Investigating Integration of Synote with Lecture Recording Systems

September 2016

MSc Web Technology
Abstract

Web accessibility has been developed extensively recently. Many technologies have been assisting different type of disabilities making users life on World Wide Web much easier. These developments have removed the barriers that prevent people with disabilities from accessing and interacting with websites. The improvement in this field has covered numerous types of disabilities that users encounter on the web including visual, auditory, physical, speech, cognitive and neurological. Further contribution has been added to this field, when Synote open source annotation system was introduced which enhance users learning experience, in particular students with disabilities and those who has learning difficulties. The system makes lecture recordings not only available at any time, it provides accessible contents consisting of audio, video, transcripts and presentation slides and enable users to search through recordings content at any point. The aim of the project is to expand the system allowing it to receive recordings from other recording systems. This enhancement will make the system more accessible and usable by numerous educational institutions. the project examined different types of recording systems and based on several criteria Opencast was chosen. Also, a web based application was created to handle recordings data and an RESTful API was developed for Synote to interact with the data. Some future work also has been suggested to integrate Synote with Moodle learning management system.
Table of Contents

Abstract ........................................................................................................................................... i
Acknowledgment .............................................................................................................................. ii
Table of Contents ............................................................................................................................ iii
List of Figures ................................................................................................................................... vi
List of Tables .................................................................................................................................... vii
Chapter 1 Introduction ..................................................................................................................... 1
  1.1 Project Aim ............................................................................................................................... 1
  1.2 Outline ..................................................................................................................................... 3
Chapter 2 Background Research ....................................................................................................... 4
  2.1 Recording Systems ..................................................................................................................... 4
    2.1.1 Introduction ....................................................................................................................... 4
    2.1.2 Echo360 ............................................................................................................................ 4
    2.1.2.1 Echo360 Documentation ............................................................................................. 4
    2.1.3 Mediasite .......................................................................................................................... 5
    2.1.3.1 Mediasite API ............................................................................................................... 5
    2.1.4 Opencast ............................................................................................................................ 8
    2.1.4.1 Opencast Documentation ............................................................................................. 8
    2.1.4.2 Representational State Transfer .................................................................................. 8
    2.1.4.3 JavaScript Simple Object Notation .............................................................................. 9
  2.2 Synote Frameworks ................................................................................................................... 9
    2.2.2 NodeJS .............................................................................................................................. 9
    2.2.3 Sails ................................................................................................................................... 9
    2.2.4 Waterlock .......................................................................................................................... 10
  2.3 Moodle ..................................................................................................................................... 11
    2.3.1 Moodle API ..................................................................................................................... 11
Chapter 3 Requirements and System Overview ................................................................................ 12
  3.1 System Requirements ............................................................................................................... 12
    3.1.1 Introduction ..................................................................................................................... 12
    3.1.2 Hardware Configuration .................................................................................................... 12
    3.1.3 Opencast Installation ........................................................................................................ 12
3.1.4 Moodle Installation

Chapter 4 Systems Analysis

4.1 Choice of a recording system

4.1.1 Introduction

4.1.2 OpenCast functionalities

4.1.3 Testing OpenCast API

4.1.3.1 Postman tool

4.1.3.2 API Authentication mechanism

4.1.3.3 Data required to perform integration

Chapter 5 Solution Design

5.1 Design phase

5.1.1 Introduction

5.1.2 UML

5.1.3 Jetty server

5.1.4 Apache2

5.1.5 PHP scripts

5.1.6 PHP CodeIgniter Framework

5.1.7 MariaDB database

Chapter 6 Solution Development

6.1 Development phase

6.1.1 Introduction

6.1.2 Technologies summary

6.2 Scenario discussion and implementation

6.2.1 Flowchart

6.2.2 Data retrieval

6.2.3 Inserting data into database

6.2.4 Automation of the process

6.2.5 Recordings API

6.2.5.1 Authentication

6.2.5.2 GET all records

6.2.5.3 GET records by ID

6.2.5.4 GET records based on date
List of Figures

Figure 2.1: Authentication API of EDAS ................................................................. 6
Figure 2.2: Presentation API of EDAS ................................................................. 6
Figure 2.3: Folder query API ................................................................................ 7
Figure 2.4: EDAS search API ............................................................................. 7
Figure 3.1: database configuration settings ......................................................... 14
Figure 4.1: Creating a recording /event ............................................................ 16
Figure 4.2: redirection to Opencast login page .................................................. 18
Figure 4.3: different response generated using cUrl ....................................... 18
Figure 4.4: JSON content retrieved using cUrl ............................................... 19
Figure 4.5: Structure of episode JSON data ...................................................... 20
Figure 5.1: UML component diagram of the system ........................................ 21
Figure 6.1: Data flow from Opencast to the database ....................................... 26
Figure 6.2: cUrl code for requesting JSON content ....................................... 27
Figure 6.3: This part responsible for validation agent any errors .................... 27
Figure 6.4: A sample of JSON content from episode API ............................... 28
Figure 6.5: A sample of JSON content retrieved from episode API ............... 29
Figure 6.6: A sample of inserted records .......................................................... 30
Figure 6.7: Preventing duplicates indicated with “Records exists” .................. 30
Figure 6.8: A screen shot of the data being inserted automatically ................. 31
Figure 6.9: A new record being added through checking every 10 minutes ...... 32
Figure 6.10: Adding authentication header ..................................................... 33
Figure 6.11: A sample code of GET all recordings .......................................... 34
Figure 6.12: A sample code of GET recordings based on ID ............................ 34
Figure 6.13: recordings response in JSON format ......................................... 35
Figure 6.14: recording response based on ID ................................................... 36
Figure 6.15: A sample of data restored by title argument ............................... 36
Figure 6.16: Retrieving JSON content using Date argument ......................... 37
Figure 6.17: A more specific API using Date and Time for content retrieval ... 37
Figure 7.1: Error in publishing only audio ......................................................... 39
Figure 7.2: Publishing an event with Audio/Video succeeded ......................... 40
Figure 7.3: Empty field called from Opencast ............................................... 40
Figure 7.4: Empty cells and zero values issues .............................................. 41
Figure 7.5: Database issues being fixed ............................................................. 41
Figure 7.6: Record inserted using CRON job .................................................... 42
Figure 7.7: Unauthorized response when credentials or IP are invalid ............ 42
Figure 7.8: Meaningful error message ............................................................. 43
Figure 7.9: handling of special characters ...................................................... 43
Figure 7.10: Response of querying data between different times .................... 44
Figure A.1: Gantt Chart of the project ............................................................... 51
Figure B.1: Configured htaccess for making changes in the server ............... 52
Figure B.2: Whitelisting configuration ............................................................ 52
Figure B.3: Configuring Basic Auth users ....................................................... 53
Figure B.4: Configuring database connection in CodeIgniter ....................... 53
Figure B.5: Auto-load database in CodeIgniter ............................................ 54
Figure C.1: Automating process using CRONTAB ........................................ 55
List of Tables

Table 6.1: Summary of the technologies that have been used................................................................. 24
Table 6.2: Scenarios summary for implementation ..................................................................................... 25
Table 6.3: Directories of the required scripts .................................................................................................. 33
Chapter 1 Introduction

Web accessibility is a concept that means ability of users with disabilities to access a website. This accessibility includes their ability of perceiving, understanding, navigating and interacting with the web, besides being able to contribute contents to the World Wide Web [1]. It might be thought that web accessibility is benefiting only those people who are challenged by various types of disabilities, however, web accessibility is advantageous to elderly, international students, and even those without disabilities, as well. Web accessibility is critical due to the fact that the web encompasses many aspects of people life such as education, employment, government and commerce [1]. Another key fact is offering equality among a community by providing equal access and equal opportunity of accessing web content for users with disabilities.

Different technologies were developed to assist students with learning difficulties. Before investigating those technologies, it is important to understand and note reasons that might affect students learning. Most overseas students are non-native English speaker; this make them face some challenges when they engage in higher education. Those challenges can be in writing, speaking and listening to native speaker lectures from different areas with different accents. A research done by [2] have reported some of the issues that international students face. The issues are difficulties in written works particularly in grammar, accuracy and academic writing skills, difficulty in listening to lectures and some issues with communicating with supervisors. These issues that overseas students face during their higher education studies give indication that their performance will decrease and it might not encourage them to continue studying aboard.

1.1 Project Aim

The idea of the pervious project was to provide an open source web based annotation system that was introduced in a course module taught by Professor. Mike Wald. Pervious students were assigned a task to integrate the annotation system that is called Synote with other systems. These systems are Panopto a recording system used for recording lectures and a learning management system were users can access recordings content. Their project objective at that time was to automate transferring of contents from the recording system to Synote, then posting links to users by email address allowing them to access course content. At this point the aim is to expand Synote making it further accessible by integrating it with other recording and LMS systems. This is critical due to the fact that there are other higher educational institutions that may use different type of systems.

