregation SQL Query Type	**Query Format Example**
Simple aggregation SQL queries	select $\alpha(A_i)$ from R.[19]
Multi-dimensional aggregation SQL queries	select $\alpha(A_i)$ from R where $(\ A_k\ =\ C_k\ )\ OP\ \ldots\ OP\ (\ A_l\ =\ C_l\ )$

4.5 Scope Verification and Row Level Security

Row-level security (RLS) controls the viewing privileges of users accessing the database and helps to achieve row-level data access control. Row security policies can be user-specific, role/group-specific, or both. When row security is enabled on a table, each attempt to access the data/row is restricted by a RLS policy, and all access to select or modify rows of the table must be checked for permissions by a corresponding row security policy, which allows different users to access the contents of different rows of data in the same table.

PostgreSQL has a separate feature for setting RLS, which allows to control which rows of a table can be accessed by a user based on specific conditions, thus enhancing the flexibility and security of resource access. At the same time, we have considered the use of Access Control Lists (ACLs) in conjunction with further granular access control to more carefully scrutinise and manage the permissions of individual rows in a table.

The Music360 database model itself contains information that can be used to control access to elements in the database. For an artist having a claim on a particular recording may read data about the recording, the claim and her/his own artist data. Due to the nature of the business of the CMOs, this information may change often, sometimes on a daily basis. It is not efficient to store access information that can be derived from the Music360 itself as an Access Control Entry (ACE) in an ACL. Moreover, it would be redundant.

Therefore, it is most effective and efficient to adopt a hybrid approach regarding representing access controls. On the one hand, access rights for rights holders should come from the Music360 data model (e.g., a rights holder's declaration of a recording or a work), which can be partially achieved by turning on and applying PostgreSQL's own RLS functionality. On the other hand, consider a scenario where additional access rights are granted to other users. The access controls now can not be directly derived from the Music360 data model. Consequently, we will use a PostgreSQL extension to implement ACLs. Specifically, eaw row will have an additional property to represent the ACL for that row. An ACL can represent multiple ACEs. Users have identity authentication via JWT scope verification (e.g., UUID, GroupID, IPI, IPN…), and policies represented as ACEs are linked to corresponding data/resource items to define access conditions based on stakeholders'/users' scope details. A user whose scope information is extracted from a JWT access token satisfies the policy associated with a data

[18] Mirabi, M., & Binnig, C. (2023). QFilter: Towards a Fine-Grained Access Control for Aggregation Query Processing over Secret Shared Data.

[19] α can be "count", "sum", or "avg" and OP can be "∧" or "∨" operator.

item is granted permission to perform a specific action (i.e., CRUD operations, an aggregation function) on that data item.

An ACE（access control entity）consists of the following: **[type] / [flags] / [who] = [mask]**.

Table 6: Access control entity construct

Components	**Description**
type	type of control
flags	can be used for custom permission
who	user or group, empty string represents all
mask	permissions

Each table in the Music360 will have an additional column, namely the ACL column. An ACL itself is a set of ACEs and represented as a JSON structure. This solution also motivates why it is not efficient to store frequently changing access control information (such as an artist who is assigned a claim on a recording) in ACLs; this information can more efficiently be derived from the Music360 data model.

Take for example the following data to store information regarding a recording

Table 7: Example of Recording _ACL_Table[20]

ISRC	Title	ACL
GBCBR2100128	Satisfaction	{a//Mick Jagger=r,a//Keith Richards=r, a//Ronnie Wood=r}
GBBKS1200164	Skyfall	{a//Adele=r}
…	…	…

There are also tables needed to store user and group information, as well as their (inherited) group membership. These tables themselves also have ACLs, because not everyone is allowed to use that data.

Table 8: Example of User_ACL_Table

User_id	User_name	ACL
14723672	Mick Jagger	{a//Mick Jagger=rwd}
89478272	Adele	{a//Adele=rwd}

[20] User names are given explicitly for readability; in the software a user will have a Unique User ID (UUID).

Table 9: Example of Group_ACL_Table[21]

Group_id	Group_name	ACL
84758394	Artist	{a//administrators=rwd}
83838383	Author	{a//administrators=rwd}
17823784	Person	{a//administrators=rwd}

Table 10: Example of Group_Group_ACL_Table

Group_id	Parent_Group_id	ACL
17823784	none	{a//administrators=rwd}
84758394	17823784	{a//administrators=rwd}
83838383	17823784	{a//administrators=rwd}

Table 11: Example of User_Group_ACL_Table

User_id	Group_id	ACL
14723672	84758394	{a//administrators=rwd}
89478272	84758394	{a//administrators=rwd}

[21] **Group** here can be seen as different role types generated from **Stakeholder** in the data model. Groups can be members of groups. For example, **Group_Performer** is the child group of **Group_Rightholder**. All privileges granted to **Group_Rightholder** will be inherited to **Group_Performer** while **Group_Performer** will have its own access control scope.

