International Journal of Current Engineering and Technology
ISSN 2277 - 4106
© 2013 INPRESSCO. All Rights Reserved.
Available at http://inpressco.com/category/ijcet

Research Article

APIS to insert Content into an Open Source Web Radio System

Sajid M. Sheikh[*a], Anselm Mathias[a], Annah M. Jeffrey[a] and Shedden Masupe[a]

[a]Department of Electrical Engineering, University of Botswana, Gaborone, Botswana

Accepted 04 August 2013, Available online10 August 2013, **Vol.3, No.3 (August 2013)**

Abstract

Application developers can use APIs to integrate different functionalities into their application to access information and data or insert content from other applications. Airtime is one of the most popular Open source radio management system used. However, the Airtime application did not have any integration or automation facilities that could be used by Third party and micro service providers. This research project was therefore aimed at developing codes/ APIs in order to achieve this task. The paper presents developed APIs to insert content and data into an open source web radio system, in order to allow any developer of micro services to have the possibility to easily use the Web Radio. A systematic development process was followed in designing and creating the APIs. The Airtime application was installed on a local Linux server using Apache, PostgreSQL and the Restler Framework. Data models were then defined for handling input and output responses by the system and the API's that were to be built. PHP code was written to address the tasks that the APIs need to perform using the Restler Framework to provide REST based implementation and integration with the Airtime application using the HTTP protocol. Testing was done using error codes. Where errors were encountered, the PHP API codes were revisited and errors traced and modified until they performed the tasks they were designed for. By reverse engineering the Postgre SQL database that supports the airtime application, suitable functions were designed and then developed to meet the objectives of inserting content into an open source web radio. The API functions that were created are track/download, track/upload, playlist/create, playlist/ edit, playlist/add track and playlist/remove track. The developed APIs allow developers to upload their own tracks, media like interviews and news broadcasts to be broadcasted on the radio. The APIs also allow developers to list the line-up of a show and change the playlist if necessary. With the assistance of the Restler (Luracast) framework, Object Oriented PHP code and SQL, scripts were successfully written and tested to provide the required functionalities.

Keywords: *Airtime, API, Open Source, PHP, Radio, Restler, SQL, Web Radio.*

1. Introduction

Application Programming Interfaces (APIs) are commonly known as web services which are accessed via hypertext transfer protocol (HTTP) from a web browser, and executed on a remote system hosting the requested services from an Application or other codes. A software company releases its API to the public so that other software developers can design products that are powered by its service. For example, Google uses APIs for showing weather reports, website visit statistics and many others. Application developers can use APIs to integrate different functionalities into their application to access information and data, or insert content from other applications. An API is a software-to-software interface and is not a Graphical User Interface (GUI). APIs allow applications to interconnect and thus create additional functionalities that are like frameworks that can be used by application developers. Computer programmers promote automation, either by use of scripts that run once
a specific event has occurred or developing multiple open ended APIs that do a variety of tasks which are integrated into a large range of products.

Applications can be either open source or closed source. Closed source softwares are developed by a single person or company. Only the final product is made available, while users don't have access to the source code. Closed source software is normally copyrighted. Open Source software on the other hand, is almost the opposite as it is free to use (most), users have access to the source code if they want to get it and they can modify and make changes. Most open source projects are being worked on by developers who do it for fun and in their own time.

Sharing code (open source-architecture) gives other developers an opportunity to add more functionality to a product, instead to spending time accomplishing a task that is already done by someone else.

*Corresponding author **Sajid M. Sheikh** & **Annah M. Jeffrey** and **Shedden Masupe** are working as Senior Lecturer; **Anselm Mathias** is a BE final year student

Radio is a traditional medium of mass communication and the development of Web Radio brings traditional radio to the digital era through access of Radio over the Internet using web browsers. Technology and software advances have resulted in automating many processes that were in the past carried out manually. There are many Web Radio Management Systems such as Airtime, Shoutcast, ampache Firefly, Darwin Streaming Server, Sam broadcaster, Dad by Enco, RCS and Audio Vault FleX. Airtime is one of the most popular Open source radio management system used. However, the Airtime application did not have any intergration or automation facilities that could be used by Third party and micro service providers. This research project is therefore aimed at developing codes/APIs in order to achieve this task. Different API Architecture's were scrutinised to find a suitable design that could be used in the development of the APIs for the Airtime software.

New content is generated and at an alarming rate and old content is stored for future use. Thus it is vital that such new information be added into the Radio broadcast streams, especially crucial information such as news, weather, etc. This task is very daunting if done manually hence it is imperative that an application be created to dynamically facilitate the addition of information. The focus of this research was thus, to create APIs to insert content into the Open Source Airtime Application.

2. Objectives

The goal of this research is to study and develop APIs to insert content into the Airtime system in order to allow any developer of micro service to have the possibility to use a webradio. Any php programmer should be able to use the php codes developed to interact with the web radio.

The envisioned implementation is such that the APIs are to be built within the Airtime application, and should be available only to developers that are authenticated by the Emerginov platform. The developed APIs need to be able to insert content into the Airtime Application from outside the application. Developed APIs would have the following functionalities

- **Upload and download of tracks**

This would allow developers to upload their own tracks, media like interviews and news broadcasts to be broadcasted on the radio. Further functionality can be added such that this service would allow other developers to download tracks in the form of podcasts.

