Diana Giova

Open Source Digital Asset Management System for Audio Content

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Media Engineering

Thesis

27 November 2015

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Diana Giova Open source digital asset management system for audio content 32 pages + 4 appendices 27 November 2015
Degree	Bachelor of Engineering
Degree Programme	Media Engineering
Specialisation option	Mobile Programming and .NET Application Development
Instructor(s)	Kari Salo, Principal Lecturer

The aim of this project was to repurpose an open source content management system to manage audio files as per the requirements of Helsinki Museum of Technology in Finland. The museum desired to have a storage system that they could use during their workshops and share the files stored in this system with their visitors while letting those visitors not only download and search files, but also upload their own creations.

The implementation of this system happened both on a developmental domain of Metropolia University of Applied Sciences, where it would be used for testing by mobile application developers, and on the server of Museum of Technology in Finland, where this content management system would reside thereafter. At the time of writing and developing this project, the museum had neither an online storage system nor any offline system to store and manage audio files.

To be able to complete this system research was done on open source digital asset management systems, REST web services and audio metadata. This led to a decision on which open source system to use which in turn led to modifications of this system according to the requirements as well as creation of REST-applicable services for it.

This thesis is a starting point for future mobile application development and for future digital asset management system implementations in educational organizations which are looking for similar functionality. Open source systems can be of use across multiple applications, regardless of their initial build requirements. It is important to realize that existing systems can be adapted to any requirements, as long as licenses permit.

Keywords	DAM, open source, audio metadata, API, REST

Contents

List of Abbreviations

DAM	Digital asset management consists of management tasks and decisions surrounding the ingestion, annotation, cataloguing, storage, retrieval, and distribution of digital assets. These systems include computer software and hardware that aid in the process of digital asset management.
cURL	This is a computer software project providing a library and command-line tool for transferring data using various protocols.
JSON	JavaScript Object Notation is an open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs.
API	Application Programming Interface which allows the use of a library of pre-defined functions to simplify and standardize otherwise complex tasks.
CSS	Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in markup language.
HTML	HyperText Markup Language, commonly referred to as HTML, is the standard markup language used to create web pages.
SQL	Structured Query Language is a special-purpose programming language designed for managing data held in a relational database management system.
PHP	PHP is a server-side scripting language designed for web development, but also used as a general-purpose programming language.
JS	JavaScript is a high-level, dynamic, untyped, and interpreted programming language.

1 Introduction

Metropolia University of Applied Sciences has been taking part in the People's Smart Sculpture which is a creative research and innovation project regarding the integration of new art, design, science, smart technologies, and user culture into the labs organized by twelve different partners across Europe. The University of Applied Sciences in Bremen has been appointed as the lead partner of this project and Metropolia University of Applied Sciences as one of the partners. The other partners have been numerous polytechnics, universities, museums and art institutions across Europe. The press release concerning this project can be found in appendix 1.

Metropolia has been collaborating with the Museum of Technology in Finland which will be taking part in a project called the Neighborhood Living Room. This was a subproject in the People's Smart Sculpture initiative. The goal was to test ideas of how visitors could participate and interact with the exhibitions. The methods that would be used in this exhibition would be applied art, social media, and mobile technologies and the main vision for this project was to bring together and induce interaction of the residents in the nearby vicinity of the museum. This would be achieved by helping the museum's visitors develop new ideas and opinions, share experiences, and help discuss their dreams and hopes as part of the museum's interactive exhibition.

Metropolia's faculties of Culture and Engineering collaborated together in order for the above vision to be realized. This thesis analyzes the work from the engineering department, specifically the development of the digital asset management system that would be used as a basis to store the assets that the museum will be using in its exhibitions. The purpose of this final year's project was to modify an open source digital asset management system so that it would be better suited for storage and manipulation of audio files as was required by the Museum of Technology in Finland. In addition this system needed APIs so that the assets in it could be searched, downloaded and uploaded by the mobile applications which would be created by the engineering students.

The museum would be using this system during its workshops, events or exhibitions where they would want to share and collect audio files with the public. Such a system would require a well-managed back-end as well as front-end and APIs that would suit

any application (be they mobile or web) that would be built in the future. Of course all the possible scenarios could not be anticipated during the development of this system, so the documentation had to be very well structured and custom code had to be commented appropriately. This would ensure successful future upgrades or modifications to the existing system.

2 Open Source Digital Asset Management Systems

The two main prerequisites for the final year project's application were that the software should be an open source digital asset management system and it should have a REST API. The following paragraphs break apart and explain those prerequisites as this is necessary for understanding how the system that would be used for this project was chosen.

2.1 Open Source Software

Open source software is a well-known term amongst programmers nowadays. For clarification, the official definition is included in appendix 2 and a summary follows. This type of software must be free for download, its code should be readily available, meaning that it is usually distributed with its source code, and the license of this software must not restrict others from altering the code or distributing any derived works from it. (Kavanagh 2004, 1.) The type of licenses that accompany open source software may be:

- GNU General Public License
- Mozilla Public License
- BSD, Apache, or MIT license
- GNU Lesser General Public License

There is a possibility that a custom license has been made to open source software therefore one must review the license in detail before its implementation.

One of the advantages of using open source software is that it is acceptable to view and modify the source code (Kavanagh 2004, 41). In this case if an application is chosen, but it doesn't exactly suit the requirements perfectly, then it is possible to modify the code in any way that is deemed necessary. This reduces working costs and time, because the majority of the application has been developed already and only a few changes are necessary. The developer must be ready and willing to understand the code of other programmers in order to succeed in this venture. In order to understand the functionality of open source software, it is preferable that a well-written documentation accompanies it. Otherwise the learning curve for such software might be too steep.