5 Data Import

The Music360 platform is, with a few exceptions, a query-only system. The vast majority of the users are only allowed to query the platform. The data providers (now: the CMOs plus BMAT) import data in the platform. The data import functionality itself is outside the scope of this deliverable, it will be the topic of WP6 as development of the Importers is part of the Living Labs.

The data providers are the owners of the data they import. They also determine the initial access of the data, and will set this accordingly. Usually, the data providers will have full access rights on the data that they import. Note that these access rights are restricted to the data the provider imports itself, so it does not imply that imported data can be accessed by other data providers.

6 OAuth 2.0 & OpenID Connect

Data access will be controlled by means of an authenticated user. Hence it is important to understand how Music360 will do user authentication. For this purpose, we will use industry standards OAuth 2.0 and OpenID Connect. Connecting these protocols to the often proprietary authentication solutions of CMOs is an important contribution of the Music360 project.

6.1 Overview

The OAuth 2.0 and OpenID Connect protocols provide specifications for sharing of user data and user identity, respectively. OAuth 2.0 initially provided a means for a resource owner to share data in a federated manner but provided no standard method to share identity data thus resulting in the advent of the OpenID Connect standard; a simple identity layer running on the foundations that the OAuth 2.0 protocol provides.

6.2 Definitions

We provide the working definitions for both OAuth 2.0 and OpenID Connect, enumerated in their respective specifications, RFC6749 and OpenID Connect Core 1.0.

6.2.1 OAuth 2.0

The following definitions are taken from various parts of the OAuth 2.0 specification (defined in RFC6749, RFC7636) and provided as a glossary to ease reading. The list is non-exhaustive.

- **Resource Owner:** An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
- **Resource Server:** The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
- **Authorization Server:** The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.
- **Client**: An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).
- **Authorization Grant**: credential representing the proof of consent a resource owner gives to a client to access their restricted resource.
- **User-Agent:** typically browsers, whether on mobile or traditional devices.
- **Redirect Endpoint:** client provided endpoint for authorization server to redirect to.
- **Request Endpoint:** endpoint for a client to initiate a request with the authorization server.
- **Token Endpoint:** endpoint for a client to exchange an authorization grant for an access (and refresh) token.
- **Access Token:** the token used to make an authorized request to the resource server. Typically short lived before expiry.
- **Refresh Token:** the token used to generate new access tokens subsequent to their expiry.
- **Scope:** the resources the client requests access to.

- **Code Verifier:** a cryptographically random string used to generate a code challenge for the Proof Key for Code Exchange methodology to prevent access code interception attacks.
- **Code Challenge:** the SHA256 hash of the ASCII encoded code verifier encoded in base64 to be sent to the authorization server to initiate the authorization flow with PKCE.

6.2.2 OpenID Connect

The following definitions are a subset taken directly from the OpenID Connect specification as defined in OpenID Connect Core 1.0 (Section 1.2).

- **Relying Party (RP):** OAuth 2.0 Client application requiring End-User Authentication and Claims from an OpenID Provider.
- **OpenID Provider (OP):** OAuth 2.0 Authorization Server that is capable of Authenticating the End-User and providing Claims to a Relying Party about the Authentication event and the End-User.

6.3 OAuth Flows & Disambiguation

OAuth 2.0 has multiple methods to enable a resource owner to grant a client access to their data, such methods are commonly referred to as *OAuth flows*. Counter-intuitively, OAuth does not define any flows, but rather specifies a list of grant types which serve as the mechanism to retrieve access tokens; each grant type represents an expected type of credential a user must exchange for an access token, enumerated below from RFC 6749.

6.3.1 OAuth Grant Disambiguation

- **Authorization Code Grant**: client obtains authorization code from a resource owner by way of the authorization server and exchanges it for an access token. Used when a client secret can be maintained.
- **Implicit Grant**: no exchange occurs, access token immediately granted. Historically used for single page web applications. Vulnerable to access code interception attacks and thus insecure.
- **Resource Owner Password Credentials Grant**: a client which is absolutely trusted with user credentials (e.g, username and password). A last resort mechanism.
- **Client Credentials Grant**: used when the client is the resource owner, **NOT** when the client is acting on behalf of the resource owner.

We predict the use of the following types of clients in the Music360 platform:

- **Music 360 Dashboard**: A single page application running in the browser which requires resource owner permission to access their data. Cannot secure a client secret.
- **Plugins**: Possible backend applications which need to connect to the resource server(s) over HTTP to do some background processing for a specific resource owner. Can secure a client secret e.g., as an environment variable in the container it runs in.