- **Create, Edit and remove playlists**

This would allow the developers to list the line-up of a show and change the playlist if necessary. The project aimed at the implementation of the Architecture of the APIs, the design of the APIs and testing of the APIs, in order to provide APIs that can be used by and distributed with the application. These APIs will allow developers to integrate Airtime functionalities into their own applications that are being developed.

3. Paper outline

Section 4 introduces a few online radio management systems, while Section 5 gives a brief overview of the research that was done on existing web radio APIs. Section 6 gives an overview of the API framework and guidelines that were studied. This section briefly presents the main web application framework that were studied, learned and used to code the APIs needed to provide the required functionalities. Section 7 presents the methodology that was followed to develop the APIs, as well as the APIs development Life cycle. Section 8 presents data manipulation results using the specified HTTP method call used for each service and the API coding. Each service is broken down into a brief description of what the service is supposed to do, followed by the parameters that are required to process the request. Section 9 presents the results from the testing that was carried out. Discussion, future works and an overview of the challenges encountered and achievements are in Sections 10 and 11. References follow at the end of the paper.

4. Online radio management systems

There are many open-source and closed source radio management systems that are existing. This section briefly presents a few such systems, namely: Campcaster, Airtime, Shoutcast and Ampache. Other web radio management systems not discussed here include radiocommands, livewebdj.com, shoutcheap.com, Firefly, Darwin Streaming Server, Sam broadcaster, Dad by Enco, RCS and Audio Vault FleX.

A. Campcaster

Campcaster is an open source Radio Management system application for live broadcasting, remote broadcast automation (via web-based scheduler) and program exchange between radio stations. In January 2011, Sourcefabric (team that was working on the updates of Campcaster) announced a rewrite of Campcaster. The new product is called Airtime. It replaces the C++ scheduler of Campcaster with Liquidsoap and includes a drag and drop web interface based on jQuery.

B. Airtime

Airtime is a Linux based free open source radio management system for remote broadcast automation (via web-based scheduler) and program exchange between radio stations. Airtime runs via the web, through a browser, allowing for remote collaboration on radio content. The latest version, 2.3 of Airtime supports 11 languages and allows for stations to localise Airtime themselves. It is a very popular application is use by web radio stations. Airtime generates audio streams using the

Liquidsoap stream-mixing framework, and can output the results simultaneously to Icecast or Shoutcast servers or through a sound card.

C. Shoutcast

Shoutcast has a radio directory of approximately more than 48,000 radio stations from all around the world. The company is owned by AOL. It develops and releases its own shoutcast APIs to provide developers with more advanced options for integrating music streaming services into web services and mobile applications using RESTful calls formatted in XML, JSON and RSS.

D. Ampache

Ampache is a web based audio/video streaming application and file manager. It allows one to access their music and videos using an internet enabled device. Ampache's usefulness is heavily dependent on being able to extract correct metadata from embedded tags in your files and/or the filename. Ampache's down side is that it is not a media organiser like Airtime .

5. Web radio APIs

Shoutcast, as mentioned in the previous section, creates its own APIs to support its Radio Management System. RadioReference.com provides a set of APIs for webmasters and application developers to integrate data from radioreference.com into their own products. There are three different APIs currently available

- SOAP Based Web service for Radio Reference Database Data
- Javascript Remote Render Service for Radio Reference Database Data
- XML/JSON based Web service for the live audio feed catalog.

Radio player API are also web radio APIs created to allow web developers to incorporate the playback of licensed Rdio music into their web applications. Rdio subscribers and trial users will be able to hear full-length songs and non-users will be able to hear music previews. They consist of an Adobe Flash file (SWF) with simple JavaScript and ActionScript 3 APIs. It can only be used in compliance with the Rdio API terms and conditions. Lastfm, another webradio has created APIs for paid up members. These APIs require authentication to use them. Once authenticated, one can tune the radio using the radio.tune API method. An example of the syntax is shown. This takes a ***station*** parameter that must correspond to a last.fm protocol station url. lastfm://<stationtype>/<resourcename>/<station-subtype>

6. API development framework and guidelines

A framework is a universal reusable software platform to develop software or APIs. These include support programs, compilers, codes, libraries, APIs and tool sets. This section briefly presents the main web application framework that were studied, learned and used to code the APIs needed to provide the required functionalities. It is necessary to understand the underlying systems that would support the API, or specification/guidelines that the API's would be built on. This purely means the protocol used for information transfer, the architecture and finally the output format that the API's would use.

A. HTTP Methods

The HyperText Transfer Protocol (HTTP) is also known as RFC 2616 and was designed to enable communication between clients and servers. HTTP uses verbs (methods) to achieve a specific task or manipulate a specific resource. Some of these verbs are GET, POST, PUT, and DELETE.