Due to the reasons mentioned earlier, this project is going to investigate how to integrate Synote with several recording systems and learning management systems. Before choosing preferable recording system RS and LMS, it is essential to consider its compatibility and applicability of development, also the utilization rate within universities. These points are discussed in the following chapter and based on the background research it was decided to
use Opencast system, which has been utilized for retrieving recordings content. Also, Moodle LMS was installed and configured for integration to be used for posting lecture recordings. However, because the project duration is limited to three months, future work suggestion is given to implement Moodle integration with Synote.
1.2 Outline

Chapter 2 begins by describing several different types of recordings systems, looking at its features, compatibility for development using its API and availability of documentation and technical support. Also, the chapter examined Synote functions and frameworks used to build it. Then, the chapter highlighted reasons Moodle LMS being important part for integration.

Chapter 3 examined and discussed configuration requirements for this project. This included hardware-level and software-level requirements. Then, the chapter demonstrated installation steps of Moodle LMS and Opencast.

Chapter 4 further gives detailed analysis of the recording system that is used for this project “Opencast”. It investigated the use of an open source software, features of the API provided, examined tools for interacting with the API. Moreover, it spotted some of the issues encountered when sending HTTP request and ways of overcoming those issues.

Chapter 5 discusses the design phase of the project. Using Unified Modelling Language, it identified system components and provided details on how each component were utilized for the solution development.

Chapter 6 demonstrates the development phase which includes an explanation of the technologies used such as software, operating system, libraries and framework. It also discusses scenario requirements that should be implemented. Moreover, the flowchart provided in the chapter explains data flow of recordings. After that, an explanation of the code that was used for the development of the web application and the API is given.

Chapter 7 demonstrates the project test implementation to find errors and bugs and, enhance the solution. This chapter evaluates each component by thoroughly testing it starting with the systems and finishing with API calls. It also explains errors that were found and the solution that were implemented.

Chapter 8 explains the processes that were considered in planning and managing the project. Moreover, it describes the challenges encountered during project implementation. Finally, chapter 9 concludes this dissertation with a summary, final thoughts and future suggested improvements.
Chapter 2 Background Research

2.1 Recording Systems

2.1.1 Introduction

The earlier project of Synote integration was done by integrating Panopto, Blackboard and Synote. However, different universities have different technology infrastructure, thus an intensive research will be carried on to look on the most suitable recording system that can be used to integrate with Synote. Several points need to be investigated on each recording software to determine qualification of a recording system; first its features and functionalities, capabilities of integration with other systems, and availability of technical support documentation.

2.1.2 Echo360

The system has many features that are usable for supporting capturing and distance learning\(^1\). This solution can be categorized into hardware and software. The former part consists of two types; Echo360 pro which support a wide input types that can be analogue or digital. The latter one is Echo360 POD which is much simpler, it can be used through plug-and-play, no installation is required and cost less than the first solution. Software solutions is also provided that consists of three types. Classroom capture can be used on any PC for scheduled or on demand lecture capturing. Moreover, personal capture is the on the go software solution, it can be installed in a laptop to provide a platform for capturing and editing learning content. Finally, a mobile solution is offered for recording and sharing contents, which is available for both iOS and Android devices. For the integration purpose, technical documentation is required and will be discussed on the proceeding sections.

2.1.2.1 Echo360 Documentation

Manuals and API documentation for Echo360 is only available to its customers. A customer account is also needed to access the API documentation. Through searching on the web, an old documentation was found, however, it might not be accurate as the documentation offered by the official website. Part of the documentation explains the steps for publishing materials on Moodle learning management system. The following section will discuss the capabilities and features available for sharing Echo360 recordings.

There are two publishing methods supported on Moodle:

- Individual link publishing:

With this method user can only see Echo contents and additional materials through Moodle. This has been provided since 2.3 version of Echo system.

\(^1\) [http://echo360.com/](http://echo360.com/)
• EchoCenter Publishing:
This method allows users to see EchoCenter page. It has been supported since version 4.0 onward. The documentation recommends the use of EchoCenter Publishing as best practice for several reasons. With this feature of publishing, users would be able to see a course list page rather than individual Echoes as described. The EchoCenter publishing method is provided as a Moodle plugin that can be downloaded from Echo360 customer support portal. The rest of the steps give technical information and configuration requirements on how to install the plugin. Other recordings systems will be discussed in the proceeding sections.

2.1.3 Mediasite
Another recording system named Mediasite offered by Sonic foundry. The solution is also categorized as hardware and software solution. The first hardware solution is RL120, this recorder is reliable and can be used to capture a video with contents, dual-video, or content with audio. Moreover, with this device, capturing can be accomplished automatically through setting schedules. The second pair of devices are RL700 and RL800, those recorders offer an integrated capturing and live streaming which is suitable for lecture halls and training facilities. Further, automation of capturing contents is available by pre scheduled start and stop recording times and schedules can be imported from existing timetable or a course management system. On the next section Mediasite documentation API will be discussed to look on the possibility of integration with Synote.

2.1.3.1 Mediasite API
The Mediasite\(^2\) official website offers basic API for interacting with External Data Access Service (EDAS). It gives the ability to integrate information from Mediasite Ex server with an application. The programming language that is used for this API is C#. Sonicfoundry Mediasite API consist of four parts:

- Authentication
Authentication against the EDAS API requires sending a login request as shown below:

\(^2\)https://support.sonicfoundry.com/Training/EDAS
Figure 2.1: Authentication API of EDAS

- Presentation

This is used to retrieve data. The code below explains how to find a presentation by name:

```csharp
public List<PresentationDetails> GetPresentationsByName(string authTicket, string presentationName) {
    List<PresentationDetails> presentations = new List<PresentationDetails>();

    var request = new QueryPresentationsByCriteriaRequest {
        Ticket = authTicket
    };

    var criteria = new PresentationQueryCriteria() {
        TitleRegEx = string.Format("{0}.*", presentationName),
        PermissionMask = ResourcePermissionMask.Read
    };

    request.QueryCriteria = criteria;

    request.Options = new QueryOptions() {
        BatchSize = 10,
        StartIndex = 0
    };

    var response = base.Client.QueryPresentationsByCriteria(request);

    if (response.Presentations != null && response.Presentations.Length > 0) {
        presentations = response.Presentations.ToList();
    }

    return presentations;
}
```

Figure 2.2: Presentation API of EDAS
• Folders

The API also provide a feature to query folders similar to the one used for querying presentations. The code below explains how to query a folder.

```csharp
public List<FolderDetails> GetFoldersByFolderId(string authTicket, string folderId) {
    List<FolderDetails> folders = null;
    var request = new QuerySubFolderDetailsRequest {
        Ticket = authTicket
    }
    request.IncludeAllSubFolders = true;
    request.PermissionMask = ResourcePermissionMask.Read;
    request.ParentFolderIdList = new string[] { folderId };
    var response = base.Client.QuerySubFolderDetails(request);
    if (response.Folders != null && response.Folders.Length > 0) {
        folders = response.Folders.ToList();
    }
    return folders;
}
```

Figure 2.3: Folder query API

• Searching

The last part of the API provides a feature for searching. By using this API, a user would be able to find presentations, templates, folders etc. (see figure 2.4).

```csharp
public List<SearchResponseDetails> FindPresentationsByDescription(string authTicket, string searchText) {
    List<SearchResponseDetails> searchResults = null;
    var request = new SearchRequest {
        Ticket = authTicket
    }
    request.SearchText = searchText;
    request.Fields = new SupportedSearchField[] { SupportedSearchField.Description }; // Remove template field
    request.Types = new SupportedSearchType[] { SupportedSearchType.Presentation }; // Replace with Presentation
    request.Options = new QueryOptions {
        BatchSize = 10,
        StartIndex = 0
    };
    var response = base.Client.Search(request);
    if (response.DetailList != null && response.DetailList.Length > 0) {
        searchResults = response.DetailList.ToList();
    }
    return searchResults;
}
```

Figure 2.4: EDAS search API
2.1.4 Opencast

Opencast is a video capture software that is flexible and customizable, it is designed for managing contents (video or audio) for educational purposes. It is an open source software built by a community of developers in partnership with global leading organization and universities. Opencast offers several features that are beneficial for users who would be interested for capturing lectures:

- Automation is possible for scheduling single or multi-stream recordings by using the administrative tools. Also, users can upload files manually, manage metadata and process functions.
- Availability of video editor that enable users to edit or trim videos before being published
- Opencast is flexible enough to be integrated with various open source or proprietary lecture recording devices.
- It has services that can prepare and package media contents with trailers, bumpers, cover slides and watermarks.