Given the conditions of use for the above grant types with our predicted use-cases, it is clear that the Authorization Code grant type is the only suitable grant for the Music360 ecosystem moving forward. However, the Authorization Code grant type is not suitable for single page applications as they are public clients which cannot maintain a client secret (which would have to be distributed with each instance of the web application). Consequently, we have to use the Proof-Key for Code Exchange mechanism, defined in RFC 7636, to secure the Music360 Dashboard application and any other browser based clients which wish to operate in the ecosystem. At the time of writing, this is the recommendation of the draft BCP in section 6.2.3.1 [2] and we henceforth refer to the combined use of these technologies as the **Authorization Flow with PKCE**.

In the case of plugins which **CAN** maintain a client secret, the authorization grant is suitable. We refer to this technology henceforth as the **Authorization Flow**.

OpenID piggybacks the Authorization Flow (with PKCE) request by appending certain values to the scope object; such scope objects do not interfere with the underlying OAuth request. Correspondingly, we can use a single request to provide authorization and identity to the dashboard from data providers.

We note that, while the Authorization Flow with PKCE ***should*** be secure for authorizing native apps (i.e., public mobile, desktop applications which cannot maintain the privacy of a client secret) in the Music360 ecosystem, we do not address their use case in this security brief and, until such a time where their security concerns are evaluated, such clients **OUGHT NOT** to be on the whitelist of any ecosystem data providers.

6.3.2 Router and Aggregation Service Client Disambiguation

The router and aggregation services are intended to serve as a level of indirection between the dashboard and the data providers such that the logic comprising complex queries is removed from the scope of the dashboard. At a surface level, it seems that we have two cases to consider when classifying the nature of the data aggregator.

- **Case 1** - Aggregation of queries on behalf of the resource owner.
    - The aggregation service can be viewed as a client **OR** as a backend extension of the dashboard. In the latter case, the aggregator can be trusted

in forwarding the access token(s) to various data providers to compile data as requested.
    - The aggregation service must not persist the resource owner's access token.
    - In the case that the access token is expired, an error must be returned to the client for it to exchange the refresh token for a fresh access token, upon completion, the aggregation request may continue.
- **Case 2** - Routing and aggregation of queries on behalf of **two or more** resource owners for a single end-user or client to consume.
    - Example: given Alice and Bob have data in the databases of various Music360 data providers.
        - Hypothesize that Alice wishes to view a subset of her data from one provider, along with a subset of Bob's data from two providers.
        - Alice must send a request to Bob for access to his data.
        - Bob must approve Alice's request.
        - Alice must have her access tokens for both data providers (re-)generated with the expanded scope.
        - Alice can now consume both her and Bob's data in the dashboard.
    - The dashboard or routing/aggregation service must coordinate with the data providers that house the data of the requested party and return a newly generated access token to the requesting party.
        - Problematically, the requesting party must wait for the requested party to approve the request for each data provider.
        - Given approval, the requesting party must have their own access tokens updated with the new scope of data accessible to them, even if such party is not registered with one of the data providers.
    - **IMPORTANTLY**: such data access **MUST NOT** be granted from the requesting party implicitly as they are not the resource owner.
    - Such behavior is essentially identical to case 1, where the aggregator takes custody of the requesting party's access token to compile data from the various sources, in this sense, the aggregator can still be viewed as a distributed component of the dashboard client.

We recommend following a policy of implementing case 1 as a priority with the option to develop case 2 at a later point in the project i.e., alongside the strategic objective of homomorphic encryption. We suspect that implementing homomorphic encryption to harden data protection guarantees will produce vast efficiencies in the implementation of case 2, thus freeing up development resources for the project's other strategic objectives.

6.4 Objectives

6.4.1 Proximate Objectives

We move forward under the following development assumptions.

- **Application Whitelisting**

    - While OAuth is a protocol which can support many clients, we recommend that parties in the Music360 ecosystem, at this current time, whitelist data access only to the following two.
        - Music360 Dashboard
        - Data aggregator
    - Whitelisting can be managed by enabling only privileged clients to provide a Redirect URI with which they attain an access code.
- **Data Aggregator as a Dashboard Backend**
    - As mentioned above, the dashboard and data aggregator may be seen as individual clients in OAuth, or as a single distributed client. Adopting the stance of the former implies that resource owners have to grant access to both the dashboard AND the aggregator. We consider such an approach redundant for the following reasons:
        - Such a policy would lead to two access tokens with the same scope, but would require the aggregator to validate and verify the dashboard's token to check for a scope match i.e., the aggregator must have access to the dashboard's access token.
        - Comparing scopes is wasted computation as there is no further security guarantee to be gained; the dashboard's access token ought to be routed through the aggregator.
        - In conclusion, the aggregator ought to be trusted by the dashboard as if it is its own backend component.
    - All communication between the dashboard, aggregator and data providers **MUST** be done over **HTTPS**.
- **Query Aggregation ONLY on behalf of the Resource Owner**
    - See case 1.
- Authorization server **MUST** enforce **ONLY** the authorization flow with PKCE.
- All major components **MUST** make a best effort to follow the relevant **BCPs.**

6.4.2 Distal Objectives

We identify the following objectives as carrying significant value to the Music360 ecosystem and its partners, but identify that such features are either supported by technology in its infancy or would bring the project into an unsustainable level of development in the acute term. Whilst our prediction is that such goals are still achievable on a longer term basis, we recommend they wait until momentum has been gained and the more proximate objectives completed.

