UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Integration of Moodle quizzes in EZcast

Dissertation presented by
Laurent MONDRY

for obtaining the Master's degree in
Computer Science

Supervisor(s)
Yves DEVILLE

Reader(s)
Frédéric FERVAILLE, Olivier ROUX , Pierre REINBOLD

Academic year 2016-2017

Contents

List of Figures

List of Listings

Abstract

Active learning is an efficient learning process that teachers can use in classrooms to put the student in charge of his own education. Outside the classroom this interaction between the teacher and his students stops.
The purpose of this master thesis is to develop an extension for EZcast, the new podcasting solution that will be introduced by the UCL starting from September 2017, to promote the use of active learning outside the classroom.
We chose to integrate Moodle quizzes in the video player of EZcast. This way, students and teachers can still interact outside of the classroom. Furthermore, the teacher can use the statistics of the students' answers to guide his courses and insists on less understandable topics.
Our extension is seamlessly integrated in EZcast both in the code and in the general design. It allows teachers to add quizzes from their Moodle courses in EZcast videos. Questions are asked directly to the students while watching their videos. After submitting a quiz, the student receives a feedback on the answers he has chosen, some comments about the questions and a comment about her/his final grade. This helps him to pinpoint topics that he does not understand.
Thus, our extension allows us to combine the quiz creation capabilities of Moodle to promote active learning outside the classroom.

Acknowledgement

First of all, I would like to thank my master thesis supervisor, Professor Deville, for his help and advice.

Second, I also would like to thank Fédéric Fervaille and Olivier Roux for their patience when they helped me through this whole project.

Thanks to my parents André Mondry and Véronique Lamolle for their support.

Thanks to my girlfriend Laurie-Anne Penet to be always here to cheer me up and keep myself motivated.

Thanks to my sister Julie Mondry and her boyfriend André Paganin for their help in rereading this document.

Finally, thanks to my friends Xavier Planckaert, Nicolas Ooghe, Paul Vanhove, Sébastien Jacques and Florent Gos for their support.

Introduction

"If you tell me, I will listen. If you show me, I will see. But if you let me experience, I will learn."
Lao-Tse (5th century B.C.)

Since the birth of the first universities, lecturing in classrooms and auditorium was the dominant and one of the only ways to give information to students [1]. Still in our days, students mainly sit in classrooms to listen to their teachers or lecturers talking about different subjects and topics, then study all of this information for the exam to mainly forget it a few weeks after. This teaching method is known as **passive learning** where the student patiently sits, listen and does nothing more. The teacher is in charge of the whole learning process [2].

Since multiple years now, students are more deeply involved in their education. When the students are in charge of their learning process, when they are curious about different topics, when they learn by themselves and try to find new related topics, this is called **active learning**. It can be defined like this: *"Active learning is a process whereby students engage in activities, such as reading, writing, discussion, or problem solving that promote analysis, synthesis, and evaluation of class content. Cooperative learning, problem-based learning, and the use of case methods and simulations are some approaches that promote active learning"* [3].
Active learning is a term that is really large and regroups a lot of ideas. Every learning process has at least a part that is active but this part can vary in size. As soon as the student is involved in another way than listening, it is an active learning [4]. In classrooms, some example of simple active learning activities are: a pause for reflection after a new idea or concept was introduced, a "minute paper" to regroup every idea of the student about a subject before starting that new subject, or also a group discussion based on a reading, video, or problem.
Some more complex examples are: experiential learning where the teacher plans a visit of a place relevant for the topics to be studied; inquiry learning where the teacher asks the student to do work of research (emit hypotheses and derive conclusions after doing an observation work) themselves after introducing a new topic, a question, ... [5]. These simple and complex examples are a non-exhaustive list.

Studies have shown that active learning is more efficient in terms of scores and success rate. In fact, passive learners are 1.5 more likely to fail and active learners have a mean of success rate 6% higher than passive learners [6].

The UCL (Université Catholique de Louvain) has already implemented multiple activities to be able to bring a more active learning process to classrooms and at home. For example: Moodle, digital televoting, online assessment and podcasts [7].

Moodle is used in most of the courses of UCL. It is used to give to the course additional resources for the students like forums, quizzes, etc. Moodle has been declined in several versions [8]:

MoodleUCL[1], the main Moodle platform for UCL, **UCLine**[2], the Moodle platform for students in continuous training and finally **Student Corner**[3], the Moodle platform for UCL Mons.

Digital televoting tools allow students to answer anonymously and in real time to multiple-choice questions that the teacher asks to the students in the classroom. These tools can also be used in the other direction: students can ask questions directly to the teacher with their phones or tablets [9]. The tool that has been chosen by UCL is **Wooclap**[4].

Thanks to online assessments tools, students can take exams and undergo evaluations at home without the cost of photocopying, handle large pile of papers, manual correction, etc. The solution that has been selected by UCL is **Safe Exam Browser**[5].

Podcasts, short audio and/or video sequences that are broadcasted on the Internet and to which a user can subscribe, allow a student to listen a second time to the course to clarify misunderstood concepts. Sometimes, podcasts are short audio and/or video sequences that are additional content for the course but not mandatory to watch. It is in this way that it can be characterized as an active learning material. The students can choose to do more research on the course topic by watching these podcasts. It can also give to classroom sessions an interactive role where the teacher gives his course depending on the students' questions about the podcast [10]. There are two solutions chosen by UCL: **Podcast@UCL**[6] [7] and **Podicampus**[7] [8] principally for students in medicine studies.

Concerning the Podcasts, the UCL has chosen a new open source platform developed by the Université Libre de Bruxelles (ULB) called **EZcast**[8]. It is a complete podcasting solution that handle the whole workflow from recording videos in classrooms to broadcasting them on the Internet. It allows teachers to easily record their videos and upload them automatically on the Internet for the students.

The objective of this master thesis is to extend this new podcasting solution to make it even more active learning oriented. To do that, we have extended the video player of EZcast by establishing a connection between it and Moodle and being able to add Moodle quizzes to EZcast videos. Those will be seamlessly integrated in EZcast. Quizzes are also editable and removable by the teacher. For the moment, 3 types of questions are supported: Multiple Choice (with one or multiple possible answers) questions, True/False questions and finally, Short Answer questions. To carry out the communication between Moodle and EZcast, we will rely on the Moodle web services that allow us to retrieve specific user and quizzes data. The integration is entirely done by ourselves in the current code of EZcast developed by the ULB. Since there are a specific level of participation needed from the student and the answers will help the teacher to guide his course and insist on some specific topics, we believe that this solution will help to improve the current status of active learning in this podcasting solution.

The present document is divided in 4 chapters:

Chapter 1 explains the different software and important concepts that are addressed in this master thesis. We explain in more details EZcast, Moodle, the Moodle quizzes, the web services

[1] https://moodleucl.uclouvain.be/
[2] http://ucline-new.uclouvain.be/
[3] https://www.student-corner.be/
[4] https://www.wooclap.com/
[5] http://safeexambrowser.org/news_en.html
[6] https://podcast.uclouvain.be/
[7] http://podcast.mede.ucl.ac.be/
[8] http://ezcast.ulb.ac.be/

and finally we compare multiple existing podcasting solutions.

In chapter 2, we describe the problem addressed in this dissertation and the different parts that should be implemented to bring a solution to the initial problem.

In chapter 3, we describe the detailed solution implemented the solve the problem. We first explain the underlying operations of the web services introduced in chapter 1. Then, we show and explain diagrams that present different execution scenarios of the solution implemented. We then present a summary of the files that were added, created or heavily modified. Finally, we explain the detailed steps to deploy the solution on the existing infrastructure.

In chapter 4, we make some analysis on the developed solution and on the initial code of EZcast. We first explain the differences between what should have been implemented and what was really implemented. Then we discuss the encountered problems. Finally, we investigate on the EZcast initial code and how it could be refactored.

Finally, we present a conclusion to this master thesis with a summary of all that has been done.

Chapter 1

Background

In this chapter, we detail all the technologies used in this master thesis. We begin with a complete explanation of the podcasting solution EZcast and its video player Flowplayer. After that, we present the Moodle software and its important features. Then we describe the web services of Moodle that allows a user to exploit a lot functions of the API. Finally, we end this section with a comparison of existing podcasting solutions.

1.1 EZcast

In this section, we present the EZcast software suite and then explain its complete infrastructure.

1.1.1 Presentation

EZcast is a complete podcasting solution. It can record videos, edits them in post-production, upload them and finally display them in a video player named EZplayer. To be technically correct, EZcast only refers to the set of web interfaces accessible by a user. These web interfaces are: EZadmin, EZmanager and EZplayer. However, in this document, we use EZcast to refer to the whole infrastructure (that also contain EZrecorder, EZrendrer and the Repository) because no global name exists to define them.

This software suite is developed since 2011 by Univerité Libre de Bruxelles (ULB) in the Centre des Technologies au service de l'Enseignement de l'ULB (CTE) with the objective to create "*(...) un lecteur audiovisuel qui permet aux étudiants de chapitrer les vidéos, de mettre des signes temporels au sein des vidéos en donnant un titre, une description, des mots clés, ce qui leur permet de mettre un ensemble de signets sur la vidéo, de faire de la recherche, de trouver une information qui leur semble pertinente (...)*" (Nicolas Roland, chercheur CTE-ULB). [11]. It is distributed under LPGL v3 open-source license.

1.1.2 Infrastructure

In this section, we present the complete infrastructure of EZcast composed of EZadmin, EZmanager, EZplayer, EZrecorder, EZrenderer and the Repository. A diagram of that infrastructure can be seen on the figure 1.1 [12].

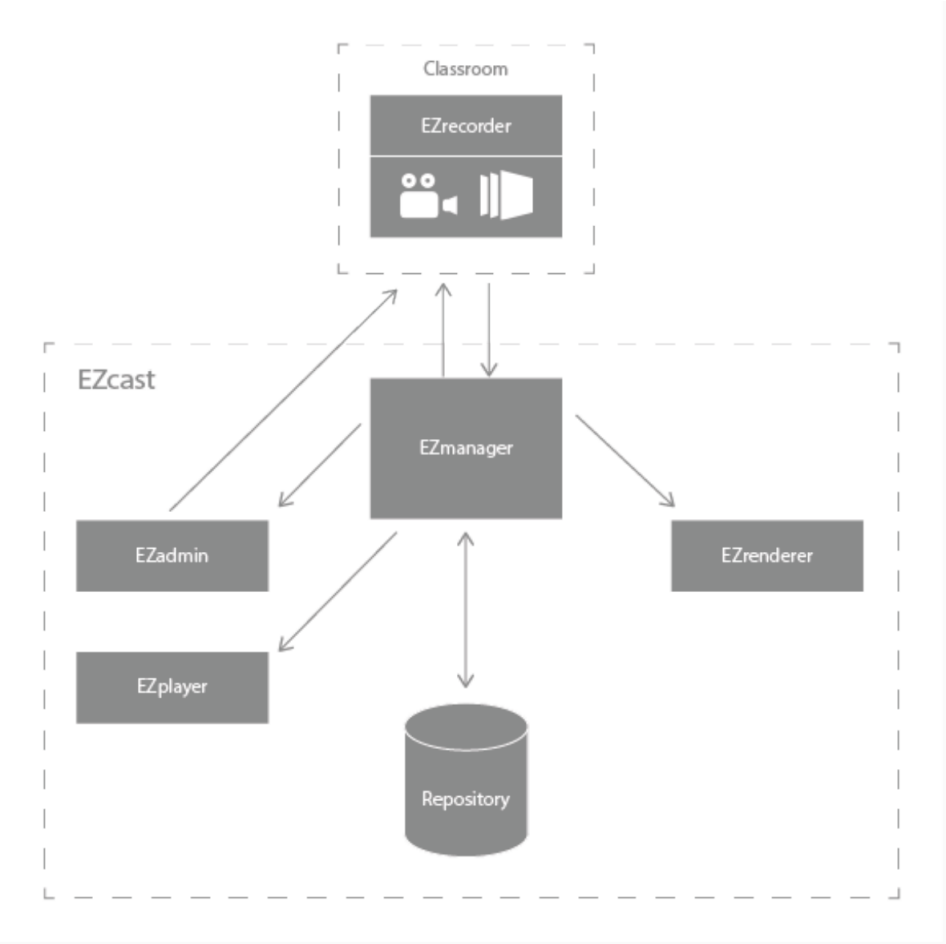


Figure 1.1: Diagram of the interactions between the different products of the EZcast infrastructure

EZadmin

EZadmin is the web interface by which users with administrator right create courses, users and associate courses to users. It also allows to administrate EZrecorder, EZmanager and EZplayer. Thanks to EZadmin, some statistics are calculated as the number of video files being processed, the use rate of the renderers or the number of views for each uploaded video files. It communicates with EZrecorder to update information of users and contains libraries for the management of the database.

EZmanager

EZmanager is the core of the system. It is located on the same device as EZadmin and EZplayer. It makes a connection between all parts of the infrastructure. This web interface is only available for course owners and administrators. EZmanager has several valuable features. First of all, it allows users to manage public and private albums associated to their courses and to manage video files associated to these albums. Second, users can upload video files which have not been recorded in class by means of EZrecorder. Third, EZmanager serves as communication between EZadmin, EZplayer and the Repository and, last but not the least, it exploits multiple security protocols to communicate with EZrenderer and EZrecorder: `SSH`, `RSYNC`, `HTTPS`. EZmanager contains libraries to manage video files and publish them on EZplayer for the students.

EZplayer

EZplayer is the web interface that displays the list of videos uploaded with EZmanager and offers the possibility for the users to watch them. It allows import, export and creation of official bookmarks for the teacher or non-officials (called personal) bookmarks for the students. This interface contains the enriched video player of EZcast that allows to watch videos uploaded by the course manager and to download them if the option is activated. For more information about the video player, please read the section 1.1.3. EZplayer is the interface concerned by this master thesis.

EZrecorder

EZrecorded is an application on its own. It allows a teacher to record himself with a video camera, the screen of its computer and the voices and sounds with microphones. These recordings are autonomous and automatic in rooms equipped with required hardware. This application also processes and makes some modification to the videos before transferring them to EZmanager over SSH/RSYNC.

EZrenderer

EZrenderer is also an application on its own but does not have any web interface. Every video recorded by the recorders (computers with an installation of EZrecorder in rooms equipped with required hardwares) or manually uploaded using EZmanager are sent here to be processed and modified. The modifications may include these different elements:

- Adding a title.
- Adding an end credit.
- Modifying the initial format to be readable by the video player.