2.1.4.1 Opencast Documentation

The recording software Opencast Matterhorn offers an application API for technical integration with other applications like learning management system\(^3\). It is used to allow applications to provide access and management of resources revealed through the API. The design and implementation of the API is intended to offer support for large number of clients. Furthermore, to ensure data protection of the managed data, and to promote various views of the managed data security has been given a considerable attention. As indicated the API can be implemented as an abstraction layer to multiple internal APIs, which can be used for manipulating resources such as series, events (event is referred to recordings and series is collection of events) or users.

2.1.4.2 Representational State Transfer

Representational State Transfer (REST) is an architecture style used for Internet resources manipulation over Hypertext Transfer Protocol (HTTP). This is the structure that Opencast API is offering. The advantage of using REST is, simplification of interaction with resources by using unified recourse identifier (URI) and standard HTTP methods. Resources in REST structure is identified by URI which enable clients retrieving presentations of a resource through sending an HTTP GET request to a server. Then, a representation of a resource is delivered to clients as an HTTP response in the format agreed by both client and server (HTML, XML, JSON, etc.). In Opencast, resources are represented in either XML or JSON. Most RESTful services require type of authentication (discussed in section 4.1.3), if authenticated, clients are enabled to create, modify or delete a resource by sending POST, PUT or DELETE requests to the server [6].

\(^3\) http://www.opencast.org/
2.1.4.3 JavaScript Simple Object Notation

JavaScript Simple Object Notation known as JSON is a famous data interchange format utilised for transferring data in the World Wide Web among web clients and servers. Unlike XML, JSON is human-readable and a lightweight structure, which can be parsed without affecting processing performance due to the fact that parsing is implemented on client-side only [7]. Thus, JSON is convenient for transferring data from Opencast recording management system to Synote. Furthermore, JSON is a text-based format language-independent making it applicable for encoding and decoding with various programming languages. The structure consists of name value pairs; key on left side and value on the right side.

2.2 Synote Frameworks

Synote version that is required to be integrated is developed using NodeJS, and using Sails.js framework. For authentication purposes, Waterlock authentication framework was implemented. Front-end interface of Synote was developed using AngularJS a single page application.

2.2.2 NodeJS

NodeJS⁴ is a server side scripting language that is built based on JavaScript. It is considered lightweight and efficient due to the use of event-driven, non-blocking model. The software is used to serve HTTP request in an event-driven way. Significantly, development of a considerable scale project requires using a framework, for this purpose SailsJS was used for Synote development.

2.2.3 Sails

Sails is useful for building a customized enterprise-grade NodeJS applications. It is an MVC framework for NodeJS. There are several key features offered by this framework:

- Advanced object relational mapper
- Automation of creating RESTful APIs and built-in socket support
- Authorization system using reusable classes, which is called policies.
- Extensible through creating Sails hooks modules

Sails is being chosen for the fact that one of the Synote project objective is to support various clients such as mobile devices and tablets. With the RESTful APIs and sockets generated by SailsJS, supporting different front-end clients is possible. Currently, AngularJS single page application front-end is implemented on Synote. As a result, the solution that need to be developed must confirm to this front-end framework.

⁴ https://nodejs.org/en/
2.2.4 Waterlock

Waterlock\(^5\) is an authentication framework used by Synote that is designed on a principle of reusable authentication modular. Various authentication modules were created for Waterlock which includes modules for Facebook, Google and Twitter. Furthermore, unlike traditional application this authentication framework uses two modules; an Auth module which is used for login (contains username and password), and User module that contains general information about users. Each user is associated with Auth model; this allows’ users to concurrently utilize several authentication methods.

\(^5\) http://waterlock.ninja/
2.3 Moodle

Moodle is one of the most famous course management system used by many organization and educational institutions now days, which should be considered for integration with Synote. From the statistics available at [5] it can be seen reasons of Moodle importance. The statistics shows that there are about 70,000 Moodle registered sites in 232 countries around the globe. Moreover, number of courses available on these sites have exceeded 10 million with incredibly huge number of users reaching more than 89 million. Furthermore, usage of Moodle is increasingly high; this is obvious from the number of enrolments 270 million, forum posts 183 million, resources 90 million, and 468 million quiz questions.

In terms of top countries that utilizes Moodle sites [5] the United States is on the top with approximately 10,000 registered sites, Spain have 7,000 sites, Brazil got above 4,000 sites, then comes the United Kingdom with more than 3,000 registered Moodle websites, while the rest of countries have registered sites ranged from 1,000 to 3,500. This statistic significantly gives great indication that Moodle could be a valuable choice for integrating with Synote, which means number of e-learning contents will be available for a wider range of users which in turns gives higher opportunity for international and students with disability of accessing accessible online contents.

Another reason that make Moodle LMS a preferable choice for integration with Opencast recording software and Synote the annotation web application is the cost. Sustainability and cost efficiency are essentially required for a learning management system (LMS) by most higher educational institutions. Fortunately, Moodle LMS solution is excellent as an open source web based application, cost efficient and availability of community support, which might not be the case with proprietary LMS solutions [6]. Besides these, implementation requirements do not require sophisticated hardware and software solutions and documentation of the API is available, which will facilitate integration process. Next section will discuss API documentation of Moodle.

2.3.1 Moodle API

Moodle\(^6\) offers an API documentation that can be used for the integration process. To begin with, access API provides functions for determining users' authority and it allows expanding of new capabilities. The system uses role based access control model by which representation of entities such as users, courses, modules and blocks are defined like hierarchy called context tree. Also, user capabilities are defined using roles that represents user's ability of doing something. Another API that is important to look at is data manipulation. It allows developers to perform reading and writing to databases in a robust secure manner. The functions that are presents in the official documentation of Moodle is only for use with version 2.0 upwards. Thus, database manipulation is guaranteed to function with various RDBMSes.

\(^6\) https://docs.moodle.org/dev/Core-APIs#Data_manipulation_API_.28dml.29
Chapter 3 Requirements and System Overview

3.1 System Requirements

3.1.1 Introduction

This chapter will discuss configuration requirements that will be used for developing the solution required for integration. For this project systems that will be installed and configured are; Opencast Matterhorn the recording software, and Moodle the learning management environment. The software will be installed locally on a personal computer, while Synote is installed on a testing server at University of Southampton.

3.1.2 Hardware Configuration

For this purpose, several hardware and software were essentially important to accomplish this stage. Firstly, due to time limitation it has been decided to install most of the software on a PC, which has a 16 GB of RAM memory and 1 TB of storage capacity, rather than requesting a server from technical support that might take long time. On the software side, working on virtual environment was considered, to avoid any crashes that might cause failure on the hardware, as a result, VMware Workstation 12 pro was used. After that, a decision was made to use Linux environment which is Ubuntu 16.04 as a hosting operating system for installing Opencast and Moodle. Following sections will demonstrate installation and configuration required.

3.1.3 Opencast Installation

The installation of Opencast was complex and required long time to make the system functioning accurately. For the purpose of this project Opencast version 2.2.0 was installed on Ubuntu16.04. Several dependencies were required before it could be possible to install Opencast. The list below lists those mandatory packages:

- openjdk-8-jdk
- ffmpeg >= 3.0.1
- maven >= 3.1
- unzip
- g++
- tar
- bzip2

Although several of these packages were installed straightforward, some were installed manually due to system requirements of Opencast, which requires specific version. Once the installation of the packages is finished, Opencast can be built and installed using the following
command:
- git clone https://bitbucket.org/opencast-community/matterhorn.git
- cd matterhorn
- git tag  <- List all available versions
- git checkout 2.2.0

3.1.4 Moodle Installation
Installation of Moodle was carried out on another virtual environment that is Linux Ubuntu16.04\(^7\), the LMS requires several components before proceeding with installation that are Apache, and SQL server (MySQL database was installed), and PHP, known as LAMP platform. Hardware requirements were met with higher specifications for better performance.

Once all the components are set and configured accurately, downloading the software can be done from Linux terminal using the following command:

```
$ git clone --depth=1 -b MOODLE_31_STABLE git://git.moodle.org/moodle.git
```

Then, installation is better performed on a web browser, but it is essential to have a working server (Apache is this case). Moodle folder must be placed at this directory var/www/html/moodle. After, accessing files will be through entering localhost/moodle. However, folder and files must have administrative privileges to give the software the ability to read, write and add files.