2.2 Digital Asset Management System

A digital asset management system, or DAM, should be able to manipulate as well as protect from unintentional alteration the digital assets that are being stored in it (Jacobsen, Schenker and Edwards 2005, 3). In order to understand the types of data that this system stores it is essential to understand what a digital asset is.

One of the definitions in the IT and Media world of a digital asset is that it is a file which is tagged with the information about it. This way the file retains that information and is thusly not lost amongst all the other files found in a management system. This definition, that "an asset is a file plus metadata" (Jacobsen, Schenker and Edwards 2005, 3), is usually used by large companies. The second definition is that an asset is a file and its rights. This essentially means that content has value as long as its owner has the right to utilize it. For example an mp3 file created by a famous artist would not have any value unless someone had the right to use it. (Sutter, Notebaert, Walle 2006.)

The two definitions are complementary where one usually does not exist without the other. If a file has no metadata then this would make it a useless asset even if one had the right to use it. On the other hand, if a file has extensive metadata, and yet there are no usage rights, this also produces a similar problem.

Based on these findings it is determined that a digital asset management system should contain digital files along with their metadata as well as usage rights. The management aspect of the system is fairly straightforward to understand as it is the actions which are required to be executed onto these assets. The aforementioned assets could be adding, removing, and editing assets' data or metadata. This in turn would ensure that the digital integrity of the asset is maintained while "providing re-purposed files for every media need" (Roskiewics 2005, 7).

2.3 RESTful Web Service

Dealing with data is one of the major elements of building the World Wide Web. This data is connected from system to system by services that are either delivering or consuming it. In order for this to happen, web services come into play. Some of the available web services are Representational State Transfer (REST), Remote Procedure Call

(RPC) and Simple Object Access Protocol (SOAP). This project will be focusing on REST services, as this was one of the requirements for the system.

REST is a set of principles that governs the transfer of data and is usually tied to HTTP, HyperText Transfer Protocol. HTTP is a “stateless transport protocol where each request is executed independently, without the knowledge of the request that came before it” (Abeysinghe 2008, 11). Each HTTP transaction consists of a request and a response, where HTTP verbs are used to make the requests and the response is sent in a variety of formats, one of which is JavaScript Object Notation, or JSON.

The four basic HTTP verbs are used to provide the operations which are applied to resources. Resources are the individual data records in a system (O’Reilly 2013, 55). Those verbs are POST, GET, PUT, and DELETE which stand for Create, Read, Update, and Delete respectively, or CRUD. It is also possible to see an implementation of PATCH which allows partial update of a record. The use of PATCH is not very common however.

RESTful services deal with transferring representations of resources and this representation might be in the format of JSON. Transferring of resources essentially means transferring the data that these resources contain, which would make up the content of the request. The data about the request goes into the headers while the actual content is in the main body of the communication. This means that a well-structured RESTful service will use the verbs, status codes and headers for extra information regarding the request or response, reserving the body for the content only. (O’Reilly 2013, 55.)

To create resources in a system a POST request needs to be made to the specific collection where this resource is supposed to go to. The Content-Type header of such a request needs to be set appropriately so that the server will be able to understand it. Once the resource has been created a successful status code needs to be included with the response. A successful status code could be 201 (meaning ‘created’) or it is acceptable to set a Location header which will redirect the user to the URI of the new record. In the event that resource creation failed it is common to output a status code of 400 ‘Bad Request’ or 406 ‘Not acceptable’.

In order to read resources from where they are located the GET verb is applied. In an unsuccessful read attempt the most often seen status code is 404 which indicates the

record was not found or does not exist. There are many possibilities for which codes to use, depending on the individual cases. For example, the code 403 means ‘Forbidden’ which usually means the user does not have the correct rights to access this service. It’s also possible that the user has exceeded their allotted number of requests in a given time frame and this would produce a code of 429 ‘Too many requests’. Exposing too much information with the status codes is contraindicated. On the other hand, exposing too little information would be practically useless.

In order to create resources the verb PUT is used. It is essential to note that the permission of the web server should be set to writable as the default will prevent any users to upload their resources to the server through a request. In the real world instead of PUT the POST request is used more often (Abeysinghe 2008, 80) due to the previously described inconvenience regarding server permissions. If the server is set to writeable then anyone with malicious code will be able to access and modify the files there. Hence POST is used to create as well as update resources although in theory it might look incorrect to do so. POST will send data to a specified URL and the server will do whatever the program on its side is designed to do. It can either store the data received or it can use it as input for existing resources.

In conclusion a RESTful web service will allow the communication between the application containing that service and applications accessing that service, for example a mobile app, a Curl application or a REST plugin for the web browser. Therefore it can be inferred that the creation, deletion, reading, and updating of files will be performed in the chosen digital asset management system so when choosing it these features have to be taken into consideration.

2.4 Comparison of Available Systems

There are many different kinds of digital asset management systems. Some might be specifically designed for image files while others might be designed to handle audio, video or pdf files. Based on the initial prerequisites and the ensuing findings discussed in previous paragraphs, a search was launched using the Google search engine for an ‘open source digital asset management system’. This produced a multitude of results, all of which had to be tested for their suitability for this project.

In order to make a final decision of a suitable system for this project an effort was made to find an open source system that can accommodate audio files along with their metadata and one which has APIs available already or a possibility to create new ones for it.