The actions performed by these verbs are based on their name, where GET requests a resource from the server and POST submits information to the server. The line below is an example of GET that assigns "variable 1" the value of "value1", and "variable 2" the value of "value2".

```
/test/form.asp?variable1=value1&variable2=value2
```

The same thing can be achieved using POST as shown in the snippet code below

```
POST /test/form.asp HTTP/1.1
Host: localhost.com variable1=value1&variable2=value2
```

There are many formats in which the system can give an output such as XML, JSON, plain text, and CSV. Thus, it becomes very crucial to understand and select a suitable output format. It should be noted that plain text and CSV are not used in web-oriented architectures.

B. XML

Extensible Markup Language (XML) is a specific form of writing text, which is defined by the World Wide Web Consortium(W3C). It was designed to be easily implemented and used with HTML; it was also designed to be both human and machine-readable. XML has been used as the core language in communication of different protocols. XML uses tags – however, these tags are not predefined. The tag is defined by either a person or the automated machine.

C. JSON

JavaScript Object Notation is a text format that is designed to be human readable and is used for interchange of data. JSON is derived from and supported within Javascript, while XML requires libraries to retrieve additional data. JSON is also language independent. It was designed to be minimal, portable, and textual.

Through implementation and case studies, it was realized that JSON is considerably faster as it uses less

resources, is easier to read and interpret by a human, and generally, simpler to use than XML. It is crucial to understand that the APIs sole purpose is to manipulate data, where JSON fits more naturally as it is data-oriented in design. Hence, JSON format was used to display the output in this project.

D. SOAP-Based Web Service

Simple Object Access Protocol (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It makes use of XML technologies, by mostly relying on the application layer, and can be used over any transport protocol such as TCP, HTTP, SMTP, or even MSMQ. Developed by a group of vendors such as Microsoft, IBM, and Lotus, SOAP became a W3C Recommendation on 24th June 2003 .

At the core, SOAP defines a way to move XML messages between two systems. The SOAP message consists of a header element made up of header information, a body element that contains call and response information, and a fault element containing errors and status information. The actual SOAP message is shown in Figure 1.

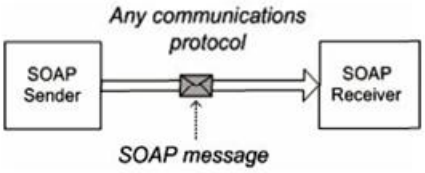

Figure 1: Simple SOAP messaging

In SOAP, XML is used to provide an output that is acquired from any script. SOAP provides flexibility such that other protocols can be stacked over the HTTP, which is used as the transportation protocol. The above advantage can also pose as a security risk, as the SOAP standard also does not have any in-built security facilities. SOAP also has other problems, in terms that the service is only one-way (from client to server), and that it is slower than other technologies.

E. RESTful Web service

Representational State Transfer (REST) is a style of software architecture for distributed hyper media systems such as the World Wide Web. REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions. A general construct is shown in Figure 4. Roy Thomas Fielding coined the term "Representational State Transfer", in his Doctorate dissertation entitled "Architectural Styles and the Design of Network-based Software Architectures".

An alternative way to distinguish and define REST would be to say that it is a stateless server. Each request from a client contains all the information necessary to service the request, which clearly separates the clients and servers.

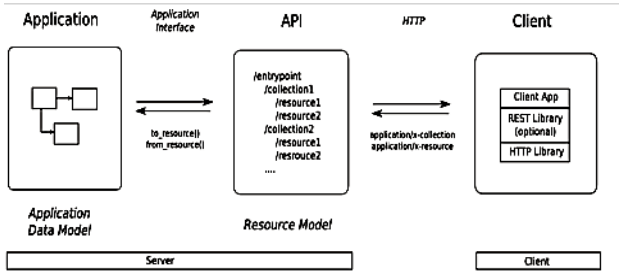

Figure 2: REST Relationship diagram

REST clients can cache responses if allowed. There is a direct interface between clients and server, and the client can then request code from the server and execute it on demand.

REST has a loosely typed architecture, but uses nouns and verbs to promote flexibility and usability. REST is not restricted to parsing out only XML, it also uses less bandwidth and provides error checking.

In order for REST APIs to work, it should be able to identify the resource e.g. by using the Universal Resource Indicator (URI), and be able to manipulate the resource through representation (i.e. modify or delete), using the HTTP request of GET, POST, PUT, and DELETE. REST's own flexibility can be a problem, as there is no common accepted standard for REST, thus interoperability between large systems might be a problem.

By using the above architectures, it clearly demonstrates how the system would get the data, and how it would respond. By understanding the advantages and disadvantages of each type of entity, the REST architecture was selected, as it would perfectly fit the scalability needs of the application.

F. Restler framework

There are many Frameworks that are designed for various reasons and in different languages, such as Tonic, Restler, Slim, FRAPI, Zend Framework just to name a few. From these many options, the Restler framework was selected.

This web-API framework provides easy deployment of the REST architecture. It is prebuilt using different applications to provide an inbuilt solution, thus avoiding the re-writing of the code. It is very simple to use if object-oriented programming of PHP is already understood. Restler also supports different formats like JSON, XML, yaml, amf and plist.

G. Emerginov platform

Emerginov is an open-source solution developed by Orange Labs, which provides the opportunity to build applications with the use of service such as SMS and Voice. One such platform is located locally in Botswana, for the encouragement of local industries to use, create solutions, and nurture ideas in fields such as m-health and

agriculture. This platform was used to demonstrate the capabilities of the APIs developed for this project .