The next step is to create a database for the installation, which was accomplished using Mysql. The database was created by entering the following command using Ubuntu terminal:

```
CREATE DATABASE moodle DEFAULT CHARACTER SET UTF8 COLLATE utf8_unicode_ci;
```

After that, a user is created with required roles for accessing and making modification:

```
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,CREATE TEMPORARY TABLES,DROP,INDEX,ALTER ON moodle.* TO 'admin'@'localhost' IDENTIFIED BY 'admin';
```

The forth step is to create a data directory for holding Moodle files. It must be saved in a

\(^7\) [https://docs.moodle.org/31/en/Installing_Moodle](https://docs.moodle.org/31/en/Installing_Moodle)
directory that is not the same as the server directory and it is mandatory to provide the necessary permissions in order to allow writing to files by a web server user. After that, installation will be carried out on the web browser and some database settings are configured as displayed on (figure 3.1).

Figure 3.1: database configuration settings
Chapter 4 Systems Analysis

4.1 Choice of a recording system

4.1.1 Introduction
There are many firms offering different types of recording systems that was discussed earlier (refer to chapter 2). However, most of the systems are proprietary, thus it requires a purchase of the solution to get technical support and documentation. Although some those systems offer API documentation, it is limited and do not give detailed information. Consequently, it is preferable to work on an open source recording system that is Opencast. There are many benefits for using a system licensed with open source license. First, Opencast can be downloaded without making any initial payment or subscription. This will encourage educational institution to use the system. Second, through searching it has been found many ways where admin user and developer can receive technical support, which the developer of this project has benefited during configuration phase and solving technical issues. These support sites consist of a google group\(^8\) were users can post their questions to be answered with other advanced users. Moreover, users can benefit from Opencast wiki\(^9\) were many information can be found and many topics being discussed about the Matterhorn project. Finally, a detailed API documentation is available but only after installation of the system\(^10\). Following section will discuss Opencast API functionalities and ways for retrieving required data from it.

4.1.2 Opencast functionalities
Opencast recording system is a web based application, it can be accessed from the web browser through localhost server\(^11\). Recordings can be added manually or automatically by connecting it with a recording hardware. For the simplicity, recordings were added manually through adding events (figure 4.1). Then, details can be added and a recording can be added in three fields a video for presentation, presenter and audio, or as it was tested it is also possible to upload only one resource. Another feature for organising events is through using a collection to assign each event with a particular series. This explains that recordings are stored separately known as events or in collection by creating series. Next, a demonstration will be given on data that is required to implement integration.

---

\(^8\) [https://groups.google.com/a/opencast.org/forum/#!forum/users](https://groups.google.com/a/opencast.org/forum/#!forum/users)

\(^9\) [https://opencast.jira.com/wiki/](https://opencast.jira.com/wiki/)

\(^10\) [http://localhost:8080/rest_docs.html](http://localhost:8080/rest_docs.html)

4.1.3 Testing Opencast API
After installing and configuring OC recording management system, its API documentation is accessible through localhost server rest_docs.html. There are many APIs, but the focus is on those that can generate recording JSON data. For analysing JSON structure of the API Postman tool were used.

4.1.3.1 Postman tool
Postman\textsuperscript{12} is a tool that enable users to interact with APIs by initiating HTTP verbs GET, POST, UPDATE, etc. The tool is simple to use and has well designed interface. Unlike other tools which requires knowledge of writing command in terminal, with this tool almost all parameters can be add by choosing from a dropdown menu or filling textboxes. Also, since securing an API is a critical aspect, and many APIs providers require using different authentication mechanisms, it can be configured easily using Postman.

4.1.3.2 API Authentication mechanism
The official website of Opencast\textsuperscript{13} demonstrated in its documentation the use of several types of API authentication such as Basic Auth and Digest Auth. However, after researching and

\textsuperscript{12}https://www.getpostman.com
\textsuperscript{13}https://cast.switch.ch/documentation/api/1.0.0/authentication/#basic-authentication
communicating with Opencast community it was found that Digest Auth is used and it is the one that worked on Opencast 2.2.0 version. This type of authentication is different than Basic Auth in the way that credentials are sent over HTTP request. Password and username are sent in clear text when using Basic, while with Digest credential is encrypted with MD5 making it less vulnerable to sniffing attacks [4]. Although it seemed a secured authentication mechanism, it is not considered the ultimate security solution on the cyber space. This is because Digest scheme dose not encrypt message content [5].

To establish a connection with Opencast API, Digest Auth must be configure first. Official documentation of Digest Auth is not clear, it has been a process of trial and error to figure out how to perform HTTP requests. Also, some explanation was given by establishing an online conversation with Opencast community. Sending a request through Postman with or without authentication will respond with a redirection to opencast login page (figure 4.2). It was suggested to use cUrl tool on terminal to investigate further the issue of not retrieving JSON content. It can be seen from the response generated by curl, opencast sever included some parameters which were probably not capture by postman (figure 4.3). from this point 401 response was sent back asking for full authentication. Another request was constructed with pervious parameters being set, as a result, JSON content was retrieved successfully (figure 4.4). the missing parameter that was required in postman is “nonce” by coping it to postman the request succeeded in retrieving JSON.
Figure 4.2: redirection to Opencast login page

Figure 4.3: different response generated using cUrl
4.1.3.3 Data required to perform integration

Before deciding which information is required for the integration process, it is crucial to understand the way Synote consume data. From synote.org website, it can be seen that creating a recording takes three parameters; title of the recording, description and URL. In Opencast API, smellier information is provided as well, however, the structure of the JSON data retrieved is complex. It consists of numerous number of nested arrays. Consequently, looping through arrays is needed to retrieve required data. Figure (4.5) displays an example of JSON data structure of archive/episode.json API. Next chapter will demonstrate the solution development phase of the Synote integration project.
Figure 4.5: Structure of episode JSON data
Chapter 5 Solution Design

5.1 Design phase

5.1.1 Introduction
This chapter will give explanation the solution design that is used for the purpose of integration. First of all, the solution design will be discussed, this will include a diagram that shows the application components. After that, each component will be explained with further details.

5.1.2 UML
The Unified Modelling Language (UML) is a popular common purposes utilized for specifying, visualizing, constructing and documenting software components. It has numerous features that assist in making decisions and understanding of a particular system [8]. There several types of UML diagrams, however, Component Diagram will be used for describing different parts of the software [9]. The main purpose of using UML component diagram is to represent libraries, packages, files etc. [10]. The following sections will provide further details.

Figure 5.1: UML component diagram of the system
5.1.3 Jetty server
The first component of the UML diagram is Jetty server\textsuperscript{14}, which is an open source HTTP server. It is installed alongside with Opencast system. This server provides a web server and a javax.servlet container also provides support for integration of several services such as HTTP, WebSocket, JMX and JAAS. Moreover, it can be used for various projects whether in production or development and can be installed in different frameworks, application servers and clusters. Opencast dependence on Jetty to run, thus it must be configured according to the required specifications.

5.1.4 Apache2
Second component is Apache http server that was developed for providing a powerful open-source web server\textsuperscript{15}. The sever provides various features. It provides support for server side programming of some common languages such as Perl, Python, and PHP. Also, several types of authentication modules are supported; mod_access, mod_auth, and mod_digest. Other features included with the server is a support of a variety of secured layers; Secure Sockets Layer, Transport Layer Security, Mod_SSL, Mod_Proxy, and aURL. The main purpose of using this sever is to host a local copy of the solution and run PHP scripts.

5.1.5 PHP scripts
A Hypertext processor known as (PHP) is an open source widely used server-side scripting language. This scripting language offers various features that enables web developers to create dynamic websites faster. Furthermore, PHP enable users create some functions to interact with a system to perform create, open, read and write tasks on system files. Other features also include gathering data from forms for storage or manipulation before delivering it back to end-users. Interacting with databases is also possible; PHP can perform create, read, update and delete (CRUD) on several types of databases [11]. For this project PHP is being used for retrieving data from Opencast using cUrl framework, interacting with MariaDB database, and creating an API using CodeIgniter framework.

5.1.6 PHP CodeIgniter Framework
Instead of developing new wheels it is preferable to use a framework. This component is critical for developing an API that will allow Synote application to consume Opencast data. CodeIgniter is an application development framework that uses Model View Controller (MVC) concept. The main feature of this framework is to enable PHP developers building web apps faster while leaving small footprint and delivering high performance. This framework was used in combination with libraries offered in Github by “chriskacerguis” which provides full RESTful server implementation.

\textsuperscript{14} http://www.eclipse.org/jetty/
\textsuperscript{15} https://secure.php.net/
5.1.7 MariaDB database

Last but not the least component of the UML diagram is the data storage MariaDB\(^{16}\). This is one of the most famous open source database servers and it is used by remarkable companies such as Wikipedia, Facebook and Google. In contrast to MySQL, MariaDB is much faster, scalable and robust, plus it provides an SQL interface to access data. This component is configured at first for storing Opencast data, instead of using the default built-in database, which affected the system performance. Also, another database was created for storing filtered data that can be accessed by the API allowing Synote to consume the required data. In the Following chapter solution development phase will be discussed.