Error! Reference source not found. depicts a representation of open source DAMs along with their features, license and programming language(s). The selection was compiled from a webpage reviewing open source DAM software (Sarwan 2014) as well as from a basic Google search for 'open source audio management audio'. The best suited systems were chosen from Sarwan's website as well as from the results based on Google search. Each chosen system was researched and tested which led to the construction of the table below.

Table 1: List of DAMs. Data gathered from Sarwan (2014) and from Google search.

Software name	Features	License	Language
Telemeta	• open source • web audio archiving software • metadata • user management • English and French support • REST API	CeCILL	Python and JavaScript
ResourceSpace	• fully featured DAM system • user management • API available • plugins available • metadata	BSD	PHP and SQL
Phrasenet	• DAM system • user management • images/video/documents support	GPL3	PHP
EnterMedia	• typical DAM system • uses XML, but database possible • plugins available • metadata	LGPL	JAVA

By looking at this table it can be determined what the best candidates for an audio system might be. Telemeta seems to have been built exclusively for audio files containing metadata, it is open source, has user management and supports a REST API. Therefore it can be assumed it will be the best suited system for the requirements of the final year's project. The runner up candidate would be Resource Space as it also seems suited well for audio storage, has user management and has an API. However, according to Roskiewics (2005), it is important to know what the requirements for such a system are as well as which programming language is best suited for the organization implementing such a system before reaching a final decision.

In order to decide which system will be implemented all of the above systems were tested for their suitability for this project and their compatibility with the initial requirements. The following paragraph provides the conclusion regarding these tests.

2.5 Conclusion

The system that was chosen to be implemented was ResourceSpace. Comparing to other systems it offered services that were required. Although Telemeta seemed to be better suited to the requirements, it was not chosen because it was still in development mode and was not compatible with Windows systems, (cf. appendix 3). In addition the use of PHP was preferable to the project's author. Prior to starting testing of this system, audio metadata and application requirements had to be analyzed, as will be discussed in the following chapters.

3 Application Requirements

Detailed application requirements were discussed at a meeting with Päivi Takala, a senior lecturer in Sound Design from Metropolia University of Applied Sciences, and one of her students. In addition to what was already known, as outlined in previous chapters, the following topics were discussed and added to the requirements list.

- Metadata for audio files, to be researched further
- Formats of audio files such as wav, mp3, raw, or flac
- Possible categories such as nature, humans, machines, and stories
- Possible sound types such as soundscapes, ambience, and effects
- Filtering for user management: museum admin and general users
- Client interface
- API access for mobile users: downloading from mobile app, uploading through mobile app, and searching through mobile app

The above topics will be discussed in the following chapters.

3.1 Audio Storage with Metadata

Metadata is information about a resource that is stored with this resource. Simply defined it is “data about data” (Zeng and Qin 2008, 7). The definition has been refined over time to “structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource” (NISO, 2004,1, cited in Zeng and Qin 2008, 7). There are many metadata standards which provide guidelines regarding content, structure and values. One such standard is the Dublin Core. It is well known because it ‘has the most mapped element sets among and across domain specific and community oriented metadata standards’ (Zeng and Qin, 2008, 16). Using Dublin Core’s metadata list as a base, a new concise list was created

and used in the first version of this software implementation. The first version can be found in appendix 4 and the second version can be seen in Table 2. At the time of writing the second version had been implemented.

Table 2: Metadata Fields.

Metadata Fields	Input style
Title of recording	free-form text
Tags	free-form text
Description	free-form text
Length (sec)	number value
Category – 1. nature 2. human 3. machine 4. story	options tick list
Sound Type – 1. soundscapes 2. ambience 3. effects	options tick list
Location – latitude	number value
Location - longitude	number value
File size (MB)	automatically generated
File extension	automatically generated
Creation date	automatically generated

Table 2 depicts the names of metadata fields that accompany an audio file. An explanation is given after each metadata fields as to how a user can input data into it. These fields are changeable only by a system administrator and the role of the user of the system is to either browse through those fields or edit/input values for them.

3.2 Management Interface

The management interface of the system that the person in charge of audio files will be accessing, should be straightforward to use. A user guide will be provided. In this particular situation the person in charge of this system will be called the museum admin and will be referred to as such from here on forward. The museum admin should be able to upload files and assign metadata field values to them. These metadata fields should be editable along with their options and if needed, it should be possible to create new fields. Regarding audio files, the museum admin should be able to delete their own files as well as those uploaded by different users. In addition the museum admin

should be able to create and delete users of this system as well as share selected audio files with them.

3.3 REST API Access

The RESTful web services that were discussed in the previous chapter are implemented in an API. An API is an Application Programming Interface, in other words a program, that lets a client make a call to the server and get data in return. This call is made using the aforementioned RESTful services. The requirements for this project's APIs were the ability to upload (POST), download (GET) and search (GET). This in turn would serve mobile users with their custom made applications as they would be accessing the API and consequently interacting with the system.

In addition to the basic functions of uploading, downloading, and searching the files it was decided to include an authentication API. This API would be a sort of starting point from which entry into the system would be granted. By using the authentication API only users which are granted access by the museum are able to use the other APIs and consequently upload, download, or search. This is an obvious security measure so that unauthorized users will not be able to access the system and alter its contents. It is possible to use HTTP basic authentication, OAuth, Custom Headers, or design your own application (O'Reilly 2013, 26). As a result a decision was made to create an authentication API which would be customized precisely to this system.

4 Application Design

Upon initial inspection of Resource Space as a digital asset management system, it was determined that indeed the decision to implement specifically this system was well-founded. However in order to customize the system according to the requirements of this project, a detailed breakdown of use cases and user stories was essential.