- **Application Blacklisting**
    - A shift in strategic approach to the point where the Music360 platform becomes open.
    - Correspondingly, our policy ought to change from whitelisting privileged clients to only blacklisting those who have proven to be malicious.
    - Attack vectors ought to be identified and provisions taken to limit vulnerabilities to resource owners and providers.
- **Hardened Data Protection**
    - Multi-party data aggregation (case 2).
    - Homomorphic encryption.

- Rich authorization requests (see below).

6.5 OAuth Integration in Music360

The data providers in the Music360 ecosystem must implement the OAuth 2.0 protocol by establishing both authorization and resource services. Such services may be provided in one or more server applications. We aim to provide, and henceforth assume as the default case in the following discussion, a single authorization and a single resource server.

6.5.1 Registration Provisions

Before any OAuth requests can be sent, trust between the authorization server and the client must first be established. In the Music360 ecosystem, trust is implicitly provided to the dashboard client and aggregator as they are developed by project partners. When registering with the various authorization servers of the data providers, **at least two** pieces of information must be provided:

- **Client type - MANDATORY**
- **Redirection URI(s) - MANDATORY**
- Other information required by the authorization server to determine trust. Examples being:
  - Application name
  - Website
  - Description
  - Acceptance of legal terms, etc.

6.5.1.1 Client type

- ***confidential***: clients capable of maintaining a client secret e.g., traditional web applications.
- ***public***: clients incapable of maintaining a client secret e.g., SPA, native apps. This is the case for the dashboard. Accordingly, the authorization server **MUST** require a redirect endpoint (S3.1.2.2. RFC 6749).

6.5.1.2 Redirect URIs

- **One or more** absolute URI(s) (as defined in RFC3986).
  - **MAY** contain an application/x-www-form-urlencoded formatted query parameter.
  - **MAY NOT** contain a fragment component.
- Scheme **ABSOLUTELY MUST** be **HTTPS**, except in development environments.
- Clients **OUGHT** to not include any third party scripts in the redirect URI endpoint. If such scripts are included, the client **MUST** ensure its own scripts used to extract credentials from the URI execute first.

Upon successful completion of registration, the authorization server issues the registered client a unique client ID. Such client IDs are exposed to resource owners and are not secrets. The authorization **OUGHT** to document the size of any client IDs it issues.

Clients **OUGHT** to be able to deregister themselves from the authorization server and the authorization server **MUST** be able to deregister clients. We consider it out of scope of the document to suggest the means and manner in which such processes ought to be undertaken.

6.5.2 Discovery Provisions

In order to kick off the authorization flow with PKCE, the dashboard application must acquire metadata from the various authorization servers. Such metadata **MUST** be made available by the authorization server in the following manner (RFC 8414, OpenID Connect Discovery 1.0).

6.5.2.1 Metadata

*Unique OpenID Discovery metadata parameters are stylized in ***bold and italics***.

- Metadata **MUST** be made available as a Well-known URI (RFC 8615).
  - Default value as per RFC 8414 is "/.well-known/oauth-authorization-server".
  - As we opt to include the OpenID specification, the URI **OUGHT** to be **"/.well-known/openid-configuration"**. See section 3 and 5 of RFC 8414.
  - **MUST** use the **HTTPS** scheme.
- Metadata **MUST** be in JSON format.
- Metadata **MUST** include the following:
  - issuer
  - authorization_endpoint
  - token_endpoint
  - response_types_supported
  - ***subject_types_supported***
  - ***id_token_signing_alg_values_supported***
- Metadata is **RECOMMENDED** to include the following:
  - scopes_supported
  - ***userinfo_endpoint***
  - ***claims_supported***
- Metadata **MAY** include the following:
  - jwks_uri (see RFC 7517)
  - grant_types_supported
  - revocation_endpoint
  - code_challenge_methods_supported

For brevity, we only include the metadata items we predict we shall make use of in this project for all authorization servers, leaving their descriptions to the relevant standards. This list may be subject to change as unforeseen implementation details arise. We intend to implement JWKs according to RFC 7517 for authorization services.

6.5.2.2 Metadata Requests

- **MUST** be queried using HTTP **GET** at its specified path.

6.5.2.3 Metadata Responses

- **MUST** use **200 OK HTTP** status code.
- **MUST** return a JSON object using content type: **application/json**.

- Claims with zero values **MUST** be omitted from the response.
- Claims with multiple return values **MUST** be represented as JSON arrays.
- Error responses use applicable HTTP status code.
- Example seen at section 3.2 of RFC 8414

6.5.2.4 Metadata Validation

- ***issuer*** **MUST** be identical to the authorization server's URI**.** If not, the metadata response **MUST NOT** be used.
- String comparison must be done in accordance with RFC 8259 section 8.3.
- TLS certificate checking **MUST** be performed by the client when making an authorization server metadata request. See RFC 8414 section 6.