H. Guidelines

In designing APIs there are no fixed rules, however, there are the recommended practices that have been perfected over the years of development by professionals. The guidelines below have been used in the code design and implementation.

- Structure - Normally APIs follow the standard design procedure shown in the Figure 3

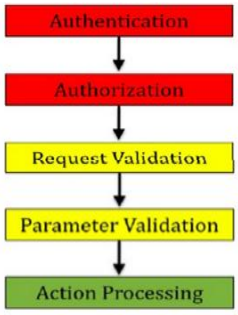

Figure3: Anatomy of API

- Versioning - It is very important to provide versioning as it and helps in large scale development without hindering older users. For example, Facebook versions such as "?v=1.0"
- Status codes - Use of HTTP status codes to relevant standard-based codes for errors. For example, Google uses 200, 201, 304, 400, 401, 403, 404, 409, 410, 500. Information that is provided by the server should be as descriptive as possible, to relay any problems that have occurred which needs to be rectified.

For example

{
“status”: 409
“property”: “name”,
“message”: “A directory named ‘ Avengers’
already exists”,
“developerMessage”:”a directory named ‘Avenger’
already exists. If you have a stale local cache, please expire it now.”
“moreInfo”:www.checkitt.com/errors/31337
}

For the code designed, the inbuilt Restler error codes were used.

- Authentications - Sessions are to be avoided whenever possible and provide authentication for every request if necessary. Existing protocol such as Oauth 1.0a or Oauth2 or SSL are to be used. In this project, the authentication is done by the Emerginov platform.

7. Methodology

A. Web Radio APIs Development Cycle

Figure 4 below shows the systematic development process that was followed in designing and creating the APIs. The Airtime application was installed on a local Linux server using Apache, PostgreSQL and the Restler Framework. Data models were then define for handling input and output responses by the system and the API’s that were to be built. PHP code was then written to address the tasks that the APIs need to perform using the Restler Framework, to provide REST based implementation and integration with the Airtime application using the HTTP protocol. The outputs created were in JSON format. As the last phase of the cycle, testing was done using error codes. Where errors were encountered, the PHP API codes were revisited and errors traced and modified until they performed the tasks they were designed for.

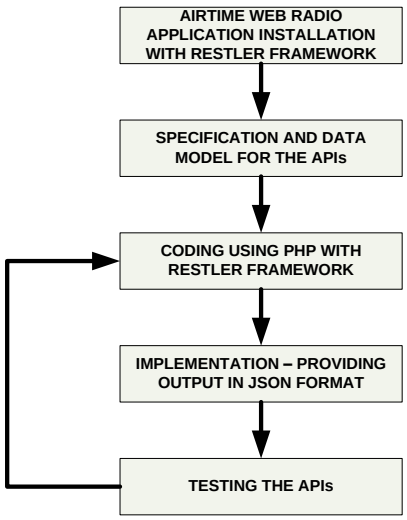

Figure 4: Web Radio APIs Development Cycle

The Airtime application operates with a database running to store all its data. The APIs were to be built by manipulating the PostgreSQL Database on which the application was built upon. Figure 5 shows the design of the proposed Implementation. The APIs that needed to be developed were to be built within the Airtime application, and be available only to the developers that are authenticated by the Emerginov platform. The final APIs developed are only for developers. Users or listeners are not aware of the coding and complexities and listeners will only be able to access broadcast through Icecast.

B. AIRTIME Database Model

The manipulation of the application to develop the APIs is done through the database. Thus, it is important to understand the tables and the information each field holds. The complete relational database is shown in Figure 6 presented at the end of this paper. The database has a complex structure with a relational database structure linking many primary keys from different tables. In order to manipulate the data on the server side and create the APIs, it is important to understand the relationship between the tables found in the database. An SQL