4.1 Use Cases

Figure 1, Figure 2, and Figure 3 are Venn diagrams depicting the use cases for this system. As can be seen from the diagrams there will be three user groups. The developer will install the system as required and maintain the integrity of it if needed. The museum admin will be the person(s) in charge of distributing and manipulating the audio resources stored in the system. The mobile user(s) will be accessing the system to search and retrieve audio files as well as upload their own.

Figure 1: Museum Admin Use Cases.

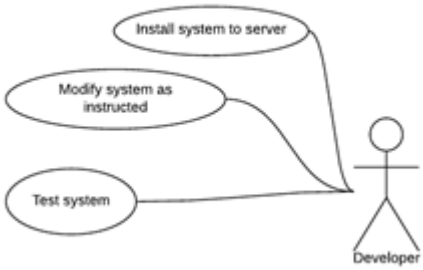

Figure 2: Developer Use Cases.

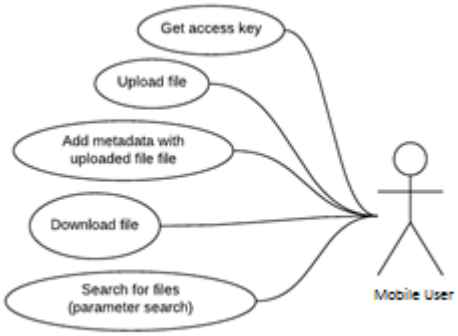

Figure 3: Mobile User Use Cases.

4.2 User Stories

Based on the use cases depicted in the last chapter, it was possible to determine what types of user stories might be applicable to the system. There will be three types of users with the museum admin being the most important one. Therefore they have the most diverse list of stories which follows below.

- As a museum admin I want to: create user accounts.
- As a museum admin I want to: delete user accounts.
- As a museum admin I want to: upload audio files to system.
- As a museum admin I want to: delete an audio file from system.
- As a museum admin I want to: create a new collection.
- As a museum admin I want to: view a list of all collections.

- As a museum admin I want to: add files to collection.
- As a museum admin I want to: move files among collections.
- As a museum admin I want to: edit audio files' metadata values.
- As a museum admin I want to: edit audio files' metadata fields.
- As a museum admin I want to: create metadata fields.
- As a museum admin I want to: delete metadata fields.
- As a museum admin I want to: change my dashboard layout.
- As a museum admin I want to: search for a file.
- As a museum admin I want to: see all the files in the system.

The developer will be in charge of setting up the system. For that reason their stories are quite straightforward and limited. In the case of repairing the system the user stories might change, but this is beyond the scope of this thesis. The developer's stories are listed below.

- As a developer I want to: set up RS on the server.
- As a developer I want to: modify RS according the user guide.
- As a developer I want to: test the system.

Finally the mobile user is the last in line to use this system. The mobile user will be able to use this system only if the museum admin gives them permission and populates the storage with data. A list of user stories relevant to the mobile user is shown below.

- As a mobile user I want to: receive an access key for using the APIs.
- As a mobile user I want to: search for files.
- As a mobile user I want to: search for files based on parameters I set.
- As a mobile user I want to: download an audio file.
- As a mobile user I want to: upload an audio file.
- As a mobile user I want to: upload an audio file with metadata fields.

The users along with their stories have been determined and explained. As can be seen the mobile user does interact directly with the system, but only with the APIs. Therefore this should be taken into account when customizing the application and

building the APIs. Relevant user guides will need to be created for each user so that once the application is complete the roles of these users will not become confused.

4.3 Structure of Resource Space

ResourceSpace is running on a server and has a management interface which is accessed from the web. It is programmed using PHP, MySQL, HTML, CSS and JS. This system can be installed onto Linux/Unix, Windows, Mac OS X, and Synology DSM and it works with the most web servers including Apache and IIS. It is possible to install a stand-alone version or a development edition from Subversion. ResourceSpace requires PHP greater than or equal to version 5, latest release recommended, and MySQL greater than or equal to version 5.0.15, latest release recommended. ResourceSpace has documentation online where one can find detailed instructions (ResourceSpace Documentation Wiki 2014).

4.3.1 Management Interface

The management interface, what a user of ResourceSpace is using once they log into the system, is fairly straightforward to understand and use. A few aesthetic changes were needed to be applied to it in order for the future museum admin to be able to quickly understand how to use the system. In addition, a screenshot of the dashboard was included in the user guide along with explanations. This screenshot can be seen below, Figure 4. The screenshot has been edited with Microsoft Paint and quick notes about its functionality have been inserted into the picture.

Figure 4: AudioResourceSpace dashboard. Screenshot Giova (2015).

In order for ResourceSpace to be able to accommodate only audio files and have the necessary metadata fields a few changes were done through the Team Center which can be accessed through the top navigation bar in the dashboard. In addition relevant rights were set for admin and general users. The instructions on how to make these changes were written into the developer's user guide in a step by step fashion. As a final step to the metamorphosis of the original ResourceSpace system, the application was renamed to AudioResourceSpace.

4.3.2 Storage of resources

Digital assets, called resources, are stored according to their IDs in a 'resources' table. The metadata fields like author, location or title are dynamically added to this table as they are created in the management interface. These metadata fields are linked by ids so the database can populate the fields based on the values that are set in the management interface. A simplified depiction of the database is shown below.