Repository

It is used to store video files and is constructed as a tree structure. This tree structure has 4 levels:

- Level 1: Separates the public and private videos of each album by using 2 repositories for each course, one for the public videos and one for the private ones.
- Level 2: Holds metadata of the albums and a security token to be able to watch videos. It also contains assets (which are subdirectories) named after the date and hour of the submission. It is here that the official bookmarks are stored inside an XML file.
- Level 3: Holds metadata of the assets, a security token and multiple subdirectories containing the different video files (High-Definition/Standard-Definition, Slides/Video Camera/Slides + Video Camera, Original video file). It is also here that the quiz files are stored in an XML file.
- Level 4: It is on that level that the video files are stored. It also holds metadata for that file.

A diagram of the repository can be found on the figure 1.2 [13].

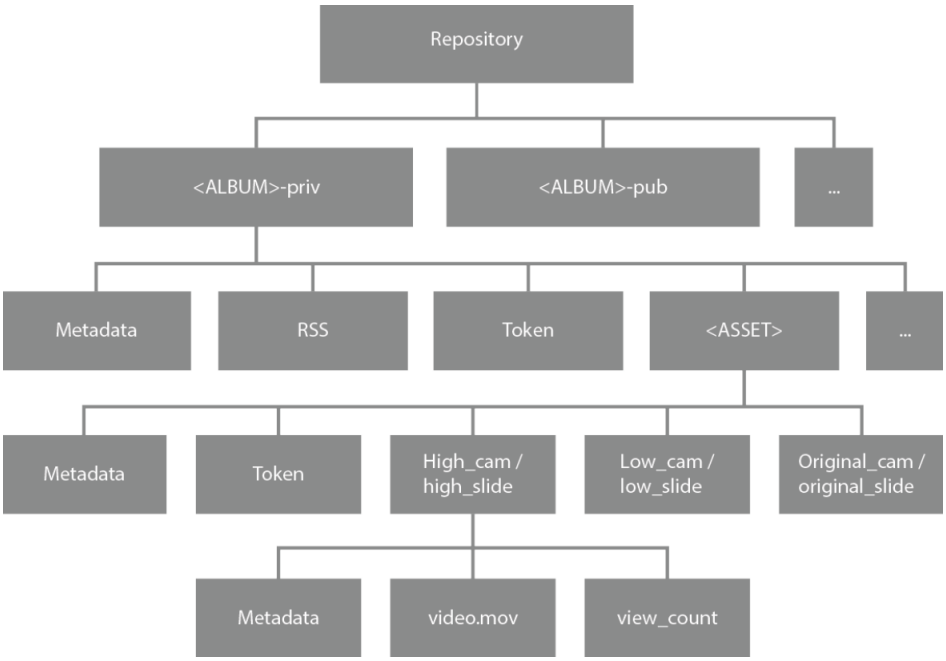


Figure 1.2: Diagram of the repository

1.1.3 Flowplayer

"*Flowplayer is an Open Source video player for your website. For site owners, developers, hobbyists, businesses and programmers.*" [14]. It is used by more than 450,000 websites around the world, for example IBM, Massachusetts Institute of Technology (MIT) or NASA websites.

This video player is open-source and customizable. It is for this reason that it has been chosen to be the video player of EZplayer. It has a modular interface. Indeed, every part of the player is a plugin, it is therefore easily customizable. The main drawback of this player is that only few plugins developed by the community are available, compared to other open-source video players (see Video.js in section 1.3.1).

In EZcast, the player has been modified and three different new functions were implemented: switching between video flows, bookmarks and threads.

Switching between video flows

This additional functionality allows the student to switch between two video flows (when both are available), one for the video of the teacher recorded by the video camera and one for the recording of the teacher's computer screen (that usually displays slides).

Bookmarks

Another interesting added functionality is the possibility to add **bookmarks** that interact with the video. Indeed, the teachers and the students can add markers at key points of the video to be able to quick jump to these moment by clicking to the corresponding bookmark. The bookmarks added by the teacher are called official bookmarks and those added by students are called personal bookmarks. Official bookmarks are the same for everyone but the personal bookmarks are independent for every user (and stored in a different location than the official ones). They are displayed in a column next to the video player. When a user clicks on a bookmark in that column, he is directly redirected to the correct time of that bookmark in the video.

Threads

Finally, the last interesting feature added is the **threads**. Users can add comments on the video and others can reply to these comments. These threads have a time stamp and when a video is played and arrives at that time, a notification is displayed to say that there is a thread for that moment of the video.

1.2 Moodle

1.2.1 Presentation

"Moodle is a learning platform designed to provide educators, administrators and learners with a single robust, secure and integrated system to create personalized learning environments" [15]. It is used by the Université Catholique de Louvain (UCL) since 2015 [16]. Moodle is free and open-source. It is distributed under General Public License (GPL). It is written in PHP and JavaScript. It uses an SQL database to store data related, for example, to the courses or to the quizzes. This learning management system has been created by Martin Dougiamas in 2002 and is currently used by more than 86,000 websites in 237 countries. There are a lot of add-ons and complementary modules available to extend the functionalities of the software. Those add-ons and modules are also open-source and developed by the very active community. A mobile application has also been developed and has most of the functionalities of the website because it is using its API.

Moodle stands for Modular Object Oriented Dynamic Learning Environment. This learning Environment allows teachers to create and manage courses. These courses are customizable with multiple **resources**. A resource is defined in the official Moodle documentation page as "*(...) an item that a teacher can use to support learning, such as a file or link. Moodle supports a range of resource types which teachers can add to their courses*" [17]. There are multiple types of resources: Book, File, Folder, IMS Content Package, Label, Page, and URL [17]. A course can also be customized by **activities**. An activity is defined as "*(...) a general name for a group of features in a Moodle course. Usually an activity is something that a student will do that interacts with other students and or the teacher.*" [18]. The main difference between an activity and a resource is that a resource is presented to the student by the teacher but there is no interaction between them. On the other side, an activity implies that the student interacts with that activity. There are multiple types of activities: Assignments, Chat, Choice, Database, Feedback, Forum, Glossary, Lesson, (LTI) External tool, Quiz, SCORM, Survey, Wiki [18]. Quizzes, which are studied in this document, are explained in more details in the next section.

1.2.2 Quiz

A quiz is a type of activity (explained in the section 1.2.1). A quiz activity allows to evaluate students using multiple types of questions. These types are: Calculated, Calculated multi-choice, Calculated simple, Drag and drop into text, Drag and drop markers, Drag and drop onto an image, Description, Essay, Matching, Embedded Answers (Cloze Test / Gap Fill), Multiple-choice, Short Answer, Numerical, Random short-answer matching, Select missing words, True/False [19].

A quiz can be available at any time or available during a defined period. It can also have a time limit to answer the quiz in a defined number of minutes once the quiz is started. The number of times a quiz can be redone is also customizable. The best setting to use with EZcast is the "Unlimited" option so that you can redo the test each time you see the video.
During the quiz creation, the teacher can select the number of questions per page. The best setting to use with EZcast is the "Unlimited" option so that it is easier and faster to retrieve each question. The order of the questions can also be random.

Every type of question can be corrected automatically except for the question type Essay because there is no unique correct answer. So, except for the type Essay, we could receive a feedback for each question and a global feedback for the whole quiz. This project only limits this capability to 3 question types : Multiple choice, True/False and Short Answer.

1.2.3 Web Services

A web service is a technology that allows applications (usually web-based applications) to communicate in a common language, regardless of the languages and technologies used to design and operate them internally. A web service can be defined by a term that "*(...) describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing which services are available. Used primarily as a means for businesses to communicate with each other and with clients, Web services allow organizations to communicate data without intimate knowledge of each other's IT systems behind the firewall.*" [20]

Moodle has an API (Application Programming Interface) that allows a user to write new add-ons and plugins that can be integrated in the software. Moodle enables the user to employ a majority of the functions of its API with its web services. Thanks to them, we can access the majority of the functions of Moodle from other websites or applications to write software that interacts with Moodle.
To use them, it is mandatory to define a new External Service in the Moodle site administration page. In this external service, you have to define every function of the API that has to be accessible remotely (aka the web services). Another parameter that is mandatory to use them is a token, which is retrievable thanks to a special web service for a remote identification. Moodle enables you to utilize 4 different protocols to communicate with: REST (returning JSON), REST (returning XML), SOAP and XML-RPC. Two more protocols will be implemented in the future: JSON and REST (returning JSONP).
A simple explanation on how they are used can be found in section 3.1.1.

1.3 Existing Solutions

In this section, we present some existing podcasting solutions. The first one, Pod, allows teachers to add quizzes for the students and let them answer questions while the video is being played, no other compared podcasting solution has the same functionality. The two others are Podcast@UCL and Podicampus, two solutions currently used at the UCL.

1.3.1 Pod

Presentation

Pod is a platform developed and deployed by the Université de Lille and promoted by the ESUP portail [21] which is a consortium that aims to provide software, courses and numerical resources to universities [22]. It is an easy-to-use podcast website designed to be used in research and teaching environments. It is used in multiple French universities like: Le Mans, Valenciennes, Lille, Pau, Avignon, Nice, Centrale Lille, etc. [23]. It has been developed in Python with the Django framework and use video.js as video player (see section 1.3.1). This solution does not support the recording of the videos in specially equipped classrooms.

It has functions similar to EZcast :

- It can have private and public videos.

- Its main functions are:
  - Editing videos.
  - Storing videos.
  - Encoding videos.
  - Sharing videos.
- There is a possibility to search for videos.
- Videos are sorted by courses.
- Support of multiple video qualities.

But it also has many differences [21]:

- All identified users can upload audio and video files.
- Users can add subtitles, comments or downloadable files to their videos.
- It has bookmarks like EZplayer but only the user that has uploaded the video can add them. So in terms of "EZcast vocabulary", videos can only have official bookmarks.
- Users can add images, HTML code, web pages, text files, PDF files or media (like YouTube videos, Moodle quizzes, etc.) on a video so they appear next to the video at the moment the owner of it wants to.
- Videos can be embedded in Moodle

As said in the differences, it already has a Moodle quiz integration but not the same as the one we tried to develop. Indeed, their integration is a web page with a quiz at a defined moment of the video. The video playback resumes when the whole quiz is submitted. In the version we have implemented, we were able to choose a time code for every question and only one question at a time appears at the defined moment of the video.

Video.js

"*Video.js is a web video player built from the ground up for an HTML5 world. It supports HTML5 and Flash video, as well as YouTube and Vimeo (through plugins). It supports video playback on desktops and mobile devices. This project was started mid-2010, and the player is now used on over 400,000 websites.*" [24]. Many well-known websites use video.js, for example IGN and The Guardians [25].

Video.js is open source and customizable. It is distributed under an Apache License Version 2.0. It has a modular interface. Every element of the player is a plugin, for example we can change the timeline by changing its plugin. It is easily installable using NPM [26]. The main advantage of this video player is its community, and therefore the number of available plugins. For instance, on the NPM website, when searching for "video.js", the number of results is 175, 1 package for video.js and 174 for its plugins [27]. Thanks to its community and its open-source design, this video player is actively maintained and under active development [28].

1.3.2 Podcast@UCL

As it was already introduced in the Introduction, Podcast@UCL is a solution that is currently used at UCL. This solution was introduced in 2009 in the framework of a project financed by the FDP[1] and promoted by Marcel Lebrun, a techno pedagogue, professor in education sciences and advisor at the Louvain Learning Lab of the UCL [29]. It was developed by **Taktik**[2] on the basis of their application **Defiris**.

The main idea of the project was not to record courses in classrooms but to allow teachers to upload any audio or video files to enrich courses (first on iCampus and now on Moodle).
Video and audio files were typically interviews, parts of documentaries, recorded scientific experiments, demonstrations, etc.

The teacher can submit files of any formats thanks to a web interface. Then the server converts the video in multiple format:

- MP4 HQ for computers.
- MP4 LQ for iPad and tablets.
- MP3 for audio files.

This podcast tool has a basic access management. Each video is associated to a token and every person that knows a token can access the video associated with it. This token can have an availability period with a start date and an expiration date [30].

To make the video accessible by others, teachers have a web interface available. This interface allows a teacher to regroup videos by albums. Videos' links can be shared or directly integrated to MoodleUCL with the link of the video (that includes the token).
There is no public interface for the students to broadcast or search for videos and, since there is no public interface, there is no enriched video player [31].

1.3.3 Podicampus

Podicampus is another podcasting tool used by UCL but intended solely for the students in the health sector or every member of the community of that sector (teaching assistants, nurse and nursing staff of university clinics of St. Luc and Mont-Godinne) [32].

This solution was introduced in 2010 and financed by the Health Sciences Sector of the UCL. The objective of this solution was different than Podcast@UCL. In this case, the objective was to alleviate the growing number of students in the health sector. Indeed the classrooms were too small for all the students and they had to find a solution to give everyone a chance to follow the course. The solution found was to record courses and give the links of the videos to the students. These videos are recorded by the RecordingBox (1 mac mini that record independently two different video streams: the screen of the teacher and the teacher himself. The sound comes from the second video stream. Initially there were only two classrooms equipped with that hardware but there are now 6 equipped classrooms and two mobile RecordingBox units. The system is also composed of the RecordingBox server. It receives the two video streams and generate the final video file with a jingle. All this process is done automatically. The video file is then synced with the back-end server that put the video online.

[1] Fond de Développement Pédagogique
[2] http://www.taktik.com/fr/

The RecordingBox system was developed by Impart[3]. The integration and web interface was developed by Philippe Meurrens, Head of the Audiovisual & Podicampus Center.

After this first step, it was decided in 2014 to extend this solution by providing students with a better web interface to broadcast and search videos. This interface includes a video player enriched with the possibility of adding personal notes to videos. These notes are not time-related so this is not like the bookmarks in EZcast and Pod. Teachers can restrict access to videos only to people of UCL. There is an album-like functionality. Indeed, there are four categories:

- Podicours: to show recorded courses, these courses are also grouped by the year of study (BAC1, BAC2, etc.).
- Podimed: formative videos for the students and the nursing staff.
- Podiform: videos used in the framework of specialisations or complementary formations.
- Podiconf: videos of medical conferences.

This was developed by Tapptic[4] [30, 33].

It is possible to submit video files that were not recorded by the RecordingBox but the file conversion has to be done manually. All the files can be shared on Moodle or another website with a link.

1.3.4 Comparison of the Functionalities

In table 1.1, we can find a summary of the existing podcasting solutions presented above. We can note that both EZcast and Pod have very similar functionalities.