\(^{16}\)http://mariadb.org/about/
Chapter 6 Solution Development

6.1 Development phase

6.1.1 Introduction
After discussing each part of the system that is required for the integration, this section will evolve into demonstrating development of the application. To begin with, first section will include scenarios that should be implemented. Then, further explanation is given on how the application was developed. Table 6.1 gives a summary of the technologies that were used.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Programming language</th>
<th>Framework</th>
<th>Database</th>
<th>Server</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu 16.04</td>
<td>PHP</td>
<td>CodeIgniter</td>
<td>MariaDB</td>
<td>Apache2</td>
<td>Workstation12</td>
</tr>
<tr>
<td>cUrl (library)</td>
<td></td>
<td></td>
<td></td>
<td>Jetty</td>
<td>Opencast</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Visual code</td>
</tr>
</tbody>
</table>

Table 6.1: Summary of the technologies that have been used

6.1.2 Technologies summary
These technologies have added to the knowledge and technical skills of the developer of this project. The VMware workstation12 is a critical part, this is because it hosts the Linux environment, which host the rest of software; servers, frameworks, and database. Briefly, Opencast is the recording system that will be utilized to retrieve recordings data. Then, PHP alongside with cUrl framework are being used for the JSON retrieval and filtering process. The database is mainly used for storing required JSON data (title, URL, Description etc.) that will eventually be used to query it using an API.

6.2 Scenario discussion and implementation
There are several scenarios that were discussed before starting implementation, which took place in several meetings with Prof. Mike Wald, and Dr. Yunjia. Table 6.2 summaries these scenarios:
<table>
<thead>
<tr>
<th>ID</th>
<th>Scenario</th>
<th>Importance</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Query data from Opencast API. The required data are; URL, Title, Description.</td>
<td>MUST</td>
<td>DONE</td>
</tr>
<tr>
<td>2</td>
<td>Queried data should be inserted into database.</td>
<td>OPTIONAL</td>
<td>DONE</td>
</tr>
<tr>
<td>3</td>
<td>The database should not be queried directly by third party clients. Create an API that will allow users/applications to retrieve the data.</td>
<td>MUST</td>
<td>DONE</td>
</tr>
<tr>
<td>4</td>
<td>New recordings must be added automatically in appropriate intervals (hourly based is preferable)</td>
<td>MUST</td>
<td>DONE</td>
</tr>
<tr>
<td>5</td>
<td>Provide and option to retrieve JSON data from the API based on date (greater than or equal to specific date)</td>
<td>MUST</td>
<td>DONE</td>
</tr>
<tr>
<td>6</td>
<td>Find a way to post those recordings to Moodle LMS</td>
<td>OPTIONAL</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 6.2: Scenarios summary for implementation

6.2.1 Flowchart
A flowchart is a visual representation of steps describing a process that would be established by the system. The reason of visualizing these steps is to determine which step begin first, examine conditions, and inspect data flow. Figure 6.1 displays the solution processes. Data flow will start by retrieving JSON content, converting it to PHP object, selecting required parameters, inserting it into database, and finally the API will enable other clients to consume the data.
6.2.2 Data retrieval
The first scenario is to query JSON data. It was accomplished using PHP cUrl library to initiate a GET request. It is a library that is supported by PHP which allows to initiate communication with various types of servers\textsuperscript{17}. The first three lines are used to define the server to be connected to (in this case, Opencast API), and username and password. Then, options are defined depending on the requirements of Opencast. The header option was disabled, this is used for preventing a printout of response header, and retrieving only JSON content. The second part of the code is to validate if any error might occur as displayed in figure 6.3. A sample of the output is displayed in figure 6.4.

\textsuperscript{17} http://php.net/manual/en/intro.curl.php
```php
$url = "http://localhost:8080/archive/episode.json";
$username = "opencast_system_account";
$password = "CHANGE_ME";
$options = array(
    CURLOPT_URL => $url,
    CURLOPT_HEADER => false, // to avoid printing response header this must be false
    CURLOPT_VERBOSE => true,
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_FOLLOWLOCATION => true,
    CURLOPT_SSL_VERIFYPEER => true, // for https
    CURLOPT_USERPWD => $username . ":" . $password,
    CURLOPT_HTTPAUTH => CURLAUTH_DIGEST,
    CURLOPT_CUSTOMREQUEST => "GET"
);

$ch = curl_init();

$opt = curl_setopt($ch, CURLOPT_HTTPHEADER, array(  
    'Content-Type:application/json',  
    'X-Requested-With:Digest',  
    'X-Opencast-Matterhorn-Authentication : true'
));

curl_setopt_array($ch, $options);

try {
    $raw_response = curl_exec($ch);
    
    // validate CURL status
    if(curl_errno($ch))
        throw new Exception(curl_error($ch), 500);

    // validate HTTP status code (user/password credential issues)
    $status_code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
    if ($status_code != 200)
        throw new Exception("Response with Status Code [" . $status_code . "].", 500);
} catch (Exception $ex) {
    if ($ch != null) curl_close($ch);
    throw new Exception($ex);
}

if ($ch != null) curl_close($ch);
$data = json_decode($raw_response, true);
```

Figure 6.2: cUrl code for requesting JSON content

Figure 6.3: This part responsible for validation agent any errors
Figure 6.4: A sample of JSON content from episode API
6.2.3 Inserting data into database

Second requirement is to get the retrieved content, filter data that is only needed for integration and store it to the database. This has been done by writing a PHP script; the first step is to establish a connection with SQL. After that, JSON content is included using include function `include('.....')`. As can be seen in figure 6.5, the code is divided into three parts. First, a for loop is executed to retrieve three parameters from episode JSON API, which are “Title”, “URL”, and “Description. Second, to avoid inserting duplicates of the same records that might eventually overload the database, an SQL query is executed to select all records for comparing it with records that is aimed for insertion. The last part is for executing a message in case of successfully inserting records, or when records being already inserted earlier.

```php
<?php
// will loop through json file to get the title and url for each recording
$count = count($data['search-results']['result']);
for($i = 0; $i < $count ; $i++) {
    $title = $data['search-results']['result'][$i]['mediapackage']['title'];
    $desc = $data['search-results']['result'][$i]['metadata']['description'];
    if($desc == null){
        $desc = "recordings test";
    }
    $url = $data['search-results']['result'][$i]['mediapackage']['publications']['publication']['media']['track']['url'];
    // this query will check if recordings data were already inserted
    $check = mysqli_query($link, 'select * from testi where rec_title="'.$title.'" and rec_url="'.$url.'"');
    if($checkrows == 0) {
    $insert = "INSERT INTO testi (ID, rec_title, rec_desc, rec_url) VALUES(0,"'.$title.'","'.$desc.'","'.$url.'")";
    mysqli_query($link, $insert);
    $inserted = true;
    }
    // display a message in either cases
    if($inserted){
        echo "records data inserted successfully";
    }else{
        echo "Record exists";
    }
}
mysqli_close($link);
?>
```

Figure 6.5: A sample of JSON content retrieved from episode API
Available Recordings

![Figure 6.6: A sample of inserted records](image)