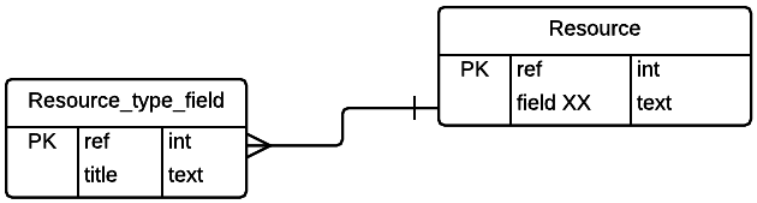

Figure 5: RS Database simplified.

By following this structure it is easy to add, edit, or remove the values that had been set for the metadata fields. In addition it is simple to modify or delete existing metadata fields as well as create new ones. The database updates its structure based on what is set in the web management interface by the admin, so the more custom metadata fields that are created, the bigger the table becomes.

There are many other tables in the RS database which are used to specify which collection resources are in, who has the rights to these resources, how many users there are, various reports, how many users have accessed the APIs, etc. The tables are very informative which simplifies the programming of the APIs, for example, as the relevant data can be fetched by joining or pivoting tables with MySQL.

In ResourceSpace resources are based on their ids. If a user sends an audio file into another user's collection then only a record of that action is saved into the database, the files are not duplicated or physically copied on the server space. In order to duplicate audio files these files would need to be downloaded and then uploaded. This in turn saves server space since the files are not created each time they are copied.

4.3.3 APIs

The basic installation of ResourceSpace contains two built in APIs, an upload and search API. Both were tested to see if they comply with the museum's requirements. The upload API would need a few minor additions, but the search API was fairly useless. Modifying it would require too many custom changes and this action interfered with the application's core code.

Therefore it was decided to construct a completely custom search API which would use only a few of the core files. The search API would also serve as a download API because when searching for files one of the options when listing their information could be to show a link to that searched file. This link would provide the functionality for downloading that file. Therefore search and download API will be called Search API and the ability to download files from it will be set in one of its parameters.

As discussed in chapter 3, Application Requirements, an authentication API will need to be created so that the mobile users would be able to receive an access key with which to use the upload and search APIs. The mobile users would need to send to the authentication API a username and password given to them by the museum admin. If the credentials would match with the approved ones then this user will receive an access key.

5 Custom API Design

According to the requirements three APIs were needed: an authentication (auth), an upload, and a search API. The authentication and search APIs would be created anew while the upload API will be modified since the existing code is acceptable to the requirements. The APIs have been coded procedurally and no PHP framework was used as neither ResourceSpace used any. The following paragraphs describe how each one works.

5.1 Auth API

The authentication API is needed specifically by the mobile users so that they can receive an access key which will in turn be used with the search and upload API. This authentication gives an access key to existing users of AudioResourceSpace, which means that only users that have been created by the museum admin are able to get that key. The mobile application will send a POST request to the API. This request should be in the form of JSON and the form of it is shown in Listing 1.

```
{"username":"xxxx","password":"xxxx"}
```

Listing 1: Auth request.

In the auth request, listing 1, the ‘xxxx’ stands for real values. The auth API receives these values and checks them against the database entries after scrambling the password using a hashing algorithm. This algorithm uses the SHA-256 hash on the MD5 hash which hashes the letters “RS” and the received username and password.

In the case of a match the API outputs an access key which is unique for the credentials that were sent in the request. The access key is output in JSON in the form as shown in Listing 2.

```
{"api_key":"xxxx"}
```

Listing 2: Auth response.

In Listing 2, just as in Listing 1 the ‘xxxx’ stands for real values.

This functionality lets the mobile application receive an access key as soon as they have received official credentials from the museum admin and thus proceed to use search and upload APIs.

5.2 Search API

The search API outputs all the audio files stored in AudioResourceSpace or specific audio files based on search parameters that are set along with the search request. This API accepts a GET request which is structured as follows (cf. Listing 3).

```
/plugins/api_audio_search/index.php/?key=insert key here&parameter
```

Listing 3: Search URL.

The table below defines all the available parameters of a search request, Table 3. The parameters are added at the end of the request.

Table 3: SearchAPI parameters.

Parameter:	Info:
	outputs the metadata of ALL resources
help=true	display help file (must read before developing mobile app)
format=[string]	displays all files with the set extension, e.g.format=wav or format=mp3, etc.
size=<50KB	displays files of size smaller than 50KB
size=<100KB	displays files of size smaller than 100KB
size=<200KB	displays files of size smaller than 200KB
collection=[integer]	Collection to search within, see below to reference which collection, e.g collection=1
category=[string]	Categories to search within, see below, e.g. category=nature (see below to reference which categories)
tag=[string]	Searches the tag field of audio resource for set string, e.g. tag=dog
link=true	Return the link to the original file so you can download it.
search=[string]	Return results of any valid search string, searches all metadata fields, e.g. search=wood
sound_type=[string]	Sound_types to search within, see below, e.g. sound_type=effects (see below to reference which sound_types)
resource_created_by=[string]	Search all resources created by a certain user, e.g resource_created_by=UserA

Parameters do not need to be written in any specific order neither is it required to have all of them in the request. Following are a few examples of the requests (cf. Listing 4).

- `url/plugins/api_audio_search/index.php/?key=`*`api_key_written_here`*`&help=true`
- `url/plugins/api_audio_search/index.php/?key=`*`api_key_written_here`*`&format=wav&link=true`
- `url/plugins/api_audio_search/index.php/?key=`*`api_key_written_here`*`&size=<50KB&category=human&search=dog&link=true`
- `url/plugins/api_audio_search/index.php/?key=`*`api_key_written_here`*`&format=mp3&size=<100KB&category=nature&sound_type=effects&created_by=museumAdmin&search=wood`

Listing 4: Search request examples.