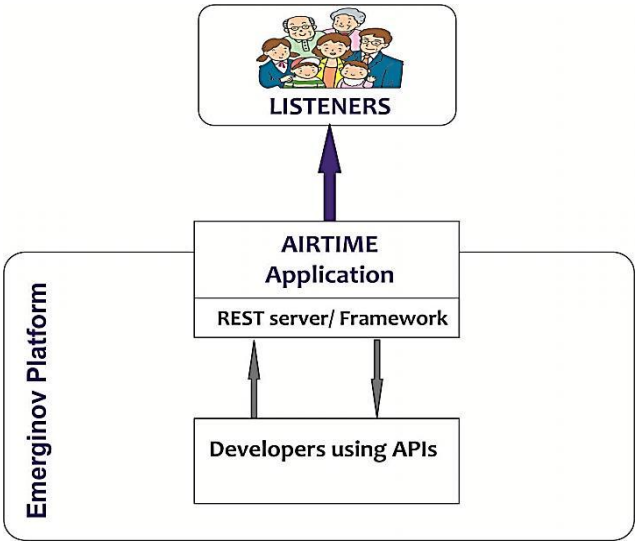

Figure 5: Design of proposed implementation

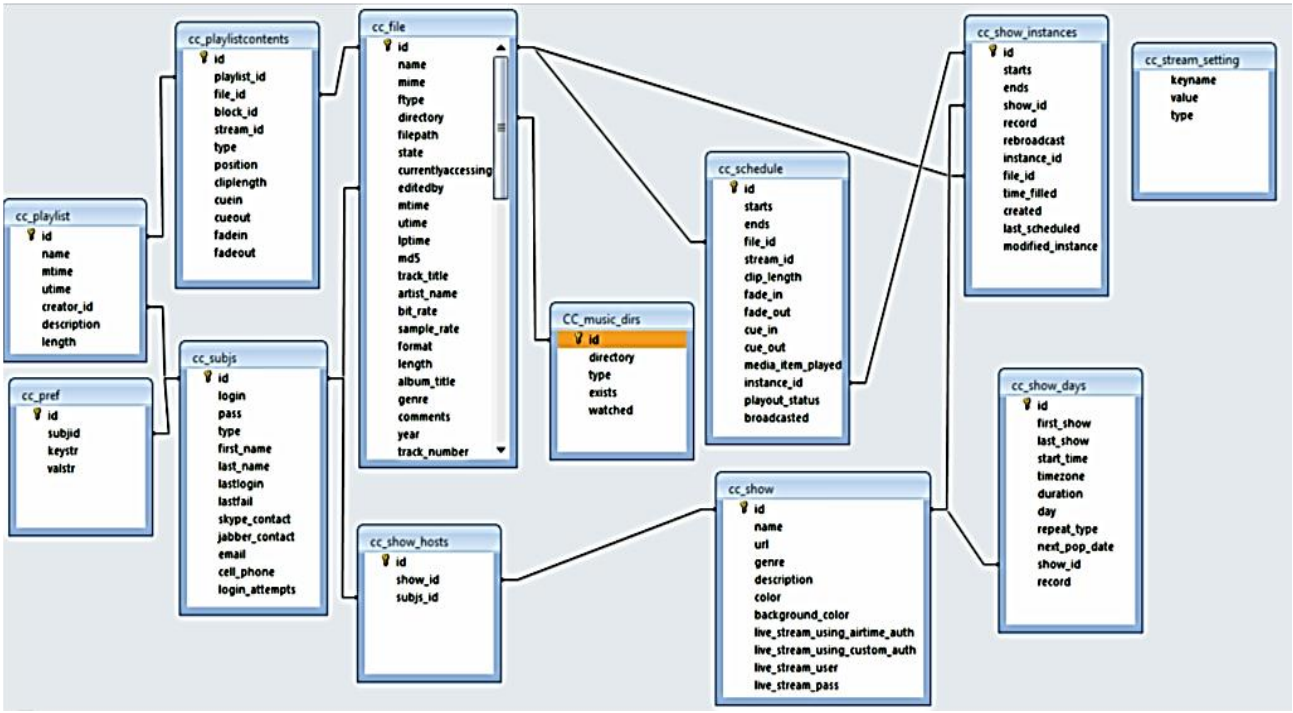

Figure 6: Relationships of table in the database

command that is used when changing data in one table from the API codes is "JOIN". This command ensures that when data is changed in one table, the related table should also be changed.

For illustration purposes of the complexity of the relational database and how it is used, the cc_ show table is shown below. When a show in the Radio Management system namely Airtime is updated, its details are stored in the table labelled "cc_show" shown in Figure 6 below. This table contains information such as name of the show, aesthetic information on how the show will look like in the calendar and options for authenticating users who would join into the live stream.

C. Proposed Error Codes

The error codes are important as they are a means to provide insight into the inner workings of a code without going through the code itself, in case a problem arises from the use of the code. The most standardised error messages were written for the HTTP, from which these error codes were derived.

Table 1 below describes all the errors codes and the description associated with them, which were used in the development of the APIs. These are few of the error codes that are built into the Restler framework, that are going to be used in the APIs.

cc_show[7]:

Column	Type	Not Null	Default	Constraints
id	integer	NOT NULL	nextval('cc_show_id_seq'::regclass)	
name	character varying(255)	NOT NULL	''::character varying	
url	character varying(255)		''::character varying	
genre	character varying(255)		''::character varying	
description	character varying(512)			
color	character varying(6)			
background_color	character varying(6)			
live_stream_using_airtime_auth	boolean		false	
live_stream_using_custom_auth	boolean		false	
live_stream_user	character varying(255)			
live_stream_pass	character varying(255)			

Figure 7: Table cc_show

Table 1: Proposed use of errors

Error Code	Error Text	Description
200	Ok	This is when the request has been successfully carried out.
201	Created	Resource has been successfully modified.
400	Bad Request	The request cannot be fulfilled due to bad syntax.
404	Not Found	The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible.
500	Internal Server Error	This is when either a connection or query has failed to get the required information.

8. Results

This section briefly shows the php coding developed to create the required APIs to perform the required tasks. Table 2 found at the end of this paper shows the inputs, methods and outputs associated to an API function. The output results contain the information that will be obtained by the use of the specific API.