EZcast stands out from the three other tools by its abilities to directly record podcasts. Indeed, EZcast has an interface called EZrecorder (see section 1.1.2) that allows teachers to record their courses, sends them automatically on the repository (see section 1.1.2) after they have been modified by EZrenderer (see section 1.1.2) to finally be available on EZplayer (see section 1.1.2). All of these exchanges are done automatically and significantly reduce the workload for the teachers and the technical staff. Another tool, Podicampus with its RecordingBox, has the same functionality, but what makes EZcast special is that it has an enriched video player and is open-source.

On two other functionalities, Pod stands out from the other three. For the enrichment of the video player, Pod has the most features. Indeed, in Pod, the owner of the video can add images, HTML code, web pages, text files, PDF files or media (like YouTube videos, Moodle quizzes, etc.) that appear next to the player at the time that the owner has chosen. This is not available in the other tools. EZcast can add bookmarks and threads, Podicampus can only add notes. It also has the option to add Moodle quizzes to EZcast, which is the functionality that is studied in this master thesis.

Comparatively, Pod seems like a very good alternative to EZcast but it was missing the important functionality of video recording. Moreover, EZcast is coded in PHP and JavaScript like Moodle. The utilization of EZcast is also a very good opportunity to collaborate with another important Belgian University, the Université Libre de Bruxelles.

[3] http://www.impart.nl/
[4] https://tapptic.com/

Comparison criterion \ Podcasting solution	EZcast	Pod	Podcast@UCL	Podicampus
Open-source	✓	✓	✗	✗
Public and private videos	✓	✓	✓(if token not publicly shared)	✓
Record videos	✓	✗	✗	✓
Edit videos	✓	✓	✓	✓
Convert external videos	✓	✓	✓	✗
Share videos	✓	✓	✓	✓
Public interface	✓	✓	✗	✓
Search videos	✓	✓	✗	✓
Albums	✓	✓	✓	✗(classification but no real albums)
Multiple video qualities	✓	✓	✓	✗
All users can upload videos	✗	✓	✗	✗
Enriched video player	✓(very complete)	✓(complete)	✗	✓(only notes)
Quizzes	✗	✓	✗	✗
Bookmarks	✓	✓	✗	✗
Videos integrated in Moodle	✓(with the link)	✓(with the link)	✓(directly)	✓(with the link)

Table 1.1: Comparison of the different podcasting solutions explained above

Chapter 2

The Problem

In this chapter, we present the technical specifications of the new functionality for the video player of EZcast developed for this master thesis. We begin with the definition of the given problem. Afterwards we describe the objective of the new functionality and the constraint we had to respect. Then we list every actor in the system. And finally we describe every aspect of the new functionality from the student's point of view and then from the teacher's point of view and also the non-functional requirements.

2.1 The Given Problem

Nowadays, learning materials can be given in two different ways: orally in the classroom in real-time, or online with course materials like videos, interesting PDFs and web pages, etc. The second way allows the student to go further with his reflection and personal learning process. As we have already seen in the Introduction, there are two different learning processes: passive learning and active learning. The first way seen above belongs to the passive learning when the students sit in class and do nothing else than listening to the course. The second way belongs to the active learning where the student is more involved in his own education.

The UCL already had solutions to give to students extra learning materials in a video format (podcasts) : Podcast@UCL (see section 1.3.2) and Podicampus (see section 1.3.3) but with no further interactions. The new platform that has been chosen by UCL to provide these podcasts, EZcast, has more features to facilitate this active learning process with the shareable personal bookmarks and the threads used by the students to communicate, share their knowledge and think together about a common problem.

The given problem is the following : how can we go even further in this active learning process and how can we do that in EZcast ?
We have decided to add, in EZcast, quizzes from Moodle by using its web service. This way, there is an interaction between the teacher and the students. This interaction where the student has to reflect on something is one of the activities of the active learning process [5].

This solution also solves another problem with podcasts. With the use of courses taught using videos, teachers can no longer have feedback as they might have in class. Having quizzes in EZcast allows the teacher to have that feedback, identify parts of the course that is problematic and organize what is called "Flipped classrooms". Flipped classrooms can be defined by the *"(...) reversal of traditional teaching where students gain first exposure to new material outside of class, usually via reading or lecture videos, and then class time is used to do the harder work of assimilating that knowledge through strategies such as problem-solving, discussion or debates"* [34]. By using the response of his students, the teacher can feed a conversation, fuel a debate or adapt

his course. Flipped classrooms also belong to the active learning process.

2.2 Objective

The objective of this master thesis is to create an extension of the ULB's EZcast software. This extension enriches the video player of EZplayer with a new feature. The proposed extension would be the addition of quizzes (called quiz on Moodle, without the ending "z") from Moodle at key moments of the video, chosen by the teacher. It partially works like the bookmarks and partially like the threads (for bookmarks and threads, see section 1.1.3) .
Questions are displayed in a column on the right of the video player and are clickable, like bookmarks. These questions appear on the video in the same fashion as the threads notification. Indeed, thread notifications appear in front of the video player at the defined time. The idea is to pause the video each time we encounter the time stamp of a question, and display it in front of the video player.

The new functionality has to be coded in PHP and JavaScript to allow application maintenance and development continuity with the system currently in place. Indeed, as Moodle and EZcast are already developed in PHP and Javascript, it makes more sense to reuse these languages.
The new feature must be developed in a way that it is easy of use and integrated seamlessly into the current version of EZcast.

2.3 Actors of the system

This system has four different actors:

- Anonymous users: They can access to public videos of EZplayer but they are not registered on EZcast, nor on Moodle.
- Courses manager / Teachers: They manage the course in Moodle and EZcast. They are the people who create quizzes and publish them on Moodle.
- Students who are registered on Moodle and enrolled to the Moodle course: They are registered on EZcast, Moodle and enrolled in the course of the teacher that has created the quiz.
- Students who are registered on Moodle but are not enrolled to the Moodle course: They are registered on EZcast, Moodle but not enrolled in the course of the teacher that has created the quiz.

2.4 Functional Requirements

In this section, we describe the functionalities that the developed solution should have. We group these functionalities in three groups : functionalities for anonymous users and students not enrolled in the course with a quiz, functionalities for students registered on Moodle and enrolled to the Moodle course and finally, functionalities for the teacher or course manager.

2.4.1 For anonymous users and students in Moodle but not enrolled in the course with a quiz

These people can see the videos but are not able to use the new functionality. The quizzes are not going to appear on videos. The explanation for this restriction is simple. Moodle has two activities that can be used to evaluate users: Quiz and Lesson. Lesson works for everyone (even

anonymous users) and Quiz only works for authenticated people. Unfortunately, Lesson functions are not yet implemented to use them with Moodle web services.

2.4.2 For students that are registered on Moodle and enrolled to the Moodle course

Answer to a quiz

The student logs in EZcast, chooses a course, chooses a video in which the teacher has added a Moodle quiz. If there is any quiz available for the video, a pop-up is shown to the user to know if he wants to answer the quiz. If he says yes, the quiz starts.

Answer to a Question

If he has chosen to answer to the quiz, the student answers to the question each time the current time of the video matches the time code of a question that the teacher has defined. Each time the current time of a video hits the time code the teacher has defined, the video is paused and the question is displayed to the student in one of the two video flows. At that point, the student has to answer the question. Three types of questions are treated in this master thesis: Multiple choice, Short Answer, True/False.

Receive a feedback

If he has chosen to answer to the quiz, the student receives the automatic feedback from Moodle depending on which option the teacher has chosen:

- Feedback at the end of the video: Only one feedback at the end of the video. The global result of the quiz and the results of each question are displayed at the end of the video, these results and feedback are automatically calculated and generated.
- Feedback after each question: After each question, the result and the feedback are calculated and generated automatically and shown to the user. He has to click on a "Next" button to continue the quiz and the video.
- Repeated feedback: The question is being shown repeatedly until the right answer is selected.

The feedback at the end of the video and the feedback after each question can be combined and used with the same quiz: one feedback after each question and one feedback at the end of the video.

Display the list of questions

If he has chosen to answer to the quiz, the student can see on the right of the video the list of questions he has to answer to complete its quiz. Questions already answered and questions not answered should appear with different colors. When clicking on the question, it should jump to the time code of that question (behavior similar to the bookmarks).

2.4.3 For the teacher or the course manager

Select a course

During a video, the teacher or course manager selects the form to add a new quiz and selects a course between all the courses he manages or teaches in a drop-down list.

Select a quiz

During a video, after selecting the form and the course that contains the quiz, the course manager or teacher selects a quiz from those he created for this course on Moodle in a drop-down list.

Encode the time code of the video for each question

During a video, after selecting the form, the course and the quiz, the list of the questions for that quiz are displayed. The teacher or course manager can go through the video timeline and encode the time code for each question.

Select the Feedback Option

During a video, after selecting the form, the course and the quiz and the time code of each question, the course manager or teacher are able to select the type of feedback the student will receive after submitting his quiz. The different options available are described on the section 2.4.2.

Edit a Quiz

Following the same reasoning of the bookmarks, the quiz and its questions appear in a section to the right of the video player. A button is available to enter in the edit mode. The time code and the type of feedback are directly editable in that right column. This behavior is similar to the bookmarks. A quiz has to be modified on its whole even if it is to change the time code, it is not possible to edit only a question. To edit one question, you have to edit the whole quiz and only change the time code for that question.

Delete a Quiz

Similarly to the bookmarks, the quiz and its questions appear in a section to the right of the video player. A button is available to enter in deleting mode. When selected, a confirmation message pops-up and if the confirm button is pushed, the quiz is deleted from the column and from the xml file.

2.5 Non-functional Requirements

In this section, we develop two non-functional requirements that we need to respect in order to keep EZcast simple and we code our new functionality seamlessly in the current implementation.

Ergonomics

EZcast has been created to be really easy to use and thought to be really intuitive and optimized at a pedagogical level. We have to keep this in mind while implementing the new feature so it does not alter the current ergonomics.

Future Development

EZcast is the new podcast platform of the UCL. As Moodle, it will obviously evolve and be modified by the technical staff of the UCL. To facilitate their future work, we have to keep the current coding fashion used in EZcast. This way, every future refactorating and modification will not be a problem since the code is not going to behave differently than the current one.

Chapter 3

Development of a Solution

In this chapter, we first describe in detail how Moodle's web services work with the data structures to send and the response we receive. In a second moment we see sequence diagrams of some scenarios using the new developed feature, from the creation of the quiz by the teacher to the submission of the answers by the students. Then we list the different files that were added, created of modified for the purpose of this project. After that, we explain how the quizzes are saved in the repository. Finally, we explain the detailed steps to deploy the solution on the existing software.

3.1 Web Services

In this section we see the technical details of the web services (already introduced in section 1.2.3) used in the external service "EZplayerQuiz" created for this dissertation so we can interact with the Moodle server. We retrieve JSON files using the REST service of Moodle located at "`<Moodle URL>/webservice/rest/server.php`". These web services allow us to execute multiple actions, for example we can retrieve a token, retrieve quizzes and answer them.

3.1.1 Using the Moodle Web Services

To introduce the web services section, we first have to explain how web services work and why they are used for.

When a user connects on the Moodle website, he sends instructions to Moodle such as "Give me the list of my courses" or "Start this quiz for me". Then the actions are processed by the Moodle infrastructure that interacts with its database to give responses and perform the required actions. This simple process is represented in figure 3.1.

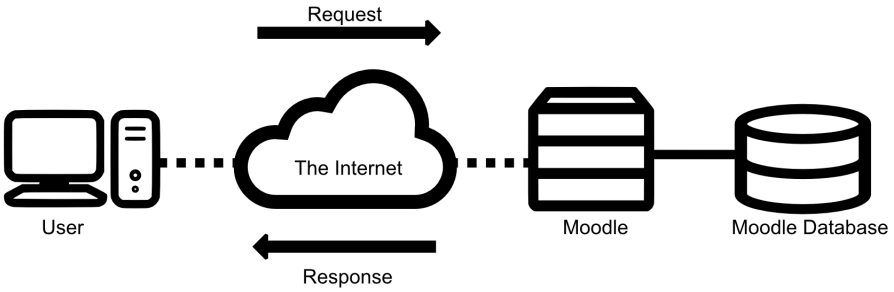


Figure 3.1: Simple Request to Moodle

Moodle offers another way to interact with its infrastructure. Instead of directly connecting

to Moodle over the Internet, the user first connects through an intermediary. In our case this intermediary is EZcast. All the required instructions to do a specific task are performed by EZcast. It handles the requests and process the responses to retrieve data or present them to the final user in a specific way to achieve a specific task, in our case, answer to quizzes. This more complex process is represented in figure 3.2.

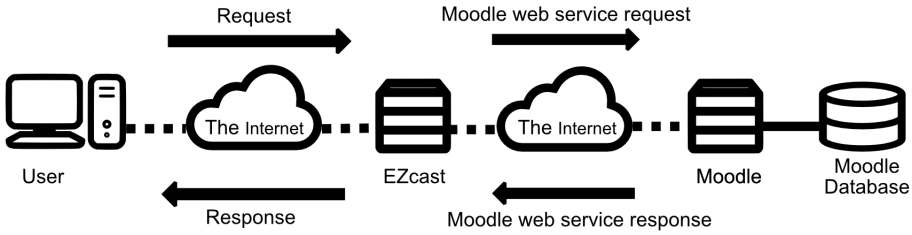


Figure 3.2: Requests to Moodle by Using Web Services

Of course this is just an example. For instance, in our EZcast case, the first exchange, that we have between the user and EZcast is done over the Internet. But when the Moodle Mobile app is used, this exchange is executed directly between the user and his phone. The rest of the diagram remains the same.

3.1.2 Receive a Token

To be able to use the web services the most important thing to have is a token. Without a token, we are not able to use any web services because we have to transmit it every time we make a request to Moodle.
To retrieve a token, we have to send a request to `"<Moodle URL>/login/token.php"` and send 3 different data: the Moodle username of the user, the Moodle password of the user and the external service you want to use. The data structure sent is presented in listing 1.

```
{
    username: 'Username',
    password: 'Password',
    service: 'EZplayerQuiz'
}
```

Listing 1: Data structure for identification

The response to this request is the token (in a JSON format) generated for this external service for this user so he can use every web services included in that external service. We have to mention that users must have the "moodle/create:token" capability to be able to generate a token. For some security reason, administrators cannot have this capability so they cannot generate a token automatically with the above request [35]. For those, it is mandatory to generate a token manually by navigating to the Moodle settings (***Settings*** → ***Site Administration*** → ***Plugins*** → ***Web services*** → ***Manage***).

3.1.3 Getting the Courses

To add a new quiz to a video, we first have to retrieve the courses, to do that we use two functions: the first one to retrieve the user id, the second one to retrieve the courses. These functions are used by the teacher to add a new quiz inside EZcast and to populate drop down lists in the form to add a new quiz.