As can be noticed from these requests, different parameters can be written to their end. A user can guess just by looking at this string what a request is searching for. The only requirements for structure of the request are that they contain an access key, which was provided by the auth API. Omission of this access key will lead to an error telling the user to read the user guide.

5.3 Upload API

The upload API lets users who possess a valid authentication key upload their audio files along with metadata they choose to transmit to AudioResourceSpace. This API is based on REST principles, using POST for requests and using JSON to output the response. To upload a file the following needs to be typed into the address bar (to form the URL) as shown in Listing 5.

```
url/plugins/api_upload/?key=insert key
here&parameter_values
```

Listing 5: Upload URL.

The parameter values are seen in Table 4 on the next page. The request will be visible in the address bar once it is executed.

Table 4: Upload parameters.

Value	Info
key=[string]	authentication key
userfile=[@file]	set the file path
resourcetype=4	default is 4
collection=[int]	collection id, required field see /help_collections.php
'metadata DB field'=[string]	see /help_metadata.php

The following is a sample upload URL (cf. Listing 6).

```
url/plugins/api_upload/?key=api_key_here&collection=3&resourc
etype=4&field8=Dogs&field73=barking&field74=woof+dog&field75=
human&field76=effects&field77=20&field78=05&field79=25
```

Listing 6: Upload URL.

Just like it was seen in the search API requests this request is just as informative and any user can guess what is being sent. There is no reference to the file which is being sent in the request because the file, or representation of the resource, is being sent in the body of the HTTP request (O'Reilly 2013, 61). The content-type which is used when submitting this file is called multipart/form-data and is set as the attribute to the form which processes the uploading functionality.

6 Testing

Initial testing of this system was done using cURL. Curl (written as cURL and pronounced see ‘URL’) is a command line tool with which web requests are made. In addition it can be written into a PHP file so the command line is not always necessary. These types of tests proceeded throughout development up until the application was transferred to the admin in charge of a trial installation on the server of Metropolia. The final testing was done on the server of the Museum of Technology in Finland.

6.1 Auth API

The cURL tests with this API helped improve its functionality where needed. Improvements were adding error messages in case of wrong credentials as well as optimizing the code. Since the development proceeded simultaneously with the cURL tests the testing of this API was dynamic, meaning that an error was fixed as soon as it was encountered up until the point that no errors occurred.

Testing on Metropolia’s server produced no errors. In other words this means that this API worked as expected and fulfilled all of its requirements.

6.2 Search API

The Search API worked as planned while testing it with cURL. However, when testing it on Metropolia’s server a major flaw was discovered. This flaw was not so much in the functionality, but in the understanding of how audio files will be shared amongst users in AudioResourceSpace. What was understood was that the museum admin would create new guest users and upload audio files directly into their accounts. However, the requirement was to let the museum admin share files from their own collection with the mobile users. The problem was that an audio file that had been shared by an admin to the collection of another user would not be seen by the Search API at all.

In order to fix this the majority of the code needed to be modified because completely different tables should have had been accessed when getting reference ids to those shared files. In addition those shared files need to be cross referenced with the admin’s

private collection as well as with any duplicates. After modifying the API to suit the most recent requests a subsequent test proved successful and the API functionality was accepted.

6.3 Upload API

Tests with upload worked as expected since this API was well designed by the original developers of ResourceSpace. There was one minor detail which needed to be addressed and it was regarding the size audio property that needed to be stored in the database. To fix this a simple addition of retrieving the size from the uploaded file as well as inserting it to the database was sufficient.

7 Finalizing Application for Client

In order to make this system complete for the client, in addition to providing the code, relevant user guides needed to be created. These user guides needed to be tested with a new installation which was done on the development server under a different directory. In addition feedback from the museum server's admin, the museum admin and the mobile application programmers was received.

7.1 Compiling User Guides

The following user guides were created:

1. AudioResourceSpace Installation Guide – aimed at server admin.
2. AudioResourceSpace User Guide – aimed at museum admin.
3. Api Auth User Guide – aimed at API application developers.
4. Api Audio Search User Guide – aimed at API application developers.
5. Api Upload User Guide – aimed at API application developers.

All of these guides were distributed amongst the appropriate parties as well as included in the final disk which contains the installation files in addition to the user guides.

7.2 Trial Installations

In order to test the server admin's and museum admin's user guides it was necessary to perform a few brand new installations of AudioResourceSpace. Those were implemented while following the steps outlined in the guide and simultaneously rewriting any instructions that were incorrect or not clear enough. In total five new installations of AudioResourceSpace were required to achieve a flawless install and functionality. At some point a glitch occurred in one installation that could not be explained or fixed. However the next installation did not have that glitch even though the install files were identical. This occurrence is left as an unexplained mystery.

8 Feedback

8.1 Feedback from Developer

The developer is whoever manages the museum's server, installs AudioResourceSpace on it and modifies the system to the museum's requirements. Shortly after commencing the installation process the following error occurred (cf. Listing 7).

```
Sorry, an error has occurred
Please go back and try something else.
You can check your installation configuration.
/var/www/resourcespace/include/db.php line 221: Cannot
modify header information - headers already sent by
(output started at
/var/www/resourcespace/include/config.php:143)
```

Listing 7. Error output code.

This happened because the user guide was not clear enough on the instructions regarding input of relevant installation configurations. The input concerning installation configurations such as database name and password happens online. The developer assumed that the files that are located on the server need to be edited manually. This produced an error because the system is not designed to be installed this way and the developer became very confused.

To rectify this problem the instructions in the user guide were written more clearly and a new copy of the user guide was delivered to the developer. Using the updated instructions the installation proceeded without errors.