In order to produce the required results, object-oriented programming of PHP had to be used by the use of classes. The sample extracts of "index.php" are shown below. This page is accessed through following "http://localhost/apiv1/index.php" on the local machine after deployment of Airtime. The "index.php" is the main page where any site starts. As soon as the web server calls on the "index.php" file, the Restler framework shown by the snippet below was loaded.

```
require_once '../vendor/restler.php';
  use Luracast\Restler\Restler;
```

A new instance of the object Restler is created, and then parses the data that is required for the class function to be acted upon as shown below. This is how Restler is controlled and works. The default format is XML, so Restler has to be manually set to JSON format

```
$r = new Restler();
$r->setSupportedFormats('JsonFormat');
$r->addAPIClass('playlist');
```

The API then acquires the credential stored in the "/etc/airtime/airtime.conf" file, required in connection to the PostgreSQL database which is stored in an array called “airtime_config” as shown below.

```
    $airtime config                                    =
parse_ini_file("/etc/airtime/airtime.conf");
```

The snippet below shows when a connection called “dbconn” is established with the credentials acquired.

```
    $dbconn=
pg_connect("host=$airtime_config[host]
    dbname=$airtime_config[dbname]
user=$airtime_config[dbuser]
  password=$airtime_config[dbpass]");
```

Error checking is done on every stage; the code snippet below is a condition that gets called in case the connection to the database fails. This is very crucial as any tasks and functions that are to follow will not work, and the web server is forced to “exit” before parsing any other lines.

```
    if (!$dbconn) { // error if the connection
fails
        echo "An error occured connecting to
Database. \n";
        exit;
    }
```

One of the many other functions within the playlist class below, is extracts of the create function. Its main purpose is to make empty playlists which will later be filled with media by using the addtrack and removetrack function’s. By default, Restler uses the GET HTTP method. If this has to be changed to POST, the changes have to be done manually by inserting the snippet below.

```
/* Manual Routing of some functions
* @param string $playlist_name
* @param string $playlist_description
 * @url POST create
*/
```

Just like the previous function mentioned, the create function has two variables (namely playlist_name and playlist_description), and the function continues only if both these variables are set.

```
Functioncreate
($playlist_name=null,$playlist_description=null){
if(isset($playlist_name,$playlist_description)){
```

The SQL connection and the owner ID that is associated to the APIs is then called in, as well as the current time which is obtained in UTC format which the database uses. The owner id is by default set to 1, which is the Admin account. This is usually changed to a new user, which is purely dedicated to the API usage. This new account is crucial as this would assist in accountability.

```
global$dbconn;
global$owner_id;
$mtime=gmdate('Y-m-d H:i:s');//Getting time in UTC format
// the owner_id is set to 1 , which is the admin user, from
index.php
```

All these values are then inserted into the cc_playlist table, and the relevant checks are carried out.

```
$result=pg_query($dbconn,"INSERT INTO cc_playlist (name,mtime,utime,creator_id,description) VALUES ('$playlist_name','$mtime','$mtime','$owner_id','$playlist_description') ");

if(!$result){// if the query fails
throwneRestException(500,'Query has failed');
}
throwneRestException(201);

}

else{
throwneRestException(400,'required parameters not filled');
}
```

Since the APIs are web based, the APIs are run using a standard web browser that can test the functionalities provided by the newly build APIs. Such an example of sending data is shown below that uses multiple and single variables.

Example:
http://*localhost/apiv1/playlist/search?playlist_name=test&playlist_description=new music*
If multiple, the first variable starts with a "?" then the variable name, followed by the assigned value. However, the variables that follow start with the "&" sign.

Another example, using a different function called "details", which is in the same class as "playlist" but caters for only one variable, has a slightly different structure as shown in the Figure 8.

Figure 8: GET request for class playlist, and details function and playlist_id

The sample response acquired by the client is shown in Figure9.


```
{
  "Playlist_length": "00:13:31.412",
  "Track_position": "0",
  "ID": "14",
  "Playlist_name": "slim mix",
  "Track_title": "Dreams"
},
{
  "Playlist_length": "00:13:31.412",
  "Track_position": "1",
  "ID": "16",
  "Playlist_name": "slim mix",
  "Track_title": "Glasgow"
},
{
  "Playlist_length": "00:13:31.412",
  "Track_position": "2",
  "ID": "1",
  "Playlist_name": "slim mix",
  "Track_title": "I Can Only Imagine - Feat. Chris Brown & Lil Wayne "
}
```


Figure 9: JSON response of the playlist/details GET request

This is how all the other classes and APIs work, but having their own functionality.

9. Testing

The APIs created are web based and therefore, testing was done using a web browser that can test the functionalities provided by the newly build APIs. Testing is an important phase in the development of any product. Below are different tests carried out to find the durability of the developed code and possibly rectify where it has it shortfalls. The input value for playlist details is the playlist ID, (which is an integer). So if a string is placed as shown in Figure, an exception is thrown as shown in Figure which shows an error "400". By referencing it to the design, it means that a bad syntax has been used.

Figure 10: Variable testing


```
{
  "error": {
    "code": 400,
    "message": "Bad Request: not valid input"
  }
}
```


Figure 11: Error, Bad syntax

It is rare to find an internal error, but it has to be catered for. Table 3 has summarized values of the results that were obtained under testing

10. Discussion and future work

The development of APIs had started on Airtime version 2.2 in October 2012, and most functions were then built. Earlier in the year (2013), version 2.3 of the Airtime application was released. When the new update was installed, it was realized that some functions failed to work completely, and as such the API development had to restarte in relation to a few functions that were updated in the new version. Thus, the Application updates have an effect on the running of the APIs. The APIs have been developed using the current Database structure and Application of Airtime 2.3. Updates to the application may require minor changes to the php code to work with the new updates.