Retrieve the User ID

To retrieve the user id we have to use the function "`core_webservice_get_site_info`". We send it with the token and ask to receive it in a JSON format. One of the response fields is the user id. The data structure to send is presented in listing 2.

```
{
    wstoken: 'the token',
    wsfunction: 'core_webservice_get_site_info',
    moodlewsrestformat: 'json'
}
```

Listing 2: Data structure to get the user id

Retrieve the Courses

To retrieve the courses, we have to send the user id together with the token and use the function "`core_enrol_get_users_courses`". The data structure to send is presented in listing 3.

```
{
    wstoken: 'the token',
    wsfunction: 'core_enrol_get_users_courses',
    moodlewsrestformat: 'json',
    userid: 'the user id'
}
```

Listing 3: Data structure to get the courses

The response is in a JSON format and is an array of all the courses the user manages or is enrolled in. All courses have multiple fields like the name or the course id.

3.1.4 Getting the Quizzes

After retrieving the course (and their ids), we have to retrieve the quizzes. To do so, we use the function "`mod_quiz_get_quizzes_by_courses`" and send the token and the course ids (written like an array) to have all the quizzes of these courses. If the course ids are not given, the function returns all the quizzes accessible to the user independently of the courses. The data structure to send (with course ids) is presented in listing 4.

```
{
    wstoken: 'the token',
    wsfunction: 'mod_quiz_get_quizzes_by_courses',
    moodlewsrestformat: 'json',
    userid[0] = id0,
    userid[1] = id1,
    ...
}
```

Listing 4: Data structure to get the quizzes

The response is in a JSON format and is an array of all the quizzes of the course ids sent. All quizzes have multiple fields like the name or the quiz id.

3.1.5 Begin a Quiz

Once a quiz is selected, to begin a quiz, we have to start it and retrieve the quiz data. To do that, we use two functions: the first to start the quiz and the second to retrieve the quiz data:

Start the quiz

To start a quiz, we use the quiz id and the token. With those we can send an extra parameter called "`forcenew`" with value 0 or 1. If the value is 1 and a quiz attempt has already been started, it is stopped and a new quiz attempt starts. The function used to start the quiz is "`mod_quiz_start_attempt`". The data structure to send is presented in listing 5.

```
{
    wstoken: 'the token',
    wsfunction: 'mod_quiz_start_attempt',
    moodlewsrestformat: 'json',
    quizid: 'the quiz id',
    forcenew: 1
}
```

Listing 5: Data structure to start a quiz

The response, which is in a JSON format, contains a lot fields. The most important and the one we are using is the attempt id. It is used to retrieve the quiz attempt data. Indeed, with the function to start the quiz we do not receive any information about the questions of that quiz.

Retrieve the Quiz Data

As explained in the previous section, we do not receive any information about the questions when we start an attempt. To receive them we have to send the attempt id and the current page of the quiz. Indeed, a quiz may be composed of multiple pages (and multiple questions can be on one page). The function to use is "`mod_quiz_get_attempt_data`" and he data structure to send is presented in listing 6.

```
{
    wstoken: 'the token',
    wsfunction: 'mod_quiz_get_attempt_data',
    moodlewsrestformat: 'json',
    attemptid: 'the attempt id',
    page: 0
}
```

Listing 6: Data structure to receive the attempt data

The quiz begins at page 0. In the response there is an interesting field called nextpage. Its value is obviously the next page, but when the value equals "-1", it means that there is no more page in that quiz. The response in JSON format also contains the list of questions of that page. For each question there are multiple interesting fields, the page of the question, the type of the question ("`multichoice`", "`truefalse`", etc.), the `slot` which is similar to an id for that question and the most important field: `html`. The `html` contains an HTML code with the question with the layout and the values of the responses to send back to Moodle. These values are explained in the next section.

3.1.6 Answer a Question

To answer a quiz we need 4 values, they are all in the `html` mentioned above. The first two are the name and value of the input tag of the div tag "`formulation`" field. It is called the sequence check. The other two fields are the name and value of the input tag of the div tag "`answer`". It is called the input. They are needed to identify the question and the answer selected. The function to use is "`mod_quiz_process_attempt`". To send the answer we need to send the token, the function name, the attempt id of the quiz and the 4 values to identify the question and the answer. The data structure to send is presented in listing 7.

```
{
    wstoken: 'the token',
    wsfunction: 'mod_quiz_process_attempt',
    moodlewsrestformat: 'json',
    attemptid: 'the attempt id',
    page: 0 ,
    data:
    {
        {
            name: 'sequencecheck name',
            value: 'sequencecheck value'
        },
        {
            name: 'input name',
            value: 'input value'
        }
    },
    finishattempt: 0
}
```

Listing 7: Data structure to answer a question

By extending the data field with new names and values for the sequence check and the input, you can submit multiple answers.

3.1.7 Stop a quiz

To stop a quiz, we use the same function as the one employed to submit an answer (seen in the section above) but with the "`finishattempt`" field with a value of 1. Answers can still be sent in this request but the quiz is submitted right after.

3.1.8 Get a feedback

To receive a feedback we first need to get the final grade, then retrieve the feedback corresponding to that grade.

Get the final grade

To get the final grade, we first have to finish the current attempt. Once it is finished, we use the function "`mod_quiz_get_attempt_review`" and send the `attemptid` and the `page` number. The data structure to send is presented in listing 8.

In the response, we receive the final `grade`. If we send a page number equal to -1, we also receive the correction for all questions of all pages that are useful for the feedback at the end of the quiz seen in section 2.4.2.

```
{
    wstoken: 'the token',
    wsfunction: 'mod_quiz_get_attempt_review',
    attemptid: 'the attempt id',
    page: -1
}
```

Listing 8: Data structure to receive a grade for the attempt

Get the General Feedback

The general feedback is common to all the students. Thus, to receive it, we need to send to Moodle a `grade` and a `quiz_id` to receive the correct feedback for that `grade` and `quiz_id`. The function used is "`mod_quiz_get_quiz_feedback_for_grade`" and the data structure to send is presented in listing 9.

```
{
    wstoken: 'the token',
    wsfunction: 'mod_quiz_get_quiz_feedback_for_grade',
    quizid: 'the quiz id',
    grade: int
}
```

Listing 9: Data structure to receive a feedback for grade

In the response, we should receive the feedback of that quiz for the grade sent.

3.2 Functional description of Moodle quizzes in EZcast

In this section we present the classic scenario for adding and answering a quiz by using both EZplayer and Moodle. The communication with Moodle is made by using the web services presented in the section 3.1. The three other actors used in these scenarios are the Student, the Teacher and the Repository. We gradually see the steps to, firstly, retrieve the Moodle token, then add a quiz to a video. After that we see how we can answer this quiz and finally how to submit it to Moodle. We also present an alternative scenario presenting the case where a user quits the asset page where the video is displayed before answering all the questions. Finally, we describe the process to receive a feedback for the submitted quiz.

3.2.1 Retrieve Moodle Token

The first step is to receive a Moodle token. The data structure to send to Moodle has been explained in the section 3.1.2. This step has been integrated to the initial EZplayer connection page. The Moodle token is then added to the PHP `$_SESSION` array. The process to retrieve this token is presented in the figure 3.3.

3.2.2 Add a new quiz

A quiz can only be added by a teacher or a course manager but, here, we use the term "Teacher" for these two roles. To avoid lots of processing and loading time, the teacher needs to click on the "add quiz" button to load resources to add a new quiz. After clicking, EZplayer first loads every course, quiz and questions into an array that is used to access every information needed to add a new quiz. It begins by retrieving the `user_id` with the EZplayer function `get_site_info(token)`,

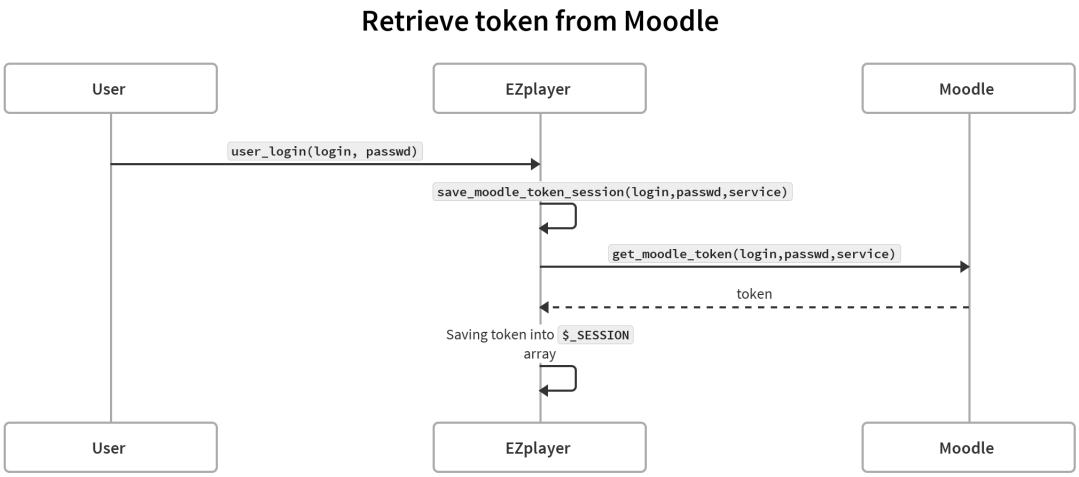


Figure 3.3: Moodle and EZcast token retrieval

then the `courses` with EZplayer function `get_user_courses(token,user_id)`. After that, for every `course` we retrieve all quizzes available of the teacher. There is only one way to retrieve all the questions for a quiz: start a new attempt of every quiz with the function `start_quiz(token,attempt_id,page)`, loop through all pages of the quiz, save every question of each page and finally finish the quiz without sending any answer with the function `stop_quiz(attempt_id)`. We have to do this for every quiz (that we get with `get_course_quizzes(token,course_id)`. With this array it is easier to navigate through all the courses and quizzes instantly. This process is detailed in the figure 3.4. The web services used in this scenario are :

- `core_webservice_get_site_info`
- `core_enrol_get_users_courses`
- `mod_quiz_get_quizzes_by_courses`
- `mod_quiz_start_attempt`
- `mod_quiz_get_attempt_data`
- `mod_quiz_process_attempt`

After retrieving this array, the teacher needs to choose the course and the quiz. After the selection, the questions of that quiz then appear below and the teacher has to choose the time stamp for every question. Then he has to choose if he wants to receive a feedback from Moodle when the quiz is submitted. After clicking on the validation button, the form is validated and all the useful information is saved in an XML file as we see in section 3.4. This process is detailed in the figure 3.5.

3.2.3 Starting a quiz attempt

A quiz can be answered by both a student and a teacher. In this section, the presented scenario to start a new quiz is when a student answers it. There are four actors in this scenario: Student, EZplayer, Moodle and the Repository.
This scenario begins when a student navigates through the asset page. The column next to the video player holds the list of the different questions for that quiz, so, when the web page is being loaded, the function `quiz_list_update()` is being called. This function then calls

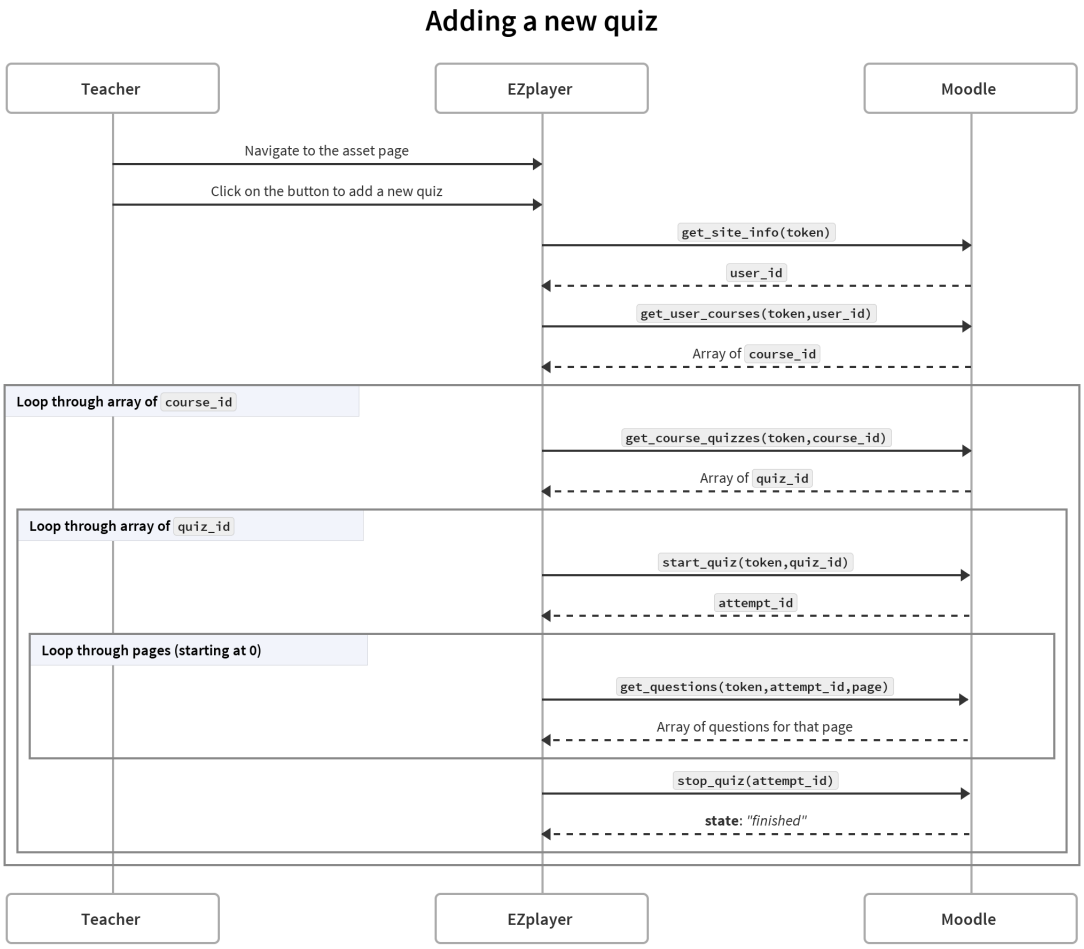


Figure 3.4: Add a new quiz - Part 1

`quiz_asset_question_list_get(album,asset)` that retrieves all the questions saved in the XML file.
The second part of this scenario is, with the information stored in the XML file, to retrieve all the questions from Moodle to be able to answer them if the Student wants to start the Moodle quiz. After reading the XML file, if there is a quiz available, a pop-up is shown to the Student to know if he wants to begin it. If he says yes, we first start a quiz with the function `start_quiz(token,quiz_id)`, with the previous function we receive an `attempt_id`, this id let us retrieve every question in a loop with the function `get_question(token, attempt_id)`. These questions are stored in a JavaScript data structure. Retrieving all the questions in one time improves the global performance because there is no loading time when a question needs to be displayed. This process is detailed in the figure 3.6. The web services used in this scenario are :

- `mod_quiz_start_attempt`
- `mod_quiz_get_attempt_data`
- `mod_quiz_process_attempt`

3.2.4 Answer a quiz

This scenario begins right after the scenario to start a quiz (section 3.2.3) when the Student has decided to answer the quiz. Flowplayer (see section 1.1.3) has an event called `timeupdate`.