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1</td>
<td>recordings test</td>
<td><a href="http://localhost:8080/static/mh_default.org/internal/5c0db77-6592-4217-9820-455dbede66be/39e5c0e-a-594d-4234-b05c-9916d3edf9d/How_to_download_Youtube_Video_in_UBUNTU_Linux_UPDATED_t_xPlsYmXYY.mp4">http://localhost:8080/static/mh_default.org/internal/5c0db77-6592-4217-9820-455dbede66be/39e5c0e-a-594d-4234-b05c-9916d3edf9d/How_to_download_Youtube_Video_in_UBUNTU_Linux_UPDATED_t_xPlsYmXYY.mp4</a></td>
</tr>
<tr>
<td>Symantic WEB</td>
<td>recordings test</td>
<td><a href="http://localhost:8080/static/mh_default.org/internal/5d50a599-6d84-4194-9da2-6d02ec6645328e/bfa2e9e-2b-2450-4f19-2ba2-0e1de73a2e25e4">http://localhost:8080/static/mh_default.org/internal/5d50a599-6d84-4194-9da2-6d02ec6645328e/bfa2e9e-2b-2450-4f19-2ba2-0e1de73a2e25e4</a></td>
</tr>
<tr>
<td>Accessibility</td>
<td>test</td>
<td><a href="http://localhost:8080/static/mh_default.org/internal/1a8d9bf-9e38-4e6b-bd27-03db5d4c4f19/bfa19d90-23e-466e-9eb0-8542093f796/How_to_download_Youtube_Video_in_UBUNTU_Linux_UPDATED_t_xPlsYmXYY.mp4">http://localhost:8080/static/mh_default.org/internal/1a8d9bf-9e38-4e6b-bd27-03db5d4c4f19/bfa19d90-23e-466e-9eb0-8542093f796/How_to_download_Youtube_Video_in_UBUNTU_Linux_UPDATED_t_xPlsYmXYY.mp4</a></td>
</tr>
<tr>
<td>Security</td>
<td>test</td>
<td><a href="http://localhost:8080/static/mh_default.org/internal/690c890-fd7-43c7-7e6a-02d663511bcb/740f10c-9dce-4808-be7b-1a2c557fad1/How_to_download_Youtube_Video_in_UBUNTU_Linux_UPDATED_t_xPlsYmXYY.mp4">http://localhost:8080/static/mh_default.org/internal/690c890-fd7-43c7-7e6a-02d663511bcb/740f10c-9dce-4808-be7b-1a2c557fad1/How_to_download_Youtube_Video_in_UBUNTU_Linux_UPDATED_t_xPlsYmXYY.mp4</a></td>
</tr>
<tr>
<td>Web Development</td>
<td>all types of recordings</td>
<td><a href="http://localhost:8080/static/mh_default.org/internal/7e8d779b-15c-4d4d-83db-231965024545/143db39-9ac7-47fa-8890-65cf3350f61/composite.mp4">http://localhost:8080/static/mh_default.org/internal/7e8d779b-15c-4d4d-83db-231965024545/143db39-9ac7-47fa-8890-65cf3350f61/composite.mp4</a></td>
</tr>
<tr>
<td>Web Architecture</td>
<td>test</td>
<td><a href="http://localhost:8080/static/mh_default.org/internal/41d2a615-5e7a-40d0-83ab-408d3d697741/08120287-08c7-43e4-a1bc-655e561bb74a/composite.mp4">http://localhost:8080/static/mh_default.org/internal/41d2a615-5e7a-40d0-83ab-408d3d697741/08120287-08c7-43e4-a1bc-655e561bb74a/composite.mp4</a></td>
</tr>
</tbody>
</table>

Figure 6.7: Preventing duplicates indicated with “Records exists”
6.2.4 automation of the process

PHP scripts that was written should run automatically without requiring human interaction. In contrast to JavaScript, JS functions can be invoked through defining a duration and by listening to initiated events by clients, PHP has no such features because it is invoked on the server-side. After searching, creating CRON jobs was found to be the solution for creating tasks on the server that will execute PHP scripts based on predefined frequency (minutes, hours, days etc.) [12, 13]. As it was decided earlier the frequency of checking for new recording is based on hours, which means the scripts will run 24 times a day. For the purpose of testing, the command is set to run every ten minutes temporarily as follows:

- */10 * * * * /usr/bin/wget -q -O temp.txt http://localhost/sync/index.php

And the result can be seen in figure 6.8, a screen shot of the inserted data retrieved using MySQL Workbench software:

![Figure 6.8: A screen shot of the data being inserted automatically](image-url)
To check whether new recordings data are inserted into the database, a recording was added to Opencast system. As can be seen in figure 6.10, a record with “Web Technologies” title was added successfully.

Figure 6.9: A new record being added through checking every 10 minutes

6.2.5 Recordings API
The last requirement to be fulfilled is to build an API that will initially enable third party applications to consume recordings data. There are many ways for developing an API web application, however, utilizing a framework with some libraries will facilitate several processes unlike writing a program from scratch. Consequently, CodeIgniter PHP framework in combination with REST Server library[^18] were used to accomplish this part. There are four main scripts need to be added to the project directory:

1. REST_Controller.php
2. rest.php
3. Format.php
4. MY_Model.php

[^18]: https://github.com/chriskacerguis/codeigniter-restserver
These PHP scripts must be added to specific directories as can be seen in Table 6.3:

<table>
<thead>
<tr>
<th>Name</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 REST_Controller.php</td>
<td>/var/www/html/records_api/application/libraries/REST_Controller.php</td>
</tr>
<tr>
<td>2 rest.php</td>
<td>/var/www/html/records_api/application/config/rest.php</td>
</tr>
</tbody>
</table>

Table 6.3: Directories of the required scripts

The first script is a library used for building a restful API calls (GET, PUT, DELETE, UPDATE etc.). Then, rest.php is used to define configuration required for the API. From the script name, Format.php is a library used for converting between various formats such as JSON, XML and CSV. Finally, MY_Model.php is a base model that adds CRUD SQL functions, which facilitate interaction with databases.

6.2.5.1 Authentication

An authentication mechanism that is Basic authentication was added to improve the security of the API which only allows valid users to access it. Moreover, because Basic Auth is not considered the complete security solution, whitelisting of IP was introduced, as well. Even if credentials were granted by non-authorized client, access to the API will be prevented as long as the intruder IP address is not defined in the whitelist. Currently, localhost is the only enabled IP address. To gain access to recordings API using Postman, define Basic Auth in the authentication tab and add username and password (see figure 6.10).

![Figure 6.10: Adding authentication header](image)
6.2.5.2 GET all records
The first API that was coded is for allowing users to retrieve all records data. A class was written first to extend REST_Controller. Then, recordAll function was added to allow querying all recordings data. The figure below displays a sample of the code.

```php
function recordAll_get() {
    $this->load->model('Model_records');
    $record = $this->Model_records->get_all();
    if (isset($record)){
        $this->response(array('records' => $record));
    }else{
        $this->response(array('status' => 'failure', 'message' => 'the specified recording could not be found', REST_Controller::HTTP_NOT_FOUND));
    }
}
```

Figure 6.11: A sample code of GET all recordings

6.2.5.3 GET records by ID
The second API is for retrieving specific recordings data by providing an ID number. The ID can be added in the third argument of the API URL. Then, it will be checked against database records.

```php
class Api extends REST_Controller {
    function __construct(){
        parent::__construct();
    }

    // to get single record based on ID
    function recordById_get() {
        $record_id = $this->uri->segment(3);
        $this->load->model('Model_records');
        $record = $this->Model_records->get_by(array('ID' => $record_id));
        if (isset($record)){
            $this->response(array('status' => 'success', 'message' => $record));
        }else{
            $this->response(array('status' => 'failure', 'message' => 'the specified record could not be found', REST_Controller::HTTP_NOT_FOUND));
        }
    }
}
```

Figure 6.12: A sample code of GET recordings based on ID
6.2.5.4 GET records based on date
This API was added to enable users to query data based on specific date. The API dose not retrieve single value; it retrieves multiple values depending upon the specified date. For instance, if the specified date is 10-9-2016, the request will query all recordings beginning from that date to the latest added recording.

6.2.5.5 GET records based on title
To give more preferences for end-users JSON content can be retrieved by specifying the recording title. Instead of remembering the ID number of particular record, which by time might be a long integer. Knowing the title of a recording will facilitate the process.

6.2.5.6 Testing APIs
The APIs for requesting all recordings data and by specifying an ID were tested using Postman tool. First, to query everything the following URL is requested using GET method (figure 6.13 displays the result):

- **GET** [http://localhost/recording.Api/recordAll/](http://localhost/recording.Api/recordAll/)

![GET](http://localhost/recording.Api/recordAll/)

Figure 6.13: recordings response in JSON format

To query a specific recording data, the second API call accept an ID parameter in the third URL argument. The following URL is used (see figure 6.14):

- **[http://localhost/recording.Api/recordById/[ID]](http://localhost/recording.Api/recordById/[ID])**
The following API is for retrieving data based on recording title. The endpoint URL mentioned below is used for this task (see figure 6.15):

- [http://localhost/recording/Api/recordByTitle/{Title}](http://localhost/recording/Api/recordByTitle/{Title})

Querying contents by date has two APIs. The first one, allows users to request data based on date only (../recordByDate/{Date}). It can be defined according to this format (Y-m-d), for instance, the third argument of the URL can be written as 2016-9-13. The response retrieved all records equal to and greater than the specified date. However, Synote client may requests data on the same date and at different time, which can be achieved using the second endpoint (../recordByDate/{Date Time}) in accordance with this format (Y-m-d H:i:s). The following endpoints are used for this part (figure 6.16-17):

- [http://localhost/recording/Api/recordByDate/{Date}](http://localhost/recording/Api/recordByDate/{Date})
Figure 6.16: Retrieving JSON content using Date argument

- [URL](http://localhost/recording/Api/recordByTime/{Date_Time})

Figure 6.17: A more specific API using Date and Time for content retrieval
6.3 Web app and Recordings API Documentation

For any project documentation is an essential task, otherwise knowledge would not be delivered to the intended audience. Thus, a web based application was designed for this purpose. Also, some features were developed to assist end users to accomplish their tasks with ease. To begin with, the web app has a function to look for new recordings. By clicking the “Recordings” tab users can review recordings contents available in Opencast and check if new recordings were added in Opencast. It will also execute a query to compare with existing recordings data and will insert the new records. Moreover, a quick link tab is provided to access Opencast administration control panel. The documentation tab includes six endpoints with details and examples on each URL usage and a side menu is designed for fast access to each API section. Finally, some instructions are given that demonstrates the use of Basic Auth to access the API. The documentation is available as web page (http://localhost/sync/apiDoc.php).
Chapter 7 Testing and Evaluation

7.1 Testing Systems

7.1.1 Introduction
Testing in any software development project is an important task that should be carried out. It helps in identifying technical issues or bugs at early stages before it becomes major problems later on. Thus, for this project, testing and evaluation were implemented regularly throughout development of the solution. The testing took place with different systems.