The Airtime application used in this research is an Open Source application. Most open source applications are developed by people who do it for fun and in their own time and there is no guarantee to find documentation on the application and its coding. Thus, access to explanation of the codes can become a challenge with some open source applications. The main problem encountered in this research was the lack of documentation associated with the technical design of the application, as most developers add functionality without providing the technical documentation of the design procedure, since it is not a key requirement.

Reverse Engineering of the application was carried to understand the code and out to find out what their functions are, where and how they are used, the security, and the validations associated with it. Developing APIs for open source applications without documentation for their codes and database can be challenging.

Table 2: Inputs, methods and outputs associated to an Api function

API name	Description	Parameters	Parameters Type	HTTP Method	Output results contents
Track/ download	this method allows the developer to download a media file from the server , The below method are completely based on knowing the track id, and the files if imported are stored as /SRV/AIRTIME/<owner_id>/<artist>/<album-name>/<track-number>-<track-name>	**Track_id.**	**Required**	**GET**	• meda file.
Track/ Upload	This feature assists the developers to upload tracks to the server. File has to be in right format (wav, flac, ogg, mp3,acc). Maximum upload size by default is 500 MB.	**File**	**Required**	**POST**	• System response
Playlist/ create	This method creates a new playlist	**Playlist_name Playlist_Description**	**Required**	**POST**	• OK. (System response)
Playlist/ edit	This is to assist the developer to modify a specify playlist	**Playlist_id**	**Required**	**POST**	• OK. (System response)
		Playlist_name Playlist_Description	**Optional**		
Playlist/ addtrack	this should help the developer add a specific track into a playlist	**playlist_id track_id position**	**Required**	**POST**	• OK. (System response)
Playlist/ removetrack	this should help the developer remove a specific track into a playlist	**playlist_id, file_id**	**Required**	**POST**	• OK. (System response)

Table 3: Summarized outcome of testing

Class/ function	Tested Variable /Type	Input type	Input value	Expected Response	Response acquired
File/ download	track_id/integer	Integer	20	Pass	pass
	track_id/integer	String	twenty	Fail	fail
File/ upload	url /String	String	http://localhost/test.mp3	Pass	pass
Playlist/ create	playlist_name	String	Mixtape	Pass	Pass
	playlist_description /String	String	nice music	Pass	Pass
Playlist/ edit	Playlist_id /integer	Integer	5	Pass	pass
	Playlist_id /integer	String	New	Fail	Fail
	playlist_name /String	String	Mixtaped	Pass	Pass
	playlist_description /String	String	nice music collection	Pass	Pass
Playlist/ addtrack	Playlist_id /integer	Integer	5	Pass	pass
	Playlist_id /integer	String	New	Fail	Fail
	track_id / Integer	Integer	5	Pass	pass
	track_id / Integer	String	New	Fail	Fail
	Position / Integer	Integer	5	Pass	pass
	Position / Integer	String	New	Fail	Fail
Playlist/ removetrack	playlist_id / Integer	Integer	5	Pass	pass
	playlist_id / string	String	New	Fail	Fail
	track_id /integer	Integer	5	Pass	pass
	track_id /integer	String	New	Fail	Fail

The APIs functionalities developed for this Radio Management System called Airtime are not exhaustive, and many more functionalities can further be developed. Future work can be further carried out by other developers to diversify the functionalities provided by the API, and to add other functions that were not catered for. APIs can also be developed for Web Radio Management Systems such as Campcaster.

Conclusion

The APIs created are completely open source and can be used by developers to add Airtime functionalities into their own Applications that are being developed. The developed APIs will give external developers the opportunity to access the application without having to physically login, but having valid platform credentials.

The many APIs coded have various functions, where an API to upload and download media files were also developed called "file/download" and "file/upload". An API was also developed to create new playlists, add and remove media track with automatic positioning of the track by using "playlist/create", "playlslit/addtrack", "playlist/removetrack".

An implementation area for use of the developed APIs is in creating a solution called “Breaking News", which is to be deployed with the use of the Emerginov Platform. With this system, customers are given a deadline to call in before an allocated time, and then report news on a specific mobile number. This could be either by voice call or as SMS text, which will then be converted on the Emerginov platform using the Google Text-to-Voice API. These audio files will be stored in a media folder, and the APIs written can then be called to automatically load all these mp3 files into a radio playlist and integrate into a show. Once in, the radio can automatically broadcast.

Acknowledgments

The Authors will like to thank the people at Orange Labs namely Arnaud Morin and Morgan Richomme for the guidance they have shown with the API coding and the use of the Emerginov Platform. The Authors will also like to thank the community namely from “Stackoverflow” and “sourcefabric” in assisting when issues with the code were encountered.