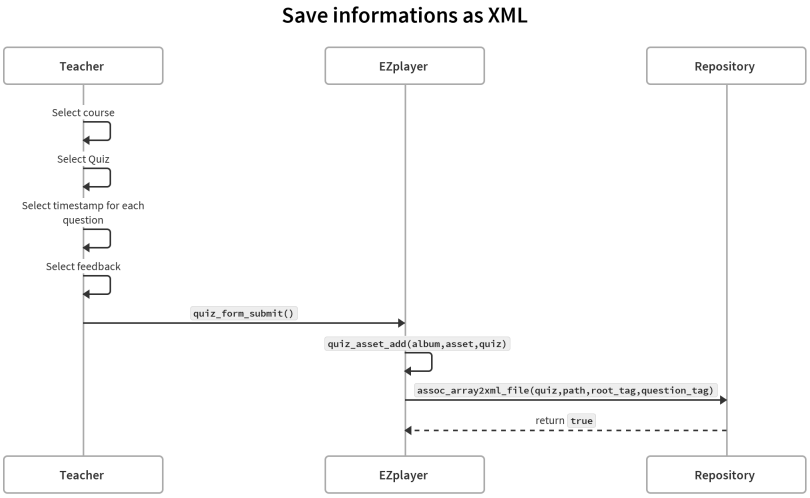


Figure 3.5: Add a new quiz - Part 2

It is triggered every second when a video is played. When the current time of the video equals one of the time stamp of the questions, the video is paused thanks to the function `player_video_play_toggle()`, the video player is paused and the question box (a pop-up with the question) is shown. The student then answers the question and send it to EZplayer with `submit_question(index,attempt_id)`, this question is marked as `done` in the JavaScript data structure so that it is not displayed again if the time stamp matches the current time again. Every time a question is answered, EZplayer checks if all the questions are already answered (all the questions marked as `done`). To send the question to Moodle, we use the function `process_attempt_question(attempt_id,<data of the answer>,finishattempt)`. The `<data of the answer>` part of the function definition is composed of four fields: the `name` and `value` of the input selected, and also the `name` and `value` of a sequence check that identifies the attempt. If all the questions have been answered, `finishattempt` is equal to 1 and the quiz is ended. Else `finishattempt` is equal to 0 and the quiz continues.
After every question, the question box is hidden, and the video is started again thanks to the function `player_video_play_toggle()`. This process is detailed in figure 3.7. The web service used in this scenario is: `mod_quiz_process_attempt`.

3.2.5 Quit an unfinished quiz

The previous scenario in section 3.2.4 happens when everything works as it should. But sometimes, Internet problems or other inconveniences may happen and the asset may be quit voluntarily or not. In these special cases, when we quit the asset page, the browser special events `onunload` or `onbeforeupload` depending on the browser (for example, Google Chrome recognizes the event `onbeforeupload` but not the other one) are triggered and execute the function `process_attempt_question(attempt_id,null,1)` seen in the section 3.2.4 and with `finishattempt` that is equal to 1. With this alternative scenario, the quiz is closed no matter if the student answers all the questions or not. This process is detailed in figure 3.8. The web service used in this scenario is : `mod_quiz_process_attempt`.

3.2.6 Receive a feedback

After submitting a quiz (see section 3.2.4), if the option was activated by the teacher, the student receives a feedback about its grade, the answers he chose and a general comment on the question. First EZplayer sends a request to Moodle to receive the review of the submitted attempt. As a response, it receives an array of reviews, one for each question and a grade for the whole quiz.

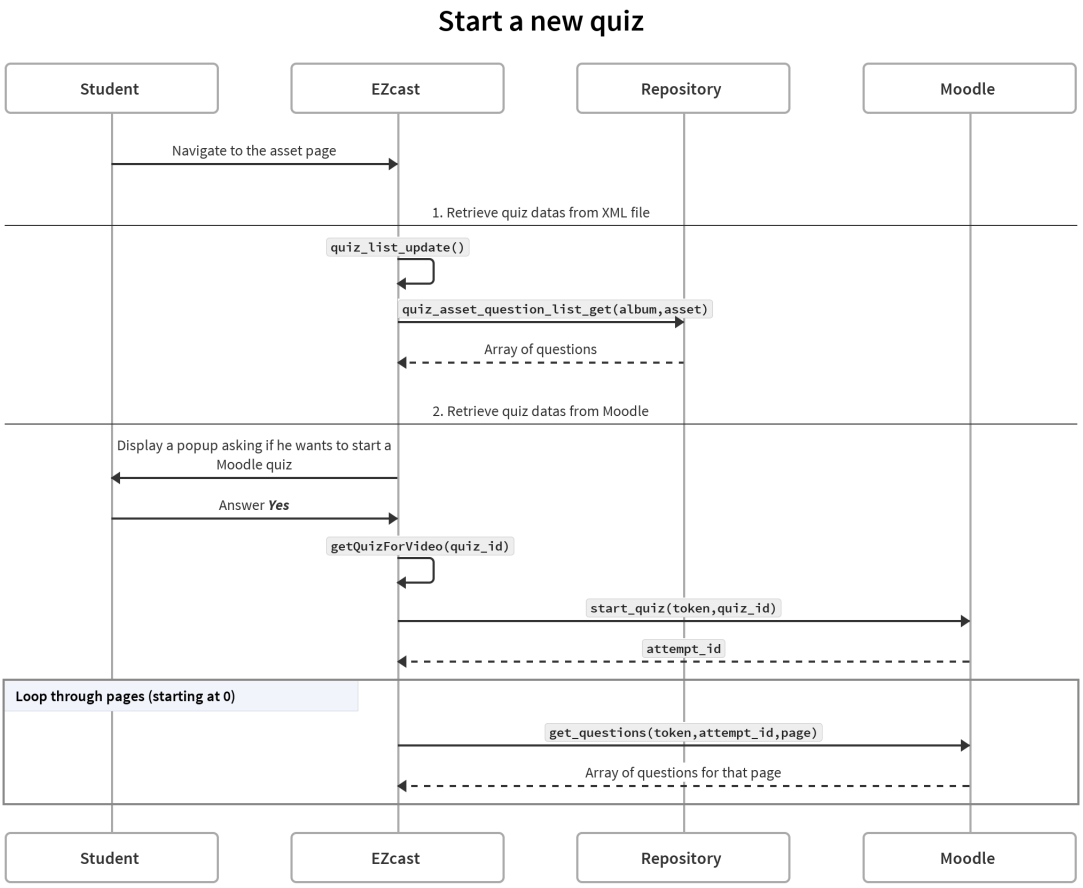


Figure 3.6: Start a quiz attempt

To have a feedback about the grade, EZplayer sends a new request to Moodle with the quiz ID and the received grade. The response received contains a text, written by the teacher, with a general comment about this grade. This comment is the same for every student that received this grade. This process is detailed in figure 3.9. The web services used in this scenario are :

- `mod_quiz_get_attempt_review`
- `mod_quiz_get_quiz_feedback_for_grade`

3.3 Implementation of Moodle quizzes in EZcast

To be able to do this master thesis, multiple files needed to be modified and others needed to be created in a way that respects the coding fashion already imposed by the previous code. Other external libraries were also added. The files that underwent lots of modifications, that were created or that were added are explained below.

JavaScript

The files below are JavaScript files that were created or modified:

- **`lib_quiz.js` (created)** : This file is mainly a library to call web services from Moodle. It has Ajax functions that query Moodle web services without the need to reload the page. Ajax is a jQuery functions that is used to perform asynchronous HTTP requests [36] and jQuery is a JavaScript API to manipulate HTML documents, handle events, animate

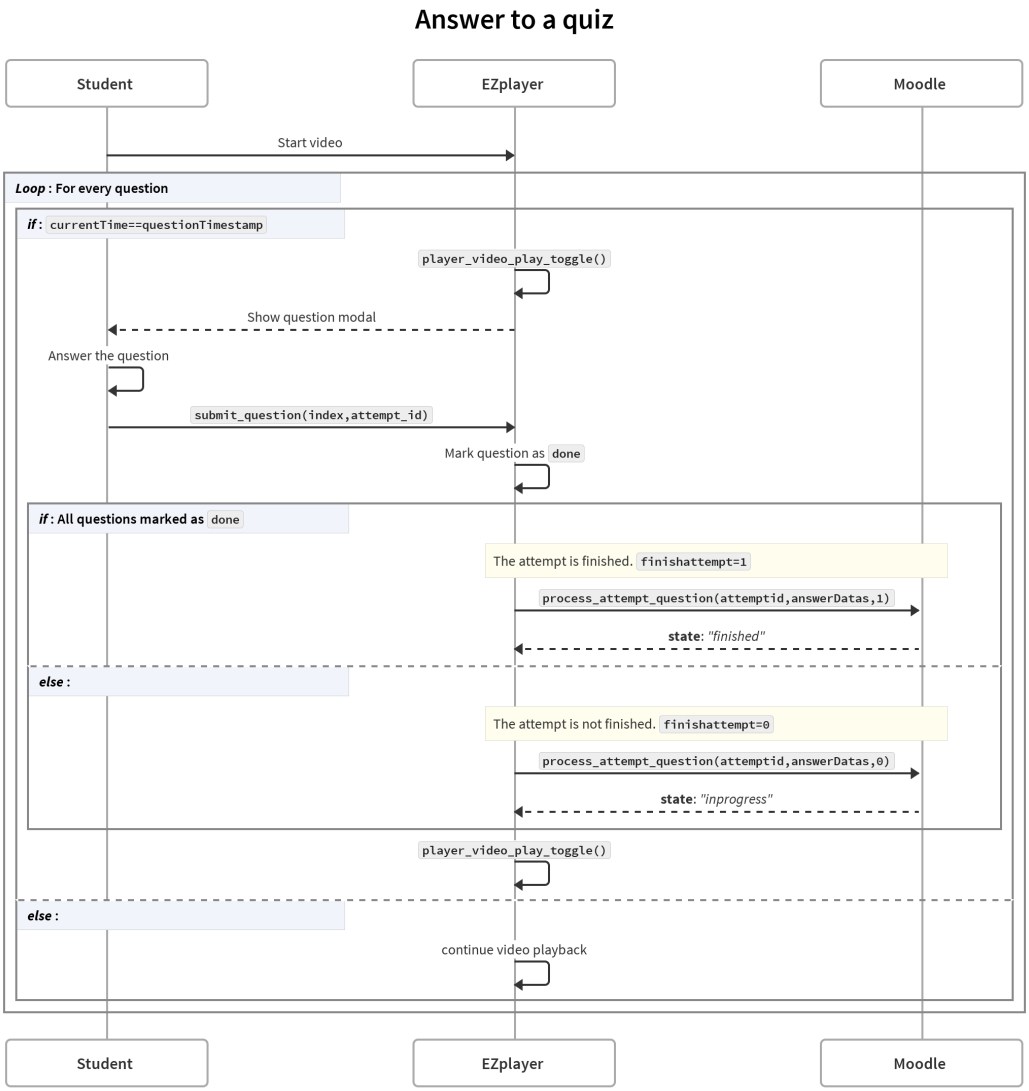


Figure 3.7: Answer questions of a quiz attempt

content and do Ajax requests [37]. It is used by the `lib_player.js` file to retrieve the quiz associated to the video and the questions of that quiz. It is also used by `div_quiz_form.php` (see below) to submit the quiz add form to `web_index.php` (see below).

- **`lib_player.js` (modified)** : This file was already existing but was modified so it can be used with the quizzes. These changes include the start of the quiz, the question retrieval (using `lib_quiz.js`), the verification each second for the time stamp of each question. The functions to display the questions, answer them, finish the quiz and receive the feedback are also implemented here. We have also added multiple checks to do actions when the video player is closed prematurely. These actions can save responses and finish a quiz before quitting the window. This file also interacts with the css attributes (defined in `ezplayer_style_v2.css`) of `div_left_details.php` to be able to display the questions and the feedback. This PHP file mainly displays the video player and the form to add a quiz, a bookmark or a thread.

PHP

The files below are PHP files that were created, added or heavily modified:

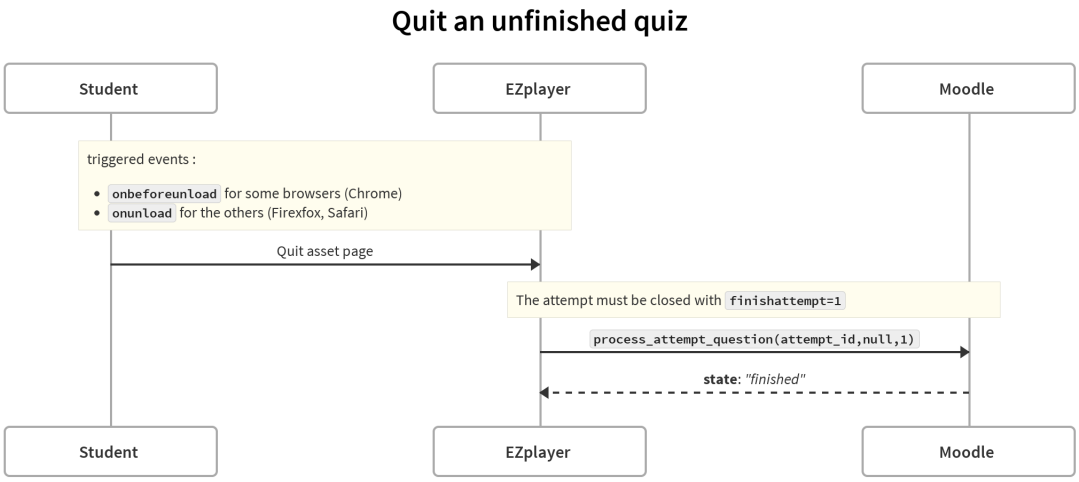


Figure 3.8: Quit an unfinished quiz

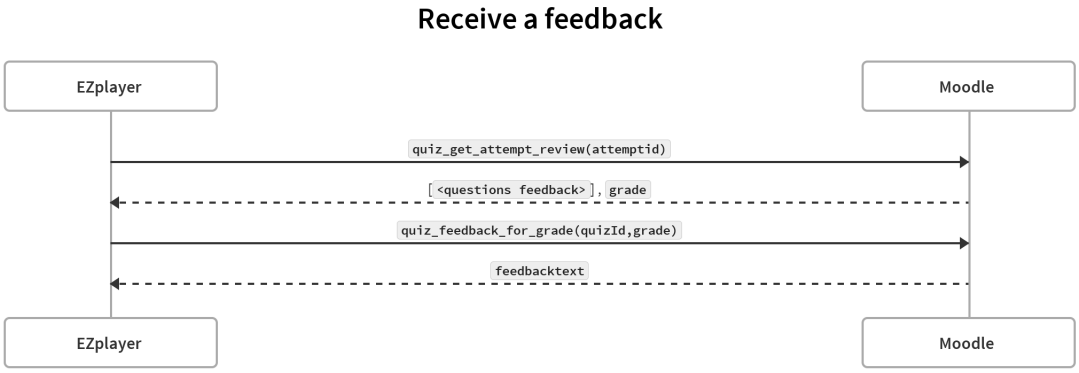


Figure 3.9: Quit an unfinished quiz

- **`lib_curl.php` (added)** : This file has been created by Dongsheng Cai. It is a wrapper class to facilitate the use of cURL in PHP [38]. It is used in the file `lib_moodle.php` to make requests cURL to Moodle. cURL is a library and command-line tool to request and transfer data by using multiple protocols [39], and here HTML.