6.5.3 Authorization Request Provisions

In order to facilitate authorization requests, authorization servers **MUST** provide the two following endpoints:

- Authorization endpoint
- Token endpoint.

While clients must provide a single redirect endpoint.

6.5.3.1 Authorization Endpoint

Used to interact with the resource owner who wishes to provide the authorization grant to the client which initiated the request. It must behave in the following manner:

- This endpoint **MUST** verify the identity of the resource owner.
- The authorization endpoint **MUST** be located from the discovery metadata after such metadata has been verified.
- **MAY** contain an application/x-www-form-urlencoded formatted query parameter.
- **MAY NOT** contain a fragment component.
- **MUST** only communicate over **HTTPS**.
- **MUST** support HTTP GET method.
- **MAY** support HTTP POST additionally.
- **MUST** prune parameters sent without a value as if they were omitted from the request.
- **MUST** ignore unrecognized parameters.
- Request and response parameters **MUST NOT** be included more than once.
- **MUST ONLY** support the **S256** code_challenge verification method.

6.5.3.2 Token Endpoint

Used by the client to obtain an access token by presenting either its:

- Authorization grant.
- Refresh token.

The token endpoint must behave in the following manner:

- The token endpoint **MUST** be located from the discovery metadata after such metadata has been verified.
- **MAY** contain an application/x-www-form-urlencoded formatted query parameter.
- **MAY NOT** contain a fragment component.
- **MUST** only communicate over **HTTPS**.
- **MUST** support HTTP POST method.
- The client **MUST** make HTTP POST requests when making access token requests.
- **MUST** prune parameters sent without a value as if they were omitted from the request.
- **MUST** ignore unrecognized parameters.
- Request and response parameters **MUST NOT** be included more than once.
- **MUST** ensure the original authorization code is sent with a token request.
- **MUST** ensure that the authorization code is **VALID.**
- **MUST** ensure the authorization code was issued to the clientID.
- **MUST** ensure the redirect uri parameter is present if it was included in the initial authorization request (which will **ALWAYS** be the case with regards to the authorization flow with PKCE used in the Music360 ecosystem).
- **MUST** ensure code verifier hashes to meet code challenge **BEFORE** access and refresh tokens are granted, in the reverse manner by which the code challenge was created.
- **OUGHT** to record any size value of any issued tokens.
- **MUST** validate any refresh tokens.
- **MAY** issue new refresh tokens in exchange for old ones with identical scope.

6.5.3.3 Redirect Endpoint

See registration provisions above.

6.5.4 Authorization Request

With the above measures taken, clients are now empowered to make authorization requests to the various data providers' authorization servers. Below we enumerate the request parameters we predict we will use, along with explanations of expected requests and responses.

6.5.4.1 Code Verifier

- Client **MUST** create a code verifier for the authorization request.
- Code verifier **MUST** be generated from a high entropy cryptographic random string using the unreserved characters [A-Z] / [a-z] / [0-9] / "-" / "." / "_" / "~" with a minimum length of 43 characters and a maximum of 128. See section 4.1 and section 2.3 of RFC 7636 and RFC 3986, respectively.
- Such a string **MUST** then be base64url-encoded.
- The resultant base64url-encoded string then becomes the code_verifier.

6.5.4.2 Code Challenge

- Client **MUST** derive a code challenge from the code_verifier.
- Code challenge **MUST** be generated via algorithm **S256**.
- The resultant code_challenge is thus generated via **BASE64URL-ENCODE(SHA256(ASCII(code_verifier)))**.

6.5.4.3 Scopes

Subsequent to the code verifier and code challenge generation, the scope of the authorization request must be generated from the dashboard. The essence of the scope is a match with the resources available in the resource server. The authorization request scope may contain elements from the OpenID Connect Core 1.0 standard along with custom scope objects, which OAuth 2.0 defines the syntax of in RFC 6749 Appendix A.4. The scopes for the dashboard's request to the authorization servers are as of yet undefined, however, we predict their format as follows:

6.5.4.3.1 OAuth 2.0 Example Scopes

- ***user.payments***: permits all access operations to the user payments details.
- ***user.payments.read***: permits only read access to the user payments details.
- ***user.locations_activity.read***: permits read access to location activity.
- ***user.subscriptions.read:write***: permits read and write access to user subscriptions.

At this current time, we consider it unforeseeable which operations may be required based on the need to cement an initial scope specification. Correspondingly, we may opt to change the format of any custom scope objects; and may opt to dispense with their use entirely should the advent of full RAR (see below) support come to keycloak in the near future.

6.5.4.3.2 OpenID

OpenID Connect Core 1.0 section 5.4 provides five scope claims, all of which are optional. We enumerate them below.