The process was performed iteratively in several steps. First, it is critical to be sure that the recording system is publishing recordings and producing metadata, which is consumed by Synote. Second, requesting JSON content from opencast should retrieve a loop of specific data not everything due to the complexity of the data structure. Third, database is where recordings content is stored, thus testing must be performed to guarantee data validity and integrity. Forth, because the process of checking for new recordings must be functioning automatically, testing it is critical. Finally, the recordings API is tested to make sure it produces the required response and to handle errors that might occur, which was tested using RESTClient tool. Following sections will discuss these steps in greater details.

7.1.2 Testing Opencast
The recordings uploaded to this recording system need to be tested with different video formats. Several popular formats were chosen for this task; AVI, MP4, FLV, WMV, and MP3 to test the audio format. All of the formats were uploaded and published successfully except with publishing an mp3 file (see figure 7.1). This happens because Opencast is configured to combine a video and an audio files into one recording. In case of other format that includes an audio there were no publishing error. An event was created to test combining audio and video files by uploading them into audio field and video field and it worked successfully (see figure 7.2). Even though different video formats were uploaded, they were converted to mp4 format. This configuration can be changed if required, or customization of encoding and decoding a video could be configured on Synote side.

![Figure 7.1: Error in publishing only audio](https://documentation.opencast.org/r/2.2.x/admin/configuration/encoding/)

19 https://documentation.opencast.org/r/2.2.x/admin/configuration/encoding/
7.1.3 Testing requesting JSON content
It is critical to be sure that the required data from Opencast (title, description and url) be available before being inserted to the database. In some circumstances, Opencast fails in publishing recordings, which results in empty cells as can be seen in figure 7.3. This error was handled by avoiding retrieval of empty URL. However, it did not work because empty cell actually returns a string value (white space). Thus, it was handled by counting the number of characters which should be greater than zero.

![Figure 7.3: Empty field called from Opencast](image)

7.1.4 Testing database insertion of recordings
The database was tested to prevent any integrity errors that may occur. Although handling empty URL worked without issues, it was found that the database contains empty URL fields. Another problem was found as well, which causes date field to be set with zeros value (see figure 7.4). The former issue was fixed by counting URL length that will result in a value greater than zero before inserting it, otherwise the recording will be avoided. With the latter issue, instead of commanding the database to insert a time-stamp when a new recording get inserted, date field was inserted by adding “now()” function in PHP inserting script (see figure 7.5). Also, to prevent replication of records, the script will always compare Opencast recordings with records already inserted.
7.1.5 Testing automatic script execution

New recordings might be added at any time. Thus, checking on regular bases is important. For this task CRON job was created, which should execute a PHP script, that will check and insert recordings based on each hour. This was accomplished by creating an event and waiting for the execution to happen. This exaction exactly occurred after one hour, the new record can be seen in the database (see figure 7.6).
7.1.6 Testing API Output

In this section recording APIs were tested to check whether output generated is accurate and to check handling of error when it occurs. The first part that was tested is API security, which uses Basic Auth and IP whitelisting. The username was changed to a user that does not exist in the system. This resulted in an unauthorized response 401 as displayed in (figure 7.7). Localhost IP is enabled by default, to make sure whitelisting works the IP was removed and added to blacklist which responded with unauthorized access. Using whitelisting is preferable since only allowed clients are added. The second part of this task was to test the API output and handling of errors. The API responded with the required data as long as the third argument provided in the URL is valid and relevant. However, it was found that the generated error messages are identical, thus it was modified to give more meaningful information for end-users (see figure 7.8). Moreover, SQL injection is well handled by the library used for creating these APIs. Any special charters entered with a request, responded with bad request error code 400 (see figure 7.9).
Figure 7.8: Meaningful error message

Figure 7.9: handling of special characters
During testing of the APIs with Professor Mike Wald on 19th September 2016, a new requirement was suggested to retrieve recordings data based on between dates. This was accomplished by adding an API that can send a request with two arguments. These arguments should be in an order that the first date must be lower than the latter date with time included. The reason for defining time with date is that a user might demand to retrieve recordings content in the same date, but at different time. However, if only date was defined in the first and second arguments the response will be empty. Thus, including time with date will guarantee content retrieval. The following API is used for this task:

- **GET** [http://localhost/recording/Api/recordBetDate/{Date_Time1}/{Date_Time2}](http://localhost/recording/Api/recordBetDate/{Date_Time1}/{Date_Time2})

Figure 7.10 shows an example to querying data between “2016-09-19 12:34:51” and “2016-09-19 13:46:54”. It can be seen that the dates are similar, but it varies in time. Moreover, to prevent errors that may occur while inputting dates, the code checks for several conditions. First, it looks for date format that should be based on “Y-m-d H-i-s”, otherwise the API will not retrieve anything. Furthermore, comparison check is provided to make sure first date is earlier than the second date. Finally, another check is implemented to guarantee alphabetic characters are not allowed.

![Figure 7.10: Response of querying data between different times](image)

```json
1. {
2.   "status": "success",
3.   "message":
4.   [
5.     {
6.       "ID": "98",
7.       "rec_title": "testWMV",
8.       "rec_desc": "testing wmv format",
9.       "rec_url": "http://localhost:8080/static/mh_default_org/internal/fd18cd48-a69d-42a0-96bf-1bd87b0a8970/6a866d9c-988a-4a9b-ad1d-cdb628f51a1f/synote.mp4",
10.      "rec_date": "2016-09-19 12:34:51"
11.    },
12.   {
13.     "ID": "99",
14.     "rec_title": "testMP4",
15.     "rec_desc": "testing mp4 format",
16.     "rec_url": "http://localhost:8080/static/mh_default_org/internal/b23544b7-ea82-4fd8-9fd4-fc388ac7785/6108db4c-4939-4e60-a453-6fe6381baa7d/synote.mp4",
17.      "rec_date": "2016-09-19 13:46:54"
18.    },
19.   {
20.     "ID": "100",
21.     "rec_title": "testAudioAndVideo",
22.     "rec_desc": "testing adding two sources audio and video",
23.     "rec_url": "http://localhost:8080/static/mh_default_org/internal/dbf81237-8bf3-448c-980d-bd6563f8a2f2/679ee28d-a977-4c6c-8cc3-ed9f1dbb8b88/synote_no_audio.mp4",
24.      "rec_date": "2016-09-19 12:34:51"
25.   }]
```
8.1 Project Plan
The project was allocated a fourteen weeks beginning from June 1, 2016 to September 2. However, due to unexpected circumstances the duration of this project was extended by three more weeks, to be submitted on September 23, 2016. To ensure on time submission of deliverables, a project plan was created, which specified a time frame of the tasks that need to be carried out. The project plan included several main tasks and some were further divided into subtasks. The project progress was proceeding despite the challenges and difficulties the developer had during implementation phase that resulted in a need for time adjustment on several tasks (see appendix A).

The first three weeks were devoted for background research about Web Accessibility technologies, investigating integration process, and gathering requirements that is needed to be implemented for this project. However, understanding how the pervious integration was implemented was a challenging task, besides resources were difficult to be found. This is because the project nature requires excellent skills in back-end programming. Furthermore, since time is limited, instead of performing integration with multiple recording system, a decision was made to work on Opencast because it is an open source software also many documentation and technical support are available.

The next three weeks, after finalizing background research, the second task was to configure system requirements that consist of hardware and software levels. Hardware specification for this project was configured with ease. However, software configuration took longer duration than expected. this is due to several reasons; first is lack of knowledge in some areas such as Linux environment. Second, Opencast recordings systems was not working properly after installation because of some issues with several packages. This required communicating with Opencast developers to solve technical issues with the software, which in some cases required several days to receive the demanded instruction.

Then, the third task was to make sure that systems configuration requirements were working appropriately. This task was accomplished after implementing the technical instruction received from Opencast community for fixing some errors. After that, several meetings were attended for getting further knowledge of the requirements and scenarios for the project development.