- **`lib_moodle.php` (created)** : This file is also a Moodle library for EZcast. It is used alongside the JavaScript library seen above because some calls are made before the page loads to alleviate the number of Ajax requests.

- **`lib_quiz.php` (created)** : This file contains the functions to add, edit or delete the quiz. It interacts with XML files where the quizzes are stored. More information about XML quiz files can be found in the section 3.4. This file also has the task to generate the HTML code for the "quiz add" form by using the data received by the cURL requests.

- **`lib_simple_html_dom.php` (added)** : This file has been created by S.C. Chen. It is an HTML DOM parser written in PHP5 for PHP [40]. This is used to parse the response of some cURL requests to Moodle. Indeed, some responses contain HTML code and some information in this code were mandatory to display the questions.

- **`div_quiz_form.php` and `div_quiz_inner_form.php` (created)** : These files are used to display the form to add a new quiz. The initial values are retrieved in `lib_quiz.php`, then integrated into the templates `div_quiz_form.php` and `div_quiz_inner_form.php`. The

HTML code is finally updated by JavaScript (included in that file) every time the course or the quiz is updated in the drop-down lists.

- **`div_right_details.php` (modified)** : This file contains the code to display the official and personal bookmarks in a column at the right of the video player. A tab has been added to display the questions of the quiz. Each question can be directly accessed if clicked and the questions that are already answered appear in a different color. In this column, there are buttons to edit and delete the quiz. For the edition, a form is directly displayed in the column instead of the data shown before.

- **`main.php` (modified)** : This file is modified by adding the functions to verify every data inserted in the "quiz add" and "quiz edit" form.

- **`web_index.php` (modified)** : This file has been modified to receive Ajax requests sent by the JavaScript file `lib_quiz.js` when adding, editing or deleting a quiz and redirect them into the correct controller (in the controller folder). These controllers call function defined in `lib_quiz.php`. This file also contains the functions to reload the quiz list in the column of the `div_right_details.php` each time a quiz is deleted, updated or added. Finally, we have added the process to get the Moodle token to be able to use the web services and save it into the PHP `$_SESSION` variable when a user logs in.

- **`tmpl_source/popup_quiz*.php` (created)** : Multiple files beginning by "`popup_quiz`" that holds templates to display pop-ups to add or delete a quiz, to warn the teacher of the loading of his quizzes and, last but not least, to display the questions of the quizzes and feedbacks to the students. Data from the controllers are integrated in these files.

- **`controller/quiz*.php` (created)** : Multiple files beginning by "`quiz`" in the controller folder. These files are used to call the templates seen above and pass them the values to be displayed. Some of them are also used to perform actions, for instance, to add or delete quizzes.

As we can see, the functions of these files are somewhat mixed up. We call that a Spaghetti code". We address this problem in section 4.3.4.

3.4 Save the Quizzes

In a previous section, we have seen basic scenarios on how the new feature should behave. In this section we see how to bind the two different scenarios seen in section 3.2.2 and section 3.2.4 which are adding a new quiz and answering it.

To be able to save the information regarding the quizzes we chose to adapt the behavior used for the official bookmarks but with some changes. Official bookmarks are all stored in the course folder of the Repository so they can be accessible on the course page of EZplayer. This characteristic does not bring added value for the quizzes since we only display questions in the column on the right of the video on the asset page. For this reason we chose to store them directly in the asset's folders.

The files are saved in an XML file. XML stands for eXtensible Markup Language. According to the World Wide Web Consortium (W3C), "it is a simple, very flexible text format" [41]. The XML language can be seen as a tree with multiple nodes. In the format we chose, we have a root node called `<quiz>` and multiple nodes called `<question>`. Each `<question>` node holds the information for one question of the quiz. If there are three questions there are going to be three `<question>` nodes. Each `<question>` node contains multiple other nodes:

- `<album>`: the album the quiz is made for.
- `<asset>`: the asset the quiz is made for.
- `<description>`: a description of the quiz (optional)
- `<courseId>`: the Moodle ID of the course that contains the quiz.
- `<quizId>`: the Moodle ID of the quiz for the video.
- `<questionId>`: the Moodle ID of the question of the quiz.
- `<timecode>`: the time (in seconds) of the video when the quiz should be displayed.

For example, in the XML code presented in the listing 10, the quiz with ID 11 has two questions (ID equals to 1 and 2) at the 23rd and the 67th second. The quiz is called "test" and is included in the video of the asset "`2016_11_23_14h09`" for the course "`Cours_test-pub`". This quiz does not have a description.

```
<quiz>
    <question>
        <album>Cours_test-pub</album>
        <asset>2016_11_23_14h09</asset>
        <title>test</title>
        <description/>
        <courseId>2</courseId>
        <quizId>11</quizId>
        <questionId>1</questionId>
        <timecode>23</timecode>
    </question>
    <question>
        <album>Cours_test-pub</album>
        <asset>2016_11_23_14h09</asset>
        <title>test</title>
        <description/>
        <courseId>2</courseId>
        <quizId>11</quizId>
        <questionId>2</questionId>
        <timecode>67</timecode>
    </question>
</quiz>
```

Listing 10: XML format for a saved quiz

3.5 Deployment of the solution

In this section, we describe the different configurations and parameters to enable in Moodle and EZcast to use our new developed functionality.

3.5.1 Moodle

The web services for the quizzes are available since the Moodle Mobile app (with the version 3.1 of Moodle), so we have to use Moodle with version equal or greater than 3.1 to be able to use the

new feature because before this version, no web services were available to use the quizzes outside Moodle. The solution is deployed in Moodle in two steps: in the first one we see how to activate the web services in the Moodle administration interface and in the second one we present the Moodle plugin developed to be able to use the new feature in Moodle and how to install it.

Web services activation

At the time of this writing, Moodle is actually in version 3.2.2, so we describe the procedure to activate the web services for this version. To be able to use web services in Moodle we need to configure three settings [42] [35]:

- Enable web services: Go to ***Site administration***, then ***Advanced features*** and finally, check the ***Enable web services*** checkbox.
- Activate the `REST` protocol: Go to ***Site administration***, then ***Plugins***, then ***Web services***, then ***Manage protocols***
- Finally, activate the creation of tokens for authenticated users: Go to ***Site administration***, then ***Users***, then ***Permissions***, then ***Define Roles*** and select ***Authenticated user***. In the opened page, edit the roles and allow the capability `moodle/webservice:createtoken`. This enables the creation of a token for every user that interacts with Moodle by using its web services.

The first two settings may already be activated if Moodle has previously been configured to be used with the Moodle app (navigate to ***Site administration*** → ***Mobile app*** → ***Mobile settings***, then check the option "***Enable web services for mobile devices***") [43].

Moodle plugin

To simplify the creation of the external service, a local plugin has been created but, of course, everything could have been done manually. In this plugin, we define every Moodle functions that need to be accessed [44]. These functions are the ones explained in section 3.1:

To install it, we need to go to ***Site administration*** → ***Plugins*** → ***Install plugins*** then drag and drop the plugin folder to the special upload zone.

3.5.2 EZcast configuration file

The installation of EZcast is separated in two parts, the first one is an installation using the UNIX terminal and the second is a web install. We focus on the second one because it is the part that generates the configuration file called `config.inc`. We need to add a field to enter the URL of the Moodle server and another one to enter the Moodle external service so they are available in the configuration file.

```php
<?php
$services = array(
    'EZplayer Quiz' => array(
        'functions' => array (
            'core_webservice_get_site_info',
            'core_enrol_get_users_courses',
            'mod_quiz_get_quizzes_by_courses',
            'mod_quiz_start_attempt',
            'mod_quiz_get_attempt_data',
            'mod_quiz_process_attempt',
            'mod_quiz_get_attempt_review',
            'mod_quiz_get_quiz_feedback_for_grade'
        ),
        'restrictedusers' => 0, // "If enabled, the Moodle administrator must link
                                // some user to this service into the administration".
        'enabled'=>1,           // "If enabled, the service can be reachable on a
                                // default installation"
        'shortname' => 'ezplayerquiz'
    )
);
?>
```

Listing 11: Local Moodle Plugin

Chapter 4

Results and Analysis

In this chapter, we discuss the differences between the technical specifications described in section 2.4. We resume what has been implemented in section 4.1. We then explain the various problems encountered during this project. Finally, we list different design flaws (described in [45] and [46]) and propose some refactoring solutions.

4.1 Results

This section presents the features that have been implemented in the last version of the code that has been written for this master thesis. Some features are accompanied with a screenshot when it is relevant. We have also recorded a short video to show the features below in a real-world scenario. The video is accessible with the URL: `https://vimeo.com/220134487`. The presentation uses a YouTube video called "***Le PIB, cette fausse boussole #DATAGUEULE 54***" [47] to simulate an extra course material for an "***Introduction to Economy***" class. A quiz for this video has been created by ourselves to match the topic and ask relevant questions. Since this video (that was given to us) is in French, the screenshots present the questions, feedback and interface are in French.

4.1.1 For students that are registered on Moodle and enrolled to the Moodle course

Answer to a quiz

A pop-up is displayed if there is a Moodle quiz available for that video. If the student hits the "Oui" (or "Yes" in English) button, the quiz and the questions are loaded. The feature is represented in figure 4.1.

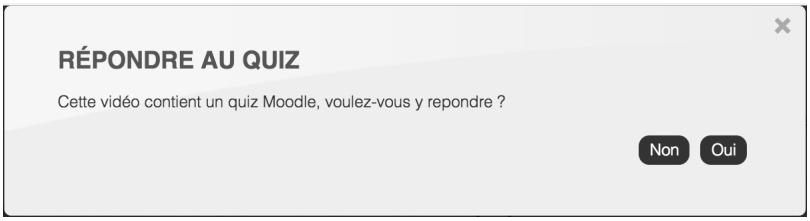
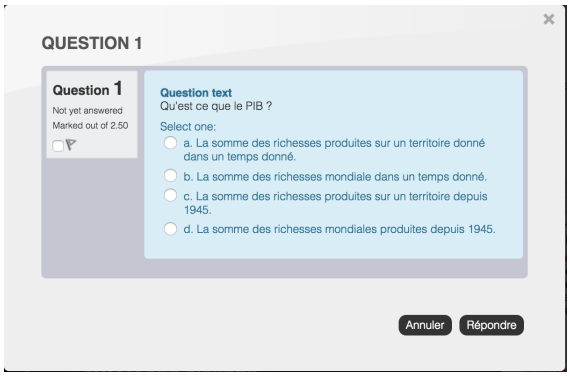


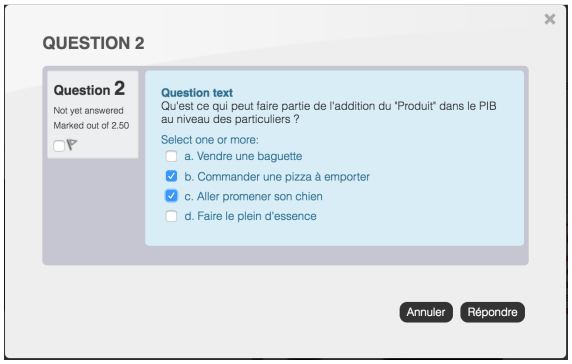
Figure 4.1: Pop-up to start a quiz attempt

Answer to a Question

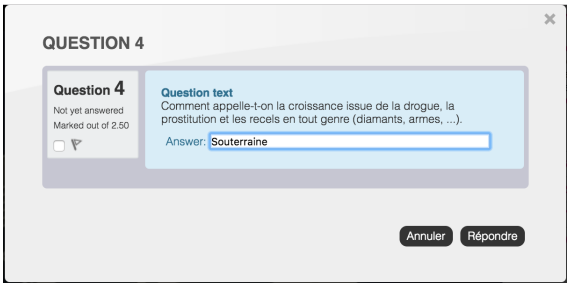
When the time code of the question equals the current time code of the video, the video is paused and the question is displayed to be completed. Three question types are compatible with



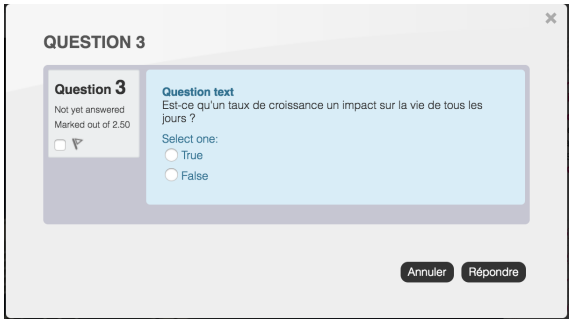
(a) Multiple-choice question (1 answer)



(b) Multiple-choice question (multiple answer)



(c) Short answer question



(d) True/False question

Figure 4.2: The three different types of question available

the functionality : Multiple choice (with one or multiple answers, see figures 4.2a and 4.2b), Short Answer (see figure 4.2c) and True/False (see figure 4.2d). After sending the answer, the response cannot be changed and the questions are not displayed anymore. As shown, we can cancel the question by clicking on "Annuler" (or "Cancel") and come back later to answer it. It is not mandatory to answer them in the predefined order (order of the video).