- ***openid*****:** claim to dictate request is an OpenID request. Requests to following claims without this result in undefined behavior from the authorization server.
- ***profile***: requests access to the end-user's default profile claims. Refer to OpenID Connect Core 1.0 section 5.4 for further details.
- ***email***: requests access to user email and email_verified.
- ***address***: requests access to user's address.
- ***phone***: requests access to the user's phone_number and phone_number_verified claim.

6.5.4.4 Rich Authorization Requests

The RAR standard dictated by RFC 9396 works to invalidate the need for a scope object and replace it instead with a JSON formatted request object which provides the means for even more fine-grained access control. At the time of writing, the adopted authorization server of choice, keycloak (D2.2), does not appear to have sufficient adoption for the standard and thus development must proceed around the scope object for now.

6.5.4.5 State

The state object is typically a random value used to protect against CSRF attacks. We do not expect any specific vulnerability to CSRF attacks as we expect to store all access and refresh tokens in browser based persistence and not cookies.

6.5.4.6 Request

The OAuth authorization flow request from the dashboard is expected to be composed of the following query parameters:

- **response_type** (mandatory)
- **client_id** (mandatory)
- **redirect_uri** (mandatory in case of the music360 dashboard)
- **scope** (mandatory in case of the music360 dashboard)
- **state** (optional, but recommended)
- **code_challenge** (mandatory)
- **code_challenge_method** (mandatory value S256 in case of the music360 dashboard)

An example URI to the authorization server's request endpoint (HTTP **GET**) with query parameters separated by color:

https://auth.music360.org/oauth2/auth? client_id=dashboard123 &redirec t_uri=https%3A%2F%2Finsights.music360.org%2Fcallback &scope=openid%20 email%20user.payments.read%20ipi%20ipn &response_type=code &state=xyz & code_challenge=8_TBPZfTvVrVJSeg4C1J__GfsaZE2STZgegbex46wNY &code_chal lenge_method=S256

If the authorization server finds the state of the request sufficient according to rules defined above, it authenticates the resource owner and awaits a decision. At this stage it is worth noting that the authorization server **MUST** persist the code challenge and the code challenge method.

6.5.4.7 Response

Upon the authorization decision, the authorization server redirects the user-agent to the redirect URI with certain query parameters attached depending on the success or failure of the request.

- Success
  - **code** (mandatory): exchanged along with code verifier to retrieve access and refresh tokens.
    - **MUST** expire at most 10 minutes after generation.
    - **MUST** not be used more than once.

        - If used more than once, authorization server **MUST** deny the request and invalidate all tokens previously issued based on that specific authorization code.
        - Code **MUST** be bound to the clientID and redirect URI.
    - **state** (mandatory): if present in the authorization request.
- Failure
    - **error** (mandatory, see RFC 6749 for use details of each code)
        - invalid_request (applicable in case of no code_challenge present in PKCE based request)
        - unauthorized_client
        - access_denied
        - unsupported_response_type
        - invalid_scope
        - server_error
        - temporarily_unavailable
    - **error_description** (optional): A human readable ASCII text providing additional information. Must exclude certain characters. See section 4.1.2.1 of RFC 6749 for details. **OUGHT** to be used to explain any errors.
    - **error_uri** (optional): a URI providing a webpage used to provide the client with additional information. Character set restrictions apply.
    - **state** (mandatory): if present in the authorization request.

Using the original example to demonstrate, the user-agent would be redirected as follows

- **Error**: https://insights.music360.org/callback?error=invalid_request&error_description=no code challenge present
- **Success**: https://insights.music360.org/callback?code=SplxlOBeZQQYbYS6WxSbIA&state=xyz

6.5.5 Access Token Acquisition

Subsequent to its acquisition of an authorization code following the above steps, the client must exchange such a code for both an access and refresh token.

6.5.5.1 Request

The client must make a request to the Token Endpoint of the authorization server in the following manner:

- **MUST** include the following parameters:
    - grant_type = “authorization_code”.
    - code = the code issued in the success callback.
    - redirect_uri = the identical uri used to initiate the authorization request.
    - client_id = the identical client id used to initiate the authorization request.

    - code_verifier = the plaintext string used to generate the code challenge.
- **MUST** use the application/x-www-form-urlencoded format.
- **MUST** encode characters in the HTTP request entity-body with UTF-8.