References

Software & Information Industry Association. (2001, February). Software as a Service: Strategic Backgrounder. Retrieved February 28, 2013, from Software & Information Industry Association: http://www.siia.net/estore/pubs/SSB-01.pdf

How to Leverage an API for Conferencing. (n.d.). Retrieved April 26, 2013, from How stuff works: http://money.howstuffworks.com/business-communications/ how-to-leverage-an-api-for conferencing3.htm

Online webpage: Open vs. Closed Source Software, retrieved on 10 June 2013, http:// scienceinafrica. com/ old/ index.php?q=2004/january/ software. htm

Online webpage: Open Source versus Closed Source, retrieved on 10 June 2013, http://mongers.org/open-vs-closed

Online webpage, retrieved on 10 June 2013, http:// en.wikipedia.org/ wiki/ Campcaster

Online webpage, retrieved on 10 June 2013, http:// en. wikipedia.org/wiki/ Airtime

Online webpage, retrieved on 10 June 2013, http:// lwn.net/Articles/ 481607/

Online webpage, retrieved on 10 June 2013, http:// www. shoutcast.com/

Online webpage, retrieved on 10 June 2013, http:// blog.programmableweb.com/2012/01/26/50000-radio-stations-in-one-api/

Online webpage, retrieved on 10 June 2013, http://ampache.org/

Online webpage, retrieved on 10 June 2013, http://wiki.radioreference.com/index.php/API

Online webpage, retrieved on 10 June 2013, http://developer.rdio.com/docs/Web_Playback_API

Online webpage, retrieved on 10 June 2013, http://www.last.fm/api/radio

R. F. (1999). Hypertext Transfer Protocol -- HTTP/1.1 . Retrieved April 12, 2013, from world Wide web consortium:http://www.w3.org/Protocols/rfc2616/rfc2616.html

Extensible Markup Language (XML) 1.0 (Fifth Edition). (2008, November 26). Retrieved April 4, 2013, from world wide web consortium: http://www.w3.org/ TR/REC-xml/

Nurzhan Nurseitov, M. P. Comparison of JSON and XML Data Interchange Formats: A Case Study.

drrwebber. (2013, April 26). Analysis of JSON use cases compared to XML. Retrieved May 12, 2013, from Oracle blogs: https:// blogs. oracle.com/ xmlorb/entry/analysis_of_json_use_ cases

W3G Recommendations (2007, April 27). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Retrieved January 23, 2013, from World Wide Web Consortium: http://www.w3.org/TR/soap12-part1/#intro

Kyrnin, J. (n.d.). web design /HTML . Retrieved December 29, 2012, from About.com: http://webdesign.about.com/od/soap/a/what-is-xml-soap.ht

SOAP Introduction. (n.d.). Retrieved march 12, 2013, from w3schools.com: http:// www.w3schools.com/ soap/soap_intro.asp

Skonnard, A. (2003, march). Understanding SOAP. Retrieved January 18, 2013, from MSDN: http://msdn.microsoft.com/en-us/library/ms995800.asp

The Advantages and Disadvantages of Using SOAP Messages. (n.d.). Retrieved Feburary 23, 2013, from xyzws: http://www.xyzws.com/ scdjws/studyguide/ soap_chapter8. Html

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Retrieved October 21, 2012, from http://www.ics.uci.edu/~ fielding/pubs/dissertation/rest_arch_style.htm

Olson, M. (2002, July 03). The Python Web services developer: Messaging technologies compared. Retrieved march 15, 2013, from IBM Developer Works: http://www.ibm.com/developerworks/library/ws-pyth9/

Elkstein, D. M. (n.d.). Rest Server Responses. Retrieved January 12, 2012, from Learn REST: A Tutorial: http://rest.elkstein.org/2008/02/rest-server-responses.html

Oracle. (n.d.). Web Services: REST vs. SOAP. Retrieved Feburary 18, 2013, from Milan's blog by Oracle: https://blogs.oracle.com/milan/entry/web_services_rest_vs_soap

SOAP vs. REST. (2010, January 15). Retrieved March 12, 2013,

from spf13: http://spf13.com/post/soap-vs-res

Huppo. (2011, june). Compare and contrast REST and SOAP web services. Retrieved march 14, 2013, from stackoverflow:http://stackoverflow.com/questions/10975863/compare-and-contrast-rest-and-soap-web-services

Lane, K. (2011, Septmeber 23). Short List of RESTful API Frameworks for PHP. Retrieved March 14, 2013, from Programmableweb:http://blog.programmableweb.com/2011/09/23/short-list-of-restful-api-frameworks-for-php/

Orange. (2012, December 04). Emerginov, an innovative solution for mobile services development in Africa. Retrieved February 19, 2013, from Orange news: http://www.orange.com/en/news/2012/novembre/Emerginov-an-innovative-solution-for-mobile-services-development-in-Africa

Orange. (n.d.). About. Retrieved February 19, 2013, from Emerginov: http://www.emerginov.org/about.php

Jansen, G. (2011). The Job of the API Designer. Retrieved January 19, 2013, from Restful API design: https://restful-api design. readthedocs. org/en/latest/ scope. Html

Mathias A., Sheikh S.M., Jeffrey A.M and Masupe S. (July 2013) APIs to extract information from an existing web radio application. *International Journal of Electronics and Communication*, Vol. 2, Issue 3, pp. 117-132.