Receive a feedback

After answering the last unanswered question of the quiz, a modal is displayed. This modal contains the final feedback for the grade, all the questions, their answers, the teacher comment on the response chosen by the student and a general feedback for the question. All questions and feedbacks are displayed with the same template. The feature is represented in figure 4.3. On this picture, we can see the feedback for one question, the answer is false but there is a highlighted comment and below, in the yellow box, some additional information about the question's topic.

Display the list of questions

On the column at the right of the video player, the quiz name and all the questions of that quiz are displayed. Answered questions have a different layout to easily identify the questions that still need to be completed. The feature is represented in figure 4.4. As we can see, a question is already completed and displayed with a different style. We have also extended the title section with a small arrow next to the title. When clicked, the quiz description and indication of the final feedback are displayed. Finally, we can observe an icon that represents a small curved arrow. This icon allows a student to redo a quiz (only if no other quiz is started at that moment).

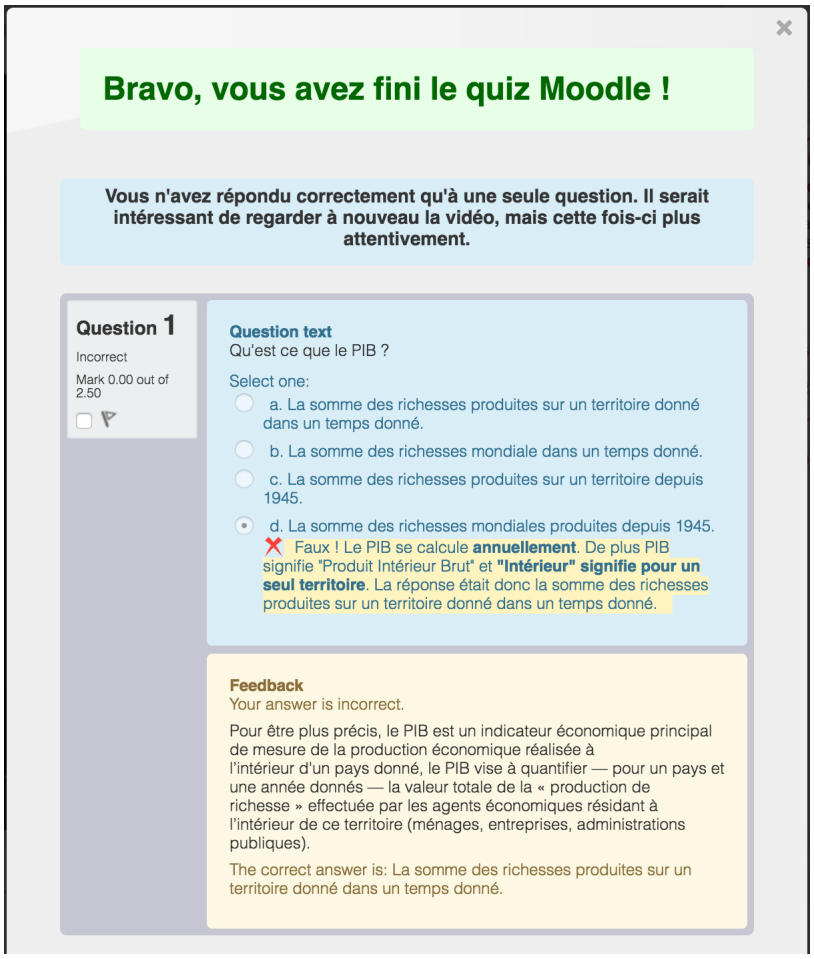


Figure 4.3: Example of a feedback received after finishing a quiz attempt

4.1.2 For the teacher or the course manager

Display the "quiz add" form

Before adding a Moodle quiz to an EZcast video, we first need to click on the check mark icon in the toolbar below the video player. This action triggers the display of a pop-up that informs the teacher that all its courses and quizzes are going to be loaded and that this process can take some time, this process is described in section 3.2.2 and the pop-up can be seen in figure 4.5.

Select a course and a quiz

In the "quiz add" form, the teacher can select a course and then a quiz that he has created in that course. A figure of the complete form can be found at figure 4.6 and the feature described can be seen at the top of the form. Note that the title and description fields are automatically updated when the Moodle quiz has this information.

Encode the time code of the video for each question

Once the quiz is selected, the question list is displayed in the form. Each question has a button next to it and, when clicked, the current time of the video is added in the input field next to that question. A preview of the question text, type and answers is also displayed. The complete form can be found at figure 4.6 and the feature described can be seen at the bottom of the form. Below the list of questions, we observe a check box to activate the feedback after the quiz submission.

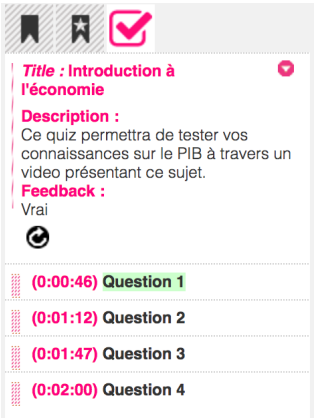


Figure 4.4: Example of the question list for the student with a answered question

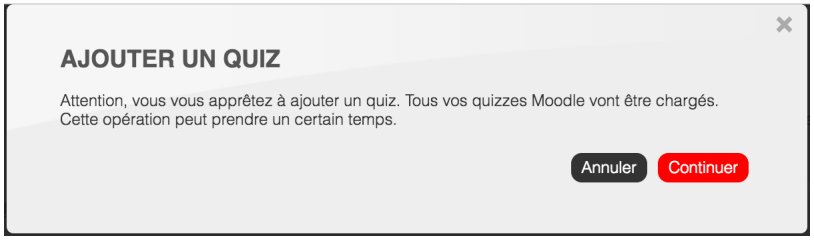


Figure 4.5: Pop-up to inform the teacher that the process could take some time

Edit and Delete a Quiz

In the rightmost column, where the questions are listed (see figure 4.7a), the teacher can edit and remove the quiz. The edition regroups the change of title, description and time code of each question. A representation of that feature can be seen in figure 4.7b. A button, added next to the edition button, can be seen in picture 4.7a.

4.2 Not implemented features

Most of the functionalities have been implemented to be able to achieve the final goal which is the integration Moodle quizzes in EZcast videos. However, not all of the functionalities in the functional requirements (see section 2.4) have been implemented the way in which they have been defined.

Feedbacks

We have described in the functional requirements that we would propose multiple types of feedback. Since the Moodle web services for the quizzes have been introduced in the end of May 2016, they were not very documented yet and we did not find how to get every quiz information that is available in Moodle. We only succeeded in getting one feedback (containing a comment of the chosen answer and a general feedback about the question) for each question after submitting the whole quiz for review. Each of our other attempts with different web services were unsuccessful and tells us that we should finish and submit the quiz before receiving a feedback. Therefore, we also did not implement the drop-down list to choose between feedback types in the "quiz add" form. Instead, we have replaced this drop-down list with a simple check box to activate the feedback after the submission of the quiz.

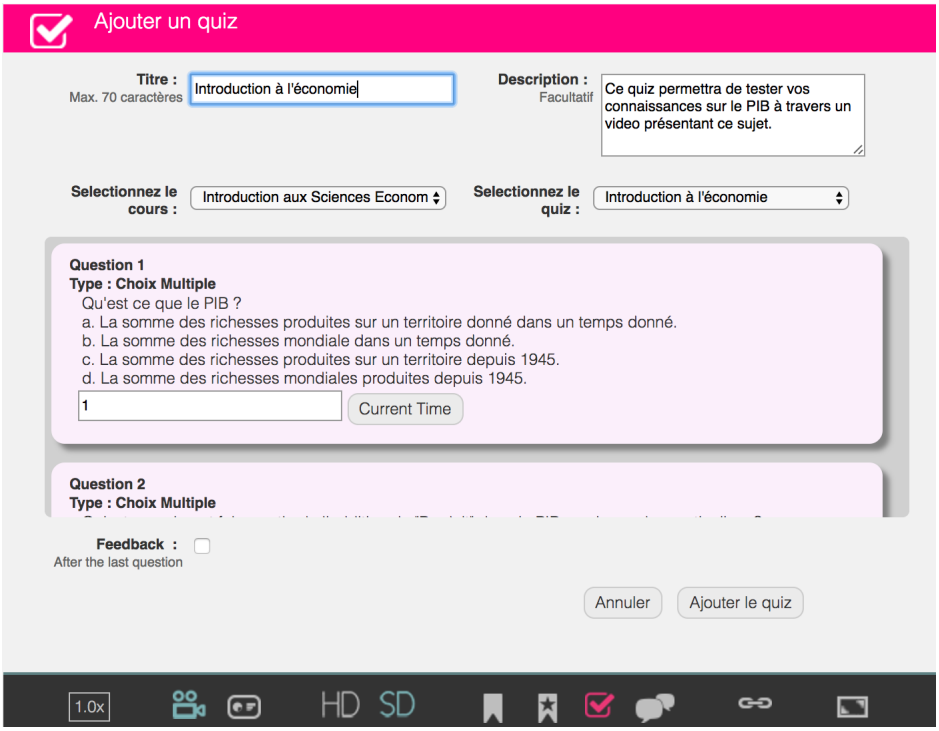


Figure 4.6: The "quiz add" form for the teachers

Display Questions in the Second Video Flow

According to the functional requirements, the quizzes were to be displayed in the second video flow when there is only one or in place of the of camera flow when both flows are available. Instead, we showed the question in a JavaScript modal ("*flexible dialog prompts powered by JavaScript*" [48]). This solution was much easier to implement and just as effective as the solution described in the functional requirements.

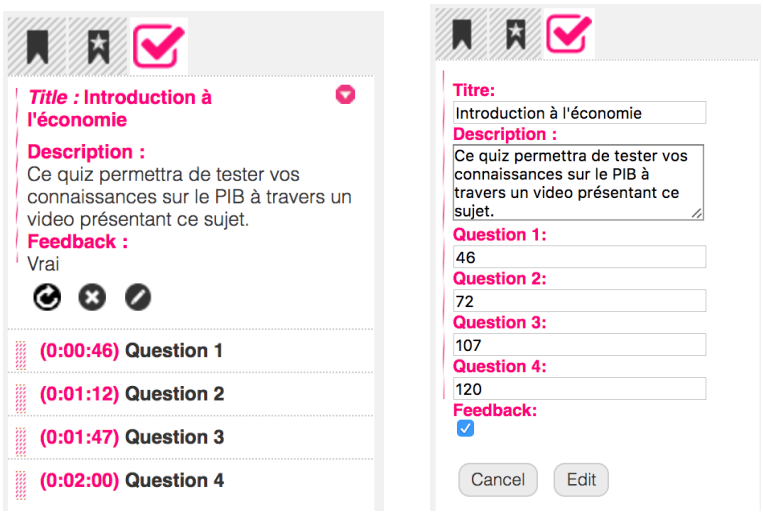
4.3 Difficulties in EZcast

We have some problems during the production of this master thesis. We had difficulties at the beginning with the installation, with the initial source code when implementing the solution and at the end to package every modified, created or added files.

4.3.1 Installation

The software sources are hosted on Github. Github is a web tool that allows its users to host and share code. This sharing promote collaboration, code review and project management [49]. Github supports simultaneous development of several functionalities thanks to several branches. The original code of EZcast can be found at this address:
`https://github.com/ulbpodcast/ezcast`.
When we had to install the software, we used the main branch called `master` but the installation scripts contained a lot of errors. After trying every branch (five in total), we made the same findings. So, after a lot of tries, we decided to correct the installation scripts and files ourselves. At the time of this writing, a lot of errors seem to have been corrected in the branch `master`.

The first step explained above installs EZadmin, EZplayer and EZmanager (see the section 1.1.2). To have the minimal working environment, we also need to install EZrenderer so that the system can convert and process the videos uploaded manually. But there is a problem: it



(a) List of questions (b) Edition of a quiz

Figure 4.7: This is the view of the quiz tab for the teacher, (a) in normal mode, (b) in edition mode

needs a lot of tweaking to run EZrenderer on the same hardware as EZadmin, EZplayer and EZmanager and, even after that EZrenderer seemed to be correctly installed, we did not succeed in processing a video. The error message encountered told us that there was an encoding error even if all the libraries and dependencies were correctly installed. As, after a lot of research we did not find a solution to this problem, we decided to short the processing of the video and directly paste already converted videos to the correct folder. These videos were already processed and converted by using the EZrenderer instance of the UCL. With this method, we finally had a working environment.

4.3.2 Initial Code

At first sight, the code found on the Github repository given above seems to be well organized and divided into meaningful files. After studying the implementation of the files of EZplayer (which is the only relevant folder for us), we saw that the reality is far from what we first saw.

The code is over 390 000 lines, which is really big. This should not have been a problem if it was well organized. Here, variables or functions declared in one file, are used in multiple other files. As an analogy, you can see it as a big puzzle: you know the kind of piece you need, but you have to search in the whole box to find it after trying multiple other pieces. For example, lots of variable used in the JavaScript files are declared in PHP files which are not in the same folder when installing. Indeed, the folder where JavaScript files are stored is in the web root of the computer, and the folder that contains PHP files is stored somewhere else in the computer (on UNIX, the default location is `/usr/local/ezcast/`). With this organization, it was really hard to follow the logical path that the data uses and it took us a lot of time to understand what was going on. It is what is called a "spaghetti code". This name comes from the fact that we can see the software as a bowl of spaghetti and it is very difficult to follow one from beginning to end because they are all twisted and tangled [50].
For instance, we used the jQuery function `$(document).ready()` to dynamically load elements of the "quiz add" but, as in other files this function already existed, the content was loaded twice, made Ajax requests that should not be done twice and caused unexpected behaviours. This is the reason why we had to create the Moodle library in PHP explained in 3.3.

4.3.3 Copying Back New Code in Installation Files

In this section, we discuss about the difficulty to package the created, added and modified files (described in 3.3) into the installation folder. To understand this difficulty we first need to know two important steps of the installation process.