An example URI to the authorization server's request endpoint (HTTP **POST**, Content-Type: application/x-www-form-urlencoded) with query parameters separated by color:

https://auth.music360.org/oauth2/token? grant_type=authorization_code &code=SplxlOBeZQQYbYS6WxSbIA& client_id=dashboard123& redirect_uri=htt ps%3A%2F%2Finsights.music360.org%2Fcallback &code_verifier=DECAFBAD

6.5.5.2 Response

After the authorization server processes the request in the manner specified in step three, there are two possible responses:

- Success
    - Issuance of an access and refresh token via HTTP 200 (OK) response with the following parameters (see RFC 6749 section 5.1 for further details of parameter values):
        - access_token
        - token_type
        - refresh_token
        - scope
        - expires_in (while recommended, we intend to make this part of the access token claim, thus unnecessary).
    - The parameters in the entity-body of the HTTP response **MUST** use the application/json media type.
    - HTTP "Cache-Control" response header **MUST** be included in the response with a "no-store" value.
    - HTTP "Pragma" response header **MUST** be included in the response with a "no-cache" value.
    - Access and refresh tokens **MUST EXCLUSIVELY** be used as a **Bearer** token.
- Failure
    - Issuance of a HTTP 400 (Bad Request) with the following parameters defined in RFC 6749 section 5.2 (ASCII formatted error code; for forbidden characters see also see section 5.2):
        - error
            - invalid_request
            - invalid_grant (in case of PKCE error whereby code challenge and code verifier are unequal after correct processing)
            - invalid_client
            - unauthorized_client
            - unsupported_grant_type
            - Invalid_scope
        - error_description

        - error_uri
    - The parameters in the entity-body of the HTTP response **MUST** use the application/json media type.

The client **MUST** ignore any unrecognized names in the response and **OUGHT NOT** make any assumptions about token sizes.

6.5.6 Access Token Refreshing

We expect to use short lived access tokens and longer lived refresh tokens to facilitate data transfer. Refreshing refers to exchanging of the refresh token for a short lived access token and should be facilitated described in the subsequent manner.

6.5.6.1 Request

- **MUST** include the following parameters:
    - grant_type = “refresh_token”
    - refresh_token = the refresh token to be used.
    - scope = the full or subset of permissions which are granted to the refresh token.
- **MUST** use the application/x-www-form-urlencoded format.
- **MUST** encode characters in the HTTP request entity-body with UTF-8.
- **MUST** discard any old refresh tokens if new refresh tokens are issued.

6.5.6.2 Response

See step 5.

6.5.6.3 Token Structure

Our access and refresh tokens will follow the relevant JWT standards (RFC 7515 through to RFC 7519) along with the best practices defined in RFC 8725 to enhance system security. N.B., we intend to add an additional claim to our JWTs in the “scope” object, which will carry sufficient information to implement our RLS policies in the database. As noted earlier, we have a preference to implement the rich authorization requests standard in RFC 9396 which would append an “authorization_details” claim to the payload of the JWT in preference of the scope. Both custom claims will be interoperable with each other. An example access token payload as follows:

```
{
    "aud":"https://consumer.example.com",
    "iss":"https://issuer.example.com",
    "exp":1443904177,
    "nbf":1443904077,
    "sub":"dgaf4mvfs75Fci_FL3heQA",
    "scope":"email profile phone address"
 }
```

6.5.6.4 Token Storage

In accordance with the OAuth 2.0 for Browser Based Apps draft BCP, we opt to use the browser's persistence mechanisms (i.e., local/session storage or IndexedDB) until such a time where XSS attacks in browsers become untenable. To reduce attack surface, we opt to use as few external ECMAscript or Typescript libraries in clients as possible.

6.5.7 Querying

6.5.7.1 Client Requests

- The client **MUST** ensure the access and refresh tokens are not expired.
- If an access token is expired, the client **MUST** request a new access token from the token endpoint in accordance with step 6.

6.5.7.2 Resource Server

- **MAY** offer a reasonable grace period to access and refresh tokens which are expired e.g., large requests at time of sending, network latency etc. Such a grace period ought to not exceed the order of seconds.
- **MUST** verify the signature of the JWT with the public key of the issuing authority.
    - Inspecting the "iss" claim is not sufficient for such a task. The issuer must be verified before any inbound requests reach the resource server.
- **MUST NOT** inspect the algorithm type of the JWT and instead assume use of an agreed upon signing algorithm to prevent the "none" and "RSA to HMAC" attacks. As of writing, the agreed upon algorithm is expected to be RS256. We note that EdDSA typically provides better performance under decryption but at this current time lacks sufficiently robust open source implementations for us to prefer it in the proof of concept system.
- **MUST** implement the **CORS** standard to the minimal degree to share data with authorized clients in the Music360 ecosystem.

6.5.8 Token Revocation

In accordance with **RFC 7009** our system aims to implement revocation of both access and refresh tokens. Revocation shall be processed at the revocation endpoint defined in the discovery provisions above.

6.5.8.1 Revocation Request

The revocation request to the authorization server should be provisioned for in the following manner:

- **MUST** use the application/x-www-form-urlencoded format.
- **MUST** encode characters in the HTTP request entity-body with UTF-8.
- **MUST** use HTTP **POST**.
- Include the following query parameters:
    - token (mandatory)
    - token_type_hint (optional, to optimize token lookup)
        - access_token
        - refresh_token

6.5.8.2 Revocation Response

- Success:
  - 200 OK.
- Error:
  - unsupported_token_type: the authorization server does not support the revocation of the presented token type. With error 503.