In addition the developer wished to remove the link with which a new user could apply for an account for the AudioResourceSpace system. This function is unnecessary with the given requirements so it was removed for the time being by making it not visible with CSS. The code snippet was sent to the developer so the relevant CSS file can be

altered. This ensures an easier modification in the future, in case the requirements change.

As a final test, once the installation was complete, the system and the APIs were tested by this developer. The developer uploaded a few audio files to the system, created a few extra users, and shared some of the audio files with those users. It is not known how the API was tested, however the developer assured that it worked properly. In conclusion, the results of using AudioResourceSpace by the developer were satisfactory and the system worked flawlessly.

8.2 Feedback from Museum Admin

The museum admin tested the system according the user guide provided. This user guide was specifically written for the museum admin containing step by step instructions on how to upload files, create other users, share files with those users and edit metadata. There were no errors or questions regarding the installation. In addition to the step by step instructions that the museum admin followed, in the same user guide file are general instructions which cover any possible problems that the admin might run into. It might be impossible to anticipate all the problems that might happen on the backend or the front end of AudioResourceSpace. However, the majority of errors has been anticipated because so much time went into learning and testing this open source system, ResourceSpace.

8.3 Feedback from Mobile App Developers

The group of students who were developing mobile apps which would be accessing AudioResourceSpace are the mobile app developers. The feedback received was that they wished the API would output an empty array in JSON if the results for search were null. The response they got from a search that produced no results is shown in Listing 8.

```
Unexpected token T
```

Listing 8: Response error.

When the search API was created, the output for null results was a custom error message explaining what the problem was and which parameter caused it. This error message was output as normal text instead of JSON. Since the developers requested an empty JSON array to be output the search API was modified in such a way as to produce this result. Instead of outputting an error message the below code would be used (cf. Listing 9).

```
echo json_encode(array());
```

Listing 9. Empty JSON array.

The search API had to be modified in numerous places because there were a lot of instances when such an output might be needed. After modification the API was tested and it seemed to work without error.

Regarding the other APIs, the upload and auth, there was no feedback so it is assumed they work as they should.

9 Discussion and Evaluation

The most important part of this project was to understand the requirements of the museum and implement them correctly when customizing and building the audio storage system. It was challenging to find a suitable open source system because there were quite many so it took quite a long time to test them out before deciding which one will be used, ResourceSpace. In addition it was not very clear what types of users will be using the system and if those users will be created as part of that system or independently from it. Towards the middle of the project it was made clear that the mobile users will not be part of AudioResourceSpace (former ResourceSpace) which led to a rush to design and create suitable APIs for them.

As a developer of this system and all its ensuing components it was necessary not only to have skills in coding, but as well in organizational techniques. As described in an article about DAMs (Roskiewics 2005, 5), the author states that the person or team responsible for creating and successfully maintaining such a system should understand all “the interdependencies of all of the components in the system and the support path for each is important”. In order to achieve that one needs to be able to prioritize the goals of the project as well as be able to adapt to any changes that might arise during development or alterations of requirements.

Taking into consideration the feedback from the developer the system developed for the final year project was a success. There was just one minor error during installation, which was more a technicality in understanding the instructions than a major flaw in coding. The user guides were one of the hardest things to write because every possible situation that could occur with this system had to be imagined and written about. This process took quite a lot of time and tedious work, especially with installing the system over and over again.

The feedback from the mobile application developers was of most use. It is assumed that their application receives all responses from the API in JSON format. This means that the programming of those applications is done in such a way that JSON is parsed and any other format is unrecognizable or produces an error. Since they did say that they receive an unexpected token if there are no search results the above assumption is well justified. Such an error would produce quite a lot of unnecessary coding for the

mobile application developers. It would be much easier if the response was in the correct format.

Overall this project proceeded quite slowly. In the beginning there was a lot of research to be done regarding open source management systems, metadata standards and web services. There was quite a steep learning curve for determining how ResourceSpace works as an application because it had multiple features and it was not very clear which features would be used and which will be ignored. Towards the end of the project the requirements became clearer and as mentioned earlier there was a rush to fulfill them. This resulted in the final changes being done long after the project's allotted time was concluded.

10 Conclusion

The purpose of the final year project, or thesis, was to create a system for the Museum of Technology in Finland. This system would be able to store and handle digital assets in the form of audio files. It was preferable to create an application with minimal resources in terms of time, personnel and costs. One way to approach this would be to find a suitable open source digital asset management system, modify it accordingly and create any extra functionalities that it might need.

The system that was chosen was open source and it was able to store and manipulate the audio files as required. The extra functionality, in the form of custom APIs was created and after testing showed to be working as expected. This system is designed in such a way that a subversion installation of it goes to the client which ensures constant updates and patches. With the customizations performed this option was not possible as any official updates to the system would render the APIs dysfunctional. This might bring problems in the future with security or programming language incompatibilities.

In a perfect scenario a totally new system would have been preferable to be built. This way it could be designed specifically for the requirements of the museum. However in the long run it would also need to be updated, which in a way negates the argument of building a personally designed digital asset management system. In either scenario the DAM will need to be regularly updated to ensure security and functionality.

One of the advantages of ResourceSpace is that it is very versatile and can accommodate different types of assets. Therefore this thesis could be used a base for the next development of an open source digital asset management system for another client. Obviously a lot of changes will need to be made and a newer version of the application installed, but at least after reading what types of changes were made for this version the next developer can infer what might be expected of them.

Overall this project was a success and fulfilled all of the requirements. The only way to see how well it actually does in practice is visit the museum and become part of their audio exhibition.