The first one is when the installation script copies the folder named "`htdocs`" in the web root folder of the web server. This folder contains all JavaScript files, CSS files and resources used in the web interfaces. To package these files in the installation folder we just need to move the content of the folder in the web root into the "`htdocs`" folder. Thus this step is pretty easy.
The real problem comes from the second step. During the installation process, a folder named "`tmpl_source`" is used to generate another folder "`tmpl`" containing two other folders "`fr`" for the French language and "`en`" for the English language. The files in "`tmpl_source`" are templates where every displayed sentence that need to be translated have an id surrounded by ® symbols. When connecting to EZplayer (or another web interface of EZcast), you have to choose between French and English and the working directory is chosen according to this language choice. Hence, to be able to develop and see the results, we have to choose a language (for example French) and consequently, a working directory in the "`tmpl`" folder (for example "`fr`"). To package these files into the install folder, we have to move all the modified or created files from the language folder to the "tmpl" folder and then change the displayed text by an id surrounded by ® symbols and add the correct translation into `translation.xml` which is a file in the installation folder of each EZcast interfaces. From our point of view, this process is not particularly difficult, but it is painful and not efficient.

4.3.4 EZplayer Design Flaws

Since we had a lot of problems to understand the code because of how it was written, we have decided to write, in this section, some "code smell" that we have experienced in the code during the development of this project. These design flaws are mostly described in [45] and [46].

Long Method

Long methods are a bad practice. They are much more difficult to read and error-prone. The longer a method is, the more difficult it is to read the code and understand the purpose of it. It is really bad for maintainability since it is much more difficult to make modifications, to test them [51] and to search for bugs.
For example, we found two very long methods and, in total, seven methods with more than 100 lines of code only in the EZplayer code. The first one is `load_page()` in the file `web_index.php` that has a length of 308 lines of code and over 50 `case` conditions. This method is really long but, surprisingly, it is pretty easy to read. Each case calls a controller and that is all it does. The second one is more problematic. It is located in the file `lib_threads_pdo.php`, is named `thread_search()` and has 255 lines of code. Even if we have an idea of what the method should do with its name, the rest of the code is more obscure. There are multiple SQL requests then some processing on the results of these SQL requests but it is not clear what and why. In total, the NPath complexity ("*the number of acyclic execution paths through that method*" [52] which means, the number of "*(...) "paths" there are in the flow of your code in the functio*n" [53]) of this method is more than 6.000.000.000 and the best practice told us that it should not exceed 200.
This second example demonstrates the need to refactor some of those long methods. The solution to adopt is the refactoring pattern called **Extract Method**. This pattern divides the long method into multiple smaller one. Usually the comments help to know what block of code should become a method. Indeed, when there is a block of code and a comment that explains what it

does, it is usually the sign that this block should become a method. Loops also give some signs of extraction.

Shotgun Surgery

Usually this code smell says that "*making any modifications requires that you make many small changes to many different classes*" [45]. Since we do not have classes in the code, we derive this code smell by saying that a shotgun surgery happens if any modifications requires that you have to make many small changes to many different files. For example, to save a new quiz in XML just after sending the form, we had to change 5 files. Even worse, to delete a quiz we had to create or modify a total of 8 files. Moreover, it was really difficult to follow the data flow in all those files. The refactoring solution is called **Move Method**. This solution implies to move methods from one class to the other, and in our case, from one file to the other. It is not always easy to move the methods that should belong together because multiple files could hold them so that the implementation remains logical overall.

Combinitorial Explosion and Code Duplication

Combinitorial Explosion happens when a lot of code does almost the same thing. For example, in the file `lib_various.php`, three methods almost have the same processing:
`simple_assoc_array2xml_file`, `assoc_array2xml_file`, `assoc_array2xml_string`.

Moreover, these three methods use almost the same code except for the return statement and one of the loop conditions. It is called **Code Duplication**. According to [46], it is the worst bad smell you can encounter in a code and, since we found it once and there are more than 390.000 lines of code, it is certainly not an isolated case.
The refactoring solution we have to use for a duplicate code is the **Extract Method** already explained in the **Long Method** section.

Conclusion

In this dissertation we intended to develop a new tool that could extend the use of active learning methods. This tool is a new feature for the video player of EZcast. Our purpose was to implement, in EZplayer, an interaction between students and teachers by the means of quizzes.

In the first chapter we have presented the different technologies used in this master dissertation and then compared different podcasting tools that promote the active learning process. In the framework of this project, we have principally developed Pod which is as complete as EZcast if not more.

In the second chapter, we have exposed the problem and the main purpose of this project: how to promote active learning within a podcasting solution and further, how to strengthen the communication and encourage interactions between students and teachers. Given the comparison made in the previous chapter, we have decided to develop a functionality already included in Pod, EZcast's main competitor: the integration of Moodle quizzes during the playback of video uploaded by the teacher.
We have described the different functionalities required for these quizzes, for instance, a form for the teacher to add a quiz, or the presentation of the question to the student in the second video flow. Afterwards, we have explained the different functional requirements that needed to concur the current implementation of EZcast as well as the future development that it will undergo.

In the third chapter, we have described the complete operation of the web services as well as the data structure to exploit them. Additionally, we have presented different possible scenarios that EZcast and its new features might experience. We have also dived more deeply into the code by explaining the different files that were added, modified or created, and then, by furnishing a list of instructions to allow a Moodle administrator configuring a new external service. This will be used by EZcast to identify Moodle users, load quizzes and answer them.

In the fourth and last chapter, we have exposed our results and dwelt on the encountered problems during the realization of this master thesis. All the essential functionalities have been successfully implemented except for the type of feedback received and how a question is displayed. Finally, we have analyzed the initial code to point out some well-known design flaws and propose some refactoring solutions.

To conclude this dissertation, we can say that the use of online tools such as podcasting solutions are still relatively new in the world of education. Many improvements can still be made. For example, we could imagine live chat or video streaming to allow students to directly ask questions to their teacher. Additionally, some other learning material could be added to EZcast in the same way as the quizzes.
We are firmly convinced that EZcast will be an amazing learning tool for students. However, future development of this platform could be greatly improved by the use of web frameworks or a redesign by using object-oriented programming.

We never stop learning, thence we never stop improving our learning and teaching methods. Nowadays considerable progres has been made and already a lot has been achieved in the field of educational methods since the rise of the first European Universities. We sincerely hope these accomplishments will undertake a long-lasting path and we are keen to observe which role scholars, teachers and students will play to steer learning methods towards innovative and unexplored frontiers.

Acronyms

API Application Programming Interface.

CSS Cascade Style Sheets.

CTE Centre des Technologies au service de l'Enseignement.

cURL client URL.

GPL General Public License.

HQ High Quality.

HTML HyperText Markup Language.

HTTPS HyperText Transfer Protocol Secure.

ID Identifiant.

IMS content package IMS is a body which helps define technical standards for various things, including e-learning material [54]..

JSON JavaScript Object Notation.

JSONP JSON with Padding, to request data from a server from a different domain [55]..

LQ Low Quality.

LTI Learning Tools Interoperability.

Moodle Modular Object Oriented Dynamic Learning Environment.

PDF Portable Document Format.

PHP Hypertext Preprocessor, web language.

PIB Produit Intérieur Brut.

REST REpresentational State Transfer.

RSYNC Remote Synchronization.

SCORM Sharable Content Object Reference Model.

SOAP Simple Object Access Protocol.

SQL Structure Query Language.

SSH Secure Shell.

UCL Université Catholique de Louvain.

UDDI Universal Description Discovery and Integration.

ULB Université Libre de Bruxelles.

URL Uniform Resource Locator.

W3C World Wide Web Consortium.

WSDL Web Services Description Language.

XML eXtensible Markup Language.

XMLRPC Remote Procedure Call using XML standard.

Bibliography

[1] Hilde de Ridder-Symoens. *A history of the university in Europe: Volume 2*, volume 2. Cambridge University Press, 2003.

[2] How people learn: Active vs passive learning | openlearning. `https://www.openlearning.com/blog/HowPeopleLearnActiveVsPassiveLearning`. Accessed: 15/05/2017.

[3] Active learning | center for research on learning and teaching. `http://www.crlt.umich.edu/tstrategies/tsal`. Accessed: 16/05/2017.

[4] David Weltman. *A comparison of traditional and active learning methods: An empirical investigation utilizing a linear mixed model*. The University of Texas at Arlington, 2007.

[5] Active learning continuum | center for research on learning and teaching. `http://www.crlt.umich.edu/sites/default/files/resource_files/Active%20Learning%20Continuum.pdf`. Accessed: 16/05/2017.

[6] Scott Freeman, Sarah L Eddy, Miles McDonough, Michelle K Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 111(23):8410–8415, 2014.

[7] Educational technology | uclouvain. `https://uclouvain.be/fr/decouvrir/elearning.html`. Accessed: 16/05/2017.

[8] Les plateformes de cours en ligne | uclouvain. `https://uclouvain.be/fr/decouvrir/elearning.html`. Accessed: 16/05/2017.

[9] Dynamiser son amphi avec les télévoteurs | uclouvain. `https://uclouvain.be/fr/decouvrir/elearning.html`. Accessed: 16/05/2017.

[10] Podcast | uclouvain. `https://uclouvain.be/fr/etudier/lll/podcast.html`. Accessed: 16/05/2017.

[11] Ezcast, outil opensource de webcasting développé par l'ulb | awt. `http://www.awt.be/web/edu/index.aspx?page=edu,fr,foc,100,155`. Accessed: 08/11/2017.

[12] Diagram of the interactions between the different products of the ezcast infrastructure | ezcast infrastructures. `http://ezcast.ulb.ac.be/files/EZcast_Infrastructure_doc.pdf`. Accessed: 09/11/2016.

[13] Diagram of the repository | ezcast infrastructures. `http://ezcast.ulb.ac.be/files/EZcast_Infrastructure_doc.pdf`. Accessed: 09/11/2016.

[14] Flowplayer, online video platform | flowplayer website. `https://flowplayer.org`. Accessed: 08/11/2017.

[15] Moodle development page | github. https://github.com/moodle/moodle. Accessed: 06/11/2017.

[16] Les plateformes de cours en ligne | uclouvain. http://www.uclouvain.be/326571.html. Accessed: 25/10/2016.

[17] Resources | moodle documentation. https://docs.moodle.org/32/en/Resources. Accessed: 01/02/2017.

[18] Activities | moodle documentation. https://docs.moodle.org/32/en/Activities. Accessed: 01/02/2017.

[19] Question type | moodle documentation. https://docs.moodle.org/32/en/Question_types. Accessed: 01/02/2017.

[20] What are web services ? | webopedia. http://www.webopedia.com/TERM/W/Web_Services.html. Accessed: 04/02/2017.

[21] Pod platform | github. https://github.com/EsupPortail/pod. Accessed: 09/11/2017.

[22] Esupportail | wikipédia. https://fr.wikipedia.org/wiki/Esup-Portail. Accessed: 05/03/2017.

[23] Pod : La nouvelle plateforme de podcasts "made in france" | esup-portail. https://www.esup-portail.org/wiki/download/attachments/479068181/esupdays20--08-pod.pdf?version=1&modificationDate=1443192647000&api=v2. Accessed: 09/11/2017.

[24] Video.js development page | github. https://github.com/videojs/video.js. Accessed: 08/11/2017.

[25] Video.js: the player framework | video.js. http://videojs.com. Accessed: 08/11/201.

[26] Npm. https://www.npmjs.com. Accessed: 12/04/2017.

[27] Video.js | npm search. https://www.npmjs.com/search?q=videojs&page=1&ranking=optimal. Accessed: 12/04/2017.

[28] Video.js, an html5 video player | drupal). https://www.drupal.org/project/videojs. Accessed: 12/04/2017.

[29] Marcel lebrun | wikipédia. https://fr.wikipedia.org/wiki/Marcel_Lebrun. Accessed: 17/05/2017.

[30] Frederic Fervaille. Private meeting. Location: Louvain-la-Neuve, Date: 17/05/2017.

[31] Construire mon scénario pédagogique : Diffuser des vidéos ou des sons | moodleucl. https://moodleucl.uclouvain.be/mod/data/view.php?d=1684&rid=70178#Utiliser_le_service_de_podcast_de_lUCL_pour_diffuser_des_vid%C3%A9os_ou_des_sons. Accessed: 17/05/2017.

[32] Michel Van Lierde. Podicampus mode d'emploi.

[33] Philippe Meurrens. Mail exchange. Date: 18/05/2017.

[34] Flipping the classroom | center for teaching. https://cft.vanderbilt.edu//cft/guides-sub-pages/flipping-the-classroom/. Accessed: 18/11/2016.

[35] Using web services | moodle documentation. https://docs.moodle.org/32/en/Using_web_services. Accessed: 28/10/2017.

[36] jquery ajax() method | w3school. https://www.w3schools.com/jquery/ajax_ajax.asp. Accessed: 21/11/2016.

[37] What is jquery | jquery. https://jquery.com/. Accessed: 21/11/2016.

[38] curl wrapper class for php | github. https://github.com/dcai/curl. Accessed: 05/03/2017.

[39] curl | wikipedia. https://en.wikipedia.org/wiki/CURL. Accessed: 15/03/2017.

[40] Php simple html dom parser | sourceforge. http://simplehtmldom.sourceforge.net/. Accessed: 15/03/2017.

[41] Extensible markup language (xml) | w3. https://www.w3.org/XML/. Accessed: 09/11/2017.

[42] Using moodle web services | rumours. http://www.rumours.co.nz/manuals/using_moodle_web_services.htm. Accessed: 28/10/2017.

[43] Moodle web services | moodle documentation. https://docs.moodle.org/32/en/Mobile_web_services. Accessed: 28/10/2017.

[44] Adding a web service to a plugin | moodle documentation. https://docs.moodle.org/dev/Adding_a_web_service_to_a_plugin. Accessed: 15/04/2017.

[45] Bad code smells | sourcemaking. https://sourcemaking.com/refactoring. Accessed: 22/05/2017.

[46] Martin Fowler and Kent Beck. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[47] DataGueule. Le pib, cette fausse boussole #datagueule 54.

[48] Modal | bootstrap. https://v4-alpha.getbootstrap.com/components/modal/. Accessed: 03/06/2017.

[49] How developers work | github. https://github.com/features. Accessed: 23/05/2017.

[50] Spaghetti code | wikipedia. https://en.wikipedia.org/wiki/Spaghetti_code. Accessed: 23/05/2017.

[51] Code smells in php | slideshare. https://fr.slideshare.net/dagfinnr/code-smells-in-php. Accessed: 24/05/2017.

[52] Code size rules | phpmd. https://phpmd.org/rules/codesize.html. Accessed: 24/05/2017.

[53] Npath complexity and cyclomatic complexity explained | modess. https://modess.io/npath-complexity-cyclomatic-complexity-explained/. Accessed: 24/05/2017.

[54] Ims content package | moodledocs. https://docs.moodle.org/31/en/IMS_content_package. Accessed: 04/06/2017.

[55] Jsonp | wikipedia. https://en.wikipedia.org/wiki/JSONP. Accessed: 04/06/2017.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**