6.5.8.3 Token Blacklisting

Upon an authorized revocation request, the authorization server attempts to revoke the token using an event system (message bus); small propagation delays may be experienced. We may pursue a policy of atomicity amongst services implementing access/refresh token (i.e., the resource server(s)) before issuing a 200 OK response to the client.

In the case that a token is a refresh token, the authorization server **MUST** attempt to invalidate all of the access tokens derived from it. Moreover, the resource server(s) must make provisions to deal with asynchronous access/refresh token revocation whilst continuing to make their data available and consistent.

7 View Applications

The dashboard application:

- **MUST** register views for OAuth and OpenID requests to all data providers in the Music360 ecosystem.
- **OUGHT** to register the means for the request of revocation of specific access and refresh tokens.
- **MUST** use as few external libraries as possible, and, when using libraries, conduct a scrutinous security analysis of them.
- **MUST** be in accordance with all client requirements defined above.
- **MUST** ensure use of PKCE in all OAuth and OpenID requests.
- **MUST** make general provisions to ensure XSS, CSRF and client-side injection based attacks are minimized.

The authorization server for all data providers in the Music360 ecosystem:

- **MUST** register views to:
  - Authenticate resource owners.
  - Query resource owners for their permission to share their data with third parties (i.e., clients, and possibly later other resource owners).
  - Revoke the data access of all clients and parties for any given resource owner.
- **MUST** register the means for the revocation of specific access and refresh tokens.

8 Data Import

In the Music360 ecosystem platform, the database will have the characteristics like distributed, replicated, and partitioned. In principle, every data owner / the data provider (CMOs and BMAT) need to take control of their own data, by being the administrative and technical responsible entity for their own data. After transforming these raw data into a unified Music360 data model, we need to collect and utilize data from diverse sources from their own private database, and then load it to the Music360 database architecture. The Music360 database which is partitioned and/or replicated by each data provider will be implemented on the basis of PostgreSQL. Our initial data import solution will be listed as follows:

- **CMOs data resource import:** For the private and existing database of the data providers, the databases currently used by CMOs are quite different in terms of technology as well as data model. As a result, we have proposed a unified data model to try to remove some of the semantic barriers and prepare for data integration. The current assumption is that CMOs' databases can be exported via APIs or database dumps and then loaded into the Music360 platform's database partitions after the data model has been transformed.
- **Other data resource import:** To facilitate collaboration and interoperability, the database supports standardised data exchange formats like JSON and XML. Well-defined API endpoints enable external software users to seamlessly query the Music360 platform programmatically. Appropriate API endpoints will be defined and utilized for retrieving data resources like user information, licence details, Living Lab data and so on.

9 Conclusion

This document explained the first version of the security architecture for the Music360 platform. The architecture focuses on securing access to data stored by the platform. The important assumption is that data processors are trusted and behave as agreed, e.g. stating by a legally binding contract. This assumption will be relaxed in the second version of the deliverable.

Data is protected where it is stored, namely by the database management systems. We delegate as much as possible access control to the database management system. There are arguments to do this, the most important one is that implementation of security control is then concentrated in one central part of the architecture, namely the database, and not scattered over many components, e.g. back-end components. It also allows to keep data access management under the control of the data provider, e.g. a CMO or BMAT.

To authenticate users, we use the industry standard OAuth 2.0 / OpenID connect. It allows decentralization of authentication (e.g. each CMO can authenticate users using its own policy). Moreover, it advances the state-of-the-art, as the industry uses proprietary solutions for this now.

10 Standards

1. OpenID Connect Core 1.0
2. OpenID Connect Discovery 1.0
3. OAuth 2.0 for browser based apps (Draft BCP)
4. OAuth Parameters
5. RFC 6749 - The OAuth 2.0 Authorization Framework
6. RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage
7. RFC 6819 - OAuth 2.0 Threat Model and Security Considerations
8. RFC 7009 - OAuth 2.0 Token Revocation
9. RFC 7515 - JSON Web Signature (JWS)
10. RFC 7516 - JSON Web Encryption (JWE)
11. RFC 7517 - JSON Web Key (JWK)
12. RFC 7518 - JSON Web Algorithms (JWA)
13. RFC 7519 - JSON Web Token (JWT)
14. RFC 7523 - JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants
15. RFC 7636 - Proof Key for Code Exchange by OAuth Public Clients
16. RFC 8252 - OAuth 2.0 for Native Apps
17. RFC 8259 - The JavaScript Object Notation (JSON) Data Interchange Format
18. RFC 8414 - OAuth 2.0 Authorization Server Metadata
19. RFC 8693 - OAuth 2.0 Token Exchange
20. RFC 8725 - JSON Web Token Best Current Practices
21. RFC 9396 - OAuth 2.0 Rich Authorization Requests