References

Abeysinghe S. RESTful PHP web services. Birmingham, UK: Packt; 2008.

Giova D. AudioResourceSpace [screenshot].
URL: http://dev.mw.metropolia.fi/dianag/AudioResourceSpace/pages/home.php.
Accessed 20 October 2015.

Jacobsen J, Schlenker T, Edwards L. Implementing a digital asset management system. Massachusetts, USA: Elsevier Inc; 2005.

Kavanagh P. Open source software. Boston, Massachusetts, USA: Elsevier Inc; 2004.

Mitchell L. PHP web services. Sebastopol, CA, USA: O'Reilly Media, Inc; 2013.

Open Source Initiative [online]. 22 March 2007.
URL: https://opensource.org/osd.
Accessed 10 June 2010.

ResourceSpace documentation wiki (2014) [online].
URL: http://wiki.resourcespace.org/index.php/.
Accessed 23 October 2015.

Riszkiewicz R. A survival guide to customizing and outsourcing a DAM system. Seybold Report: Analyzing Publishing Technologies 2005; 5(1):5-11.

Sarwan Naresh (2014) Open source digital asset management [online].
URL: http://www.opensourcedigitalassetmanagement.org/reviews/available-open-source-dam/.
Accessed 2 June 2015.

Sutter R, Notebaert S, Walle R. Evaluation of metadata standards in the context of digital audio-visual libraries. In: Gonzalo J, Thanos C, Verdejo M, Carrasco R, editors. Research and advanced technology for digital libraries. Berlin, Germany; 2006. p.220-231.

Zeng M, Qin J. Metadata. London, England: Facet; 2008.

Press Release: The people's smart sculpture

PRESS RELEASE

The People's Smart Sculpture

Co-funded by the
Creative Europe Programme
of the European Union

December 2014

Start of smart participation project, co-funded by the Creative Europe Programme of the European Union:

The People's Smart Sculpture

The People's Smart Sculpture – Social Art in European Spaces is a creative research and innovation project about the cultural evolution of the European city of the future. It addresses the growing complexity of life in today's city spaces and imminent challenges to the development of the urban environment. The People's Smart Sculpture PS2 explores the possibilities of participation that will become a smart culture technique as a result of the ongoing digitalization of society. 12 partners – including universities, museums, galleries, theatres and research institutes – in 8 European countries will organize 11 connected open labs integrating new art, design thinking, science, smart technologies and user culture for the participatory re-design of urbanity. The project has a budget of 2 million Euros and is funded by the European Commission within the Creative Europe programme for 3.5 years.

11 creative experiments in participatory art and design for the city of the future

The 12 project partners will implement 11 experimental sub-projects and a European study about new forms of participation. While some PS2 sub-projects shed light on the ways we perceive our city space, or create speculative city environments, others will analyse problems, identify challenges and explore interdisciplinary solutions with citizens. The variety of approaches will reflect the diversity of people, skills, urban art, social processes and urban development. Renowned artists and designers from 29 countries will participate in the sub-projects. PS2 will explore and document new strategies for involving digital media and ICT in the development of user-centered culture.

Development of new forms of participation for Smart Cities

Scientists from media-labs, computer science, cultural science, art history, sociology, architecture, design and urban planning will engage with the creative processes. Digital technologies will not only play an important role in the PS2 project art activities themselves, but directly support the innovation process by offering new opportunities for empowerment and societal integration of people of all social groups. The project will connect people and foster the exchange of ideas about and for smart cities. It is the base for cutting-edge communication between science and art, creatives, artists, media designers and citizens, and between the people and their governments. At the same time it will motivate the broad dissemination of new skills, design expertise and social knowledge relevant to urban re-design.

The People's Smart Sculpture

The University of Applied Sciences Bremen is the lead partner of the project. The lead coordinator is Martin Koplin, director of the M2C Institute for Applied Media Technology and Culture. The partners are: Helsinki Metropolia University of Applied Sciences, GAUSS Institute, National Institute and Museum Bitola, Kristianstad University, Warehouse9, Museum of Broken Relationships, Oslo Barnemuseum, University of Oslo, University of Applied Sciences Düsseldorf, Gdańsk City Gallery.

Contact:
Martin Koplin
Coordinator The People´s Smart Sculpture PS2
Director / CEO
M2C Institute for Applied Media Technologies and Culture
at the University of Applied Sciences Bremen
Phone: +49-421-5905-5402 E-Mail: koplin@m2c-bremen.de Web: www.m2c-bremen.de
Post address: Flughafenallee 10, D-28199 Bremen, Germany, EU

Open Source Definition (Open Source Initiative, 2007)

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

Email from Telemeta's developer

Guillaume Pellerin notifications@github.com

Thu, Jun 4, 2015

Hi Diana,

The next release of Telemeta coming in a few days will introduce a new install procedure based on docker which will make it compatible with windows.

For now, and as a developer, you can grab and start it with our DevBox: https://github.com/Parisson/DevBox

Please give us some feedback if any trouble (it has not been tested so much on Windows). Thanks!
Guillaume

First version of metadata

1. partial path to file
2. origin of recording (location)
3. timestamp of recording
4. description
5. recorder with¨
6. sample rate
7. filesize
8. bitdepth
9. channels
10. filetype (wav, mp3, etc..)
11. uploaded timestamp
12. uploaded by
13. average bytes per sec
14. bits per sample
15. keywords/tags
16. length
17. artist
18. author
19. sound type
20. category
21. publisher
22. copyright message
23. title
24. subtitle
25. language
26. original filename
27. media type
28. release year
29. place of recording
30. geotag