On week seven and eight, the task was to understand structure of Opencast API and which endpoint would retrieve the required data. At first, the task was challenging because API documentation of the recording system give basic information. As a result, the process was to test those API endpoints. Although jQuery was used for this purpose, after getting feedback from the technical supervisor, it was decided to use back-end programing language to retrieve JSON contents. Thus, PHP scripting was used to build the solution.
On the proceeding weeks (9 and 10) the task was to develop the web application. However, the developer of this project is a beginner in the field of back-end development, attending several online courses on Lynda and Udemy were critical. After that, the web app was developed, which has functionalities to retrieve data from Opencast server, then insert the filtered data into a database. Moreover, it was important to make the script check for new recordings uploaded on the server and this was accomplished during these two weeks. At this point, data is available on the database, but third party application in this project case Synote must be able to request data from the database.

Because of that it was recommended to develop an API for performing GET request. Unfortunately, resources on developing an API using PHP scripting language are scarce. Thus, watching an online course have assisted in grabbing fundamental knowledge of API development process. Then, a library was found in Github that facilitate building an API with GET, PUT, POST, and DELETE methods, so it was used for this project. This task took 3 weeks to be accomplished, because the library requires use of CodeIgniter PHP framework, which uses MVC model that the developer had to learn and understand how it works.

8.2 Risk Management

Measuring risks associated with any project is an extremely important task to prevent any unexpected incidents. OneNote software was used to document of all the tasks and configuration settings applied on the local virtual machine. This has been useful during implementation phase due to that several applications did not work at some points, which required reconfiguring. Also, instead of installing the systems on a personal computer, a virtual machine was created. A laptop hardware and software could crash at any time, however, using a virtual machine prevents these risks. Moreover, creating backup files is faster and easier with VM, which was considered before proceeding with any critical task. To be on the safe side, one local and two external hard disks were used for backups. The local one was used for fast restore in case of any crashes, while the external was created in case the local backup get corrupted. Project source code was also considered as high risk. Thus, OneDrive has been used to make an online backup with appropriate versioning of the files depending on the changes made on the source code. The risk of losing the documentation was considered, as well. As a result, all documents were stored locally and on the cloud.

8.3 Challenges and Reflections

The challenge of integrating systems is that in involves understanding how the systems works. To acquire this knowledge, it was essential to interact with several systems. Also, the project required installing and configuring different systems, which was challenging due to technical issues that took considerable time to find a solution and sometimes the only way to progress was proceeding through trial and error process. Then, the challenge was to understand the solution that need to be implemented to accomplish implementation of the project. Back-end development has been challenging too, because I had basic skills in software development. This issue was resolved by watching many
courses online such as Javascript, Nodejs, PHP, CodeIgniter framework, RESTful API, and cUrl. Also, the advantage of working with open source programming languages is the availability of communities that can provide valuable instructions and examples. Meeting the supervisors was useful too, it helped in getting some tips, feedback and discussing the project requirements.

Despite the challenges that was encountered, working with the project made me gain many skills. First, APIs has great power in improving and enriching a particular system. RESTful API is used for developing web services and I had used it to build an API which Synote can use to consume recordings data. Plus, it was the first time to build a web application using MVC model, which was used for developing the recordings API. From this experience, I would suggest several points that would help those students who are going to work on similar project:

- Time is a scarce resource; it is very important to make a good plan to manage your time effectively. Make sure to have a plan that includes all the activities that you aim to achieve.
- Documentation is your path to success. Working on a technical project means a lot of configuration set ups. Sometimes things go wrong which will require you to reconfigure settings, executing commands having them documented will save time and effort.
- Allocate time for learning programming in case it is needed. Online course will assist you in gaining programming skills, Udemy and Lynda websites have a lot of courses.
- Virtualization is preferable to avoid any hardware failure or software crashes. This will be useful to easily restore a backup.
Chapter 9 Conclusion

This is the final chapter that concludes this project report with final thoughts. Moreover, it demonstrates some suggestions for future enhancements of expanding Synote integration with other recordings systems and Moodle LMS.

9.1 Summary
The dissertation has demonstrated the process of developing and implementing integration of Opencast recording system with Synote. It started by investigating several recording systems to examine availability of API, compatibility of integration, availability of documentation and technical support which resulted in utilizing Opencast for implementation. Then, the report analysed systems components and requirements needed for integration process. Furthermore, a demonstration was given on the development process of a web application that automates retrieval and insertion of recordings content and provides API documentation for end-users. Moreover, recording API development was explained, which is the critical part that enables Synote to consume recordings data. Also, a testing of the application was implemented and documented to ensure it works accurately. The objective of the project has been met as outlined in the beginning and further suggestion for improvements has been given to be implemented in the future.

9.2 Future work
Several enhancements can be further applied to the integration of Synote to include different types of recording systems and learning management systems. First of all, the web application would be more enriched with lecture recordings if more recordings data could be retrieved from another recording systems API. Even though the web application API provides the required recordings data, there are other features that should be considered such as users’ authorization and single sign-on. Also, this project can be developed further to add some features that will enable users to access recordings through Moodle LMS. This will organize sharing of lecture recordings with end-users by enrolling them to courses instead of sharing all contents, which may affect performance. Finally, annotated recordings can be embedded into LMSs to facilitate interaction with the content.
References


Figure A.1: Gantt Chart of the project
Appendix B API Configuration

In order to make the API works, there several configurations need to be applied. First, insure a working Apache server is configured properly with PHP and SQL. Then, enable .htaccess and change the folder directory according to your server (see figure B.1):

```php
RewriteEngine On
RewriteBase /recording/
#Removes access to the system folder by users
RewriteCond %{REQUEST_URI} ^system.*
RewriteRule ^(.*)$ index.php?/$1 [L]
#When application folder isn't in the system folder
RewriteCond %{REQUEST_URI} ^application.*
RewriteRule ^(.*)$ /index.php?/$1 [L]
#Checks to see if the user is attempting to access a valid file,
#such as an image or css document, if this isn't true it sends the
#request to index.php
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php?/$1 [L]
```

**Figure B.1: Configured .htaccess for making changes in the server**

Whitelisting is enabled for security purposes, if the API is to be used by specific IP addresses it must added in “application/config/rest.php” as can be seen in figure B.2:

```php
$cfg = array( 
    'rest_ip_whitelist_enabled' => TRUE,
    
    'rest_ip_whitelist' => array(),
    
    'rest_ip_whitelist' => array(123.456.789.0),
);
```

**Figure B.2: Whitelisting configuration**
Alongside with whitelisting security layer, Basic Auth was configured, as well. To access the API username and password must be provided “admin,1234”. Other users can be added as displayed in figure B.3:

```php
$conf['rest_valid_logins'] = ['admin' => '1234', 'synote' => '$note2016'];
```

**Figure B.3: Configuring Basic Auth users**

Finally, the application uses a database for storing recordings data. Thus, database configuration is set in “application/config/database.php”, then it should be enabled in “application/config/autoload.php”, (see figures B.4, B.5):

```php
$db['default'] = array(    'dsn' => '',    'hostname' => 'localhost',    'username' => 'root',    'password' => 'root',    'database' => 'midware',    'dbdriver' => 'mysqli',    'dbprefix' => '',    'pconnect' => FALSE,    'db_debug' => (ENVIRONMENT !== 'production'),    'cache_on' => FALSE,    'cachedir' => '',    'char_set' => 'utf8',    'dbcollat' => 'utf8_general_ci',    'swap_pre' => '',    'encrypt' => FALSE,    'compress' => FALSE,    'stricton' => FALSE,    'failover' => array(),    'save_queries' => TRUE );
```

**Figure B.4: Configuring database connection in CodeIgniter**

API documentation is provided that demonstrates features and give some examples on how to use the APIs. It can be accessed through the following URL:

These are the classes located in system/libraries/ or your application/libraries/ directory, with the addition of the 'database' library, which is somewhat of a special case.

Prototype:

```php
'autoload' => array('database', 'email', 'session'),
```

You can also supply an alternative library name to be assigned in the controller:

```php
'autoload' => array('user_agent' => 'ua'),
```

Figure B.5: Auto-load database in CodeIgniter
Appendix C System Configuration

The OS used for this application is Linux Ubuntu 16.04. Some configurations are required for the web application to run successfully. The application is supposed to run in frequent bases that is each hour, to check for new recordings. Thus, CRON job must be added to run “recordings.php” script regularly (see figure C.1):

![Figure C.1: Automating process using CRONTAB](image)

```plaintext
# daemon's notion of time and timezones.

# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).

# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/

# For more information see the manual pages of crontab(5) and cron(8)
# m h dom mon dow command

0 0 * * * /usr/bin/wget -q -O temp.txt http://localhost/sync/recordings.php
```